Struts Bridge v.1.0.1

Project Documentation

Apache Software Foundation 04 March 2007

Table of Contents

1.1 Summary

Summary

The Struts Bridge allows Struts and Struts Tiles applications to be run as JSR-168 compliant Portlets.

Existing or new Struts Applications can be transparently deployed as Portlet Application or Web Application.
The Bridge wraps and enhances the native Struts Framework to overcome its limitations and incompatibilities with the Java Portlet Standard requirements.

An existing Struts Application can be used as a Portlet *without* changes to the code or JSP files if:

- a few common sense rules, based on a proper MVC architecture, are followed for the Struts Action configurations
- Struts Tags are used for rendering all resource paths (like images) and action links

If a Struts Portlet doesn't use Portlet specific features, it can also be accessed and used as Web Application at the same time. Even testing the Struts Application can then be done completely independent of a Portal.

The Struts Bridge is developed to be independent from specific Portals and uses only a very small interface to the Portal to be able to get access to the Servlet environment at runtime. As all JSR-168 Portlet Containers are required to build upon the Servlet specification, providing this interface for a specific Portal is usually very easy to do, if not done already.

The Apache Portals Jetspeed-2 Portal provides this interface natively, as well as example Struts Portlet Applications using the Struts Bridge like a JPetstore Portlet.

1.2 Features

Features

The Struts Bridge provides the following features for developing and running Struts Applications as Portlets:

- Provides a "virtual" page context and a "normal" page url to the Struts Application
- Allows Servlet invocation from an ActionRequest using an separate lightweight interface to the Portal
- Provides extension points if a more complex interface to the Portal is required
- Automatically separates Struts Action Processing and View Rendering when accessed from an ActionRequest
- Transparent communication of request attributes between ActionRequest and RenderRequests
- Transparent translation of "normal" Web application resource and action link urls to valid Portlet urls
- Tiles support and automatic configuration of the correct (Portlet)RequestProcessor (enhanced for 1.0.1)
- Deploy and run Struts Applications as Portlet and as Web Application at the same time
- 1.0.1: Multiple StrutsPortlet instances with the StrutsServlet session isolated to PORTLET_SCOPE
- Supports Struts release 1.2.7

Struts Page URL

Struts determines the Actions to perform on the request url. A Portlet on the other hand doesn't have or receive an url, only request parameters.

To be able to use Struts within a Portlet, a current Page context is maintained by the Struts Bridge which is used to provide Struts with a "virtual" request context just as if it were accessed from a Web Application.

The Struts Page URL is maintained as a Portlet `RenderParameter` and included in all Portlet URLs generated for the Portlet. When the `StrutsPortlet` receives a request it uses the Struts Page URL to create a new `HttpServletRequest` with a redefined request url before invoking the Struts `PortletServlet` (extending the Struts `ActionServlet`).

None of the Action Mappings for an existing Struts Application have to be changed (in this respect at least) to allow it to run in a Portlet context.

The ServletContextProvider interface

Processing `ActionRequest` and `RenderRequest` events from a Servlet application is a problem with the current Portlet Specification. Although Servlets (and JSPs) can be invoked from a Portlet, the Portlet

Specification only supports this from the `RenderRequest`.

As request parameters are only available during the `ActionRequest`, not being able to use a Struts for processing these is a serious limitation. Together with no clear inter-portlet application communication support, this is probably one of the biggest omissions from the Portlet Specification.

But, because the Portlet Specification is build upon the Servlet Specification (2.3), the Portlet Container implementation (like Pluto) and/or the Portal itself, usually has direct access to, and also uses the `ServletContext`, even during the `ActionRequest` event.

It is therefore usually quite easy to provide access to the `ServletContext`, although it will require some knowledge of the Portal and/or the Portlet Container implementation.

The Struts Bridge uses the `ServletContextProvider` interface from the portals-bridges-common library to get access to the required `ServletContext`.

To be able to use the Struts Bridge, a `StrutsPortlet` must therefore be provided with a concrete implementation of the `ServletContextProvider` interface (through its initialization parameters). This implementation class can be supplied together with the Portlet or be provided natively by the Portal as the Jetspeed-2 Portal does.

Portals requiring a more complex interface

Although the `ServletContextProvider` interface has been defined to be easily implemented for a specific Portal, it might not be generic enough for all.

To support those kind of Portals, the `StrutsPortlet` doesn't uses the interface directly, but only through protected methods of its own which can be extended if needed.

Handling ActionRequests

During an `ActionRequest` a Portlet (or invoked Servlet) is not allowed to write content to the response. This is only allowed during the `RenderRequest` event but then a Portlet doesn't have access anymore to the (input) request parameters which makes the RenderRequest generally useless (and bad practice anyway) for (Struts) Action processing.

Struts on the other hand only knows one request event and is largely build upon server-side forwarding to a JSP (or other View rendering technology) after an Action has been processed.

The Struts Bridge solves this problem by allowing Struts Action processing to occur during the ActionRequest. But, as soon as Struts forwards (or includes) to another handler (JSP, Velocity script, *or even another Action: ActionChaining*) it will intercept and freeze the processing.
The Struts Bridge will then save the current processing context (like the path to include or forward to) in a separate object: the `StrutsRenderContext`, and end the ActionRequest.
Once the RenderRequest is invoked, the Struts Bridge will use the `StrutsRenderContext` to restore the state Struts (and/or an included or forwarded JSP) will expect and continues the processing where it left off.

Also saved in the StrutsRenderContext are the current `ActionForm` (if it isn't already session bounded), the `ActionMessages` and/or `ActionErrors` (if not already session bounded which is possible since

Struts 1.2.1).

Important: the StrutsRenderContext is restored only once!

The Portlet Specification expects Portlets to use a true MVC architecture: changing the state of a Portlet is expected to be done during ActionRequest only. The RenderRequest, which may occur many times, is expected to only render the current state of the Portlet.

For a proper usage of the Struts Bridge and to conform to the Portlet Specification requirements, the Struts application too really should be configured to use separate Action processing and View rendering Actions.

The dependency on the StrutsRenderContext restoring the state during the (next) RenderRequest should be limited to one situation only: input error processing.

When Struts encounters input validation errors, it forwards (back) to the page specified as input for the Action. The Struts Bridge handles this situation likewise. When it detects ActionErrors are defined after the ActionRequest event, it will reset the Struts Page URL to the one from which the ActionRequest was initiated so subsequent (multiple) RenderRequest events will always render that original page.

Note: this will only result the same behavior as when the Struts Application is accessed as Web Application if the Action Mapping input attribute indeed points to the originating input page, which normally will be the case. *The Struts Bridge will not (re)set the Struts Page URL to the Action input attribute value (Struts itself will forward to that url) but to the page from which the ActionRequest was initiated!*

As said, for a proper usage of the Struts Bridge the Struts application should be configured to use separate Action processing and View processing Actions. The Struts Bridge intercepts the first include or forward initiated from a Struts Action during the ActionRequest. During only the first following RenderRequest it will invoke Struts with *the same Struts Page URL* as for the ActionRequest but intercept before the actual execution of the mapped Action and then include (never forward) the intercepted target url of the ActionRequest (normally a JSP).

A properly MVC configured Struts application though can (and should) make use of a (client-side) redirect after the Action processing Action by using an ActionForward with redirect="true". The behavior of such a Struts Application then truly complies to the requirements for a Portlet. In fact, the reference implementation of the Portlet Specification, Pluto , actually sends a redirect to the client (browser) after an ActionRequest.

The Struts Bridge supports this behavior as it will intercept a client-side redirect during an ActionRequest by changing the Portlet Page URL to be used by subsequent RenderRequests to the targeted redirect url.

A Struts Application not complying to the above "rules" might not behave as expected in a Portlet context:
without this configuration a second RenderRequest event will not get the previously saved StrutsRenderContext restored and the invoked Struts Action will be executed again (this time without any parameters).

Migrating an existing Struts Application to a Struts Portlet with doesn't comply to these "rules" might seem problematic without major changes to the code, but it doesn't have to be.

RenderContextAttributes to the rescue

Although a StrutsRenderContext can only be used once to communicate request parameters from the ActionRequest to the next RenderRequest (stored in the ActionForm), the Struts Bridge provides another, completely transparent, solution which can be used without any required changes to the Struts Application.

The Struts Bridge can be configured, using a simple xml definition, to always save specific named request attributes during an ActionRequest and restore these only during the next RenderRequest or optionally every RenderRequest until the next ActionRequest or a RenderRequest for a different Struts Page URL is invoked.

The request attribute values will be saved in the web session which requires them to implement the `Serializable` interface.

An example struts-portlet-config.xml (the name of this file is configurable) definition might look like:

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
  <render-context>
    <attribute name="errors"/>
    <attribute name="message" keep="true"/>
    <attribute prefix="com.foo.bar" keep="true"/>
  </render-context>
</config>
```

(This (adapted) example is taken from the JPetstore Portlet example provided with the Jetspeed-2 Portal.)

With the above example configuration, all request attributes named *errors* and *message* are saved after the ActionRequest. The *errors* object is restored only once, the *message* object will be restored until the next ActionRequest or when the Struts Page URL is changed.

Furthermore, every request attribute which name start with *com.foo.bar* is also saved after the ActionRequest. With the "prefix" attribute type definition, a whole range of attributes can be configured at once.

Using the RenderContextAttributes configuration, a Struts Action can still forward (without redirecting) to a JSP in a Portlet while passing its required data through the request attributes. Request scoped ActionForms can also safely be used for this as the Struts Bridge will automatically remove them from the session again when they go out of scope!

Using the "single use only" configuration (meaning: without the keep="true" attribute) allows one to pass on rather large objects to the RenderRequest because, even if they are stored in the Session, this will only be for a very short period (milliseconds). Of course, with a subsequent RenderRequest these objects won't be available anymore and the specific View Renderer must be able to handle that situation properly. Error objects or error messages (other than ActionMessages and ActionErrors which are already automatically saved in the StrutsRenderContext) are also reasonable candidates to use like this.

Rendering valid Portlet urls with the Struts JSP Tags

Another problem with using Struts for Portlet Development is rendering valid urls.

There are actually two different type of problems: urls for resources likes images, Javascript or CSS

stylesheets which needs to be loaded by the client (browser), and urls for interacting with the Portlet itself.

The problem with resource urls is that the content of a Portlet is only included by the Portal, together with possible content of other Portlets shown on the same page. Although the Portlet runs within its own context (the same as the web application it is based on), the client (browser) sees only the context of the Portal page it is currently pointed at.

A Portlet therefore cannot make use of relative src locations for a resource url while this is perfectly valid (and usually preferred) in a Web Application. For a client to be able to retrieve a resource, its url really must include the full context path of the Web Application.

For interacting with the Portlet special PortletURLs must be used which can only be generated using the Portlet API or through the Portlet JSP Tags. The Struts JSP Tags only generate urls valid for a Web Application and also doesn't provide some kind of URLProvider interface like JSF.

Furthermore there are two different PortletURL types: ActionURLs and RenderURLs. It is very important to generate the correct type of url for a certain interaction. Forms must *always* use a POST to an ActionURL, but for generated links it will have to be determined which type of url actually is needed.

To solve these problems, the Struts Bridge provides enhanced versions of the Struts HTML JSP Tags (including EL variants), which take the Portlet environment into account. Furthermore, using a separate element in the struts-portlet-config.xml definition, specific url paths can be configured to be of a certain type (Resource, ActionURL, RenderURL) only.

For the enhanced Struts HTML JSP Tags (html:form, html:link, html:rewrite, html:image and html:img), four different TLDs are provided: an EL and non-EL variant, and for both a full replacement of the Struts supplied struts-html.tld as well as a separate struts-portlet.tld containing only the enhanced Tags definitions.

The full replacement TLDs makes it very easy for migrating a Struts Application to a Portlet context: simply map the new TLD in web.xml or redefine the taglib uri in the JSP using:

```
            <%@ taglib
uri="http://portals.apache.org/bridges/struts/tags-portlet-html" prefix="html" %>
```

or

```
            <%@ taglib
uri="http://portals.apache.org/bridges/struts/tags-portlet-html-el" prefix="html-el"
%>
```

(Note: the Struts Bridge library contains all the TLDs so you don't need to add them yourself somewhere under WEB-INF/.)

For the enhanced html:link and html:rewrite Tags, three additional boolean attributes are defined: actionURL, renderURL and resourceURL. Using these attributes one can specify which type of url must be generated (only value "true" is supported).

If none of these attributes is specified (with value "true") these Tags will by default generate a RenderURL.

But, the default can be changed (only between renderURL and actionURL) using a struts-portlet-config.xml definition. Furthermore, for specific url path (prefixes) you can specify which type of url must be generated (but with one of the three additional boolean url type attributes this can always be overriden).

An example struts-portlet-config.xml with portlet url type definitions might look like:

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
  <render-context>
    <attribute name="errors"/>
    <attribute name="message" keep="true"/>
  </render-context>
  <portlet-url-type>
    <action path="/shop/add"/>
    <action path="/shop/switch"/>
    <action path="/shop/remove"/>
    <action path="/shop/signoff"/>
    <action path="/shop/viewCategory"/>
    <action path="/shop/viewItem"/>
    <action path="/shop/viewProduct"/>
    <action path="/shop/viewCart"/>
    <action path="/shop/newOrder"/>
    <render path="/shop/newOrderForm"/>
    <action path="/shop/listOrders"/>
    <resource path="/images/"/>
  </portlet-url-type>
</config>
```

(This is the full configuration used for the JPetstore Portlet example provided with the Jetspeed-2 Portal.)

In the above example the portlet-url-type element contains three different sub elements: action, render and resource. Their path attribute specifies the prefix to which specific urls are matched. This matching will be done using best match. Internally, the paths will be ordered on longest definition first.

Not shown in the above example is the optional default="render"|"action" attribute for the portlet-url-type element. With it, the default type of portlet-url can be changed from "render" to "action" as is used in the Struts Mailreader Demo Portlet (also provided by the Jetspeed-2 Portal):

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
  <portlet-url-type default="action"/>
</config>
```

This feature is especially useful for Struts Applications who were configured using an earlier version of the Struts Bridge (0.1) which used an ActionURL as default, like the above Struts Mailreader Demo Portlet.

The enhanced html:img and html:image Tags, as well as the html:link and html:rewrite Tags, can be used to generate resource urls using relative (not prefixed with a '/') src references. The Tags will then

automatically determine the correct url to generate using the current Struts Page URL as defined for this RenderRequest.

This allows one to replace hard coded resource urls like

```
<img src="../images/struts-bridge.gif"/>
```

with

```
<html:image src="../images/struts-bridge.gif"/>
```

A simple change with no technical impact, but it will allow the same JSP to be used for both a Web Application and a Portlet. The JPetstore Portlet example as provided by Jetspeed-2 makes full use of this feature.

PortletRequestProcessor and PortletTilesRequestProcessor

The Struts Bridge requires a different RequestProcessor to be used for the Struts Application: a `PortletRequestProcessor`. If a Struts config doesn't define one, or defines one which isn't based on the PortletRequestProcessor, it will be replaced automatically by the Struts Bridge.

The Struts Bridge also supports Tiles to be used with the `PortletTilesRequestProcessor`. If the TilesPlugin is defined (as required to be able to use Tiles in Struts) it will be recognized by the Struts Bridge and then the PortletTilesRequestProcessor will be used instead (if not configured already).

Tiles can be used without problem within a Portlet context, even for Action Mapping or ActionForward paths, although the same considerations and restrictions must be taken into account as described above.

1.0.1: For proper handling of the Tiles Context, with version 1.0 the Tiles ComponentConstants.COMPONENT_CONTEXT had to be set as RenderContextAttribute like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
  <render-context>
    <attribute name="org.apache.struts.taglib.tiles.CompContext"/>
    ...
  </render-context>
  ...
</config>
```

With the 1.0.1 release, this isn't necessary anymore as the Struts Bridge will register this special attribute itself automatically if it hasn't been defined in the struts-portlet-config.xml already.

Running your application as Portlet and Web Application at the same time

Although the Struts Bridge is created to allow Struts Applications to run in a Portlet context, it also allows the same application to be run in a Servlet context too.

If the Struts Bridge PortletServlet (which extends ActionServlet) detects it is not accessed from a Portlet

context is simply delegates back to the underlying Struts Framework.

Provided the Struts Application doesn't use any Portlet specific features and the Action Mapping configuration is also (still) valid for a Web Application, the same application war can be deployed as Web application and as Portlet Application. When deployed in a Portal, the application can even be accessed as Portlet or as Web Application at the same time!

The JPetstore Portlet example provided by Jetspeed-2 is one example of such a "dual" mode Struts Application.

Another benefit of such a "dual" mode Struts Application is that it can be tested as Web Application which *is* somewhat easier than testing it as a Portlet.

1.0.1: Multiple StrutsPortlet instances PORTLET_SCOPE isolated StrutsServlet sessions

The Struts framework itself, as well as many Struts applications make have use of the (Servlet) Session for storing user state. When you want to use multiple StrutsPortlets from the same web application this can cause conflicts and session state corruption. With Struts Bridge release 1.0.1, this now has been solved by (optionally, and not by default) isolating the Session scope seen by the StrutsServlet to PortletSession.PORTLET_SCOPE.

If the optional StrutsPortlet init parameter `PortletScopeStrutsSession` is set to true, the Session object provided to the StrutsServlet will be a Proxy for the PortletSession and will provide only access to the PortletSession.PORTLET_SCOPE attributes.

One caveat though: If you also need to use direct Servlet access to the Struts application (meaning: the browser is invoking the Servlet directly, not through the Portlet), this Servlet won't "see" the previously APPLICATION_SCOPE Session attributes anymore.
This can be an issue for features like binary file (pdf) rendering through a servlet which needs to use the Session to access its data. To solve this, the utility PortletWindowUtils class can be used which provides methods to access a specific Portlet instance its PORTLET_SCOPE session attributes based on its PortletWindowId.
That PortletWindowId (which a Portlet can retrieve itself using the PortletWindowUtils) needs to be passed on to the Servlet first (for example as query-string parameter on a link send to the browser) for it to be able to make use of it.