CustomConfigTarget

The [XPatchTaskUsage] task enables easier and survivable customization of the key Cocoon configuration files: * cocoon.xconf * the root sitemap.xmap * web.xml * and logkit.xconf \\ This can be utilized in several ways. The build currently uses the xpatch task in at least two places: the blocks build, and a new target described below. h3. New build properties _from build.properties_ {{{# ---- Configuration ---- \# define the oracle driver class in web.xml: \#include.driver.oracle=true \# define the postgresql driver class in web.xml: \#include.driver.postgre=true \# define the odbc driver class in web.xml: \#include.driver.odbc=true \# modify web.xml to allow use of ?cocoon-reload=true: \#config.allow-reloads=true \# modify web.xml to enable file uploads: \#config.enable-uploads=true \# currently unused, but useful for distinguishing environments: \#config.live=true }}} These properties enable or disable xpatch files for a growing number of the most common config file modifications. The target files can as always be edited by hand, but the advantage of using a local version of these build properties is that changes survive subsequent Cocoon rebuilds. h3. The custom-conf target The build properties are nice, but what if your desired changes are not triggered by existing properties? A target is defined in the webapp build for 2.1 (new June 2003) which uses the {{xpatch}} task. The webapp target depends on this target, but it can also be run standalone with the command: {{build.bat custom-conf}} The target is defined as below: {{{ }}} As you can see, it reads patch files out of the {{$\{customconf\} }} directory, which is defined by default as {{customconf=$\{src\} /confpatch}}. Of course, this location can be overridden as well. So, to perform configuration changes, simply drop an xpatch file into that directory and run the {{webapp}} or {{custom-conf}} target. Your xpatch files do not have to rely on build properties, but you are free to do so if that fits your need. For example, you may need to define different datasource parameters for your dev, stage and live environments. Using the if-prop attribute in your xpatch file with a build property like {{config.live=true}} or {{config.dev=true}} can accomplish this for you in an easily mantainable way. A powerful way to seed your own Cocoon based application would be to set {{build.webapp}} to point to your project directory, define your xpatch files - including changes necessary to the root sitemap - and run the build. If you set this up carefully, you should be able to build cocoon repeatedly to keep up with new releases without ruining your configuration set up.