Building a WML Application with Enhydra

Ryan Fife, Anywhereyougo.com

The first question that you may have is, "What is Enhydra?" Enhydra is an open source application server written entirely in Java. It supports many standards, including XML, HTML, Java Servlets, and JavaServer Pages (JSP).

Lutris Technologies, an Internet consulting company based in California, started developing Enhydra over four years ago because there were no application servers on the market that could meet their needs. Enhydra is built with Java to make it deployable on whatever platform best suits your needs. Lutris was not in the business of selling application servers, so they released Enhydra under an open source license to allow others to benefit from their efforts.

Today, Enhydra has grown into a robust Java application server that is not only feature rich, but also has a diverse group of loyal users and developers. Enhydra remains true to its roots, and is still able to be developed with and deployed on various platforms and web serving environments. It is now more alive than ever, with all the work done to support the latest standards in web development as well as integrating the work of other open source projects such as Xerces, Castor, JSP, Tomcat, JTidy, and so on. In addition to the advantages of the "behind the scenes" technology, Enhydra offers many advantages for the Internet developer:

- Familiar servlet interface.
- Integrated web server to speed the development testing lifecycle.
- Web-based administration tool for servlet control and debugging.
- W3C-DOM based view of HTML and XML files for easier manipulation and separation of business logic and presentation code.
- Built-in logging facilities with a configurable level of output to log files.
- The source code! If you have a question, the code is *the* authoritative answer.

In this talk, we will be covering how Enhydra works with some of the technologies discussed in the next section, and assume you have at least a working knowledge of some of them (Java, WML, HTML, and servlets). Today we will be covering:

- Installation requirements of Enhydra for development
  - Compatible with ANY platform with JDK 1.1! (Linux, Solaris, Windows 2000, etc.)
- The basic structure of an Enhydra application
- How to develop an Enhydra application that can publish to both WML and HTML

Enhydra's Limitations

Enhydra can help developers of Internet based applications in a variety of ways, but it does not pretend to do everything. Things that Enhydra is not:

- An Integrated Development Environment (IDE). Enhydra does not include its own Java, HTML, or WML editors. It is, however, capable of integrating with some IDEs. The Kelp Working Group is adding functionality towards this goal. (http://www.enhydra.org/community/workingGroups/Kelp.html) Currently supported IDEs are JBuilder and Oracle JDeveloper.

- A Java compiler. JDK 1.1 or higher must be installed to develop and run Enhydra applications, and it is recommended to have 1.2 installed, as the 1.1 family does feature some bugs that can cause problems in the administration program.
- A point and click solution. Enhydra gives you the tools and framework to build robust applications — it is up to you to architect and build those solutions.

You may be wondering what limitations Enhydra puts on your development. I know that many of the application servers I have worked with prior to my Enhydra experience limit you in a variety of ways, and I won't lie to you and say that Enhydra doesn't do this in some manner, but I will say that it does it less than most of them. The fact that Enhydra is open source does not detract from its capabilities or stability.

Enhydra has all of the features of other major application servers I've used, and then some. The fact that there are no licensing fees charged for the software allows you to try it out and see for yourself without any monetary commitment. The largest advantage Enhydra has over its closed source brethren is the ease with which you can determine whether a problem with your application is due to your code or that of Enhydra. The advantage for the rest of the world is that if you do find a problem with Enhydra and submit the fix back to the community, others won't have the same problems you did. Remember, this process often works to your advantage as well.

Doing Things the Enhydra Way

Saying that Enhydra promotes a way of doing things would be better than saying it makes you do things a certain way. The Enhydra tools are built to use a 3-tier architecture by default that separates your presentation, business, and data classes. However, the default setup is just there to help you get started: you can build a client-server type application with Enhydra if you wish. I often build all my data, business, and presentation logic all in the presentation layer for prototyping, and then figure out the best way to separate them cleanly before building my final product. I highly recommend that you follow the 3-tier architecture in your final product, especially if you plan to publish to multiple platforms.

Enhydra also suggests that you follow certain standards. The XMLC tool uses the Java port of Tidy (JTidy) as the default parser, and will notify you if you don't close tags properly or use unsupported tags. These are only warnings, though, and they can be ignored if you don't mind the messages. One thing to note is that the parser will automatically create "correct" HTML for output. This can cause some difficulty with complex layouts that rely on invalid HTML for proper placement of page items. In cases where adherence to the standards is required, such as writing WML or XML, Enhydra can force you to follow the proper markup language and prevent a lot of headaches during development.

Now that the high-level introduction to Enhydra is taken care of, let's talk about what you need to know to use it effectively. Enhydra uses the UNIX philosophy of building upon multiple pieces that all do their job well, so the list of things to know is relatively long. A comforting thought to consider is that most of these will at least be familiar to a person developing for the Web and/or WAP anyway:

- Java 1.1
- XML (WML, XHTML)
- HTML
- W3C Document Object Model (DOM) — http://www.w3c.org/
- Java servlets — not essential, as the framework takes care of this part anyway, but knowing how they work is a great help.

Installing Enhydra

We'll take a high-level look at installing Enhydra first. To install Enhydra for development, you will need the following tools:

- Java Developer Kit for version 1.1 of the language
- GNU make environment:
  - Requires basic UNIX tools such as sed and awk (free GNU tools available)
  - Can use CYGWIN under Win32 environments
  - Any other platform with support for GNU make
- Java and markup language editor(s) of your choice:
  - XEmacs supports this environment nicely
  - Kelp project helps integrate with popular IDEs, such as JBuilder Foundation (Standard, Professional, or Enterprise) and Oracle JDeveloper

Enhydra includes a built in web server that is useful for debugging code during the development cycle, but you need to use an external web server for deployment. The web servers that Enhydra is known to work with include:

- Apache, through Apache Module Support or the JServ environment
- Netscape server/iPlanet support through NSAPI
- Microsoft IIS support through ISAPI
- Any web server with a servlet runner
- Any web server with CGI support (with obvious performance trade-offs)

Another piece worthy of mention is Enhydra Director, enabled by the Module Support listed above. This product enables enterprise level features such as load balancing between multiple instances, and failover for downed machines. Enhydra Director's goal is to let both Enhydra and the complementary web server concentrate on their primary functions. It accomplishes this by using web server specific APIs to listen for files that Enhydra knows how to manipulate. If the web server is trying to serve a `.po` file, Enhydra will take the request, run the servlet, then send the resulting file to the server. If the web server gets a request for a GIF image, Enhydra quietly ignores this and lets the web server take care of fulfilling that request.

Database support is an interesting topic. Enhydra contains a relational-to-object mapping tool called Data Object Design Studio (DODS), further information about which can be found at http://www.enhydra.org/software/documentation/enhydra/DODS.html DODS is a useful tool that supports many databases, including:

- Informix
- Microsoft SQL
- Oracle
- PostgreSQL
- Sybase
- Most databases with standard JDBC support

Of course, you can access your database using any tool you'd like. You can use a commercial object-to-relational mapping tool, or talk to JDBC directly and process the result sets yourself. If you choose to talk to JDBC directly, you can use any database with JDBC drivers available on your development and deployment platforms. You can also achieve the same thing with ODBC databases using the standard `odbc:jdbc` bridge.

Now that you know what you need and what you're going to get, you need to know where to get it. Go to http://www.enhydra.org for the latest version and information about the product. You have one more decision though: do you get a source release or a binary distribution? There are a few guidelines to use when making this decision:

Source Code

- You need access to immature features
- You will be adding new features to Enhydra
- You just like to have the latest version of the software, and it is useful to be able to correct bugs that you find in the engine and not have to wait for a service pack/update release.

Binary Release

- Enhydra currently supports everything you want to accomplish
- You don't want to compile the source code
- You feel more confident about officially sanctioned releases

Either way, I recommend at least downloading the source, just in case you decide to look through it.

> *This is definitely not a thorough explanation of installing Enhydra, so the following URLs will provide further help (as well as being interesting!) if you need it:*
>
> *http://www.enhydra.org/software/documentation/enhydra/index.html*
> *http://www.enhydra.org/home/faq/cache/1.html*
>
> *If you'll be using the product, you should also sign up for the Enhydra mailing list (http://www.enhydra.org/community/mailingLists/index.html). There are a variety of people from around the world on the list, and they are all achieving a fascinating array of different things with Enhydra. There is a really good chance that somebody will be able to answer just about any question you may have. In a way, the community is more than just a list — it's the main technical support center, and also features a searchable archive.*

Using Enhydra

Now that we've got the background out of the way, we can move on to actually using Enhydra. I'll discuss some things that help newcomers to Enhydra, as well as seasoned veteran users. The first thing we'll take a look at is the newapp script, which sets up a basic directory structure, the default build environment, and some Enhydra configurations that you won't need to worry about when writing your first application.

You will definitely modify these files later, but you don't need to worry about them right now. I'm going to go over building your first application, and then we'll switch over to a running system and run through the demonstration for real. This way, you won't be lost when I run through things. A typical application development cycle looks something like this:

- Write out technical documentation for the project, including drafts of the code. This helps greatly to keep on top of where you are, and not forget things.
- Run newapp to get the default tree created
- Modify the business, presentation, and data objects to suit your project

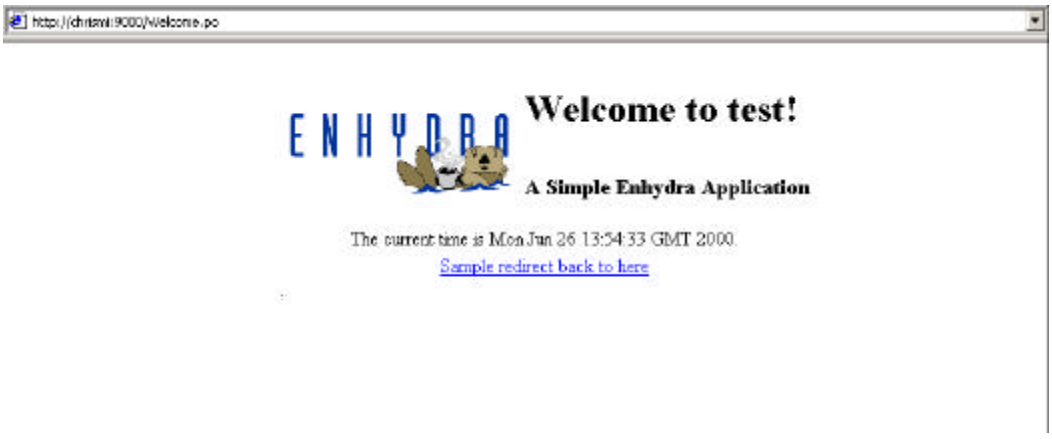
- Modify the makefile(s) to include your new classes
- Build and test
- Repeat

Newapp/Makefiles

After running newapp, the directory structure looks like this:

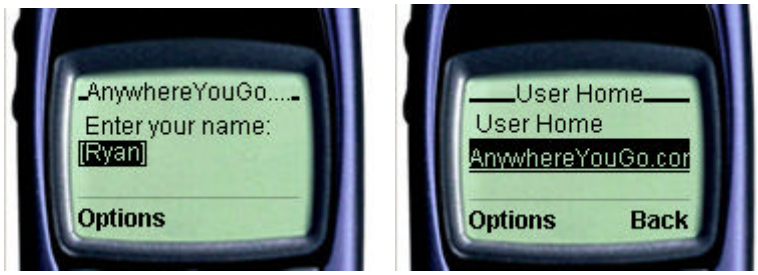
The newapp command creates a working Enhydra application (in this case, `intro`) that you can build and run right away. The resulting application demonstrates some basic Enhydra patterns: the default directory structure, naming conventions for presentation objects and HTML files, and how to run and test your application locally. Notice that there are business, data, and presentation directories created by default — 3-tier code is promoted from the beginning.

The newapp command also creates makefiles for you — anyone who uses makefiles knows how powerful and painful they can be, so automation in this area is greatly appreciated. Adding new objects to your application requires a basic understanding of makefiles, and usually involves only minor changes. To start the program, we change to the output directory and run the `start` command. It then starts up and tells us what host and port to connect to. The default application looks like this:



We're going to modify the default application to use WML, since that's what we're really interested in accomplishing here. I have two WML files that we are going to bring into the Enhydra build and use to write a simple application that asks the user for their name. If it is a recognized name, we append a link to their homepage to the response; otherwise they get the generic welcome message.

The WML screens look like this:



The default build environment does not include WML build rules. I have made some rules that you can add to the default `config.mk` file after the include directive at the end of the file, to make building WML files easier. The lines to add to `config.mk` are:

```
$(PACKAGE_OUTPUT)/%WML.class: $(WML_DIR)/%.wml $(XMLC_WML_OPTS_FILE)
$(XMLC_%_OPTS_FILE)
   @mkdir -p $(PACKAGE_OUTPUT)
ifeq ($(XMLC_AUTO_COMP),YES)
   cp -f $(WML_DIR)/$*.wml $(PACKAGE_OUTPUT)
endif
   @CLASSPATH="$(ENHYDRA_CLASSPATH)" ; export CLASSPATH ; \
   set -x ; \
   $(XMLC_CMD) -class $(PACKAGE).$*WML $(XMLC_WML_OPTS) $(XMLC_$*_OPTS)
$(XMLC_JAVAC) $(XMLC_WML_OPTS_FILE) $(XMLC_$*_OPTS_FILE) $(WML_DIR)/$*.wml

$(PACKAGE_OUTPUT)/%WML.class: %.wml $(XMLC_WML_OPTS_FILE)
   @mkdir -p $(PACKAGE_OUTPUT)
   @CLASSPATH="$(ENHYDRA_CLASSPATH)" ; export CLASSPATH ; \
   set -x ; \
   $(XMLC_CMD) -class $(PACKAGE).$*WML -d $(PACKAGE_OUTPUT) $(XMLC_WML_OPTS)
$(XMLC_$*_OPTS) $(XMLC_JAVAC) $(XMLC_WML_OPTS_FILE) $*.wml

do_xmlc_html_targets:: $(WML_CLASSES:%=${PACKAGE_OUTPUT}/%.class)
```

XMLC

XMLC stands for "XML Compiler". It converts text-based markup, such as HTML and WML, into objects that conform to the W3C Document Object Model (W3C-DOM). At least, that's the official explanation. The simpler version is that XMLC is a program that converts XML-based content into trees of Java objects that can be manipulated in an Enhydra application. This is the preferred method of manipulating WML, HTML, and XML files in Enhydra.

Once you have the makefile rules in place, you can put the WML files into a `wml` directory under the presentation directory, and add them to the build with the following rules:

```
XMLC_WML_DIR = ./wml
XMLC_WML_CLASSES = InputWML ResultWML
```

We then run `make`, and see that XMLC is invoked to compile our WML files into DOM trees with Java bindings. Let's look a little closer at what XMLC is doing.

XMLC is not a trivial program. XMLC translates your XML-based markup into Java objects by parsing the HTML and creating a DOM tree that represents the markup. The end result is that you manipulate a tree of objects, which is efficient in Java — the alternative being `String` manipulation, which is inefficient in Java. (XMLC is totally object-oriented, which is not the case with string manipulation.)

I have to take some time to talk about the Document Object Model, since it's likely to confuse someone who is not familiar with it. This comes straight from W3C:

> *"The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to access and update the content, structure and style of documents dynamically. The document can be further processed and the results of that processing can be incorporated back into the presented page. This is an overview of DOM-related materials here at W3C and around the Web."*
>
> *Further information can be found at http://www.w3.org/DOM/*

XMLC supports some useful command line parameters that allow us to understand what it is doing without viewing the generated source code. Some of the more useful ones follow:

- The `–dump` flag shows you the generated DOM tree structure on standard output.
- The `–methods` flag shows you the convenience methods that XMLC generated for this file. These methods use the `id` attribute in the markup to generate a method of the form `getElement`*`NamedIdAtrribute`*`()` For example, if you had a tag in your markup like `<a id="linkName" href="nextpage.wml">`, XMLC would generate a convenience method called `getElementLinkName()`.
- The `–nocompile` flag tells XMLC not to actually compile the output, but to parse the file only. It is used in conjunction with other flags.

Looking at our `input.wml` file, with all of the flags shown above, we see how XMLC interprets our DOM structure and what convenience methods it creates for us:

```
xmlc -dump -nocompile Input.wml

DOM hierarchy:
  Document:
    Document type =>
      DocumentType: name=wml
        Entities =>
        Notations =>
        ElementNS: wml
        Element: head
        Element: meta: content='max-age=0' forua='true' http-equiv='Cache-Control'
        Element: card: id='home' title='AnywhereYouGo.com'
        Element: p: id='title'
          Text:
            Enter your name:
        Element: br
          Text:
        Element: input: emptyok='false'
                        id='iName' name='iName' title='name' type='text' value=''
          Text:
        Element: do: id='sendName' label='Login' type='accept'
        Element: go: href='/WAPDemo.po' method='post'
        Element: postfield: name='name' value='$iName'
        Element: postfield: name='postedTo' value='true'
```

Let's build our servlet that displays the page. In the language of Enhydra, this is called a presentation object (PO). The PO is responsible for creating the user interface based on the HTTP request given it, just like a servlet or a CGI program. This Java class must implement an Enhydra interface that holds a "run" method. Enhydra supplies an object to this method that allows you to access the request and response objects. These objects encapsulate things like HTTP headers, HTTP post parameters, referrer URL, and other variables that are familiar to web developers. Our skeleton presentation object doesn't do anything — it just implements the correct interface, which we now have to fill in and give a purpose.

Applying our Work

Now we need to make it do something useful. We will read the name parameter from the request — we don't care if this information was posted to us, or came from the query string in a get request (as in the GET and POST methods of HTML), we read it the same way. The syntax is as follows:

```
String name = comms.request.getParameter("name");
```

We can use the convenience method we saw XMLC generate for us to grab the text message from the WML template file. We pass this node and the name into our business class for additional processing (we'll get to the details of this in a minute), and then print out the document to a string so we can send it to the browser.

The business object is just a regular Java class, and I've already created the business object skeleton to save time — all we need to do now is fill it in (this filling can also be done in PO, as no business layer is provided). Note that since we are dealing with pass-by-reference, anything we do to the node that is passed in will be reflected in the document the display class sends out. First, we check if the name is usable — if we have a null or an empty string, there is no point in doing any further processing. Next, we check to see if the name is a key in our knownIdentities dictionary — if we know who it is, we create a link to their site.

This is done with the Document.createElement() method. We want to create a link that looks like somesite. The code to do this is shown below — we basically ask the document to create an element based on a tag name. At this stage, I'd really like to reiterate that getting the documentation for the W3C DOM is a good idea if you are going to use XMLC and Enhydra; the resulting code is:

```
    // See if we recognize the user
    if (validUsers.containsKey (user)) {
      // Get the owner doc so we can create elements
      Document doc = refLink.getOwnerDocument();

      // Create an "a" element for our link.  Set the href attribute
      // so the device knows where to go
      Element link = doc.createElement ("a");
      String linkName = (String) validUsers.get (user);
      String url = (String) namedLinks.get (linkName);
      link.setAttribute ("href", url);

      // We have to append a text node to the link so the user sees
      // something.  This is the text between the open and close of
      // the a tag:
      // <a href="somelink">text node we are creating</a>
      Text linkText = doc.createTextNode(linkName);
      link.appendChild (linkText);

      // Append the link to the document
      Node parent = refLink.getParentNode();
      parent.appendChild (link);

      // Because of the way the dom works - if you append a child that
      // is already in the child list, it will be removed first - this
      // could be done in multiple steps, but it is not necessary.
      parent.appendChild (doc.createElement("br"));
      parent.appendChild (refLink);
    }
```

I don't show it in the code for this tutorial, but a good way to build documents is to grab pieces from multiple documents and merge them into one final document that is sent to the browser. As an example, let's say you had a file for your home page that had HTML like this:

```
<table>
   <tr> <td id="featureSection"> This is the features section </td> </tr>
</table>
```

Then you have a "features" file that contains items you would like to feature on your home page:

```
<table id="features">
   <tr><td> New WAP emulator available </td></tr>
   <tr><td> AnywhereYouGo.com releases on-line WAP testing tool </td></tr>
</table>
```

We then take the features table and insert it into our home page. The process requires an extra step, though: you have to import the content you want to put from the features document into your home page document — you use the `importNode()` method to accomplish this. This (for me, at least) is the most common way to build documents with Enhydra. The following code imports and appends a node from another document:

```
// Get our documents
HomeHTML home = new HomeHTML();
Document doc = home.getDocument();
FeaturesHTML features = new FeaturesHTML();

// Grab the insertion point and node we want to import
Node insertionPoint = home.getElementFeaturesSection();
Node featureTable = features.getElementFeatures();

// Import the features table and replace the current content in the template
insertionPoint.appendChild (doc.importNode (featureTable));
```

Testing our Creation

Now we can run the code and see what happens on the emulator. This is the part where everyone in the audience with a WAP phone can participate! If you go to http://demo.ayg.com, you should see the demo we just created. I have added some business logic that will tell 5 people that they won. If you get that message, show me your phone and I'll give you an AnywhereYouGo.com T-shirt. The code for the demonstration we built, and the modified one, is on the Anywhereyougo.com site at http://www.AnywhereYouGo.com/wrox2000/

The next part of the demonstration has less to do with WML than it does with HTML, but it's a good example of how easy it is to publish to multiple markup languages with Enhydra (using XML as a middle step). I already have the HTML built, and it looks similar to the WML version.

`input.html`:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<html>
  <head>
    <title>Welcome to intro!</title>
    <!-- Changed by: Ryan Fife, 30-Apr-2000 -->
  </head>
  <body bgcolor="#FFFFFF">
    <form action="WebDemo.po" method="post">
      <p id="title">
        Enter your name:<br>
        <input name="name" value="" type="text" id="iName">
      </p>
      <p>
        <input type="submit" value="Log In" align="middle">
      </p>
      <input type="hidden" name="postedTo" value="true">
    </form>
  </body>
</html>
```

response.html

```
<html>
  <head>
    <title> Enhydra Demo </title>
    <!-- Changed by: Ryan Fife, 30-Apr-2000 -->
  </head>
  <body bgcolor="#ffffff">
    <p id="userTitle">
      User Home
    </p>

    <p id="userLinks">
      <a id="AYGLink" href="http://www.ayg.com/">AnywhereYouGo.com</a>
    </p>
  </body>
</html>
```

We can now copy the `WAPDemo.java` file to `WebDemo.java`, and edit it to display HTML. All we have to do is change the document we instantiate for the input and result pages; everything else stays the same. Let's take another look at the business class to see why that is.

When we create the new node with the link and append it after the message node, we used the document to create a named element. Since a link is created with an <a> tag in both WML and HTML, the document created the appropriate tag for the appropriate document type — we don't have to change anything. We have made some assumptions about tags that exist in both presentation types, but as long as you can do that, you can write business classes that will work with either front end.

Now you can update the business class, and both the HTML and WML versions get the changes. If you add more names you recognize, they will be recognized no matter how the user comes in. With the use of a database on the backend and some more business classes, you can see how you could build an interesting Internet application using Enhydra.

To Conclude

Let's review quickly what we've covered:

- Basic Enhydra structure and layout
- XMLC usage, and some useful flags
- Markup language independent business object creation, including multi-device output.

There are some pieces of Enhydra that we didn't cover — I mention them here only for completeness. Check out the Enhydra web site for more information:

- Data Object Design Studio (DODS): This is the object-to-relational mapping tool that Enhydra uses to make developing applications easier. It supports most major databases, and automatically handles connection pooling and some data caching.
- Multiserver Admin: A GUI application that controls applications within Enhydra. You can use it to control multiple Enhydra applications on one machine, step through requests for debugging, and start and stop Enhydra applications. It's really useful for large-scale deployment of Enhydra applications.
- Enhydra Working Groups: The working groups are project teams comprised of Enhydra developers that build specific areas of functionality into Enhydra. This includes projects to integrate Enhydra into IDEs, internationalization efforts, Enterprise JavaBeans, and various other enterprise-level efforts.

In conclusion, Enhydra conforms to standards and promotes good programming practices that help you to develop and publish your site to a variety of platforms with very little trouble. It is open source, and builds upon other open source projects such as Xerces, Castor, JTidy, and Tomcat, and focuses on doing what it takes to develop a solid, enterprise-level application for the Internet world of today. The most important considerations for this are:

- Scalability: performance and ability to handle greater load without reconfiguring the application, just by adding the appropriate hardware when needed.
- Stability (robustness): making sure that the application is stable and (relatively) bug free in the first place.
- Usability: the application needs to cater for as many different device configurations as possible, taking into account all of the difference in set-up you can get, what with all the different platforms and devices available.