# WoodyIntro

## Woody: Cocoon Forms

Three important aspects of web application development are publishing, flow control and form handling. Traditionally, Cocoon has been very strong in the publishing aspect with its XML-transforming pipelines. With the advent of flowscript and its continuations concept, handling transitions between pages has become dead easy. And now Woody ("Cocoon Forms") fills the form handling gap.

In Woody, the model of a form is defined by a **form definition**. This is an XML file describing the structure of the form, by declaring the widgets it consists of. This file doesn't contain any presentational information. Based on the form definition, a **form instance** can be created. This is a specific instance of the form that will hold actual data. The form defintion is to the form instance what a Java class is to a Java object, or what an XML Schema is to an XML document.

Since the form definition is simply described in an XML file, this means you can create forms without any knowledge of Java. On the other hand, hardcore Java programmers do not have to fear that Woody is not for them.

As said before, a form consists of a number of widgets. A **widget** is an object that knows how to read its state from a Request object, how to validate itself, and can generate an XML representation of itself. A widget can remember its state itself, so unlike Struts, you do not have to write a form bean for that purpose. A widget can hold strongly typed data. For example, you can indicate that a field should contain a date. If you do this, you can be assured that after the form is successfully validated, and you retrieve its value, you will get a Date object. So your own business logic doesn't need to care about converting strings to other types, and all the locale and formatting related issues of this.

Woody contains a flexible set of widgets that should cover most needs (but, like everything in Woody, it is extensible with your own types). One special widget is the repeater widget, which "repeats" a number of other widgets multiple times, as is needed to generate table-like structures. Widgets can thus have child widgets, so a form instance is effectively a **widget tree**.

For presenting the form, we have a solution that's both easy and powerful. Basically, you write a **template file** (e.g. an XHTML file, but this could be any markup) and on the places you want a widget to appear, you insert a special tag referencing that widget. After processing by the Woody Template Transformer, these tags will be replaced by the XML representation of the widget, which contains all state information of the widget (its value, validation errors, ...). These bits of XML can then be transformed to plain HTML by an XSLT. Note that this XSLT only has to know how to style certain kinds of widgets, but not individual widget instances itself. Thus one template in this XSLT can style all widgets of a certain type that appear on all forms you might have. Cocoon includes a flexible, WoodyXSLT that covers most needs.

Lets look at a typical scenario to see how things fit together:
http://outerthought.net/~bruno/images/woody_overview.png

- Initially, the controller logic asks the FormManager component to create a form instance based on a form definition (form definitions are cached, so creating an instance is very fast).
- The controller can then optionally pre-populate this form object with some data. To fill the form with data from a bean or XML document, a Woody Binding is available.
- Then the form is shown by calling a pipeline.
- When the form is submitted, the controller will let the form instance object process the request, so that all widgets can read their value from the request. Some might generate events, which will be handled by event handlers. Validation of the widget tree is also triggered, whereby all widgets will validate themselves based on validation rules described in the form definition. The controller can afterwards perform application-specific validation logic.
- If there were validation errors, the form will be redisplayed. Otherwise the controller will decide what's the next step, for example saving the form data back to a bean or calling some backend process using data from the form.

So to summarize some important points:

- by having a separate form definition and form template, strong separation of concerns is achieved, both conceptually and skill-wise. The same form definition can be used with different templates.
- it is not required to write Java classes for a form, nor do you need to be Java programmer to benifit from Woody.
- strong datatyping on the form definition level makes it so you don't have to worry about that yourself.
- Woody isn't concerned with page flow; this is left to other solutions such as flowscript (for people who don't like flowscript: Woody doesn't depend on flowscript).
- Woody integrates nicely within the rest of Cocoon.

**Next steps:** run through the WoodySample to get a feeling of form definitions and form templates. Also have a look at the examples included with Cocoon.