Calyxo Panels

Table of contents

1. Introduction

Developing and maintaining dynamic views is challenging, since there are often aspects of both the content and the layout that are common across multiple views. When content and layout are interwined, it is harder to maintain and extend the views. Reuse and modularity also suffer when common code is duplicated across views.

Calyxo Panels realize the *Composite View* design pattern to enable management of views in a flexible and maintainable way.

The *Calyxo Panels* component enables the definition of views in a container/component manner. Pages are dynamically composed from a tree of page fragments. Panel definitions are centralized in an XML configuration file.

Reusability

As a major plus, *Calyxo Panels* bring the benefits of object oriented programming into the view layer. Since your page fragments are reusable units, there's no need for code duplication.

We'll refer to page fragments as *templates*.

A *panel* defines how to bundle templates into a concrete view. Panel definitions themselves may use inheritance by extending other panel definitions, which is also of great use.

Flexibility

Beyond linking to templates, panel definitions may specify parameters and lists, which are made available to templates through custom tags (and *Calyxo* accessors).

Yet flexible, the *Calyxo Panels* component is impressively small. The configuration introduces just four elements (including the root element). The tag library contains three tags and there's one accessor. This condensed design helps to make the component very easy to understand and use.

Maintainability and Consistency

With *Calyxo Panels*, view maintenance is greatly simplified. Adding a new view requires just adding a panel definition and providing the template(s) containing content unique to that view.

Avoiding duplication of content by reuse leads to a high level of consistency. Keeping a consistent layout is made much easier.

Portability

You may use *Calyxo Panels* with *Calyxo Control* or with Struts. The information provided on the following pages apply to either environment.

2. Panels Concepts

Many aspects of a web application's presentation layer are common to all views. For example, pages may be composed of header, footer, menu and content areas. To keep maintainability and consistency, reuse of view components is an important issue. Thus, we need a mechanism to define our views as compositions of reusable page fragments.

Templates

To achieve this, we split our views into *templates*, which are combined into pages at runtime. A template may include other templates and may be included by other templates. In this model, an actual view may be seen as a template tree.

To be reusable, a template uses symbolic names to specify templates to be included. The actual template path has to be resolved according to some definition. Such a definition defines a template tree mentioned before.

Panels

In Calyxo terminology, these definitions are called *panels*.

- Panels may be nested to arbitrary depth. We'll call a panels's children *subpanels*. We distinguish between *toplevel* panels and *nested* panels.
- A panel has a name. Since this is the symbolic name used by a template to include a child template, it must be unique under its panel siblings.
- A panel definition may extend a toplevel panel definition. That is, panel definitions are reusable by themselves through inheritance.
- A panel may pass parameters to its associated template. These parameters are made available to the template through a Calyxo accessor.
- Toplevel panel definitions may be localized. That is, a panel definition can be associated with a locale. During panel lookup this is taken into account using *generalization*.
- A panel is dynamically associated with a template path. The associated template is resolved at instantiation time using inheritance and generalization. This selection process assigns concrete paths to the subpanel names used by templates to include their child templates.

Instantiation

Toplevel panels are visible to the controller. In a *Calyxo Panels* environment, you can use their names as resource paths, just as you do with static resources. Calyxo will dynamically lookup subpanels and compose and render the corresponding templates. Thus, you may see toplevel panels as *virtual views*.

We call the process of selecting a panel and rendering its associated template *instantiation*. When a toplevel panel is requested for instantiation,

1. a stack is created with the panel pushed onto it
2. the template associated with the panel is resolved
3. the request is dispatched to the associated template

During instantiation, the panel element on top of the maintained stack is called the *current* panel. The stack itself is kept in request scope.

When a template is executed, it includes templates associated with subpanels of the current panel. When a nested panel is requested for instantiation,

1. the requested subpanel is looked up in the current panel
2. the subpanel is pushed onto the stack (making it the current panel)
3. the template associated with the subpanel is resolved
4. the request is dispatched to the associated template
5. the subpanel is removed from the stack by performing a pop

This way, we traverse the tree of associated templates and render the results into the current response.

2.1. Configuration

As usual, the *Calyxo Panels* component is configured per module in one or more XML configuration files.

Namespace

Each *Calyxo* component uses its own separate namespace for configuration elements. The elements described in the following sections are defined within namespace `http://calyxo.odysseus.de/xml/ns/panels`.

Schema/DTD Validation

When loading configuration files, *Calyxo* forces the XML parser to validate documents against an XML Schema definition (XSD) or a Document Type Declaration (DTD). *Calyxo* prefers XML Schema validation, so, if your parser supports it, it is used. Otherwise, DTD validation is used. In this case, configuration files must declare the *Calyxo Panels* document type:

```
<!DOCTYPE calyxo-panels-config
  PUBLIC "-//Odysseus Software GmbH//DTD Calyxo Panels 0.9//EN"
  "calyxo-panels-config.dtd">
```

Copies of the DTD and XSD are located at `CALYXO_HOME/calyxo-panels/conf/share/calyxo-panels-config.*`.

Note

Even if your parser supports XSD, you may want to include the DTD declaration in your documents. A reason can be the use of an XML editor, which supports code assistance only for DTDs. Regardless of that, Calyxo will try to use XML Schema validation during parse.

Document structure

The root element is `<calyxo-panels-config>`. As common to all of Calyxo's configuration files, the root element specifies the `xmlns` and `version` attributes and may optionally contain `<base:import>` elements, followed by `<base:functions>`, `<base:set>` and `<base:use>` elements.

Following these common elements, the root contains one ore more `<panels>` elements. A `<panels>` element can be associated with a locale by specifying the `language`, `country` and `variant` attributes. This advanced feature is covered in the i18n section.

For now, let's consider the usual case, where we have exactly one `<panels>` section, without specifying a locale. Thus, our configuration document looks like this:

```
<calyxo-panels-config version="0.9"
  xmlns="http://calyxo.odysseus.de/xml/ns/panels"
  xmlns:base="http://calyxo.odysseus.de/xml/ns/base">

  <!-- base:import elements can go here -->
  <!-- base:functions, base:set and base:use
       elements can go here -->

  <panels>

    <!-- panel definitions go here -->

  </panels>

</calyxo-panels-config>
```

A `<panels>` element contains one or more panel definitions.

2.2. Panels

Panels may be declared as children of a `<panels>` element using `<panel>` elements. Since panels may be nested to arbitrary depth, `<panel>` elements can contain `<panel>` child elements defining their subpanels.

A `<panel>` element requires the `name` attribute. The name must be unique under the `<panel>`'s siblings. For toplevel panels, the application uses this name like a resource path to instantiate it; for nested panels, the name is used by templates to include a template associated with a subpanel of the current panel. To reflect this, we use the convention to assign path-like names to toplevel panels and identifier-like names to subpanels.

The optional `template` attribute may be used to specify the associated template.

The optional `super` attribute can be used to specify a toplevel panel to extend. This is where inheritance comes into play. See the inheritance section for more on this.

Let's illustrate this by giving an example. Consider the following panel definitions:

```
<panel name="/base.page" template="/WEB-INF/jsp/layout/page.jsp">
  <panel name="header" template="/WEB-INF/jsp/layout/header.jsp"/>
  <panel name="menu" template="/WEB-INF/jsp/layout/menu.jsp"/>
  <panel name="content"/>
  <panel name="footer" template="/WEB-INF/jsp/layout/footer.jsp"/>
</panel>

<panel name="/foo.page" super="/base.page">
  <panel name="content" template="/WEB-INF/jsp/content/foo.jsp"/>
</panel>
```

- Following our convention, we assigned path-like names to toplevel panels and identifier-like names to nested panels.
- The `/base.page` panel defines the subpanels `header`, `menu`, `content` and `footer`. The `template` attributes denote the associated templates.
- The `/base.page`'s `content` panel does not specify a template, making `/base.page` *abstract*. That is, `/base.page` cannot be used as a dispatch target. Rather, it acts as a base definition, which may be extended by other panels.
- The `/foo.page` panel extends `/base.page` by specifying `super="/base.page"` and associates a template with its `content` subpanel, making it *concrete*. Thus, `/foo.page` can be used as a dispatch path by the controller.
- The `/WEB-INF/jsp/layout/page.jsp` template instantiates subpanels using the `<panel>` tag from the custom tag library provided by the *Calyxo Panels* component like this:

  ```
  <jsp:root version="2.0"
    xmlns:panels="http://calyxo.odysseus.de/jsp/panels"
    xmlns:jsp="http://java.sun.com/JSP/Page">
    ...
    <panels:panel name="header"/>
    ...
    <panels:panel name="menu"/>
    ...
    <panels:panel name="content"/>
    ...
    <panels:panel name="footer"/>
    ...
  </jsp:root>
  ```

To your application, concrete toplevel panels are of particular interest, because they can be instantiated by the controller. Technically spoken, you may dispatch to toplevel panels using their (path-like) names, just as you do with any resources.

Beside subpanels, a `<panel>` element may contain elements defining parameters and lists.

2.3. Parameters

You may define parameters within your panel definitions using the `<param>` element. The `<param>` element takes a mandatory `name` attribute and an optional `value` attribute. The `name` attribute values must be unique under the `<panel>`'s `<param>` children. The optional `value` attribute defines the parameter value.

Omitting the `value` attribute leaves the parameter value undefined. Accessing an undefined parameter value will cause an exception to be thrown. Undefined parameters should be overridden by derived panels or specified by the template during inclusion using a dynamic parameter. Use `value="${null}"` to specify value `null`.

The parameters of the current panel are accessible by templates using a *Calyxo* accessor.

Consider the following example:

```
<panel name="..." template="/foobar.jsp">
  ...
  <param name="foo" value="hi!"/>
  <param name="bar"/>
  ...
</panel>
```

- The first `<param>` element defines the parameter `foo` with value `"hi!"`
- The second `<param>` element defines the parameter `bar` without specifying a value.
- Now, the associated template `/foobar.jsp`, may access the current panel's parameters from a JSP EL expression using the `${calyxo.panels.param[...]}` accessor:

```
<jsp:root version="2.0" xmlns:jsp="http://java.sun.com/JSP/Page">
  ...
  <!-- access parameter via EL -->
  <sometag someattr="${calyxo.panels.param['foo']}">...</sometag>
  ...
</jsp:root>
```

Dynamic parameters

It is also possible to specify *dynamic* parameters during panel inclusion. A template uses the `<param>` tag inside a `<panel>` tag like this:

```
<jsp:root version="2.0"
  xmlns:panels="http://calyxo.odysseus.de/jsp/panels"
  xmlns:jsp="http://java.sun.com/JSP/Page">
  ...
  <panels:panel name="/button">
    <panels:param name="label" value="Save"/>
    <panels:param name="name" value="save"/>
  </panels:panel>
  ...
```

```
</jsp:root>
```

During instantiation of an included panel, dynamic parameters are added to its namespace, overriding static parameter definitions from the panel's configuration.

2.4. Lists

Up to now, templates must "know" subpanels and parameters by name to use them. This isn't a problem in most circumstances. However, a template might want to render a menu tree, where items are passed through by the panel definition; or, it might want to layout news items in a column, where the blocks of content are defined by a list of subpanels.

To satisfy these needs, `<panel>` elements may contain `<list>` elements as children. A `<list>` element requires a `name` attribute. As with subpanels and parameters, the `name` attribute values must be unique under the `<panel>`'s `<list>` children.

A list contains a sequence of `<item>` elements. An `<item>` element contains a mixed sequence of `<panel>`, `<param>` and - you guessed it - `<list>` elements.

A template uses the `<list>` tag to *iterate* over a list contained in the current panel. During iteration, the current item behaves as if its children elements were moved up to the current panel. That is - semantically - the current item is merged with the current panel.

As an example, consider the following panel definition:

```
<panel name="/blurb" template="/jsp/column.jsp">
  <list name="blurbs">
    <item>
      <param name="head" value="Item 1"/>
      <panel name="body" template="/jsp/blurb1.jsp"/>
    </item>
    <item>
      <param name="head" value="Item 2"/>
      <panel name="body" template="/jsp/blurb2.jsp"/>
    </item>
    <item>
      <param name="head" value="Item 3"/>
      <panel name="body" template="/jsp/blurb3.jsp"/>
    </item>
  </list>
</panel>
```

- The `/blurb` panel defines the `blurbs` list, containing three items, each containing a parameter named `head` and a panel named `body`.
- The layout template `/jsp/column.jsp` iterates over the list:

```
<jsp:root version="2.0"
  xmlns:panels="http://calyxo.odysseus.de/jsp/panels"
  xmlns:jsp="http://java.sun.com/JSP/Page">
  <panels:list name="blurbs">
```

```
      <h4><panels:param name="head"/></h4>
      <p>
        <panels:panel name="body"/>
      </p>
    </panels:list>
  </jsp:root>
```

Thus, when rendering `/blurb`, the above template creates paragraphs containing contents of `/jsp/blurb1.jsp` with heading `"Item 1"`, `/jsp/blurb2.jsp` with heading `"Item 2"` and `/jsp/blurb3.jsp` with heading `"Item 3"`.

The `<panels:list>` element also accepts an optional `varStatus` attribute. If set, the tag will export a *list status* object to that attribute in page scope. The list status object provides methods `boolean isFirst()`, `boolean isLast()` and `int getIndex()`, which may be used to retrieve information about the position of the current item within the list.

2.5. Inheritance

As mentioned before, a panel may explicitly *extend* a toplevel panel by specifying the `super` attribute. A *derived* panel extends its *base* panel in the following ways:

- An attribute that appears in the derived panel overrides an attribute in the base panel with the same name.
- A parameter that appears in the derived panel overrides a parameter in the base panel with the same name.
- A list that appears in the derived panel overrides a list in the base panel with the same name.
- Attributes, parameters, lists and subpanels, that are omitted in the derived panel, are inherited from the base panel.
- A derived panel may specify attributes, add lists, parameters and subpanels, that do not appear in the base definition.
- Subpanels of the derived panel implicitly extend their counterparts (subpanels with the same name) in the base panel; all rules apply to them as well.

During instantiation, these rules are to be applied when resolving attributes, subpanels, parameters and lists in the current panel.

Let us examine an example to make this clear:

```
<panel name="/base">
  <panel name="nested">
    <param name="param1"/>
    <param name="param2" value="p2"/>
    <panel name="nested1nested"/>
  </param>
</panel>

<panel name="/derived" super="/base" template="/WEB-INF/derived.jsp">
  <panel name="nested" template="/WEB-INF/nested1.jsp">
```

```
    <param name="param1" value="p1"/>
    <param name="param2" value="override p2"/>
    <param name="param3" value="add p3"/>
  </param>
  <panel name="nested2" template="/WEB-INF/nested2.jsp"/>
</panel>

<panel name="/concrete" template="/WEB-INF/derived2.jsp">
  <panel name="foo" super="/derived">
    <panel name="nested">
      <panel name="nested1nested" template="/WEB-INF/bar.jsp"/>
      <param name="param3" value="override p3"/>
    </panel>
  </panel>
</panel>
```

Don't run away! It looks harder than it is... Let's work out what happens here:

- Toplevel panel `/base` defines an abstract subpanel `nested`. It is abstract, because it does not specify a `template` attribute and also, because it has an abstract subpanel `nested1nested`. Furthermore, the `nested` subpanel contains the undefined parameter `param1`.
- Panel `/derived` extends `/base` and assigns a `template` value to subpanel `nested`, defines a value for `nested`'s `param1`, overrides the `param2` parameter and adds the `param3` parameter. Finally, it adds `nested2`, another nested panel. However, `/derived` is still abstract, because it inherits the abstract `nested1nested` panel.
- The `/concrete` panel definition takes a `foo` subpanel, which is derived from `/derived`. The `foo` panel overrides `param3`. Since it assigns a `template` attribute to `nested1nested`, `foo` is concrete and thus `/concrete` is concrete.

2.6. I18n

The *Calyxo Panels* component supports locale-dependent variants of panel definitions. This advanced feature may be used to define different view variants, depending on the user's locale settings.

A locale specifies a *language* code, and may additionally specify a *country* code. If it does so, it may also specify a *variant* code (see class `java.util.Locale`). The *generalization* of a locale is defined by stripping off the last and least significant code. For example, generalizing `en_US` leads to `en`. Generalizing `en` leads to an unspecified locale.

Localized panel definitions are collected into separate groups of `<panels>` elements. According to the locale properties, the `<panels>` element accepts attributes `language`, `country` and `variant` to specify a locale for the contained `<panel>` elements.

During instantiation, the process of searching attributes, subpanels, parameters and lists not only uses inheritance, but also generalization: if the attribute or element we're looking for is not available for the desired locale, the search proceeds using the generalized locale. In particular, generalization is used:

- when looking up a toplevel panel
- when searching for a base panel
- when searching for an associated template
- when looking up a subpanel, parameter or list

Finally, let's consider some examples:

```
<panels>
  <panel name="/address.page" template="/WEB-INF/address.jsp">
    <param name="foo" value="bar"/>
  </panel>
</panels>

<panels language="de">
  <panel name="/address.page" template="/WEB-INF/address_de.jsp"/>
</panels>
```

Depending on the desired locale, instantiation of `/address.page` will use the appropriate definition. Generalization will cause parameter `foo` to be found when using a german locale.

```
<panels>
  <panel name="/base.page" template="/WEB-INF/base.jsp">
    ...
  </panel>

  <panel name="/derived.page" super="/base.page">
    ...
  </panel>
</panels>

<panels language="de">
  <panel name="/base.page">
    ...
  </panel>
</panels>
```

Depending on the desired locale, instantiation of `/derived.page` will use the appropriate definition of `/base.page` to inherit from. Furthermore, generalization will cause `/WEB-INF/base.jsp` to be identified as associated template.

3. Reference

3.1. Configuration

Throughout this reference, required attributes appear **strong**. Dynamic attributes (attributes, whose value may contains EL expressions) appear *emphasized*.

The elements described in the following sections are defined within namespace

```
http://calyxo.odysseus.de/xml/ns/panels
```

If the XML parser doesn't support XML Schema, DTD validation has to be used by declaring the *Calyxo Panels* document type as in:

```
<!DOCTYPE calyxo-panels-config
  PUBLIC "-//Odysseus Software GmbH//DTD Calyxo Panels 0.9//EN"
  "calyxo-panels-config.dtd">
```

Elements

Name	Description
`calyxo-panels-config`	Root element of a *Calyxo Panels* configuration file.
`panels`	Define a set of panel definitions, optionally specify a locale.
`panel`	Define a panel.
`param`	Define a parameter.
`list`	Define a list.
`item`	Define a list item.

3.1.1. The <calyxo-panels-config> Element

Purpose

The `<calyxo-panels-config>` element is the root element of a *Calyxo Panels* configuration file.

As common to all of *Calyxo*'s configuration files, the root element uses the `xmlns` attribute to specify the namespace and the `version` attribute to specify the XML schema/DTD version.

Attributes

Name	Type	Description
`xmlns`	`CDATA`	Required - XML namespace. Must be `http://calyxo.odysseus.de/xml/ns/panels`.
`version`	`NMTOKEN`	Required - DTD/Schema version number. Must be `0.9`.

Body

The body of the `<calyxo-panels-config>` element is defined by the following sequence:

```
(base:import*, (base:functions | base:set | base:use)*, panels*)
```

The first four elements are common to all *Calyxo* components. They are described in the

Calyxo Base configuration reference. The <panels> element and its children are described in the following sections.

Related elements

<panels>

3.1.2. The <panels> Element

Purpose

The <panels> element defines a set of panel definitions.

Attributes

Name	Type	Description
language	NMTOKEN	The language code of the locale, which applies to all panels in this <panels> section.
country	NMTOKEN	The country code of the locale, which applies to all forms in this <panels> section. (Setting the country attribute does only make sense, if the language is also specified.)
variant	NMTOKEN	The variant code of the locale, which applies to all forms in this <forms> section. (Setting the variant attribute does only make sense, if language and country are also specified.)

Body

The body of the <panels> element is defined by the following sequence:

(panel*)

Related elements

<panel>

3.1.3. The <panel> Element

Purpose

The <panel> element defines a panel.

Attributes

Name	Type	Description
name	NMTOKEN	Required - The name of the panel.
super	NMTOKEN	Name of a toplevel panel, from which this panel inherits.
template	CDATA	Dynamic - The panel's template path.

Body

The body of the <panel> element is defined by the following sequence:

```
(param | panel | list)*
```

Related elements

<param>, <panel>, <list>

3.1.4. The <param> Element

Purpose

The <param> element defines a parameter.

Attributes

Name	Type	Description
name	NMTOKEN	Required - The parameter name.
value	CDATA	Dynamic - The parameter value.

Body

The <param> element has no body.

Related elements

<panel>, <item>

3.1.5. The <list> Element

Purpose

The <list> element defines a list of items. Each item may contain panels, params and lists.

Attributes

Name	Type	Description
name	NMTOKEN	Required - The name of the list.

Body

The body of the <list> element is defined by the following sequence:

```
(item*)
```

Related elements

<item>

3.1.6. The <item> Element

Purpose

The <item> element defines a list item.

Attributes

The <item> element has no attributes.

Body

The body of the <item> element is defined by the following sequence:

```
(param | panel | list)*
```

Related elements

<param>, <panel>, <list>

3.2. Accessors

The ${calyxo.panels} accessors provides access to data related to the *Calyxo Panels* component. They are automatically registered to the *Calyxo Base*' access support when the module is loaded.

To make the accessors available in JSP pages, the <base:access> tag is used to install the access tree into request scope:

```
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.0"
  xmlns:base="http://calyxo.odysseus.de/jsp/base">
    ...
    <base:access var="calyxo"/>
```

```
    ...
</jsp:root>
```

In our examples, we assume, that the accessors have already been installed at request attribute `calyxo`.

3.2.1. The panels.param accessor

param[name]

Searches for a parameter with the specified name in the current panel. If the parameter exists, it evaluates to its value. Otherwise, an Exception is thrown.

Example

The expression `${calyxo.panels.param['foo']}` searches the current panel for a parameter named `foo` and evaluates to its value.

3.3. Tag Library

The *Calyxo Panels* custom tag library contains tags to include panels, access parameters and iterate over lists. In a JSP file, just associate the prefix you want to use for the tags with URI `http://calyxo.odysseus.de/jsp/panels`:

```
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.0"
  xmlns:panels="http://calyxo.odysseus.de/jsp/panels">
  ...
</jsp:root>
```

Since the tag library descriptor is contained in the *Calyxo Panels* jar file, it is already available to applications. The container will automatically find it. For documentation purposes, a copy is located in `CALYXO_HOME/calyxo-panels/conf/share/calyxo-panels.tld`.

Tags

Name	Description
`panel`	Include a subpanel (or toplevel panel).
`param`	Pass a dynamic parameter value to a panel.
`list`	Iterate over a list of items.

3.3.1. The <panel> Tag

Purpose

Include a subpanel of the current panel or a toplevel panel by name.

Attributes

Name	Description
name	Required - Specifies the name of a panel to be searched in the current panel.

Body

The <panel> tag may have a body containing <param> tags. The tag body is evaluated, but not included.

Related tags

<list>, <param>

3.3.2. The <param> Tag

Purpose

Pass a dynamic parameter to a panel.

Requirements

A <param> tag has to be embedded in a <panel> tag.

Attributes

Name	Description
name	Required - Specifies the name of the parameter.
value	Required - Specifies the value of the parameter.

Body

The <param> tag has no body.

Related tags

<panel>, <list>

3.3.3. The <list> Tag

Purpose

Iterate over a list contained in the current panel.

Attributes

Name	Description
`name`	Required - Specifies the name of a list to be searched in the current panel.
`varStatus`	If set, specifies the name of a page scope attribute. The tag will export a *list status* object to that attribute. The list status object has methods `boolean isFirst()`, `boolean isLast()` and `int getIndex()`, which may be used to retrieve information about the position of the current item.

Body

The `<list>` tag has no body.

Related tags

`<panel>`, `<param>`

4. Integration

In order to use the *Calyxo Panels* component, it must be somehow integrated into the application's controller. This is achieved by the use of *plugins*.

- The component comes with a plugin for *Calyxo Control*.
- The *Calyxo Struts* component provides a similar plugin for Struts.

In either environment, the plugins modify the controller's dispatch behavior: before the 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.

Furthermore, the plugins register the *Calyxo Panels* accessors.

4.1. Panels Plugin for Calyxo Control

The plugin provides a *dispatcher* named `panels` for the *Calyxo Control* component. The `panels` dispatcher can be installed as the default dispatcher for the module. Alternatively, the `panels` dispatcher can be specified for a single `action` element or even for a single `dispatch` element.

The plugin is loaded in the module's controller configuration:

```
<calyxo-control-config version="0.9"
  xmlns="http://calyxo.odysseus.de/xml/ns/control">
  ...
  <plugin class="de.odysseus.calyxo.panels.control.PanelsPlugin">
    <param name="config" value="/WEB-INF/calyxo-panels-config.xml"/>
    <param name="global" value="true"/>
  </plugin>
  ...
</calyxo-control-config>
```

The mandatory `config` parameter specifies the *Calyxo Panels* configuration file.

If the optional `global` parameter is set to `"true"`, the panels dispatcher is installed as the default dispatcher.

4.2. Panels Plugin for Struts

The panels plugin for Struts enables full use of the *Calyxo Panels* component with Struts. Plugin configuration details are covered in the *Calyxo Struts*' Panels Plugin documentation.

5. Project

5.1. History of Changes

Version 0.9.0 (2006/10/28)

developer: **cbe** ***context:*** **code** ***type:*** **add**
Added `PanelsContext.isEmpty()`.
developer: **cbe** ***context:*** **code** ***type:*** **update**
Minor refactoring to eliminate package cycle between `control` and `control.misc`.
developer: **cbe** ***context:*** **code** ***type:*** **update**
Make sure to pop() panels context when an exception is thrown while forwarding to or including templates.

Version 0.9.0-rc3 (2006/02/12)

developer: **cbe** ***context:*** **code** ***type:*** **update**
Renamed webapp `demo` to `test`.
developer: **cbe** ***context:*** **code** ***type:*** **update**
Eliminated dependencies on `commons-collections`.
developer: **cbe** ***context:*** **code** ***type:*** **update**
Cache `I18nSupport` instance in `PanelsDispatcher`.
developer: **cbe** ***context:*** **code** ***type:*** **update**
In the <panels>, evaluate dynamic parameters *before* pushing the included template on top of the stack to allow passing through parameters using the `panels.param` accessor.

Version 0.9.0-rc1 (2005/03/07)

developer: **cbe** ***context:*** **code** ***type:*** **update**
Changed namespace to `http://calyxo.odysseus.de/xml/ns/panels`.
developer: **cbe** ***context:*** **code** ***type:*** **update**
Renamed panel's `dispatch` attribute to `template`.

Version 0.9.0-b5 (2005/01/04)

developer: **cbe** ***context:*** **docs** ***type:*** **update**
Minor documentation changes.

Version 0.9.0-b4 (2004/11/21)

developer: **cbe** ***context:*** **code** ***type:*** **update**
Updated controller plugin/dispatcher code to `calyxo-control-0.9.0-b4`.
developer: **cbe** ***context:*** **code** ***type:*** **add**
Added class `DynamicDispatchConfig`.

Version 0.9.0-b3 (2004/10/20)

developer: **cbe** ***context:*** **docs** ***type:*** **update**
Improved documentation.
developer: **cbe** ***context:*** **code** ***type:*** **add**
dispatch parameters from control-config are now added as request parameters.
developer: **cbe** ***context:*** **code** ***type:*** **update**
`<panel name="...">` tag now also searches top-level panels.
developer: **cbe** ***context:*** **code** ***type:*** **add**
Added `varStatus` attribute to `<list>` tag.
developer: **cbe** ***context:*** **code** ***type:*** **update**
Changed `<param>` tag semantics! The `<param>` tag is now used inside a `<panel>` tag to specify a dynamic parameter. Use the `${calyxo.panels.param[...]}` accessor to access parameters.

Version 0.9.0-b2 (2004/07/04)

developer: **cbe** ***context:*** **docs** ***type:*** **add**
Added documentation
developer: **cbe** ***context:*** **design** ***type:*** **update**
Renamed `<include>` tag to `<panel>`

Version 0.9.0-b1 (2004/06/03)

developer: **cbe** ***context:*** **admin** ***type:*** **add**
First public release

5.2. Todo List

medium priority

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