# Webapps Developer Documentation (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. Overview

This section contains several documents about developing a portal with Cocoon.

The portal framework is a stable portal framework that uses the authentication framework. It can be used to quickly develop portal applications.

The new portal engine is a new implementation of a portal engine which focuses on more flexibility and ease-of-use. In addition it supports the JSR-168. The current state of this engine is *alpha* which means that the engine can still change in some aspects.

## 2. Features

The portal framework is a portal server that runs inside Cocoon - or to be more precise inside the Cocoon servlet. It contains a portlet container that is called coplet container. Coplet stands for *Cocoon Portlet* and is the Cocoon equivalent to portlet.

Due to the highly extensible nature of Cocoon, the portal is configurable and extensible as well and provides many hooks and switches to easily adapt the portal for specific needs. As the portal is integrated in Cocoon it has the advantage that all features of Cocoon can easily be used. Cocoon is very strong in fetching data from various source and delivering it in the various output formats requested by different clients (like HTML, WML, PDF etc.).

With the latest features, like the form handling framework, the authentication framework and the flow concept, it is very easy to develop and maintain complex applications with Cocoon. And these applications can in turn be viewed as portlets inside a portal. But even by just using the flexible pipeline concept from Cocoon it's possible to develop complex and nice looking portals.

The portal view is described by an XML document. This document contains the layout and the ordering of the selected portlets. The layout is a tree-like structure, it can for example contain rows, columns or tabs in any ordering or nesting. This allows the description of the portal view in a parent-child relationship. Even complex layouts with portlets spanning several columns or tabs inside a single column are possible.

For each portlet a placeholder is defined in the XML document. Special components, so called renderers, are used to generate (or render) the portal view. Each layout object (each row, column etc.) has an configurable renderer associated that generates the output of the portal for this layout object. For example a renderer for a row could create the required HTML tags.

The renderers can either directly generate the required format (like HTML) or they can create an XML document that is later transformed by Cocoon to the format requested by the client using a stylesheet. This depends on the requirements of the application and how it is developed. The output of all involved renderers is aggregated and this creates the (layout) document for the portal view.

After the layout is rendered, the placeholders for the different portlets are replaced with the content from the portlets. Therefore the portlet container is asked to deliver the content of a given portlet and this content is inserted at the correct places in the portal view.

In Cocoon a portlet can have different types:

- Static: Reading a static file, like an HTML document or a PDF file

- URI: Reads information using a URI
- Pipeline: Uses the Cocoon pipelining concept to dynamically generate the content
- Custom Types: If the need arises, it's possible to develop a custom type and use it.

In Cocoon, the use of the pipeline type is possibly the most common one right now.

The complete configuration (the portal view, the available portlets, their settings) is done using XML documents. So it is possible to develop a portal application by just changing the configuration without any Java coding. However if the need arises nearly any part of the portal can be changed/extended by writing an additional component (in Java) and plugging it into the portal engine (by configuration). For example, one component - the profile manager - is responsible for getting the profile of the current user (the user associated to the current request). A profile contains the portal view (layout, ordering etc.) and the configuration of the different portlets for this user.

There is one implementation that reads this profile from any database accessible from Cocoon on a per user base. The profile can for example be stored in a database, on a file system or in a WebDAV repository. Therefore every user has their own portal view and can customize this to their needs. Another implementation is more *static*. This means every user gets the same layout and the same portlets. So in fact the portal can be used for web pages that have a portal like structure but don't have any personalization at all.

All changes that may occur inside a portal are propagated through a flexible event management. The event handling follows the usual publisher/subscribe pattern. A component, for example a portlet, that is interested in a special event can subscribe for all events of this kind. The component is notified when such an event occurs and can react on this event by changing its status, sending new events or whatever is appropriate.

This event handling is for example used to change preferences, to change the portal profile or to change the status of a portlet. For example, events can be sent to add a new portlet to the portal view or to minimize a portlet. In addition this event handling can be used for communication between portlets. Imagine a portlet where the user chooses a city he wants to travel to. This selection is broadcast to other portlets using the event handling. Another portlet, displaying the current weather information of a city, receives this event and displays the weather information for the city selected in a different portlet. This is a very simple example for inter portlet communication but it shows the potential.

As Cocoon and therefore the portal as well is based on the request response cycle, a request for displaying the portal view triggers two tasks that are executed one after the other. The first task is the event handling phase. In this phase all events that are triggered by the request are processed and published. For example if the user clicks on the minimize button of a portlet, a request is sent to Cocoon and a minimize event for the portlet is published triggering the status change by some receiver of the event.

This processing of the request information can trigger new events that are published as well and so on. When all events are published and consumed, the second task is executed that actually renders the portal view as described above. And the portal view is sent as a response back to the client. On the client the user can navigate through the portal and trigger some action like minimizing, enlarging a single portlet to a full screen mode temporarily hiding the other portlets and so on.

Some installations of the Cocoon portal are going far beyond these usual use cases. They use the included components for example to create a view to a complete web application running on a different server. One single portlet shows the application and a click/action in this portlet triggers a request to Cocoon that is forwarded to the distant application and the new *state* of the application is then displayed in the portlet again.

But by using Cocoon this can even go further: stylesheets can be used to change the layout of the integrated application. So you can give an application a totally different lock and feel using a portlet.

And - of course - SoC (separation of concerns) applies to developing portals with Cocoon as well, so you can develop your portal in groups each group concentrating on their concern.

## 1. Comments

add your comments