Configuring the Cocoon Portal (2.1 legacy document)

Table of contents

Table of contents

Warning:
This document was copied as is from the Cocoon 2.1 documentation, but has not yet been fully reviewed or moved to its new home.

1. Introducing the Cocoon Portal

This document describes the use and configuration of the (new) cocoon portal that you can find in the "portal" block. (Don't mix this with the older portal version that you can find in the "portal-fw" block.)

1.1. Important parts of the Cocoon Portal

TBD

1.2. How is a portal page created by Cocoon?

TBD

1.3. I want to build my own portal! An approach

TBD

2. Configuring the Portal contents

The configuration of a coplet is done in several steps that are outlined in the next chapters.

2.1. Configuring Coplets

Configuring coplets is like defining a class and creating their instances. So in fact, you define the available coplets (= classes) and each portal view gets some instances of these coplets.

2.1.1. Available Coplet Types

Before you can define your available coplets, you have to define the available coplet types, or the so called coplet base data. The current sample contains an XML document for this:

... <coplets> <coplet-base-data id="URICoplet"> <coplet-adapter>uri</coplet-adapter> </coplet-base-data> </coplets> ...

In the example above, we define one coplet type, the *URICoplet*, that uses the *uri coplet adapter*. By this we define a type, that uses URIs to get the content of a coplet.

You can add different coplet types with additional configuration here, but rarely have to touch this file.

2.1.2. Available Coplets

Based on the coplet types, you can define the available coplets in your portal application (= classes). In the example portal an own configuration file contains these so called coplet datas. Here is an excerpt:

... <coplets> <coplet-data id="CZ Weblog" name="standard"> <title>CZ's Weblog</title> <coplet-base-data>URICoplet</coplet-base-data> <attribute> <name>uri</name> <value xsi:type="java:java.lang.String" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">cocoon:/news/liverss?feed=http://radio.we </attribute> <attribute> <name>buffer</name> <value xsi:type="java:java.lang.Boolean" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">true</value> </attribute>

<attribute> <name>error-uri</name> <value xsi:type="java:java.lang.String" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">cocoon:/news/CZ_weblog.rss</value> </attribute> </coplet-data> </coplets> ...

Each coplet data contains a unique id and additional configuration. A required configuration is the underlying coplet base data. In the example above, the *URICoplet* is used here.

The above configured coplet data requires some configuration, like the URI to invoke to fetch the content for this coplet. This configuration is passed in the different attributes you see above. Each attribute has a name and value.

The set of coplet datas defines the set of available coplets a user can choose from. If a user chooses to view a coplet, an instance of this coplet data is created. If, e.g. the user chooses the same coplet twice, two instances are created. This is useful for configurable coplets where the user can choose the same coplet with different configurations.

2.1.3. Selected Coplets

The selected coplets are described by the set of coplet instance datas.

... <coplets> <coplet-instance-data id="CZ Weblog-1" name="standard"> <coplet-data>CZ Weblog</coplet-data> </coplet-instance-data> </coplets> ...

The coplet instance data refers to its coplet data by specifying the unique ID. The instance itself has a unique ID as well that is referenced from the portal view.

In addition, a coplet instance data could have own configuration information.

2.2. Configuring the arrangement of the defined Coplets

The portal view defines the ordering and arrangement of the coplets. This view is defined in a hierarchical manner by nesting layout objects. At each place, a coplet is located, a reference to a coplet instance data is included.

The Cocoon portal provides several predefined layout elements you can use for your portal view to create a nice layout:

- row - a row of items
- column - a column of items
- tab - a tab

These are the "high-level" objects, you can use to define your structure. You can nest them in any order to create a complex layout. The layout is defined in an XML document as well, so let's have a look at an example:

... <composite-layout name="row"> <item> <coplet-layout name="coplet"> <coplet-instance-data>Portal-Intro-1</coplet-instance-data> </coplet-layout> </item> <item> <coplet-layout name="coplet"> <coplet-instance-data>Portal-Bottom-1</coplet-instance-data> </coplet-layout> </item> </composite-layout> ...

In the example above, we define a row containing two coplets. This is done by selecting the row layout and defining the childs (or items) of this layout. In this case the items are two new layout objects, the coplet layouts that can contain a coplet. The coplet layout has a reference to the coplet instance data.

2.3. The Rendering Process

Each layout object has a defined renderer that is used to render this layout object. You can find the renderers in the cocoon.xconf. Each renderer has a unique name that is used to identify this renderer.

A central component, the layout factory (configured in the cocoon.xconf as well), contains a list of all available layout objects, like the row, the column etc. The configuration for each layout object contains also the corresponding renderer information. So, here is the configuration which renderer will be used to render the layout object.

A renderer itself can be configured in various ways. The portal engine uses so called aspects (don't mess them with AOP), that are used to enhance to features of renderer, allowing - simplifying - a multiple inheritance which is not possible in Java. Have a look at the cocoon.xconf for the different renderer configurations.

3. Create a new skin for your portal

This section will explain the concepts of the portal layout, considering the skins/basic/ skin provided with cocoon, and will describe how to create a new skin by extending the existing stylesheets.

The skin path can be changed in the portal's sitemap. There is a global-variable specifying the path to the skin folder.

The basic cocoon portal skin is a nice and simple example how to visualize a portal. There are several elements that allow to customize the look and feel of the portal. A portal with the basic skin consists of

- a *header*
- a *tab row*
- a *content section* containing the coplet windows
- and a *footer*

Parts of the portal

The tab row is actually a part of the content section. As well, a tab row can be provided to any coplet window.

3.1. The skin's stylesheets

If we take a look at the skins/basic/styles directory, we find a number of stylesheets:

- portal-page.xsl: Creates final HTML page
- tab.xsl: layout of the tab row.
- window.xsl: coplet window layout
- column.xsl: layout of a column
- row.xsl: layout of a row
- login-page.xsl: layout of the login page

The window.xsl stylesheet determines the layout of a coplet window. Normally, a coplet window will contain a header row with a title and buttons like minimize, close, etc.

These coplet windows are arranged in rows and columns to create the arrangement typical for portals. There can be several rows per column and several columns in the content section. So the thinking is a little different than in usual HTML tables.

The content section or content row usually has a tab row located at the top and the coplet windows are arranged below. The layout of the tabs is specified in the tab.xsl stylesheet.

The portal-page.xsl stylesheet encapsulates the content section with the tab row and allows to define a page header and a footer. This is probably the first stylesheet we want to take a closer look at.

The tab.xsl, column.xsl, row.xsl and window.xsl stylesheets are used by the portal components directly and won't be found anywhere in the sitemap. The cocoon.xconf defines which stylesheets will be used by the portal.

3.2. The portal-page.xsl

Here is the place to change the design of the header and footer row. This stylesheet is used to construct the final HTML page, which displays the portal. The code sample here displays the main template match node of the stylesheet.

... <xsl:template match="/"> <html> <head> <link type="text/css" rel="stylesheet" href="page.css"/> </head> <body> <table bgColor="#ffffff" border="0" cellPadding="0" cellSpacing="0" width="100%"> <tbody> <!-- header row --> <tr> <td colspan="2"> <table border="2" cellPadding="0" cellSpacing="0" width="100%"> <tbody> <tr> <td colspan="2" noWrap="" height="10" bgcolor="#DDDDDD"> </td> </tr> <tr> <td bgcolor="#CCCCCC" height="100" align="center" valign="middle" width="100%"> Cocoon Portal </td> </tr> <tr> <td colspan="2" noWrap="" height="10" bgcolor="#DDDDDD"> </td> </tr> </tbody> </table> </td> </tr> <!-- content/tab row --> <tr> <td> <xsl:apply-templates/> </td> </tr> <!-- footer row --> <tr> <td colspan="2"> <table border="2" cellPadding="0" cellSpacing="0" width="100%"> <tbody> <tr> <td colspan="2" noWrap="" height="10" bgcolor="#DDDDDD"> </td> </tr> <tr> <td colspan="2" noWrap="" height="30" bgcolor="#CCCCCC"> </td> </tr> </tbody> </table> </td> </tr> </tbody> </table> </body> </html> </xsl:template> ...

3.3. The tab.xsl

From the portal-page.xsl stylesheet, we will now move upwards in the XSL transformation steps and take a look at the stylesheet that was processed before, the tab.xsl.

Again, this source snippet shows the main template match node of the stylesheet:

... <!-- Process a tab --> <xsl:template match="tab-layout"> <!-- ~~~~~ Begin body table ~~~~~ --> <table border="2" cellpadding="0" cellspacing="0" width="100%"> <!-- ~~~~~ Begin tab row ~~~~~ --> <tr> <td> <table summary="tab bar" border="2" cellpadding="0" cellspacing="0" width="100%"> <tr vAlign="top"> <xsl:for-each select="named-item"> <xsl:choose> <xsl:when test="@selected"> <!-- ~~~~~ begin selected tab ~~~~~ --> <td valign="middle" bgcolor="#DDDDDD">   <xsl:value-of select="@name"/>   </td> <!-- ~~~~~ end selected tab ~~~~~ --> </xsl:when> <xsl:otherwise> <!-- ~~~~~ begin non selected tab ~~~~~ --> <td valign="middle" bgcolor="#CCCCCC" > <xsl:value-of select="@name"/> </td> <!-- ~~~~~ end non selected tab ~~~~~ --> </xsl:otherwise> </xsl:choose> </xsl:for-each> <!-- ~~~~~ last "blank" tab ~~~~~ --> <td width="99%" bgcolor="#CCCCCC" align="right"> </td> </tr> </table> </td> </tr> <!-- ~~~~~ End tab row ~~~~~ --> <!-- ~~~~~ Begin content row ~~~~~ --> <tr> <td bgcolor="#FFFFFF"> <xsl:apply-templates/> </td> </tr> <!-- ~~~~~ End content row ~~~~~ --> </table> </xsl:template> ...

The first row that is created here contains the definition of the tabs. The <xsl:choose> element differentiates between the currently selected tab and all other tabs. The @selected attribute is generated by the portal and can be accessed as shown above. As well, the portal provides the tab's @parameter (usually the tab's link) and @name attributes.

Depending on the configuration of the portal, it is possible that tabbed areas are nested

inside each other. So be careful how a tab row might look in the middle of another content section.

Nice looking tabs can become pretty complex, take a look at the tab.xml stylesheet in the skins/common/ skin to see another example.

Below the tabs, another table cell is defined that will be filled with the contents of the tabbed area. This content will have been processed by the columns.xsl stylesheet before - and before that by the row.xsl stylesheet.

3.4. The column.xsl and row.xsl

The column.xsl and row.xsl stylesheets define the look of the tables in which the coplet windows will be placed. Usually, nothing exciting happens in these stylesheets. Here is a listing of the important parts, just to give a complete overview.

The main template match node of the column.xsl stylesheet:

... <!-- Process a Column --> <xsl:template match="column-layout"> ... <table border="{$border}" cellSpacing="0" cellpadding="0" width="100%"> <xsl:if test="@bgcolor"> <xsl:attribute name="bgcolor"> <xsl:value-of select="@bgcolor" /> </xsl:attribute> </xsl:if> <tr vAlign="top"> <xsl:for-each select="item"> <td> <xsl:if test="@bgcolor"> <xsl:attribute name="bgcolor"> <xsl:value-of select="@bgcolor" /> </xsl:attribute> </xsl:if> <xsl:if test="@width"> <xsl:attribute name="width"> <xsl:value-of select="@width" /> </xsl:attribute> </xsl:if> <xsl:apply-templates /> </td> </xsl:for-each> </tr> </table> </xsl:template> ...

The main template match node of the row.xsl stylesheet:

... <!-- Process a row --> <xsl:template match="row-layout"> ... <table border="{$border}" cellSpacing="10" width="100%"> <xsl:if test="@bgcolor"> <xsl:attribute name="bgcolor"> <xsl:value-of select="@bgcolor" /> </xsl:attribute> </xsl:if> <xsl:for-each select="item"> <tr vAlign="top"> <xsl:if test="@bgcolor"> <xsl:attribute name="bgcolor"> <xsl:value-of select="@bgcolor" /> </xsl:attribute> </xsl:if> <td> <xsl:apply-templates /> </td> </tr> </xsl:for-each> </table> </xsl:template> ...

3.5. The window.xsl

The window.xsl stylesheet determines the design of the coplet windows and probably takes the most design effort compared to the other stylesheets. A coplet window consists of a table header with the window title and a number of buttons like close, minimize, maximize etc. The basic skin provides some images in the images/ subfolder. The rest of the window will be filled with the coplet content, depending on the configuration of the coplet.

A slightly more complex example can be found in the skins/common/ skin.

The listing below shows the main template match node:

... <xsl:template match="window"> ... <table border="2" cellSpacing="0" cellpadding="0" width="100%"> <tr vAlign="top"> <td bgColor="{$bgColor}" valign="middle"> <xsl:attribute name="color">#ffffff</xsl:attribute> <xsl:attribute name="face">Arial</xsl:attribute> <xsl:attribute name="size">2</xsl:attribute> <xsl:choose> <xsl:when test="@title"> <xsl:value-of select="@title"/> </xsl:when> <xsl:otherwise> <xsl:value-of select="title"/> </xsl:otherwise> </xsl:choose> </td> <td align="right" bgColor="{$bgColor}"> <xsl:if test="fullscreen-uri">   </xsl:if> <xsl:if test="maxpage-uri"> <img src="show.gif" border="0" alt="Max

Page"/> </xsl:if> <xsl:if test="maximize-uri">   </xsl:if> <xsl:if test="minimize-uri">   </xsl:if> <xsl:if test="remove-uri">   </xsl:if> </td> </tr> <tr> <td colSpan="2"> <xsl:apply-templates select="content"/> </td> </tr> </table> </xsl:template> ...

4. Further topics

1. Comments

add your comments