# Advanced Control Flow (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. Using Cocoon's Control Flow

The general flow of actions in an application which uses the control flow is as described below.

The request is received by Cocoon and passed to the sitemap for processing. In the sitemap, you can do two things to pass the control to the Control Flow layer:

- You can invoke a JavaScript top-level function to start processing a logically grouped sequences of pages. Each time a response page is being sent back to the client browser from this function, the processing of the JavaScript code stops at the point the page is sent back, and the HTTP request finishes. Through the magic of continuations, the execution state is saved in a continuation object. Each continuation is given a unique string id, which could be embedded in generated page, so that you can restart the saved computation later on.
- To invoke a top level JavaScript function in the Control Flow, you use the <map:call function="function-name"/> construction.
- To restart the computation of a previously stopped function, you use the <map:call continuation="..."/> construction. This restarts the computation saved in a continuation object identified by the string value of the continuation attribute. This value could be extracted in the sitemap from the requested URL, from a POST or GET parameter etc. When the computation stored in the continuation object is restarted, it appears as if nothing happened, all the local and global variables have exactly the same values as they had when the computation was stopped.

Once the JavaScript function in the control layer is restarted, you're effectively inside the Control Flow. Here you have access to the request parameters, and to the business logic objects. The controller script takes the appropriate actions to invoke the business logic, usually written in Java, creating objects, setting various values on them etc...

When the business logic is invoked, you're inside the Model. The business logic takes whatever actions are needed, accessing a database, making a SOAP request to a Web service etc. When this logic finishes, the program control goes back to the Control Flow.

Once here, the Control Flow has to decide which page needs to be sent back to the client browser. To do this, the script can invoke one of the cocoon.sendPageAndWait() or cocoon.sendPage() functions. These functions take two parameters, the relative URL of the page to be sent back to the client, and a context object which can be accessed inside this page to extract various values and place them in the generated page.

The second argument to cocoon.sendPageAndWait() and cocoon.sendPage() is a context object, which can be a simple dictionary with values that need to be displayed by the View. More generally any Java or JavaScript object can be passed here, as long as the necessary get methods for the important values are provided.

The page specified by the URL is processed by the sitemap, using the normal sitemap rules. The simplest case is a generator followed by an XSLT transformation and a serializer. This page generation is part of the View layer. To process a page you can make use of several Cocoon generators to retrieve values from the context objects passed by the Control Flow.

Going back to the cocoon.sendPageAndWait() and sendPage() functions, there is a big difference between them. The first function will send the response back to the client browser, and will stop the processing of the JavaScript script by saving it into a continuation object. The other function,

cocoon.sendPage() will send the response, but it will not stop the computation. This is useful for example when you need to exit a top-level JavaScript function invoked with <map:call function="..."/>.

The above explains how MVC could be really achieved in Cocoon with the control flow layer. Note that there is no direct communication between Model and View, everything is directed by the Control Flow by passing to View a context object constructed from Model data.

### 1.1. Basic usage

As hinted in the previous section, an application using Cocoon's MVC approach is composed of three layers:

- A JavaScript controller which implements the interaction with the client
- The business logic model which implements your application
- The page templates, which describe the content of the pages, and XSLT stylesheets which describe the look of the content.

In more complex applications, the flow of pages can be thought of smaller sequences of pages which are composed together. The natural analogy is to describe these sequences in separate JavaScript functions, which can then be called either from the sitemap, can call each other freely.

An example of such an application is the user login and preferences sample

This application is composed of four top-level JavaScript functions:

- login,
- registerUser,
- edit and
- logout.

The entry level point in the application can be any of these functions, but in order for a user to use the application, (s)he must login first. Once the user logs in, we want to maintain the Java User object which represents the user between top-level function invocations.

Even if you don't need complex control flow in your application, you may still choose to use the MVC pattern described above. You can have top- level JavaScript functions which obtain the request parameters, invoke the business logic and then call cocoon.sendPage() to generate a response page and return from the computation. Since there's no continuation object being created by this function, and no global scope being saved, there's no memory resource being eaten. The approach provides a clean way of separating logic and content, and makes things easy to follow, since you have to look at a single script to understand what's going on.

## 1. Comments

add your comments