# Cocoon Forms: field widget (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. Concept

The field widget is the most common widget. It is used both for text boxes or selection lists. It can be associated with different datatypes such as string, long or date to ask for different types of data.

Diagram showing the associations between field, datatype, convertor, selection list and validators.

## 1.1. Datatypes

A datatype represents a certain type of data, such as a string, integer, decimal or date. Each datatype matches to a certain Java class. If you associate a field widget with a datatype, its setValue(Object) and getValue() methods will take, respectively return objects that are instances of that Java class (or subclasses thereof).

Each datatype is associated with a **convertor**. The task of the convertor is to convert from string representation to object representation, and vice versa.

The string to object conversion usually happens when converting the value entered by the user to an object. This process can fail if the user entered an incorrect string, for example abc when a number is required. In this case an appropriate validation error will be set on the widget. String to object conversion also happens when parsing data in selection lists (if the selection list is retrieved as XML) and can also be used as part of the binding.

The object to string conversion happens when the state of the widget is spit out as XML, this is mostly when injecting the widget XML in the publishing pipeline.

By having a field widget associated with a datatype, you can be sure that, after successful validation of the widget, retrieving the value of the widget will give you an object of the correct type.

The available datatypes and their respective convertors are documented in a separate document.

## 1.2. Selection lists

A field widget can furthermore be associated with a selection list. This makes that the field widget could be rendered either as a textbox or a list, depending on whether its datatype has a selection list. The selection list is related with the datatype: the values in the selection list should be of the same type as the datatype.

Selection list data can be specified directly in the form definition (for short, unchanging lists), retrieved from external sources (i.e. a Cocoon pipeline), or pulled from an oject structure. Full details on selection lists are also in a separate document.

## 1.3. Conclusion

If we wouldn't make these datatype and selection list associations, we would need to create specific widgets for each possible combination: StringField, LongField, DateField, StringSelectionList, LongSelectionList, ...

# 2. Configuration

Configuration example:

```
<fd:field id="..." required="true|false"> <fd:label>...</fd:label> <fd:hint>...</fd:hint>
<fd:help>...</fd:help> <fd:datatype base="..."> [...] </fd:datatype> <fd:selection-list .../>
<fd:validation> [...] </fd:validation> <fd:on-value-changed> [...] </fd:on-value-changed>
</fd:field>
```

The field element takes a required **id** attribute. This id should be unique among all widgets in the same container (i.e. inside the same fd:widgets element).

The **required** attribute is optional, by default it is false. It indicates whether this field is required. This is a static property of the widget. If you want the field to be "conditionally required", then set this to false and use custom validation logic to check the requiredness of the field.

The **fd:label** element contains the label for this widget. This element is optional. It can contain mixed content. For internationalised labels, use i18n-tags in combination with Cocoon's I18nTransformer.

The **fd:hint** element contains a hint for the form control of this widget. This element is optional. It can contain a hint about the input control. For internationalised labels, use i18n-tags in combination with Cocoon's I18nTransformer.

The **fd:help** element contains more help for the form control of this widget. This element is optional. It can contain text help about the input control. For internationalised labels, use i18n-tags in combination with Cocoon's I18nTransformer.

The **fd:datatype** element indicates the datatype for this field. This element is required. The base attribute specifies on which built-in type this datatype should be based. The contents of the fd:datatype element can contain further configuration information for the datatype. The possible datatypes and their configuration options are described over here.

The **fd:selection-list** element is used to associate a selection list with this field. See Datatypes for more details.

The **fd:validation** element specifies widget validators. See Validation for more details.

The **fd:on-value-changed** element specifies event handlers to be executed in case the value of this field changes. See also Event Handling. The interface to be implemented for Java event listeners is org.apache.cocoon.forms.event.ValueChangedListener. The WidgetEvent subclass is org.apache.cocoon.forms.event.ValueChangedEvent.

**Note:** Events used in <fd:on-value-changed> require that the form instance is stored serverside (because otherwise CForms doesn't know what the previous values of the fields were). This is automatically the case when you use flowscript. If you don't use flowscript you could store the form instance in e.g. the session.

## 1. Comments

add your comments