YourCocoonBasedProject

About

This page describes a hack to get you doing some automated build for your Cocoon based projects depending on some particular Cocoon distribution. This is based on how Cocoon blocks themselves are being build in the new (as of Cocoon 2.1) build system. It was introduced here after some discussion on the cocoon-dev mailing-list (see http://marc.theaimsgroup.com/?t=105347757500001&r=1&w=2)

Disclaimer: Limited shelf life of this: while the described ideas should work for anything starting the 2.1M* releases, investigating how this could be useful in a 2.0.x is TBD. Most, if not all, of this will probably stop being relevant when Blocks are getting what they are intended to be. (See BlocksDefinition for that)

Feel free to extend and modify this page with your own additions / hacks / ideas.

Update

Enhanced and updated version can be found at YourCocoonBasedProjectAnt16

The works

This approach starts from (but is not limited to) the idea that you are having more then one Cocoon distribution sitting on your hard drive, and you want to be able to develop your project versus any of them in a flexible way.

So suppose you have some locations where there are cocoon-2.1 directories, e.g.:

1. sandbox of cvs-head
2. sandbox of cvs with sticky tag on some specific data or release
3. unzipped-untarred Milestone releases.

At each time your project will point to one of these directories as its (current) 'Cocoon distribution of choice' (through some cocoon-dist.home property)

Inside your project-directory you then provide some structure resembling this:

```
+ myproject
   +- lib
        ! not containing the cocoon jars,
        ! since that depends on the distribution used
   +- src
      +- java
      +- cocoon
         +- local.build.properties
              ! block and build customization file as suggested in the cocoon
              ! install and readme files, to include only those blocks your
              ! project will need
         +- webapp
           +- some-mount-subdir
              +- sitemap.xmap
         +- xconf
           +- *.xconf, *.xweb, *.xmap, *.xlog:
                ! using xpatch-task syntax to augment the xml config files:
                ! cocoon.xconf, web.xml, sitemap.xmap resp. logkit.xconf
   +- user.properties:
          ! specifies cocoon-dist.home of distribution to use in your local case
   +- build.xml
```

adding to this additional directories making space for

- ./build: the typical build products of your project
- ./tools/cocoon: stuff you will dynamically get from the cocoon distribution

This 'getting' of stuff from the Cocoon distribution is done through a simple convenience script:

For Windows, some get_cocoon.bat:

```
@echo off
cd %COCOON_HOME%
build.bat -propertyfile %PROJECT_PROPERTIES% clean webapp -Dbuild.webapp=%PROJECT_WEBAPP%
 -Dtools.tasks.dest=%PROJECT_TASKDEFS%
```

Which gets called from some ant target setting the environment

```
  <target name="cocoon.get">
    <exec executable="get_cocoon.bat" os="Windows 2000" >
      <env key="COCOON_HOME" file="${cocoon-dist.home}" />
      <env key="PROJECT_PROPERTIES" file="src/etc/cocoon/local.blocks.properties" />
      <env key="PROJECT_WEBAPP" file="tools/cocoon/webapp" />
      <env key="PROJECT_TASKDEFS" file="tools/cocoon/taskdefs" />
    </exec>
  </target>
```

A similar trick can be used to also test-run your project application using the Jetty container that is nowadays shipped with Cocoon.

A simple convenience script:

For Windows some run_cocoon.bat

```
@echo off
set JETTY_WEBAPP=%PROJECT_WEBAPP%

cd %COCOON_HOME%
cocoon.bat servlet
```

In combination with its build-target:

```
  <target name="cocoon.run" >
    <exec executable="runc2.bat" os="Windows 2000" >
      <env key="COCOON_HOME" file="${cocoon-dist.home}" />
      <env key="PROJECT_WEBAPP" file="tools/cocoon/webapp" />
    </exec>
  </target>
```

The remaining thing now is to actually create your own webapp based on the one that was pulled in with the `ant cocoon.get` (you would typically only call this cocoon.get when you want to upgrade or switch to another Cocoon distribution)

The basic parts of this webapp creation are

- compiling specific code, jarring it up and copying that to the WEB-INF/classes and lib
- copying over verbatim content, configuration and stylesheet files
- modification of existing XML config files for Cocoon

This last thing is what is the biggest issue for most of us. However, stealing from how the blocks inside cocoon are working we can achieve this through the use of the new xpatch task. (Inside Cocoon distro)

This piece of ant build.xml does the trick:

```
<target name="-cocoon.patch">
  <echo>
    Patching ${cocoon.patch.target} with
    ${cocoon-xconf.dir}/*.${cocoon.patch.src-extension} ...
  </echo>
  <xpatch file="${cocoon.patch.target}"
          srcdir="${cocoon-xconf.dir}"
          includes="**/*.${cocoon.patch.src-extension}"/>
</target>

<target name="cocoon.xconf" depends="-cocoon.check">
  <path id="cocoon-tasks.cp">
    <pathelement path="${cocoon.tasks}" />
    <path>
      <fileset dir="${cocoon.lib}">
        <include name="xalan*.jar" />
        <include name="xerces*.jar" />
        <include name="xml*.jar" />
      </fileset>
    </path>
  </path>

  <taskdef name="xpatch"
           classname="XConfToolTask"
           classpathref="cocoon-tasks.cp"/>

  <antcall target="-cocoon.patch" >
    <param name="cocoon.patch.target"
           value="${cocoon.webapp}/WEB-INF/cocoon.xconf" />
    <param name="cocoon.patch.src-extension"
           value="xconf" />
  </antcall>

  <antcall target="-cocoon.patch" >
    <param name="cocoon.patch.target"
           value="${cocoon.webapp}/WEB-INF/logkit.xconf" />
    <param name="cocoon.patch.src-extension"
           value="xlog" />
  </antcall>

  <antcall target="-cocoon.patch" >
    <param name="cocoon.patch.target"
           value="${cocoon.webapp}/sitemap.xmap" />
    <param name="cocoon.patch.src-extension"
           value="xmap" />
  </antcall>

  <antcall target="-cocoon.patch" >
    <param name="cocoon.patch.target"
           value="${cocoon.webapp}/WEB-INF/web.xml" />
    <param name="cocoon.patch.src-extension"
           value="xweb" />
  </antcall>

</target>
```

In combination with the xconf-patch-files in the ${cocoon-xconf.dir} directory.

A very classical example of such file would be:

```
<?xml version="1.0"?>

<xconf xpath="/cocoon"
       unless="component[@role='org.outerj.apples.AppleInstanceManager']">

  <!--+
      | Just an example section that will be slided into the cocoon.xconf
      +-->
  <component
      class="org.outerj.apples.SimplisticAppleInstanceManager"
            role="org.outerj.apples.AppleInstanceManager"
            logger="sitemap.action.apple-manager" />

</xconf>
```

For a more complete usage of this patching mechanism see XPatchTaskUsage – *(thx to GeoffHoward)*

Addendum

Some people have asked how to do the patching based on values of system properties. For this you can consider creating your xconf files on the fly with a trick like this:

```
    <target name="make-xpatchfile">
        <echo file="myfile.xconf"><![CDATA[<?xml version="1.0"?>
....

  <component class="${component-classname}" ..

....

]]></echo>
    </target>
```

in which case you could call ant with some {-Dcomponent-classname=package.subpackage.MyClass}

NB: I have tested this and looked at the code, and don't think this works. (GeoffHoward) Completion: (as checked with Geoff) this works alright. However: you have to make sure that the property to be substituted has a value set. If not the property-substitution will stay unchanged. And then Geoff is very right in saying that the xpatch task will not be doing that substitution for you.

While at it, this is another technique achieving the same:

Have a inside your file mypatch.xconf
{{{...

<component class="" ..

...
}}}

and use the copy-token-filtering support of ant:

```
    <mkdir dir="${cocoon-xconf.dir}" />
    <filter token="classname" value="${component-classname}"/>
    <copy todir="${cocoon-xconf.dir}" filtering="on" overwrite="true">
        <fileset dir="${src.xconf}" />
    </copy>
```

Addendum

I recently automated some more of this in a packaged set of ant-tasks that you can include in your own project's build.xml.

It defers with the above only in details (and it should work for linux as well, please fix/comment if it doesn't)

```
<!--
This Apache Ant build.xml snippet contains targets for helping out
```

```
managing the Cocoon dependencies of your project.

Usage: (assuming you use Apache Ant for your projects)

1) Copy this file to somewhere in your project.
   (e.g. to ./src/targets/cocoon-build.xml)
2) Add the following to the top of your project's Ant build.xml script
   (possibly adjusting the path):

  <!DOCTYPE project [
    <!ENTITY cocoon-targets SYSTEM "file:./src/targets/cocoon-build.xml">
  ]>

3) Before the closing '</project>' in your build.xml, add this:

  &cocoon-targets;


This is like expanding a macro: it pulls in the contents of this file.


All targets in this build file snippet depend upon
the following properties being set
        1. cocoon-dist.home
             location of src distribution of cocoon to use
        2. cocoon-build.properties
             property file with specific cocoon build settings
             (selecting which blocks, samples,...)
             typically src/cocoon/local.build.properties
        3. cocoon-xconf.dir
             location where the appropriate patch files can be found
             typically src/cocoon/xconf
        4. cocoon-tool.dir
             where cocoon is build inside your project
             typically this is tools/cocoon


A minimal build.xml would thus be:

<!DOCTYPE project [
<!ENTITY cocoon-targets SYSTEM "file:./cocoon-targets.ent">
]>

<project default="site">
    <property name="cocoon-dist.home" value="location-to-cocoon-src-dist" />
    <property name="cocoon-build.properties" value="src/cocoon/local.build.properties"
        <property name="cocoon-xconf.dir" value="src/cocoon/xconf" />
        <property name="cocoon-tool.dir" value="tools/cocoon" />

    &cocoon-targets;
</project>
-->


        <!-- sets some essesntial properties for these targets -->
    <target name="-cocoon.init">
                        <mkdir dir="${cocoon-tool.dir}" />
                        <property name="cocoon.webapp" value="${cocoon-tool.dir}/webapp" />
                        <property name="cocoon.tasks" value="${cocoon-tool.dir}/taskdefs" />
                        <property name="cocoon.lib" value="${cocoon.webapp}/WEB-INF/lib" />

    </target>



        <!-- checks what kind of OS this is running on -->
    <target name="-cocoon.oscheck" >
            <condition property="isWindows">
                    <os family="windows" />
                </condition>
    </target>
```

```
        <!-- creates Windows batch files for cocoon dependencies -->
    <target name="-cocoon.bat" if="isWindows"
                depends="-cocoon.init, -cocoon.oscheck" >

        <echo>Building batch files for support on windows OS</echo>
                <property name="shbat" value="bat" />

            <echo file="${cocoon-tool.dir}/getc2.${shbat}"><![CDATA[
@echo off
cd %COCOON_HOME%
build.bat -propertyfile %PROJECT_PROPERTIES% clean webapp "-Dbuild.webapp=%PROJECT_WEBAPP%"
 "-Dtools.tasks.dest=%PROJECT_TASKDEFS%"
]]></echo>

            <echo file="${cocoon-tool.dir}/runc2.${shbat}"><![CDATA[
@echo off
set JETTY_WEBAPP=%PROJECT_WEBAPP%
cd %COCOON_HOME%
cocoon.bat servlet-debug
]]></echo>
    </target>



        <!-- creates shell scripts for cocoon dependencies -->
    <target name="-cocoon.sh" unless="isWindows"
                depends="-cocoon.init, -cocoon.oscheck" >

        <echo>Building shell scripts for support on non-windows</echo>
                <property name="shbat" value="sh" />

            <echo file="${cocoon-tool.dir}/getc2.${shbat}"><![CDATA[#!/bin/sh
cd $COCOON_HOME
build.sh -propertyfile $PROJECT_PROPERTIES clean webapp -Dbuild.webapp=$PROJECT_WEBAPP
 -Dtools.tasks.dest=$PROJECT_TASKDEFS
]]></echo>
       <chmod file="${cocoon-tool.dir}/getc2.${shbat}" perm="u+x"/>

            <echo file="${cocoon-tool.dir}/runc2.${shbat}"><![CDATA[#!/bin/sh
export JETTY_WEBAPP=$PROJECT_WEBAPP
cd $COCOON_HOME
cocoon.sh servlet-debug
]]></echo>
        <chmod file="${cocoon-tool.dir}/runc2.${shbat}" perm="u+x"/>

    </target>



    <!-- creates as needed batch files or shell scripts -->
    <target name="-cocoon.shbat" depends="-cocoon.bat, -cocoon.sh" />



        <!-- checks if the cocoon dependency is holding what we expect
                 sets a variable if all is ok -->
        <target name="-cocoon.test" depends="-cocoon.init">
                <condition property="cocoon.ok" value="true">
                        <and>
                                <available type="dir" file="${cocoon.lib}" />
                                <available classname="XConfToolTask"
                                           classpath="${cocoon.tasks}"/>
                        </and>
                </condition>
        </target>
```

```
        <!-- fails the build if the cocoon dependency is not met -->
        <target name="-cocoon.check" depends="-cocoon.test" unless="cocoon.ok">
                <fail>No cocoon available. Run 'ant get_cocoon' first.</fail>
        </target>




        <target name="-cocoon.patch">
                <echo>Patching ${cocoon.patch.target} with
${cocoon-xconf.dir}/*.${cocoon.patch.src-extension} ...</echo>
                <xpatch
                        file="${cocoon.patch.target}"
            srcdir="${cocoon-xconf.dir}"
            includes="**/*.${cocoon.patch.src-extension}"/>
    </target>


    <!-- applies the patch files in the ${cocoon-xconf.dir}
              on the various cocoon conf files -->
        <target name="cocoon.xconf" depends="-cocoon.check">
                <path id="cocoon-tasks.cp">
                        <pathelement path="${cocoon.tasks}" />
                        <path>
                                <fileset dir="${cocoon.lib}">
                                        <include name="xalan*.jar" />
                                        <include name="xerces*.jar" />
                                        <include name="xml*.jar" />
                                </fileset>
                        </path>
                </path>

                <taskdef
                    name="xpatch"
                    classname="XConfToolTask"
                    classpathref="cocoon-tasks.cp"/>

                <antcall target="-cocoon.patch" >
                        <param name="cocoon.patch.target"
                                   value="${cocoon.webapp}/WEB-INF/cocoon.xconf" />
                        <param name="cocoon.patch.src-extension"
                                   value="xconf" />
                </antcall>

                <antcall target="-cocoon.patch" >
                        <param name="cocoon.patch.target"
                                   value="${cocoon.webapp}/WEB-INF/logkit.xconf" />
                        <param name="cocoon.patch.src-extension"
                                   value="xlog" />
                </antcall>

                <antcall target="-cocoon.patch" >
                        <param name="cocoon.patch.target"
                                   value="${cocoon.webapp}/sitemap.xmap" />
                        <param name="cocoon.patch.src-extension"
                                   value="xmap" />
                </antcall>

                <antcall target="-cocoon.patch" >
                        <param name="cocoon.patch.target"
                                   value="${cocoon.webapp}/WEB-INF/web.xml" />
                        <param name="cocoon.patch.src-extension"
                                   value="xweb" />
                </antcall>

        </target>



    <!-- gets cocoon-webapp into this project -->
    <target name="cocoon.get" depends="-cocoon.shbat">
```

```
    <exec executable="${cocoon-tool.dir}/getc2.${shbat}" >
                  <env key="COCOON_HOME" file="${cocoon-dist.home}" />
                  <env key="PROJECT_PROPERTIES" file="${cocoon-build.properties}" />
                  <env key="PROJECT_WEBAPP" file="${cocoon.webapp}" />
                  <env key="PROJECT_TASKDEFS" file="${cocoon.tasks}" />
    </exec>
</target>



<!-- runs cocoon on the built in jetty to test -->
<target name="cocoon.run" depends="-cocoon.shbat, -cocoon.check">
    <exec executable="${cocoon-tool.dir}/runc2.${shbat}" >
                  <env key="COCOON_HOME" file="${cocoon-dist.home}" />
            <env key="PROJECT_WEBAPP" file="${cocoon.webapp}" />
    </exec>
</target>
```