

# Calyxo Struts

## Table of contents

1 Introduction.....	2
2 Calyxo Struts Plugins.....	2
2.1 Base plugin for Struts.....	2
2.1.1 Struts Configuration.....	2
2.1.2 Struts modules.....	3
2.1.3 Struts i18n.....	3
2.2 Panels plugin for Struts.....	4
2.2.1 Struts Configuration.....	4
2.2.2 Customizing the Request Processor.....	5
2.3 Forms plugin for Struts.....	6
2.3.1 Struts Configuration.....	6
2.3.2 Calyxo Action Forms.....	7
2.3.3 Struts "Legacy" Forms.....	8
2.3.4 Actions.....	9
2.3.5 Error Messages.....	10
3 Project.....	11
3.1 History of Changes.....	11
3.2 Todo List.....	12

## 1. Introduction

The *Calyxo Struts* component focusses on integrating *Calyxo* components into the very popular Struts framework by the Apache Group.

*Calyxo* is for you, the Struts developer, if

- you like the core controller part of Struts, but feel strange about Struts Tiles, the Struts Validator and Struts' JSP taglibs; you're definitely right.
- you like all of Struts, including Tiles and the Validator; you'll love *Calyxo* anyway!
- your boss likes Struts; *Calyxo* will try to prevent you from suffering...

*Calyxo Struts* provides plugins to fully integrate the *Calyxo Base*, *Calyxo Panels* and *Calyxo Forms* components. *Calyxo Struts* requires Struts version 1.2.x or later. The current release has been tested with Struts versions 1.2.9 and 1.3.5.

Go ahead, Struts developer, have fun and take profit!

## 2. Calyxo Struts Plugins

Currently, the following plugins are available:

- the [Calyxo Base Struts plugin](#) to provide basic services like accessors,
- the [Calyxo Panels Struts plugin](#) as an alternative to Struts Tiles,
- the [Calyxo Forms Struts plugin](#) as an alternative to the Struts Validator.

### 2.1. Base plugin for Struts

The base plugin makes the *Calyxo Base* component available to Struts applications.

The plugin is a prerequisite to the other plugins provided by the *Calyxo Struts* component. It integrates *Calyxo* modules with Struts modules, *Calyxo*'s i18n support with Struts message resources and enables usage of *Calyxo* accessors.

Though, however, this plugin is mainly used in conjunction with other *Calyxo Struts* plugins, it may be used alone. This way, an application may load *Calyxo Base* configuration files and use all the accessors and JSP tags provided by *Calyxo Base*.

#### 2.1.1. Struts Configuration

The plugin is loaded in the Struts configuration file:

```
<!DOCTYPE struts-config PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
  "http://struts.apache.org/dtds/struts-config_1_2.dtd">

<struts-config>
```

```

...

<!-- load base plugin -->
<plug-in className="de.odysseus.calyxo.struts.base.BasePlugIn">
  <!-- optional configuration file -->
  <set-property property="config" value="/WEB-INF/calyxo-base-config.xml"/>
  <!-- optional module name -->
  <set-property
    property="module"
    value="main"/>
</plugin>

...

</struts-config>

```

The base plugin takes an optional config parameter to specify a *Calyxo Base* configuration file.

The optional module parameter can be used to specify the *Calyxo* module name. If omitted, the Struts module prefix is taken as module name.

The *Calyxo Base* plugin is a prerequisite to all the other plugins provided by the *Calyxo Struts* component. In the configuration, it must be located *before* all other *Calyxo Struts* plugins.

### 2.1.2. Struts modules

Struts modules are fully supported. The *Calyxo Struts* component provides the `StrutsModuleContext` module context implementation. This class supports URL prefix- as well as extension-mapped modules.

Here are the relevant details of the Struts module context implementation:

- `getName()` answers the module prefix, which is the empty string for the default module and a string starting with a slash ("/") for other modules.
- `getInitParameter(String name)` simply answers the servlet configuration's init parameter for the given name.
- `getPath(String action)` answers a context-relative path for the specified action mapping path. The action may have a query and/or anchor appended.
- Module scope is simulated by mapping key "<k>" to attribute "<k><p>" in application scope, where <p> is the module prefix. Thus, `module.getAttribute("foo")` is equivalent to `module.getServletContext().getAttribute("foo" + module.getName())`.

The module scope doesn't prevent conflicts to application scope. In particular, module scope and application scope are the same in the default module. The reason we implemented it this way is that Struts uses the same mapping strategy to save module-related stuff, e.g. message resources and configuration data.

### 2.1.3. Struts i18n

The `StrutsI18nSupport` class extends the *Calyxo Base* i18n support base class and will be used by *Calyxo* to resolve localized resources and to format messages.

The implementation simply forwards to the Struts message resources mechanism, so there's not much to say, here.

For example, an accessor expression like `${calyxo.base.i18n.bundle['foo'].resource['bar']}` will lookup key "bar" in the Struts message resource, that you registered in your Struts configuration with key="foo".

You may refer to the default message resource (which has no key attribute set in its configuration) by specifying bundle null.

## 2.2. Panels plugin for Struts

The panels plugin makes the *Calyxo Panels* component available to Struts applications. Please, consult the *Calyxo Panels* documentation to learn what you can do with panels.

### 2.2.1. Struts Configuration

The panels plugin requires the [base plugin](#) to be installed. In the Struts configuration, the base plugin must be declared *before* the forms plugin:

```
<!DOCTYPE struts-config PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
  "http://struts.apache.org/dtds/struts-config_1_2.dtd">

<struts-config>

  ...

  <!-- load base plugin -->
  <plug-in className="de.odysseus.calyxo.struts.base.BasePlugIn"/>

  <!-- load panels plugin -->
  <plug-in className="de.odysseus.calyxo.struts.panels.PanelsPlugIn">
    <set-property property="config" value="/WEB-INF/calyxo-panels-config.xml"/>
  </plug-in>

</struts-config>
```

The panels plugin takes a mandatory config parameter to specify the *Calyxo Panels* configuration file.

### Struts 1.3

Struts 1.3 requires to configure the request processing chain via XML in order to use *Calyxo Panels*. The *Calyxo Struts* jar contains a chain definition which emulates Struts 1.2 and adds

the Calyxo Panels commands. The chainConfig module parameter must be set to use this configuration instead of the default:

```
<init-param>
  <param-name>chainConfig</param-name>
  <param-value>
    de/odysseus/calyxo/struts/panels/calyxo-panels-chain-config.xml
  </param-value>
</init-param>
```

#### Note

The above example configures the default module's chain. If you're using Struts modules, module foo will take parameter foo/chainConfig, module bar will take parameter bar/chainConfig and so on.

## 2.2.2. Customizing the Request Processor

When the Struts controller dispatches a request to a specified path (through a forward or include), the plugin checks, if that path is the name of a toplevel panel definition. If so, it causes the controller to dispatch to the panel's associated template, instead. Otherwise, the request is dispatched to the specified path, as usual.

### Struts 1.3

Struts 1.3 provides `org.apache.struts.chain.ComposableRequestProcessor` which allows to configure the request processing chain via XML. To use *Calyxo Panels* with this processor, the `de.odysseus.calyxo.struts.panels.PanelsCommand` must be added to the chain.

The *Calyxo Struts* jar contains a chain definition which emulates Struts 1.2 and adds the Calyxo Panels commands. Refer to the [configuration](#) section on how to use it. A copy of the modified chain config file is located in `calyxo-struts/conf/share/calyxo-panels-chain-config.xml`.

If you need to specify your own customized chain configuration, you will need to include the *Calyxo Panels* command yourself.

### Struts 1.2

Under Struts 1.2.x, intercepting the process of request dispatching requires subclassing the module's request processor class. Therefore, the panels plugin comes with its own processor class. The plugin will automatically cause Struts to use this processor class as default.

However, if an application wants to use a custom processor class, it must extend the plugin's processor class!

In the Struts configuration file, the request processor class may be specified using the `processorClass` attribute of the `<controller>` element. For the panels plugin to work, the

request processor class used must be the same or a subclass of `de.odysseus.calyxo.struts.panels.PanelsRequestProcessor`.

## 2.3. Forms plugin for Struts

The forms plugin makes the *Calyxo Forms* component available to Struts applications. Please, consult the *Calyxo Forms* documentation to learn what forms can do for you.

Though the *Calyxo Forms* component comes with its own form/input JSP tag library, the plugin supports use of the tags provided by Struts.

- The *Calyxo Forms* tags are to be used in conjunction with the generic [Calyxo Action Forms](#), which free the application developer from implementing an action form class per HTML form.
- The Struts tags are to be used in conjunction with [Struts Legacy Forms](#). Plain action forms as well as DynaForms are supported.

### 2.3.1. Struts Configuration

#### Plugin

The forms plugin requires the [base plugin](#) to be installed. In the Struts configuration, the base plugin must be declared *before* the forms plugin:

```
<!DOCTYPE struts-config PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
  "http://struts.apache.org/dtds/struts-config_1_2.dtd">

<struts-config>

  ...

  <!-- load base plugin -->
  <plug-in className="de.odysseus.calyxo.struts.base.BasePlugIn"/>

  <!-- load forms plugin -->
  <plug-in className="de.odysseus.calyxo.struts.forms.FormsPlugIn">
    <set-property property="config" value="/WEB-INF/calyxo-forms-config.xml"/>
  </plug-in>

</struts-config>
```

The forms plugin takes a mandatory `config` parameter to specify the *Calyxo Forms* configuration file.

#### Form Beans

Generally, a `<form-bean>`'s `name` attribute references a form in the *Calyxo Forms* configuration

file.

Depending on if you want to use Struts' html tag library or the *Calyxo Forms* form/input tags, you will need to use different action form classes. Please refer to the sections on [Struts Legacy Forms](#) and [Calyxo Action Forms](#), respectively.

## Actions

Struts `<action>` elements are configured for validation as usual.

### 2.3.2. Calyxo Action Forms

The *Calyxo Forms* component comes with its own tag library. This library contains custom tags related to the HTML form- and input tags. These tags directly access your form data, so form bean properties are ignored. To use these tags with Struts, you have to use the form bean class

```
de.odysseus.calyxo.struts.forms.CalyxoActionForm
```

or a subclass of it. The input values to be validated are directly taken from the request. Since `CalyxoActionForm` is a generic form class, Struts users should not derive from this class to add form bean properties as they are used to with ordinary action forms: the populated bean properties will neither be used as validation inputs, nor will the *Calyxo Forms* input tags pull their display values out of these properties.

*Calyxo* forms are declared in the Struts configuration file as usual:

```
<!DOCTYPE struts-config PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
  "http://struts.apache.org/dtds/struts-config_1_2.dtd">

<struts-config>
  <form-beans>
    <form-bean
      name="loginForm"
      type="de.odysseus.calyxo.struts.forms.CalyxoActionForm"/>
    ...
  </form-beans>
  ...
</struts-config>
```

This also means, that you no longer need to implement dozens of form bean classes! The `CalyxoActionForm` class will keep the validated data in a map.

#### Warning

Again: this action form implementation does not use bean properties or mapped properties and does not work with the form input tags shipped with Struts! If you want to use action forms with the Struts tags, consult the [Struts "Legacy" Forms](#) section.

### 2.3.3. Struts "Legacy" Forms

Though [Calyxo Action Forms](#) are easier to use, there may be reasons to keep using the Struts HTML tags and form beans.

This might be the case if you're migrating from the Struts Validator to *Calyxo Forms*. It would be an extra effort to migrate your JSP pages to use *Calyxo Forms*' form/input tags. Or, you may depend on a special feature offered by some Struts tag, or...

Thus, we provide "legacy" action forms to be used with the Struts tag library, located in package `de.odysseus.calyxo.struts.forms.legacy`. There are two flavours of action form base classes:

- *Simple forms* – the classes `SimpleActionForm` and `SimpleDynaActionForm` are drop-in replacements for action form classes in existing applications.
- *Flushable forms* – the classes `FlushableActionForm` and `FlushableDynaActionForm` are intended for use in new projects, that want to stay with the Struts form/input tag library.

#### Note

Currently, there's one notable restriction: all form properties of legacy action forms that are to be validated by *Calyxo Forms* must be of type `java.lang.String`.

### Simple Action Forms

Let's continue with the migration scenario. Consider you have an application which validates inputs by hand or uses the Struts Validator. Now, you would like to use *Calyxo Forms* to validate your forms. The migration path is short and simple 1-2-3 procedure:

1. Create your forms configuration file(s). Please, read the *Calyxo Forms* documentation to learn about that. The form names in the forms configuration correspond to form names in your Struts configuration.
2. Make your form beans subclasses of `SimpleActionForm`. Or, if you're using `DynaForms`, derive from `SimpleDynaActionForm`. You may also want to remove hand-coded validation code, that will now be covered by validation rules.
3. Load the *Calyxo Forms* for Struts Plugin as described in the [configuration](#) section.

You're done. Note, that you don't have to touch any action classes and JSP files!

The simple action forms have been designed for the "Enhance an existing application with minimal change effort" scenario. Thus, they do not need to offer things, that aren't available when using plain action forms or the Struts validator. In particular, committing form properties in a simple action form has no effect.

The major drawback of Struts action forms in general is, that they don't maintain a valid state. Thus, after an action form has once been populated with invalid data, this data will be shown again if the user navigates back to a page containing the form. Ugly!

## Flushable Action Forms

The flushable action forms fix this. They are to be used with the Struts tags and keep validated form inputs for you.

Why are flushable forms called flushable? Flushable forms store committed form properties in a map. The *Calyxo Forms* component is able to format these values back into strings, that are to be displayed in the HTML form elements. However, the Struts tags pull their values out of the form bean properties, which still contain the unformatted input strings. So, when we *flush* our form, we format back our form data and re-populate the form bean properties. That way, we synchronize our view with the valid values. When redisplaying the form, these values will appear in a standard, localized format.

Flushing a form is triggered by the `<flush>` tag, provided by the *Calyxo Struts* component. The tag has to be placed inside a Struts form tag, *before* any input elements:

```
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.0"
  xmlns:calyxo="http://calyxo.odysseus.de/jsp/struts"
  xmlns:html="http://struts.apache.org/tags-html">
  <html:form method="POST" action="/foo">
    <calyxo:flush/>
    Enter a number
    <html:text property="number"/>
    <html:submit/>
  </html:form>
</jsp:root>
```

### 2.3.4. Actions

Using *Calyxo Forms* with Struts doesn't require special action classes. However, you may want to interfere with *Calyxo Forms* when accessing validated form properties or committed form data. Everything starts by getting a reference to the module's `FormsSupport` instance:

```
FormsSupport support = FormsSupport.getInstance(request);
```

### Form Properties

The most essential thing is to access the validated form properties:

```
FormProperties properties = support.getFormProperties(request, mapping.getPath());
Date arrival = (Date)properties.getProperty("arrival");
```

To commit the form properties, use

```
properties.commit();
```

Please refer to the *Calyxo Forms Concepts* for further details.

## Form Data

The FormsSupport instance may also be used to access, remove or re-create saved form data for some action. This is useful if you want to initialize form data before (re)displaying the form:

```
FormData data = support.getFormData(request, "/checkin", true);
data._setProperty("arrival", new Date());
```

or

```
support.removeFormData(request, "/checkin");
```

Please refer to the *Calyxo Forms* Concepts for further details.

### 2.3.5. Error Messages

Struts messages and *Calyxo* messages differ in that *Calyxo* messages may have a resource key *and* bundle name whereas Struts messages only have a key. Thus, when creating Struts action errors, the bundle "gets lost". To prevent from this, the validator plugin does the following:

If a Struts message has to be created from a *Calyxo* message with a bundle set, the plugin formats the message during validation and creates a Struts action message with key "", which takes the preformatted message string as an argument.

Therefore, to display validation error messages, you should either add the line

```
={0}
```

to your messages properties file, or check for messages with empty keys in your JSP and simply output their argument.

If you want to use Struts' `<html:messages>` tag to display your messages, you have to take the first option. Below is a JSP fragment which displays all messages using the `struts.messages` accessor provided by *Calyxo Struts*:

```
<c:set var="errors" scope="page"
  value="${calyxo.struts.messages['org.apache.struts.action.ERROR']}" />
<c:if test="${!empty pageScope['errors']}">
  <c:set var="bundle" value="${calyxo.base.i18n.bundle['application']}" />
  <ul>
    <c:forEach var="error" items="${errors}">
      <c:choose>
        <!-- formatted message (if you don't have line "{0}") -->
        <c:when test="${empty error.key}">
          <li>${error.values[0]}</li>
        </c:when>
        <!-- message without arguments -->
```

```

    <c:when test="{empty error.values}">
      <li>${bundle.message[error.key]}</li>
    </c:when>
    <!-- message with one or more arguments -->
    <c:otherwise>
      <li>${bundle.message[error.key][error.values]}</li>
    </c:otherwise>
  </c:choose>
</c:forEach>
</ul>
</c:if>

```

## 3. Project

### 3.1. History of Changes

#### **Version 0.9.0 (2006/10/28)**

**developer: cbe context: code type: update**

Upgrade to Struts 1.3.5.

**developer: cbe context: code type: update**

Minor refactoring to eliminate package cycle between control and control.misc.

**developer: cbe context: code type: fix**

Fixed NPE in StrutModuleContext if the action servlet has no servlet mapping at all.

**developer: cbe context: code type: update**

Make sure to pop() panels context when an exception is thrown when forwarding to or including templates.

**developer: cbe context: code type: update**

Made PanelsPlugIn accept ComposableRequestProcessor as request processor class to open up for struts 1.3.

**developer: cbe context: code type: update**

Do not create validator message resources if this has been done in the default module.

#### **Version 0.9.0-rc3 (2006/02/12)**

**developer: cbe context: code type: update**

Renamed FormsSupport by StrutsFormsSupport.

**developer: cbe context: code type: update**

Define message resources for key calyxo-forms-validators in FormsPlugin if this has not been done in the struts configuration.

**developer: cbe context: code type: add**

Added optional plugin property name to specify a module name different from the module's prefix.

**developer: cbe context: code type: update**

Replaced StrutsModuleSupport by StrutsModuleGroup.

**developer: cbe context: code type: update**

BasePlugin now installs access support *before* parsing the configuration to allow for adding accessors from xml.

**developer: cbe context: code type: add**

Added StrutsModuleContext.setClassLoader().

**Version 0.9.0-rc1 (2005/07/03)**

**developer: cbe context: code type: add**

Added optional base plugin config parameter to load a *Calyxo Base* configuration file.

**developer: ost context: code type: add**

Introduced interface FormProperties.

**developer: cbe context: code type: update**

Renamed component validation to forms.

**developer: cbe context: code type: update**

Panels: renamed panel's dispatch attribute to template.

**Version 0.9.0-b5 (2005/01/04)**

**developer: cbe context: code type: update**

Updated webapps code to calyxo-validation-0.9.0-b5.

**Version 0.9.0-b4 (2004/11/21)**

**developer: cbe context: code type: update**

Updated code to calyxo-validation-0.9.0-b4.

**Version 0.9.0-b3 (2004/10/20)**

**developer: cbe context: docs type: add**

Added documentation.

**developer: cbe context: design type: update**

Refactored classes in legacy package.

**developer: cbe context: code type: update**

Upgraded to Struts 1.2.4

**Version 0.9.0-b2 (2004/07/04)**

**developer: cbe context: admin type: add**

Initial Import

## 3.2. Todo List

### *medium priority*

- **[code]** Write more unit tests. >> Christoph
- **[docs]** Improve documentation. >> Christoph