4

Generic Faces Portlet

The portlet bridge isn't a portlet. It is a distinct subsystem a portlet acquires, instantiates, and executes when it needs to handle requests that target Faces artifacts. Though the bridge has a simple API and relies on familiar lifecycles and mechanisms, many portlets will find it more convenient to subclass an object that manages the details of the bridge's execution than to manage the bridge directly. The `GenericFacesPortlet` is provided for this purpose. Though a client of the bridge, `GenrericFacesPortlet` is included as part of the specification to provide an out-of-the-box solution for portlet developers as well as to provide portlet developers a better appreciation of how and how much work is involved in using the bridge.

4.1 Configuration

To provide flexibility, the `GenericFacesPortlet` recognizes the following application initialization parameters:

`javax.portlet.faces.BridgeClassName`

`BridgeClassName` specifies the name of the bridge class the `GenericFacesPortlet` uses to instantiate bridge instances for all portlets in this web application. If not specified, the `GenericFacesPortlet` defaults to finding the class name by looking for the context resource (file) called "`META-INF/services/javax.portlet.faces.Bridge`" and reading the first line (excluding preceding and succeeding white space).

Because bridge implementations are expected to include such a context resource file in their jar file, portlet applications relying on the `GenericFacesPortlet` typically do not set this initialization parameter. It is provided for those environments where multiple such resource files exist (because there are multiple bridge implementations in the environment) and resource resolution doesn't yield the desired class name.

In addition the `GenericFacesPortlet` reads the following portlet initialization parameters and either sets the appropriate context attributes[4.1] to direct the bridge's execution or uses it to impact its own behavior:

`javax.portlet.faces.defaultViewId.[mode]`

`javax.portlet.faces.excludedRequestAttributes`
`javax.portlet.faces.preserveActionParams`
`javax.portlet.faces.bridgeEventHandler`
`javax.portlet.faces.bridgePublicRenderParameterHandler`
`javax.portlet.faces.autoDispatchEvents`
`javax.portlet.faces.defaultRenderKitId`

`javax.portlet.faces.defaultViewId.[mode]` specifies the name of the `defaultViewId` used for a named portlet `PortletMode`. Note: *mode* is the `String` form of portlet `PortletMode`.

`javax.portlet.faces.excludedRequestAttributes` is a comma delimited list of attribute names the bridge is to exclude from managing in the bridge request scope. The values in this list will either identify a specific attribute name to exclude or a set of attributes. The later is identified by a String value with the wildcarded suffix "*.*". *Such wildcard usage indicates the bridge is to exclude all those attributes within the namespace identified by removing the*"", non-recursive.

`javax.portlet.faces.preserveActionParams` is a boolean valued String that when true indicates the bridge must maintain the action's request parameters for the duration of the bridge request scope 5.1.2. When this initialization parameter isn't present or is `false` the action's request parameters are only maintained for the duration of the portlet request scope. The exception to this is the `ResponseStateManager.VIEW_STATE_PARAM` parameter which is always maintained in the bridge request scope regardless of this setting.

`javax.portlet.faces.bridgeEventHandler` is a `String` naming the class that implements the `javax.portlet.faces.BridgeEventHandler` interface that the bridge uses to process events. See section 5.2.5 for detail concerning event support.

`javax.portlet.faces.bridgeRenderParameterHandler` is a `String` naming the class that implements the `javax.portlet.faces.BridgePublicRenderParameterHandler` interface that the bridge calls after pushing mapped public render parameters into beans. This handler gives the portlet an opportunity to recompute and get into a new consistent state after such changes. See section 5.3 for detail concerning public render parameter support.

`javax.portlet.faces.autoDisptachEvents` is a boolean valued `String` defining whether the bridge overrides regular portlet event processing dispatching all events to the bridge. If `true` the `GenericFacesPortlet` overrides processEvent() and dispatches all events to the bridge. If `false` the `GenericFacesPortlet` delegates all event processing to the standard portlet model. Use `true` if the portlet is written entirely in JSF. Use `false`, if your portlet uses a mixture of view technologies. In the later case, the portlet is responsible for recognizing the events which should be handled in JSF and dispatch them directly to the bridge. The default value is `true`. See section 5.2.5 for detail concerning handling events.

`javax.portlet.faces.defaultRenderKitId` is a `String` defining this portlet's default renderkit Id. Providing a value will cause the `GenericFacesPortlet` to configure the bridge to use this specific default renderkit Id for the portlet rather than application level default (set in the `faces-config.xml`).

There are two portlet initialization parameters that were recognized by the portlet 1.0 bridge that are no longer recognized and used in the portlet 2.0 bridge:

```
javax.portlet.faces.defaultContentType
javax.portlet.faces.defaultCharacterSetEncoding
```

These parameters were used by the GenericFacesPortlet to preset the response content type before delegating to the bridge to handle the request. Because the portlet 2.0 bridge now uses dispatch.forward() to render Faces views that are jsps, this presetting is no longer needed. The requirement to recognize these init parameters no longer exists and their presence is ignored.

4.2 Structure

The `GenericFacesPortlet` subclasses `javax.portlet.GenericPortlet`. It overrides the `init`, `destroy`, `doDispatch`, `doEdit`, `doHelp`, `doView`, and `processAction` methods. In addition it defines the following new methods:

```
getBridgeClassName
getDefaultViewIdMap
getExcludedRequestAttributes
isPreserveActionParameters
getResponseContentType
getResponseCharacterSetEncoding
getBridgeEventHandler
getBridgePublicRenderParameterHandler
isAutoDispatchEvents
getFacesBridge
```

`getBridgeClassName()` allows a subclass to override the GenericFacesPortlet mechanism for determining which bridge class to instantiate 4.2.6.

`getDefaultViewIdMap()` allows a subclass to override the `GenericFacesPortlet` mechanism for determining the `Map` of the default `viewId` that should be used in each `PortletMode` when the request doesn't otherwise indicate a specific view 4.2.7. This `Map` contains one entry per `PortletMode`. The entry key is the name of the `PortletMode`. The entry value is the mode's default `viewId`.

`getExcludedRequestAttributes()` allows a subclass to override the `GenericFacesPortlet` mechanism for determining the set of request attributes to exclude from the bridge request scope 4.2.8.

`getPreserveActionParameters()` allows a subclass to override the `GenericFacesPortlet` mechanism for determining whether the bridge needs to preserve action parameters for subsequent renders 4.2.9.

`getResponseContentType()` is deprecated as it is no longer called by the `GenericFacesPortlet`. It exists merely for backwards compatibility in the off chance that a subclass called it.

`getResponseCharacterSetEncoding()` is deprecated as it is no longer called by the `GenericFacesPortlet`. It exists merely for backwards compatibility in the off chance that a subclass called.

`getBridgeEventHandler()` allows a subclass to override the `GenericFacesPortlet` mechanism for determining the `BridgeEventHandler` the bridge should use to process events 4.2.12.

`getBridgePublicRenderParameterHandler()` allows a subclass to override the `GenericFacesPortlet` mechanism for determining the `BridgePublicRenderParameterHandler` the bridge should use to post process incoming public render parameters 4.2.13.

`isAutoDispatchEvents()` allows a subclass to override the `GenericFacesPortlet` mechanism for determining the setting for the `autoDispatchEvents boolean` 4.2.14.

`getFacesBridge()` primarily used by a subclass to get the `GenericFacesPortlet`'s bridge in situations where the subclass needs to directly call its `doFacesRequest` method. 4.2.15.

`getDefaultRenderKitId()` allows a subclass to override the `GenericFacesPortlet` mechanism for determining the renderkit Id (if any) the bridge conveys to Faces for use as the default renderkit Id when acquiring a `RenderKit` and rendering 4.2.16.

4.2.1 init():

The `GenericFacesPortlet` overrides the `init` method and does the following[4.2]:

- it determines the specific `Bridge` implementation class to use for this portlet by calling `getBridgeClassName()`.
- it calls `getExcludedRequestAttributes()`. The result of this call is set as a `PortletContext` attribute as per this specification 3.2.
- it calls `isPreserveActionParameters()`. The result of this call is set as a `PortletContext` attribute as per this specification 3.2.

- it calls `getDefaultViewIdMap()`. The result of this call is set as a `PortletContext` attribute as per this specification 3.2.
- it calls `getBridgeEventHandler()`. The result of this call, if not null, is set as a `PortletContext` attribute as per this specification 3.2.
- it calls `getBridgePublicRenderParameterHandler()`. The result of this call, if not `null`, is set as a `PortletContext` attribute as per this specification 3.2.
- it calls `getDefaultRenderKitId()`. The result of this call, if not null, is set as a `PortletContext` attribute as per this specification 3.2.

Note: instantiating and initializing the bridge may be deferred until it needs to dispatch the first request.

4.2.2 destroy():

The `GenericFacesPortlet` overrides the `destroy` method and does the following[NT]:

- if it has activated a bridge, it calls the bridge's `destroy()` method.
- it resets its internal data members to an uninitialized state.

4.2.3 doDispatch():

The `GenericFacesPortlet` overrides the `doDispatch` method and does the following[NT]:

- if the requested portlet `Mode` is `View`, `Help`, or `Edit`, it delegates to its superclass without doing any work.
- otherwise, for non-standard `Modes`, it processes the request as described in section 4.2.5.

4.2.4 doView(), doEdit(), doHelp(), processAction(), serveResource(), processEvent():

The `GenericFacesPortlet` overrides the `doView()`, `doEdit()`, `doHelp()`, `processAction()`, and `serveResource()` methods and executes the request via the bridge. I.e. portlets that want to override this behavior to detect nonFaces targets should either provide this logic in `doDispatch()` or in a subclassed method of the above.

The GenericFacesPortlet overrides the processEvents() method and if isAutoDispatchEvents() returns true executes this request via the bridge otherwise it delegates the handling of this event to its superclass. I.e. where the subclassing portlet has first shot at overriding the above methods, in the case of events, the GenericFacesPortlet typically has the first shot (assuming the portlet is using the preferred method of annotating event methods).

4.2.5 Request Processing

When not overridden by a subclass, the GenericFacesPortlet processes the request by first determining if the target of the request is a Faces or a nonFaces view. A nonFaces view target is recognized if the request contains the parameter _jsfBridgeNonFacesView. The value of this parameter is the ContextPath relative path to the nonFaces target. To handle this request the GenericFacesPortlet sets the response contentType, if not already set, using the preferred contentType expressed by the portlet container. It then uses a portlet RequestDispatcher to dispatch(include) the nonFaces target[4.3].

All other requests are assumed by the GenericFacesPortlet to be Faces requests and are executed by calling the bridge. To facilitate navigations from nonFaces views to Faces views, the GenericFacesPortlet recognizes the request parameters _jsfBridgeViewId and _jsfBridgeViewPath. Prior to executing the bridge, the GenericFacesPortlet must check for both the _jsfBridgeViewId request parameter and the _jsfBridgeViewPath request parameter. If either the _jsfBridgeViewId parameter exists or both parameters exist and the _jsfBridgeViewId parameter value is non null, the GenericFacesPortlet must set the value of the request attribute javax.portlet.faces.viewId to the value of the _jsfBridgeViewId parameter[4.4]. If only the _jsfBridgeViewPath parameter exists and contains a non null value, the GenericFacesPortlet must set the value of the request attribute javax.portlet.faces.viewPath to the value of the _jsfBridgeViewPath parameter[4.5]. Otherwise the bridge is called without either of these attributes being set[4.6].

4.2.6 getBridgeClassName()

The bridge calls this method during init(). getBridgeClassName returns the name of the class the GenericFacesPortlet uses to instantiate the bridge. The default (GenericFacesPortlet) implementation returns the value of the javax.portlet.faces.BridgeClassName PortletContext initialization parameter, if it exists. If it doesn't exist, it calls getResourceAsStream() using the current thread's context class loader passing the resource path "META-INF/services/javax.portlet.faces.Bridge". It returns the first line of this stream excluding leading and trailing white space[4.7]. If it can not resolve a class name, it throws a PortletException.

As noted 4.1, bridge implementations are expected to include this resource file in their implementation jar, hence the default behavior of the `GenericFacesPortlet` is to "discover" the appropriate bridge implementation rather then relying on a specific web application configuration settings.

4.2.7 getDefaultViewIdMap()

Unlike direct web access, in the portlet environment target `viewIds` aren't directly reflected in the request `URL`. The bridge is responsible for mapping an incoming request to the correct target. In situations where the incoming request doesn't contain specific bridge encoded target information it must map to a default. This default is provided by the portlet in a `Map` containing an entry of defaults, one per supported `PortletMode`. During its `init()` processing, the `GenericFacesPortlet` called `getDefaultViewIdMap()` to acquire this map which it then sets at a `PortletContext` attribute 3.2 for the bridge's use. The default (`GenericFacesPortlet`) implementation of `getDefaultViewIdMap()` reads each portlet initialization parameter prefixed named `javax.portlet.faces.defaultViewId.[mode]` where *mode* is the `String` form of a supported `PortletMode`. For each entry it adds a Map entry with mode as the key value and the initialization parameter value as the map value[4.8].

For the bridge to work properly one entry per supported mode must be provided in the `Map`. If in later use, the bridge can't find a needed entry, it throws the `BridgeDefaultViewNotSpecifiedException`.

4.2.8 getExcludedRequestAttributes()

As a portlet lifecycle allows multiple (re)renders to occur following an action, the bridge manages an extended notion of a request scope to ensure that such rerenders produces identical results. Specifically, portlet scoped request attributes are saved/restored by the bridge across such rerenders 5.1.2. However, sometimes a portlet request scoped attribute truly must be removed when the request scope ends. The bridge uses multiple mechanisms for determining which attributes are marked for exclusion from its managed scope. The portlet can directly instruct the bridge to exclude attributes on a per portlet basis by setting a PortletContext attribute 3.2. This attribute's value is a `List` containing the excluded attribute names.

The `GenericFacesPortlet` sets this attribute based on the result of calling `getExcludedRequestAttributes()` in its `init()` method. The default (`GenericFacesPortlet`) implementation for `getExcludedRequestAttributes()` returns a `List` constructed by parsing the comma delimited `String` value from the corresponding portlet initialization parameter, `javax.portlet.faces.excludedRequestAttributes`[4
If this initialization parameter isn't present `null` is returned which causes

the `GenericFacesPortlet` to not set the corresponding `PortletContext` attribute[4.10].

4.2.9 isPreserveActionParameters()

By default the bridge doesn't preserve action parameters into subsequent renders. This can be overridden on a per portlet basis by passing a value of true in the appropriate `PortletContext` attribute 3.2. To determine the setting for this attributes for this particular portlet, the `GenericFacesPortlet` calls is`PreserveActionParameters()` in its `init()` method. The default (`GenericFacesPortlet`) implementation returns the `boolean` value corresponding to the `String` value represented in the portlet initialization parameter, `javax.portlet.faces.preserveActionParams`[4.11]. If this initialization parameter doesn't exist, `false` is returned[4.12].

4.2.10 getResponseContentType()

This is a deprecated method that is no longer called or used by the GenericFacesPortlet. It returns the portlet container's indication of the preferred content type for this response[4.13].

4.2.11 getResponseCharacterSetEncoding()

This is a deprecated method that is no longer called or used by the GenericFacesPortlet. It returns null[4.14].

4.2.12 getBridgeEventHandler()

Because portlet events contain arbitrary (typed) payloads, event processing is delegated by the bridge back to the portlet application. The configured `BridgeEventHandler` is called by the bridge at the appropriate point in the lifecycle to allow the application to update the model state from information in the event. See 5.2.5 for more details.

The specific `BridgeEventHandler` is conveyed to the bridge via a `PortletContext` attribute 3.2. The `GenericFacesPortlet` gets the instance it sets for this attribute by calling `getBridgeEventHandler`. If not overridden, the `GenericFacesPortlet`'s default behavior is to read the portlet initialization parameter `javax.portlet.faces.bridgeEventHandler` and return an instance of the class that corresponds to its value[4.15]. If this initialization parameter doesn't exist, null is returned[4.16].

4.2.13 getBridgePublicRenderParameterHandler()

The bridge gives the portlet an opportunity to recompute and resynchronize its models after it has pushed new public render parameter values into any corresponding mapped managed beans by calling the `BridgePublicRenderParameterHandler` conveyed to it by the portlet via a `PortletContext` attribute 3.2. See 5.3 for more details. The `GenericFacesPortlet` gets the instance it sets for this attribute by calling `getBridgePublicRenderParameterHandler`. If not overridden, the `GenericFacesPortlet`'s default behavior is to read the portlet initialization parameter `javax.portlet.faces.bridgePublicRenderParameterHandler` and return an instance of the class that corresponds to its value[4.17]. If this initialization parameter doesn't exist, `null` is returned[4.18].

4.2.14 isAutoDispatchEvents()

In general, the bridge is designed to defer to normal portlet processing to ensure it works well in a mixed use environment. For example, action, render, and resource requests execute through the complete (`GenericPortlet`) portlet call sequence before the `GenericFacesPortlet` dispatches the request to the bridge. This allows none bridge related requests to be handled directly by the portlet. Unfortunately, the portlet event model doesn't lend itself as well to this delegation model. The portlet event model (via the `GenericPortlet`) encourages the use of method annotation to mark the portlet's event handlers. The `GenericPortlet`'s `ProcessEvent()` method provides this behavior. Given that the model also doesn't support marking an event as handled, the `GenericFacesPortlet` needs fore knowledge on dispatching events to the bridge. The `autoDispatchEvents` boolean provides this information. If true, the `GenericFacesPortlet` handles all events by dispatching them directly to the bridge. If false, the `GenericFacesPortlet` doesn't handle any events directly, instead it delegates to the standard portlet model which is then responsible for dispatching JSF related events directly via the bridge. If not overridden, the `GenericFacesPortlet`'s default behavior for this method is to read the portlet initialization parameter `javax.portlet.faces.autoDispatchEvents`. If it exists the value is interpreted as a boolean valued `String` (i.e. "true" is `true` while all other values are `false`)[4.19]. If this initialization parameter doesn't exist, `true` is returned[4.20].

4.2.15 getFacesBridge()

The `GenericFacesPortlet` is designed to be self-servicing. However, there may be times, in particular when dealing with events that aren't auto-dispatched, in which a subclass needs to dispatch a request directly to the bridge. To support this the `GenericFacesPortlet`, via this method, returns a properly initialized

and active bridge which a subclass can use to call one of the `doFacesRequest()` methods[4.21]. The method is passed two parameters, the `PortletRequest` and `PortletResponse`. The `GenericFacesPortlet` uses these to fully prepare the bridge for a `doFacesRequest()` call.

4.2.16 getDefaultRenderKitId()

The `RenderKit` Faces uses commonly resolves to the default set by the application. It is therefore application wide. The default Id is resolved first by looking for a well known request parameter. If this parameter doesn't exist then the default Id is taken from a configuration setting in the `faces-config.xml`. Finally, if no specific default Id is configured an internal default is used. To allow a developer to differentiate the renderkits utilized by individual portlets in an application, the `GenericFacesPortlet` calls this method during `init()` to acquire the specific default renderkit Id it should configure for this portlet. If not overridden, the `GenericFacesPortlet`'s default behavior is to return the value of the portlet initialization parameter `javax.portlet.faces.defaultRenderKitId`[4.22]. If this initialization parameter doesn't exist, `null` is returned[4.23].