XMLC Automatic Recompilation and Automatic Reloading

Table of Contents

Chapter 1. Introduction

Automatic XMLC re-compilation enables you to change the HTML 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, recompiles the page, and uses it in the application. Configuring autorecompilation to work is also simple once you understand the details behind it.

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 Automatic Recompilation 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. The difference between these two approaches is that in XMLC Automatic Recompilation the changed HTML pages are automatically compiled in runtime, while before performing the Automatic Reloading, it is necessary to compile the changed application, and then perform exchange of the existing application (or its parts) with new one(s).

Chapter 2. Summary

- Use xmlcFactory to instantiate pages in your Java code, i.e.

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

  The creation of Java objects with DOM presentation of HTML pages in presentation objects, in the case of the Automatic Recompilation and Automatic Reloading must be done only by create method of the xmlcFactory class (like in the example above). The creation of instances with the Java key word new must not be performed. This way of creating instances (with new) can be used in applications in which Automatic Recompilation and Automatic Reloading are not used.

- In app.conf set

  ```
  Server.ClassPath[] = "@DEPLOY_PATH@/../classes"
  ```

  The Web application with performed Automatic Recompilation or Automatic Reloading is not started from jar file, and parameter above must contain the path of the application's classes root directory (application's classes can be placed in different root directory from mentioned, and then the correction of the root path is needed). The path can be absolute (without @DEPLOY_PATH@ parameter).

- To reload classes only:

  - In each subdirectory of the resources directory (with html files) in file options.xmlc must be set:

    ```
    -generate both
    ```

  - In app.conf set

    ```
    Server.AutoReload = true
    ```

- To reload classes and recompile HTML:

  - In each subdirectory of the resources directory (with html files) in file options.xmlc must be set:

    ```
    -for-recomp
    ```

  - In app.conf set

    ```
    Server.XMLC.AutoRecompilation = true
    ```

  - The HTML files must be in the same directories as the corresponding compiled java classes.

- Optionally add the XMLC option for logging, i.e.:

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

Chapter 3. What the Options Do

There are three ideas you need to understand in order to make recompilation work.

- The application cannot deploy in a jar file.

  Enhydra applications typically deploy as a single Java Archive File with all of the necessary classes. However, updating a jar file at runtime with new classes does not make sense, so the application's classes must be in a regular directory structure. The root directory is defined with the directive

  ```
  Server.ClassPath[] = "@DEPLOY_PATH@/../classes"
  ```
  in the app.conf file (configuration file of the application, where app presents a name of the application). The HTML files must be in the same directories as the corresponding compiled java classes.

- Pages must be instantiated with the xmlcFactory. In general, XMLC page objects can be instantiated with either new or the xmlcFactory as shown in the following code:

  ```
  WelcomeHTML welcome = new WelcomeHTML();
  ```
  or

  ```
  WelcomeHTML welcome = (WelcomeHTML)comms.xmlcFactory.create(WelcomeHTML.class);
  ```
  However, with autorecompilation, you must instantiate a page with the create method of xmlcFactory. The factory does the work of checking the timestamps on the files and recompiling an HTML page if it has been changed. Note that the factory must work with interfaces not directly on Java classes. Taking the example above, Welcome.html must be compiled with an option so WelcomeHTML.class is an interface not a class. So Welcome.html has an interface WelcomeHTML.class, and an implementation of that interface in WelcomeHTMLImpl.class. The options to generate interfaces are described below. On a final note, it's a better programming practice always to use xmlcFactory to instantiate pages. By default, it simply does a new on the page, and using it gives you the option of turning on recompilation or other factory features at a later date.

- Either the class file or HTML file can be updated at runtime.

  Consider the following :

  - The source HTML file is compiled . . . .
  - into a Java class file that is . . .
  - loaded by the server at runtime.

To replace the Java class representing a page at runtime, you can replace either the original HTML file or the class file compiled from the page. For example, a running application has loaded WelcomeHTMLImpl.class. To change the look of the page it generates, you could either replace the file Welcome.html or recompile Welcome.html yourself and replace the file WelcomeHTMLImpl.class.

The options you specify in options.xmlc and app.conf reflect these two possibilities. To only reload classes invoke XMLC with the -generate both option. This generates both an interface and an implementation class. To reload from classes and HTML use the XMLC option -for-recomp. This option tells XMLC to generate an interface, an implementation class, and *.xmlc file for each html page. The *.xmlc file includes the options used to compile the page so the same options are called when it is dynamically recompiled. The *.xmlc file contains options written in the MetaData xml language.

There are also options set in app.conf to turn on reloading and recompilation at runtime. Specifiy

```
Server.AutoReload = true
```
to enable class reloading and

```
Server.XMLC.AutoRecompilation = true
```

to enable both class and page reloading.

Note that the autorecompilation's capability comes at the cost of checking the timestamp on either the class file or the class and HTML files each time the page is instantiated at runtime.

During the application development, when a lot of compiling is done, if we want to have the mentioned parameters always preset in app.conf file, it is necessary to include them in app.conf.in file of the input directory. It is because the app.conf file is created based on app.conf.in file.

Chapter 4. Configuring autorecompilation (example 1)

This example demonstrates how to configure autorecompilation. It is written against the 5.1 version of Enhydra. Create a new application called testApp with appwizard. Choose in appwizard for Component Type: Enhydra Application, for Project directory name: testApp, for Package: testApp.

To run the example do the following:

- Set the code in Welcome.java to use the xmlcFactory syntax like it does by default.
- In options.xmlc file, set the option

  ```
  -for-recomp
  ```

- In testApp.conf.in (input directory) set

  ```
  Server.ClassPath[] =  "@DEPLOY_PATH@/../classes"
  ```

  or you may set the apsolute path of the application root directory.
- In testApp.conf.in (input directory) add

  ```
  Server.XMLC.AutoRecompilation = true
  ```

- Run

  ```
  ant clean
  ```

  and then

  ```
  ant
  ```

- By default, code based on the HTML pages will be created in directory "testApp/classes/Generated Source". In this example, in subdirectory "testApp/classes/Generated Source/testApp/presentation" following files are generated:

  `WelcomeHTML.java (interface)`
  and

  `WelcomeHTMLImpl.java (implementation).`
- Copy Welcome.html in the same directory as the corresponding compiled java classes ("testApp/classes/testApp/presentation").
- Go to output directory and start application. To start the application, enter command ./ run in the Enhydra shell (UNIX) or command run in the command window (WINDOWS):

  - UNIX

    ```
    cd output
    ./run
    ```

  - WINDOWS

    ```
    cd output
    run
    ```

- After starting the application, test the functionality by adding a word like "howdy" in the body section of

"testApp/classes/testApp/presentation/Welcome.html". Hit shift-refresh on your web-browser and you should see the changes.

Creating necessary classes and files for Automatic Recompilation is possible to be done with XMLC from command line using apropriate options which are shown ih the following line (they are defined for this example):

```
xmlc -keep -class testApp.presentation.WelcomeHTML -for-recomp Welcome.html
```

You will see the following generated files: Welcome.html (the original file) WelcomeHTML.java (the text version of the interface) WelcomeHTML.class (the class file for the interface) WelcomeHTMLImpl.java (the text version of the interface implementation) WelcomeHTMLImpl.class (the class file for the implementation) Welcome-HTML.xmlc (an options file for future compilations)

Note that files which are generated in this way are placed in the same directory as the source HTML file. If we would like to use them, we would have to copy them in appropriate subdirectory structure.

Chapter 5. Configuring autoreloading (example 2)

This example demonstrates how to configure autoreloading. It is written against the 5.1 version of Enhydra. Create a new application called testApp with appwizard. Choose in appwizard for Component Type: Enhydra Application, for Project directory name: testApp, for Package: testApp.

To run the example do the following:

- Set the code in Welcome.java to use the xmlcFactory syntax like it does by default.
- In options.xmlc file, set the option

  ```
  -generate both
  ```

- In testApp.conf.in (input directory) set

  ```
  Server.ClassPath[] =  "@DEPLOY_PATH@/../classes"
  ```

  or you may set the apsolute path of the application root directory.
- In testApp.conf.in (input directory) set

  ```
  Server.AutoReload = true
  ```

- Run

  ```
  ant clean
  ```

  and then

  ```
  ant
  ```

- By default, code based on the HTML pages will be created in directory "testApp/classes/Generated Source". In this example, in subdirectory "testApp/classes/Generated Source/testApp/presentation" following files are generated:

  `WelcomeHTML.java (interface)`
  and

  `WelcomeHTMLImpl.java (implementation).`
- Go to output directory and start application. To start the application, enter command ./ run in the Enhydra shell (UNIX) or command run in the command window (WINDOWS):

  - UNIX

    ```
    cd output
    ./run
    ```

  - WINDOWS

    ```
    cd output
    run
    ```

- After starting the application, test the functionality by adding a word like "howdy" in the body section of testApp/src/testApp/resources/Welcome.html. Then, recompile the application with the ant command (from the other Enhydra shell (UNIX), or from the other command window (WINDOWS)) without stopping the application. Hit shift-refresh on your web-browser and you should see the changes.

Creating necessary classes and files for Automatic Reloading is possible to be done with XMLC from command line using apropriate options which are shown ih the following line (they are defined for this example):

```
xmlc -keep -class testApp.presentation.WelcomeHTML -generate both Welcome.html
```

You will see the following generated files: Welcome.html (the original file) WelcomeHTML.java (the text version of the interface) WelcomeHTML.class (the class file for the interface) WelcomeHTMLImpl.java (the text version of the interface implementation) WelcomeHTMLImpl.class (the class file for the implementation)

Note that files which are generated in this way are placed in the same directory as the source HTML file. If we would like to use them, we would have to copy them in appropriate subdirectory structure.

Chapter 6. DiscRack demo example and Automatic Reloading (example 3)

To make discRack example to be an application with Automatic Reloading possibilities, the following steps must be done:

- In <DiscRack_root>/discRack/discRack.conf.in template file two parameters must be set as lines below:

  ```
  Server.ClassPath[] = @OUTPUT@/../classes

  Server.AutoReload = true
  ```

- In all options.xmlc files

  ```
  -generate both
  ```
  options must be added.

- Compile discRack using ant command

- Start discRack by enter the following commands in the Enhydra shell (UNIX) or in the command window (WINDOWS).

  - UNIX

    ```
    cd output

    ./run
    ```

  - WINDOWS

    ```
    cd output

    run
    ```
  and view results in your Web browser on 5555 port (http://localhost:5555).

- Change something in html or java files and then compile it (withouth stopping the application). ant clean and ant command can be used to rebuild the whole changed application but careful, because your database will be erased or overriden and all data entered before will be lost. Better way is to perform particular compilation of changed files. For example, change something in DiscCatalog.html and than from command line make appropriate class files:

  ```
  xmlc -class discRack.presentation.discMgmt.DiscCatalogHTML options.xmlc DiscCatalog.html
  ```
  where options.xmlc is file from directory which corresponds to DiscCatalog.html

- Copy created files (DiscCatalogHTML.class and DiscCatalogHTMLImpl.class) in appropriate directory in the classes root directory, which will override existing files.

- Reload page in your web browser

Chapter 7. DiscRack demo example and Automatic Recompilation (example 4)

To make discRack example to be an application with Automatic Recompilation possibilities, the following steps must be done.

- In <DiscRack_root>/discRack/discRack.conf.in template file two parameters must be set as lines below:

  ```
  Server.ClassPath[] = @OUTPUT@/../classes

  Server.XMLC.AutoRecompilation = true
  ```

- In all options.xmlc files

  ```
  -for-recomp
  ```
  options must be added.

- Compile discRack using ant command

- Start discRack by enter the following commands in the Enhydra shell (UNIX) or in the command window (WINDOWS).

  - UNIX

    ```
    cd output

    ./run
    ```

  - WINDOWS

    ```
    cd output

    run
    ```
  and view results in your Web browser on 5555 port (http://localhost:5555).

- Copy all html files from directory resources next to his pair of generetad java class files in directory classes.

- Change something in some of the copied html files in directory classes.

- Reload page in your web browser.