Cocoon Forms: Event Handling (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. Intro

Some types of widgets can emit events. For example, the action widget produces ActionEvents and the field widget produces ValueChangedEvents. Next to these events, there are also ProcessingPhaseEvents, fired in between the various phases of the processing of a request.

Handling events can be done in three ways:

- by defining event listeners in the form definition (as child of wd:on-action for the action widget, or wd:on-value-changed for the field widget, ...).
- by adding event listeners dynamically on widget instances.
- by registering a FormHandler on the Form object. This FormHandler will receive all events from all widgets.

2. When are events processed? (Request processing phases)

To answer the question "When are events processed?", we have to look a bit deeper into how a form request is handled. This is separated in a couple of phases, more specifically the following ones:

- Any outstanding events are broadcasted to the event listeners.
  The reason this is done is because events might have been collected while the form was loaded with values by the binding framework.
- ProcessingPhaseListeners are informed that the LOAD_MODEL phase has ended.
- All widgets in the widget tree read their value from the request. If a widget decides it has to produce an event, it is added to a global (i.e. form-level) list (but not yet executed).
- Once all widgets had the opportunity to read their value from the request, the events are broadcasted to the event listeners. This assures that event listeners have access to the values of all widgets in the tree.
- ProcessingPhaseListeners are informed that the READ_FROM_REQUEST phase has ended.
- It is possible that processing ends now. This usually happens when an action widget has caused an event.
- All widgets in the widget tree validate themselves.
- ProcessingPhaseListeners are informed that the VALIDATE phase has ended.

3. Recursive event loops

Event listeners themselves might call methods on widgets which cause new events to be generated. You have to be careful not to cause recursive event loops by doing this.

For example, calling setValue on a widget in a ValueChangedEvent caused by that widget will schedule a new ValueChangedEvent, which will then again cause the execution of the event listener which will then again call setValue and thus again cause a new event to be generated, and so on.

4. Defining event handlers in the form definition

Event handlers can be specified as part of the form definition, as child of the various wd:on-xxx elements, such as wd:on-action for the action widget.

Event handlers can be written in either javascript or java. The form definition syntax is as follows:

<fd:on-xxxx> <javascript> ... some inline javascript code ... </javascript> <java class="..."/> </fd:on-xxxx>

You can specify as many <javascript> and/or <java> event listeners as you want.

4.1. Javascript event listeners

Objects available in the Javascript snippet:

- event: a subclass of WidgetEvent. The reference documentation of the individual widgets mentions which WidgetEvent subclass they provide in their events. You can then check the javadoc for those classes to see what they provide.
- viewData: any data that is normally passed from the flowlayer to the view (pipeline). Exact contents depends on which flowscript API version you use.
- if the form processing was started from a flowscript, then everything available from the scope of that flowscript, such as global variables, functions and the cocoon object (see also Flow Object Model).

It does not make sense to create continuations from the Javascript event handler. In other words, do not call cocoon.sendPageAndWait or form.showForm from there.

4.2. Java event listeners

The Java class specified in the class attribute on the java element should implement a certain event listener interface. Which interface depends on the type of widget. See the documentation of the individual widgets for more information.

5. Adding event listeners on widget instances

Adding event listeners on widgets instances allows to dynamically add event listeners at runtime. This is often convenient: as you control the creation of the event listeners yourself, you can pass them any information you need.

To add an event listener on a widget instance, simply call the appropriate method on the widget (e.g. addValueChangedListener) with an appropriate listener object as argument. You can of course also remove the event listener afterwards (e.g. removeValueChangedListener).

When using flowscript, it is possible to simply assign Javascript functions as event listeners. This is a very easy and powerful way to create event listeners. See the flowscript API section for more information.

6. Handling events using the FormHandler

To handle events using a FormHandler, write a class implementing the following interface:

org.apache.cocoon.woody.event.FormHandler

Alternatively you can extend from the following abstract class:

org.apache.cocoon.woody.event.AbstractFormHandler

which will split ActionEvents and ValueChangedEvents to two different methods. See the javadocs of these interfaces and classes for more details.

Once you created the FormHandler, register it on a form instance by calling the method setFormHandler(FormHandler formHandler) on it.

7. Overview of supported events

The figure below shows the 3 types of events we currently support, each extending from the common WidgetEvent class.

Overview of event types

The full types of the event listeners and event objects are:

org.apache.cocoon.forms.event.ValueChangedListener
org.apache.cocoon.forms.event.ValueChangedEvent
org.apache.cocoon.forms.event.ActionListener org.apache.cocoon.forms.event.ActionEvent
org.apache.cocoon.forms.event.ProcessingPhaseListener
org.apache.cocoon.forms.event.ProcessingPhaseEvent

The table below gives an overview of what events are supported on what widgets.

Widget	Supports ValueChangedEvents	Supports ActionEvents
field	yes	
multivaluefield	TODO	
booleanfield	yes	
repeater		
output		
submit		yes
action		yes
repeater-action		yes
row-action		yes
aggregatefield	TODO	
upload		
messages		

1. Comments

add your comments