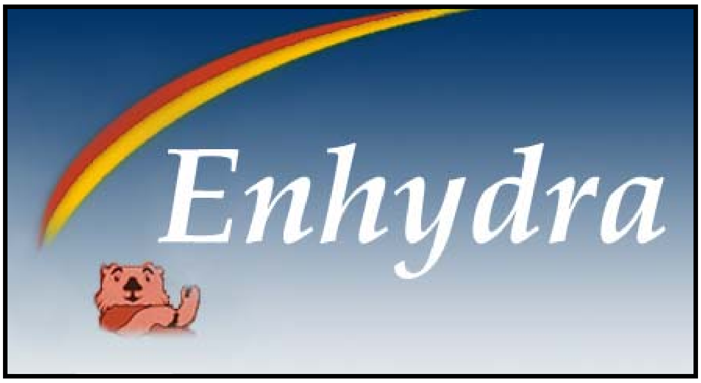
Getting Started

with

Enhydra



Contents

Chapter 1

Introduction

This book introduces the Enhydra™ ver. 6.x application server and the Enhydra development environment. It provides an introductory overview of Enhydra and explains how to develop an application by using an example to illustrate some of the key principles of Enhydra applications.

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.

- 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, • 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: • Commands that you enter directly, code examples, utility programs, and options. For example, • cd mydir • System.out.println("Hello World"); • make utility • -keep option • Java packages, classes, methods, objects, and other identifiers. For example, • ErrorHandler class • run() method • Session object **Note:** Method names are suffixed with empty parentheses, even if the method takes parameters. **Note:** Only specific references to object names are in fixed-width; generic references to objects are shown in plain text. • File and directory names. For example: • /usr/local/bin **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 (/).

Table 1.1 Typographical conventions (continued)

Convention	Description
Fixed-width italic and <*Fixed-width italic*>	Indicates variables in commands and code. For example, • xmlc [*options*|*optfile*.xmlc ...] *docfile* **Note:** Angle brackets (< >) are used to indicate variables in directory paths and command options. For example, • -class <*class*>
Boldface	Used for the words **Note**, **Tip**, **Important**, and **Warning** when they are used as headings that are drawing 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, • Ctrl-C.
{ } (braces)	Indicates a set of required choices in a syntax line. For example, • {a|b|c} means you must choose a, b, or c.
[ ] (brackets)	Indicates optional items in a syntax line. For example, • [a|b|c] 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, • xmlc [*options*|*optfile*.xmlc ...] *docfile*
plain text	Used for URLs and generic references to objects. For example, • http://www.lutris.com/documentation/index.html • The presentation object is in the presentation layer
ALL CAPS	Indicates SQL statements. For example: • CREATE statement

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 choice. 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, • /usr/local/bin
URLs	URLs are indicated in plain text and are generally fully qualified. For example, • http://www.lutris.com/documentation/index.html
Screen shots	Most screen shots reflect the Microsoft Windows look and feel.

Getting Started

Getting Started with Enhydra introduces the fundamentals of Enhydra. The purpose of this book is to introduce Enhydra and provide 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.

Enhydra 6.x information avalable on enhydra.org

You can find a variety of information about open-source Enhydra at the Enhydra website:

http://www.enhydra.org

The Enhydra site is the home of the Enhydra 6.x 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.

Chapter 2

Installation

Complete step-by-step installation instructions for Enhydra and related software (including bundled third-party software) are provided in Enhydra documentation, directory ***<enhydra_root>/multiserver/webapps/enhydra-docs***. To begin, refer to the top-level ***index.html*** file of this directory.

For convenience, we recommend that you print the file containing the Enhydra installation instructions prior to installation. (The instructions are included in HTML and PDF format). However, you can also follow the step-by-step installation instructions online (you can toggle back and forth between the installation program and browser).

Chapter 3

Overview

This chapter provides a high-level overview of Enhydra, Enhydra applications, and the tools used to create and run Enhydra applications. The following topics are covered:

- What is Enhydra 6.x?
- Enhydra documentation
- Anatomy of an Enhydra application
- Servlet containers
- Enhydra Application Framework
- Enhydra tools

What is Enhydra 6.x?

Enhydra is an application server for running robust and scalable multi-tier Web applications, and a set of application development tools.

An *application server* usually operates between a web server and a database server, and provides dynamically-generated content for the web server to send to web browser clients.

An *Enhydra application* is a java program that runs in servlet container and uses the Enhydra application framework at runtime.

Enhydra has three parts:

- **Application framework:** Collection of Java classes, which provide the runtime infrastructure for Enhydra applications.

- **Enhydra tools:** Use to develop Enhydra applications.

- **Servlet container (Tomcat, Jetty, …):** Servlet container is not a part of Enhydra any more, but it is needed for running Enhydra applications. Any servlet container that supports servlet 2.3 specification can be used for running Enhydra applications. Three server containers are included with Enhydra distribution: *Tomcat*, *JOnAS Tomcat* and *JOnAS Jetty*.

The following sections describe Enhydra and Enhydra applications in more detail.

For more information on	See this topic
Enhydra application architecture	“Anatomy of an Enhydra application” in the chapter 3
Servlet containers	“Servlet container” in the chapter 3
Application framework	“Enhydra Application Framework” in the chapter 3
Enhydra tools	“Enhydra tools” in the chapter 3

Documentation

The documentation for this release is located in *<enhydra_root>/multiserver/webapps/enhydra-docs* directory. Refer to *index.html* document of this directory.

Documentation updates

- *Getting Started with Enhydra* has been updated to reflect changes to Enhydra and revised for accuracy.

Anatomy of an Enhydra application

An Enhydra application can be either:

- An ***Enhydra super-servlet application*** that uses Enhydra's own application model

- A ***servlet application*** that uses the J2EE servlet application model

These two kinds of applications are similar in many ways, but have some important differences. Generally, a *servlet application* has a servlet for each page (HTML, WML, and so on) in the application. In contrast, a *super-servlet application* consists of a single servlet that contains a *presentation object* for each page.

You can use Enhydra tools such as *XMLC* and *DODS* to create both kinds of applications, and you can run both kinds of applications in any standard servlet runner, such as *Tomcat*, *JOnAS Tomcat* and *JOnAS Jetty*.

Enhydra super-servlet applications

An Enhydra application has (minimum):

- A single application object

- One presentation object for each page to be dynamically generated

These objects run in the context of the Enhydra application framework, as described in "Enhydra Application Framework", later in this chapter.

Application objects

The *application object* is the central hub of an Enhydra application. It is a subclass of *com.lutris.appserver.server.StandardApplication* and contains application-wide data, such as:

- Name of the application

- Status of the application (for example, running/stopped/dead)

- Name and location of the configuration file that initializes the application

- Log channel to use for logging

- References to the application's session manager, database manager, and presentation manager (see"Enhydra Application Framework", later in this chapter).

Properties

You can add properties (instance variables) to the application object to store information that needs to be accessible throughout the application. For example, if your application has a dozen pages that need to share a collection of customer data, you can make a vector containing the data a property of the application object so all pages can easily access it.

Methods

Each application object has the following methods:

- **startup()** starts the application

You can extend this to perform other startup functions, such as reading settings from the configuration file.

- **requestPreprocessor()** initializes the Session data structure

You can extend this as needed; for example, to check for HTTP basic authorization.

In general, application objects do not deal with HTML, handle requests, or otherwise talk to the network; presentation objects perform these tasks. The next section describes presentation objects.

Presentation objects

A *presentation object* generates dynamic content for one or more pages in an Enhydra super-servlet application.

When a browser requests a URL that ends in '.po', Enhydra passes the request on to the corresponding presentation object. Enhydra then instantiates and calls the presentation object. For example, for the URL http://www.foo.com/myapp/Xyz.po, Enhydra calls the presentation object Xyz.

Note Enhydra only calls a presentation object for URLs with a '.po' suffix. The Web server generally serves a static file for other requests.

Presentation objects must implement the interface:
com.lutris.appserver.server.httpPresentation.HttpPresentation.

This interface has one method, *run()*, that Enhydra calls, passing it an HTTP request. Presentation objects differ from servlets in that they need handle only a single request at a time. No concurrency control is required.

Enhydra also provides a response object that a presentation object can use to write data in response to HTTP requests (similar to a servlet's *service()* method). Presentation objects usually handle GET requests (for example, form submissions) and respond by writing HTML, but they can perform other functions (for example, read files sent by a POST request).

Servlet applications

In addition to super-servlet applications, you can also create and run standard servlet applications (sometimes called web applications) with Enhydra. Servlet applications conform to the Java servlet API specification, part of the Java 2 Enterprise Edition (J2EE) specification from Sun Microsystems. It is a popular application model for interactive Web applications.

For detailed information on the servlet application model, see

http://java.sun.com/products/servlet/index.html .

In a servlet application, each servlet is responsible for a single page of output (although this is not required, it is common practice). Each servlet must be a subclass of *javax.servlet.http.HttpServlet*, and will generally override the *doGet()* method, and possibly other methods, such as *init().* The general architecture of a servlet application is illustrated in Figure 3.1.

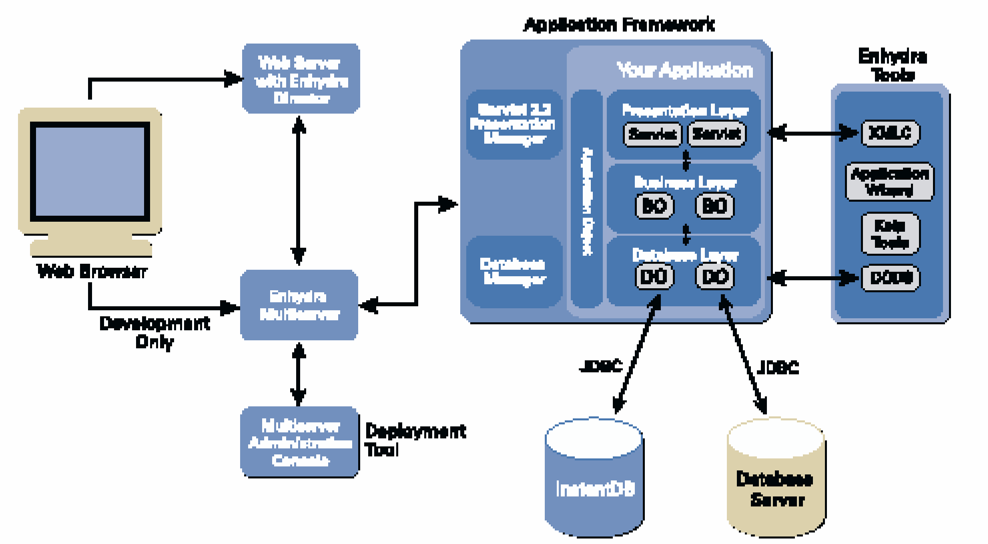


Figure 3.1 Servlet application model

Although JavaServer Pages (JSPs) are often used to create the presentation layer of servlet applications, XMLC is generally better because it provides a cleaner separation between layout code (such as HTML) and presentation logic code. Fortunately, you can use XMLC to help generate presentation code for servlet applications too.

Servlet versus super-servlet applications

While the servlet application model and the super-servlet application model are similar in many ways, they also have some key differences. You can run both kinds of applications in a servlet container, and you can use Enhydra tools such as XMLC and DODS to help create both kinds of applications.

Since the servlet API performs its own presentation management, there is no need for the Enhydra Presentation Manager (see “Presentation Manager”, later in this chapter). Likewise, the servlet API provides session management, so there is no need for the Enhydra Session Manager (see “Session Manager”, later in this chapter).

One of the key differences in the two application models is how the objects in the presentation layer are instantiated. In a super-servlet application, each request creates a new instance of the requested presentation object, and the PO executes in a single-thread. In contrast, in a servlet application there is only one instance of any given servlet, and it is multithreaded (one thread for each request). So, POs can have member variables that are local to each instance, while in a servlet application, any member variable is global to all threads of the instance of the servlet.

Presentation objects have several features that you cannot take advantage of in a servlet application:

- Dynamic page recompilation (so you can change page content while an application is running)
- URL-encoding of session information for cookieless session maintenance
- Automatic setting of MIME-types, for applications that generate multiple document types (for example, HTML and WML)

The Enhydra application framework provides a number of capabilities that are very useful, including:

- Database management
- Logging

Although these are not part of the standard servlet application model, you can save a lot of development time by using them; however, your application will then be dependent on the Enhydra class libraries (contained in separate binaries - jar files).

Application layers

Regardless of the application model you use, you should divide your application into three distinct parts or *layers* for modularity and ease of maintenance:

- The **Presentation layer** handles how the application is presented to web browsers through HTML. In a super-servlet application, this layer consists of presentation objects (POs); in a servlet application, it consists of servlets.
- The **Business layer** contains business objects. Business objects contain the application's business logic, including algorithms and specialized functions, but not data access or display functions.
- The **Data layer** handles the interface with the persistent data source, which is typically a relational database.

An additional benefit of having a distinct data layer is that you can use the Data Object Design Studio (DODS) to create your data objects. DODS creates data objects to populate the data layer, and creates both java code and SQL code to create the corresponding tables in the database. For more on DODS, see “Data Object Design Studio (DODS)”, later in this chapter.

Note The Enhydra application framework only requires that you use an application object and presentation objects. The business and data classes you create are up to you. Dividing your application into these three layers minimizes maintenance cost because it isolates the application's data layer from the user interface. This, in turn, lets you change the data layer without affecting the presentation layer.

Servlet containers

Servlet container provides the services that an Enhydra application uses to communicate with the web server, and performs all other runtime functions. Enhydra is independent of servlet container – any servlet container (that supports Servlet 2.3 specification) can be used for running Enhydra applications (Tomcat, JOnAS Tomcat, JOnAS Jetty,…).

A servlet is a java class that dynamically extends the functionality of a web server. Normally, when a browser sends a request to a web server, the server simply finds the files identified by the requested URL and returns them to the browser. However, if the browser requests a page constructed by a servlet, the server sends the request information to a servlet, which constructs the response dynamically and returns it to the server.

The Java Servlet API is a standard extension to java and is a part of the Java 2 Enterprise Edition (J2EE). Some web servers support the Servlet API directly, while others require an adjunct servlet runner, such as JServ for the Apache Web server. Enhydra 3.0, and later, supports the Servlet API version 2.2.

Each Enhydra application runs as a single servlet, in contrast to a generic servlet application, which typically has one servlet for each dynamically-generated page.

Enhydra Application Framework

The Enhydra application framework includes:

- Presentation Manager

- Session Manager

- Database Manager

In general, the application framework includes all the classes in the *com.lutris.appserver.server.** packages, which provide the infrastructure that Enhydra applications use at runtime.

The general architecture of an Enhydra application in the context of the application framework is illustrated in Figure 3.2, “Enhydra application and Enhydra framework”.

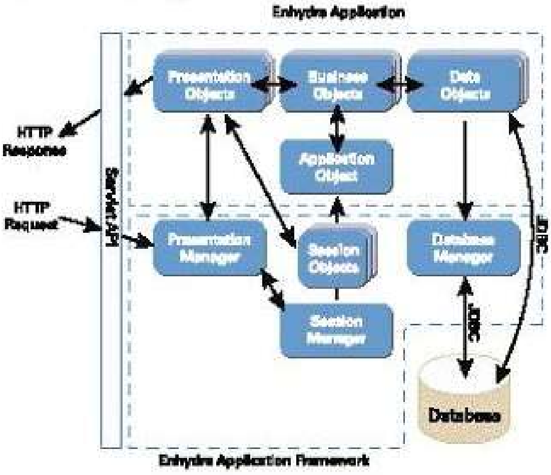


Figure 3.2 Enhydra application and Enhydra framework

Presentation Manager

The Enhydra Presentation Manager handles the loading and execution of the presentation objects in an Enhydra application. The Presentation Manager maps URLs to presentation objects and calls the *run()* method of the presentation object.

Each Enhydra application has one instance of a Presentation Manager. To increase performance, the Presentation Manager caches presentation objects and associated files in memory as necessary. The Presentation Manager also provides the key that the session manager uses to locate a session. This key is either a cookie or a string appended to each URL in the application.

Each application has a Presentation Manager that is an instance of the class:
com.lutris.appserver.server.httpPresentation.HttpPresentationManager .

The *com.lutris.appserver.server.httpPresentation* package contains classes and interfaces that the Presentation Manager and presentation objects use.

Session Manager

The Enhydra *Session Manager* enables an application to maintain state throughout a session. A *session* is defined as a series of requests from the same user (browser client) during a specified time period. Enhydra provides a general implementation of session management that you can extend to create more sophisticated state models.

Enhydra maintains user state by creating a Session object for each user. When a user first makes a request to an application, the Session Manager creates a new Session object and assigns it a unique session ID. The Session Manager uses the session ID to retrieve the Session object for subsequent requests. Applications can add user-specific information to the Session object and then access the Session object from the request object, as it is passed through the application.

If a user has been idle (has not issued a request to the application) for more than the period specified in the configuration file, the user's session becomes invalid, and the Session Manager releases the corresponding Session object. This makes it possible to implement security schemes that require users to log in before accessing the application. In such a scheme, the user enters an appropriate password and gains access to the rest of the application; however, once the user's session has been idle for more than the allowed time, the application requires the user to log in again.

Before version 6.x each application had a Session Manager that is an instance of the class *com.lutris.appserver.server.session.StandardSessionManager*. When it was created, the Session Manager read the maximum time that a session can persist, the maximum session idle time, and other related information from the application configuration file (*<appName>.conf* or *web.xml* file). The *com.lutris.appserver.server.session* package contained classes and interfaces that the Session Manager and the application used for session management.

Sessions in Enhydra 6 should be managed by the servlet container, the session data is being stored in the *HttpSession* object. To achieve this, and to improve the flexibility of the enhydra framework, different implementations of the session manager have been developed. The class that implements the one used with each of the applications is to be stated in the configuration file, for example:

```
SessionManager.Class=
        com.lutris.appserver.server.sessionEnhydra.SimpleServletSessionManager
```

If the *SessionManager.Class* is not specified, the old `com.lutris.appserver.server.sessionEnhydra.StandardSessionManager` is used (it is kept for compatibility reasons, but it's use is not encouraged).

The available session manager adapters are:

- *com.lutris.appserver.server.sessionEnhydra.SimpleServletSessionManager*
  This is a simple session manager which interconnects the servlet container sessions and enhydra sessions by using the same session keys (generated by session container).

- *com.lutris.appserver.server.sessionContainerAdapter.ContainerAdapterSessionManager*
  Simple session manager to be used with servlet container capable of managing their sessions. It uses HttpSession to keep the session data. The sessions are completely managed by the session container and are configured in the servlet container configuration files. Any session configuration parameters defined in the application configuration file are ignored (except SessionManager.Class). The persistence across restarts of the application and container is realized by the appropriate servlet container mechanisms.

- *com.lutris.appserver.server.sessionContainerAdapter.TomcatContainerAdapterSessionManager*
  Tomcat specific session manager, extends ContainerAdapterSessionManager, witch interacts directly with (wraps) Tomcat Session Manager.

- *com.lutris.appserver.server.sessionContainerAdapter.JmxContainerAdapterSessionManager*
  Tomcat specific session manager, extends ContainerAdapterSessionManager, by using JMX MBeans to obtain some session information from the session container.

Note: for the latter two containers, session data must be serializable in order to fully utilize the persistence obtained by the session container.

Database Manager

The Enhydra Database Manager controls a pool of database connections for the application. The Database Manager works with logical databases. A logical database is an abstraction that hides the differences between different database types. A logical database uses Java Database Connectivity (JDBC) to communicate with database servers such as Oracle, Sybase, Informix, Microsoft SQL Server, PostgreSQL, InterBase, and InstantDB.

The Database Manager is responsible for the state of a database connection, the SQL statements that are being executed, and the result sets that are in progress.

Specifically, the Database Manager:

- Allocates and releases connections to the logical database

- Allocates object IDs from the logical database

- Creates queries and transactions

- Maintains other database-related information

Each application has a Database Manager that is an instance of the class *com.lutris.appserver.server.sql.StandardDatabaseManager*. When it is created, the Database Manager reads a configuration file that specifies the logical database to use, the actual database types to which it maps, and other related information.

The *com.lutris.appserver.server.sql* package contains the classes and interfaces that the Database Manager and data objects use.

Enhydra tools

Enhydra includes the following tools to help you create applications:

- Enhydra Application Wizard
- Extensible Markup Language Compiler (XMLC)
- Data Object Design Studio (DODS)

Enhydra Application Wizard

The Enhydra Application Wizard (*appwizard*) is a tool with both a command-line and a graphical user interface. The wizard creates a basic framework for an Enhydra application. The wizard lets you create and run a new "stub" application in a matter of minutes, giving your development project a jump-start. For an example of using the Application Wizard GUI, see "Creating your first application" in the chapter 4.

Note The Application Wizard has changed significantly with the release of Enhydra 3.5. Previously, the Application Wizard was a command-line tool, started by entering 'newapp' with a project name parameter. The command for starting the Application Wizard and the parameters required to run it as a command-line tool have changed. The basic framework of files and directories generated by the Application Wizard has changed as well.

Extensible Markup Language Compiler (XMLC)

The *Extensible Markup Language Compiler* (XMLC) creates a java object that mirrors the structure of an eXtensible Markup Language (XML) document. *XML*, defined by the World Wide Web Consortium (W3C), is the universal format for structured documents and data on the Web. XMLC uses the Document Object Model (DOM), a W3C standard interface, to let programs access and update the content and structure of XML documents.

Note Although XMLC works with XML documents, this book will focus on its use with HTML pages.

XMLC lets you separate HTML templates in your application. These templates are typically created by page designers from java code, which is usually created by programmers. This functionality provides increased modularity and eases team development and application maintenance. Page designers can change the user interface of the application without requiring any code changes, and the programmers can change the "back-end" Java code without requiring any changes to the HTML.

This command-line tool generates a Java class file from a HTML input file. An application can use the Java class at runtime to change the content or attributes of any tags with ID or CLASS attributes. For an example using the XMLC, see "Tutorial: Building Enhydra applications" in the chapter 4.

Auto Reloading, Memory Persistence and XMLC Deferred Parsing

XMLC Deferred parsing enables you to change a HTML page in a Web application at runtime (which means without stopping the application). So, if you want a new look on a page, you can change desired HTML page in the application. The application detects the new timestamp on the file, dynamically loads the page, and uses it in the application. Configuring deferred parsing to work is also simple once you understand the details behind it. Deferred parsing comes with XMLC2.2 and it represents replacement for the earlier possibility of XMLC, auto recompilation. For more information about deferred parsing refer to XMLC2.2 documentation.

Automatic Reloading also enables you to change the Web application at runtime (which means without stopping the application). This changes can be seen (by web users) only if the application, or parts of application that had been changed, are compiled or transformed from HTML form into Java classes by XMLC, and then put in the place of old application's classes.

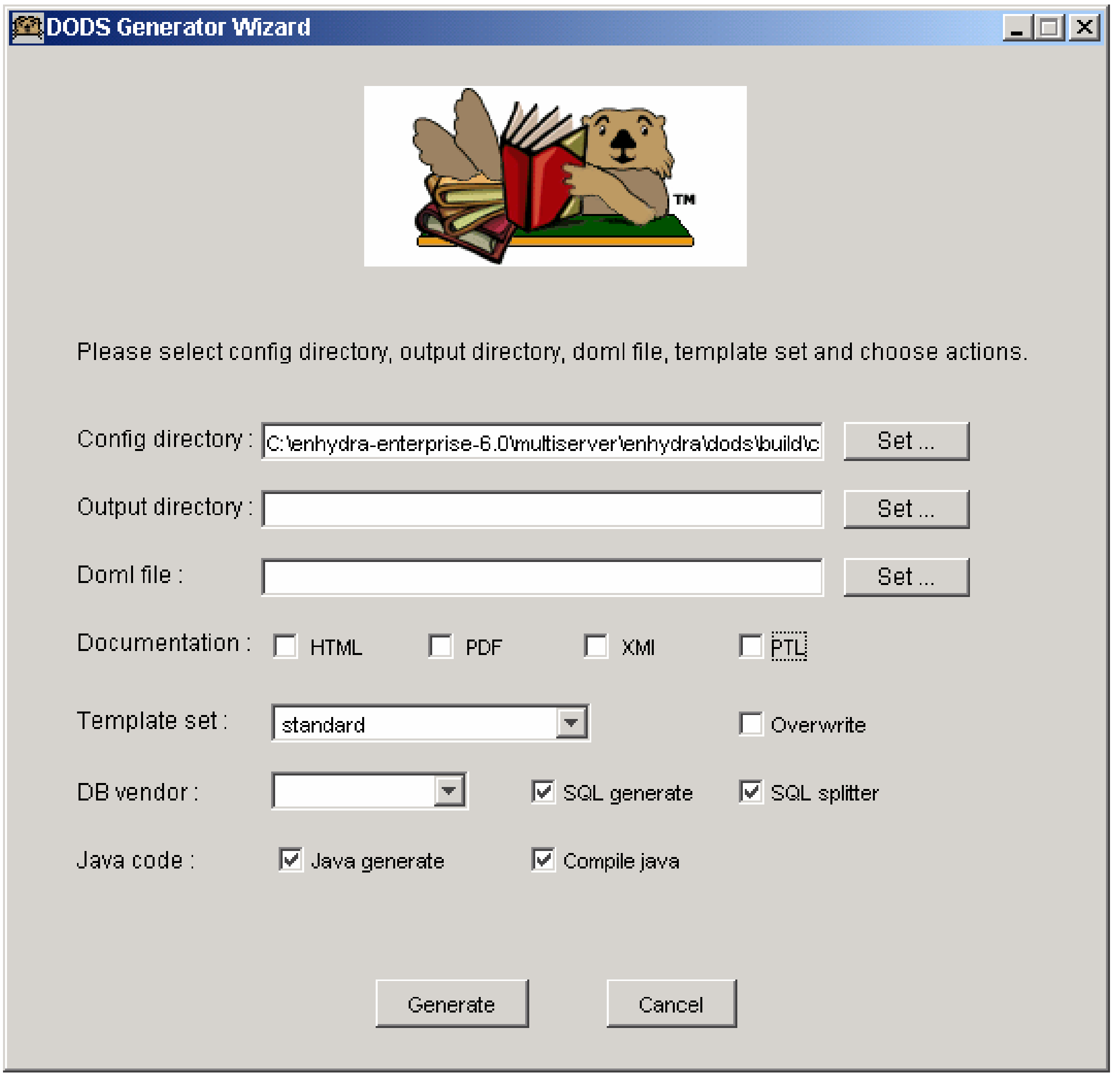
Both Deferred parsing and Automatic Reloading enable changes of the Web application without stopping the application and then restarting it, so that Web users, except when the application is seriously changed, can not notice any change in the Web application. The changes in particular Web page are applied when the page is called by HTTP request for the first time. These two approaches can work together. The difference between these two approaches is that in XMLC Deferred Parsing the changed HTML pages are dynamically loaded into the system, during the runtime, as:
org.w3c.dom.Document/org.w3c.dom.html.HTMLDocument
without having to pre-compile them by using XMLC, while before performing the Automatic Reloading, it is necessary to compile the changed HTML files (and/or other changed application classes).

Enhydra's possibility called Memory Persistence is used in order to hold application's session objects in memory while particular application is stopped, or during the restart of the application. When application is restarted, it can use its old session data (data stored in session object) as they were left in the time the application had been stopped. Of course, it is assumed that used servlet container wasn't stopped in the meantime.

This is explained detailly in document "*enhydra_utilities*" in
<enhydra_root>/multiserver/webapps/enhydra-docs/enhydra directory, in html and pdf form.

Data Object Design Studio (DODS)

The Data Object Design Studio (DODS), shown in Figure 3.3, is a tool which, for the given doml file, can generate SQL script files for creating tables (for each table separately and one cumulative file for creating all tables), one file for deleting all tables, and/or java code for data objects described in the given doml file. DODS also has possibility to compile generated java classes and to parse SQL files (to split cumulative SQL into more separated SQL files using SQLSplitter tool).



Please select config directory, output directory, doml file, template set and choose actions.

Config directory : C:\enhydra-enterprise-6.0\multiserver\enhydra\dods\build\c

Output directory :

Doml file :

Documentation : ☐ HTML ☐ PDF ☐ XMI ☐ PTL

Template set : standard ☐ Overwrite

DB vendor : ☑ SQL generate ☑ SQL splitter

Java code : ☑ Java generate ☑ Compile java

Figure 3.3 DODS Generator Wizard

Data objects described in the given DOML file correspond to tables in the database. Each data object has *attributes*, which describe database columns, and *reference attributes*, which refer to other data objects. Reference attributes let you create a hierarchy of data objects (for example, many-to-one or many-to-many relationships).

For the given DOML, DODS generates all of the code to implement it. For example:

- SQL code to define the database tables
- Java code to create the corresponding application data objects

For each data object, DODS generates a set of source files. For example, if your DOML file includes the definition of an entity named “thing,” then DODS would generate the following:

- A file named *thing.sql* containing the SQL CREATE TABLE command to construct a table in a relational database.
- Java source file defining a data object representing a row in the table.

  This class provides a “set” and “get” method for each attribute, methods to handle caching, and is a subclass of the Enhydra framework class *GenericDO*. In this example, the class would be named *ThingDO*.

- Java source file that defines a query class, which provides SQL query access to the database table.

  The query class returns a collection of *ThingDO* objects that represent the rows found in the table matching criteria passed from the application.

  For an example using DODS, see “Using DODS” in the chapter 4.

Chapter 4

Tutorial: Building Enhydra applications

This chapter describes how to build an Enhydra application from the ground up, and provides important tips on Enhydra application development. In this tutorial, you will:

- Use the Application Wizard to create a starting framework
- How to configure application
- Use Admin applications for Enhydra applications
- Use XMLC to expand the application
- Use DODS for database access

If you are already familiar with the basics of Enhydra, you may want to skip to the chapter 5, “DiscRack sample application,” for a look at an application with more advanced features.

Creating your first application

The *Enhydra Application Wizard* (sometimes referred to as *appwizard*) is a quick way to get up and running with Enhydra. The Application Wizard generates basic Java files and directory structures for new applications. For the tutorial, you will use the Application Wizard GUI.

To create a simple application with the Application Wizard:

1 Create a directory to contain your new application and name it anything you want. For example in directory <enhydra_root>/multiserver/enhydra, create subdirectory *myapps*:

```
mkdir myapps (UNIX)
```

or

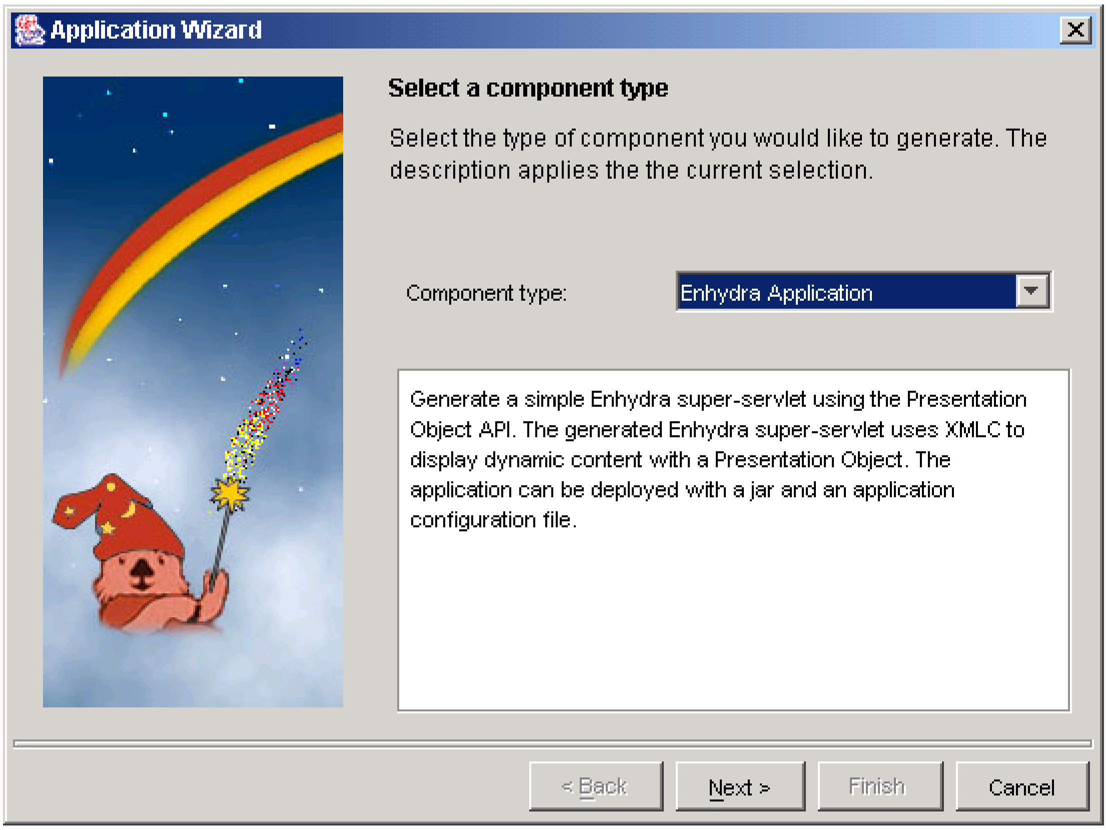
```
md myapps (WINDOWS)
```

2 Open a shell window (UNIX) or command window (WINDOWS) and make the new directory the current directory. For example:

```
cd myapps
```

3 Start the Application Wizard GUI by entering *appwizard* at the command prompt (or use Start menu to start *appwizard*). Entering *appwizard* at the command prompt with no arguments brings up the Application Wizard GUI. The Application Wizard can generate two distinct types of Enhydra projects: a Enhydra super-servlet application, and an Web Application. For this tutorial, you will generate a super-servlet application.

Note If the Application Wizard does not start, the path environment variable is not set correctly. The Enhydra installation instructions provide information about setting your path environment variable. The installation instructions are available in HTML and PDF format in Enhydra documentation, *<enhydra_root>/multiserver/webapps/enhydra-/doc* directory (refer to the top-level *index.html*).



Select a component type

Select the type of component you would like to generate. The description applies the the current selection.

Component type: Enhydra Application

Generate a simple Enhydra super-servlet using the Presentation Object API. The generated Enhydra super-servlet uses XMLC to display dynamic content with a Presentation Object. The application can be deployed with a jar and an application configuration file.

Figure 4.1 Application Wizard GUI

Note The Application Wizard has changed significantly with the release of Lutris Enhydra 3.5. Previously, the Application Wizard was a command-line tool, started by entering newapp with a parameter for the project name. The command for starting the Application Wizard and the parameters required to run it as a command-line tool have changed. The basic framework of files and directories generated by the Application Wizard has changed as well.

4 Use the Application Wizard GUI to generate a simple Enhydra application. The Application Wizard GUI steps you through the process of generating an Enhydra project.

1 Select a Component type.
Select **Enhydra Application** from the Component type pull-down menu and click Next.

2 Specify Client type and directory details.
Accept the default client type of **HTML**. Enter **simpleApp** for the Project directory name. Enter **simpleapp** (note the difference in case) for Package. Set the Root path to, for example:

<enhydra_root>/multiserver/enhydra/myapps.

3 Specify the copyright material to use.
Click Next to accept the default, No copyright setting.

4 Specify which Supplemental files to generate.
Select Generate start script and command line build files and click Finish.

The Application Wizard creates a new directory called *simpleApp***.** This directory is sometimes referred to as the *application root directory*.

5 Make the application root directory the active directory:

```
cd simpleApp
```

6 Application root directory has EJOSA folder structure (for more information about EJOSA, see http://ejosa.sourceforge.net/). Browse the application root directory and note the following items created by the Application Wizard:

- file **build.xml**
- file **readme.html** that contains some simple instructions to build and run the application
- **specification** directory that contains source (interfaces and classes); can be used for independent development of project. You can make presentation and specification layer only. In this case specification layer is changed for business layer. Specification layer also can be used with business layer to replace data layer in development stage of application
- **data** directory that contains source that implements data logic
- **business** directory that contains source that implements business logic
- **presentation** directory that contains source that implements presentation logic
- **application** directory that contains batch files (*start-enhydra.bat* and *stop-enhydra.bat*) to start or stop application on Windows, and also shell scripts (*start-enhydra* and *stop-enhydra*) to start or stop application on Linux.

Building and starting the application

1 To build the application:

In the shell window (UNIX) or in the command window (WINDOWS), enter the command enhydra-ant from the application root directory.

During the Enhydra installation, in your path should be added path to the *<enhydra_root>/multiserver/enhydra/bin* directory which contains *enhydra-ant.bat* (*enhydra-ant* shell script for Linux):

```
cd /myapps/simpleApp
enhydra-ant
```

This process will create *bin* directory in *simpleApp/application/* directory that practically presents CATALINA_BASE directory structure (JONAS_BASE for Enhydra Enterprise) which contains application binaries and configuration files place in *webapps* (*webapps/autoload*) subdirectory.

File *build.xml* contains directives that tell *ant* to recursively descend the application directory tree. When you build the application *ant* compiles the files located in the simpleApp source directories and creates a corresponding classes directory structure.

Optionally, command:

1. **UNIX**

```
./enhydra-ant makeWAR
```

2. **WINDOWS**

```
enhydra-ant makeWAR
```

additionally makes simpleApp.war in *simpleApp/application/bin/webapps* directory. This war will have all nedded files from *presentation*, *data*, *business* and *specification* folder.

2 To start the application, enter the following commands in the Enhydra shell (UNIX) or in the command window (WINDOWS).

1. **UNIX**

```
cd application
cd bin
./start-enhydra
```

2. **WINDOWS**

```
cd application
cd bin
start-enhydra
```

3 To stop the application, enter the following commands in the Enhydra shell (UNIX) or in the command window (WINDOWS).

1. **UNIX**

```
./stop-enhydra
```

2. **WINDOWS**

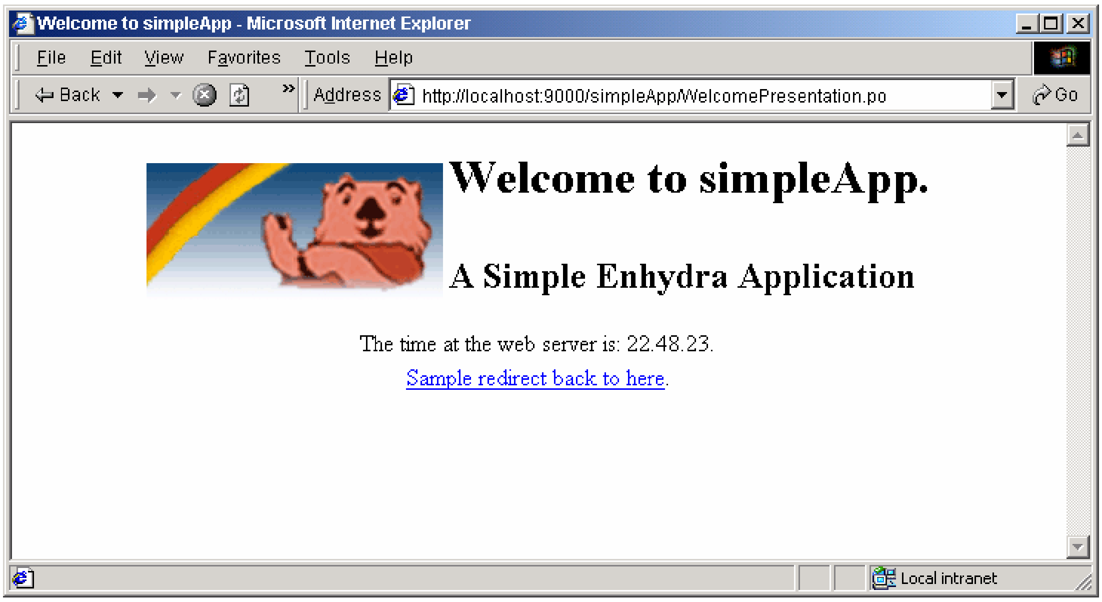
```
stop-enhydra
```

4 To access the application, enter the following URL in your browser's location field:

http://localhost:8080/simpleApp
http://localhost:9000/simpleApp

Note: It is assumed that Enhydra 6.x is installed with default connection port setting (8080 for Enhydra and 9000 for Enhydra Enterprise installation).

The browser will display the Welcome page for the simpleApp application.



http://localhost:9000/simpleApp/WelcomePresentation.po

Welcome to simpleApp.

A Simple Enhydra Application

The time at the web server is: 22.48.23.

Sample redirect back to here.

Figure 4.2 Browser with the simpleApp welcome page loaded

You have just built and run your first Enhydra application.

You should see "Welcome to simpleApp..." and the current time. Hit the reload button and watch the time update. The time is a sample of dynamic HTML: It could be replaced with anything you can access or compute in java.

Clicking on the link "Sample redirect back to here" sends the browser to a special page that always responds with a redirect request back to the main page. Sometimes this occurs so quickly you may not notice anything happening. Redirects are very useful when you are building a Web application.

How it works

The application created by the Application Wizard provides a simple example of how Enhydra works.

Look at the file *myapps/simpleApp/presentation/resource/Welcome.html*, which contains a few dozen HTML tags. Notice tags such as these:

```
<center>
 The time at the web server is:
 <span id="time">1/1/00 00:00:00 (static)</span>.
</center>
```

At runtime, Enhydra replaces the content of the <SPAN> tag with a date. The text in the ID attribute is just a placeholder; it will never appear at runtime. The period outside the <SPAN> tag will not be replaced. Thus, the sentence will always end with a period.

Look also at the *WelcomePresentation.java* file in the directory:
myapps/simpleApp/presentation/src/simpleapp/presentation
In particular, notice these lines of code:

```
WelcomeHTML welcome;
String now;
...
welcome = (WelcomeHTML)comms.xmlcFactory.create(WelcomeHTML.class);
  try {
    Dater dater =
                   DaterFactory.createDater("simpleapp.business.DaterImpl");
    now = dater.getDate();
  } catch (Exception ex){ . . . }
  welcome.setTextTime(now);
  comms.response.writeDOM(welcome);
```

This code is used for replacing the text inside the <SPAN> tags.

The first couple lines in the code snippet define *welcome* as an instance of the *WelcomeHTML* class, and *now* as an instance of the String class. The xmlcFactory in the next line is used to instantiate your HTML page. Next, the variable now is set to the current Date formatted as time. The last line of the snippet sets the time element in the welcome class to the value of now, and returns the value.

When you build the application, the Extensible Markup Language Compiler (XMLC) finds the <SPAN> tag in the HTML and recognizes the ID attribute with value “time”. It creates a Java class called WelcomeHTML with a method *getElementTime()*. The application uses getElementTime() to modify the text content of the <SPAN> tag.

Note In general, XMLC will create a *getElementxxx()* method for each <SPAN> tag with ID attribute value *xxx*. The *xxx* in the method name is replaced by the capitalized spelling of the ID attribute value of the SPAN tag.

At runtime, the application replaces the original text content of the <SPAN> tag with a string representation of the current date. Then, the call to *write* method writes the document out to the HTTP response, looking something like this:

```
...
<CENTER>
 The time at the web server is: <SPAN>10:40 AM</SPAN>.
</CENTER>
...
```

For a more detailed explanation of XMLC, see "Using XMLC", later in this chapter

Configuration file

The application configuration file contain critical information that determine how an Enhydra application runs. Application configuration file can be *<AppName>.conf* or application's *web.xml* file (default).

Application configuration is explained detailly in document "*configuration*" in *<enhydra_root>/multiserver/webapps/enhydra-docs/configuration* directory, in html and pdf form.

Application administration

For managing Enhydra applications are used Tomcat Administration or JOnAS Administration web application and Enhydra Admin application.

These applications are graphical tools that allow a system manager to configure and monitor an instance of Enhydra and associated applications. All configuration information for Enhydra and Enhydra applications are stored in configuration files. When Enhydra starts, it reads these configuration files and starts the server process and any specified applications. Once the instance of Enhydra is running, the administration applications are able to perform management operations. All management operations work in the same manner; the active state (including resource parameters) of Enhydra, an application, or servlet is changed and the change may be saved in the configuration file.

When Enhydra is installed, all Enhydra applications are placed in <enhydra_root>/multiserver/webapps (<enhydra_root>/multiserver/webapps/autoload for Enterprise installation) directory as deployed (unpacked) war files, where <enhydra_root> is the directory where Enhydra is installed.

Tomcat Admin (url *http://localhost:<communication_port>/admin*) and JOnAS admin (url *http://localhost:<communication_port>/jonasAdmin*) are graphical tools based on the Struts framework and the JMX technology.

Note: Communication port value is one of Enhydra 6.x installation options. Default value is set to 8080 for Enhydra and 9000 for Enhydra Enterprise installation.

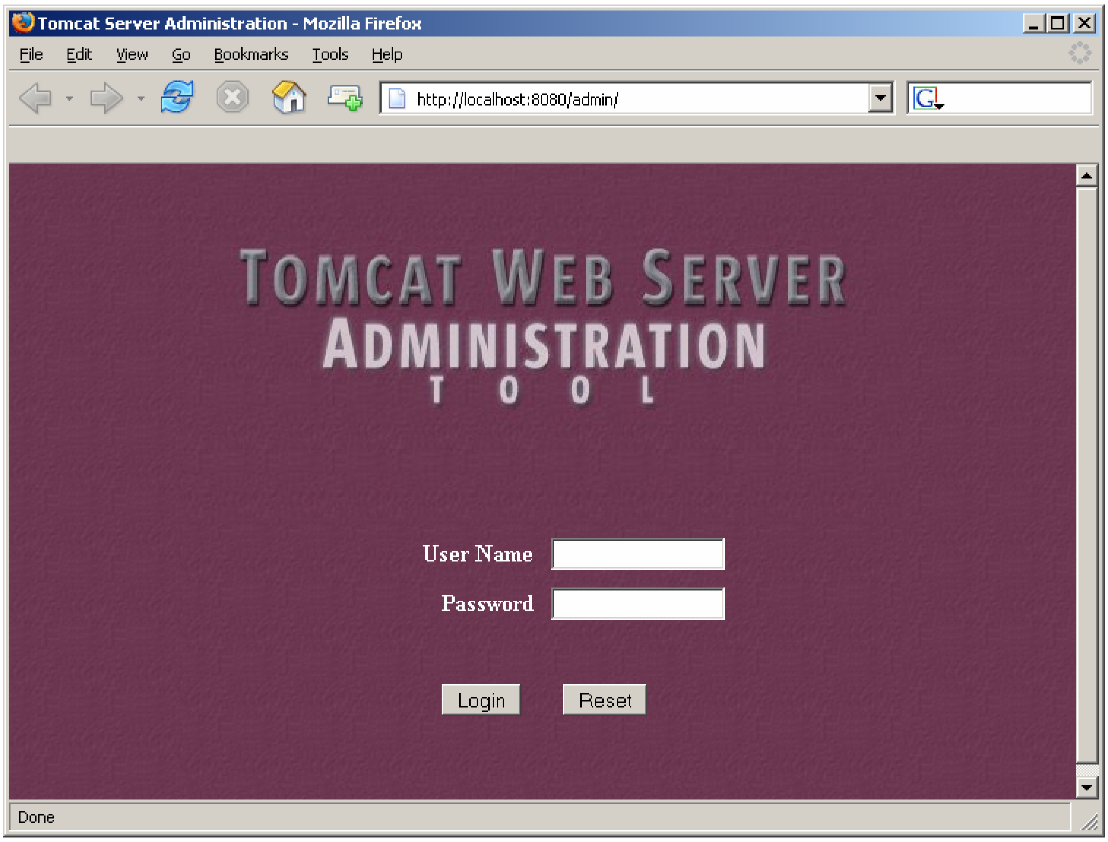

Tomcat Server Administration - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8080/admin/
TOMCAT WEB SERVER
ADMINISTRATION
TOOL
User Name
Password
Login
Reset
Done

Figure 4.3a Tomcat Admin application

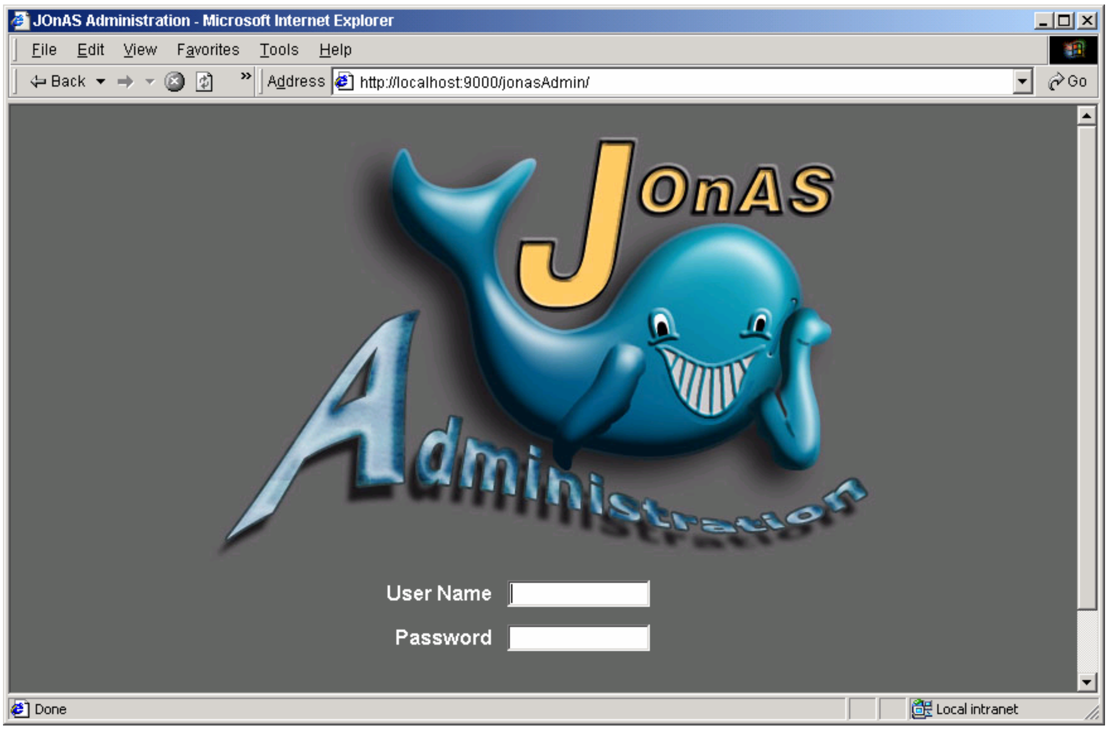

JOnAS Administration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address
http://localhost:9000/jonasAdmin/
Go
JOnAS
Administration
User Name
Password
Done
Local intranet

Figure 4.3b JOnAS Admin application

To enter Tomcat Admin or JOnAS Admin, please enter (appropriate) values for User Name and Password.

Note: Administration User Name and Password values are Enhydra 6.x installation options. Default values are set to 'admin' (for User Name value) and 'enhydra' (for Password value).

Then you get the main Tomcat and JOnAS Admin window:

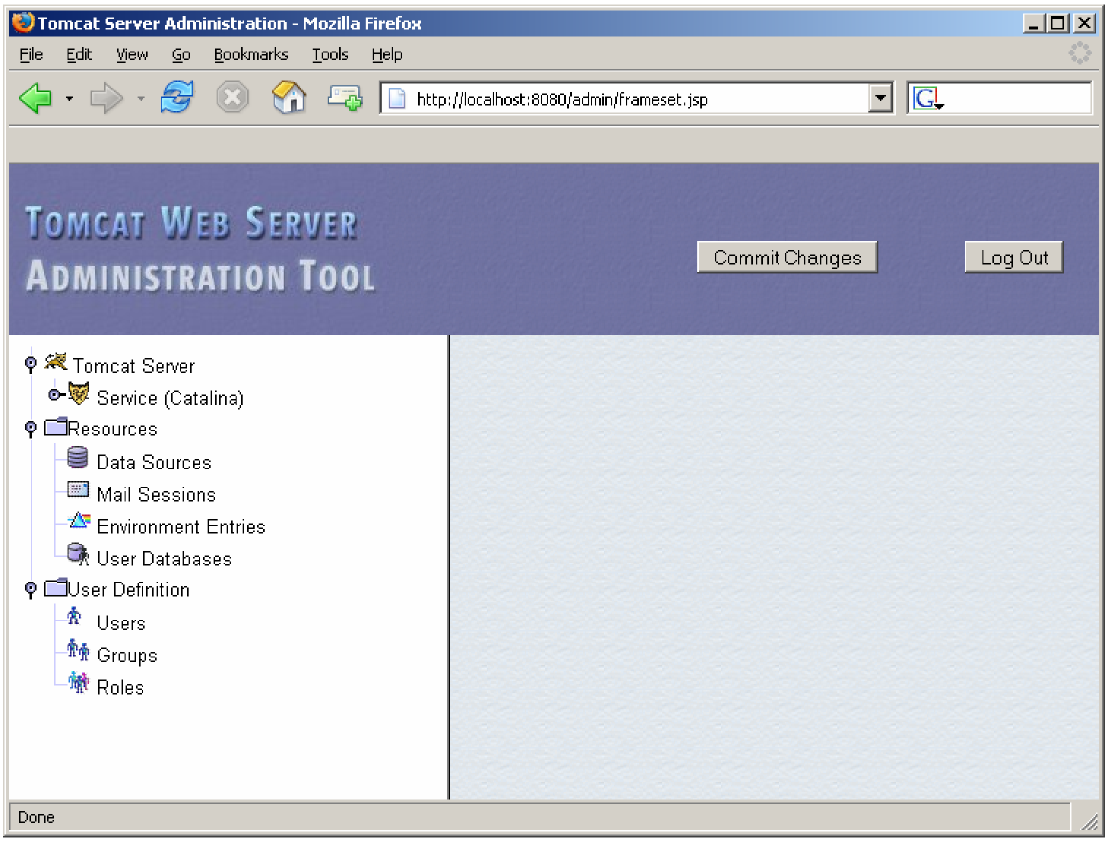


Figure 4.4a Tomcat Admin main window

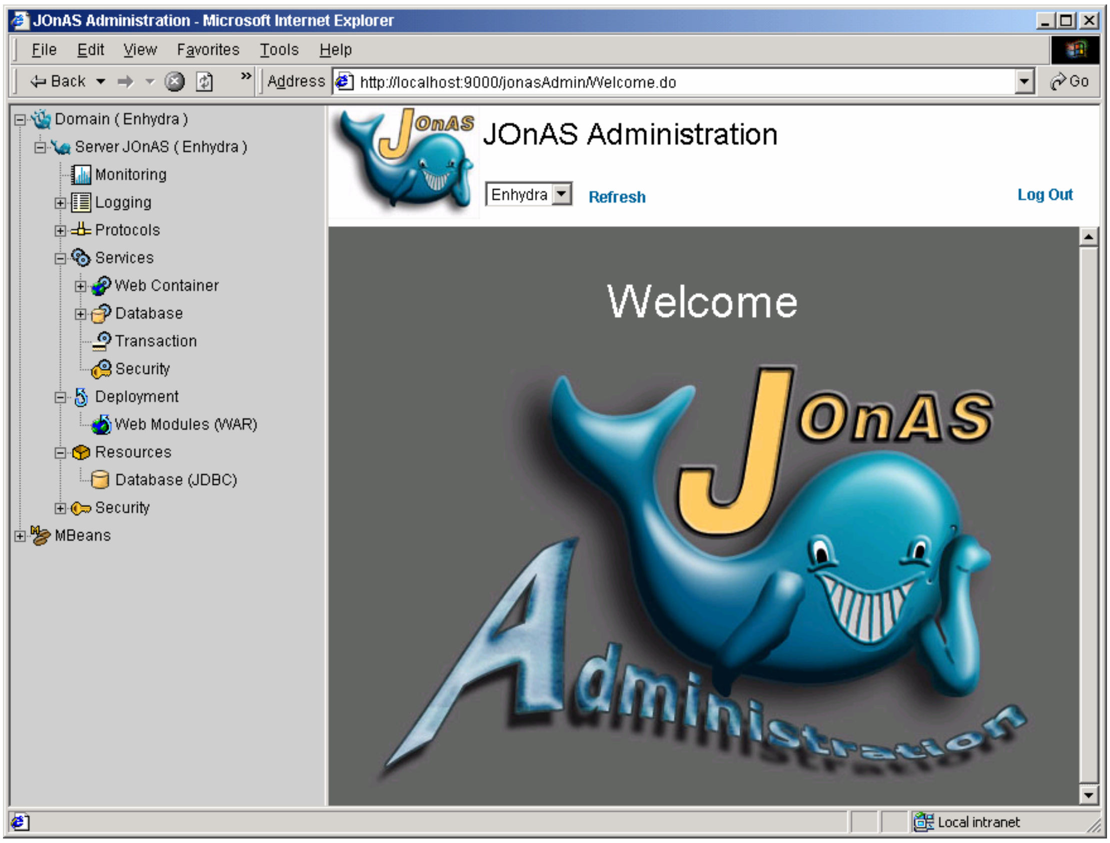
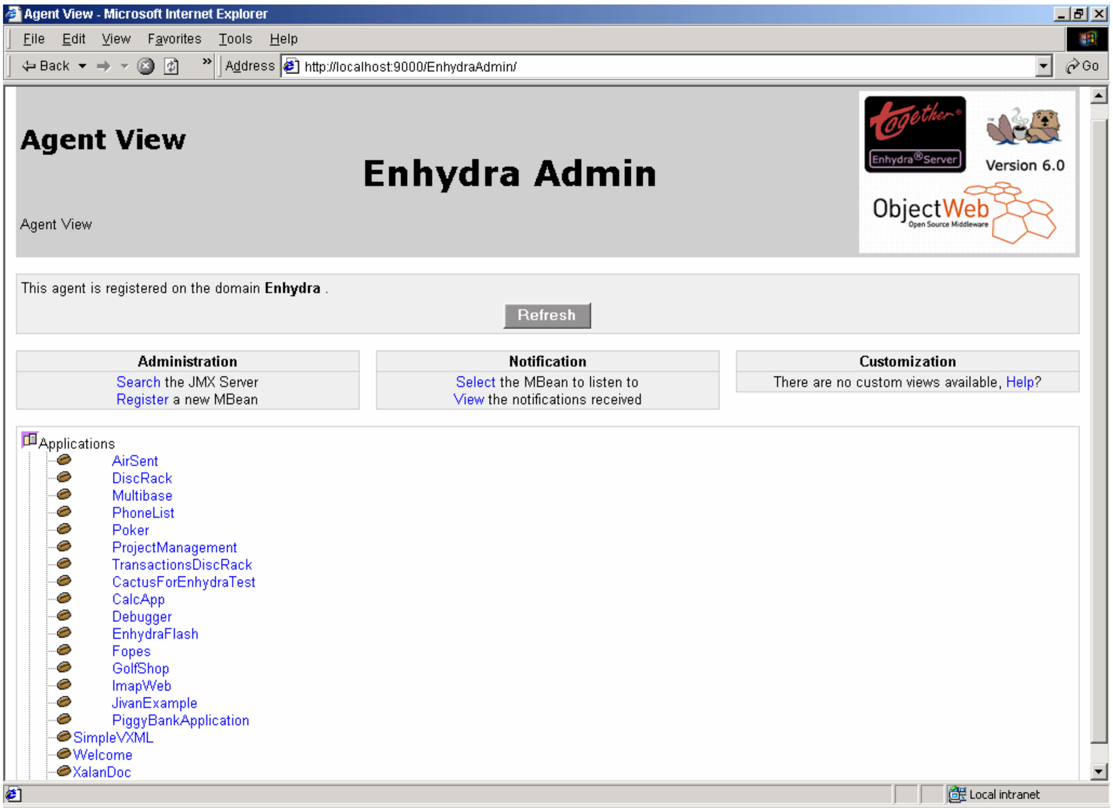


Figure 4.4b JOnAS Admin main window

Enhydra Admin application (url *http://localhost:<communication_port>/EnhydraAdmin*) is also graphical tools based on the Struts framework and the JMX technology.

Note: Communication port value is one of Enhydra 6.x installation options. Default value is set to 8080 for Enhydra and 9000 for Enhydra Enterprise installation.



This agent is registered on the domain **Enhydra** .

Figure 4.5 Enhydra Admin

It is a graphical tool that enables modification of the operational attributes for Enhydra (application or servlet) and cache administration.

Application administration is explained detailly in document "*enhydra_app*" in *<enhydra_root>/multiserver/webapps/enhydra-docs/enhydra* directory, in html and pdf form.

Using XMLC

XMLC, the Extensible Markup Language Compiler, was introduced in the chapter 3, "Overview". It is a powerful tool that you can use to create applications that have a clean separation between the user interface and the back-end programming logic.

Note In general, XMLC can work with XML pages, but for practical reasons, the remainder of this chapter focuses on how it works with HTML pages.

XMLC parses a HTML file and creates a Java object that enables an application to change the HTML file's content at runtime, without regard for its formatting. The java objects that XMLC creates have interfaces defined by the Document Object Model (DOM) standard from the World Wide Web Consortium (W3C).

Adding a hit counter

To get a feel for how XMLC works, you are going to extend your application to display a “hit counter” that shows the number of users who have accessed it.

1 Find the files *Welcome.html* and *WelcomePresentation.java* in the presentation directory.

2 Add the following line of HTML to *Welcome.html* before the closing the last </CENTER> tag:

```
<P>Number of hits on this page: <SPAN ID="HitCount">no count</SPAN>
```

The ID attribute tells XMLC to generate an object corresponding to the <SPAN> tag, so that it can replace the text “no count” at runtime.

3 Add the lines of code shown in **bold** in the following code sample, to WelcomePresentation.java:

```
 //Add the following line
static int hitCount=0; //All Welcome PO's will share this

 public void run(HttpPresentationComms comms)
  throws HttpPresentationException, IOException {

  WelcomeHTML welcome;
  String now;

  welcome = (WelcomeHTML)comms.xmlcFactory.createWelcomeHTML.class);
  try {
    Dater dater =
       DaterFactory.createDater("simpleapp.business.DaterImpl");
    now = dater.getDate();
  }catch (Exception ex){ . . . }

  welcome.setTextTime(now);
//Increment the count and write into the html
//Add the following line
  welcome.setTextHitCount(String.valueOf(++hitCount));
  comms.response.writeDOM(welcome);
 }
}
```

4 Build the application by running enhydra-ant from the top-level simpleApp directory and then restart Enhydra. To do this, enter the following commands in the shell window (UNIX) or in command window (WINDOWS):

1. **UNIX**:

```
cd /myapps/simpleApp
enhydra-ant
cd application
cd bin
./start-enhydra
```

2. **WINDOWS**:

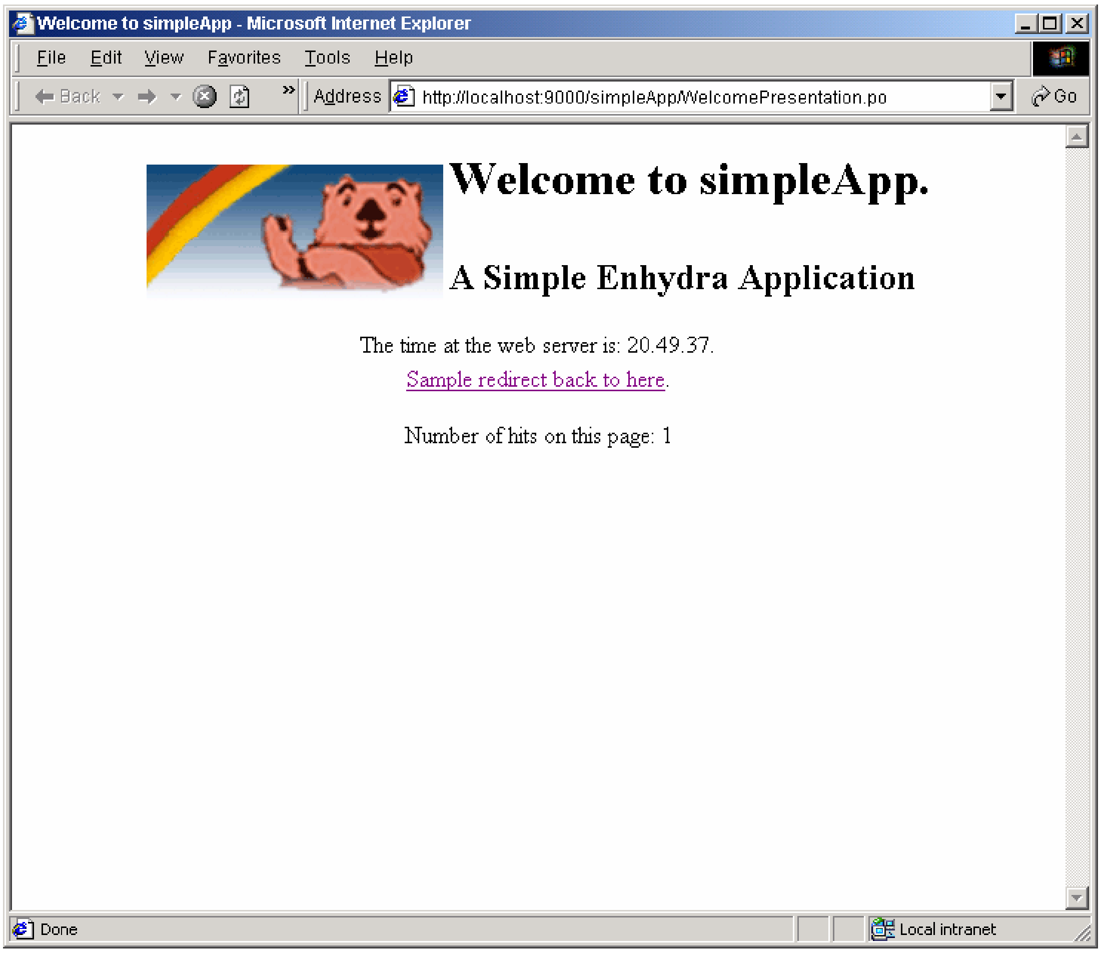
```
cd /myapps/simpleApp
enhydra-ant
cd application
cd bin
start-enhydra
```

Building the application with enhydra-ant runs XMLC on all HTML files in the application, in this case, just Welcome.html.

5 Test the application by loading http://localhost:8080/simpleApp (http://localhost:9000/simpleApp) in your browser.

Note: Communication port value is one of Enhydra 6.x installation options. Default value is set to 8080 for Enhydra and 9000 for Enhydra Enterprise installation.

The browser will display the Welcome page for the simpleApp application. The Welcome page should now have a hit counter beneath the redirect link, as shown in Figure 4.6.


Welcome to simpleApp - Microsoft Internet Explorer
File
Edit
View
Favorites
Tools
Help
Back
Address
http://localhost:9000/simpleApp/WelcomePresentation.po
Go
Done
Local intranet

Welcome to simpleApp.

A Simple Enhydra Application

The time at the web server is: 20.49.37.

Sample redirect back to here.

Number of hits on this page: 1

Figure 4.6 Browser displaying the simpleApp Welcome page with a hit counter

The page now displays the number of times it has been accessed.

6 Reload the page several times to verify that it works correctly.

The count should increment each time you access the application.

The application is doing two things:

- Storing the hit count in *hitCount*, a static property of the Welcome presentation object
- Writing the hit count to the Web page with the *setTextHitCount()* method

Recall that the PresentationManager instantiates a presentation object for each request. So, the WelcomePresentation class is instantiated once per browser request. Because hitCount is a static property, it is shared by all WelcomePresentation objects and its value gets incremented by each request.

In the same way that it added a *getElementTime()* method for the `<SPAN ID="time">` tag, XMLC creates a *setTextHitCount()* method for the `<SPAN ID="HitCount">` tag. The application then uses the *setTextHitCount()* method to write the value of hitCount into the page, within the corresponding `<SPAN>` tag.

Note XMLC creates the WelcomeHTML class, but by default it deletes the java source file.

Understanding the Document Object Model

HTML documents have a hierarchical or tree-like structure that can be modeled in an object-oriented language like java. The Worldwide Web Consortium (W3C) standard for the XML/HTML object model is called the Document Object Model (DOM).

Enhydra applications use the DOM to manipulate HTML content at runtime. For example, consider the following HTML:

```
<TABLE>
  <TR>
    <TD ID="cellOne">Shady Grove</TD>
    <TD ID="cellTwo">Aeolian</TD>
  </TR>
  <TR>
    <TD ID="cellThree">Over the River, Charlie</TD>
    <TD ID="cellFour">Dorian</TD>
  </TR>
</TABLE>
```

This HTML snippet has a `<TABLE>` tag that contains `<TR>` tags, which in turn contain `<TD>` tags containing text (or data). This defines a tree-like hierarchy, as illustrated in Figure 4.7.

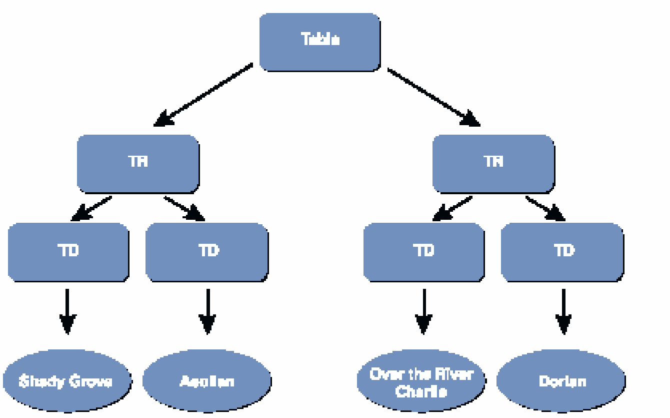


Figure 4.7 DOM tree of HTML

Each box or ellipse in this figure is a *node* in the tree. The node above another node in the hierarchy is called its *parent*. The nodes below the parent are called its *children*.

Some nodes (like HTML tags) have attributes (for example, a table cell has a background color attribute). W3C defines packages and interfaces that mirror the object hierarchy of nodes in an HTML document. In addition, XMLC includes an API for changing attribute values.

For example, use code like the following to set the color of one of the table cells:

```
HTMLTableCellElement cellOne = theDocument.getElementCellOne();
cellOne.setBgColor("red");
```

In this example, the class *HTMLTableElement* and the method *setBgColor()* come from the

W3C packages; *getElementCellOne()* comes from XMLC.

This code illustrates one important thing that XMLC does - create methods to access nodes in the DOM. XMLC generates the *getElementxxx()* methods that return objects corresponding to tags with ID attributes. You could change the color of a table cell with the W3C classes alone, but your code would have to traverse the DOM tree, so it would be more laborious.

SPAN and DIV tags

`<SPAN>` and `<DIV>` are HTML tags that you may not be familiar with. They are typically used to apply styles using cascading style sheets (CSS). Outside of that, they are largely ignored by browsers. However, XMLC makes extensive use of them.

- Use the `<SPAN>` tag to enclose a block of text that you want to replace at runtime.

  In general, a `<SPAN>` tag can enclose any text or inline tag. An *inline tag* is any tag that does not cause a line break in the layout; for example, `<A>` (anchor) or `<B>` (bold) tags.

  Note Do not use `<SPAN>` tags to enclose other tags, such as `<TABLE>` or `<P>` (paragraph).

- Use the `<DIV>` tag to enclose block tags, such as `<TABLE>`, that cause a line break in the HTML layout.

Using XMLC from the command line

Previously, you ran XMLC implicitly when you built the project with enhydra-ant.

The basic command-line syntax of XMLC is:

```
xmlc -options file.html
```

where options is a set of command-line options, and file is the name of the input file.

There are several dozen command-line options. In this section, we introduce three immediately useful ones: *-dump*, *-class*, and *-keep*.

-dump option

The -dump option makes XMLC display the DOM tree for a document. This is primarily useful as a learning tool; once you are familiar with XMLC, you will rarely use it.

1 Create a new file called *Simple.html* in the simpleApp/presentation/resource directory.

2 Add the following HTML to it:

```
<HTML>
  <HEAD>
    <TITLE>Simple Enhydra Page</TITLE>
  </HEAD>
  <BODY>
    <H1 ID="MyHeading">Ollie Says</H1>
    The current time is <SPAN ID="time">00:00:00</SPAN>.
  </BODY>
</HTML>
```

3 Change to the resource directory and enter this command:

```
xmlc -dump Simple.html
```

XMLC displays the following in the shell window (UNIX) or in command window (WINDOWS):

```
DOM hierarchy:
  LazyHTMLDocument%[T]:
      HTMLHtmlElementImpl: HTML
          HTMLHeadElementImpl: HEAD
              HTMLTitleElementImpl: TITLE
                  LazyText: Simple Enhydra Page
          HTMLBodyElementImpl: BODY
              HTMLHeadingElementImpl: H1: id="MyHeading"
                  LazyText: Ollie Says
              LazyText: The current time is
              LazyHTMLElement: SPAN: id="time"
                  LazyText: 00:00:00
              LazyText: .
```

Each line shows the DOM object name followed by a colon and then the corresponding HTML tag. If the tag has attributes, they are listed following the tag in name/value pairs. For instance, *HTMLHeadingElement* is the DOM name for the `<H1>` tag, and it has an ID attribute with the value "*MyHeading*".

The level of indenting shows the object relationships. So, for example, you can see that the first *HTMLHeadingElement* is the child of *HTMLBodyElement*.

-class and -keep options

By default, XMLC creates a class with the same name as the HTML file. So, for *Simple.html*, it would create *Simple.java*. To create a class with a different name, use the *-class* option to specify a name for the class that XMLC creates.

By default, XMLC deletes the java source file that it creates, leaving only the compiled class file. The source file is useful primarily for understanding how XMLC and the DOM API works. Use the *-keep* option to keep the java source file.

1 To create a java object named *SimpleHTML* for the HTML file *Simple.html* and to keep the java source file, enter this command:

```
xmlc -keep -class simpleapp.presentation.SimpleHTML Simple.html
```

XMLC generates two files: *SimpleHTML.java* and *SimpleHTML.class*.

2 Open *SimpleHTML.java* and look at the generated code.

Within the file, you will find two methods, *getElementMyHeading()* and *getElementTime()*. XMLC recognized the ID attributes of the heading and `<SPAN>` tags in *Simple.html* and generated these methods. Look through the file to get an idea of the object that XMLC creates for a very simple document.

3 Once you are done looking at *SimpleHTML.java* and *SimpleHTML.class*, delete them.

You are done exploring how XMLC works for now, but keep your *Simple.html* file because you are going to use it later in this tutorial.

Enhydra programming

This section covers more topics essential to Enhydra application development:

- Maintaining session state
- Adding a new page to the application
- Populating a table
- Adding a business object

Maintaining session state

Because HTTP is a stateless protocol, an application that needs to keep user-specific information across multiple requests must perform session maintenance. For an overview of how Enhydra performs session maintenance, see “Session Manager” in the chapter 3.

Think of the user’s session as a container in which the application can store any information associated with a particular user. The class that you use as the container is:

```
com.lutris.appserver.server.session.SessionData
```

It is similar to a hash table in that it has these methods:

- *set()* method to which you pass a string key and an object to store
- *get()* method which returns the object, given the string key

Enhydra matches a user to a particular SessionData object with a *session key*, a very (24 character) long randomly-generated string. When the Enhydra Session Manager first creates a SessionData object for a user, it generates a session key and stores it in its internal data structure. Enhydra also gives the session key to the client, either passed as a cookie or appended to the URL. The next time the client makes a request, the Session Manager uses the key to find that user’s SessionData object.

Generally, you don’t need to worry about the session key - Enhydra handles all those details for you “under the hood.” You do, however, need to keep track of keyword strings that you use to get and set each object you want to save to the session.

To help you understand session maintenance, you are going to enhance your application so that the Welcome page displays the number of times a particular user has accessed it, in addition to the total “hits” on the page. For fun, you’ll also display the session key on the page.

1 Add these four lines of HTML just before the closing the last `</CENTER>` tag in *Welcome.html*:

```
<P>Number of hits from you:
<SPAN ID="PersonalHitCount">no count</SPAN>
<P>Session identifier:
<SPAN ID="SessionID">no count</SPAN>
```

2 Now add this import statement to *WelcomePresentation.java*:

```
import com.lutris.util.*;
```

3 In *WelcomePresentation.java*, add this member property to the WelcomePresentation class (just after *hitCount*):

```
final String hits = "HITS";
```

The string “HITS” is the keyword that the application uses to save and recall the hit count information.

4 Add the following code to the *run()* method, placing the line beneath the line you added with the *setTextHitCount* method.

You can find this code in the *SessionMaint.java* file located in the <enhydra_root>/multiserver/webapps/enhydra-docs/getting_started/samples directory.

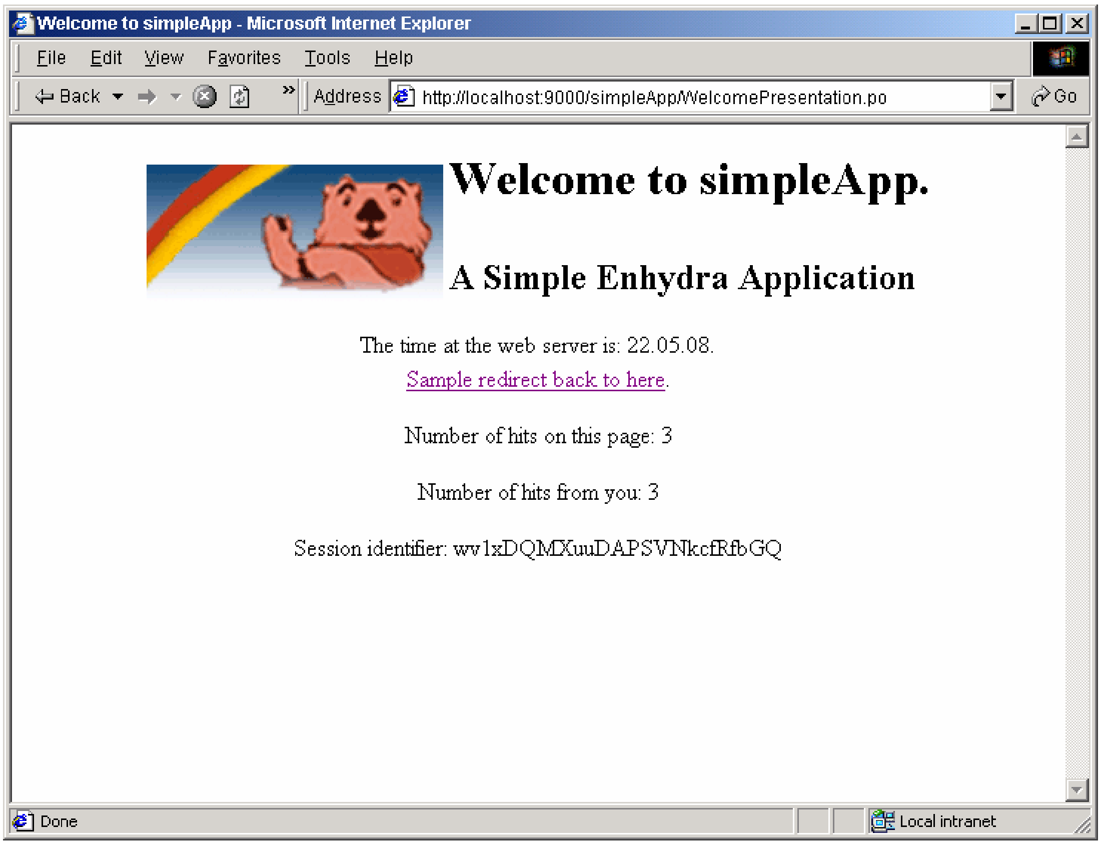
```
try {
 Integer personalHits =
               (Integer)comms.session.getSessionData().get(hits);
 if(personalHits == null) {
  personalHits = new Integer(1);
 }
 else {
  personalHits = new Integer(personalHits.intValue() + 1);
 }
 comms.session.getSessionData().set(hits, personalHits);
 // Save personalHits to the user’s session.

 welcome.setTextPersonalHitCount( personalHits.toString() );
 welcome.setTextSessionID( comms.session.getSessionKey() );
 // Shows the session key value used for session tracking.
}  catch (KeywordValueException e) {
 comms.response.writeHTML("Session access error" + e.getMessage());
}
```

This code begins by calling *getSessionData().get(hits)* to get the value stored for the keyword string “HITS.” Because SessionData stores only generic java.lang.Objects, you have to typecast it to Integer. If the object has not been previously stored in the session, the code creates a new Integer of value 1 (one). If the object exists, the value is incremented.

The code then saves the Integer object back into the session with *setSessionData().set(hits, personalHits)* and writes the value into the Web page with *getSessionKey()*. Normally, you would not need to deal with the session key, but for curiosity’s sake this example shows you how to display it.

5 Rebuild and start the application, and access the page with your browser. The Welcome page displays the total number of hits, as well as the number of hits from a particular client, as well as the client’s unique Session Identifier, as shown in Figure 4.8.


Welcome to simpleApp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address
http://localhost:9000/simpleApp/WelcomePresentation.po
Go
Done
Local intranet

Welcome to simpleApp.

A Simple Enhydra Application

The time at the web server is: 22.05.08.

Sample redirect back to here.

Number of hits on this page: 3

Number of hits from you: 3

Session identifier: wv1xDQMXuuDAPSVNkcfRfbGQ

Figure 4.8 Browser displaying the simpleApp Welcome page with a Session Identifier

Because you are running the application on your Localhost server, it is not accessible to any other clients, so these numbers will always be the same. However, if the application was run on a “real” server, you would see different numbers depending on how many times you had accessed the page versus the total number of hits. Notice also that the session ID string always stays the same.

Adding a new page to the application

Next, you are going to add a new page (HTML file and presentation object) to your application. You’re going to use the little HTML file you created previously, *Simple.html*. In addition to learning how to add a page, you’re also going to play around with the DOM a little bit to become more familiar with it.

To create a new presentation object:

1 Copy the file *WelcomePresentation.java* and call it *SimplePresentation.java*.

2 Open *SimplePresentation.java* and change the name of the class from *WelcomePresentation* to *SimplePresentation*.

3 Remove all the session-related code.

4 Change all the occurrences of *WelcomeHTML* to *SimpleHTML*.

5 Change the *welcome* identifier to *simple*.

Now you have “stripped down” presentation object. The *run()* method should look like this:

```
public void run(HttpPresentationComms comms)
  throws HttpPresentationException, IOException {

  SimpleHTML simple;
  String now;

  simple = (SimpleHTML)comms.xmlcFactory.create(SimpleHTML.class);
  try {
    Dater dater =
           DaterFactory.createDater("simpleapp.business.DaterImpl");
    now = dater.getDate();
  } catch (Exception ex){ . . . }

  simple.setTextTime(now);
  comms.response.writeDOM(simple);
}
```

6 Add these statements at the top of the file, after the other import statements:

```
import org.w3c.dom.*;
import org.w3c.dom.html.*;
```

7 Add the following lines just *before* the last statement in the *run()* method:

```
HTMLHeadingElement heading = simple.getElementMyHeading();
heading.setAttribute( "align", "center" );
Text heading_text = (Text) heading.getFirstChild();
heading_text.setData( "Mr. Ollie Otter says:" );
```

This code does the following:

- Gets the HTMLHeadingElement object named *MyHeading* from the DOM
- Sets its ALIGN attribute to *CENTER* to center the heading on the page
- Gets the child object of the heading (a Text object)
- Sets a new value for the text string, “Mr. Ollie Otter says:”

You could have done the same thing by putting a `<SPAN>` tag around the text in the heading. XMLC would then have generated a *setTextMethod()* that you could have called in the code. This example, however, illustrates how to do it with the DOM.

Note This code performs some low-level DOM manipulation that you should normally not do in your application because it violates the separation of presentation and business logic. It is only presented here to help explain the DOM.

The convention is to create fooHTML class for a file foo.html.

8 Save all files and run enhydra-ant in the applications root directory to build the package.

To create a link from the Welcome page to your new page:

1 Add the following HTML at the bottom of *Welcome.html*:

```
<A HREF=”SimplePresentation.po”>Go to Simple Page</A>
```

2 If you have not already done so, stop your servlet container by pressing Ctrl-C in the shell window (UNIX), or in command window (WINDOWS) or by starting stop-enhydra batch/shell script file, build the application from the top level and start Enhydra:

1. **UNIX**:

```
cd /multiserver/enhydra/myapps/simpleApp
./enhydra-ant
cd application
cd bin
./start-enhydra
```

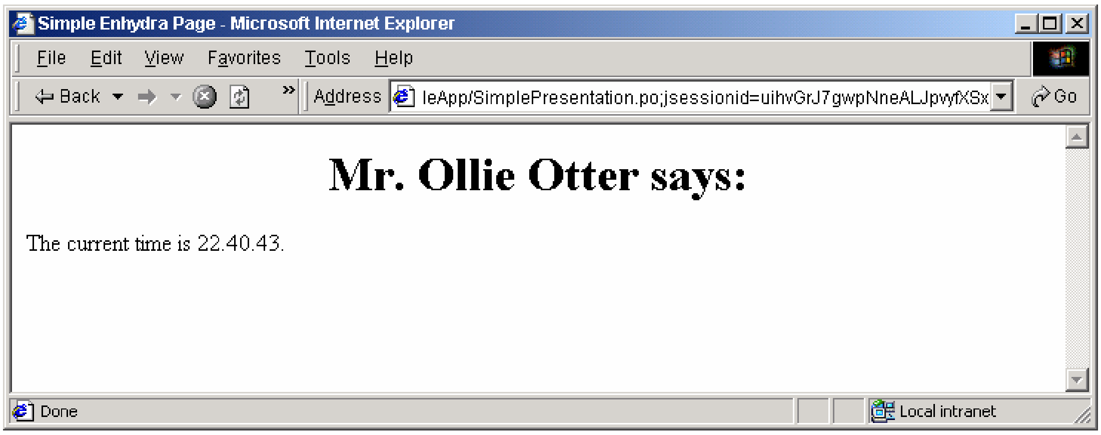
2. **WINDOWS**:

```
cd /multiserver/enhydra/myapps/simpleApp
enhydra-ant
cd application
cd bin
start-enhydra
```

3 Now access the application from your browser, as you did before.

4 Click the Go to Simple Page link to view the SimplePresentation PO.

The Simple presentation object has only a heading and the current time. You're going to make this page more interesting in the next section.



leApp/SimplePresentation.po;jsessionid=uihvGrJ7gwpNneALJpwyfXSx

Mr. Ollie Otter says:

The current time is 22.40.43.

Figure 4.9 Browser displaying the simpleApp Simple presentation object

Populating a table

Another common task in Web application development is populating an HTML table with dynamic data. This section discusses populating a table using a static String array as the data source. In a later section, you will modify the code to get data from a database.

Follow these steps to populate a table:

- Create the table in HTML
- Programmatically populate the table
- Rebuild and run the application

Create the table in HTML

In the *Simple.html* file, create an HTML table with a template row and an ID attribute.

1 Edit the file *Simple.html* in your simpleApp project.

2 Add the HTML shown below just before the end of the `<BODY>` tag.

Note If you don't want to type in all this HTML, you can copy and paste it from <enhydra_root>/multiserver/webapps/enhydra-docs/enhydra/getting_started/samples/TableCode.html file.

```
<H2 align=center>Disc List</H2>
<TABLE border=3>
  <TR>
    <TH>Artist</TH> <TH>Title</TH> <TH>Genre</TH>
    <TH>I Like This Disc</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>
    <TD><SPAN id=LikeThisDisc>Yes</SPAN></TD>
  </TR>
</TABLE>
```

This HTML contains a table with a single "template" row (in the second `<TR>` tag).

Notice that both this row and the `<SPAN>` tags enclosing the cell contents have `ID` attributes. This is called a template row, because it is used as a model from which you construct further rows of the table.

Programmatically populate the table

To programmatically populate the table, edit the presentation object corresponding to *Simple.html*. In the following steps, you will add code to *Simple.java* to iteratively replace the HTML table elements with text from an array of strings.

1 Copy the file: <enhydra_root>/multiserver/webapps/enhydra-docs/getting_started/samples/TableCode.java into your application's presentation directory and rename it **SimplePresentation.java**.

Note If you like, you can save your old SimplePresentation.java to SimplePresentation.sav for future reference.

2 Now, look at your new *SimplePresentation.java*.

In addition to the standard features of a presentation object, the first thing you'll notice in this code is a member property that is an array of strings representing the content the application will use to populate the table. This array takes the place of a database result set for this example:

```
String[][] discList =
{ { "Felonious Monk Fish", "Deep Sea Blues", "Jazz", "Yes" },
{ "Funky Urchin", "Lovely Spines", "Techno Pop", "Yes" },
{ "Stinky Pups", "Shark Attack", "Hardcore", "No" } };
```

The next new section of code gets the document objects for the table elements:

```
HTMLTableRowElement templateRow = simple.getElementTemplateRow();
```

```
HTMLElement artistCellTemplate = simple.getElementArtist();
HTMLElement titleCellTemplate = simple.getElementTitle();
HTMLElement genreCellTemplate = simple.getElementGenre();
HTMLElement likeThisDisc = simple.getElementLikeThisDisc();
```

The next section of code removes the `ID` attributes from these objects. The reason for this is that the DOM requires each `ID` in the document to be unique. When you make a copy of the table row, you would otherwise have duplicate IDs.

The *removeAttribute()* method removes the attribute with the specified name:

```
templateRow.removeAttribute("id");
artistCellTemplate.removeAttribute("id");
titleCellTemplate.removeAttribute("id");
genreCellTemplate.removeAttribute("id");
likeThisDisc.removeAttribute("id");
```

Then, a call to *getParentNode()* gets a reference to the table document object, which you'll be using later:

```
Node discTable = templateRow.getParentNode();
```

Next comes the heart of the code, a for loop that iterates through each "row" in the "result set", puts text in each cell in the table row, and then appends a copy (or *clone*) of the row to the table:

```
for (int numDiscs = 0; numDiscs < discList.length; numDiscs++) {
 simple.setTextArtist(discList[numDiscs][0]);
 simple.setTextTitle(discList[numDiscs][1]);
 simple.setTextGenre(discList[numDiscs][2]);
 simple.setTextLikeThisDisc(discList[numDiscs][3]);
 discTable.appendChild(templateRow.cloneNode(true));
}
```

That last statement is crucial: The *cloneNode()* method creates a copy of the Node object that calls it; in this case, templateRow. The boolean argument true determines if it copies only the node itself or the node and all its children, and their children, and so on. In this example, the argument is true because you want to copy the row and its child nodes (the table cells and the text inside them).

Finally, *removeChild()* removes the template row from the table. This ensures that the "dummy data" in the template does not show up in the runtime page.

```
discTable.removeChild(templateRow);
```

Rebuild and run the application

Now rebuild the application and load the page in your browser.

1 If you have not already done so, stop your servlet container, and build the application from the top level:

1. **UNIX**:

```
cd /enhydra/myapps/simpleApp
./enhydra-ant
cd application
cd bin
./start-enhydra
```

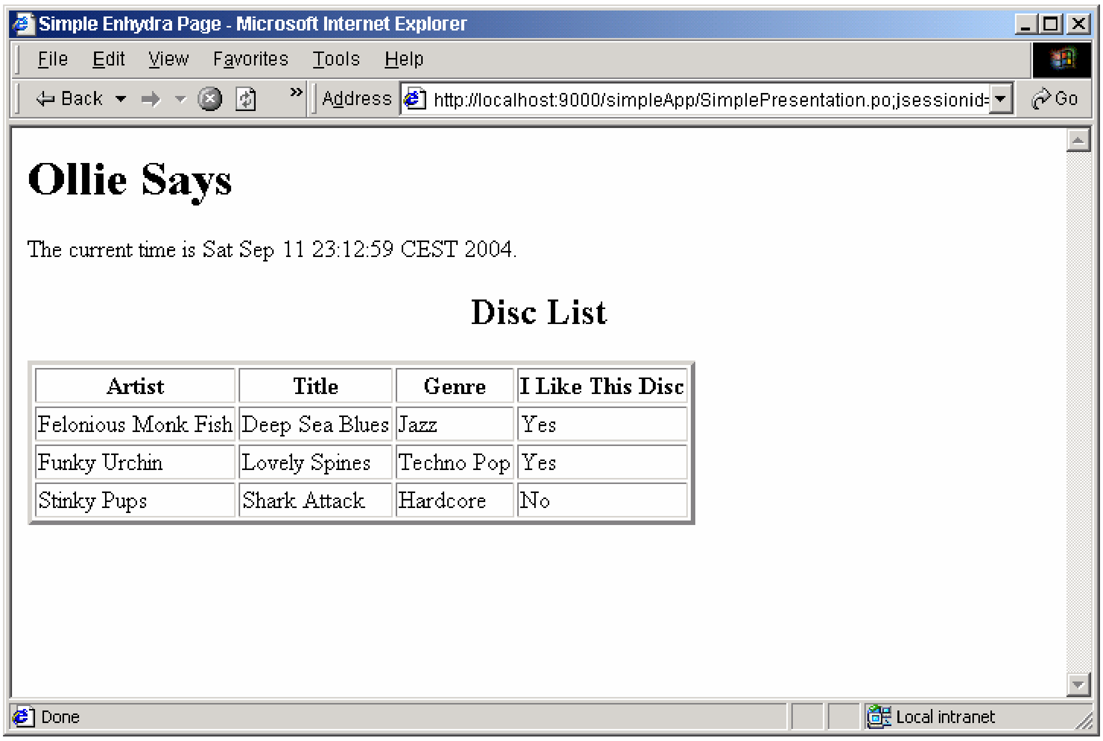
2. **WINDOWS**:

```
cd /enhydra/myapps/simpleApp
enhydra-ant
cd application
cd bin
start-enhydra
```

2 Now access the application from your browser, as you did before.

3 Click the Go to Simple Page link to view the new SimplePresentation PO.

The Simple presentation object now includes a Disc List table, as shown in Figure 4.10.


Simple Enhydra Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address
Go
Done
Local intranet

Address http://localhost:9000/simpleApp/SimplePresentation.po;jsessionid=

Ollie Says

The current time is Sat Sep 11 23:12:59 CEST 2004.

Disc List

Artist	Title	Genre	I Like This Disc
Felonious Monk Fish	Deep Sea Blues	Jazz	Yes
Funky Urchin	Lovely Spines	Techno Pop	Yes
Stinky Pups	Shark Attack	Hardcore	No

Figure 4.10 Simple PO with a programmatically populated Disc List table

Adding a business object

So far, your application has three objects: the *SimpleApp* application object, and two presentation objects, *Welcome* and *Simple*. Now, you are going to add a business object that you will use in the following sections. This will not change what the application displays.

The business object represents a list of discs. This is not terribly useful, but it does illustrate a basic role of business objects as you proceed.

To add a business object:

Since version 6.x presentation and business layer don't communicate directly, but though specification layer. The reason for this separation is providing possibility of separate development of presentation and business layer. Due to this new logic, to add a new business object, three classes should be created. Firstly, in specification layer should be created interface that business object will implement and that will be used in the presentation layer, and the factory class that will create the business object. Secondly, the implementation of the interface should be created in business layer. This implementation presents the business object. In the following text is detailly explained their creation.

1 Create new file called *SimpleListOfDiscs.java* in simpleApp/specification/src/simpleapp/spec directory. This file will be the mentioned interface. Since the interface is in specification layer, it will belong to package:

```
package simpleapp.spec;
```

This interface has only one method (it will be used in presentation layer):

```
public String[][] getDiscList();
```

2 The complete interface can be found in directory <enhydra_root>/multiserver/webapps/enhydra-docs/getting_started/samples and copied to simpleApp/specification/src/simpleapp/spec directory. It looks like:

```
package simpleapp.spec;

public interface SimpleListOfDiscs {

  public String[][] getDiscList();

}
```

3 Create new factory class called SimpleListOfDiscsFactory.java in directory simpleApp/specification/src/simpleapp/spec directory. This class also belongs to package:

```
package simpleapp.spec;
```

This class has one constructor and only one method which creates list od discs:

```
public static SimpleListOfDiscs createSimpleListOfDiscs(String
                    fullClassName)
```

This method has one argument – fullClassName. It contains the full class name of the implementation (business object) that will be created. The method returns this object as it's interface type. This is the way how the separation of presentation and business layers is done.

4 The complete class can be found in directory <enhydra_root>/multiserver/webapps/enhydra-docs/getting_started/samples and copied to simpleApp/specification/src/simpleapp/spec directory. It looks like:

```
package simpleapp.spec;

public class SimpleListOfDiscsFactory {

  private SimpleListOfDiscsFactory() { }

  public static SimpleListOfDiscs createSimpleListOfDiscs(String
                                                    fullClassName) {
    SimpleListOfDiscs result = null;
    Class objectClass = null;
    try {
      // Create the value object
      objectClass = Class.forName(fullClassName);
      result = (SimpleListOfDiscs) objectClass.newInstance();
    } catch (Exception ex) {
      System.out.println("Error on creating the object" + ex);
    }
    return result;
  }
}
```

5 Create new file called *SimpleListOfDiscsImpl.java* in the business directory, in simpleApp/business/src/simpleapp/business. It represents the business object that implements the interface from the specification layer. Because *SimpleListOfDiscsImpl.java* is in your application's business package, the first line in the file will be:

```
package simpleapp.business;
```

The class implements the interfase *SimpleListOfDiscs* that must be imported:

```
import simpleapp.spec.SimpleListOfDiscs;
```

Add the following lines (cut and paste the array initializer from the Simple class, but be sure to add the underscore (_) in front of the identifier *discList*):

```
                    String[][] discList =
{ { "Felonious Monk Fish", "Deep Sea Blues", "Jazz", "Yes" },
   { "Funky Urchin", "Lovely Spines", "Techno Pop", "Yes" },
     { "Stinky Pups", "Shark Attack", "Hardcore", "No" } };
```

The method has default constructor and only one method that returns the variable *_discList*:

```
public String[][] getDiscList() {
  return _discList;
}
```

6 The complete file can be found in directory <enhydra_root>/multiserver/webapps/enhydra-docs/getting_started/samples and copied to simpleApp/business/src/simpleapp/business directory. It looks like:

```
package simpleapp.business;
import simpleapp.spec.SimpleListOfDiscs;

  public class SimpleListOfDiscsImpl implements SimpleListOfDiscs{
  String[][] _discList =
    { { "Felonious Monk Fish", "Deep Sea Blues", "Jazz", "Yes" },
    { "Funky Urchin", "Lovely Spines", "Techno Pop", "Yes" },
    { "Stinky Pups", "Shark Attack", "Hardcore", "No" } };

  public SimpleListOfDiscsImpl() {}

  public String[][] getDiscList() {
    return _discList;
  }
}
```

7 When all these classes are created, go back in the presentation directory, and edit *SimplePresentation.java* as follows:

1 Import new classes (interface and factory class):

```
import simpleapp.spec.SimpleListOfDiscsFactory;
import simpleapp.spec.SimpleListOfDiscs;
```

2 Add the following lines in *run* method to create an instance of your new business object (instead of the part that was populating *discList* variable with the string), and call it *getDiscList()* method. These lines take the place of the static array initializer in the previous section.

```
String[][] discList;
  try {
    SimpleListOfDiscs sdl = SimpleListOfDiscsFactory.
                          createSimpleListOfDiscs(
                                "simpleapp.business.SimpleListOfDiscsImpl");
    discList = sdl.getDiscList();
  } catch (Exception ex){. . . }
```

8 The complete class can be found in directory <enhydra_root>/multiserver/webapps/enhydra-docs/getting_started/samples and copied to simpleApp/presentation/src/simpleapp/presentation directory.

10 Rebuild and test your application.

You won't see anything different, but you have extracted some functionality out of the presentation object into the new business object. This will come in handy in an upcoming section, when you replace the static array with a real database query. In that case, you won't have to change your presentation class because the business object provides a buffer between it and the data layer.

Using DODS

The Data Object Design Studio (DODS), is a tool which, for the given doml file, can generate SQL script files for creating tables (for each table separately and one cumulative file for creating all tables), one file for deleting all tables, and/or java code for data objects described in the given doml file. DODS also has possibility to compile generated java classes and to parse SQL files (to split cumulative SQL into more separated SQL files using SQLSplitter tool).

DODS is one part of Enhydra 6.x. If Enhydra 6.x is installed, so is DODS. In this case, DODS home directory <dods_home> is: <enhydra_home>/multiserver/enhydra/dods.

Since 5.1 version, DODS has become independent from Enhydra, which means that can be used without it. In this case, DODS home directory <dods_home> is the directory in which independent DODS is installed.

If Enhydra is installed, DODS is included within, so nothing more needs to be done.

Note: DODS have methods: getTableName(), getHandle(), getVersion(), getNewVersion(), getOId(), getVersionColumnName(), getOIdColumnName(), getOriginDatabase(), getPrimaryKeyName() and the same set methods. Due to a possible method name collisions, please don't call your columns of tables by names: **tableName**, **handle**, **version**, **newVersion**, **oId**, **versionColumnName**, **oIdColumnName**, **originDatabase**, **primaryKeyName**. These DODS methods are deprecated and will be removed in the future DODS version. New methods have syntax **get_xxx()** and **set_xxx(…)**, so, the columns names should not start with "_" sign.

Also, DODS has method **getConfigurationAdministration**() that is not deprecated, so, column name **configurationAdministration** also must not be used.

DODS source building

If Enhydra 6.x is installed, DODS is included within, so nothing more needs to be done.

To build independent DODS, it is necessary to do the following actions:

- Unix stile slashes (/) must always be used instead of Dos stile backslashes (\).

- Start Command Promt and go to <dods_source> directory.
- To configure DODS you can call configure batch file with following options:

```
Configure [-version version_number] [-release release_tag]
          [-jdkhome jdk_home_dir] [-debug on/off]
          [-optimize on/off] [-instdir installdir]
```

where:

- version sets version_number.
- release sets release_tag.
- jdkhome sets java jdk_home_dir. Defaults: Path to system registred (if eny) jdk.
- debug compiles source with debug information (on/off). Default: off .
- optimize sets whether the source should be compiled with optimization or not (on/off). Default: on.
- instdir the path to your installation directory (see 'Make Options' --> make install)

Configure without parameters sets configuration parameters to default values.

- To build DODS start Command Promt and go to <dods_source> directory.

DODS building is completely Ant based. You can give one of the following options to the make command:

- **make** - builds and configures DODS with javadoc and docbook documentation
- **make buildAll** - builds and configures DODS with javadoc and docbook documentation
- **make buildOptimize** - builds, optimizes and configures DODS with javadoc and docbook documentation
- **make buildNoDoc** - builds and configures DODS without documentation building
- **make install** - copies and configures DODS without source compiling
- **make distributions** - builds and configures DODS with javadoc and docbook documentation and creates distribution; nsis 2.0b should be included in DODS if doesn't exist (files makensis.exe and makensisw.exe in Dods/Install/Windows/install directory)
- **make optimizeDistributions** - builds and configures DODS with javadoc and docbook documentation and creates optimized distribution; nsis 2.0b should be included in DODS if doesn't exist (files makensis.exe and makensisw.exe in Dods/Install/Windows/install directory)
- **make clean** - removes the output folder (in order to start a new compilation from the scratch)
- **make help** - displays all options

where <dods_home> is directory in which DODS is built.

Note After DODS building, you MUST add <dods_home>/bin directory to the begining of the system PATH.

DOML file syntax

This section 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:

```
<doml>
    <author/>
    <projectname/>
    <database>
        <package>
            <package>
                <package>
                    <table>
                        <column>
                            <type/>
                            <initialValue/>
                        </column>
                        <column>
                            <type/>
                        </column>
                        <index>
                            <indexColumn/>
                        </index>
                    </table>
                </package>
                <package>
                    <table>
                        <column>
                            <type/>
                        </column>
                        <column>
                            <javadoc/>
                            <referenceObject/>
                            <type/>
                        </column>
                        <column>
                            <javadoc/>
                            <type/>
                        </column>
                    </table>
                </package>
            </package>
        </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 which can contain contentTag, its general syntax would look like:

```
<contextTag>
   <sampleTag attribute1 attribute2 ...>
      <contentTag/>
   </sampleTag>
</contextTag>
```

<author>

Author of doml project.

Content	None
Attributes	None
Context	<doml>

<column>

<column> describes a column in a database table.

Content	<javadoc> <referenceObject> <type> <initialValue>
Attributes	**id -** The name of the column in the database table. **usedForQuery -** Specifies whether the values of the column will be used for queries. The possible values for usedForQuery are: true and false. **isConstant-** Specifies whether the column contains a constant value. The possible values for isConstant are: true and false. **generateSecure -** Specifies whether secure methods (methods with check of Users access) should be generated for the column. The possible values for generateSecure are: true and false. The default value is false. **generateInsecure -** Specifies whether insecure methods (methods without check of Users access) should be generated for the column. The possible values for generateInsecure are: true and false. The default value is true.
Context	<table>

<database>

Contains the package hierarchy, and specifies the database.

Content <package>

Attributes **database -** The database attribute specifies the database vendor. The valid types are as follows:

1. Standard - generated SQL code will conform to standard JDBC SQL. This is the default value.
2. Oracle - DODS will generate SQL optimized for Oracle databases.
3. Informix - DODS will generate SQL optimized for Informix databases.
4. MySQL - DODS will generate SQL optimized for MySQL databases.
5. MSQL - DODS will generate SQL optimized for MSQL databases.
6. Sybase - DODS will generate SQL optimized for Sybase databases.
7. PostgreSQL - DODS will generate SQL optimized for PostgreSQL databases.
8. DB2 - DODS will generate SQL optimized for DB2 databases.
9. QED - DODS will generate SQL optimized for QED databases.
10. InstantDB - DODS will generate SQL optimized for InstantDB databases.
11. HypersonicSQL- DODS will generate SQL optimized for HypersonicSQL databases.
12. MckoiSQL - DODS will generate SQL optimized for MckoiSQL

templateset - Template set that will be used for java code generation. The possible values for templateset are: standard and <any user defined template>. Default value is "standard".

generateSecure - Specifies whether secure methods (methods with check of Users access) should be generated for the database. The possible values for generateSecure are: true and false. The default value is false.

generateInsecure - Specifies whether insecure methods (methods without check of Users access) should be generated for the database. The possible values for generateInsecure are: true and false. The default value is true.

dirtyDOs – This parameter is attribute of <database> and <table> tag in DOML file. Optionally, "dirty" methods (methods that can create DOs in memory without transactions) can be marked as "deprecated" or even not be generated at all. If set to "Compatible", "dirty" methods will be generated (as before), if set to "Deprecate", "dirty" methods will be generated as deprecated, and if set to "Omit", "dirty" methods will not be generated at all. If parameter is set on <table> tag they override default value and value on <database> tag.

massDeletes - When turned allow you to build data layer including classes xxxDelete. These class provide you QueryBuilder speed in massive update operations, while maintaining caches (both global and transactions) valid.

massUpdates When turned allow you to build data layer including classes xxxUpdate. These class provide you QueryBuilder speed in massive update operations, while maintaining caches (both global and transactions) valid..

Context <doml>

<doml>

Root element of DOML files. This tag contains a database hierarchy that contains all the packages.

Content <author>

<projectname>

<database>

Attributes None

Context None

<index>

<index> is used to specify index columns.

Content <indexColumn>

Attributes **id -** The name of the index constraint in the database table.

unique - Specifies whether the index constraint is unique. The possible values for unique are: true and false. The default value is false.

clustered - Specifies whether the index constraint is clustered. The possible values for clustered are: true and false. The default value is false.

Context <table>

<indexColumn>

<indexColumn> is used to specify each index column in the constraint index.

Content None

Attributes **id -** The name of the column in the database table.

Context <index>

<initialValue>

<initialValue>is used to specify a default initial value for the column.

Content	None
Attributes	None
Context	<column>

<javadoc>

The <javadoc> tag contains the text for Javadoc entries for the column.

Content	None
Attributes	None
Context	<column>

<package>

Each package can contain a sub-package or a table structure.

Content	<package> <table>
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>

<projectname>

Contains information about project.

Content	None
Attributes	None
Context	<doml>

<referenceObject>

If the column is a reference to another table, <referenceObject> specifies the table.

Content None

Attributes **constraint -** Specifies whether the specified table row must exist. The possible values for constraint are: true and false.

reference - Specifies the ID of the referenced table.

Context <column>

<table>

<table> describes a table in a database.

Content <column>

<index>

Attributes **id -** Similar to the id attribute in <package>, <table>'s id contains the value of the table name located in the package. For example, if I had a package myPackage, a subpackage 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.

isView - This attribute is not currently used by DODS.

generateSecure - Specifies whether secure methods (methods with check of Users access) should be generated for the table. The possible values for generateSecure are: true and false. The default value is false.

generateInsecure - Specifies whether insecure methods (methods without check of Users access) should be generated for the table. The possible values for generateInsecure are: true and false. The default value is true.

massDeletes - When turned allow you to build data layer including classes xxxDelete. These class provide you QueryBuilder speed in massive update operations, while maintaining caches (both global and transactions) valid.

massUpdates - When turned allow you to build data layer including classes xxxUpdate. These class provide you QueryBuilder speed in massive update operations, while maintaining caches (both global and transactions) valid.

multidb - Specifies whether code for multi database support should be generated. The possible values for multidb are: true and false. The default value is false.

dirtyDOs – This parameter is attribute of <database> and <table> tag in DOML file. Optionally, "dirty" methods (methods that can create DOs in memory without transactions) can be marked as "deprecated" or even not be generated at all. If set to "Compatible", "dirty" methods will be generated (as before), if set to "Deprecate", "dirty" methods will be generated as deprecated, and if set to

"Omit", "dirty" methods will not be generated at all. If parameter is set on <table> tag they override default value and value on <database> tag.

Context <package>

<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 **size -** Specifies the size of data types that are commonly measured in width, like VARCHAR. Size must be an integer.

canBeNull - Specifies whether the column can contain null values. The possible values for canBeNull are: true and 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 .

Context <column>

Sample DOML file

The following snippet shows content of a DOML file, *discRack.doml*, which creates tables containing data person and its discs. This file can be found in discRack example, in *<dods_home>/examples/discrack* directory.

```
<?xml version="1.0" encoding="UTF-8"?>
<doml>
  <database database="Standard">
    <package id="discRack">
      <package id="discRack.data">
        <package id="discRack.data.person">
          <table id="discRack.data.person.Person" dbTableName="person">
            <column id="login" usedForQuery="true">
              <type dbType="VARCHAR" javaType="String"/>
            </column>
            <column id="password" usedForQuery="true" generateSecure="true">
              <type dbType="VARCHAR" javaType="String"/>
            </column>
            <column id="firstname" usedForQuery="true">
              <type dbType="VARCHAR" javaType="String"/>
            </column>
            <column id="lastname" usedForQuery="true">
              <type dbType="VARCHAR" javaType="String"/>
            </column>
          </table>
        </package>
        <package id="discRack.data.disc">
          <table id="discRack.data.disc.Disc">
            <column id="title" usedForQuery="true">
              <type dbType="VARCHAR" javaType="String"/>
            </column>
            <column id="artist" usedForQuery="true">
              <type dbType="VARCHAR" javaType="String"/>
            </column>
            <column id="genre" usedForQuery="true">
              <type dbType="VARCHAR" javaType="String"/>
            </column>
            <column id="owner" usedForQuery="true">
              <javadoc>/**
 *Attribute describing a link to the owner of this disc.
 */</javadoc>
              <referenceObject constraint="true"
reference="discRack.data.person.Person"/>
              <type dbType="none" javaType="discRack.data.person.PersonDO"/>
            </column>
            <column id="isLiked" usedForQuery="true">
              <javadoc>/**
 * A flag indicating whether the user likes this disc
 */</javadoc>
              <type dbType="BIT" javaType="boolean"/>
            </column>
          </table>
        </package>
      </package>
    </package>
  </database>
</doml>
```

Sample of part of DOML file for using indexes

The following snippet shows part of a DOML file, Computers.doml (creates tables containing data about computers and their parts).

```
<table id="firm.computers.hardware.parts.motherboard"  dbTableName="Motherboard">
  <column id="manufacturrer" generateSecure="false"  generateInsecure="false">
    <type canBeNull="false" dbType="CHAR" javaType="String" size="40"/>
  </column>
  <column id="type" generateSecure="false">
    <type dbType="CHAR" javaType="String" size="40"/>
  </column>
  <column id="chipSet" generateSecure="false">
    <type dbType="CHAR" javaType="String" size="40"/>
  </column>
  <column id="compName" generateSecure="false">
    <referenceObject constraint="true" reference="firm.comp.hardware.Computers" />
    <type dbType="none" javaType="firm.computers.hardware.ComputersDO"/>
  </column>
  <column id="integratedGraphicAdapter">
    <type dbType="BIT" javaType="boolean"/>
  </column>
  <column id="integratedModem" >
    <type dbType="BIT" javaType="boolean"/>
  </column>
  <column id="integratedNetworkKard">
    <type dbType="BIT" javaType="boolean"/>
  </column>
  <column id="integratedMusicKard">
    <type dbType="BIT" javaType="boolean"/>
  </column>
<!-- Each computer has only one motherboard. (There are all common computers.) -->
  <index id="computerName" unique="true">
    <indexColumn id="compName"/>
  </index>
</table>
. . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . .
<table id="firm.computers.hardware.parts.monitor" dbTableName="Monitor">
  <column id="manufacturrer">
    <type canBeNull="false" dbType="CHAR" javaType="String" size="40"/>
  </column>
  <column id="type" >
    <type dbType="CHAR" javaType="String" size="40"/>
  </column>
  <column id="maxResolution">
    <type canBeNull="true" dbType="CHAR" javaType="String" size="20"/>
  </column>
  <column id="refreshFrequency" >
    <type canBeNull="true" dbType="INTEGER" javaType="int"/>
  </column>
  <column id="compName" >
    <referenceObject constraint="true" reference="firm.computers.hardware.Computers"/>
    <type dbType="none" javaType="firm.computers.hardware.ComputersDO"/>
  </column>
<!-- Each computer has only one monitor. (There are all common computers.) -->
  <index id="computerName" unique="true">
    <indexColumn id="compName"/>
  </index>
</table>
```

The whole DOML file *Computers.doml* in <dods_home>/examples/doml_examples directory.

Starting dods generator

There are two different ways to run dods generator. If you want to start generator quickly, you can start wizard by typing:

```
dods
```

without any parameter. Those files are located in :

- *<enhydra_home>multiserver/enhydra/bin* folder, for DODS in Enhydra.
- *<dods_home>/bin* folder, for independent DODS.

Note: *<enhydra_home>multiserver/enhydra/bin* (in the case DODS is used in Enhydra), or *<dods_home>/bin* folder (for independent DODS) should be added in the system path. Then, DODS can be started from any directory (by typing *dods*).

This will be described in the *Quick Compile* section.

If you want to start generator without wizard, you need to type (in the command line) `dods` with additional parameters. You can find details in *Custom Compile* section.

File location

After generating, locations of generated files are:

```
<OUTPUT_DIRECTORY>\SQLcreate.sql
<OUTPUT_DIRECTORY>\<PACKAGE_0>\..\<PACKAGE_N>\<TableName>DataStruct.java
<OUTPUT_DIRECTORY>\<PACKAGE_0>\..\<PACKAGE_N>\<TableName>DOI.java
<OUTPUT_DIRECTORY>\<PACKAGE_0>\..\<PACKAGE_N>\<TableName>DO.java
<OUTPUT_DIRECTORY>\<PACKAGE_0>\..\<PACKAGE_N>\<TableName>Query.java
<OUTPUT_DIRECTORY>\<PACKAGE_0>\..\<PACKAGE_N>\<TableName>.xml
```

where <OUTPUT_DIRECTORY> is base directory of your project, <PACKAGE_0>\..\<PACKAGE_N> is generated from last package id attribute of DOML file, and <TableName> is the name of the table from your database. For example, if part of your DOML file looks like this:

```
<package id="discRack">
  <package id="discRack.data">
    <table id=" discRack.data.Person"  dbTableName="Person">
...
    </table>
```

you will get file structure as follows:

```
<OUTPUT_DIRECTORY>\discRack\data\PersonDataStruct.java
<OUTPUT_DIRECTORY>\discRack\data\PersonDOI.java
<OUTPUT_DIRECTORY>\discRack\data\PersonDO.java
<OUTPUT_DIRECTORY>\discRack\data\PersonQuery.java
<OUTPUT_DIRECTORY>\discRack\data\Person.xml
```

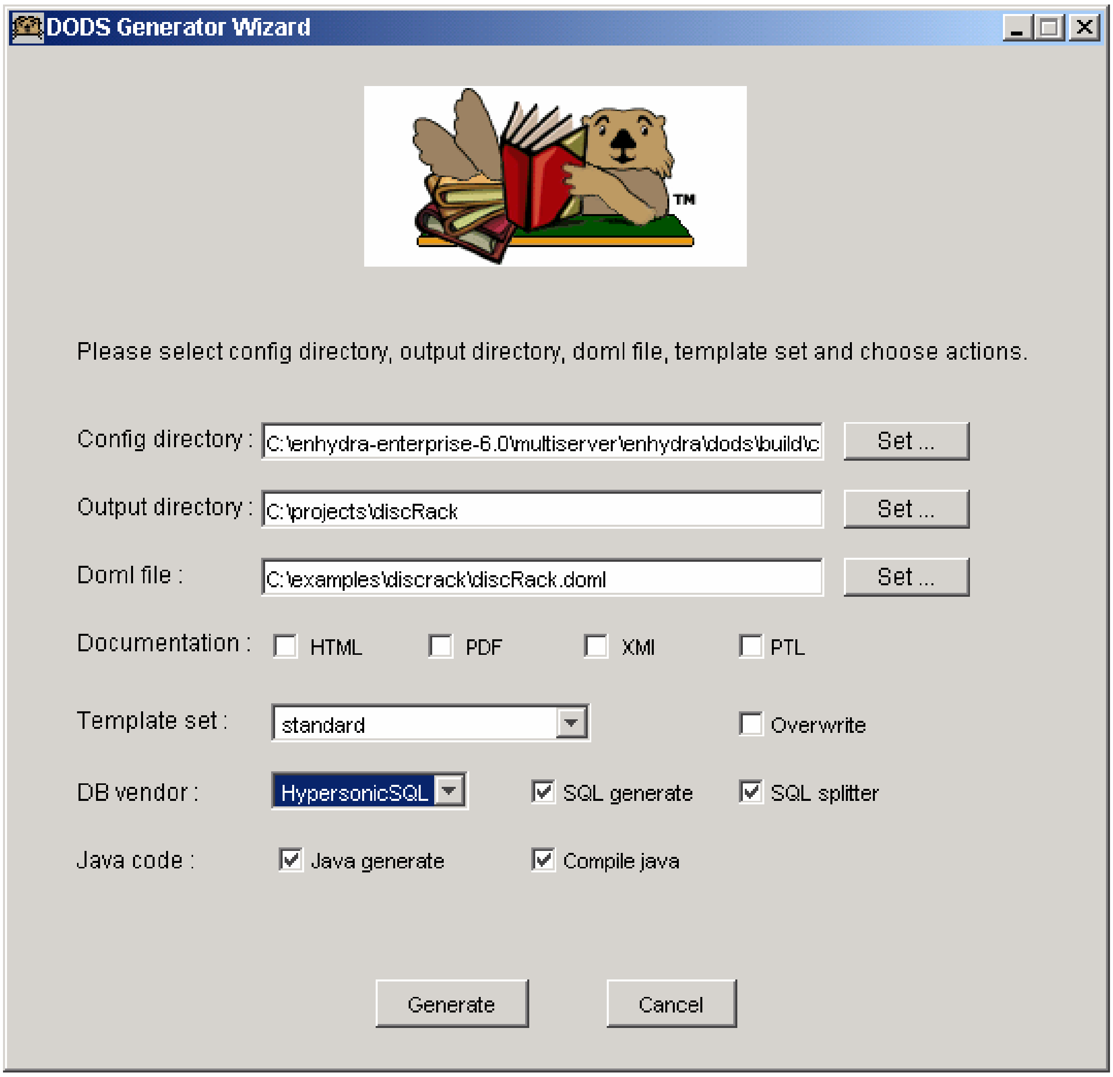
There are transient XML files that are generated from DOML file, before java code is generated. The java code, mentioned before, is actually generated from those transient xml files. If you want, you can change these xml files instead of DOML file and generate java code directly, without using the DOML file. You can find instructions for this in *Advanced Custom Compile* section.

If you change the DOML file, all java classes will be generated again, but, if you change transient xml files instead of the DOML file, only changed xml files are generated in java files. Other java files (whose xml files are not changed) are left as they are.

Quick Compile

DODS Generator Wizard is a graphical tool that helps you to easily generate Java and SQL files. It is recommended for the first time users.

When you start dods, you will get window like on Figure 4.11.



Please select config directory, output directory, doml file, template set and choose actions.

Config directory : C:\enhydra-enterprise-6.0\multiserver\enhydra\dods\build\c

Output directory : C:\projects\discRack

Doml file : C:\examples\discrack\discRack.doml

Documentation : ☐ HTML ☐ PDF ☐ XMI ☐ PTL

Template set : standard ☐ Overwrite

DB vendor : HypersonicSQL ☑ SQL generate ☑ SQL splitter

Java code : ☑ Java generate ☑ Compile java

Figure 4.11 DODS Generator Wizard

In the *Output directory* field you should input directory with full path of output directory that will be used.

DOML file field should be used for entering your DOML file.

Config directory field contains path to custom configuration folder (which contains *dodsConf.xml* file). It is used to generate java source code and SQL scripts. If the path is set to any other path than default (offered), in the application's configuration file shoul be set parameter :

```
DatabaseManager.ConfigurationDir
```

to new path of the custom configuration folder.

example:

```
DatabaseManager.ConfigurationDir=C:\configurations\dods
```

There are four options on the Generator Wizard:

- *Generate SQL:*

  This field should be checked if you want to generate: SQL files for each table separately, one cumulutave SQL file for creating all tables (SQLcreate.sql), and one file for deleting those tables (SQLdrop.sql).

  - *SQL Splitter*:

    It is used for creating separated cumulative SQL files (for creating tables, for adding foreign keys, primary keys and for deleting tables). This option enables creating tables without cross references, and after their creation, adding needed references.

    SQL Splitter copies all SQL commands from all SQL files which are situated in the working directory and all its subdirectories into SQL files.

    Original SQL files are created by DODS - Enhydra.

    All SQL commands are copied into file separateCreate.sql except sql commands which reference to foreign and primary key columns.

    In the *separateIntegrity.sql* file class puts ALTER TABLE sql commands with adding foreign key references.

    In the *separatePrimary.sql* file class puts ALTER TABLE sql commands with adding primary keys.

    In the *separateDropTable.sql* file class puts DROP TABLE sql commands for all tables which were created by create table SQL statements in the first file (separateCreate.sql).

    In the *separateIndex.sql* file class puts CREATE INDEX sql commands for all tables which were created by create table SQL statements in the first file(separateCreate.sql).

    In the *separateDropIntegrity.sql* file class puts DROP foreign key sql commands for all tables which were created by create table SQL statements in the first file (separateCreate.sql).

    In the *separateDropPrimary.sql* file class puts DROP primary sql commands for all tables which created by create table SQL statements in the first file (separateCreate.sql).

    All others Sql commands class puts into separate file.

  Unless Generate SQL field is checked, this field can not be checked. If this option is checked, Generator Wizard doesn't create cumulative SQL files.

- *Generate Java*:

  This field should be checked if you want to generate Java files (DO, Query, DOI and DataStruct objects).

  - *Compile Java*:

    It is used for compiling generated java files. Compiled files will be located in folder *<output_directory>/classes*. Unless Generate Java field is checked, this field can not be checked.

  If you do not need both Java and SQL generation, you can choose one of them instead of both. At least one of these options must be checked.

There are two combo boxes on the Generator Wizard. *Template set* combo box contains possible template sets:

- *standard*

  If this template set is chosen, DODS generates standard code.

- *<user_defined_templates>*:

  Users can define their own tempate sets.

Selected template set depends on <template_set> tag in doml file. If this tag is not set, defaut template set is "standard". If this tag is set, the value of this tag will be selected in template set combo box.

DB vendor combo box contains list of database vendors. If one of these vendors is selected, this database will overwrite database declared in doml file. Possible database vendors are:

- *MSQL*
- *Oracle*
- *Informix*
- *Sybase*
- *PostgreSQL*
- *MySQL*
- *Standard*
- *DB2*
- *QED*
- *HypersonicSQL*
- *InstantDB*
- *MckoiSQL*

There is a possibility on the Generator Wizard for generating following types of documentation:

- *HTML*: If you check this field, doml file will be converted into html file.
- *PDF*: If you check this field, doml file will be converted into pdf file.
- *XMI*: If you check this field, doml file will be converted into xmi file.
- *PTL*: If you check this field, doml file will be converted into ptl (Rational Rose) file.

On the Generator Wizard, there is also a check box:

- *Overwrite* - for code generating (java and sql), no matter if the code already existed.

Custom Compile

In case you want to generate Java and SQL code manually, type dods in the command line with desired parameters.

Command line:

```
        dods  [-?/help] [-a action] [-t templateset] [-b/-database] [-c confPath]
              [-f/force]    [-h/html] [-p/pdf] [-x/xmi] [-r/ptl] domlfile outputdir
```

where:

- **outputdir** is full path to output directory that will be used.
- **domlfile** is full path to .doml file for generating code.

options:

[**-? -help**] shows help.

[**-a action**] - ant task parameter for code generation:

- **dods:build_all** - to create all sql files and java classes (default).
- **dods:sql** - to create only sql files.
- **dods:java** - to create only java files and to compile them.
- **dods:javaNoCompile** - to create only java files and not to compile them.
- **dods:noCompile** - to create SQL files and java files and not to compile them.
- **dods:build_all_split** - to create all sql files and java classes and to compile it. SQL files will be divided into separate files using SQLSplitter .
- **dods:sqlsplit** - to create only sql files and separate in different files using SQLSplitter.
- **dods:noCompileSplit** - to create SQL files and separate sql commands using SQLSplitter and java files and not to compile them.
- **dods**:**generatorOff** - to disable generating and compiling of java source code, for generating documentation only (you stil need to set documentation propertys: html, pdf, ptl, xmi).

[**-t templateset**] - template set for generating java and sql code:

- **standard** - generate standard java code (default).
- **<user defined>** - any user defined template set.

[-b/-database] - sets database vendor for generating sql

[-c confPath] - sets folder with dodsConf.xml file

[**-f/-force**] - with this switch, code will be always generated, without it, only changes will be regenerated.

[**-h/-html**] - generates DODS html documentation from .doml file.

[**-p/-pdf**] - generates DODS pdf documentation from .doml file.

[**-x/-xmi**] - generates DODS xmi documentation from .doml file.

[**-r**/**-ptl**] - generates DODS ptl (Rational Rose) documentation from .doml file.

Advanced Custom Compile

In this section you can find information about advanced settings for generation of Java files.

One XML file is generated for every table from DOML file (situated in table folder with other java and sql files). That XML file is used as a base for generating four Java files.

DTD for that file can be found in *<dods_home>/dtd/temporaryXML.dtd* file. Some tags could be changed, i.e. `<AUTHOR>`.

Important: some tags should not be changed, or otherwise generated code will not be compailable.

Structure

The hierarchy of tags in a XML file is, as follows:

```
<TABLE>
    <PACKAGE/>
    <AUTHOR/>
    <PROJECT_NAME/>
    <TABLE_NAME/>
    <CLASS_NAME/>
    <DB_VENDOR/>
    <TEMPLATE_SET/>
    <COLUMN>
        <REFERENCE_OBJECT>
          <CONSTRAINT/>
          <IS_ABSTRACT/>
          <IS_FOREIGN_KEY/>
          <TABLE_NAME/>
          <PACKAGE/>
        </REFERENCE_OBJECT>
        <IS_CONSTANT/>
        <JAVADOC/>
        <DB_TYPE/>
        <JAVA_TYPE/>
        <JAVA_DEFAULT_VALUE/>
        <USED_FOR_QUERY/>
        <CAN_BE_NULL/>
        <IS_PRIMARY_KEY/>
        <SIZE/>
        <GENERATE_SECURE/>
        <GENERATE_INSECURE/>
    </COLUMN>
    <REFERRER>
        <REFATTR/>
    </REFERRER>
    <INDEX>
        <INDEX_COLUMN/>
    </INDEX>
    <DO_IS_OID_BASED/>
    <IS_ABSTRACT/>
    <DELETE_CASCADES/>
    <DO_IS_MULTIDB_BASED/>
    <IS_ANY_COLUMN_SECURE/>
    <GENERATE_DIRTY/>
    <GENERATE_SECURE/>
    <GENERATE_INSECURE/>
    <MASS_UPDATES/>
    <MASS_DELETES/>
</TABLE>
```

Tag reference

This section contains an alphabetical reference of all the XML tags that DODS can generate using given DOML file. Every tag 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.

<author>

Author of the Java code (your name).

Content	None
Attributes	None
Context	<table>

<can_be_null>

Can column be null. Possible values for can_be_null are: true and false.

Content	None
Attributes	None
Context	<column>

<class_name>

The name of the class which represents table in the database, mostly, it is the TABLE_NAME.

Content	None
Attributes	None
Context	<table>

<column>

Represents one column in the table.

Content	<reference_object> <is_constant> <javadoc> <db_type> <java_type> <java_default_value> <used_for_query> <can_be_null> <is_primary_key> <size> <generate_secure> <generate_insecure>
Attributes	**name -** Name of the column in the table.
Context	<table>

<constraint>

Specifies whether the specified table row must exist. Possible values for <constraint> are: true and false.

Content	None
Attributes	None
Context	<reference_object>

<db_type>

Data type from database that represents column.

Content	None
Attributes	None
Context	<column>

<db_vendor>

The database type. Possible values are Standard, InstantDB, Oracle, Informix, MySQL, Sybase, PostgreSQL, MSQL, DB2, QED or HypersonicSQL.

Content	None
Attributes	None
Context	<table>

<delete_cascades>

This is value retrieved from the configuration file and is used for sql code generation.

Content	None
Attributes	None
Context	<table>

<do_is_multidb_based>

Contains information about generation code for multi database support. Possible values: true and false.

Content	None
Attributes	None
Context	<database>

<do_is_oid_based>

Is table based on OID keys. Currently do_is_oid_based tag has only one value: true.

Content	None
Attributes	None
Context	<table>

<generate_dirty>

Specifies whether 'dirty' methods are to be generated (“Compatible”) - as before, deprecated (“Depricate”), or not generated at all (“Omit”).

Content	None

Attributes None

Context <table>

<generate_insecure>

True if insecure methods should be generated, otherwise false.

Content None

Attributes None

Context <table>

<column>

<generate_secure>

True if secure methods should be generated, otherwise false.

Content None

Attributes None

Context <table>

<column>

<index>

Represents table index.

Content <index_column>

Attributes **id -** Id of index.

unique - True if index is unique, otherwise false.

clustered - True if index is clustered, otherwise false.

Context <table>

<index_column>

Identifies index column.

Content None.

Attributes	**id -** Id of index column, same as name of column.
Context	<index>

<is_abstract>

Is generated class abstract. Currently is_abstract tag has only one value: false.

Content	None
Attributes	None
Context	<table> <reference_object>

<is_any_column_secure>

It is true if for any table column are generated secure methods, otherwise, is false.

Content	None
Attributes	None
Context	<table>

<is_constant>

Does column have constant value, that is, does it represent constant class attribute (not taken from database). Possible values for is_constant are: true and false.

Content	None
Attributes	None
Context	<column>

<is_foreign_key>

Is column used as a foreign key. Currently is_foreign_key tag has only one value: false.

Content	None
Attributes	None
Context	<reference_object>

<is_primary_key>

Is column a primary key. Currently is_primary_key tag has only one value: false.

Content	None
Attributes	None
Context	<column>

<javadoc>

Text for Javadoc documentation.

Content	None
Attributes	None
Context	<column>

<java_default_value>

Default value for Java data type.

Content	None
Attributes	None
Context	<column>

<java_type>

Data type from Java that represents column.

Content	None
Attributes	None
Context	<column>

<mass_deletes>

When turned allow you to build data layer including classes xxxDelete. These class provide you QueryBuilder speed in massive update operations, while maintaining caches (both global and transactions) valid.

Content	None

Attributes	None
Context	<table>

<mass_updates>

When turned allow you to build data layer including classes xxxUpdate. These class provide you QueryBuilder speed in massive update operations, while maintaining caches (both global and transactions) valid.

Content	None
Attributes	None
Context	<table>

<package>

Package that contains Java files.

Content	None
Attributes	None
Context	<table> <reference_object>

<project_name>

The project name.

Content	None
Attributes	None
Context	<table>

<refattr>

Tag that is used like attribute for tag <referrer>. It represents column of table that references generated class.

Content	None.
Attributes	**name -** Name of the column that references some DO objects. It is object of generated class, mostly.

	do_name - Name of the DO object that is referenced by attribute.
	generateSecure - True if secure methods should be generated, otherwise false.
Context	<referrer>

<reference_object>

If the column is a reference to another table, <reference_object> specifies the table.

Content	<constraint>
	<is_abstract>
	<is_foreign_key>
	<package>
	<table_name>
Attributes	**name -** Name of the reference object class.
Context	<column>

<referrer>

Outer table that references generated class.

Content	<refattr>
Attributes	**name -** Name of the outer table that references generated class.
	package - Name of the outer table package that references generated class.
	generateSecure - True if secure methods should be generated, otherwise false.
Context	<table>

<size>

Specifies the size of data types that are commonly measured in width, like VARCHAR. size must be an integer.

Content	None
Attributes	None
Context	<column>

<table>

Root element of XML files. It contains one table from database.

Content	<package>
	<author>
	<project_name>
	<table_name>
	<class_name>
	<db_vendor>
	<template_set>
	<do_is_oid_based>
	<is_any_column_secure>
	<is_abstract>
	<delete_cascades>
	<column>
	<referrer>
	<index>
	<generate_secure>
	<generate_insecure>
	<do_is_multidb_based>
	<generate_dirty>
	<mass_updates>
	<mass_deletes>
Attributes	None
Context	None

<table_name>

The name of the table in the database.

Content	None
Attributes	None

Context <table>

<reference_object>

<template_set>

Template set that will be used for java code generation. The possible values for template_set are: standard (default) and <any user defined template>.

Content None

Attributes None

Context <table>

<used_for_query>

Should column be used for queries. Possible values for used_for_query are: true and false.

Content None

Attributes None

Context <column>

Sample of part of transient XML file

The following snippet shows part of a transient file, **Disc.xml** (for table Disc from DiscRack example).

```
<TABLE>(i)
   <PACKAGE>discRack.data.disc</PACKAGE>(2)
   <AUTHOR>NN</AUTHOR>(2)
   <PROJECT_NAME>DiscRack</PROJECT_NAME>(2)
   <TABLE_NAME>Disc</TABLE_NAME>(2)
   <CLASS_NAME>Disc</CLASS_NAME>(2)
   <DB_VENDOR>Standard</DB_VENDOR>(2)
   <TEMPLATE_SET>standard</TEMPLATE_SET>(4)
   <GENERATE_SECURE>false</GENERATE_SECURE>(2)
   <GENERATE_INSECURE>true</GENERATE_INSECURE>(2)
   <DO_IS_OID_BASED>true</DO_IS_OID_BASED>(2)
   <IS_ABSTRACT>false</IS_ABSTRACT>(2)
   <DELETE_CASCADES>false</DELETE_CASCADES>(2)
   <DO_IS_MULTIDB_BASED>false</DO_IS_MULTIDB_BASED>(2)
   <IS_ANY_COLUMN_SECURE>false</IS_ANY_COLUMN_SECURE>(2)
   <GENERATE_DIRTY>Compatible</GENERATE_DIRTY>(2)
. . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
   <COLUMN name="artist">(2)
      <IS_CONSTANT>false</IS_CONSTANT>(2)
      <DB_TYPE>VARCHAR</DB_TYPE>(2)
      <JAVA_TYPE>String</JAVA_TYPE>(2)
      <USED_FOR_QUERY>true</USED_FOR_QUERY>(2)
      <CAN_BE_NULL>false</CAN_BE_NULL>(2)
      <IS_PRIMARY_KEY>false</IS_PRIMARY_KEY>(2)
      <SIZE>32</SIZE>(2)
      <GENERATE_SECURE>false</GENERATE_SECURE>(2)
      <GENERATE_INSECURE>true</GENERATE_INSECURE>(2)
   </COLUMN>(2)
. . . . . . . . . . . . . .
. . . . . . . . . . . . . .
   <COLUMN name="owner">(2)
      <REFERENCE_OBJECT name="Person">(3)
         <CONSTRAINT>true</CONSTRAINT>(3)
         <IS_ABSTRACT>false</IS_ABSTRACT>
         <IS_FOREIGN_KEY>false</IS_FOREIGN_KEY>(3)
         <PACKAGE>discRack.data.person</PACKAGE>(3)
         <TABLE_NAME>person</TABLE_NAME>(3)
      </REFERENCE_OBJECT>(3)
      <IS_CONSTANT>false</IS_CONSTANT>(2)
      <JAVADOC>/**
 *Attribute describing a link to the owner of this disc.
 */</JAVADOC>(4)
      <DB_TYPE>none</DB_TYPE>(2)
      <JAVA_TYPE>discRack.data.person.PersonDO</JAVA_TYPE>(2)
      <JAVA_DEFAULT_VALUE></JAVA_DEFAULT_VALUE>(4)
      <USED_FOR_QUERY>true</USED_FOR_QUERY>(2)
      <CAN_BE_NULL>false</CAN_BE_NULL>(2)
      <IS_PRIMARY_KEY>false</IS_PRIMARY_KEY>(2)
      <IS_ARRAY>false</IS_ARRAY>(2)
      <GENERATE_SECURE>false</GENERATE_SECURE>(2)
      <GENERATE_INSECURE>true</GENERATE_INSECURE>(2)
   </COLUMN>(2)
</TABLE>(1)
```

(1) red; (2) green; (3) violet; (4) blue

Different line colors are used to describe tag existence and possibility to change tag values.

`Red tags` must exist and must NOT be changed.

`Green tags` must exist and can be changed.

`Blue tags` are not required.

`Violet tags` can exist and must NOT be changed.

Structure of new modular DODS 6.x

DODS 6.x jars:

Dods runtime:

- *dbmanager-api.jar* - contains DatabaseManager interfaces and exceptions
- *dbmanager.jar* - DatabaseManager core - standard Database Manager implementation

Implementations:

- *stdconnection.jar* - standard implementations of ConnectionAllocator and DBConnection
- *stdtransaction.jar* - standard implementation of DBTransaction
- *stdcaches.jar* - standard cache implementations (DataStruct cache, Query caches)
- *dsconnection.jar* - Implementation of ConnectionAllocator and DBConnection which supports connection to the database using DataSource objects and which supports using DataSource connection pool - used in Enhydra 6.x.

Generator:

- *dods.jar*
- *ejen.jar*

How to use different implementations:

ConnectionAllocator

- Parameter: ConnectionAllocator. Default value: none (DODS will use StandardConnectionAllocator)
- File: configuration file
- Context: DatabaseManager, Database
- If this parameter is set to full class name of class that implements ExtendedConnectionAllocator interface, DODS will use this class to create Connection Allocator, if parameter is not set DODS will use default ExtendedConnectionAllocator implementation - StandardConnectionAllocator.

Connection

- Parameter ConnectionFactory - Default value: none (DODS will use StandardDBConnectionFactory)
- File: configuration file
- Context: Connection
- If this parameter is set to full class name of class that implements AbstractDBConnectionFactory interface DODS will use this class to create database connection factory, if parameter is not set DODS will use default AbstractDBConnectionFactory implementation - StandardDBConnectionFactory

Transaction

- Parameter : TransactionFactory. Default value: none (DODS will use StandardTransactionFactory)
- File: configuration file
- Context: DatabaseManager, Database
- If this parameter is set to full class name of class that implements AbstractDBTransactionFactory interface DODS will use this class to create database transaction factory, if parameter is not se DODS will use default AbstractDBTransactionFactory implementation - StandardDBTransactionFactory.

Cache implementations

- Parameter: QueryCacheImplClass. Default value: none (DODS will use QueryCacheImpl class)
- File: configuration file
- Context: DatabaseManager, Database
- If this parameter is set to full class name of class that extends DataStructCache abstract class DODS will use this class to create data struct cache for xxxDO class, if parameter is not set DODS will use class QueryCacheImpl as default.

DODS independence

Since version 5.1, DODS is independent from Enhydra. This means that it is possible for user to make any application (it doesn't need to be enhydra application) that can use DODS.

DODS works with DatabaseManagers. DatabaseManager is a class that provides facilities for work with databases.

There are two modes of using DODS:

- *non-threading*
  In non-threading mode, only one DatabaseManager is used for the whole application, no matter the application has one or more Threads.

- *threading*
  In threading mode, there is one DatabaseManager for every Thread. User needs, for every Thread, to define DatabaseManager. If, for any Thread, the DatabaseManager is not defined, the default DatabaseManager is used.

In the following text, the DODS independence is explaned for non-threading mode.

- in *main* application,
  add code that makes new DatabaseManager and registers it in DODS (by using class *DODS.java* from *org.enhydra.dods* package):

```
try {
. . .
String fileName = “discRack.conf”;
DODS.startup(fileName);
. . .
} catch (Exception e) {
      e.printStackTrace();
}
```

where "discRack.conf" is an example of application's configuration file. This file is the same as the Database Manager section of Enhydra application's configuration file.

This file can look like this:

```
#-----------------------------------------------------------------------------
# Database Manager Configuration
#-----------------------------------------------------------------------------
#
# The databases that are used by CSAM. Each of these databases
# has configuration parameters set under DatabaseManager.DB."databaseName".
#
DatabaseManager.Databases[] = "sid1"
#
# The default database used in this application.
#
DatabaseManager.DefaultDatabase = "sid1"
#
# Turn on/off debugging for transactions or queries. Valid values
# are "true" or "false".
#
DatabaseManager.Debug = "false"
#
# The type of database. Normally this is "Standard".
#
DatabaseManager.DB.sid1.ClassType = "Standard"
# DatabaseManager.DB.sid1.ClassType = "Oracle"
#
# The jdbc driver to use.
#
DatabaseManager.DB.sid1.JdbcDriver = "org.enhydra.instantdb.jdbc.idbDriver"
# DatabaseManager.DB.sid1.JdbcDriver = "oracle.jdbc.driver.OracleDriver"
# DatabaseManager.DB.sid1.JdbcDriver = "sun.jdbc.odbc.JdbcOdbcDriver"
#
# Database url.
#
DatabaseManager.DB.sid1.Connection.Url =
 "jdbc:idb:C:/DODS_5.1/output/examples/discrack/output/discRack.prp"
```

```
# DatabaseManager.DB.sid1.Connection.Url =
 "jdbc:oracle:thin:@MyHost:MyPort:MyDBName"
# DatabaseManager.DB.sid1.Connection.Url = "jdbc:odbc:discRack"
#
# Database user name. All connection are allocated by this user.
#
DatabaseManager.DB.sid1.Connection.User = "scott"
#DatabaseManager.DB.sid1.Connection.User = "Admin"
# Database user password.
#
DatabaseManager.DB.sid1.Connection.Password = "tiger"
#DatabaseManager.DB.sid1.Connection.Password = ""
#
# The maximum number of connections that a connection
# pool will hold. If set to zero, then connections
# are allocated indefinitly or until the database
# refuses to allocate any new connections.
#
DatabaseManager.DB.sid1.Connection.MaxPoolSize = 30
#
# Maximum amount of time that a thread will wait for
# a connection from the connection pool before an
# exception is thrown. This will prevent possible dead
# locks. The time out is in milliseconds. If the
# time out is <= zero, the allocation of connections
# will wait indefinitely.
#
DatabaseManager.DB.sid1.Connection.AllocationTimeout = 10000
#
# Used to log database (SQL) activity.
#
DatabaseManager.DB.sid1.Connection.Logging = false

#
# The number of object identifiers that are allocated
# as a group and held in memory. These identifiers
# are assigned to new data objects that are inserted
# into the database.
#
DatabaseManager.DB.sid1.ObjectId.CacheSize = 20
DatabaseManager.DB.sid1.ObjectId.MinValue = 1000000
#
# User wildcards
#
DatabaseManager.DB.User.userWildcard = "*"
DatabaseManager.DB.User.userSingleWildcard = "_"
DatabaseManager.DB.User.userSingleWildcardEscape = "$"
DatabaseManager.DB.User.userWildcardEscape = "$"
#
# Default table configuration
#
# DatabaseManager.defaults.AllReadOnly = false
DatabaseManager.defaults.lazyLoading = false
DatabaseManager.defaults.maxExecuteTime = 200
DatabaseManager.defaults.TransactionCaches = true
DatabaseManager.defaults.TransactionCheck = true
DatabaseManager.defaults.AutoSave = true
#
# Default database configuration
#
DatabaseManager.DB.sid1.TransactionCheck = false
DatabaseManager.DB.sid1.AutoSave = false
#
# Default cache configuration
#
DatabaseManager.defaults.cache.maxCacheSize = 500
DatabaseManager.defaults.cache.maxSimpleCacheSize = 100
DatabaseManager.defaults.cache.maxComplexCacheSize = 10
DatabaseManager.defaults.cache.reserveFactor = 0.1
```

Table and cache configuration is explained in sections “Table configuration” and “Cache configuration”, later in this chapter.

The example of non-enhydra application that uses DODS is DiscRack application, explained in next section.

Examples of non-enhydra applications

Examples of DODS non-enhydra applications are included in DODS installation and they are in DODS, in directory:

<DODS_HOME>/examples

Process of running non-enhydra application will be presented in this section on the example Disc Rack. This example application is in *<DODS_HOME>/examples/discrack* directory.

To run an example, these steps have to be done in Command Promt:

- first, go to wanted example (directory)

```
cd <DODS_HOME>/examples /discrack
```

- second, run ant,by typing:

```
ant
```

ant will build this application in its <output_directory>

- then, go to application’s output directory:

```
cd <output_directory>
```

- then, example will be run with:

```
run
```

The ant, which is used here, must be DODS’s **ant.bat**, which means that path *<DODS_HOME>/bin* must be included at the beginning of the system path.

DODS Ant task

Invokes DODS to generate a set of java classes from a doml file.The files will only be regenerated/compiled if the date of the doml file is newer than at least one of the generated files. This task extends Ant's ask.

Typically made visible to an Ant build file (*build.xml*) with the following declaration:

```
<taskdef name="dods" classname="org.enhydra.ant.taskdefs.Dods"/>
```

Parameters:

- **domlfile**- the doml input file describing data object mapping. Required = Yes.
- **outputDir** - target for generated classes, expressed as a directory path. Required = Yes.
- **force** - forces DODS always to regenerate source files. Possible values: ("true", "false"(default)). Required = No.
- **action** - name of Ant task from generate.xml. Required = No.

- **templateDir** - name of folder for template set for generating java code, expressed as a directory path. Required = No.
- **templateSet** - template set for generating java code. Required = No.
- **confDir** - path to custom configuration folder (If the path is set to any other path than default (offered), in the application's configuration file shoul be set parameter: `DatabaseManager.ConfigurationDir` to new path of the custom configuration folder). Required = No.
- **database** - sets database vendor for generating sql. Required = No.
- **html** - indicates DODS to generate html documentation from .doml file. Possible values: ("true", "false"(default)). Required = No.
- **pdf** - indicates DODS to generate pdf documentation from .doml file. Possible values: ("true", "false"(default)). Required = No.
- **xmi** - indicates DODS to generate xmi documentation from .doml file. Possible values: ("true", "false"(default)). Required = No.
- **ptl** - indicates DODS to generate ptl (Rational Rose) documentation from .doml file. Possible values: ("true", "false"(default)). Required = No.
- **without parameters** - to create all sql files and java classes and to compile it.

action parameters:

- **dods:build_all** - to create all sql files and java classes.
- **dods:sql** - to create only sql files.
- **dods:java** -to create only java files and to compile them.
- **dods:javaNoCompile** -to create only java files and not to compile them.
- **dods:noCompile** -to create SQL files and java files and not to compile them.
- **dods:build_all_split** - to create all sql files and java classes and to compile it. SQL files will be divided into separate files using SQLSplitter
- **dods:sqlsplit** - to create only sql files and separate in different files using SQLSplitter.
- **dods:noCompileSplit** - to create SQL files and separate sql commands using SQLSplitter and java files and not to compile them.
- **dods:generatorOff** - to disable generating and compiling of java source code, for generating documentation only (you stil need to set documentation propertys: html, pdf, ptl, xmi).

templateset parameters:

- **standard** - generate standard java code.
- **<user_defined>** - any user defined template set.

Example:

```
<dods domlfile ="${basedir}/discRack.doml" outputDir="${basedir}/src"
templateSet="standard"/>
```

Table configuration

Table configuration is explained on DiscRack example (directory <dods_output>/examples/discrack). The table parameters are defined on three levels.

The first level is DatabaseManager level. On this level can be defined the following parameters (all information are optional):

```
DatabaseManager.defaults.lazyLoading = true
DatabaseManager.defaults.maxExecuteTime = 200
DatabaseManager.defaults.AllReadOnly = false
```

The second level is database level. On this level can be defined the following parameters (all information are optional):

```
DatabaseManager.DB.<database_name>.lazyLoading = false
DatabaseManager.DB.<database_name>.maxExecuteTime = 350
DatabaseManager.DB.<database_name>.AllReadOnly = false
```

The third level is table level. In the case of DiscRack example, there are two tables: Disc and person. The tables can have the following parameters:

```
#
# Table Disc - table configuration
# DatabaseManager.DB.DiscRack.Disc.readOnly = false
DatabaseManager.DB.DiscRack.Disc.lazyLoading = false
DatabaseManager.DB.DiscRack.Disc.maxExecuteTime = 150
#
# Table Person - table configuration
# DatabaseManager.DB.DiscRack.person.readOnly = true
DatabaseManager.DB.DiscRack.person.lazyLoading = false
# DatabaseManager.DB.DiscRack.person.maxExecuteTime = 150
```

Table defaults on DatabaseManager and Database are default values for all application's tables. If, any of these parameters is defined on the Database level, that value is used as a default for all tables. If any of the parameters is not defined on the Database level, then, if it is defined on the DatabaseManager level, this value is used. If any of these parameters is not defined neither on the Database, nor on DatabaseManager level, DODS uses its own program defaults. For **lazyLoading**, program default is false, for **maxExecuteTime** 0 and for **readOnly** and **AllReadOnly** false.

If any of parameters **lazyLoading** or **maxExecuteTime** is defined on the table level, that value is used. If not, the default value for all tables is used (explained in previous paragraph).

Table parameter **readOnly** is true if the table is read-only, otherwise is false. If read-only is true, the operations: insert, update or delete on the tables are not possible.

If parameter **AllReadOnly** is defined and set to true (it can be defined on DatabaseManager or Database level), all applications will be read-only. In that case, table parameter **readOnly** is ignored. Only, If **AllReadOnly** is set to true and **readOnly** attribute of the table is set to false, warning is written to log during table initialization. In runtime exception is thrown on attempt of writing to that table.

Parameter **lazyLoading** is true if table supports lazy-loading, otherwise is false.

Parameter **maxExecuteTime** is time for query execution. Every query that is executed longer than maxExecuteTime is printed (SQL statement, execution time and maxExecutionTime) in application's log file.

Caching

Caching affects the behaviour of the DO class. If checked, all DO instances (their original DataStruct objects) are stored in the cache inside the DO class. Subsequent queries of the table use the Query class for queries. The results of all Queries, are complete.

Cache transformation

Since DODS 5.1 final, the **DO cache** is transformed into **DataStruct cache**. Instead of whole DOs, only their original DataStructs are added to new DataStruct cache.

DO have had only one data (DataStruct object) and all transformations were done on this object. DataStruct object contains values of columns of one table row. Now, DO holds 2 DataStruct-references:

- **originalData**

- **data**

The **originalData** holds original data (that was read from the database). This is never modified till commit, and this DataStruct object is added to DataStruct cache, if this cache exists.

The second, **data**, is only created (by copying the first one and setting version=version+1) if data is modified. If the second DataStruct exists, the DO's attribute **isDirty** is set to true. Even if after some modifications the new DataStruct holds exactly the same value as the original one, the DO is still dirty. So there is no way back from isDirty=true to isDirty=false (except during commit of the transaction). If the transaction is committed, the new datastruct is moved in place for the original datastruct. The new datastruct is NULL again, so the attribute **isDirty** becomes false again.

A newly created DO (in memory, not from the database) will just have a **data** DataStruct object. Data values in DataStruct object **originalData** is null before the **commit**().

The **oid** and the **version** attributes are moved from DO to DataStruct object.

New attributes added in DataStruct object are:

- **isEmpty**
  type: **boolean**
  default value : **true**
  Since **originalData** is being constructed for every DO, this flag "knows" if DataStruct has any useful content. If there is no data in DataStructs – except **oid** and **version**, this attribute is **true**, otherwise **false**.

- **databaseName**
  type: **String**
  default value : **null**
  The logical database to which this DataStruct belongs to.

New methods added in DataStruct object are:

- **getOId**()
  Returns DataStruct's **identifier**.

- **setDatabase**(String dbName)
  Sets attribute **databaseName**.

- **getDatabase**()
  Returns attribute **databaseName**.

- **getHandle**()
  Returns this DataStruct's **handle** (identifier as a string).

- **getCacheHandle**()
  Returns this DataStruct's **cache handle** (String in the form: "<database_name>.<indentifier_as_String>").

- **get** and **set** methods for every table column

In DO class are added new methods that work with **originalData**:

- **originalData_get<column_name>**([User usr])
  Returns the row value of the column <column_name> of the DO's **originalData** object.

- **originalData_set**(Object data)
  Sets the DO's **originalData** object.

- **getData**()
  Returns DO's DataStruct object. If DO’s **data** object exist, returns that object, otherwise returns DO’s **originalData** object.

- **originalData_get**()
  Returns DO's **originalData** object.

- **getOriginalVersion**()
  Returns the current version of DO’s originalData object.

Introduction

DODS provides the possibility for every table to have its cache.

The possible cache types are:

1. **None**
   No caching is available.

2. **LRU**
   The size of the cache is limited by the maximal number of objects that can be stored in it. When the cache is full, the objects in it are being replaced by new objects according to LRU (least recently used) algorithm. This algorithm says that the object which had been used the least recently (in the scale of time, the object to which had been accessed the longest time ago, which is on the end of LRU list) is removed from list and new one is put in front of the LRU list. If maximal number of objects is set to 0, it means that caching is not available (None type) at the moment, and if this number is set to negative number, it means that the cache is unbounded (it has no number limit).

3. **Full** (special case of LRU caching)
   This is a LRU cache which is unbounded. The entire table is queried and cached when the application starts. This is appropriate for tables of "static" data which are accessed frequently and which will not change during the execution of the application.

DODS has two levels of caching:

1. **Data Caching level**
   There is only one LRU cache: cache with DataStruct objects. The keys of this cache are cache handles – Strings in the following form:

   "<DataStruct_database_name>.<String_presentation_of_DataStruct_oid>"

   , and cache values are, as mentioned before DataStruct objects.

2. **Query caching level**
   Beside DataStruct object cache, there is a possibility of using three query caches (simple, complex and multi-join). Multi-join cache is included since DODS 6.x. All query caches are also LRU caches. The keys of these caches are Strings in the following form:

   "<query_database_name>.<String_presentation_of_query>"

and cache values are Query objects. Query objects are objects of the org.enhydra.dods.cache.**QueryCacheItem** class.

The **QueryCacheItem** object stores one query and its necessary data:

- Database of the query
- List of IDs of DataStruct object that are results of the query. This list can contain all results, or just some of them.
- Number of cached query results
- Information whether all results are in result list or not
- Information whether the query results are modified (if there have been performed inserts, updates or deletes, the results are modified)
- Time needed for query execution
- Array of conditions declared in WHERE part of the query (array of org.enhydra.dods.cache.**Condition** objects). This is needed only for simple queries.

Queries that were created with the query's and QueryBuilder's methods that support joins between tables are stored in multi-join cache. Queries that are supported by DataStruct cache are simple queries. Simple query is query that is not multi-join query and for which cache mechanisms can determine whether DataStruct object is query result or not (and query). Other queries (that are not multi-join queries) are complex queries.

The default values for maximal DataStruct cache size, simple, complex and multi-join query cache are 0 (no caching).

The default values for maximal cache size for DataStruct, simple and complex query cache are 0 (no caching).

Cache configuration

Cache configuration is explained on DiscRack example (directory <dods_output>/examples/discrack). The cache parameters are defined on three levels.
The first level is DatabaseManager level. On this level can be defined the following parameters (all information are optional):

```
# DatabaseManager.defaults.cache.maxCacheSize = 100
DatabaseManager.defaults.cache.maxSimpleCacheSize = 20
DatabaseManager.defaults.cache.maxComplexCacheSize = 5
DatabaseManager.defaults.cache.maxMultiJoinCacheSize = 3
DatabaseManager.defaults.cache.reserveFactor = 0.1
DatabaseManager.defaults.cache.CachePercentage = -1
# DatabaseManager.defaults.cache.initAllCaches = true
```

The second level is database level. On this level can be defined the following parameters (all information are optional):

```
DatabaseManager.DB.<database_name>.cache.maxCacheSize = 1100
# DatabaseManager.DB.<database_name>.cache.maxSimpleCacheSize = 10
# DatabaseManager.DB.<database_name>.cache.maxComplexCacheSize = 5
# DatabaseManager.DB.<database_name>.cache.maxMultiJoinCacheSize = 3
DatabaseManager.DB.<database_name>.cache.reserveFactor = 0.1
DatabaseManager.DB.<database_name>.cache.CachePercentage = -1
DatabaseManager.DB.<database_name>.cache.initAllCaches = true
```

The third level is table level. In the case of DiscRack example, there are two tables: Disc and person. The tables can have the following parameters:

```
#
# Table Disc - cache configuration
# DatabaseManager.DB.DiscRack.Disc.cache.maxCacheSize = 10000
DatabaseManager.DB.DiscRack.Disc.cache.maxSimpleCacheSize = 2000
DatabaseManager.DB.DiscRack.Disc.cache.maxComplexCacheSize = 250
DatabaseManager.DB.DiscRack.Disc.cache.maxMultiJoinCacheSize = 100
DatabaseManager.DB.DiscRack.Disc.cache.reserveFactor = 0.1
DatabaseManager.DB.DiscRack.Disc.cache.CachePercentage = 0.5
#
# Table Person - cache configuration
DatabaseManager.DB.DiscRack.person.cache.maxCacheSize = -1
DatabaseManager.DB.DiscRack.person.cache.maxSimpleCacheSize = 2000
DatabaseManager.DB.DiscRack.person.cache.maxComplexCacheSize = 250
DatabaseManager.DB.DiscRack.person.cache.maxMultiJoinCacheSize = 75
DatabaseManager.DB.DiscRack.person.cache.initialCondition = *
```

Cache defaults on DatabaseManager and Database are default values for all application's table caches. If, any of these parameters is defined on the Database level, that value is used as a default for all tables. If any of the parameters is not defined on the Database level, then, if it is defined on the DatabaseManager level, this value is used. If any of these parameters is not defined neither on the Database, nor on DatabaseManager level, DODS uses its own program defaults. For *maxCacheSize*, *maxSimpleCacheSize*, *maxComplexCacheSize*, *maxMultiJoinCacheSize* and *reserveFactor* program default value is 0, for *CachePercentage* is -1.0, and for *initAllCaches* is false.

If any of table level parameters *maxCacheSize*, *maxSimpleCacheSize*, *maxComplexCacheSize*, *maxMultiJoinCacheSize*, *reserveFactor* and *CachePercentage* is defined on the table level, that value is used. If not, the default value for all tables is used (explained in previous paragraph).

Table parameter *initialCondition* can be defined only on the table level. It contains "where" part of select clause. With this select clause is DataStruct cache of specified table initialized. If initialCondition = '*', the entire table will be added to the DataStruct cache in DataStruct cache initialization. If the parameter is NULL or not defined, no objects are added to the Data cache during the cache initialization.

It, for any table parameter *initialCondition* is not defined and the *initAllCaches* parameter is set to 'true' (on DatabaseManager or Database level, as explained before), the default value of *initialCondition* parameter for the table is "*".

Parameter *maxCacheSize* contains information about maximal size of DataStruct cache. Parameter *maxSimpleCacheSize* contains information about maximal size of simple query cache. Parameter *maxComplexCacheSize* contains information about maximal size of complex query cache. Parameter *maxMultiJoinCacheSize* contains information about maximal size of multi-join query cache.

Parameter *CachePercentage* is used for query to make decision what type of query will be executed: select t.* or select t.oid. If no lazy loading and caching is turned on and value of CachePercentage is less then currently used cache (in percents), t1.* is used for query statement. Otherwise select t.oid. Parameter value 0 means use always t1.oid if cache is turned on, -1 (default) means never if not lazyloading but cached. If lazy loading is on always is used t1.oid query.

In <table_name>Query.java class are added new methods:

- *setLoadData*(boolean newValue)

If parameter newValue set to true, query select t.* will be executed no matter what are the values of parameters lazyLoading and CachePercentage.

- *getLoadData*()

  Returns true if query select t.* will be executed, otherwise false.

Reserve factor is constant used in query caching. It is percent of how many more object are taken for evaluation. If num is number of needed results, then it is used

`num + reserveFactor * num`

objects for estimating what is quicker: go to database for all object that are not in the cache, or run again query on database. This value is given in percents, as number between 0 and 1 (0.25 means 25%).

For example, if reserveFactor is 0.5, and wanted number of results is 50, the estimation will be done on 75 (50 + 0.5 * 50) objects.

In the following text are explained maximal cache sizes (for DataStruct cache and query caches). The parameters *maxCacheSize*, *maxSimpleCacheSize*, *maxComplexCacheSize* and *maxMultiJoinCacheSize* of application's configuration file define these sizes.

- maxCacheSize > 0

  This cache is limited. The maximal number of elements in the cache is maxCacheSize.

- maxCacheSize = 0

  This means that there is no cache available. This value excludes cache from use.

- maxCacheSize < 0

  This cache is unlimited.

In the previous mentioned DiscRack example for cache configuration, DataStruct cache for table person has type full, because maxCacheSize is negative and initialCondition is "*". This combination of parameters values forms special case of LRU: full cache.

DODS has class org.enhydra.dods.cache.*UpdateConfigurationAdministration*. This class has public synchronized methods that provide possibility of run-time setting some cache and table parameters. This class is used by Enhydra application CacheAdmin. It is not recommended to be used by user applications.

Select statement

For query by oid (query by oid is query which "where" clause contains request for DO with specified oid), first is checked in the DataStruct cache if there is DataStruct object with desired oid. If DataStruct object is not find in the cache, hitting the database is performed, and the retrieved DataStruct object is added to the DataStruct cache. Queries by oid are not added in the query cache (they are trivial).

For full caching also, for query by oid, first is checked in the DataStruct cache if there is DataStruct object with desired oid. If DataStruct object is not find in the cache, hitting the database is not performed (all rows from the table are in the cache, so there is no result of this query).

For non-oid queries, for full caching, if the query is simple query, the query's result can be retrieved from the DataStruct cache, so there is no need to retrieve results from the database. In any other case of full caching, everything is done the same as for any other query (this is explained in the next paragraph).

For all other queries, it is checked if the query is already in the Query cache (simple, complex or multi-join). Query object has one attribute called "orderRelevant" which is true if query results must not be modified (no DO can be inserted, updated or deleted from cached query results). With the method *isOrderRelevant*() is checked whether the results of select can be modified or not.

If query is in the cache and the *isOrderRelevant*() returns false, result oids are retrieved from QueryCache. If query is in the cache and the *isOrderRelevant*() returns true, and the result oids are not modified, the result oids are also retrieved from query cache. But, if query is in the cache and the *isOrderRelevant*() returns true, but the result oids are modified, the result oids from the QueryCache are not used. Instead of that, hitting the database is performed.

If the result is found in the query cache, for every result oid, it is checked whether there is that object is in the DataStruct cache. Then, when is counted number of results that are not in the DataStruct cache, the time needed for performing queries by oid on database for all oids from the result that are not in the cache is compared against the time needed for performing the whole query.

If the time needed for performing queries by oid on database is less or equal to query execution time, results are retrieved from the cache, and those that are not there, from database (using queries by oid).

If the time is longer, or the query is not in the query cache, or the query supports joins with other tables, or cached query results are modified but for this query is order relevant, the query is performed on the database.

If the results are retrieved from the database, the query and its necessary data are put in the Query cache (simple, complex or multi-join).

If there was already that query in the query cache, but the query was executed again (because there were not enough result oids in the result list, or because the old query was modified, and for the new query isOrderRelavant is true), the old query is replaced by the new one (this query is not modified).

Insert statement

Data object is inserted in the database and first time the data is moved to original DataStruct, it is added to the DataStruct cache, after successful commit.

All complex and multi-join queries of the table that are for the database of inserted DO, are removed from the query caches.

For every simple query of the table (with the inserted DO's database) from query cache it is checked whether inserted DO is query result or not.

If new DO is query result, in the query cache is this query marked as "*modified*".

If its cached results are complete (all are in the query cache), oid of this inserted DO is added to query cached result list. If cached results are not complete oid is not added to the list.

Update statement

Data object is updated in the database and first time the data is moved to original DataStruct, it is added to the cache if commit was successful (the old DataStruct object is removed from the DataStruct cache if it was there).

All complex and multi-join queries of the table that are for the database of inserted DO are removed from the query caches.

For every simple query of the table (with the inserted DO's database) from query cache it is checked whether updated DO is the query result or not.

If yes, this query is marked as "*modified*" in the query cache, and the DO is included in query results only if it wasn't in the cache and the cached result list is complete.

If no, if DO's oid exists in the query results, it is removed from there and because of this change of the results, this query is marked as "modified" in the query cache.

Delete statement

Deletes DO from the database and removes its original DataStruct object originalData from the DataStruct cache (if it is there).

Goes through the query cache (simple, complex and multi-join) and wherever finds this DO, removes it from the query results and marks that query as "*modified*".

Cache Initialization

For every cache, it is possible to define initial query statement which contains "where" clause which is used during DataStruct cache initialization. When cache is created, query with this "where" condition is performed on the database, and the results are put in the DataStruct cache.

Before the query is executed, parameter maxDBrows is set to maxCacheSize using method

```
setMaxRows(int max)
```

of <table_name>Query.java class.

Using this method, maximum maxCacheSize DOs will be retrieved from the database and their original DataStruct objects (originalData) will be added to DataStruct cache.

If a table is fully cached, simple queries are done in the memory (even the first time this is done in the cache and not in the Database)

If initial query statement is set to "*", all rows of the table from the database (up to maxCacheSize) will be put in the DataStruct cache.

If initial query statement is set to null, no rows from the table in database will be put in the DataStruct cache (cache would be empty).

Table and cache statistics

DODS has the possibility of providing table and cache statistics.
The public method:

```
get_statistics()
```

of the <table_name>DO.java class returns the statictics object (statistics object must implement org.enhydra.dods.statistics.Statistics interface). This object provides the following methods for the table statistics and one method for retrieving cache statistics:

- *getStatisticsType*()

  Returns type of the statistics. It returns 0 if statistics is for table that has no caching, 1 if statistics is for table with only Data caching, and 2 if statistics is for table with Query caching.

- *getInsertNum*()

Returns number of insert statements performed on the table.

- *setInsertNum*(int newInsertNum)

  Sets number of insert statements performed on the table to value *newInsertNum* value.

- *incrementInsertNum*()

  Increases number of insert statements performed on the table for one.

- *getUpdateNum*()

  Returns number of update statements performed on the table.

- *setUpdateNum*(int newUpdateNum)

  Sets number of update statements performed on the table to value *newUpdateNum* value.

- *incrementUpdateNum*()

  Increases number of update statements performed on the table for one.

- *getDeleteNum*()

  Returns number of delete statements performed on the table.

- *setDeleteNum*(int newDeleteNum)

  Sets number of delete statements performed on the table to value *newDeleteNum*.

- *incrementDeleteNum*()

  Increases number of delete statements performed on table for one.

- *getDMLNum*()

  Returns number of DML operations (inserts, updates and deletes) performed on the table.

- *getLazyLoadingNum*()

  Returns number of lazy loadings performed on the table.

- *setLazyLoadingNum*(int newLazyLoadingNum)

  Sets number of lazy loadings performed on the table to value *newLazyLoadingNum*.

- *incrementLazyLoadingNum*()

  Increases number of lazy loadings performed on the table for one.

- *getStartTime*()

  Returns time when the statistics was started.

- *setStartTime*(Date startTime)

  Sets time when the statistics starts to value *startTime*.

- *getStopTime*()

  Returns time when the statistics was stopped.

- *setStopTime*(Date stopTime)

  Sets time when the statistics stops to value *stopTime*.

- *stopTime*()

  Sets stop time to current time.

- *getQueryNum*()

  Returns total number of non-oid queries performed on the table. Query by oid is query which "where" clause contains request for DO with specified oid. Non-oid query is any other query.

- *setQueryNum*(int newQueryNum)

  Sets total number of non-oid queries performed on the table to value *newQueryNum*.

- *incrementQueryNum*()

  Increases total number of non-oid queries performed on the table for one.

- *getQueryByOIdNum*()

Returns total number of queries by oid performed on the table.

- *setQueryByOIdNum*(int newQueryByOIdNum)

  Sets total number of queries by oid performed on the table to value *newQueryByOIdNum*.

- *incrementQueryByOIdNum*()

  Increases total number of queries by oid performed on the table for one.

- *getQueryAverageTime*()

  Returns average time needed for executing non-oid query.

- *updateQueryAverageTime*(int newTime)

  Updates average time needed for executing non-oid queries to value *newTime*.

- *getQueryByOIdAverageTime*()

  Returns average time needed for executing query by oid.

- *updateQueryByOIdAverageTime*(int newTime, int no)

  Updates average time for executing OId queries with time *newTime* and increments number of them by paramether *no*.

- *clear*()

  Clears DO, simple query and complex query statistics.

- *getCacheStatistics*(int type)

  Returns cache statistics (objects must implement interface org.enhydra.dods.statistics.CacheStatistics) for :

    - o DataStruct cache when parameter type equals 0
    - o simple query cache when parameter type equals 1
    - o complex query cache when parameter type equals 2
    - o multi-join query cache when parameter type equals 3

Cache statistics objects have the following methods:

- *getCacheAccessNum*()

  Returns total number of times the cache was accessed.

- *setCacheAccessNum*(int num)

  Sets total number of times the cache was accessed to value *num*.

- *incrementCacheAccessNum*(int num)

  Increases total number of times the cache was accessed for value *num*.

- *getCacheHitsNum*()

  Returns number of cache accesses that were successful.

- *setCacheHitsNum*(int cacheHitsNum)

  Sets number of of cache accesses that were successful to value *cacheHitsNum*.

- *incrementCacheHitsNum*(int num)

  Increases of cache accesses that were successful for value *num*.

- *getUsedPercents*()

  Returns how much cache is currently used. This value is given in percents. If cache is unbounded, method returns 100%.

- *getCacheHitsPercents*()

  Returns how many cache accesses were successful. This value is given in percents.

- *clearStatistics*()

  Clears statistics.

User wildcards

Like cache size, user wildcards are also defined in application's configuration file.

Example:
For file *discRack.conf* part of code for user wildcards can look like:

```
#
# User wildcards
#
DatabaseManager.DB.User.userWildcard = "*"
DatabaseManager.DB.User.userSingleWildcard = "_"
DatabaseManager.DB.User.userSingleWildcardEscape = "§"
DatabaseManager.DB.User.userWildcardEscape = "§"
```

Loading the schema

The picture 4.12 shows the schema of the Disc Rack example (the example is explained detailly in the chapter 5).

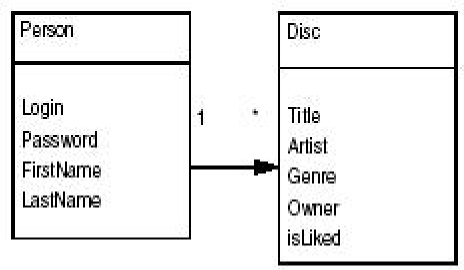


Figure 4.12 DiscRack object-model/schema

There are two tables:

- Person, which has columns: Login, Password, FirstName and LastName
- Disc, which has columns: Title, Artist, Genre, Owner and isLiked

The complete schema of the DiscRack database generated by DODS is pictured in Figure 4.13. DODS shows the features common to both the database schema and the object model. For example, the *disc* data object has *title* and *artist* fields, that are the properties (members) of the Java class, as well as the columns of the corresponding database table.

DODS creates Java code for object operations and SQL code for database operations (for example, the one-to-many relationship between person and discs).

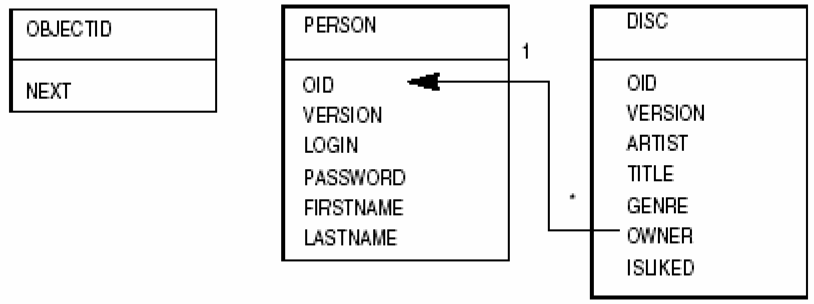


Figure 4.13 DiscRack database schema generated by DODS

There are some differences from the original database schema:

- DISC and PERSON tables have two additional fields, OID and VERSION
- There is a third table, OBJECTID, that contains one column, NEXT, with a single row

The OID column is the primary key for each table created by DODS. The application code generated by DODS ensures that every row has a value of OID that is unique within the database. Whenever a new row is added to a table, the application generates a unique object ID to put in the OID column. It uses the OBJECTID table to keep track of the next object ID to be assigned.

DODS application code uses the VERSION column in each table to ensure that the data (that application is updating) is accurate. Because many users can be accessing the database simultaneously, a record can change between the time the application retrieves it and when the application attempts to change the record.

Every time an application updates a row, it increments the VERSION column in the database. The application qualifies updates on both the VERSION and OID columns - if it finds that there are no rows that have the expected values, then it knows that another process has changed the row it is trying to update, and it throws an exception. You can catch the exception in your application code to handle such situations appropriately.

Generated structure

DODS generates the following files and subdirectories in the *data* directory:

- *disc* and *person* directories, which contain the Java code for the disc and person data objects, respectively, and an SQL file defining the corresponding database tables
- *create_tables.sql* and *drop_tables.sql* files, which contain standard SQL statements to create and remove the disc and person tables from a database, respectively
- file *build.xml*
- *classes* directory, which is initially empty

Each data object directory contains java source files to create four classes. For example, the person data object directory contains personDO, personDOI, personQuery, and personDataStruct. The data object and the query classes are the most commonly used classes.

DODS also generates build.xml file for the data layer. This lets you compile the data layer independently or along with the entire project. The empty *classes* directory is used only if you compile the data layer separately.

Database Independency

DODS generates java code that is database independent. This means that java code is the same no matter which database is used.

When you want to change the database, the only thing you need to do is to change <App_name>.conf file (update it with information considering new database). This change is necessary for connection to the database.

Using multi databases in DODS

DODS has the possibility of working with more than one database at the same time. This means that, when the application is started, it doesn't have to be stopped in order to change the database the application uses.

The table supports multi databases if the attribute *multidb* of `<table>` tag in doml file is set to true for that table.

To take advantage of simultaneous use of multiple table DODS requires separate doml file for every distinct database.

Example:

```
<doml>
   <database database="Standard">
   .........
      <package id="multibase.data.employee">
         <table id="multibase.data.employee.Employee" multidb="true">
            <column id="firstName" usedForQuery="true">
               <type dbType="VARCHAR" javaType="String"/>
            </column>
            ........
         </table>
         .........
      </package>
      .........
</doml>

<doml>
   <database database="Standard">
   ............
      <package id="multibase.data.employee.programer">
         <table               id="multibase.data.employee.programer.Programer"
      multidb="true">
            <column id="firstName" usedForQuery="true">
               <type dbType="VARCHAR" javaType="String"/>
            </column>
            ............
         </table>
         ...........
      </package>
      ...........
   </database>
</doml>
```

For that kind of table, in <App_name>.conf file must be defined all logical databases the application will use on this table. For each of these databases must be configured all needed parameters.

Example:

```
#---------------------------------------------------------------------------
#                    Database Manager Configuration
#---------------------------------------------------------------------------

DatabaseManager.Databases[] = "programer", "employee"
DatabaseManager.DefaultDatabase = "employee"

DatabaseManager.DB.programer.Connection.User = ""
DatabaseManager.DB.employee.Connection.User = ""

DatabaseManager.DB.programer.Connection.Password = ""
DatabaseManager.DB.employee.Connection.Password = ""

..............................................
```

```
DatabaseManager.DB.programer.Connection.Logging = false
DatabaseManager.DB.employee.Connection.Logging = false

DatabaseManager.DB.programer.ObjectId.CacheSize = 20
DatabaseManager.DB.programer.ObjectId.MinValue = 1000000

DatabaseManager.DB.employee.ObjectId.CacheSize = 20
DatabaseManager.DB.employee.ObjectId.MinValue = 1000000
```

If the table doesn't support multi databases, the default database will be used for this table.

When the <App_name>.conf file (with information about all databases) is updated, and the application is started, it uses the default database. The definition of the new (desired) database is being done in the stage of creation of DO and Query objects.

When a Query object is created for a database (given or default), the results of this Query are only DOs from that database, not from any other database.

If caching is used, there is only one cache for all <table_name>DO's original data originalData (these DataStruct objects can belong to different databases, but are all placed in the same DataStruct cache).

DODS takes care of referential integrities within the database which means that DODS searches referenced object in the same database in which the object that referenced it is. If you want to use referenced objects from any other database, you must yourself take care of referential integrities.

In the <table_name>DO class public constructors and methods (query, createVirgin, createCopy, createExisting) are now defined and with the database parameter.

Here are some examples of using these constructors and methods.

Query example:

```
ProgramerDO[] programers;
ProgramerQuery pQuery = new ProgramerQuery("programer");
programers = pQuery.getDOArray();
Create example:
EmployeeDO newE=EmployeeDO.createVirgin("employee");
newE.setFirstName(employees[i][0]);
..................................
newE.save();
```

Example of transferring data from one database to another

```
ProgramerDO[] programers;
ProgramerQuery pQuery = new ProgramerQuery("programer");
programers = pQuery.getDOArray();
for(int i=0; i< programers.length; i++) {
EmployeeDO newEmployee=EmployeeDO.createVirgin("employee");
newEmployee.setFirstName(programers[i].getFirstName());
newEmployee.setLastName(programers[i].getLastName());
newEmployee.setOccupation("programer");
newEmployee.setDepartment("IT");
newEmployee.save();
}
```

You can use them with this parameter in which case the object will be created for the given logical database, or you can use these constructors and methods without database parameter. In this case, they will be created for default database. If the methods with database parameter are used, and the parameter is set to null, the default database is used.

Mass modifications

DODS's duality (a problem)

DODS gives you the option to choose how you want to modify rows of table in database. Obvious one is to use instances of generated DO classes:

```
SomeDOClass sgDO = SomeDOClass.createExisting(oid);
sgDO.delete();
```

Other option is QueryBuilder, which may be used to build not only select queries, but update or delete statements, as in:

```
QueryBuilder qb = getQueryBuilderForClass("SomeDOClass");
qb.setDeleteQuery();
qb.addWhereClause("oid", Integer.parseInt(enumoid), QueryBuilder.EQUAL);
qb.executeUpdate();
```

There could be a situation where you may want to touch many rows at once.

First approach must be encompassed by loop which would iterate value of oid, thus producing many separate SQL statements. This isn't efficient at all, and it gets slower as number of rows raises.

Second approach, using QueryBuilder produces one SQL statement, and executes much faster. But: Cache implementation in DODS includes caching DataStructs and queries for table globally, and caching DO objects in transaction. Both caches are implemented in generated classes only, so using QueryBuilder won't touch caches.

Warning: Direct use of QueryBuilder is NOT recommended, since it doesn't affect any of the caches, and your application may work erroneously.

Generated classes (a solution)

New options in .doml file are massUpdates and massDeletes. They're implemented as boolean attributes of doml and table tags. Default values are false.

When turned on new options allow you to build data layer including two classes xxxUpdate and xxxDelete.

These classes provide you QueryBuilder speed in massive update operations, while maintaining caches (both global and transactions) valid.

Classes xxxUpdate and xxxDelete have constructor that takes xxxQuery as parameter. This instance of query builds WHERE clause of a statement, while setCOLUMN methods provide contents of SET part.

```
xxxQuery query = new xxxQuery(dbt);
query.setQueryCOLUMN_NAME(value);
xxxUpdate update = new xxxUpdate(query);
update.setANOTHER_COLUMN(another_column_value);
update.save();
dbt.commit();
```

In order to keep caches valid, global query caches (for affected table) are cleared, because we cannot compute their consistency without all DataStructs. DataStructs in global cache and DO objects in transaction caches are removed (for delete) or emptied (they will be loaded as with lazy load feature, for update).

New .conf parameter SelectOids introduced at DatabaseManager, LogicalDatabase, and table level, specifies whether there would be additional select statement executed to collect oids that would be affected by mass modification.

Default value for SelectOids is false, and usual override method is applied too (table level overrides database which in turn overrides manager's value). If parameter is true, before actually executing massive modifications select statement will be run to collect list of OId's. This is then used to update cache for listed DataStructs/DOs only. Otherwise (SelectOids is false), all instances of xxxDataStruct/xxxDO will be updated.

Both xxxUpdate and xxxDelete have method setSelectOIds(boolean) for developer to prevent configuration parameter SelectOids effects. If certain mass modification will affect many rows, developer may choose to prevent collecting oids, so even if administrator sets parameter to true, application doesn't lose on its speed.

```
xxxQuery query = new xxxQuery(dbt);
query.setQueryCOLUMN_NAME(value);
xxxDelete delete = new xxxDelete(query);
delete.setSelectOIds(false);
delete.save();
dbt.commit();
```

Conversion of doml file

DODS has the possibility of converting doml file, release 5.*. As described in “Quick Compile”, Generator Wizard has the possibility of converting doml file into html, pdf, xmi and ptl document types. The name of the target (html, pdf, xmi, ptl) files will be the same as the name of doml file that is being converted, and they would be located in output directory.

The doml file can also be converted manually. For this purpose are used files that are in <dods _home>/bin folder, and they are:

- **doml2html** - converts doml 5.* file into html file.
  doml2html is used with the following parameters:
  doml2html [-help] [doml5*-file] [html-file]
  where:
  help - prints message for usage and exits.
  doml5-file* - doml file relese 5.*.
  html-file **-** desired target html file.

- **doml2pdf** - converts doml 5.* file into pdf file.
  doml2pdf is used with the following parameters:
  doml2pdf [-help] [doml5*-file] [pdf-file]
  where:
  help - prints message for usage and exits.
  doml5-file* - doml file relese 5.*.
  pdf-file **-** desired target pdf file.

- **doml2xmi** - converts doml 5.* file into xmi file.
  doml2xmi is used with the following parameters:
  doml2xmi [-help] [doml5*-file] [xmi-file]
  where:
  help - prints message for usage and exits.
  doml5-file* - doml file relese 5.*.
  xmi-file **-** desired target xmi file.

- **doml2ptl** - converts doml 5.* file into ptl file.
  doml2ptl is used with the following parameters:

doml2ptl [-help] [doml5*-file] [ptl-file]

where:
help - prints message for usage and exits.
doml5-file* - doml file relese 5.*.
ptl-file **-** desired target ptl (Rational Rose) file.

- **olddoml_2_doml60** - converting doml 3.1 and 5.0 into doml 6.x file.
  olddoml_2_doml60 is used with the following parameters:
  olddoml_2_doml60 [-help] [olddoml-file] [doml60-file]
  where:
  help - prints message for usage and exits.
  olddoml_2_doml60 - doml file relese 3.1 or 5.0.
  doml60-file - desired target doml file relese 6.0.

- **doml31_2_conf60** - converting old style (doml 3.1) properties of doml elements, from doml 3.1 file, to configuration file properties parameter setings in output file.
  doml31_2_conf60 is used with the following parameters:
  doml31_2_conf60 [-help] doml31-file confProperty60-file
  [defaultLazyLoading] [defaultCaching]
  where:
  help - prints message for usage and exits.
  doml31-file - Original doml 3.1 file
  confProperty60-file - Output file where configuration properties will be written.
  defaultLazyLoading - If LazyLoading is not defined in the doml file (for that table) then this value is used (values: true/false).
  defaultCaching - If Caching is not defined in the doml file (for that table) then this value is used (values: true/false).

  If LazyLoading (or Caching) is not defined in the doml file, and there is no default value defined (defaultLazyLoading or defaultCaching), the following line is written to the application configuration file (no value) :

```
DatabaseManager.DB.<database_name>.<table_name>.cache.maxCacheSize=
DatabaseManager.DB.<database_name>.<table_name>.cache.initialCondition=
DatabaseManager.DB.<database_name>.<table_name>.lazyLoading=
```

Template sets

User can make its own template sets. All template sets (standard and users) are placed in one directory. The name and location of this directory is defined in the *dodsConf.xml* file. This file is detailly explained in the next section.

Also, this directory can be set from DODS *Ant task* (parameter *templateDir*).This was explained before in this chapter, in the DODS Ant task section.

Within this template set directory, every template set is placed in its own subdirectory.

Custom Configuration

To configure DODS, use *dodsConf.xml* file, located in <dods_root>/build/conf directory. This file contains the following information:

- Location of templates - tag `<TemplateDir>`

example:

```
<TemplateDir>C:/DODS/build/template</TemplateDir>
```

- For each database vendor, location of its configuration file (in xml format) - tag `<Database>`. The paths are relative to *dodsConf.xml* file folder.

example:

```
<Database>
   <Vendor name="Standard">StandardConf.xml</Vendor>
   <Vendor name="InstantDB">InstantDBConf.xml</Vendor>
   <Vendor name="Oracle">OracleConf.xml</Vendor>
   <Vendor name="Informix">InformixConf.xml</Vendor>
   <Vendor name="MSQL">MSQLConf.xml</Vendor>
   <Vendor name="Sybase">SybaseConf.xml</Vendor>
   <Vendor name="PostgreSQL">PostgreSQLConf.xml</Vendor>
   <Vendor name="HypersonicSQL">HypersonicSQLConf.xml</Vendor>
   <Vendor name="DB2">DB2Conf.xml</Vendor>
   <Vendor name="QED">QEDConf.xml</Vendor>
   <Vendor name="MySQL">MySQLConf.xml</Vendor>
</Database>
```

Database Vendor's configuration file contains information about that database (type of ObjectId, column name for oid and version, information about DeleteCascade, constraints, quotes,comments, characters for like and wildcard, mapping JDBC types to vendor-specific data types,...).

If tags `<ClassPath>`, `<ClassName>` in database vendor's configuration file are not mentioned, standard code is generated for that database vendor.

If database is specific, path to jar file for that database vendor is in tag `<ClassPath>`, and its main class is in tag `<ClassName>`.

example, for database Informix:

```
<ClassPath>C:/DODS/lib/dbvendors/informix.jar</ClassPath>
<ClassName>com.lutris.appserver.server.sql.informix.InformixLogicalDatabase<
/ClassName>
```

More Information

DODS, especially caching and transactions, is more detailly explained in the document:

"*Inside DODS*" (<dods_home>/doc/dods directory).

If DODS is installed within Enhydra, the Enhydra's *index.html* (<enhydra_root>/multiserver/webapps/enhydra-docs directory) containes link to *index_dods.html* (or *index_dods.pdf*) page.

In DODS, this page is root of doc directory (<dods_home>/doc).

This page (*index_dods*) contains the link to "*Inside DODS*" document.

Chapter 5

DiscRack sample application

This chapter introduces the DiscRack application, and uses it as a comprehensive example to illustrate key concepts of Enhydra application development. In order to follow that you should include sources of demos when installing Enhydra.

Building and running DiscRack

Enhydra includes the DiscRack application, which is installed in the <enhydra_root>/demos-source/discRack directory. Throughout this chapter, this top-level directory containing DiscRack is referred to as <DiscRack_root>. To build and run DiscRack, you need to do:

1 Check all application's configuration files (all discRack.conf.in files) and make sure all Database Manager configuration settings are correct

2 Build the application by entering the *enhydra-ant* command from the <DiscRack_root> directory. Building the application will generate all the classes and packages for the DiscRack application.

Note The DiscRack database and corresponding application data layer are identical to the those created in the chapter 4, "Tutorial: Building Enhydra applications," with the exception of package naming. This database schema is loaded for you by default, using an hsql database.

3 The jar files for default hSQL database are included in Enhydra.

4 To run DiscRack, enter the following commands:

1 UNIX:

```
cd <DiscRack_root>/application/bin
./start-enhydra
```

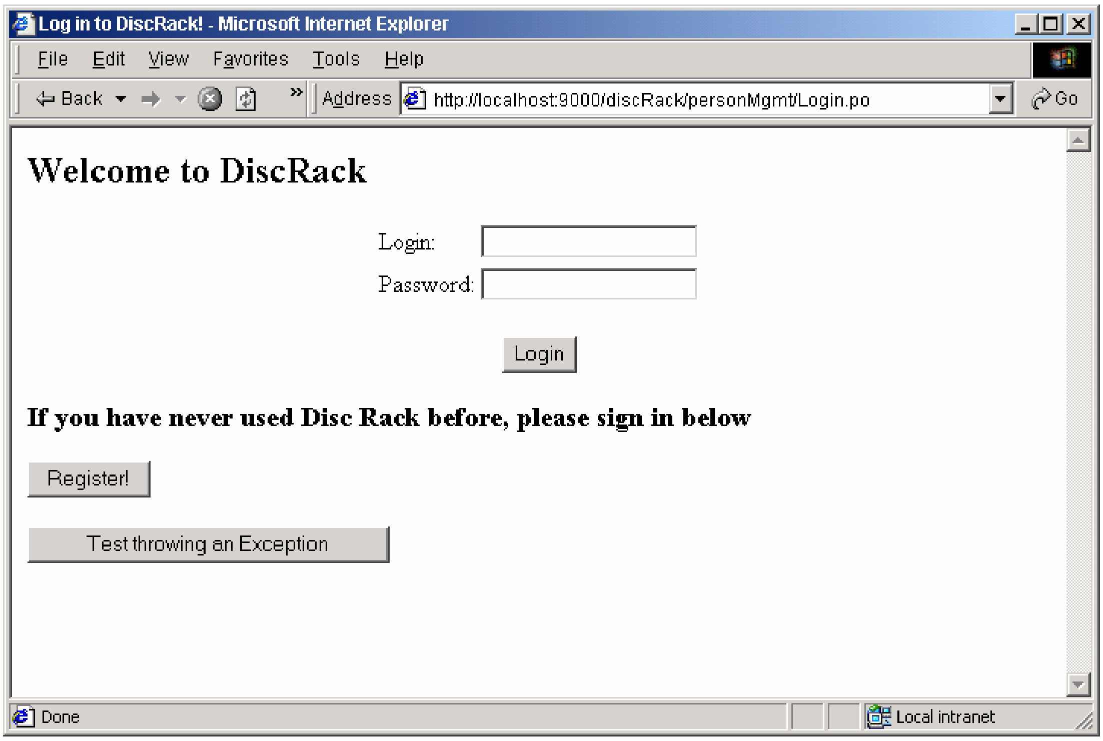
2 WINDOWS:

```
cd <DiscRack_root>/application/bin
start-enhydra
```

5 To access the application, enter the URL http://localhost:8080/discRack (http://localhost:9000/discRack for Enhydra Enterprise installation) in your browser location field.

Note: Communication port value is one of Enhydra 6.x installation options. Default value is set to 8080 for Enhydra and 9000 for Enhydra Enterprise installation.

Your browser displays the following screen:


Log in to DiscRack! - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address http://localhost:9000/discRack/personMgmt/Login.po
Go
Login
Register!
Test throwing an Exception
Done
Local intranet

Welcome to DiscRack

Login:

Password:

If you have never used Disc Rack before, please sign in below

Figure 5.1 Browser displaying the DiscRack Login presentation object

Play around with the application to get a sense for how it works:

- Click the Register button to add yourself as a user, then add some discs to your inventory.
- Try viewing your inventory and editing one of the discs.

Process and preliminaries for developing applications

Before discussing the workings of the DiscRack application, it is useful to understand how you go about developing an Enhydra application in general. You can adapt the traditional software development process to Enhydra application development to ensure that:

- The application does what it is supposed to do.
- You complete the project in a timely and cost-effective manner.
- The application is easy to maintain and upgrade.

An in-depth discussion of software methodology is beyond the scope of this book, but it is helpful to understand the basic principles and how they apply to the simple DiscRack application, so that you can reap the benefits when developing a more complex, real-world application.

A simplified Enhydra application development process consists of these steps:

- Requirements definition

  As specifically as possible, create a statement of what the application is supposed to accomplish. This statement essentially defines the high-level goals of the application.

- Functional specification

  Outline how the application solves the problem(s) stated in the requirements definition.

- Design and storyboard

  Design the presentation, data, and business layers of the application, then create the storyboard.

- Development and testing

  Code and test the application.

- Deployment

  Pack the application and install it in its operational environment.

This abbreviated methodology illustrates the key aspects of the development process. Complex, real-world applications generally call for a more comprehensive process that includes project milestones, cost analysis, documentation, and so on. The following sections illustrate these abbreviated steps.

DiscRack requirements definition

The Otter family needs a way to track their compact disc collections. Each family member has a CD collection, and they sometimes get mixed up: Otters forget who owns what. They decide that an Enhydra application would be the perfect way to help them manage their CDs. After some discussion, they arrive at a brief requirements definition:

DiscRack will let each user keep track of his or her individual CD inventory by adding, modifying, and deleting CDs as needed. The application will keep track of all the pertinent information about each CD, including artist and title.

DiscRack functional specification

Briefly, DiscRack will meet its requirements as follows:

- Maintain a list of users and passwords

  To access their CD inventory, users must log in with a user name and password.

- Allow new users to sign up by entering their name, user name, and password.

- Once logged in, a user can see his or her CD inventory and:

  - Add new CDs to the inventory.

  - Edit existing CD entries.

  - Delete an existing entry, with a confirmation prompt.

- The information that will be displayed for each CD includes artist, title, genre, and whether or not the user likes the CD.

Design and storyboard

The bulk of this step consists of the engineering design for the application, including the design of database schema and corresponding data layer, business logic, and presentation logic. The user interface design can be largely encapsulated by a storyboard.

A *storyboard* is a visual way of describing a user's navigation paths through the application. It provides an outline of the application's user interface, and a framework from which the rest of the application design can proceed.

A conceptual storyboard, which is largely an application flowchart, is sometimes referred to as a site *map*, in contrast to a mocked-up HTML storyboard. This book refers to both as a storyboard. The storyboard for DiscRack is shown in Figure 5.2.

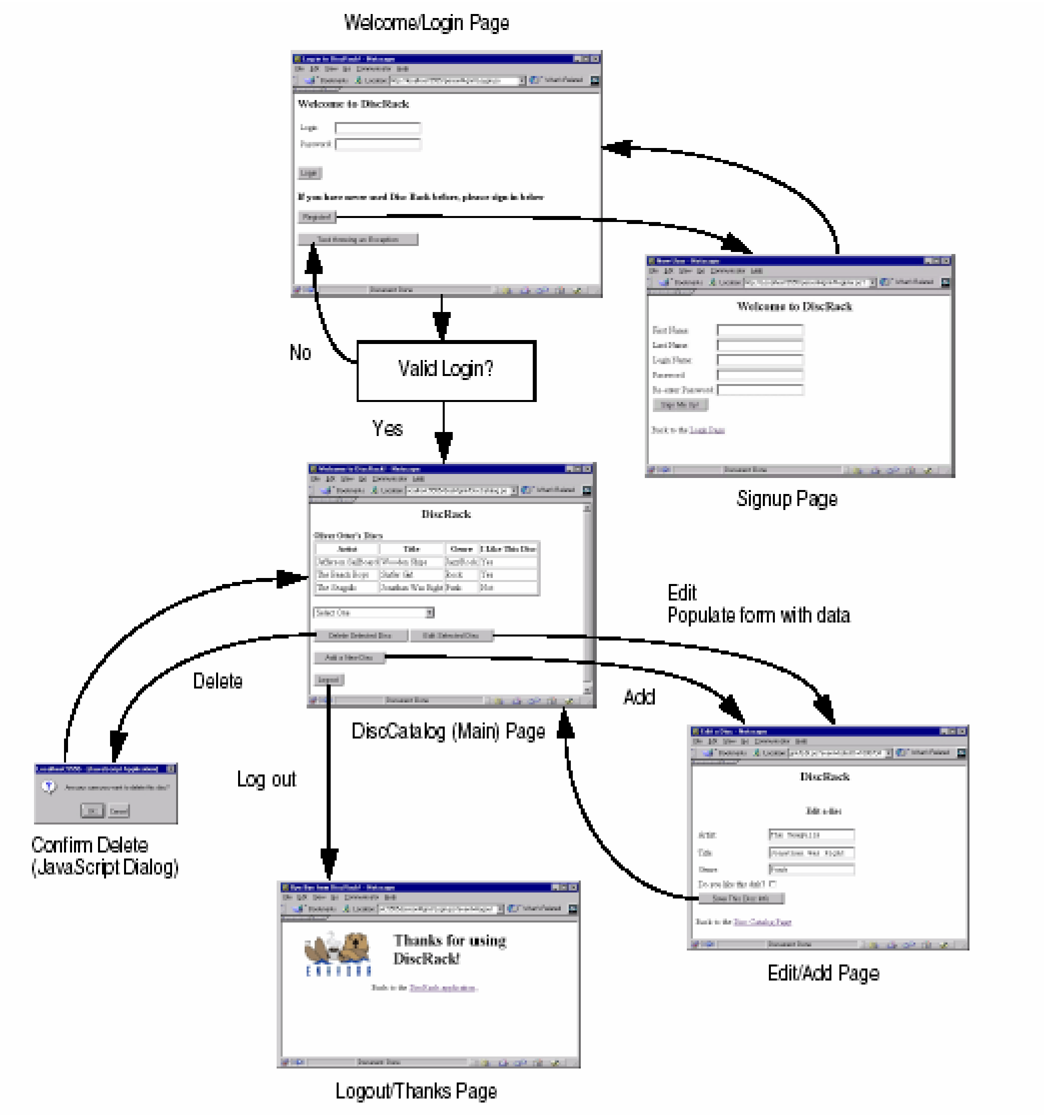


Figure 5.2 DiscRack Storyboard

You can see from the storyboard that there are five HTML pages in the application.

You can also see that the DiscCatalog page that shows the CD inventory is the central page in the application. The first page the user sees will always be the Login page; the last page will always be the Logout page.

DiscRack includes a working storyboard (or application "mockup") in the resources directory. It is a set of static HTML pages that illustrate how the application works. To see the storyboard, load this file in your browser:

```
<DiscRack_root>/discrack/presentation/resource/personMgmt/Login.html
```

This displays the DiscRack login page.

- Click the Login button to log in and see the disc catalog.
- Click the Register button to display the Register page.
- Click around on the links to can see the rest of the storyboard.

The flow of the HTML pages follows Figure 5.2. Of course, none of the back-end logic is activated - all the HTML is static. But the storyboard gives you a good feel for how the application works.

Developing, testing, and deploying

To finish the application process, the remaining steps include developing, testing, and finally deploying.

When you build an application from the top level (with *enhydra-ant* command) , the build.xml files create <DiscRack_root>/application/bin directory containing the configuration files, and *start-enhydra* and *stop-enhydra* scripts. Also, the application's .jar file is generated. It contains all the class files for the application, along with any other files (for example, GIFs or stylesheets).

To deploy the application, you need to start Enhydra with *start-enhydra* script.

The rest of this chapter describes the DiscRack application itself.

Overview of DiscRack

The basic DiscRack application consists of 23 classes in 9 packages. The fundamental package structure and class functions for DiscRack are described in Table 5.1:

Table 5.1 DiscRack Application Overview

Class or package name	Description
discRack package	
DiscRack	Application object
DiscRackException	Simple base exception class
Presentation layer/package	
BasePO	Abstract base class for all presentation objects
DiscRackSessionData	Container for session data
ErrorHandler	Class to handle exceptions not caught elsewhere in the application
DiscRackPresentationException	Presentation layer exception class
presentation.personMgmt package	Package that contains the Register and Login classes for managing presentation related to the PERSON table
presentation.discMgmt package	Package that contains the Edit and DiscCatalog classes for managing presentation related to the DISC table

Table 5.1 DiscRack Application Overview (continued)

Class or package name	Description
Business layer/package	
DiscRackBusinessException	Business layer exception class
business.person package	Package that contains two classes: • Person, which represents a person • PersonFactory, which has a single method that returns the Person object for a user name
Person object for a user name	Package that contains two classes: • Disc, which represents a disc • DiscFactory, which has methods to return a Disc object for an ID or for the owner's name.
Data layer	Described in "Loading the schema" in the chapter 4.
WAP layer/package	

The six HTML files are in the *presentation/resource* directory. These correspond to the five HTML pages shown in the storyboard, plus an error page that appears when an error occurs that is not handled by an exception.

Presentation layer

The presentation layer includes all of the HTML, Java, and JavaScript that defines the user interface of the application.

Presentation base class

All of the presentation objects in DiscRack are derived from a common base class, BasePO, which is an implementation of the Enhydra interface HttpPresentation. This interface has one method, *run()*, which takes the HTTP request as a parameter.

A presentation base class enables the application to group common functionality in one place. Notice that BasePO is an abstract class, so it cannot be instantiated itself, only subclassed. Also, some of its methods are declared abstract, so subclasses must implement them.

BasePO has methods to handle some of the key tasks for DiscRack:

- User log in and session maintenance

- Event handling and calling the HTML generation methods in the subclass presentation objects

Note It is important to realize that you are not required to use a base presentation class. An alternative is to use the Enhydra Application object to perform common tasks.

The central method in BasePO is *run()*, which makes method calls to perform session maintenance and event handling:

```
public void run(HttpPresentationComms comms) throws Exception {
  // Initialize new or get the existing session data
  initSessionData(comms);
  // Check if the user needs to be logged in for this request.
  if(this.loggedInUserRequired()) {
   checkForUserLogin();
  }
  // Handle the incoming event request
  handleEvent(comms);
}
```

Every time a client browser requests a presentation object URL, the application calls run(). Its logic is very simple:

- Initialize or get the existing session data by calling *initSessionData()*.
- If this presentation object requires a log in (as determined by *loggedInUserRequired()*, an abstract method implemented by each presentation object), then call *checkForUserLogin()* to determine if the user has already logged in. If not, then redirect the browser to the login page.
- *Call handleEvent()* to handle the current event and determine what HTML to generate.

Each of these methods are explained in the following sections.

The *run()* method has one parameter, comms, that is an object containing information about the HTTP request. Its member properties include application, exception, request, response, session, and sessionData. These six properties provide all of the information for the request.

For example, you can retrieve session data with *getComms().sessionData.get()* and query string parameters with *getComms().request.getParameter()*.

Session data and log in

The basics of Enhydra session maintenance were introduced in “Maintaining session state” in the chapter 4. In contrast to the way session information was handled in that example, DiscRack stores all its session information in a single *DiscRackSessionData* object and saves that object in the user’s session.

DiscRackSessionData is a simple container class containing methods to get and set these member properties:

- A *Person* object that represents the user
- A string, called *userMessage*, for error messages such as “Please choose a valid disc to edit”

There are several advantages of keeping session data in one object:

- It centralizes control of session information.

  This is especially helpful when multiple presentation objects access the same session data.

- It is type-safe.

  Because *Session.getSessionData()* returns a generic Object, if you store session data separately, you will have to cast each item to the appropriate type, which can lead to runtime errors that are hard to debug.

- It facilitates session data maintenance.

  If there is a large amount of session data, you can periodically clean up the unneeded data. For example, say you wanted to store an array of hundreds of discs in the user’s session to speed access, but you didn't necessarily want leave it there until they log out. With a session data object, you could easily implement a method to clean up unneeded data in the session.

initSessionData() method

The first thing each presentation object does is to call *initSessionData()*. The main portion of this method is shown here:

```
Object obj =
getComms().sessionData.get(DiscRackSessionData.SESSION_KEY);
if(null != obj) {
 this.mySessionData = (DiscRackSessionData)obj;
} else {
 this.mySessionData = new DiscRackSessionData();
 getComms().sessionData.set(DiscRackSessionData.SESSION_KEY,
 this.mySessionData);
}
```

The first statement in this code snippet gets the session data object, using the session key "DiscRackSessionData". If the session data object exists, it gets typecast to DiscRackSessionData; otherwise, the code creates a new DiscRackSessionData object and saves it to the user's session with set().

loggedInUserRequired() method

BasePO has an abstract method called *loggedInUserRequired()* that returns a boolean value, which indicates whether a user is required to be logged in to access the associated page. Thus, every presentation object is required to implement this method.

In *BasePO.run()*, if this method returns true, then *checkForUserLogin()* is called.

checkForUserLogin() method

The *checkForUserLogin()* method determines if a user has a valid login. If not, then it redirects the browser to the Login page:

```
...
Person user = getUser();
if (null == user) {
...
 throw new ClientPageRedirectException(LOGIN_PAGE);
}
...
```

Several statements that write debug messages to a log channel have been removed from this code for clarity.

The call to *getUser()* is really just a call to *getSessionData().getUser()*, which retrieves the Person object saved in the current session. If the user has not logged in, or the session has timed out, then this method returns null, and the code will throw a ClientPageRedirectException with the URL to the Login page as the argument to the constructor.

When a client browser is redirected by a ClientPageRedirectException, any parameters from a query string that were available to the original presentation object are lost. So if you want to pass an error message, you must put the information in the user's session or directly into the query string of the redirected URL.

Event handling

While you could create a separate presentation object for each task in an application, in many cases it makes sense to have a single presentation object handle multiple events. For example:

- *Edit* presentation object responds to four events - showing the add page, showing the edit page, actually adding a disc to the database, and deleting a disc from the database.

- *Login* presentation object handles three events -show page, login, and logout.

Note In this context, an "event" refers to the task a user is performing.

Setting the event parameter

DiscRack keeps track of the event it is processing with the *event* parameter, which is sent in the query string of a request. For example, this URL specifies the event *showAddPage*:

```
http://Localhost:8080/discRack/discMgmt/Edit.po?event=showAddPage
http://Localhost:9000/discRack/discMgmt/Edit.po?event=showAddPage
```

Note: Communication port value is one of Enhydra 6.x installation options. Default value is set to 8080 for Enhydra and 9000 for Enhydra enterprise installation.

DiscRack illustrates several techniques for setting the event:

- *showAddPage* event is defined in the *DiscCatalog.html* page by the JavaScript onClick event handler of the Add a New Disc button.

  This calls the JavaScript function *showAddPage()*, which explicitly adds the event to the URL requested:

  ```
  document.location='Edit.po?event=showAddPage'
  ```

  This function is defined in *presentation/resource/discMgmt/DiscCatalogScript.html*, not the DiscCatalog page, as explained in "Replacing JavaScript", later in this chapter.

- event (to add a disc to the database) is defined in the Edit.html page by a hidden form field:

  ```
  <input type="hidden" name="event" value="add" id="EventValue">
  ```

  When the user clicks the Add button, `event=add` is added to the form submission request along with the other form data the user entered.

- *exit* event is defined in the *DiscCatalog.html* page by the second form's ACTION attribute:

  ```
  "../personMgmt/Exit.html"
  ```

  At compile time, this URL, as explained in "URL mapping" (later in this chapter), is replaced by:

  ```
  ../personMgmt/Login.po?event=logout'
  ```

  Although DiscRack does not demonstrate it, you can also set the event when you throw a

PageRedirectException. You use this exception to transfer control from one presentation object to another. To specify an event, add this string to the URL string passed to the constructor of PageRedirectException:

```
"?event=someEvent"
```

handleEvent() method

Once the event is set, the *handleEvent()* method of *BasePO* performs the actual event handling:

```
String event = getComms().request.getParameter(EVENT);
String returnHTML = null;

 if (event == null || event.length() == 0) {
  returnHTML = handleDefault();
 } else {
  returnHTML = getPageContentForEvent(event);
 }
 getComms().response.writeHTML(returnHTML);
```

This method gets the event parameter from the request query string and calls the appropriate event handler. If it does not find event in the request query string, it calls handleDefault(), which is an abstract method and so must be implemented by all BasePO subclasses. Otherwise, it calls getPageContentForEvent(), which returns the string content for the specific event and PO.

This method contains the following three lines:

```
Method method = this.getClass().getMethod(toMethodName(event), null);
String thePage = (String)method.invoke(this, null);
return thePage;
```

This code uses reflection (defined in the java.lang.reflect package) to call the method in the presentation object corresponding to the current event. Reflection lets you call a method whose name is defined at runtime.

The call to *toMethodName()* returns a string, *handleXxx*, where *Xxx* is the current event (for example, *handleShowAddPage* for showAddPage). The call to method.*invoke()* then calls this method.

Reflection allows *BasePO* to call methods in its subclasses without knowing in advance the names of the methods. This scheme works as long as the presentation object code follows the appropriate naming conventions:

For every event “*foo*”, there must be a method *handleFoo()* in the presentation object class that needs to handle that event.

HTML pages

You will find the HTML pages for DiscRack in the <discRack_root>/discRack/presentation/resource directory. Keeping the HTML pages there rather than in the presentation/src directory cleanly separates the HTML files from the Java files. Although this is superfluous for small applications, it is a key advantage for large applications with a graphic design team and a programming team.

The options.xmlc files in the presentation layer (directory resources) controls how the application uses the HTML files.

Maintaining the storyboard

The *storyboard* is initially just a mockup of the application. But with a few simple steps, you can maintain a working storyboard throughout the entire development process. This capability becomes particularly important for large applications created by a team of programmers and graphic designers. Each team can work on their part of the application separately from the other.

After the graphic designers complete their work, you can then replace the old, “mock up” user interface with the new improved interface, which may include enhanced graphics, JavaScript special effects, stylesheets, and so on. An example of doing this is illustrated in “Replacing the user interface”, later in this chapter.

In addition to keeping the HTML files separate from the Java code, as described in the previous section, there are three steps you must follow during development to maintain the storyboard:

1 Define rules to map URLs like *Login.html* to *Login.po*

2 Remove dummy data from the HTML files

3 Replace JavaScript, if necessary

Each of these steps is described in detail in the following sections.

URL mapping

In the working storyboard, as in any static HTML pages, hyperlinks reference other HTML pages. That is, the URLs in hyperlinks end in *.html*. However, in the working application, links to dynamic pages reference presentation object URLs that end in *.po*. So, you need to do something to convert the “normal” URLs in the storyboard to .po URLs.

You do this by using the XMLC -urlmapping option to map URLs from one form to another. You use this option like this:

```
-urlmapping oldURL newURL
```

To use this option in the build process, you must create an XMLC options file *options.xmlc*.
For example:

The presentation/resource/discMgmt/options.xmlc file contains the lines:

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

When XMLC compiles the files in this directory, it replaces occurrences of the first string (for example, Edit.html) with the second string (for example, Edit.po) in hyperlink URLs and FORM ACTION attributes.

Removing dummy data

HTML files often contain “dummy” data to make the storyboard pages look more representative of their actual runtime appearance. You need to remove this dummy data from the production application.

Look in presentation/resource/discMgmt/*options.xmlc* again. In particular, look at the last line:

```
 -delete-class discardMe
```

The **-***delete-class* option tells XMLC to remove any tags (and their contents) whose `CLASS` attribute is *discardMe*. For example, if you look in presentation/resource/discMgmt/*DiscCatalog.html*, you see this HTML:

```
<tr class="discardMe">
 <td>Sonny and Cher</td>
 <td>Greatest Hits</td>
 <td>Boring Music</td>
 <td>Not</td>
</tr>
```

It’s not that we don’t like Sonny and Cher, however, the `CLASS` attribute in the table row definition marks the row for deletion.

Unlike `ID`, the value of a `CLASS` attribute does not have to be unique in the page. You can remove all of the dummy in the application with the same *discardMe* value.

Replacing JavaScript

In addition to replacing URLs, you often need to replace JavaScript in the storyboard with JavaScript to be used in the “real” application. For example, *presentation/resource/DiscCatalog.html* contains the following script:

```
<SCRIPT id="DummyScript">
<!--
function doDelete()
{
 document.EditForm.action='DiscCatalog.html';
 if(confirm('Are your sure you want to delete this disc?')) {
 document.EditForm.submit();
 }
}
function showAddPage()
{
 document.location='Edit.html';
}
//-->
</SCRIPT>
```

These functions help to keep the storyboard working. At runtime, though, the application needs to use the “real” functions, which are defined in presentation/*DiscCatalogScript.html*. For example:

```
...
function showAddPage()
{
 document.location='Edit.po?event=showAddPage';
}
...
```

Because XMLC views JavaScript as a comment, the URL mapping option will not work on this

URL inside the JavaScript function. So, you have to replace it at runtime with the following code in *DiscCatalog.java*:

```
DiscCatalogHTML page = new DiscCatalogHTML();
HTMLScriptElement script = new DiscCatalogScriptHTML().getElementRealScript();
XMLCUtil.replaceNode(script, page.getElementDummyScript());
```

This is an example of replacing a node with a node from another document. This implementation uses the **XMLCUtil** class.

Note Because this action happens at runtime, it may have a slight affect on performance. If performance is critical, you may want to replace the JavaScript in the final deployed version of the application.

Maintaining the storyboard seems like additional unnecessary work, but it is worth the effort when your HTML is evolving in parallel with the Java code. As an example of the power of a working storyboard, you can exchange the HTML in DiscRack from the basic HTML to designed HTML.

Replacing the user interface

Once the graphic design is completed, you can replace the user interface of the application with its final version. Enhydra includes in documentation in *<enhydra_root>/multiserver/webapps/enhydra-docs/getting_started/samples/resourceForDiscRack* directory “finished” versions of the HTML pages.

To replace the original storyboard resources with the “finished” resources:

1 Copy HTML files (Exit.html, Login.html and Register.html) from directory <enhydra_root>/multiserver/webapps/enhydra-docs/getting_started/samples/resourceForDiscRack/HTML_personMgmt to directory <DiscRack_root>/presentation/resource/personMgmt

2 Copy HTML files (DiscCatalog.html and Edit.html) from directory <enhydra_root>/multiserver/webapps/enhydra-docs/getting_started/samples/resourceForDiscRack/HTML_discMgmt to directory <DiscRack_root>/presentation/resource/discMgmt

3 Rebuild the application by entering the following commands from the application’s root directory <DiscRack_root>:

1 **UNIX:**

```
cd <DiscRack_root>
./enhydra-ant clean
./enhydra-ant
```

2 **WINDOWS:**

```
cd <DiscRack_root>
enhydra-ant clean
enhydra-ant
```

The *enhydra-ant clean* command removes all the old classes so that *enhydra-ant* will completely rebuild the application from scratch.

4 Now, run DiscRack by entering the following commands:

1 UNIX**:**

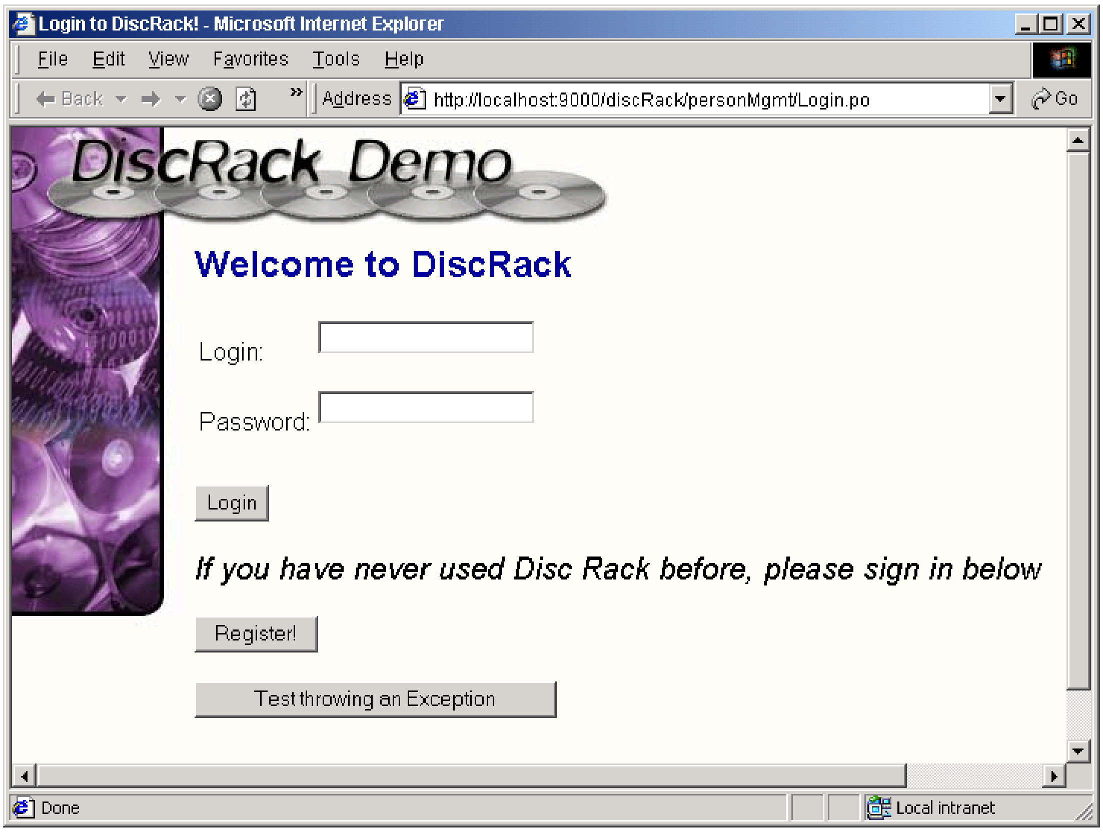
```
cd application/bin
./start-enhydra
```

2 WINDOWS:

```
cd application/bin
start-enhydra
```

5. To see the new and improved user interface, enter the URL http://localhost:8080/discRack (http://localhost:9000/discRack) in your browser location field.

> **Note:** Communication port value is one of Enhydra 6.x installation options. Default value is set to 8080 for Enhydra and 9000 for Enhydra Enterprise.



Login to DiscRack! - Microsoft Internet Explorer

Address http://localhost:9000/discRack/personMgmt/Login.po

Welcome to DiscRack

If you have never used Disc Rack before, please sign in below

Figure 5.3 Browser displaying the DiscRack Login presentation object with updated graphics

Populating a list box

The DiscCatalog page illustrates how to populate a SELECT list box, which is a common task. First, look at the HTML for the SELECT tag in DiscCatalog.html:

```
<SELECT id="TitleList" Name="discID">
<OPTION selected VALUE="invalidID">Select One</OPTION>
<OPTION id="templateOption">Van Halen: Van Halen One</OPTION>
<OPTION class="discardMe">Sonny and Cher: Greatest Hits</OPTION>
<OPTION class="discardMe">Sublime: 40 oz. to Freedom</OPTION>
</SELECT>
```

Now look in *DiscCatalog.java* for the code that populates the list box:

```
HTMLOptionElement templateOption = page.getElementTemplateOption();
Node discSelect = templateOption.getParentNode();
```

The first line retrieves the DOM object corresponding to the template OPTION tag. The second line calls *getParentNode()* to get the container SELECT tag. Because the SELECT tag has an ID attribute, this line could have also been:

```
Node discSelect = page.getElementTitleList();
```

Then, following some code for populating the table, there is one line to remove the template row.

```
templateOption.removeChild(templateOption.getFirstChild());
```

The other OPTION tags contain CLASS="discardMe", which causes XMLC to remove those items at build time, as explained before in “Removing dummy data”.

Then, within the for loop that iterates over the discs belonging to the current user, the following lines actually populate the list box:

```
HTMLOptionElement clonedOption = (HTMLOptionElement)
templateOption.cloneNode(true);
clonedOption.setValue( currentDisc.getHandle() );
Node optionTextNode =
    clonedOption.getOwnerDocument().createTextNode(currentDisc.getArtist() +
    ": " + currentDisc.getTitle());
clonedOption.appendChild(optionTextNode);
discSelect.appendChild(clonedOption);
```

The first line copies (clones) the template option element into a DOM object of type *HTMLOptionElement*. The second line sets the value attribute to the value returned by *getHandle(),* which is the disc’s OBJECTID, an unique identifier.

The third (very long) line creates a text node consisting of *artistName: titleName*. Finally, the last two lines append the text node to the option node, and then append the option node to the select node.

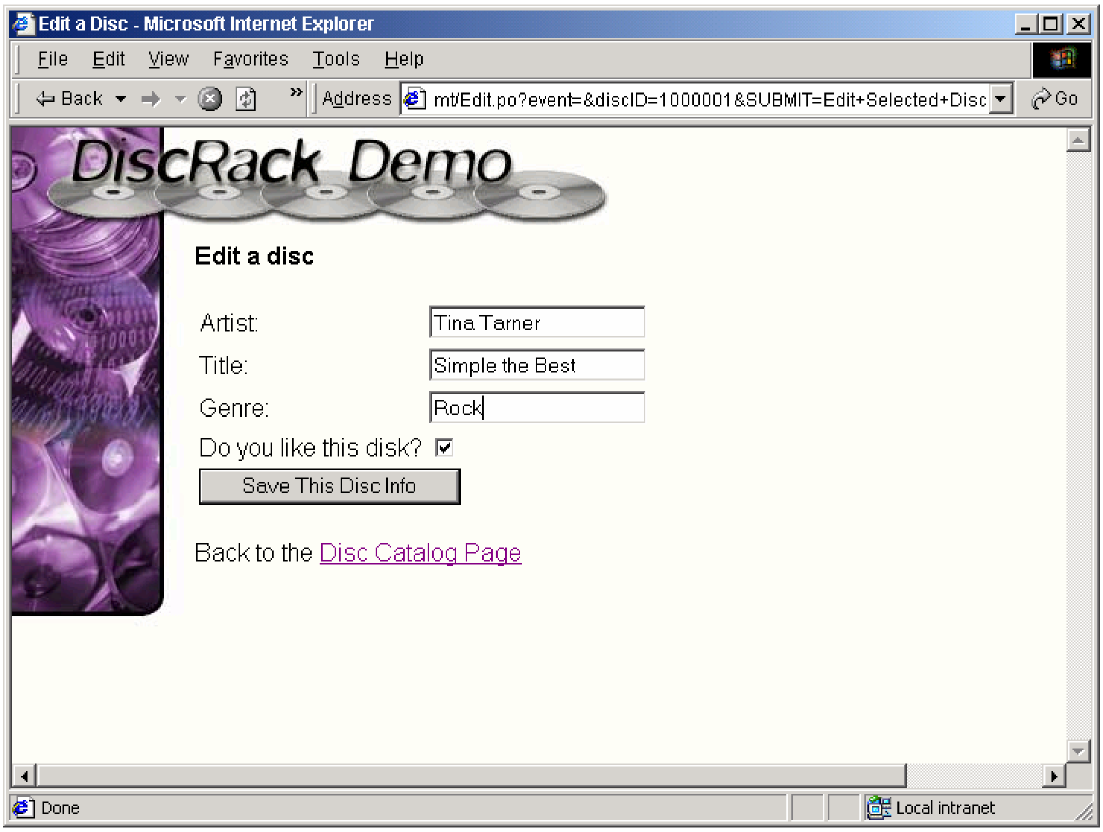
The resulting runtime HTML will look something like this:

```
<SELECT name='discID' id='TitleList'>
<OPTION value='invalidID' selected>Select One</OPTION>
<OPTION value='1000001'>Funky Urchin: Lovely Spines</OPTION>
<OPTION value='1000021'>The Seagulls: Screaming Fun</OPTION>
</SELECT>
```

Although this example might seem obscure, it is fairly short, and you can extend its basic functionality to handle more complex situations. For example, you can modify it to set the default selection based on a second query.

Populating a form

When a user chooses a disc from the list box and clicks the Edit Disc button, the browser displays a form. As shown in Figure 5.4, the edit form is populated with the existing values for that disc. The user can then edit the values and submit them back to the database.



Edit a Disc - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address
Go
DiscRack Demo
Done
Local intranet

Address: mt/Edit.po?event=&discID=1000001&SUBMIT=Edit+Selected+Disc

Edit a disc

Artist: Tina Tarner

Title: Simple the Best

Genre: Rock

Do you like this disk? ☑

Save This Disc Info

Back to the Disc Catalog Page

Figure 5.4 DiscRack disc edit form

Here is the HTML for the form elements in *Edit.html*. The TABLE tags have been omitted for clarity:

```
<INPUT TYPE="hidden" NAME="discID" VALUE="invalidID" ID="DiscID">
Artist: <input name="artist" id="Artist" >
Title: <input name="title" id="Title" >
Genre: <input name="genre" id="Genre" >
Do you like this disk?
<input TYPE="checkbox" name="like" CHECKED ID="LikeBox">
<INPUT TYPE="submit" VALUE="Save This Disc Info">
```

In *Edit.java*, the event-handling method *handleDefault()* calls *showEditPage()* with a null parameter to populate the form with the selected disc's values. Ordinarily, the only request parameter (other than the event type) is the disc ID, accessed by this statement:

```
String discID = this.getComms().request.getParameter(DISC_ID);
```

These statements also access the other request parameters, but ordinarily they are null (but see the error-handling case discussed later):

```
String title = this.getComms().request.getParameter(TITLE_NAME);
String artist = this.getComms().request.getParameter(ARTIST_NAME);
String genre = this.getComms().request.getParameter(GENRE_NAME);
```

Then, a call to *findDiscByID()* retrieves a Disc data object that has that ID:

```
disc = DiscFactory.findDiscByID(discID);
```

Next, there is a series of **if** statements that check the values of *title, artist, genre*, and *isLiked*, which are normally null. Therefore, the following statements are executed (the surrounding if statements are not shown for brevity):

```
page.getElementDiscID().setValue(disc.getHandle());
page.getElementTitle().setValue(disc.getTitle());
page.getElementArtist().setValue(disc.getArtist());
page.getElementGenre().setValue(disc.getGenre());
page.getElementLikeBox().setChecked(disc.isLiked());
```

These statements use XMLC calls to set the value attributes of the form elements; the values are retrieved from the Disc object.

When the user finishes editing and clicks Save this Disc Info, *handleEdit()* processes the changes. This method calls *saveDisc()*, which attempts to save the new values:

- If successful, it redirects the client to the DiscCatalog page.
- If any of the new values are null, though, *saveDisc()* throws an exception.

The *catch* clause then calls *showEditPage()* with an error string and request parameters.

Note *ClientPageRedirectException* is a subclass of java.lang.Error, so it is not caught by the catch clause when that statement is thrown.

```
try {
 saveDisc(disc);
 throw new ClientPageRedirectException(DISC_CATALOG_PAGE);
} catch(Exception ex) {
 return showEditPage("You must fill out all fields to edit this
 disc");
}
```

The result is that when a user tries to edit a disc and deletes some of the values, the edit page redisplays, maintaining all the non-null form element values and restoring the previous values to the null-valued form elements. The page also displays the error string.

Business layer

The DiscRack business layer is simple, consisting primarily of:

- Two packages - *disc* and *person*
- Two corresponding factory classes - *DiscFactory* and *PersonFactory*.

A *factory* is an object whose primary role is to create other objects.

Business objects

The business objects *Disc* and *Person* are largely wrappers for the corresponding data layer classes, *DiscDO* and *PersonDO*, with get and set methods for each property in the data objects (or column in the database tables). For example, *Disc* has *getArtist()* and *setArtist()* methods.

The objects in the business layer perform all the interfacing with the data layer. So, if the data layer needs to change, nothing in the presentation layer is affected.

Conversely, if the presentation layer changes, nothing in the data layer is affected.

DiscFactory has two static methods:

- *findDiscsForPerson()* returns an array of *Disc* objects that belong to the *Person* object specified as the method's argument.

- *findDiscByID()* returns the single *Disc* object that has the ID specified in the method's argument.

PersonFactory has one static method*, findPerson().* It returns a *Person* object that has the user name specified in the method's argument. If the method finds more than one person in the database, then it writes an error message to the log channel and throws an exception.

Using data objects

To help understand how DiscRack uses the DODS data layer code, look at the *findPerson()* method in *PersonFactory*. The comments have been removed from this code for brevity.

```
public static Person findPerson(String username)
throws DiscRackBusinessException
{
 try {
 PersonQuery query = new PersonQuery();
 query.setQueryLogin(username);
 query.requireUniqueInstance();
 PersonDO[] foundPerson = query.getDOArray();
 if(foundPerson.length != 0) {
  return new Person(foundPerson[0]);
 } else {
  return null;
 }
 } catch(NonUniqueQueryException ex) {
...
```

First, this method instantiates a new *PersonQuery* object. *PersonQuery* is a data layer object used to construct and execute a query on the person table. It has a number of *setQueryxxx()* methods for qualifying the query parameters (that is, setting the values to be matched in the WHERE clause of the SELECT statement). For example, the above code calls *setQueryLogin()* with *username* as a parameter to set the value to be matched in the LOGIN column.

Next, the method calls *requireUniqueInstance()*, which indicates that the query is to return a single row, and will throw an exception otherwise. Then, it calls *getDOArray()*, which executes the query, returning an array of PersonDO objects. Finally, the method returns a single *Person* object returned by the query; if the query did not return any rows, it returns null.

Appendix A

Database configurations

This appendix provides information on connecting Enhydra applications to specific database types. In general, you need to add the database configuration information to the application configuration file (e.g., simpleApp.conf). Configurable items in the code snippets that you need to specify, such as path names or database identifier, are enclosed in brackets and italicized (for example, <*path_name*> or <*database_id*>).

Driver configuration

Important Enhydra connects to databases using a JDBC driver. Enhydra has its own class loader, but the JDBC driver must be loaded by the system class loader. Therefore, it is important to specify the path to the JDBC driver in your system CLASSPATH and not in the Enhydra application's CLASSPATH.

A common way to specify the path to the JDBC driver is to save the driver in a lib directory in the project and define the CLASSPATH in the run script. To do this, follow these steps:

1 Create a lib directory in the top level of your project and copy your JDBC driver to this directory.

2 Edit your application's run file template, start.in, (in the <*appName*>*/*input directory) to place the driver in your CLASSPATH. For example:

```
...
#
# Build up classpath.
#
CLASSPATH="../lib/idb.jar\;../lib/jta-spec1_0_1.jar"
APPCP="${ENHYDRA_LIB}${PS}../classes"
...
```

3 Build the project with ant, which will copy the run script to the directory <*appName*>/output. Use this script to start your application.

Be careful to keep the right driver with your application. For example, there are multiple versions of the Oracle JDBC driver, classes111.zip. When your application goes into production, make sure that the project administrator knows to reference the correct driver when the database is upgraded in the future.

Oracle

This section presents an example of an Oracle configuration, where <*database_id*> is your database identifier.

```
#-----------------------------------------------------------------
# Database Manager Configuration
#-----------------------------------------------------------------
DatabaseManager.Databases[] = "<database_id>"
DatabaseManager.DefaultDatabase = "<database_id>"
DatabaseManager.Debug = "false"
DatabaseManager.DB.<database_id>.ClassType = "Oracle"
DatabaseManager.DB.<database_id>.JdbcDriver = "oracle.jdbc.driver.OracleDriver"
DatabaseManager.DB.<database_id>.Connection.Url =
"jdbc:oracle:thin:@<server_name>:<port#>:<db_instance>"
DatabaseManager.DB.<database_id>.Connection.User = "<user>"
DatabaseManager.DB.<database_id>.Connection.Password = "<password>"
DatabaseManager.DB.<database_id>.Connection.MaxPreparedStatements = 10
DatabaseManager.DB.<database_id>.Connection.MaxPoolSize = 30
DatabaseManager.DB.<database_id>.Connection.AllocationTimeout = 10000
DatabaseManager.DB.<database_id>.Connection.Logging = false
DatabaseManager.DB.<database_id>.ObjectId.CacheSize = 20
DatabaseManager.DB.<database_id>.ObjectId.MinValue = 1
```

The driver used here is the Oracle thin driver, and *<db_instance>* is the name of the Oracle database instance.

This is the link where you can find all needed information and downloads for Oracle database: Oracle

Informix

This section presents an example of an Informix configuration, where *<database_id>* is your database identifier.

```
#-------------------------------------------------------------------
# Database Manager Configuration
#-------------------------------------------------------------------
DatabaseManager.Databases[] = "<database_id>"
DatabaseManager.DefaultDatabase = "<database_id>"
DatabaseManager.Debug = "false"
DatabaseManager.DB.<database_id>.ClassType = "Informix"
DatabaseManager.DB.<database_id>.JdbcDriver = "com.informix.jdbc.IfxDriver"
DatabaseManager.DB.<database_id>.Connection.Url =
jdbc:informix-sqli://<hostname>:<port#>:INFORMIXSERVER=<db_instance>;
user=<user>;password=<password>
DatabaseManager.DB.<database_id>.Connection.User = "<user>"
DatabaseManager.DB.<database_id>.Connection.Password = "<password>"
DatabaseManager.DB.<database_id>.Connection.MaxPreparedStatements = 10
DatabaseManager.DB.<database_id>.Connection.MaxPoolSize = 30
DatabaseManager.DB.<database_id>.Connection.AllocationTimeout = 10000
DatabaseManager.DB.<database_id>.Connection.Logging = false
DatabaseManager.DB.<database_id>.ObjectId.CacheSize = 20
DatabaseManager.DB.<database_id>.ObjectId.MinValue = 1
```

This is the link where you can find all needed information and downloads for Informix database: Informix

Sybase

This section presents an example of a Sybase configuration, where *<database_id>* is your database identifier.

```
#-------------------------------------------------------------------
# Database Manager Configuration
#-------------------------------------------------------------------
DatabaseManager.Databases[] = "<database_id>"
DatabaseManager.DefaultDatabase = "<database_id>"
DatabaseManager.Debug = "true"
DatabaseManager.DB.<database_id>.ClassType = "Sybase"
DatabaseManager.DB.<database_id>.JdbcDriver = "com.sybase.jdbc2.jdbc.SybDriver"
DatabaseManager.DB.<database_id>.Connection.Url =
"jdbc:sybase:Tds:<hostname>.sybase.com:7100"
DatabaseManager.DB.<database_id>.Connection.User = "<name>"
DatabaseManager.DB.<database_id>.Connection.Password = "<password>"
DatabaseManager.DB.<database_id>.Connection.MaxPoolSize = "2"
DatabaseManager.DB.<database_id>.Connection.AllocationTimeout = "2"
DatabaseManager.DB.<database_id>.Connection.Logging = "true"
DatabaseManager.DB.<database_id>.Connection.MaxPreparedStatements = "2"
DatabaseManager.DB.<database_id>.ObjectId.CacheSize = 2
DatabaseManager.DB.<database_id>.ObjectId.MinValue = 1
```

This is the link where you can find all needed information and downloads for Informix database: sybase

QED

QED, the Quadcap Embeddable Database. QED is a fast, small, pure Java, relational database, implementing the SQL 92 standard, with transactions and resilient failure recovery. QED has a novel open source license permitting free use of QED by all and free redistribution in other open source projects.

```
#-------------------------------------------------------------------
# Database Manager Configuration
#-------------------------------------------------------------------
DatabaseManager.Databases[] = "<database_id>"
DatabaseManager.DefaultDatabase = "<database_id>"
DatabaseManager.Debug = "true"
DatabaseManager.DB.<database_id>.ClassType = "Sybase"
DatabaseManager.DB.<database_id>.JdbcDriver = " com.quadcap.jdbc.JdbcDriver"
DatabaseManager.DB.<database_id>.Connection.Url = " jdbc:qed:<databaseFolderPath>"
DatabaseManager.DB.<database_id>.Connection.User = "<name>"
DatabaseManager.DB.<database_id>.Connection.Password = "<password>"
DatabaseManager.DB.<database_id>.Connection.MaxPoolSize = "2"
DatabaseManager.DB.<database_id>.Connection.AllocationTimeout = "2"
DatabaseManager.DB.<database_id>.Connection.Logging = "true"
DatabaseManager.DB.<database_id>.Connection.MaxPreparedStatements = "2"
DatabaseManager.DB.<database_id>.ObjectId.CacheSize = 2
DatabaseManager.DB.<database_id>.ObjectId.MinValue = 1
```

Where <*databaseFolderPath*> is path to folder that represents QED database.

This is the link where you can find all needed information and downloads for Informix database: QED

MySQL

MySQL is an open source database that is lightweight and fast.

Note:

Prior to version 3.23, MySQL does not support transactions, and therefore does not support explicit commit (they use autocommit by default after eny SQL command). To use MySQL versions 3.22 and earlier, you have to make change to application configuration file. You will need to set parameter 'ChangeAutocommit', of logical database, to 'false' (this will disable DODS to change, database connection, autocommit property). Example:

```
DatabaseManager.DB.<database_id>.ChangeAutocommit = "true"
```

Configuration

This section presents an example of a MySQL configuration, where <*database_id*> is your database identifier.

```
#-----------------------------------------------------------------
# Database Manager Configuration
#-----------------------------------------------------------------

DatabaseManager.Databases[] = <database_id>
DatabaseManager.DefaultDatabase = <database_id>
DatabaseManager.Debug = true
DatabaseManager.DB.<database_id>.ClassType = Standard
DatabaseManager.DB.<database_id>.Connection.User = <username>
DatabaseManager.DB.<database_id>.Connection.Password = <password>
DatabaseManager.DB.<database_id>.Connection.MaxPoolSize = 5
DatabaseManager.DB.<database_id>.Connection.AllocationTimeout = 10000
DatabaseManager.DB.<database_id>.Connection.Logging = true
DatabaseManager.DB.<database_id>.ObjectId.CacheSize = 1024
DatabaseManager.DB.<database_id>.ObjectId.MinValue = 100
DatabaseManager.DB.<database_id>.JdbcDriver = org.gjt.mm.mysql.Driver
DatabaseManager.DB.<database_id>.Connection.Url =
DatabaseManager.DB.<database_id>.ChangeAutocommit = "true"

"jdbc:mysql://<hostname>:<port#>/<db_instance>"
```

This is the link where you can find all needed information and downloads for MySQL database: MySQL

PostgreSQL

Note Although other versions are available commercially, **the Together company** supports the open-source version of PostgreSQL for the Linux operating system for use with Enhydra.

PostgreSQL is a popular open-source database used with Enhydra. However, as explained in “Loading the schema” in the chapter 4, DODS requires a special column named OID 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:

```
DatabaseManager.ObjectIdColumnName = "<ColName_for_ObjectId>"
DatabaseManager.VersionColumnName = "<ColName_for_Version>"
```

where *<ColName_for_ObjectId>* and *<ColName_for_Version>* are the column names you want to use instead of OID and VERSION.

This is the link where you can find all needed information and downloads for PostgreSQL database: PostgreSQL

InstantDB

To use an InstantDB database with an Enhydra application

1 In the application configuration file *<appName>/output/conf/<appName>*.conf (or better, in *<appName>/input/conf/<appName>*.conf.in) set the following line:

```
DatabaseManager.DB.<database_id>.Connection.Url = "jdbc:idb:<propFile>.prp"
```

where *<propFile>* is the full path to the database properties file, and *<database_id>* is the database identifier used in the configuration 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 run script, in *<appName>/*run, or better in . *<appName>/*run.in.

This is the link where you can find all needed information and downloads for InstantDb database: InstantDB

Mckoi

To use an Mckoi sql database with an Enhydra application

1 In the application configuration file *<appName>/output/conf/<appName>*.conf (or better, in *<appName>/input/conf/<appName>*.conf.in) set the following line:

```
DatabaseManager.DB.<database_id>.Connection.Url = "
jdbc:mckoi:local://<confFilePath>"
```

where < confFilePath> is the full path to the database properties file, and *<database_id>* is the database identifier used in the configuration file.

2 In the same configuration file, identify the JDBC driver with the line:

```
DatabaseManager.DB.<database_id>.JdbcDriver = " com.mckoi.JDBCDriver"
```

3 Add the path to mckoidb.jar and mkjdbc.jar to the setting for CLASSPATH in the application's run script, in *<appName>/*run, or better in . *<appName>/*run.in.

This is the link where you can find all needed information and downloads for InstantDb database: Mckoi

DB2

To use an DB2 database with an Enhydra application

1 In the application configuration file *<appName>/output/conf/<appName>*.conf (or better, in *<appName>/input/conf/<appName>*.conf.in) set the following line:

```
DatabaseManager.DB.<database_id>.Connection.Url = "<url>"
```

The *<url>* argument represents a data source, and indicates what type of JDBC connectivity you are using.

For DB2 Universal JDBC Type 4 Connectivity, specify a URL of the following form:

Syntax for a URL for Universal Type 4 Connectivity:

```
>>-+-jdbc:db2:------+-//server--+-------+--/database------------>
   +-jdbc:db2j:net:-+           '-:port-'
   '-jdbc:db2j:-----'

>--+-----------------------------+-----------------------------><
   |   .-,---------------------. |
   |   V                       | |
   '-:---property--=--value--;-+-'
```

For DB2 Universal JDBC Type 2 Connectivity, specify a URL of the following form:

Syntax for a URL for Universal Type 2 Connectivity:

```
>>-jdbc:db2:database--+-----------------------------+----------><
                      |   .-,---------------------. |
                      |   V                       | |
                      '-:---property--=--value--;-+-'
```

The parts of the URL have the following meanings:

jdbc:db2: or jdbc:db2j: or jdbc:db2j:net:
jdbc:db2: indicates that the connection is to a server in the DB2 UDB family. jdbc:db2j: indicates that the connection is to a for local Cloudscape access. jdbc:db2j:net: indicates that the connection is to a remote IBM(R) Cloudscape server.

server
The domain name or IP address of the database server.

port
The TCP/IP server port number that is assigned to the database server. This is an integer between 0 and 65535. The default is 446.

database
The name of the database server.
For a connection to a DB2 UDB for Linux, UNIX(R) and Windows(R) server, the name is the database name.

For a connection to an IBM Cloudscape server, the name is the fully-qualified name of the file that contains the database. This name must be enclosed in double quotation marks ("). For example:
`"c:/databases/testdb"`

property=value;
A property for the JDBC connection. For the definitions of these properties, see Properties for the DB2 Universal JDBC Driver.

2 In the same configuration file, identify the JDBC driver with the line:

```
DatabaseManager.DB.<database_id>.JdbcDriver = ” com.ibm.db2.jcc.DB2Driver "
```

This is the link where you can find all needed information and downloads for DB2 database: DB2

HSQLDB (HypersonicSQL)

HSQLDB is a relational database engine written in Java, with a JDBC driver, supporting a rich subset of ANSI-92 SQL (BNF tree format). It offers a small (less than 160k), fast database engine which offers both in memory and disk based tables. Embedded and server modes are available. Additionally, it includes tools such as a minimal web server, in-memory query and management tools (can be run as applets) and a number of demonstration examples.

To use an HSQLDB database with an Enhydra application

1 In the application configuration file *<appName>/output/conf/<appName>*.conf (or better, in *<appName>/input/conf/<appName>*.conf.in) set the following line:

```
DatabaseManager.DB.<database_id>.Connection.Url =
“jdbc:hsqldb:hsql://<hostName>:<port>”
```

where *<hostName>* is the host name or IP adress of computer with runing database server, and *<port>* is port where database server wait for request (default: 9001).

2 In the same configuration file, identify the JDBC driver with the line:

```
DatabaseManager.DB.<database_id>.JdbcDriver = ”org.hsqldb.jdbcDriver "
```

3 Add the path to hsqldb.jar to the setting for CLASSPATH in the application's run script, in *<appName>/*run, or better in . *<appName>/*run.in.

This is the link where you can find all needed information and downloads for HSQLDB database: HSQLDB

Microsoft SQL Server

The exact configuration settings for connecting to MS SQL server depend on the JDBC driver you are using. We do not recommend using the JDBC-ODBC bridge with MS SQL Server.

This is the link where you can find all needed information and downloads for MSQL database: MSQL

JTurbo JDBC driver

We certified the JTurbo 2.0 JDBC driver, and the configuration settings for this are:

```
# JTurbo 2.0 JDBC Driver for MS SQL server
DatabaseManager.Databases [] = "my_db"
DatabaseManager.DefaultDatabase = "my_db"
DatabaseManager.DB.my_db.ClassType = "Standard"
DatabaseManager.DB.my_db.JdbcDriver = "com.inet.tds.TdsDriver"
# NOTE: substitute your server's IP address for 10.0.0.18 below
# Substitute the port your DB is listening on for 1433 below
DatabaseManager.DB.my_db.Connection.Url =
"jdbc:inetdae:10.0.0.18:1433?database=my_db"
DatabaseManager.DB.my_db.Connection.User = "<user_name>"
DatabaseManager.DB.my_db.Connection.Password = "<password>"
```

If you are using another JDBC driver, you need to determine the driver package, for the DatabaseManager.DB.my_db.JdbcDriver setting, and connection string, for DatabaseManager.DB.my_db.Connection.Url setting.

MS-JDBC driver

Configuration settings example for MS-JDBC driver are:

```
DatabaseManager.DB.my_db.JdbcDriver = "
com.microsoft.jdbc.sqlserver.SQLServerDriver"
# NOTE: substitute your server's IP address (hostname)
# Substitute the port your DB is listening on for (default: 1433)
DatabaseManager.DB.my_db.Connection.Url
 = " jdbc:microsoft:sqlserver://<hostname>:<port>; DatabaseName=
<databaseName>;SelectMethod=cursor "
DatabaseManager.DB.my_db.Connection.User = "<user_name>"
DatabaseManager.DB.my_db.Connection.Password = "<password>"
```

If you are using another JDBC driver, you need to determine the driver package, for the DatabaseManager.DB.my_db.JdbcDriver setting, and connection string, for DatabaseManager.DB.my_db.Connection.Url setting.

Microsoft Access

Microsoft Access is not a true SQL database server; as such, it is suitable for development and testing, but not for a production database. Access does not have a JDBC driver. However, Access does support ODBC, and there is a JDBC-ODBC bridge in the Sun JDK, which enables Access to work with Enhydra.

Because Access cannot read-in files containing SQL commands, you must create tables in the Access GUI. For the DiscRack example, you can also use the Access database provided in *<dods_home>/*examples/DiscRack/discRack.mdb.

To use Enhydra with Access:

1 Register the database as an ODBC data source:

1 Go to Start|Settings|Control Panel and click ODBC Data Sources.

2 Click the Add button in the dialog box that comes up.

3 Select the Microsoft Access Driver in the Create New Datasource dialog box and click Finish.

The ODBC Microsoft Access Setup dialog box appears.

4 Choose a name, like discRack, for the Data Source Name. Under Database, click the Select button, browse to the *.mdb file, select it, and click OK.

2 Place database information in the application’s configuration file, as shown in the example below. Replace *<data_source>* with the name you chose for Data Source Name in the preceding step.

Note You don’t have to place the JDBC driver in the application’s CLASSPATH because the ODBC/JDBC bridge is in the JDK and thus is already in the system’s CLASSPATH.

This section presents an example of an Access configuration, where *<database_id>* is your database identifier.

```
#-------------------------------------------------------------------
# Database Manager Configuration
#-------------------------------------------------------------------
DatabaseManager.Databases[] = "<database_id>"
DatabaseManager.DefaultDatabase = "<database_id>"
DatabaseManager.Debug = "false"
DatabaseManager.DB.<database_id>.ClassType = "Standard"
DatabaseManager.DB.<database_id>.JdbcDriver = "sun.jdbc.odbc.JdbcOdbcDriver"
DatabaseManager.DB.<database_id>.Connection.Url = "jdbc:odbc:<data_source>"
DatabaseManager.DB.<database_id>.Connection.User = "Admin"
DatabaseManager.DB.<database_id>.Connection.Password = ""
DatabaseManager.DB.<database_id>.Connection.MaxPreparedStatements = 10
DatabaseManager.DB.<database_id>.Connection.MaxPoolSize = 30
DatabaseManager.DB.<database_id>.Connection.AllocationTimeout = 10000
DatabaseManager.DB.<database_id>.Connection.Logging = false
DatabaseManager.DB.<database_id>.ObjectId.CacheSize = 20
DatabaseManager.DB.<database_id>.ObjectId.MinValue = 1
```

This is the link where you can find all needed information for Microsoft Access: Microsoft Acces

InterBase

InterBase® is an efficient and powerful RDBMS engine. Its vendor, Borland/Inprise, has released InterBase version 6.0 as an open-source product. See http://www.interbase.com for more information and product downloads.

InterClient

The JDBC driver for InterBase is called InterClient™ The InterClient system includes an all-Java thin client, and a server-side daemon (also known as a service on Microsoft Windows NT) called InterServer. This daemon accepts JDBC connection requests and in turn connects to the InterBase RDBMS daemon. The three processes (JDBC client, InterServer daemon, InterBase daemon) can run all on separate hosts, all on the same host, or in any other combination.

InterClient is a class 3 JDBC driver in that it has a separate daemon on the server to serve JDBC connections; however, it also matches the definition of a class 4 driver because the client component can connect only to one DBMS back-end, InterBase.

InterClient is installed separately from InterBase. On Windows, InterClient is commonly installed in:

```
C:\Program Files\Borland\InterClient\interclient.jar
```

Depending on the version of InterClient, it might instead be installed in:

```
C:\Program Files\InterBase Corp\InterClient\interclient.jar
```

Find the JAR file and append its location to your system CLASSPATH environment variable on the client host where you run Java applications.

Different versions of InterClient are available.

- InterClient version 1.50x works only with JDK 1.1x.
- InterClient version 1.51x works only with JDK 1.2.x.

Whichever version of InterClient you use, you must use the matching version of InterServer.

Configuration

You need to configure both the dods.conf and your <*application*>.conf to support InterClient.

DODS configuration

You should apply the following configuration edits to dods.conf to make the Standard_JDBC database class match InterBase features. This is necessary because there is not yet a specific com.lutris.appserver.server.sql.interbase package in the Enhydra sources.

```
Database.OidDbType.Standard_JDBC= "DECIMAL(9,0)"
Database.BitType.Standard_JDBC= "SMALLINT"
Database.TimeType.Standard_JDBC= "DATE"
Database.TimestampType.Standard_JDBC= "DATE"
Database.OnCascadeDelete.Standard_JDBC= true
Database.StringQuoteCharacter.Standard_JDBC= '
Database.StringMatch.Standard_JDBC= "LIKE"
Database.StringWildcard.Standard_JDBC= "%"
```

Application configuration

This section presents an example of an Interbase configuration, where <*database_id*> is your database identifier.

```
#--------------------------------------------------------------------
# Database Manager Configuration
# InterBase / InterClient
#--------------------------------------------------------------------
DatabaseManager.Databases[] = "<database_id>"
DatabaseManager.DefaultDatabase = "<database_id>"
DatabaseManager.Debug = "false"
DatabaseManager.DB.<database_id>.ClassType = "Standard"
DatabaseManager.DB.<database_id>.JdbcDriver = "interbase.interclient.Driver"
DatabaseManager.DB.<database_id>.Connection.Url =
"jdbc:interbase://loopback/<path_to_database>"
DatabaseManager.DB.<database_id>.Connection.User = "sysdba"
DatabaseManager.DB.<database_id>.Connection.Password = "masterkey"
```

Configuration notes

The JDBC driver class is interbase.interclient.Driver.

Server name

The general URL format for InterClient JDBC connections is as follows:

```
jdbc:interbase://servername/<path_to_database>
```

where <*path_to_database*> is the full path to the database file, including the name of the database (for example, /usr/local/data/inventory.gdb).

The *servername* is the hostname or IP address of the server running InterServer, the server-side daemon that accepts JDBC connection requests. If your Enhydra application runs on the same host where InterServer runs, you can use the special servername loopback.

Pathnames

The <*path_to_database*> is an absolute path to the InterBase database file on the server where the InterBase RDBMS server runs. InterBase does not have abstract handles to databases, like some database products do (for example, Oracle SIDs or BDE aliases). You must specify the real path to the database. You cannot use mapped drives or NFS filesystems in this path.

Notice the literal slash character (/) following the server name. If the absolute path starts with a slash character (/), then you should have a pair of slash characters (//) together. For example:

```
jdbc:interbase://servername//usr/local/data/inventory.gdb
```

If the server is a Windows host, the path starts with a drive letter identifier:

```
jdbc:interbase://servername/C:/data/inventory.gdb
```

If InterServer runs on a different host than the InterBase RDBMS server, you must specify this host in the path to database, with the following syntax:

```
jdbc:interbase://<interserver_host>/<interbase_host>:<path_to_database>
```

Tip Slash (/) and backslash (\) characters within path names are interchangeable to InterBase; the InterBase daemon translates these characters as needed to match the convention on the server platform. It is easier to use slashes in code, however, because escape sequences are required to represent backslashes in code.

Ports

InterBase does not take a port number argument in connection strings. InterClient and InterServer always communicate using the TCP/IP service named interserver, which defaults to port 3060. InterServer and InterBase always communicate using the TCP/IP service named gds_db , which defaults to port 3050. These services and port numbers are registered with IANA.

Username and password

The username sysdba and its default password masterkey are used in the example configuration above, but for security reasons it is recommended that you: (a) change the default sysdba password on your InterBase server, and (b) create a non-superuser login in the InterBase password database, and use that login for general database access.

C-JDBC

C-JDBC is a database cluster middleware that allows any Java application (standalone application, servlet or EJB container, ...) to transparently access a cluster of databases through JDBC. You do not have to modify client applications, application servers or database server software. You just have to ensure that all database accesses are performed through JDBC.
First you will need to install C-JDBC. The easiest way to install C-JDBC is to use the Java graphical installer. A Java Virtual Machine is of course needed in this case. Simply launch the installation program by typing:

```
java -jar c-jdbc-x.y.bin-installer.jar
```

(Check `CJDBC_HOME` environment variable)

Once you have installed the C-JDBC controller, you will find the driver JAR in the drivers/ directory of the controller installation location.

To install the C-JDBC driver, you just have to add the c-jdbc-driver.jar to the client application classpath. This driver replaces the database native driver in the client application.

The database native driver will be used by the C-JDBC controller to access your database. Therefore, the C-JDBC driver and controller can be seen as a proxy between your application and your database native driver.

DODS uses C-JDBC as any standard JDBC driver.

In the application configuration file *<appName>/output/conf/<appName>*.conf (or better, in *<appName>/input/conf/<appName>*.conf.in) identify the JDBC driver with the line:

```
DatabaseManager.DB.<database_id>.JdbcDriver=”org.objectweb.cjdbc.driver.Driver"
```

In the same configuration file, identify the database with the database URL .

The JDBC URL expected for the use with C-JDBC is the following:

```
jdbc:cjdbc://host1:port1,host2:port2/database.
```

Host is the machine name (or IP address) where the C-JDBC controller is running, port is the port where the controller is bound on this host.

At least one host must be specified but a list of comma separated hosts can be specified.
If several hosts are given, one is picked up randomly from the list. If the currently selected controller fails, another one is automatically picked up from the list.

Default port number is 25322 (C-JDBC on the phone !) if omitted. Those two URL are equivalent:

```
"jdbc:cjdbc://localhost:/tpcw"
"jdbc:cjdbc://localhost:25322/tpcw"
```

So set URL in conf file to:

```
DatabaseManager.DB.<database_id>.Connection.Url = " jdbc:cjdbc://<host>:<port>"
```

Example:

```
#-------------------------------------------------------------------
# Database Manager Configuration
#-------------------------------------------------------------------
DatabaseManager.Databases[] = "<database_id>"
DatabaseManager.DefaultDatabase = "<database_id>"
DatabaseManager.DB.<database_id>.ClassType = "Standard"
DatabaseManager.DB.<database_id>.JdbcDriver = =”org.objectweb.cjdbc.driver.Driver"
DatabaseManager.DB.<database_id>.Connection.Url = " jdbc:cjdbc://<host>:<port>"
DatabaseManager.DB.<database_id>.Connection.User = "<name>"
DatabaseManager.DB.<database_id>.Connection.Password = "<password>"
DatabaseManager.DB.<database_id>.Connection.MaxPoolSize = "2"
DatabaseManager.DB.<database_id>.Connection.AllocationTimeout = "2"
DatabaseManager.DB.<database_id>.Connection.Logging = "true"
DatabaseManager.DB.<database_id>.ObjectId.CacheSize = 2
DatabaseManager.DB.<database_id>.ObjectId.MinValue = 1
```

Details of creating, configuring and starting C-JDBC database is out of scope of this document. All details about these features can be found on C-JDBC ObjectWeb site

C-JDBC is a *free*, *open source* ObjectWeb Consortium (http://www.objectweb.org/)'s project. It is licensed under the GNU Lesser General Public License

---