CommandLine

- TARGET-AUDIENCE: ***beginner*** advanced expert
- COCOON-RELEASES: 2.1M3 and later
- DOCUMENT-STATUS: ***draft*** reviewed released
- SEE ALSO: http://cocoon.apache.org/2.1/userdocs/offline/index.html

---

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.

This page relates to 2.1M3 and above. A page describing the CLI for 2.0.4 can be found at CommandLine204.

Using the Cocoon Command Line Interface

The command line interface is accessible via the java class `org.apache.cocoon.Main`, or via the command line. On Unix use `./cocoon.sh cli <parameters>`
On Windows use `cocoon cli <parameters>`
These script files are found in the top-level directory of the Cocoon source distribution.

To use the command line interface, you do not have to have a servlet engine (e.g. Tomcat) running. All you need is a site set up with a webapp to generate from (ie. with a sitemap.xmap and a cocoon.xconf).

Configuring the Command Line Interface

In 2.1M3 there are two ways to configure the CLI: via command line parameters or via an xconf configuration file. The latter provides a greater range of options.

xconf Configuration

To start the CLI using an xconf file, on Unix do `./cocoon.sh cli -x <xconf file>`

or on Windows: `cocoon cli -x <xconf file>`
A sample xconf file is included below. One can also be found in the Cocoon source distribution (`cli.xconf`).

```
<?xml version="1.0"?>
<!--+
    |  This is the Apache Cocoon command line configuration file.
    |  Here you give the command line interface details of where
    |  to find various aspects of your Cocoon installation.
    |
    |  If you wish, you can also use this file to specify the URIs
    |  that you wish to generate.
    |
    |  The current configuration information in this file is for
    |  building the Cocoon documentation. Therefore, all links here
    |  are relative to the build context dir, which, in the build.xml
    |  file, is set to ${build.context}
    |
    |  Options:
    |    verbose:            increase amount of information presented
    |                        to standard output (default: false)
    |    follow-links:       whether linked pages should also be
    |                        generated (default: true)
    |    precompile-only:    precompile sitemaps and XSP pages, but
    |                        do not generate any pages (default: false)
    |    confirm-extensions: check the mime type for the generated page
    |                        and adjust filename and links extensions
    |                        to match the mime type
    |                        (e.g. text/html->.html)
    +-->
```

```
<cocoon verbose="true"
        follow-links="true"
        precompile-only="false"
        confirm-extensions="false">

   <!--+
       | Broken link reporting options:
       |   Report into a text file, one link per line:
       |     <broken-links type="text" report="filename"/>
       |   Report into an XML file:
       |     <broken-links type="xml" report="filename"/>
       |   Ignore broken links (default):
       |     <broken-links type="none"/>
       |   When a page includes an error, should a page be generated?
       |
       |   Two attributes to this node specify whether a page should
       |   be generated when an error occured. 'generate' specifies
       |   whether a page should be generated (default: true) and
       |   extension specifies an extension that should be appended
       |   to the generated page's filename (default: none)
       |     <broken-links generate="true" extension=".error.txt"/>
       |
       +-->
   <broken-links type="xml"
                 file="brokenlinks.xml"
                 generate="false"
                 extension=".error"/>

   <!--+
       |  Load classes at startup. This is necessary for generating
       |  from sites that use SQL databases and JDBC.
       |  The <load-class> element can be repeated if multiple classes
       |  are needed.
       +-->
   <!--
   <load-class>org.firebirdsql.jdbc.Driver</load-class>
   -->

   <!--+
       |
       +-->
   <logging log-kit="WEB-INF/logkit.xconf" logger="cli" level="ERROR" />

   <!--+
       |  The context directory is usually the webapp directory
       |  containing the sitemap.xmap file.
       |
       |  The config file is the cocoon.xconf file.
       |
       |  The work directory is used by Cocoon to store temporary
       |  files and cache files.
       |
       |  The destination directory is where generated pages will
       |  be written (assuming the 'simple' mapper is used)
       +-->
   <context-dir>.</context-dir>
   <config-file>WEB-INF/cocoon.xconf</config-file>
   <work-dir>work/docs</work-dir>
   <dest-dir>dest</dest-dir>

   <!--+
       | Specifies the filename to be appended to URIs that
       | refer to a directory (i.e. end with a forward slash).
       +-->

   <default-filename>index.html</default-filename>

   <!--+
       |  Specifies a user agent string to the sitemap when
       |  generating the site.
       +-->
```

```
  <!--
  <user-agent>xxx</user-agent>
  -->

  <!--+
      |  Specifies an accept string to the sitemap when generating
      |  the site.
      +-->

  <accept>*/*</accept>

  <!--+
      |  Specifies the URIs that should be generated (using <uri>
      |  elements, and (if necessary) what should be done with the
      |  generated pages.
      |
      |  The old behaviour - appends uri to the specified destination
      |  directory (as specified in <dest-dir>):
      |
      |   <uri>documents/index.html</uri>
      |
      |  Append: append the generated page's URI to the end of the
      |  dest URI:
      |
      |   <uri type="append" src-prefix="documents/" src="index.html"
      |   dest="build/dest/"/>
      |
      |  Replace: Completely ignore the generated page's URI - just
      |  use the destination URI:
      |
      |   <uri type="replace" src-prefix="documents/" src="index.html"
      |   dest="build/dest/docs.html"/>
      |
      |  Insert: Insert generated page's URI into the destination
      |  URI at the point marked with a * (example uses fictional
      |  zip protocol)
      |
      |   <uri type="insert" src-prefix="documents/" src="index.html"
      |   dest="zip://*.zip/page.html"/>
      |
      +-->
  <uri>favicon.ico</uri> <!-- requires 2.1.3dev or later -->
  <uri type="append" src-prefix="documents/" src="index.html" dest="docs/"/>

  <!--+
      |  File containing URIs (plain text, one per
      |  line).
      +-->

  <!--
  <uri-file></uri-file>
  -->

</cocoon>
```

Command line parameters

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

or on Windows with:
`cocoon cli -h`

Which should look something like this:

```
-------------------- Executing -----------------
Main Class: org.apache.cocoon.Main
usage: cocoon cli [options] [targets]
------------------------------------------------------------------------
cocoon 2.1m3-dev
Copyright (c) 1999-2003 Apache Software Foundation. All rights reserved.
------------------------------------------------------------------------
 -a,--userAgent            use given string for user-agent header
 -e,--confirmExtensions    confirm that file extensions match mime-type of
                           pages and amend filename accordingly (default is true)
 -C,--configFile           specify alternate location of the configuration
                           file (default is ${contextDir}/cocoon.xconf)
 -D,--defaultFilename      specify a filename to be appended to a URI when
                           the URI refers to a directory
 -L,--loadClass            specify a class to be loaded at startup
                           (specifically for use with JDBC). Can be used multiple times
 -P,--precompileOnly       generate java code for xsp and xmap files
 -V,--verbose              enable verbose messages to System.out
 -b,--brokenLinkFile       send a list of broken links to a file (one URI
                           per line)
 -c,--contextDir           use given dir as context
 -d,--destDir              use given dir as destination
 -f,--uriFile              use a text file with uris to process (one URI
                           per line)
 -h,--help                 print this message and exit
 -k,--logKitconfig         use given file for LogKit Management
                           configuration
 -l,--Logger               use given logger category as default logger for
                           the Cocoon engine
 -p,--accept               use given string for accept header
 -r,--followLinks          process pages linked from starting page or not
                           (boolean argument is expected, default is true)
 -u,--logLevel             choose the minimum log level for logging
                           (DEBUG, INFO, WARN, ERROR, FATAL_ERROR) for startup logging
 -v,--version              print the version information and exit
 -w,--workDir              use given dir as working directory
 -x,--xconf                specify a file containing XML configuration
                           details for the command line interface
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`.

Defining your Targets

The CLI can write to any destination for which a ModifiableSource exists. Specific sources are accessed via a URI. The URI to which the generated page is to be dispatched must somehow be built up from a combination of the URI specified as the destination and the URI of requested page. There are currently four ways to identify the destination URI of a page:

- Default: appends the requested page URI onto the end of the URI specified in the `<dest-dir>` element, e.g. `<uri>documents/index.html</uri>` (this only works in 2.1.3dev and later)
- Append: sticks the uri of the generated page onto the end of the dest uri, e.g. `<uri type="append" src-prefix="documents/" src="index.html" dest="build/dest/"/>` would produce `build/dest/index.html`.
- Replace: just uses the dest uri, ignoring the uri of the generated page, e.g. `<uri type="replace" src-prefix="documents/" src="index.html" dest="build/dest/docs.html"/>` would produce `build/dest/docs.html`.
- Insert: lets you put the uri of the generated page into the middle of the dest uri, e.g. `<uri type="insert" src-prefix="documents/" src="index.html" dest="zip://*.zip/page.html"/>` would produce `zip://index.html.zip/page.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>
```

The `map:views` section is not inherited to sub sitemaps, so you need this there even if you have this definition in a parent sitemap.

This definition must come before the pipelines in your sitemap.

To see the links within one of your pages using the link view, append a request parameter `cocoon-view=links` to 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. In 2.1, this is not relevant, as the sitemap is now interpreted rather than compiled.

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.

Accessing the CLI from Ant

Cocoon uses itself in command-line mode to generate its documentation, using Ant to invoke the CLI.

Below is a sample showing how to control the CLI via an Ant build script.

```
<java classname="org.apache.cocoon.Main" fork="true"
dir="${build.context}"
          failonerror="true" maxmemory="128m">
      <arg value="-xcli.xconf"/>
      <arg value="index.html"/>
      <classpath>
        <path refid="classpath"/>
        <fileset dir="${build.dir}">
          <include name="*.jar"/>
        </fileset>
        <pathelement location="${tools.jar}"/>
        <pathelement location="${build.context}/WEB-INF/classes"/>
      </classpath>
    </java>
```

Issues with the HTMLGenerator

The HTMLGenerator doesn't work correctly with the 2.1 release, due to some unresolved dependencies on the servlet class files. For using the HTMLGenerator, first you should checkout the latest cocoon version from CVS. Afterwards, just copy the `servlet_2_2.jar` file from `$COCOONBASEDIR /lib/optional` to your lib directory (for example, `WEB-INF/lib`). (**Note:** The HTMLGenerator should work correctly with the 2.1.1 release).

A few miscellaneous troubleshooting notes

Below are some notes I wrote up while trying to get CLI to work on my machine under Cocoon 2.1.2. I did get it to work at least rudimentarily. I hope these are of help to you.

- LarsHuttar

```
- Follow directions at http://wiki.cocoondev.org/Wiki.jsp?page=CommandLine
  on getting CLI working. My details:
- I'm using cli.xconf to specify cli configuration, rather than the command line.
  In the supplied cli.xconf, I changed context-dir to:
  <context-dir>C:\Program Files\Apache Group\Tomcat 4.1\webapps\cocoon</context-dir>
  and added a uri element:
  <uri src="test/1"/>
- In the Tomcat 4.1\webapps\cocoon directory, I up test\sitemap.xmap with the
  following matcher:
    <map:match pattern="1">
      <map:read src="test1.html" mime-type="text/html" />
    </map:match>
  And added the file test\test1.html with some simple HTML in it.
- Run the CLI as follows:
    ./cocoon.bat cli -x cli.xconf
- You can apparently disregard the following errors:
  - Cannot find CatalogManager.properties
  - server.properties not found, using command line or default properties
  - java.sql.SQLException: The database is already in use by another process
        at org.hsqldb.Trace.getError(Unknown Source) ...
- If there's an error with missing class ServletConfig, (in my case the process
  would seem to become a server and start waiting for connections), the fix is
  to copy lib/optional/servlet_2_2.jar to build/webapp/WEB-INF/lib/
- If the following error message appears:
  ERROR   2004-03-04 14:14:43.816 [cli.mana] (): Could not set up Component for hint [ portlet]
  org.apache.avalon.framework.context.ContextException: Unable to resolve context key: servlet-config
  .....
  uncomment the following lines from the file local.blocks.properties and compile again
     exclude.block.portal-fw=true
     exclude.block.portal=true
  for more: http://thread.gmane.org/gmane.text.xml.cocoon.user/33374
- You can tell it worked if it reports a "Total time" and creates files
  in build/dest.

- Note on missing HttpSessionBindingListener class ([http://64.233.187.104/search?q=cache:buUZF5frFgQJ:archives.
real-time.com/pipermail/cocoon-users/2004-January/045393.html+NoClassDefFoundError:+javax/servlet/http
/HttpSessionBindingListener&hl=en&client=firefox-a):])

>I built Cocoon 2.1.3 sucessfully and tried to run cocoon cli with the cli.xconf file provided with the cocoon
distribution. I got an invocationTargetException caused by a java.lang.NoClassDefFoundError: javax/servlet/http
/HttpSessionBindingListener ...

You've included a block that requires an HTTP binding (maybe the authentication framework?) You could copy
servlet.jar into lib/local and rebuild cocoon (or copy it into WEB-INF/lib), which might get you working
(you'll find it in lib/optional I believe).
I know this is not good, and at some point it needs to be dealt with (Cocoon is too HTTP centric). But
hopefully these will get you going further.
Regards, Upayavira
```

Some Keywords

CLI, command line interface, offline generation