

# Cocoon Forms: A Simple Example (2.1 legacy document)

## Table of contents

1 Comments.....	5
-----------------	---

## Table of contents

1 A simple CForms example.....	3
2 Create a form definition file.....	3
3 Create a template file for the Forms Template Transformer.....	3
4 Write a bit of flowscript.....	4
5 Add some pipelines to the sitemap.....	5
6 Next steps.....	5

**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. A simple CForms example

In this example we will show how to create a simple registration form using CForms and flowscript. We will follow to following steps:

1. Create a form definition file
2. Create a template file for the Forms Template Transformer
3. Write a bit of flowscript
4. Add some pipelines to the sitemap

Here is a screenshot of the form we're going to create:

Screenshot of the sample we're going to create.

## 2. Create a form definition file

Below the form definition file is displayed. This lists all the widgets in the form, together with their configuration information.

```
<fd:form xmlns:fd="http://apache.org/cocoon/forms/1.0#definition"> <fd:widgets> <fd:field
id="name" required="true"> <fd:label>Name:</fd:label> <fd:datatype base="string"/>
<fd:validation> <fd:length min="2"/> </fd:validation> </fd:field> <fd:field id="email"
required="true"> <fd:label>Email address:</fd:label> <fd:datatype base="string"/>
<fd:validation> <fd:email/> </fd:validation> </fd:field> <fd:field id="age"> <fd:label>Your
age:</fd:label> <fd:datatype base="long"/> <fd:validation> <fd:range min="0" max="150"/>
</fd:validation> </fd:field> <fd:field id="password" required="true">
<fd:label>Password:</fd:label> <fd:datatype base="string"/> <fd:validation> <fd:length
min="5" max="20"/> </fd:validation> </fd:field> <fd:field id="confirmPassword"
required="true"> <fd:label>Re-enter password:</fd:label> <fd:datatype base="string"/>
<fd:validation> <fd:assert test="password = confirmPassword"> <fd:failmessage>The two
passwords are not equal.</fd:failmessage> </fd:assert> </fd:validation> </fd:field>
<fd:booleanfield id="spam"> <fd:label>Send me spam</fd:label> </fd:booleanfield>
</fd:widgets> </fd:form>
```

All elements are in the Forms Definition namespace: **fd**.

Every definition file has a **<fd:form>** element as the root element.

The child widgets of the form are defined inside the **<fd:widgets>** element. As you can see, most of the widgets are field widgets. The field widget is the most important widget in CForms. It is very flexible because it can be associated with different datatypes and with a selection list. See the reference docs for more information on this and other widgets.

A nice feature is that the **fd:label** tags can contain mixed content. On the one hand, this can be used to provide rich formatting in the label. But it also enables you to put i18n-elements in there, to be interpreted by the I18nTransformer. This way, internationalisation is done using standard Cocoon techniques.

## 3. Create a template file for the Forms Template Transformer

Here's the template for our registration form example:

```
<html xmlns:ft="http://apache.org/cocoon/forms/1.0#template"
xmlns:fi="http://apache.org/cocoon/forms/1.0#instance"> <head> <title>Registration
form</title> </head> <body> <h1>Registration</h1> <ft:form-template
action="#{$continuation/id}.continue" method="POST"> <ft:widget-label id="name"/>
<ft:widget id="name"/> <br/> <ft:widget-label id="email"/> <ft:widget id="email"/> <br/>
<ft:widget-label id="age"/> <ft:widget id="age"/> <br/> <ft:widget-label id="password"/>
<ft:widget id="password"> <fi:styling type="password"/> </ft:widget> <br/> <ft:widget-label
id="confirmPassword"/> <ft:widget id="confirmPassword"> <fi:styling type="password"/>
</ft:widget> <br/> <ft:widget id="spam"/> <ft:widget-label id="spam"/> <br/> <input
type="submit"/> </ft:form-template> </body> </html>
```

The CForms-specific elements here are in the "Forms Template" namespace: **ft**.

The `<ft:widget-label>` tag will cause the label of a widget to be inserted at the location of the tag. The `<ft:widget>` tag will cause the XML representation of a widget to be inserted at the location of that tag. The inserted XML will be in the "Forms Instance" namespace: **fi**.

The XML representation of the widget will then be translated to HTML by an XSLT stylesheet (forms-samples-styling.xsl in our case -- see sitemap snippets below). This XSLT only has to handle individual widgets, and not the page as a whole, and is thus not specific for one form but can be reused across forms.

For certain widgets it may be necessary to provide extra presentation hints, such as the width of a text box, the style of a selection list (drop down, radio buttons, ...) or class and style attribute values. This can be done by putting a `fi:styling` element inside the `ft:widget` element. This element is in the `fi` namespace because it will be copied literally. The attributes and/or content of the `fi:styling` element depend on what is supported by the particular stylesheet used.

As an alternative to the template approach, you could also use the FormsGenerator, which will generate an XML representation of the whole form, and style that with a custom-written XSLT. For most users we recommend the template approach though.

## 4. Write a bit of flowscript

Flowscript is Cocoon's solution to handling the flow of a web interaction. It is based on the concept of continuations. If you don't know yet about continuations and flowscript, [learn about it here](#).

Here's the flowscript for our example, registration.js:

```
cocoon.load("resource://org/apache/cocoon/forms/flow/javascript/Form.js"); function
registration() { var form = new Form("forms/registration.xml");
form.showForm("registration-display-pipeline"); var model = form.getModel(); var bizdata = {
"username" : model.name } cocoon.sendPage("registration-success-pipeline", bizdata); } This
sample still shows the "old" flowscript API. Will be updated eventually.
```

The flowscript works as follows:

First we create a Form object, specifying the form definition file to be used. The Form object is actually a javascript wrapper around the "real" Java form instance object.

Then the showForm function is called on the form object. This will (re)display the form to the user until validation of the form succeeded. As parameter to the showForm function, we pass the sitemap pipeline to be used to display the form.

Finally we get some data from the form (the entered name), and call a sitemap pipeline to display this data. This pipeline is based on the JXTemplate generator.

## 5. Add some pipelines to the sitemap

First of all, do not forget to register the registration.js file in the map:flow section of the sitemap, as follows:

```
<map:flow language="javascript"> <map:script src="flow/registration.js"/> </map:flow>
```

And here are the pipelines we need:

```
<map:match pattern="registration"> <map:call function="registration"/> </map:match>
<map:match pattern="*.continue"> <map:call continuation="{1}"/> </map:match>
<map:match pattern="registration-display-pipeline"> <map:generate
src="forms/registration_template.xml"/> <map:transform type="forms"/> <map:transform
type="i18n"> <map:parameter name="locale" value="en-US"/> </map:transform>
<map:transform src="resources/forms-samples-styling.xsl"/> <map:serialize/> </map:match>
<map:match pattern="registration-success-pipeline"> <map:generate type="jx"
src="forms/registration_success.jx"/> <map:serialize/> </map:match>
```

The first two are for managing the flowscript: when someone hits the registration URL, we call the registration function in our flowscript.

When a form is submitted, it will be matched by the second matcher, \*.continue, which will continue the execution of the flowscript.

The third matcher is for displaying the form, and uses the Forms Template Transformer.

The fourth pipeline is for showing the "success" page using the JXTemplate generator, here is the contents of the registration\_success.jx page:

```
<html> <head> <title>Registration successful</title> </head> <body> Registration was
successful for ${username}! </body> </html>
```

## 6. Next steps

The example we've studied here is quite simple. To have a feel for the power of CForms, take a look at the examples included in the Forms block.

### 1. Comments

add your comments