CommandLine204

- TARGET-AUDIENCE: ***beginner*** advanced expert
- COCOON-RELEASES: 2.0.4
- DOCUMENT-STATUS: ***draft*** reviewed released

The Cocoon Command Line Interface

The Cocoon command line interface allows static versions of Cocoon web sites to be created.

It can follow links within pages (not just HTML) and adapt filename extensions to match the page's mime type.

Using the Cocoon Command Line Interface

The command line interface is accessible via the java class org.apache.cocoon.Main. However, in order to use it, various things must be set up (e.g. environment variables and a Java class path).

To make its use easier, a DOS batch file (run.bat) and a Unix shell script (run.sh) are available. At present, these files are only available with the source distribution, but they can be used with modification with the binary distribution. They have been created with the assumption that you build Cocoon from the sources. If you want to use the binary distribution, you will need to alter various paths within these files.

Alternative versions have been provided near the end of this page for both Unix and Windows.

[NOTE: Couldn't we have a copy of these scripts in the binary distribution, all correctly configured?]]

To use the command line interface, you do not have to have a servlet engine (e.g. Tomcat) running.

Command line parameters

You can get a listing of the parameters on unix with:
```
./run.sh -h
```

or on Windows with:
```
run.bat -h
```

Which should look something like this:

```
------------------------------------------------------------------------
Apache Cocoon 2.0.4
Copyright (c) 1999-2002 Apache Software Foundation. All rights reserved.
------------------------------------------------------------------------

Usage: java org.apache.cocoon.Main [options] [targets]

Options:
        -b, --brokenLinkFile <argument>
                send a list of broken links to a file (one URI per line)
        -f, --uriFile <argument>
                use a text file with uris to process (one URI per line)
        -h, --help
                print this message and exit
        -v, --version
                print the version information and exit
        -k, --logKitconfig <argument>
                use given file for LogKit Management configuration
        -l, --Logger <argument>
                use given logger category as default logger for the Cocoon e
                ngine
        -u, --logLevel <argument>
                choose the minimum log level for logging (DEBUG, INFO, WARN,
                 ERROR, FATAL_ERROR) for startup logging
        -c, --contextDir <argument>
                use given dir as context
        -d, --destDir <argument>
                use given dir as destination
        -w, --workDir <argument>
                use given dir as working directory
        -P, --precompileOnly
                generate java code for xsp and xmap files
        -a, --userAgent <argument>
                use given string for user-agent header
        -p, --accept <argument>
                use given string for accept header
        -r, --followLinks <argument>
                process pages linked from starting page or not (boolean argu
                ment is expected, default is true)
        -C, --configFile <argument>
                specify alternate location of the configurationfile (default
                 is ${contextDir}/cocoon.xconf)
Note: the context directory defaults to './webapp'
```

Directories

- The context directory for Cocoon is the directory containing the root sitemap and the WEB-INF folder (usually `$TOMCAT_HOME/webapps/cocoon/`).
- The config file will be `$contextDir/WEB-INF/cocoon.xconf`
- The work directory can be anywhere you choose, it is where Cocoon will store various internal files generated whilst producing your site. Tomcat uses $TOMCAT_HOME/work/localhost/cocoon/ as its work directory. You could use the same. [IS THIS CORRECT??]]
- The destination directory is where your site files will be placed when generated.
- The broken link file is used to record any links that were found that could not be successfully followed.

If either the work or destination directories do not exist, they will be created for you.

URIs and URI files

All parameters provided to the command line without a switch will be taken as the URI of a page to be generated. Alternatively, the -f switch can be used to provide a URI filename. URIs should be listed in this file one per line. Comments are not currently allowed and blank lines should be avoided.

The URIs provided should be just the part that is handled by Cocoon. So, if you wanted to generate the page that would be accessed as `http://localhost/cocoon/mysite/mypage.html`, your URI would be `mysite/mypage.html`.

Following Links

The command line interface has powerful functionality for following links within pages, using the `-r` switch. Links are identified from within the pipeline itself, and thus can be followed in any document that has href, src or xlink:link somewhere within the pipeline.

To achieve this, the command line uses a combination of the link serializer and the 'links' view.

The 'links' view is set up within the <views> section of the sitemap. It's definition is like so:

```
<map:views>

 <map:view from-position="last" name="links">
  <map:serialize type="links"/>
 </map:view>

</map:views>
```

If this definition is in your root sitemap, you do not need it in your sub-sitemaps.

This definition must come before the pipelines in your sitemap.

To see the links within one of your pages using the link view, append ?cocoon-view=links to your URL (or &cocoon-view=links if there is already a '?' in your URL).

Mime Type Checking

The command line interface can rewrite URLs for pages to make them suitable for saving in files.

To achieve this, Cocoon first identifies the mime type of a generated page and checks that the URL has an appropriate file extension. If it doesn't, it adds one.

It then replaces '' with ', ? and : with _, and adds a default filename if the URL ends with /.

Cocoon also correctly rewrites the links within your pages, so that your link hierarchy is correctly maintained.

Multiple Page Renderings

In order to follow links and rewrite URLs, Cocoon must generate pages multiple times:

- once to extract links from the page, using the links view
- once to check the mime type for URL rewriting
- once for getting page contents

In Cocoon 2.1, in certain circumstances, the number of page generations can be reduced, as described below.

If link following is not required, the links view will not be generated.

If no URLs need rewriting (i.e. they all have the correct file extensions, and none end in /), the **** switch can be used to switch off URL rewriting, thus preventing the generation of the page to get its mime type. [NOTE: no switch is available to give access to the confirmExtension option within the code]]

Logging

I have not explored the Cocoon logging functionality and can therefore add no more than is given by Cocoon help:

```
       -k, --logKitconfig <argument>
              use given file for LogKit Management configuration
       -l, --Logger <argument>
              use given logger category as default logger for the Cocoon e
              ngine
       -u, --logLevel <argument>
              choose the minimum log level for logging (DEBUG, INFO, WARN,
               ERROR, FATAL_ERROR) for startup logging
```

In Cocoon 2.1, the -V switch can be used to switch on 'verbose' output to the console window.

The Log Kit Config parameter is a path to the logkit.xconf file. The logger parameter refers to the logging category to be used, e.g. 'cli'. The logLevel defines how much logging is to be done (as described above).

Precompiling XSPs

Using the 'precompile only' switch, Cocoon will only precompile XSPs who's URIs were given as command line parameters, and will not generate any content. [IS THIS CORRECT?]]

Precompiling Sitemaps

Before generating any pages, Cocoon will compile its sitemaps and XSP pages. [IS THIS CORRECT?]]

Agent Options

Using 'agent options', the command line can be made to emulate specific browsers. This is particularly useful when your site serves different content depending upon the user's browser (otherwise known as 'user agent').

Accept Options

When requesting pages, browsers can inform the server what kinds of content they can accept (e.g. "this browser can handle 'image/png'"). If your site returns different pages depending upon browser capabilities, you may want to use this option.

Sample Unix Script

Below is a simple unix script which adds every .jar file it finds into the classpath automatically.

[NOTE: This script has only been tested with Java 1.4]]

```
#!/bin/sh
# Portions copyright (c) 2001-2002 The Apache Software Foundation.  All rights reserved.

for i in WEB-INF/lib/*.jar
do
  # if the directory is empty, then it will return the input string
  # this is stupid, so case for it
  if [ "$i" != "WEB-INF/lib/*.jar" ] ; then
    if [ -z "$LOCALCLASSPATH" ] ; then
      LOCALCLASSPATH=$i
    else
      LOCALCLASSPATH="$i":"$LOCALCLASSPATH"
    fi
  fi
done

if [ -z "$TOMCAT_HOME" ] ; then
      TOMCAT_HOME=$CATALINA_HOME
fi

if [ ! -z "$TOMCAT_HOME" ] ; then
      LOCALCLASSPATH="$TOMCAT_HOME/common/lib/servlet.jar":"$LOCALCLASSPATH"
fi

java -classpath $LOCALCLASSPATH org.apache.cocoon.Main $*
```

For Java 1.3, try adding:

```
   LOCALCLASSPATH="$LOCALCLASSPATH":"%TOMCAT_HOME%\common\lib\xml-apis.jar"
   LOCALCLASSPATH="$LOCALCLASSPATH":"%TOMCAT_HOME%\common\lib\%TOMCAT_HOME%\common\lib\xalan-2.3.1.jar"
   LOCALCLASSPATH="$LOCALCLASSPATH":"%TOMCAT_HOME%\common\lib\%TOMCAT_HOME%\common\lib\batik-all-1.5b1.jar"
   LOCALCLASSPATH="$LOCALCLASSPATH":"%TOMCAT_HOME%\common\lib\xercesImpl-2.0.0.jar"
```

Note: This has Java 1.3 tweak has not been tested.

Sample Windows Batch File

Note: this has been tested on Cocoon 2.0.4 using Java 1.3 and Tomcat 4.0.3.

```
@echo off
cls
```

```
:: ---------------------------------------------------------------------------
:: run.bat - Win32 Run Script for Apache Cocoon
::
:: $Id: run.bat,v 1.3 2002/03/06 15:44:57 nicolaken Exp $
:: ---------------------------------------------------------------------------

:: ----- Verify and Set Required Environment Variables -------------------------

if not "%JAVA_HOME%" == "" goto gotJavaHome
echo You must set JAVA_HOME to point at your Java Development Kit installation
goto cleanup
:gotJavaHome

:: ----- Verify and Set Tomcat Location (UV 10/1/03) ---------------------------
if not "%TOMCAT_HOME%" == "" goto gotTomcatHome
echo You must set TOMCAT_HOME to point to your Tomcat installation
goto cleanup
:gotTomcatHome

:: ----- Verify and Set Required Environment Variables -------------------------

if not "%COCOON_LIB%" == "" goto gotCocoonLib
set COCOON_LIB=.\WEB-INF\lib
:gotCocoonLib

if not "%COCOON_WORK%" == "" goto gotCocoonWork
set COCOON_WORK=.\work
:gotCocoonWork

:: ----- Set Up The Runtime Classpath ------------------------------------------
set CP=%JAVA_HOME%\lib\tools.jar;%COCOON_WORK%

call appendcp.bat %COCOON_LIB%\avalon-framework-20020627.jar
call appendcp.bat %COCOON_LIB%\cocoon-2.0.4.jar
call appendcp.bat %COCOON_LIB%\cocoon-scratchpad.jar
call appendcp.bat %COCOON_LIB%\excalibur-logger-20020820.jar
call appendcp.bat %COCOON_LIB%\excalibur-sourceresolve-20020820.jar
call appendcp.bat %COCOON_LIB%\excalibur-component-20020916.jar
call appendcp.bat %COCOON_LIB%\excalibur-pool-20020820.jar
call appendcp.bat %COCOON_LIB%\excalibur-cli-1.0.jar
call appendcp.bat %COCOON_LIB%\commons-logging-1.0.jar
call appendcp.bat %COCOON_LIB%\logkit-20020529.jar
call appendcp.bat %COCOON_LIB%\excalibur-collections-20020820.jar
call appendcp.bat %COCOON_LIB%\excalibur-instrument-20021108.jar
call appendcp.bat %COCOON_LIB%\excalibur-monitor-20020820.jar
call appendcp.bat %COCOON_LIB%\excalibur-xmlutil-20020820.jar
call appendcp.bat %COCOON_LIB%\commons-jxpath-1.0.jar
call appendcp.bat %COCOON_LIB%\pizza-1.1.jar
call appendcp.bat %COCOON_LIB%\velocity-1.2.jar
call appendcp.bat %COCOON_LIB%\excalibur-datasource-vm12-20021121.jar
call appendcp.bat %COCOON_LIB%\commons-collections-2.1.jar
call appendcp.bat %COCOON_LIB%\resolver-20020130.jar

:: Removed because the classpath it generates is too long:
rem for %%i in (WEB-INF\lib\*.jar) do call appendcp.bat %%i

call appendcp.bat %TOMCAT_HOME%\common\lib\servlet.jar

:: Required for Java 1.3
call appendcp.bat %TOMCAT_HOME%\common\lib\xml-apis.jar
call appendcp.bat %TOMCAT_HOME%\common\lib\xercesImpl-2.0.0.jar
call appendcp.bat %TOMCAT_HOME%\common\lib\xalan-2.3.1.jar
call appendcp.bat %TOMCAT_HOME%\common\lib\batik-all-1.5b1.jar

:: ----- Run Cocoon ------------------------------------------------------------

if "%OS%" == "Windows_NT" goto nt
%JAVA_HOME%\bin\java.exe %COCOON_OPTS% -classpath %CP% org.apache.cocoon.Main %1 %2 %3 %4 %5 %6 %7 %8 %9
:nt
%JAVA_HOME%\bin\java.exe %COCOON_OPTS% -classpath %CP% org.apache.cocoon.Main %*
```

```
:: ----- Cleanup the environment -------------------------------------------

:cleanup
set CP=
```

Save this as run.bat in the `$TOMCAT_HOME\webapps\cocoon` folder. You will also need to create a file in the same folder called `appendcp.bat`, with the following contents:

```
set CP=%CP%;%1
```

Execute run.bat as follows:
`run -c . -C WEB-INF/cocoon.xconf -u ERROR YOUR-URI`
If you find that it gives an error of:
`Exception in thread "main" java.lang.NoClassDefFoundError:.....`
look for a jar file in the WEB-INF/lib folder that is likely to contain the class that was not found, and add a line `call appendcp.bat %COCOON_LIB% \<JAR-FILENAME>` to the above script.

[Note, this script mentions each jar file by name, rather than including all jars in the WEB-INF/lib folder because doing the latter would make a classpath that is too long for a DOS prompt. Does anyone know how to get DOS to accept a longer environment variable?]]

Sample from Cocoon's own build.xml file

Cocoon uses itself in command-line mode to generate its documentation. Studying the build.xml file that comes with the Cocoon source gives a very concrete example of how to use the command-line mode.

Here's the relevant build.xml section from the current source code. Although this build information is meant for *ant*, it is fairly self-explanatory.

```
<java classname="org.apache.cocoon.Main" fork="true"
dir="${build.context}"
          failonerror="true" maxmemory="128m">
      <arg value="-c."/>
      <arg value="-d../docs"/>
      <arg value="-w../work"/>
      <arg value="-b../brokenlinks.txt"/>
      <arg value="-k../documentation/logkit.xconf"/>
      <arg value="-u${build.docs.loglevel}"/>
      <arg value="-V"/>
      <arg value="index.html"/>
      <classpath>
        <path refid="classpath"/>
        <fileset dir="${build.dir}">
          <include name="*.jar"/>
        </fileset>
        <pathelement location="${tools.jar}"/>
        <pathelement location="${build.context}/WEB-INF/classes"/>
      </classpath>
    </java>
```

Sitemap parameters to use jdbc in SQL transformer.

{{{<map:pipeline>
<map:match pattern="test.pdf">
<map:generate src="documents/test.xml"/>
<map:transform type="xslt" src="documents/test-prep.xsl">
<map:parameter name="use-request-parameters" value="true"/>
</map:transform>
<map:transform type="sql" label="raw">
<map:parameter name="dburl" value="jdbc:mysql://localhost:3306/test"/>
<map:parameter name="username" value="test"/>
<map:parameter name="password" value="secret"/>
</map:transform>
<map:transform type="xslt" src="documents/test-pdf.xsl" label="raw4"/>
<map:serialize type="fo2pdf"/>
</map:match>
}}}

Some Keywords

CLI, offline generation