Cocoon Forms: Template Transformer (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. Introduction

The FormsTemplateTransformer (simply "forms transformer" from now on) makes it possible to define the layout for your form without having to write a separate XSLT for each form. If you prefer to do everything with XSLT, you have also the option of using the FormsGenerator. In general we recommend to use the forms transformer though.

The basic principle is that the forms transformer will replace any <ft:widget id="xyz"/> elements it encounters by the XML representation of the corresponding widgets. These ft:widget elements can be embedded in e.g. a HTML layout. So after passing this template through the forms transformer you'll end up with HTML with here and there a piece of XML describing a widget. This XML description contains all state information of the widget: its value, validation errors, selection-list data if any, and so on. These widget-XML-descriptions will then typically be translated to HTML by an XSLT. This XSLT is then however not form-specific, as it simply needs to know how to translate individual widgets to HTML, and does not have to create the complete page layout. CForms contains just such an XSLT so you don't have to write it yourself (except if you need to do heavy customisation). The image below illustrates this process.

Forms Template Transformer illustration.

1.1. Where the forms transformer looks for the form instance object

Each time the forms transformer encounters a ft:form-template element (see further on), it will try to retrieve a CForms form instance object. It looks for it in the following locations:

1. if the ft:form-template element has a location attribute, then the value of that attribute will be evaluated as a JXPath expression. The result of this expression should be the form object.
2. if a parameter called "attribute-name" was supplied to the forms transformer in the sitemap, then the forms transformer will try to find the form in the request attribute with that name. (request attributes are a temporary storage area that exists for the duration of one request and is often used to communicate objects between different sitemap components such as actions and transformers)
3. finally, the forms transformer will look if a CForms form was supplied from a flowscript using the key "CocoonFormsInstance".

If the form is not found at any of these locations, an exception is thrown.

2. Forms transformer element reference

The elements to which the forms transformer reacts are all in the "ft" (Forms Template) namespace, which is identified by the following URI:

http://apache.org/cocoon/forms/1.0#template

These will generally be replaced by elements in the "fi" (Forms Instance) namespace, which is identified by the following URI:

http://apache.org/cocoon/forms/1.0#instance

2.1. ft:form-template

The ft:form-template element is always required; all other ft:* elements should occur inside a ft:form-template element. As described earlier, when the forms transformer encounters the

ft:form-template element it will try to look up the form instance object.

ft:form-template elements may not be nested.

The ft:form-template will by default copy over all attributes appearing on it, except for one attribute called "location", and it will also take special care of the action attribute.

The **action** attribute can contain JXPath expressions. As with the JXTemplateGenerator, these JXPath expressions must be embedded inside #{ and }. By allowing the use of JXPath expressions, you can embed dynamic data in the action attribute. One of the most common uses is to embed the continuation id (if you're using flowscript), for example:

```
<ft:form-template action="#{$cocoon/continuation/id}.continue" ...
```

The following objects are available in the JXPath context via the cocoon object: continuation, requests, session and parameters. The context of the JXPath expression is the map passed on from the flowscript (if any).

The **location** attribute, if present, is used to retrieve the form instance object. The value of the location attribute should be a JXPath expression, and is executed in the same context as the JXPath expressions embedded in the action attribute.

For example, if your form object is stored in the session using the key "myform", then following expression in the location attribute can be used to retrieve it:

```
<ft:form-template location="getAttribute($session, 'myform')" ...
```

If you'd like to retrieve the key "myform" from a parameters specified in the sitemap, say one called "sessionattr", then the following can be used:

```
<ft:form-template location="getAttribute($session, getParameter($parameters, 'sessionattr'))"
...
```

As mentioned before, ft:form-template elements cannot be nested, but you can have multiple ft:form-template elements on one page. Together with the location attribute, this can be used to handle multiple forms occuring on one template.

2.2. ft:widget

The ft:widget element is replaced by the forms transformer by the XML representation of a widget. Which widget is specified by the id attribute. The ft:widget element can contain a fi:styling element containing parameters to influence the styling process (the XSLT). The forms transformer will simply copy the fi:styling element over to its output.

For example:

```
<ft:widget id="pass"> <fi:styling type="password"/> <ft:widget/>
```

will be replaced by:

```
<fi:field id="pass"> [... label, validation errors, ...] <fi:styling type="password"/> </fi:field>
```

Note: it is not recommended to use the older approach, i.e. without the fi:styling element, anymore, since support for it will be dropped at some point in the future.

2.3. ft:widget-label

The ft:widget-label element will be replaced by the forms transformer with the label of a certain widget (specified by an id attribute). The label will not be wrapped in another element.

2.4. ft:continuation-id

The ft:continuation-id element will be replaced by the forms transformer by:

<fi:continuation-id> ID-of-the current-continuation </fi:continuation-id>

This might be useful for embedding the continuation ID in a hidden form field, for example.

2.5. Working with repeaters: ft:repeater-widget, ft:repeater-widget-label, ft:repeater-size

The ft:repeater-widget element is similar to the ft:widget element but provides special treatment for repeaters. The content of the ft:repeater-widget element will be used as a template to generate each of the rows of the repeater.

The ft:repeater-widget-label element is used to retrieve the label of a widget contained by a repeater. It requires two attributes: id (identifying the repeater) and widget-id (identifying the widget in the repeater).

The ft:repeater-size element inserts an element <fi:repeater-size id="..." size="..."/> containing the size (number of rows) of the repeater.

For an example of how this all fits together, take a look at the samples included in the forms block.

1. Comments

add your comments