Summary of TCK Tests related to Specification Assertions

[NT] Not Testable

Unable to define a TCK test to verify the assertion in a manner that is implementation independent. I.e. the test won't work in every portlet container/app server environment. Note, in some circumstances the test is provided so it can be used to verify behavior/compliance in environments that do provide the necessary support.

Chapter 3 Tests

[3.1] For this specification, the returned name must be "Portlet 1.0 Bridge for JavaServer Faces 1.2"

Test: Write a portlet that gets Bridge API class and calls Class.getPackage to get the Specification Title and Version to compare with above statement.

[3.2] The value `ALWAYS_DELEGATE` indicates the bridge should not render the view itself but rather always delegate the rendering.

Test: Run app with this config setting. Test by writing a ApplicationFactory to insert our own Application Impl that overrides setViewhandler which creates/installs the test ViewHandler before all others (on first call). In this text ViewHandlers render – check to see what the renderpolicy is and set an appropriate msg indicator on the request (attribute)

[3.3] The value `NEVER_DELEGATE` indicates the bridge should always render the view itself and never delegate.

Test: Run app with this config setting. Test by writing a ApplicationFactory to insert our own Application Impl that overrides setViewhandler which creates/installs the test ViewHandler before all others (on first call). In this text ViewHandlers render – check to see what the renderpolicy is and set an appropriate msg indicator on the request (attribute)

[3.4] `DEFAULT` indicates …. the bridge to first delegate the render and if and only if an Exception is thrown then render the view based on its own logic

Test: Run app with this config setting. Test by writing a ApplicationFactory to insert our own Application Impl that overrides setViewhandler which creates/installs the test ViewHandler before all others (on first call). In this text ViewHandlers render – check to see what the renderpolicy is and set an appropriate msg indicator on the request (attribute)

[3.5] If the configuration parameter is not present or has an invalid value the bridge renders using default behavior. I.e. as if `DEFAULT` is set

Test: Run app with this config setting. Test by writing a ApplicationFactory to insert our own Application Impl that overrides setViewhandler which creates/installs the test ViewHandler before all others (on first call). In this text ViewHandlers render – check to see what the renderpolicy is and set an appropriate msg indicator on the request (attribute)

[3.6] `LIFECYCLE_ID` is a `String` valued configuration parameter that describes the `ID` of the `Lifecycle` the bridge uses when executing Faces requests.

Test: Supply test FacesContextFactory. Create a new Lifecycle impl and register with its own ID. aka via faces-config.xml. Set init-param in web.xml to use. Verify in FacesContextFactory we are using this new Lifecycle – test by getting the Lifecycle by ID and verifying that its the same instance as the one passed in.

[3.7] If not set the bridge uses the ID for the default lifecycle (`LifecycleFactory.DEFAULT_LIFECYCLE`).

Test: Supply test FacesContextFactory. Don't configure any specific Lifecycle to be used. Verify in FacesContxtFactory that we are using the default one.

[3.8] `excludedRequestAttributes` is an attribute whose value is a `List` of `String` objects each of which either defines a specific attribute name the bridge is to exclude from its managed request scope or defines a set of attributes the bridge is to exclude from its managed request scope.

Test: covered by 6.1.3.1: requestMapRequestScopeTest – which tests that exclusion works.

[3.9] Such wildcard usage indicates the bridge is to exclude all those attributes within the namespace identified by removing the "*"

Test: covered by 6.1.3.1: requestMapRequestScopeTest – which tests that exclusion works.

[3.10] `preserveActionParams` is a `Boolean` valued attribute that when `TRUE` indicates the bridge must maintain the action's request parameters for the duration of the bridge request scope.

Test: covered by 6.1.3.1: getRequestParameterPreserveParamsTest, getRequestParameterNamesPreserveParamsTest, getRequestParameterValuesMapPreserveParamsTest.

[3.11] The exception to this is the `ResponseStateManager.VIEW_STATE_PARAM` parameter which is always maintained in the bridge request scope regardless of this setting. (I.e. always preserve the VIEW_STATE_PARAM regardless of preserveActionParams setting).

Test: covered by 6.1.3.1:getRequestParameterMapCoreTest, getRequestParameterNamesCoreTest, getRequestParameterValuesMapCoreTest.

[3.12] `defaultViewIdMap` is a `Map<String, String>` valued attribute containing one entry per supported `PortletMode`. The `Map` key is the `String` name of the `PortletMode`. The Map value is the default Faces `viewId` for the mode.

Test: basically covered by any test. However add a test that supports a non-View mode (Edit) with a different viewId mapping. Then write a portlet that has an initial (non-JSF) render that adds a render link back to the portlet putting it in Edit mode. Verify our view is invoked.

[3.13] Once destroy() has been called, no further requests can be processed through this bridge until a subsequent init() has occurred.

Test: subclass GFP, overriding the render behavior – get the Bridge from GFP and then in a render call, init Bridge, destroy it, then call doFacesRequest(render) verify it fails as expected (i.e. throws UninitializedException). Do same but in processAction (i.e. destroy, followed by a doFacesRender), processEvent, and serveResource.

[3.14] This call (destroy()) performs no action if the bridge is in an uninitialized state.

Test: Using test from 3.13 have it call destroy() a second time and verify that no exceptions are thrown.

[3.15] `javax.portlet.faces.viewId`: The value of this attribute identifies the Faces `viewId` the bridge must use for this request (e.g. `/myFacesPage.jsp`). This is expected to be a valid Faces viewId though it may optionally contain a query string

Test: Write a portlet that has a default viewId of X.jsp but in its render sets the above attribute to y.jsp. Verify that y.jsp is activated. I.e. the X.jsp reports an error for the test while Y.jsp reports success.

[3.16] `javax.portlet.faces.viewPath`: The value of this attribute contains the the Faces `viewId` the bridge must use for this request in `ContextPath` relative path form (e.g. `/faces/myFacesPage.jsp`). This value may optionally contain a query string.

Test: Same test as 3.15 but set a path relative viewId.

[3.17] The `BridgeUninitializedException` is thrown if this method (doFacesRequest) is called while the bridge is in an uninitialized state

Test: covered by 3.13.

[3.18] The `NullPointerException` is thrown (from doFacesRequest()) if either the passed request or response objects are `null`.

Test: call both action and render forms of doFacesRequest passing null. (Get bridge from GFP to do this).

[3.19] The `NullPointerException` is thrown (from doFacesRequest()) if either the passed request or response objects are `null`.

Test: call both action and render forms of doFacesRequest passing null. (Get bridge from GFP to do this).

[3.20] `bridgeEventHandler` is a (instanceof) `BridgeEventHandler` valued attribute identifying the instance that is called to handle portlet events received by the bridge

Test: Covered by #5.56 – Provide an eventHandler, trigger an event (in the test's action handler), have the eventHandler set a request attribute indicating its been called, have the test render check this bit.

[3.21] `bridgePublicRenderParameterHandler` is a (instanceof) `BridgePublicRenderParameterHandler` valued attribute identifying the instance that is called to postprocess incoming public render parameter changes after the bridge has pushed these values into the mapped managed beans. This handler gives the portlet an opportunity to recompute and get into a new consistent state after such changes

Test: Covered by #5.71 – Provide an PRPHandler, trigger a PRP change during an action (test's handler), have the PRPhandler set a request attribute indicating its been called, have the test render check this bit.

[3.22] … the renderkit id the bridge should encode as a request parameter in each request to ensure that Faces will use this Id when resolving the renderkit used by this portlet

Test: set the attribute (via configuration). In the test (render) have it test for existence of the request parameter that would indicate its use.

Chapter 4 Tests

[4.1] In addition the `GenericFacesPortlet` reads the following portlet initialization parameters and either sets the appropriate context attributes.

Test: In the JSF Test read the portlet context init params that correspond to the one the GFP should be setting for each of these and make sure they have the same value as the ones in portlet.xml

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

Test: Subclass GFP and override init, call super(), override each of the methods its supposed to call – verify they were called, verify appropriate context params set.

[4.3] 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.

Test: Subclass GFP and overide render (to set the testname). Run test as a multirequest test. On action set the navigation (face-config.xml) to a viewId that is a nonFaces view (has the appropriate QS parameter.

[4.4] 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`.

Test: Set the navigation rule to navigate to a response page but have in its query string both of the above (also pointing to the success page). Verify in the test that the attribute is set.

[4.5] 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.

Test: Set the navigation rule to navigate to a response page but have in its query string only the viewPath (also pointing to the success page). Verify in the test that the attribute is set.

[4.6] Otherwise the bridge is called without either of these attributes being set.

Test: Set the navigation rule to navigate to a response page. Verify in the test that the attributes aren't set.

[4.7] 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. If it can not resolve a class name, it throws a `PortletException`.

Test: override getBridgeClassName, have this method implement behavior as specified – test that result is same as what super() returned.

[4.8] 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.

Test: override method, have this method get values as specified – test that result is same as what super() returned.

[4.9] 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`.

Test: override method, have this method get values as specified – test that result is same as what super() returned.

[4.10] If this initialization parameter isn't present `null` is returned which causes the `GenericFacesPortlet` to not set the corresponding `PortletContext` attribute.

Test: override method, have this method get values as specified – test that result is same as what super() returned. – in this case don't configure in portlet.xml so verify null.

[4.11] The default (`GenericFacesPortlet`) implementation returns the `boolean` value corresponding to the `String` value represented in the portlet initialization parameter, `javax.portlet.faces.preserveActionParams`.

Test: override method, have this method get values as specified – test that result is same as what super() returned.

[4.12] If this initialization parameter doesn't exist, `false` is returned.

Test: override method, have this method get values as specified – test that result is same as what super() returned.

[4.13] It returns the portlet container's indication of the preferred content type for this response.

Test: override method, have this method get preeferred content type – test that result is same as what super() returned.

[4.14] It returns `null`.

Test: override method – test thatt super() returns null..

[4.15] read the portlet initialization parameter `javax.portlet.faces.bridgeEventHandler` and return an instance of the class that corresponds to its value.

Test: override method, have this method get values as specified – test that result is same as what super() returned.

[4.16] If this initialization parameter doesn't exist, `null` is returned.

Test: override method, test that result is from super() is null.

[4.17] read the portlet initialization parameter `javax.portlet.faces.bridgePublicRenderParameterHand` and return an instance of the class that corresponds to its value.

Test: override method, have this method get values as specified – test that result is same as what super() returned.

[4.18] If this initialization parameter doesn't exist, `null` is returned.

Test: override method, test that result is from super() is `null`.

[4.19] If it exists the value is interpreted as a `boolean` valued `String` (i.e. "true" is `true` while all other values are `false`).

Test: override method, have this method get values as specified – test that result is same as what super() returned.

[4.20] If this initialization parameter doesn't exist, `true` is returned.

Test: override method, test that result is from super() is `true`

[4.21] 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.

Test: override method, have this method get value as specified – then call doFacesRequest() and see that all works right.

[4.22] return the value of the portlet initialization parameter `javax.portlet.faces.defaultRenderKitId`.

Test: override method, have this method get values as specified – test that result is same as what super() returned.

[4.23] If this initialization parameter doesn't exist, `null` is returned.

Test: override method, test that result is from super() is `null`.

Chapter 5 Tests

[5.1] This additional associated data includes (state stored in bridge request scope):

Test: During action handling – create all attributes/messages listed (including the excluded one's). On render verify what should be there is there and what shouldn't isn't. Test needs to render same view as runs in the action.

[5.2] For each portlet action ignore any references to an existing request scope. I.e. portlet action requests always define the start of a new scope so never restore a preexisting one.

Test: During an action – add a request attribute, in a second action test and see that its not there.

[5.3] Upon completion of each portlet action preserve the state described above in a newly created bridge request scope if the action doesn't terminate because of a redirect.

Test: During an action – add a request attribute, have the action's navigation rule indicate , verify that during the render the attribute isn't there.

[5.4] … the navigational target of the action doesn't specify a portlet mode that differs from the current request.

Test: During an action – add a request attribute, have the action's navigation encode a mode change, verify that during the render the atribute isn't there.

[5.5] For each render request, the bridge identifies the corresponding bridge request scope that pertains to this request and if one exists it passes the state in this scope to the current (container) request scope.

Test: Covered by test 5.1.

[5.6] For any action parameters that have been preserved, the parameter is restored if and only if there isn't an existing parameter of the same name in the incoming request.

Test: set preserveActionParams. Cause an action (with parameters), verify in render the parameters are there.

[5.7] This corresponds to ensuring that `ViewHandler.createView()` is used to establish the viewroot vs. `ViewHandler.restoreView()`.

Test: Action followed by render, followed by render (redisplay but with a new Mode) – verify that out test ViewHandler's createView is called not its restoreView.

[5.8] Under normal conditions, a bridge request scope must be maintained at least until the next action request (a subsequent bridge request scope is established)

Test: Covered by 5.1

[5.9] The bridge must not assume that all render requests following the establishment of a request scope (and prior to a subsequent action) are executed in the same bridge request scope.

Test: Same type of test as 5.7 except add an attribute in the action and verify its not there in the redisplay.

[5.10a-e] exclude attributes based on:

- an annotated bean
- whose attribute name is listed as an excluded attribute using the `<bridge:excluded-attributes>` … in faces-config.xml
- whose attribute name is in the immediate namespace listed as a wildcarded excluded attribute using the `<bridge:excluded-attributes>` … in faces-config.xml
- whose attribute name appears in the `List` object of excluded attributes in the portlet context attribute `javax.portlet.faces.[portlet name].excludedRequestAttributes`

- whose attribute name appears in the immediate namespace listed as a wildcarded excluded attribute in the `List` object of excluded attributes in the portlet context attribute `javax.portlet.faces.[portlet name].excludedRequestAttributes`

Test: In action add request M request attributes – some to test each of the 4 conditions above – others to test the inverse – that other attributes are retained.

[5.11] The JSF 1.2 specification, section 6.1.5 requires that the various `FacesContext` methods that get `Messages` return those Messages in insertion order. The bridge must preserve this requirement while managing such Messages within the bridge request scope.

Test: Covered by 5.1 – the message portion of the test adds 2 messages to the UIView root and verifies they come back in the same order.

[5.12] For each `doFacesRequest` invocation processing a Faces target, the bridge must acquire a `FacesContext` object. The bridge must acquire the `FacesContext` by calling the `FacesContextFactory`.

Test: 5.13 because it runs in the FacesContextFactory verifies this as well.

[5.13] The `FacesContext` is acquired from the factory by passing the corresponding `PortletContext`, `PortletRequest`, `PortletResponse` objects and the `Lifecycle` …

Test: In own FacesContextFactory – verify that the correct types of objects are passed when acquiring the FacesContext.

[5.14] Prior to acquiring the FacesContextFactory the bridge is required to set the following request attribute on the passed request object.

Test: In FacesContextFactory confirm that the attribute is set with the correct value.

[5.15] If the request attribute named `javax.portlet.faces.viewId` is non null then use this value as the target `viewId`

Test: Covered by test 3.15

[5.16] If the request attribute named `javax.portlet.faces.viewPath` is non null then use this value to process the `ContextPath` relative path and extract the target `viewId`

Test: Covered by test 3.16

[5.17] If unable to extract a `viewId` from the path throw `javax.portlet.faces.BridgeInvalidViewPathEx`

Test: Added new arm to the Test portlet built for test 3.15/3.16. This portlet now also passes an invalid path via the attribute and verifies the exception is thrown.

[5.18] If the bridge is processing for the target view in a render request which occurs in the same portlet mode following another render request in which a

redirect occurred and an action request hasn't been processed in the meantime, use the target `viewId` from the prior redirect.

Test: Action navigates to render page which has a render redirect to a page that has a redisplay on it. Verify that the redisplay renders the redirected page.

[5.19] Use this `viewId`, if and only if, the current portlet mode is the same as the portlet mode in which the `viewId` was encoded into the response

Test: Verify that the current ViewId is ignored when redisplay in a new mode: Use test that includes a redisplay link (to the currentView but with a new mode). Define in portlet.xml that the new mode (edit) default view is a different view. Verify that when the redisplay/new mode occurs you are on the default view page not the original view page.

[5.20] If the bridge can't locate a default `viewId` for the current portlet mode or that default is `null`, throw `javax.portlet.faces.BridgeDefaultViewNotSpecifiedException`

Test: Use a view that includes a redisplay link to the currentView but in a different mode. Don't define a default viewId for the new mode (but do define that the mode is supported). Write a portlet that catches the expected exceptionto verify the result.

[5.21] When a query string is encountered the bridge must extract this information as part of its process of determining the `viewId` and expose the parameters in this query string as additional request parameters.

Test: 2 Tests: One where the default viewId contains the QS param and a second where the portlet sets the ViewId (on an attribute) with a QS param.

[5.22] throw the `BridgeUninitializedException` if the bridge isn't currently initialized (`init()` has been called following either the bridge's construction or a prior `release()`) (action)

Test: Covered by the action destroy portion of test 3.13

[5.23] set the `javax.portlet.faces.phase` request attribute to `Bridge.PortletPhase.ACTION_PHASE` prior to acquiring the `FacesContext`

Test: Covered by test 5.14

[5.24] recognize whether Faces considers the current request processing complete (redirect occurred) or not. If its not considered complete … (set up action response correctly)

Test: Covered by any of our action based scope tests. Test 5.1

[5.25] release the `FacesContext` (at end of action)

Test: In test `FacesContext` set a request attribute on `release()` – test in portlet that the attribute is there.

[5.26] remove the `javax.portlet.faces.phase` request attribute (at end of action)

Test: In portlet that the attribute isn't there after action processing finishes.

[5.27] throw the `BridgeUninitializedException` if the bridge isn't currently initialized (`init()` has been called following either the bridge's construction or a prior `release()`) (render)

Test: Covered by the action destroy portion of test 3.13

[5.28] set the `javax.portlet.faces.phase` request attribute to `Bridge.PortletPhase.RENDER_PHASE` prior to acquiring the `FacesContext`.

Test: Covered by test 5.14

[5.29] If `RenderResponse.getContentType()` returns `null` and there is no other indication of desired content type (not defined by this specification; i.e. an implementation specific extension) then the bridge must call `RenderResponse.setContentType()` passing the value returned from `RenderRequest.getResponseContentType()`

Test: Have portlet call doFacesRequest directly without setting the Content-Type – check in the render portion of Test that the response ContentType is the one requested (i.e. it was set by the Bridge)

[5.30] reestablish (if so indicated) the Faces request scope from the corresponding bridge request scope satisfying all requirements listed in section 5.1.2 concerning providing an idempotent rendition based on the viewId and request scope state referenced in the render request.

Test: Covered by Test 5.1

[5.31] ensure the `RenderKit`'s `ResponseStateManager isPostback()` method returns `true` if and only if a bridge request scope has been identified and restored

Test: In render following action – test that `ResponseStateManager isPostback()` returns true.

[5.32] manually restore the view from its cache if the view hasn't yet been saved by Faces

Test: Covered by Test 5.1

[5.33a-b] ensure that all `PhaseListeners` listening on the before and after phase of the `PhaseId.RESTORE_VIEW` are called

Test: Have test implement/add itself as `PhaseListener` – in before/after, test that we are in render phase and if so set attribute that our listener was called. Verify these attributes in test's render. Note: verify that the other action phases aren't called by also listening on them.

[5.34] execute the `render` phase of the Faces lifecycle.

Test: Any of the existing tests verify this in that all ultimately render a jsf view which can't happen unless this phase happens.

[5.35] recognize if a redirect occurs during this render process and handle by discarding any existing output, and rerunning this process based on the new target and its request parameters (if the target has a query string).

Test: Covered by Test 5.18

[5.36] if necessary, update the value of the `VIEW_STATE_PARAM` parameter managed in this bridge request scope.

Test: Covered by RenderRedisplayTest 5.8. I.e. since this test puts an attribute on the request in the action and verifies its still there after the redisplay the VIEW_STATE_PARAM must have been updated (internally) or else the view couldn't have been restored properly. Note: the messages tests also verify this.

[5.37] release the `FacesContext` (render).

Test: Same logic as 5.25 except check in the render phase not during an action.

[5.38] remove the `javax.portlet.faces.phase` request attribute (render).

Test: Same logic as 5.26 except check in the render phase not during an action.

[5.39] If it is not (the same mode), this target `viewId`must be ignored and the mode's default `viewId` used instead

Test: Covered by 5.19 test – i.e. this test changes the mode in the redisplay link – the portlet.xml defines a different view as the default edit mode view – it verifies we end up at the default edit view – hence verifing this assertion.

[5.40] the bridge must not restore the encoded bridge request scope if the portlet mode has changed. I.e. bridge request scopes are maintained on a per mode basis and must only be used when you (re)render in the same mode.

Test: Cover by 5.4

[5.41] the bridge must ensure that the `VIEW_STATE_PARAM` request parameter is not exposed during a render in which the mode has changed. Typically this occurs automatically as a result of not restoring the bridge request scope

Test: From action encodes its navigation rule to change to edit mode. In render, check that the parameter isn't set.

[5.42] This means the following navigation rule causes one to render the `\edit.jspx viewId` in the portlet edit mode.

Test: Covered by many tests. Here is are the baseic ones: 6.11 and 6.19

[5.43] the bridge must maintain a set of session attributes named `javax.portlet.faces.viewIdHistory.[m` where there is one such attribute per supported portlet mode.

Test: Set up portlet with 3 modes and 3 default views. In render verify there are the three associated mode/history attributes in the session each with the default view as its value.

[5.44] The value of this attribute is the last `viewId` that was active for a given mode.

Test: Covered by 5.47

[5.45] If a mode has never been active than this attribute's value must be the same as the that mode's default `viewId`.

Test: Covered by 5.43

[5.46] These extra parameters must be encoded in the `viewId` in a manner that conforms to JSF expectations for valid `viewId`s and which will be processed correctly when passed through the calls `ViewHandler.getActionURL` and the bridge's `ExternalContext.encodeActionURL` such that it results in a valid portlet `actionURL` and/or `actionResponse` which not only targets the associated `viewId` in the associated `PortletMode` but also returns to that view with the same state (both in the bridge's request scope, view state, and any other render parameters) as existed when this view was added to the history list.

Test: Covered by 5.47

[5.47] the bridge must allow, recognize, and evaluate `faces-config.xml` navigation rules where the `<to-view-id>` element is a JSF expression.

Test: Page 1 (view mode) navigates to Page 2 (view mode). Set a render parameter while in page 2. Navigate to page 3 (edit mode). Have a nav rule from page 3 to mathc the syntax in the spec for returning to the prior view in the prior mode. Verify that we go back to page 2 and that the render parameter is still available.

[5.48] For each portlet event, if a bridge request scope exists, restore its state to the current (container) request scope.

Test: In test action phase, set a request scoped attr and then raise an event that the test portlet subscribes to. In the test event handler, test that the request scope is there. Furthermore, test that its still there in the render phase.

[5.49] Upon completion of each portlet event preserve the *scoped data* (described above) into the prexisting scope if it exists or otherwise a newly created bridge request scope unless the event processing issued a redirect.

Test: In the tests event handler – add a request attribute and then return a nav result, have the navigation rule indicate , verify that during the render the attribute isn't there.

[5.50] … the navigational target of the action doesn't specify a portlet mode that differs from the current request.

Test: In the tests event handler – add a request attribute and then return a nav result, have the navigation rule indicate encode a mode change, verify that during the render the atribute isn't there.

[5.51] For each resource request, if a bridge request scope exists, do not restore the scope into the current request rather maintain a reference to the scope so new request scoped attributes can be merged back into the bridge request scope after the lifecycle has run.

Test: Covered by 5.65

[5.52] Upon completion of each resource request, preserve the *scoped data* (described above) into the prexisting scope if it exists or otherwise a newly created bridge request scope.

Test: Covered by 5.66

[5.53] … throw the `BridgeUninitializedException` if the bridge isn't currently initialized (`init()` has been called following either the bridge's construction or a prior `release()`)

Test: covered by test 3.13

[5.54] set the `javax.portlet.faces.phase` request attribute to `Bridge.PortletPhase.EVENT_PHASE` prior to acquiring the `FacesContext`

Test: Check that this attribute is set with correct value in the eventHandler and provide indication by setting a request parameter that the resulting render can read

[5.55] sets the current non-public request render parameters on the response so they are preserved for the next request.

Test: Set a private render parameter in the action, check in the render that the event was received and that this parameter is still there with the original value.

[5.56] if the portlet has registered an event handler, call the event handler passing the `FacesContext` and the `Event` retaining the returned `EventNavigationResult`.

Test: Provide an eventhandler and register it. If its called then we pass.

[5.57] if the `NavigationResult` returned from the event handler is non-null, acquire the applications `NavigationHandler` and call `handleNavigation()` passing the fromAction and outcome from the `NavigationResult`.

Test: Provide an eventHandler that navigates that returns a result that has a nav rule for a new view – if we end up in the new view we pass.

[5.58] set new public render parameters from mapped models whose values have changed.

Test: In the eventHandler set/update the value of a backing bean that is bound to a Public render parameter. Make sure that during render the new value is there.

[5.59] release the `FacesContext`

Test: Add FacesContextImpl that overrides release() and sets an attribute on session. In portlet, delegate to GFP and then test whether this attribute is there. If it is the test passed.

[5.60]remove the `javax.portlet.faces.phase` request attribute.

Test: Provide own portlet, override prcoessEvent, call super, then check that the phase attribute isn't there. If it isn't, it passed.

[5.61] If the bridge is called to process an event and it hasn't been initialized with a `BridgeEventHandler` then the bridge only sets the current non-public request render parameters on the response so they are preserved for the next request and then returns

Test: Set a PRP in an actionHandler, raise the event, but don't register an event handler. In the render verify the PRP is there and has the correct value.

[5.62] throw the `BridgeUninitializedException` if the bridge isn't currently initialized (`init()` has been called following either the bridge's construction or a prior `release()`) (resource)

Test: Covered by the resource destroy portion of test 3.13

[5.63] set the `javax.portlet.faces.phase` request attribute to `Bridge.PortletPhase.RESOURCE_PHASE` prior to acquiring the `FacesContext`.

Test: render a page that has an iFrame referencing the resource.jsp which outputs its result. In the rendering of the result – test that the phase is set.

[5.64] If the request targets a non-Faces resource, acquire a portlet `RequestDispatcher` and use `forward()` to render the resource.

Test: Write a servlet that sets a session attribute based on whether its invoked from within a forward (check for javax.servlet.forward attrs). Access this as a resource. No markup necessary..

[5.65] do not reestablish the Faces request scope from the corresponding bridge request scope.

Test: Page with an action button on it that whose action sets a request attribute, it then navigates to the results page which is a jsp containing an iFrame reference where the target of the iFrame is a JSF resource URL. Check in the rendering of the resource that the attr isn't there.

[5.66] preserve any changes in the current request scope to the bridge request. Reuse the existing bridge request scope if it exists, otherwise create one.

Test: Uses Trinidad for PPR. Uses Trinidad button to issue a PPR. Have a GoLink and test field be triggered by this PPR. In the resourec request (test method) add a request attribute. Also use this request to swap the names of the buttons so the GoLink is now the RunTest trigger. Have the GoLink represent a (redisplay) link back to this page. When it is invoked – I.e. in this render

phase, test that the request attribute still exists – this means it was preserved in the scope at the end of the resource request.

[5.67] release the FacesContext.

Test: Uses iFrame to the test page to issue the resource request. Other than that its pretty much the same logic as 5.25 except check in the resourec phase not during an action.

[5.68] remove the `javax.portlet.faces.phase` request attribute (resourec).

Test: Same logic as 5.26 except check in the resource phase not during an action.

[5.69] processes incoming public render parameters after the `RESTORE_VIEW` phase of the Faces lifecycle has executed and before any other phase runs.

Test: Provide a phase listener and two excluded scope request parameter mapped to a PRP. In before restoreView check that PRP isn't set/updated yet. In all other before phases check that PRP is set/updated.

[5.70] For each public render parameter in the incoming request that is named in a mapping in the `public-parameter-mappings` section of the `faces-config.xml application-extension` section, the bridge calls the set accessor of the associated bean's mapped data member.

Test: Though 5.69 also verifies this as well since requires a set, we write an explicit test that follows the same pattern as 5.69 but checks the values of the models durign render to ensure they are okay. In addition we register the class as a PRP handler and test that the processUpdates is called – this tests 5.71

[5.71] call its `processUpdates()` passing the `FacesContext` Test: Covered by 5.70

[5.72] `ACTION_PHASE` … If a value has changed, the public parameter is set in the response

Test: Covered by both 5.69 and 5.70 as each sets the (transient) model value in the action handler and then tests the PRP repushes this value back to the model in subsequent request(s).

[5.73] `EVENT_PHASE` … If a value has changed, the public parameter is set in the response

Test: Covered by 5.58

[5.74] set new public render parameters from mapped models whose values have changed (action).

Test: TBD In action set a new value in the underlying request scoped managed bean that is mapped to a PRP. Test in render that the new value is received and is updated into managed bean.

Chapter 6 Tests

[6.1] The file `META-INF/services/javax.faces.context.FacesContextFactory` contains the name of the bridge's concrete `FacesContextFactory` implementation class.

Test: Write a portlet that does the same work that the GFP would do – reads this file and verifies the class. Have the portlet output the response directly.

[6.2] The bridge must not assume that the `FacesContext` returned by calling `getFacesContext` is an *instanceof* its `FacesContext` implementation class.

Test: Have the TCK provide its own FacesContextFactory impl wrapping the bridges. Test is that the JSF page can perform and action/render.

[6.3] (dispatch) Dispatch a request to the specified resource to create output for this response. This must be accomplished by calling the `javax.portlet.PortletContext` method `getRequestDispatcher()`, and calling the `include()`.

Test: In the render portion of the test check that the javax.servlet.include.servletPath exists as this indicates an include happend.

[6.4] (encodeActionURL) If the `inputURL` starts with the # character return the `inputURL` unchanged.

Test: In the render portion of the test call encodeActionURL passing an URL that begins with #.

[6.5] (encodeActionURL) If the `inputURL` is an absolute path external to this portlet application return the `inputURL` unchanged

Test: In the render portion of the test call encodeActionURL passing an absolute URL.

[6.6] (encodeActionURL) If the `inputURL` contains the parameter `javax.portlet.faces.DirectLink` and its value is true then return it without removing this parameter.

Test: In the render portion of the test call encodeActionURL passing an URL containing the DirectLink parameter set to true.

[6.7] (encodeActionURL) If the `inputURL` contains the parameter `javax.portlet.faces.DirectLink` and its value is false then remove the `javax.portlet.faces.DirectLink` parameter and its value from the query string and continue processing.

Test: In the render portion of the test call encodeActionURL passing an URL containing the DirectLink parameter set to false.

[6.8] (encodeActionURL) The scheme is followed by either the keyword `render` or `action`. `render` indicates a portlet `renderURL` should be encoded.

Test: Use encodeActionURL to encode portlet:render?params url. Verify that the result is the same as calling toString on createRenderURL()

[6.9] (encodeActionURL) `action` indicates a portlet `actionURL` should be encoded.

Test: Use encodeActionURL to encode portlet:action?params url. Verify that the result is the same as calling toString on createActionURL()

[6.10] (encodeActionURL) If it is a reference to a Faces view the target is the encoded Faces `viewId`.

Test: Write test that submits an action and renders result of test (in render). If the result is rendered we successfully navigated across an action – ergo the viewId has to have been encoded someplace.

[6.11] (encodeActionURL) recognize the query string parameter `javax.portlet.faces.PortletMode` and use the value of this parameter to identify the mode that should be encoded in the generated reference. (During action)

Test: Have test's action target contain QS parameter for switching mode. Test in render that we are in that mode.

[6.12] (encodeActionURL) If the value doesn't identify a valid mode then no encoding action is taken. (During action)

Test: Have test's action target contain QS parameter for switching mode but to a mode that doesn't exist (for this portlet). Test in render that we are still in "view" mode.

[6.13] (encodeActionURL) recognize the query string parameter `javax.portlet.faces.WindowState` and use the value of this parameter to identify the window state that should be encoded in the generated reference (During action)

Test: Have test's action target contain QS parameter for changing WindowState. Test in render that we are in that window state.

[6.14] (encodeActionURL) If the value doesn't identify a valid window state then no encoding action is taken. (During action)

Test: Have test's action target contain QS parameter for switching window state but to a state that doesn't exist (for this portlet). Test in render that we are still in "normal" mode.

[6.15] (encodeActionURL) recognize the query string parameter `javax.portlet.faces.Secure` and use the value of this parameter to identify the security level that should be encoded in the generated reference (During action)

Test: Have test's action target contain security parameter for changing Security. Test in render that we are in that security state.

Note: as not all portlet containers support changing this state, this may not be testable in many environments and hence isn't a required TCK test.

[6.16] (encodeActionURL) A value of `true` or `false` is translated into the `boolean true/false` respectively regardless of case. Any other value is ignored. (During action)

Test: Have test's action target contain invalid security parameter value for changing Security. Test in render that we are in normal state.

Note: as not all portlet containers support changing this state, this may not be testable in many environments and hence isn't a required TCK test.

[6.17] (encodeActionURL) All other query string parameters are added to the `PortletURL` as parameters. (During action)

Test: Have test's action target contain additional QS params. Test in render that we received them back.

[6.18] (encodeActionURL) If it is a reference to a Faces view the target is the encoded Faces `viewId`. (During render)

Test: Basically the same test as #6.10. If we can render a page contining a form (link) and navigate through it then the first render must have properly encoded the target view id.

[6.19] (encodeActionURL) recognize the query string parameter `javax.portlet.faces.PortletMode` and use the value of this parameter to identify the mode that should be encoded in the generated reference. (During render)

Test: Have a TCK viewHandler that in its getActionURL returns an URL with the QS parameter set to switch the mode. When action is submitted, test that the mode is switched.

[6.20] (encodeActionURL) If the value doesn't identify a valid mode then no encoding action is taken. (During render)

Test: Have a TCK viewHandler that in its getActionURL returns an URL with the QS parameter set to an invalid mode. When action is submitted, test that the mode isn't switched.

[6.21] (encodeActionURL) recognize the query string parameter `javax.portlet.faces.WindowState` and use the value of this parameter to identify the window state that should be encoded in the generated reference (During render)

Test: Have a TCK viewHandler that in its getActionURL returns an URL with the QS parameter set to switch the window state. When action is submitted, test that the window state is switched.

[6.22] (encodeActionURL) If the value doesn't identify a valid window state then no encoding action is taken. (During render)

Test: Have a TCK viewHandler that in its getActionURL returns an URL with the QS parameter set to an invalid window state. When action is submitted, test that the window state isn't switched.

[6.23] (encodeActionURL) recognize the query string parameter `javax.portlet.faces.Secure` and use the value of this parameter to identify the security level that should be encoded in the generated reference (During render)

Test: Have a TCK viewHandler that in its getActionURL returns an URL with the QS parameter set to secure (mode). When action is submitted, test that we are in secure mode.

Note: as not all portlet containers support changing this state, this may not be testable in many environments and hence isn't a required TCK test.

[6.24] (encodeActionURL) A value of `true` or `false` is translated into the `boolean true/false` respectively regardless of case. Any other value is ignored. (During render)

Test: Have a TCK viewHandler that in its getActionURL returns an URL with the QS parameter set to an invalid secure mode value. When action is submitted, test that the secure mode isn't switched.

Note: as not all portlet containers support changing this state, this may not be testable in many environments and hence isn't a required TCK test.

[6.25] (encodeActionURL) All other query string parameters are added to the `PortletURL` as parameters. (During render)

Test: Have a TCK viewHandler that in its getActionURL returns an URL with the regulaar parameter set to some known value. When action is submitted, test that the parameter is received and has the known value.

[6.26] (encodeResourceURL) If the `inputURL` is *opaque*, in that it is an absolute `URI` with a scheme-specific part that doesn't begin with a slash character (e.g. `mailto:java-net@java.sun.com`), return the `inputURL` unchanged

Test: During render call encodeResourceURL with an opaqueURL and verify its uncahnged.

[6.27] (encodeResourceURL) check to see if the `inputURL` contains a query string parameter named `javax.portlet.faces.BackLink`. If it does replace it with a parameter whose name is the value of this parameter and whose value is the `String` (`URL` path) returned after calling `ViewHandler.getActionURL()` passing the current `viewId` followed by `ExternalContext.encodeActionURL()`

Test: In render, call encodeResourceURL with the BackLink param – compare it to the value from calling encodeActionURL(getActionURL(currentViewId))

[6.28] (encodeResourceURL) return `getResponse().encodeURL(inputURL)` (foreign resource URL)

Test: During render: pass a foreign resource URl to encodeResourceURL – test that the return is the same as passing the same string to getResponse.encodeURL()

[6.29] (encodeResourceURL) If `inputURL` is a relative `URL` (i.e. it is neither absolute nor starts with a '/') then the `inputURL` must be turned into a context path relative URL by constructing a new url based on going relative to the current path. The current path is defined as the path that would be used to `dispatch()` to the current view. Return `getResponse().encodeURL(inputURL)`

Test: During render: call encodeResourceURL passing a relative URL – test that the return is the same as passing the full path URL to response.encodeResourceURL()

[6.30] (encodeResourceURL) If the `inputURL` is *hierarchical* and targets a resource that is within this application … If the resulting `inputURL` contains a query string parameter named `javax.portlet.faces.BackLink` then replace it with a parameter whose name is the value of this parameter and whose value is the `String` (`URL` path) returned after calling `ViewHandler.getActionURL()` passing the current `viewId` followed by `ExternalContext.encodeActionURL()`.

Test: During render: call encodeResourceURL with an relative URL with a BackLink param – verify that the URL is correctly ouput with a correct backlink

[6.31] (encodeResourceURL) If the `inputURL` is *hierarchical* and targets a resource that is within this application … Ensure that the `inputURL` (potentially modified by the previous step) is a fully qualified path `URI` (i.e.contains the context path)

Test: During render: call encodeResourceURL with a context path relative URL – verify the result is a fully qaulified URI

[6.32] (encodeResourceURL) If the `inputURL` is *hierarchical* and targets a resource that is within this application … If the resulting `inputURL` contains a query string parameter named `javax.portlet.faces.BackLink` then replace it with a parameter whose name is the value of this parameter and whose value is the `String` (`URL` path) returned after calling `ViewHandler.getActionURL()` passing the current `viewId` followed by `ExternalContext.encodeActionURL()`

Test: During render: call encodeResourceURL with a context path relative URL with a backLink param – verify the result is a fully qaulified URI and contains a valid Backlink

[6.33] (encodeResourceURL) If the resulting `inputURL` contains a query string parameter named `javax.portlet.faces.ViewLink` then remove this query string parameter and value from the `inputURL`. If the value of this parameter was "`true`" return the result of calling `encodeActionURL(inputURL)`

Test: During render: call encodeResourceURL with a context path relative URL with a viewLink param – verify the result is same as calling encodeActionURL.

[6.34] (encodeResourceURL) If the resulting `inputURL` contains a query string parameter named `javax.portlet.faces.ViewLink` then remove this query string parameter and value from the `inputURL`. If the value of this parameter was "`true`" return the result of calling `encodeActionURL(inputURL)`

+ If the resulting `inputURL` contains a query string parameter named `javax.portlet.faces.BackLink` then replace it with a parameter whose name is the value of this parameter and whose value is the `String` (`URL` path) returned after calling `ViewHandler.getActionURL()` passing the current `viewId` followed by `ExternalContext.encodeActionURL()`

Test: During render: call encodeResourceURL with a context path relative URL with a viewLink param and backLink param – verify the result is same as calling encodeActionURL and has a correctly encoded backLink

[6.35] (setRequest) This must be the last request object set as a consequence of calling `setRequest()`.

Test: In action set a new request object and then get it to see that get returns the newly set object.

[6.36] (setCharacterSetEncoding) Overrides the name of the character encoding used in the body of this request. Calling this method during the `RENDER_PHASE` has no effect and throws no exceptions

Test: Call during render phase and verify that an Exception isn't thrown.

[6.37] (setCharacterSetEncoding) Overrides the name of the character encoding used in the body of this request.

Test: Call during action and verify its set by calling getCharacterSetEncoding.

[6.38] (getRequestHeaderMap) Return an immutable `Map` whose keys are the set of request header names included in the current request, and whose values (of type `String`) are the first (or only) value for each header name returned by the underlying request. Within a `RENDER_REQUEST`, the map must exclude the `CONTENT-TYPE` property (if it is present in the underlying request). When executing during a `RENDER_PHASE` the bridge must only ensure that `Accept` and `Accept-Language` exist (and as noted above that `Content-Type` doesn't exist).

Test: During Render: call getRequestHeaderMap – verify map is immutable, contains same values as request.getPropertyXXX, plus the Accept and Accept-Language, minus Content-Type.

[6.39] (getRequestHeaderMap) Return an immutable `Map` whose keys are the set of request header names included in the current request, and whose values (of type `String`) are the first (or only) value for each header name returned by the underlying request. In addition, to provide compatibility with servlets, the bridge must ensure that the following entries exist in the `Map` and the bridge is executing during an `ACTION_PHASE`: `Accept`, `Accept-Language`, `Content-Type`, and `Content-Length`

Test: During Action: call getRequestHeaderMap – verify map is immutable, contains same values as request.getPropertyXXX, plus the Accept, Accept-Language, and Content-Type.

[6.40] (getRequestHeaderValuesMap) Return an immutable `Map` whose keys are the set of request header names included in the current request, and whose values (of type `String[]`) are all of the value for each header name returned by the underlying request Within a `RENDER_REQUEST`, the map must exclude the `CONTENT-TYPE` property (if it is present in the underlying request). When executing during a `RENDER_PHASE` the bridge must only ensure that `Accept` and `Accept-Language` exist (and as noted above that `Content-Type` doesn't exist).

Test: During Render: call getRequestHeaderValuesMap – verify map is immutable, contains same values as request.getPropertyXXX, plus the Accept and Accept-Language, minus Content-Type.

[6.41] (getRequestHeaderValuesMap) Return an immutable `Map` whose keys are the set of request header names included in the current request, and whose values (of type `String`) are the first (or only) value for each header name returned by the underlying request. In addition, to provide compatibility with servlets, the bridge must ensure that the following entries exist in the `Map` and the bridge is executing during an `ACTION_PHASE`: `Accept`, `Accept-Language`, `Content-Type`, and `Content-Length`

Test: During Action: call getRequestHeaderValuesMap – verify map is immutable, contains same values as request.getPropertyXXX, plus the Accept, Accept-Language, and Content-Type.

[6.42] (getRequestMap) Return a mutable`Map` representing the request scope attributes for the current application….. operations must take the appropriate action on the underlying data structure

Test: During render: add two request attrs, one via ExternalContext api and other using the protlet api, verify both can be read using both APIs. Verify that remove works by doing the reverse.

[6.43] (getRequestMap) Furthermore these attributes must be managed across portlet requests according to the rules defined in section [5.1.2]

Test: During action add a set of requet attrs that will test all cases of attrs that should be retained in scope and attrs that should not be. During render, verify that only those that should be retained are there.

[6.44] (getRequestMap) If the attribute is included in the bridge request scope then regardless of whether the attribute is a managed-bean or not, if the attribute's value has one or more public no-argument void return methods annotated with `javax.portlet.faces.annotation.BridgePreDestroy`, then each such method must be called before the element is removed from the underlying data structure.

Test: Have a request scoped managed bean that records whether its predestroy methods have been properly called (puts values into request scope). Now remove the bean. Test that the expected request scope attrs indicating predestroy sequence are there.

Note: This test will not work in Portals that rely on servlet cross-context dispatching to execute portlets as the required context listeners aren't called across the contexts.

[6.45] (getRequestParameterMap) Return an immutable `Map` whose keys are the set of request parameters names included in the current request, and whose values (of type `String`) are the first (or only) value for each parameter name returned by the underlying request. In addition, during a portlet's `RENDER_PHASE`, if not otherwise already in the `Map`, the bridge must include those parameters managed in the corresponding bridge request scope. This always includes the `ResponseStateManger.VIEW_STATE_PARAM` parameter

Test: Provide form with two input fields with preset values, on action verify that the input fields/values are there. During render (that follows this action) verify the parameters aren't there but the VIEW_STATE param is.

[6.46] (getRequestParameterMap) The preservation/inclusion of the rest of the action parameters depends on the `javax.portlet.faces.[portlet name].preserveActionParams` portlet context attribute. If this context attribute exists and has a value of `Boolean.TRUE`, the additional action parameters are preserved/included.

Test: Have portlet configure this. Use same test as #6.45 but verify that the form params are also there during render.

[6.47] (getRequestParameterMap) This `Map` must be composed from the set of parameters available via the `javax.portlet.PortletRequest` methods `getParameter()` and `getParameterNames()` plus any additional parameter names encoded in the (query string) of the `viewId`

Test: Have the default viewid of the portlet contain QS params. During render verify that these params are there.

[6.48] (getRequestParameterNames) Return an `Iterator` over the names of all request parameters included in the current request. In addition, during a portlet's `RENDER_PHASE`, if not otherwise already in the `Iterator`, the bridge must include those parameter names managed in the corresponding bridge request scope. This always includes the `ResponseStateManger.VIEW_STATE_PARAM` parameter.

Test: Provide form with two input fields with preset values, on action verify that the input fields/values are there. During render (that follows this action) verify the parameters aren't there but the VIEW_STATE param is.

[6.49] (getRequestParameterNames) The preservation/inclusion of the rest of the action parameters depends on the `javax.portlet.faces.[portlet name].preserveActionParams` portlet context attribute. If this context attribute exists and has a value of `Boolean.TRUE`, the additional action parameters are preserved/included.

Test: Have portlet configure this. Use same test as #6.49 but verify that the form params are also there during render.

[6.50] (getRequestParameterNames) This `Iterator` must be composed from the set of parameters available via the `javax.portlet.PortletRequest` methods `getParameter()` and `getParameterNames()` plus any additional parameter names encoded in the (query string) of the `viewId`

Test: Have the default viewid of the portlet contain QS params. During render verify that these params are there.

[6.51] (getRequestParameterValuesMap) Return an immutable `Map` whose keys are the set of request parameters names included in the current request, and whose values (of type `String[]`) are all of the values for each parameter name returned by the underlying request In addition, during a portlet's `RENDER_PHASE`, if not otherwise already in the `Map`, the bridge must include those parameters managed in the corresponding bridge request scope. This always includes the `ResponseStateManger.VIEW_STATE_PARAM` parameter

Test: Provide form with two input fields with preset values, on action verify that the input fields/values are there. During render (that follows this action) verify the parameters aren't there but the VIEW_STATE param is.

[6.52] (getRequestParameterValuesMap) The preservation/inclusion of the rest of the action parameters depends on the `javax.portlet.faces.[portlet name].preserveActionParams` portlet context attribute. If this context attribute exists and has a value of `Boolean.TRUE`, the additional action parameters are preserved/included.

Test: Have portlet configure this. Use same test as #6.51 but verify that the form params are also there during render.

[6.53] (getRequestParameterValuesMap) This `Map` must be composed from the set of parameters available via the `javax.portlet.PortletRequest` methods `getParameter()` and `getParameterNames()` plus any additional parameter names encoded in the (query string) of the `viewId`

Test: Have the default viewid of the portlet contain QS params. During render verify that these params are there.

[6.54] (getRequestPathInfo) Return the extra path information (if any) included in the request `URI`; otherwise, return `null`. This value must represent the path portion of the current target `viewId`. Because the portlet model doesn't support a (servlet) equivalent notion of `pathInfo` and `servletPath`, the bridge must manufacture these values based on the target `viewId`. The bridge determines the target view from request parameter(s) it has previously encoded. If this information doesn't exist, the target view is the default `viewId` defined by the portlet. The associated pathInfo and `servletPath` are constructed by determining the servlet mapping of the Faces servlet and constructing the appropriate paths such that they conform to the paths the servlet container generates when

processing an http request which targets this view as defined in SRV .4.4 in the Servlet 2.5 specification

Test: TCK tests use extension mapping so verify that this method returns null.

[6.55] (getRequestServletPath) Returns the part of this request's URL that calls the servlet. This path starts with a "/" character and includes either the servlet name or a path to the servlet, but does not include any extra path information or a query string.

Test: Call getrequestServletPath – comapre result with what we know it should be based on the TCK being extension mapped.

[6.56] (getRequestCharacterEncoding) Return the character encoding currently being used to interpret this request. If called during the `RENDER_PHASE` it returns `null`.

Test: During render, call method, test that the return is null.

[6.57] (getRequestCharacterEncoding) Return the character encoding currently being used to interpret this request. If called during the `ACTION_PHASE`, returns the value from the corresponding action `request.getCharacterEncoding()`

Test: During action call method and test that its the same as calling the request method directly.

[6.58] (getRequestContentType) Return the MIME Content-Type for this request. If called during the `RENDER_PHASE` it returns `null`

Test: During render, call method, test that the return is null.

[6.59] (getRequestContentType) Return the MIME Content-Type for this request. If called during the `ACTION_PHASE`, returns the value from the corresponding action `request.getContentType()`.

Test: During action call method and test that its the same as calling the request method directly.

[6.60] (getResponseCharacterEncoding) Returns the name of the character encoding (MIME charset) used for the body sent in this response. If called during the `RENDER_PHASE`, returns the value from the corresponding render `response.getCharacterEncoding()`.

Test: During render call method and test that its the same as calling the response method directly.

[6.61] (getResponseCharacterEncoding) Returns the name of the character encoding (MIME charset) used for the body sent in this response. If called during the `ACTION_PHASE` it throws an `IllegalStateException`.

Test: During action, call method, test that the exception is thrown.

[6.62] (getResponseContentType) Return the MIME Content-Type for this response. If called during the `RENDER_PHASE`, returns the value from the corresponding render `response.getContentType()`

Test: During render call method and test that its the same as calling the response method directly.

[6.63] (getResponseContentType) Return the MIME Content-Type for this response. If called during the `ACTION_PHASE` it throws an `IllegalStateException`

Test: During action, call method, test that the exception is thrown.

[6.64] (redirect) If `redirect` is called during an action request and the target is within this web application and the query string parameter `javax.portlet.faces.DirectLink` either isn't present or has a value of `false` then ensure that the action response is set to cause the subsequent render to target this redirect view.

Test: Do as statement says – in action handler call redirect to new view. In render, verify we are in that view. Also set request attr in action and verify its not there in render to show that scope not saved.

[6.65] (redirect) If `redirect` is called during a render request and the target is a Faces view within this web application then encode the redirect target so the bridge will cease rendering its current view in a manner that ensures that existing output is not returned and instead the rendition from this new target is.

Test: In render when on first view, call redirect to second view, test that in the (subsequent) render we are in this second view.

[6.66] (redirect) If redirect is called during a render request and the target is within this web application but not a Faces view then throw an `IllegalStateException`

Test: In render when on first view, call redirect to non-jsf resource, test that exceptiojn is thrown.

[6.67] (encodeNamespace) Return the specified name, after prefixing it with a namespace that ensures that it will be unique within the context of a particular page. The returned value must be the input value prefixed by the value returned by the `javax.portlet.RenderResponse` method `getNamespace()`

Test: In render test that encodeNamespace() of an empty string returns the same thing as response.getnamespace

[6.68] (getApplicationMap) Return a mutable `Map` representing the application scope attributes for the current application. This must be the set of attributes available via the `javax.portlet.PortletContext` methods `getAttribute()` ….

Test: Get the Map and compare what is in it to what you get by calling the portlet API directly. Verify they are identical.

[6.69] (getAuthType) Return the name of the authentication scheme used to authenticate the current user, if any; otherwise, return `null`

Test: Compare the result of the ExternalContext call to the result from the direct portlet API – verify they are the same.

[6.70] (getContext) Return the application environment object instance for the current application. This must be the current application's `javax.portlet.PortletContext` instance

Test: verify that the returned value is an instance of PortletContext

[6.71] (getInitParameter) Return the value of the specified application initialization parameter (if any). This must be the result of the `javax.portlet.PortletContext` method `getInitParameter(name)`

Test: Compare/verify that this call returns the same result as calling the portlet api directly – get Enum from portlet API – read the first element and ensure you can get with the EC api and it has the same value.

[6.72] (getInitParameterMap) Return an immutable `Map` whose keys are the set of application initialization parameter names configured for this application, and whose values are the corresponding parameter values

Test: Test for immutability by trying to put something into it. Compare each entry in Map to that of the enum from the portlet API and ensure the entries are identical.

[6.73] (getRemoteUser) Return the login name of the user making the current request if any; otherwise, return `null`. This must be the value returned by the `javax.portlet.http.PortletRequest` method `getRemoteUser()`

Test: Ensure the result is identical between this api and the underlying portlet api.

[6.74] (getRequestContextPath) Return the portion of the request `URI` that identifies the web application context for this request. This must be the value returned by the `javax.portlet.PortletRequest` method `getContextPath()`

Test: Ensure the result is identical between this api and the underlying portlet api.

[6.75] (getRequestCookieMap) This must be an empty `Map`

Test: Verify it returns an empty Map

[6.76] (getRequestLocale) Return the preferred `Locale` in which the client will accept content. This must be the value returned by the `javax.portlet.PortletRequest` method `getLocale()`

Test: Ensure the result is identical between this api and the underlying portlet api.

[6.77] (getRequestLocales) Return an `Iterator` over the preferred `Locales` specified in the request, in decreasing order of preference. This must be an `Iterator` over the values returned by the `javax.portlet.PortletRequest` method `getLocales()`

Test: Ensure the result is identical between this api and the underlying portlet api.

[6.78] (getResource) Return a `URL` for the application resource mapped to the specified path, if it exists; otherwise, return `null`. This must be the value returned by the `javax.portlet.PortletContext` method `getResource(path)`

Test: Ensure the result is identical between this api and the underlying portlet api when passed a local resource URL.

[6.79] (getResourceAsStream) Return an `InputStream` for an application resource mapped to the specified path, if it exists; otherwise, return `null`. This must be the value returned by the `javax.portlet.PortletContext` method `getResourceAsStream(path)`

Test: Get a local resource via this api and the underlying portlet api. Read the bytes and compare to ensure we got the same thing.

[6.80] (getResourcePaths) Return the `Set` of resource paths for all application resources whose resource path starts with the specified argument. This must be the value returned by the `javax.portlet.PortletContext` method `getResourcePaths(path)`

Test: Ensure the result is identical between this api and the underlying portlet api.

[6.81] (setResponseCharacterEncoding) Sets the character encoding (MIME charset) of the response being sent to the client, for example, to UTF-8. This method must take no action.

Test: Use API to set character encoding to something other than what is already set, verify by getting the character encoding that its still the old value.

[6.82] (getSession) If the `create` parameter is `true`, create (if necessary) and return a session instance associated with the current request. If the `create` parameter is `false` return any existing session instance associated with the current request, or return `null` if there is no such session. This method must return the result of calling `getPortletSession(create)` on the underlying `javax.portlet.PortletRequest` instance

Test: verify that the object returned from this call is equal to the object returned from the underlying portlet api.

[6.83] (getSessionMap) Return a mutable `Map` representing the session (`PORTLET_SCOPE`) scope attributes for the current portlet

Test: Verify that the Map contains the same entries and values as when you access directly via the underlying portlet api

[6.84] (getSessionMap) If the element to be removed is a managed-bean, and it has one or more public no-argument void return methods annotated with `javax.annotation.PreDestroy`, each such method must be called before the element is removed from the underlying data structure.

Test: Cause a special test session scoped managed bean to be activated (it will push attributes as its predestroy methods are called), remove it , verify the predestroy methods were called.

Note: This test will not work in Portals that rely on servlet cross-context dispatching to execute portlets as the required context listeners aren't called across the contexts.

[6.85] (getUserPrincipal) Return the `Principal` object containing the name of the current authenticated user, if any; otherwise, return `null`. This must be the value returned by the `javax.portlet.http.PortletRequest` method `getUserPrincipal()`

Test: Ensure the result is identical between this api and the underlying portlet api.

[6.86] This type of `UIViewRoot` must be returned unless the bridge delegates `UIViewRoot` creation and the result of that delegation is a `UIViewRoot` whose implementation class (not `instanceof`) is not `javax.faces.component.UIViewRoot`.

Test: In render test the UIViewRoot and make sure its a PortletNamingContainer (as we know that no special UIViewRoots are used in this test app)

[6.87] If the request isn't a portlet request, delegate `renderView` to the parent and return the result without taking any further action. Otherwise ….

Test: Covered by #3.2, #3.3, and #3.4

[6.88] Prior to the dispatch: Add an attribute named `javax.portlet.faces.RenderContentAfterView` with a `java.lang.Boolean` value of `Boolean.TRUE` to the `request` object

Test: In render check for this attribute

[6.89] After the dispatch and after rendering the view: If non-null, copy the content into the response and remove the attribute from the request `Map`

Test: In render write the test result into the request attribute not the output – verify we get the test result. Note: no way to test the removal of the attrbiute.

[6.90] The bridge must prevent the Faces action phases (`ApplyRequestValues`, `ProcessValidations`, `UpdateModel`, and `InvokeApplication`) from executing if rendering in a restored bridge request scope.

Test: Covered by #5.33

[6.91] More specifically, a `UIViewRoot` with the `javax.portlet.faces.annotation.PortletNamingContai` annotation must implement `getContainerClientId()` to return a `String` containing (at least in part) the portlet's namespace `Id`, if and only if, called during a portlet request.

Test: In action and in render get a the UIViewRoot and verify its a PortletNamingContainer. Call its getContainerClientId() and ensureit returns something with the portletnamespace id in it.

[6.92] The namespace `Id` used in processing `getContainerClientId()` must be consistent for the lifetime of the view (across save and restore).

Test: Same test as #6.91 except get the clientId in action, preserve it, then get the namespace in render as well as the clientId and ensure that both clientIds contain this namespace.

[6.93] Because `getContainerClientId()` can be called during any portlet life-cycle phase (action or render).

Test: Covered by #6.91

[6.94] The convenience class `javax.portlet.faces.PortletNamingContainerUIViewRoot` is provided to simplify adding portlet namespacing for Faces extensions (and for internal bridge usage).

Test: Create and instance of this class to ensure it exists.

[6.95] The class is annotated with the `javax.portlet.faces.annotation.PortletNamingContainer` annotation.

Test: Ensure instance created in #6.94 is an instaceof PortletNamingContainer – this is done as part of test #6.94

[6.96] It implements `getContainerClientId()` to meet the above requirements.

Test: Call its getContainerClientId() and verify result. I.e. the id contains the portlet namespace id.

[6.97] Furthermore it must restore this request parameter at the beginning of each `RENDER_PHASE` phase that corresponds to this bridge request scope such that a call to `ExternalContext.getRequestParameterMap().get(ResponseStateManager.VIEW_STATE_P` returns the restored value.

Test: In render following an action, test that this request parameter exists.

[6.98] Finally, when its able to restore this parameter the bridge must also set the request attribute `javax.portlet.faces.isPostback` with a `Boolean` object whose value is `Boolean.TRUE`

Test: In render following an action, test that this attribute is set with a value of true.

[6.99] Note on encoding/xml escaping: because renderkits have their own pre/post processing to deal with situations calling for xml escaping in urls, the bridge must return an url that contains the identical xml escaping (or not) used in the url passed to encodeActionURL. I.e. if the incoming url is xml escaped the the returned url msut also be xml escaped, likewise if the incoming url isn't escaped the returned url must not be escaped. In the case xml escaping can't be determined from the incoming url, the bridge must assume the url is not xml escaped and return an unescaped url accordingly. Also, because there are situations where Faces components will further encode returned url strings by replacing in the url with the '+' which not all portlet containers may be able

to subsequently process, the bridge can (should) url encode the space character (%20) prior to returning the url regardless of any stipulation regarding base encoding.

Test: Single test for containing encodeActionURL and encodeResourceURL tests. In render, call encodeActionURL 3 times: pased url (1) is xml escaped, (2) isn't xml escaped, (3) contains no indication – i.e. no query satring. Test that each url returned contains the corresponding encoding/escaping. Do same for encodeResourceUrl (where the url is to another Faces view).

[6.100] EL resolution within the JSP Context. Section 6.5 defines a large set of requirements of how EL variables are resolved from within the JSP Context.

Test : A single test is provided – a JSP acquires all the various referenced values using JSP EL and then tests that they exist and are correct. It prints the results out at the end of the JSP

[6.101] EL resolution within the Faces Context. Section 6.5 defines a large set of requirements of how EL variables are resolved from within the Faces Context.

Test : A single test is provided – the test acquires all the various referenced values using Faces EL and then tests that they exist and are correct. It prints the results out at the end of the test.

[6.102] `resource` indicates a portlet `resourceURL` should be encoded

Test : Use encodeActionURL to encode portlet:resource?params url. Verify that the result is the same as calling toString on createResourceURL()

[6.103] (encodeActionURL) If it is a reference to a Faces view the target is the encoded Faces `viewId`.

Test: Write test that raises an event in an action, then in the event handler navigates to the and renders result of test (in render). If the result is rendered we successfully navigated across an action – ergo the viewId has to have been encoded someplace.

[6.104] (encodeActionURL) recognize the query string parameter `javax.portlet.faces.PortletMode` and use the value of this parameter to identify the mode that should be encoded in the generated reference. (During event)

Test: Write test that raises an event in an action, then in the event handler navigates to the results view. Have test's event target contain QS parameter for switching mode. Test in render that we are in that mode.

[6.105] (encodeActionURL) If the value doesn't identify a valid mode then no encoding action is taken. (During event)

Test: Write test that raises an event in an action, then in the event handler navigates to the results view. Have test's event target contain QS parameter for switching mode but to a mode that doesn't exist (for this portlet). Test in render that we are still in "view" mode.

[6.106] (encodeActionURL) recognize the query string parameter `javax.portlet.faces.WindowState` and use the value of this parameter to identify the window state that should be encoded in the generated reference (During event)

Test: Write test that raises an event in an action, then in the event handler navigates to the results view. Have test's event target contain QS parameter for changing WindowState. Test in render that we are in that window state.

[6.107] (encodeActionURL) If the value doesn't identify a valid window state then no encoding action is taken. (During event)

Test: Write test that raises an event in an action, then in the event handler navigates to the results view. Have test's event target contain QS parameter for switching window state but to a state that doesn't exist (for this portlet). Test in render that we are still in "normal" mode.

[6.108] (encodeActionURL) recognize the query string parameter `javax.portlet.faces.Secure` and use the value of this parameter to identify the security level that should be encoded in the generated reference (During event)

Test: Write test that raises an event in an action, then in the event handler navigates to the results view. Have test's event target contain security parameter for changing Security. Test in render that we are in that security state.

Note: as not all portlet containers support changing this state, this may not be testable in many environments and hence isn't a required TCK test.

[6.109] (encodeActionURL) A value of `true` or `false` is translated into the `boolean true/false` respectively regardless of case. Any other value is ignored. (During event)

Test: Write test that raises an event in an action, then in the event handler navigates to the results view. Have test's event target contain invalid security parameter value for changing Security. Test in render that we are in normal state.

Note: as not all portlet containers support changing this state, this may not be testable in many environments and hence isn't a required TCK test.

[6.110] (encodeActionURL) All other query string parameters are added to the `PortletURL` as parameters. (During event)

Test: Write test that raises an event in an action, then in the event handler navigates to the results view. Have test's event target contain additional QS params. Test in render that we received them back.

[6.111] (encodeActionURL – during resource) If it is a reference to a Faces view the target is the encoded Faces `viewId`.

Test: No direct way to test from within a resource – so test by ensuring that the actionURL generated during the resource request is the same as the one

used during the equivalent action/render tests. I.e. if the URLs are the same AND the other test passes then by transitive behavior the URL is being properly encoded in the resource request (even though we never actually execut the URL).

[6.112] (encodeActionURL) recognize the query string parameter `javax.portlet.faces.PortletMode` and use the value of this parameter to identify the mode that should be encoded in the generated reference. (During resource)

Test: No direct way to test from within a resource – so test by ensuring that the actionURL generated during the resource request is the same as the one used during the equivalent action/render tests. I.e. if the URLs are the same AND the other test passes then by transitive behavior the URL is being properly encoded in the resource request (even though we never actually execut the URL).

[6.113] (encodeActionURL) If the value doesn't identify a valid mode then no encoding action is taken. (During resource)

Test: No direct way to test from within a resource – so test by ensuring that the actionURL generated during the resource request is the same as the one used during the equivalent action/render tests. I.e. if the URLs are the same AND the other test passes then by transitive behavior the URL is being properly encoded in the resource request (even though we never actually execut the URL).

[6.114] (encodeActionURL) recognize the query string parameter `javax.portlet.faces.WindowState` and use the value of this parameter to identify the window state that should be encoded in the generated reference (During resource)

Test: No direct way to test from within a resource – so test by ensuring that the actionURL generated during the resource request is the same as the one used during the equivalent action/render tests. I.e. if the URLs are the same AND the other test passes then by transitive behavior the URL is being properly encoded in the resource request (even though we never actually execut the URL).

[6.115] (encodeActionURL) If the value doesn't identify a valid window state then no encoding action is taken. (During resource)

Test: No direct way to test from within a resource – so test by ensuring that the actionURL generated during the resource request is the same as the one used during the equivalent action/render tests. I.e. if the URLs are the same AND the other test passes then by transitive behavior the URL is being properly encoded in the resource request (even though we never actually execut the URL).

[6.116] (encodeActionURL) recognize the query string parameter `javax.portlet.faces.Secure` and use the value of this parameter to identify the security level that should be encoded in the generated reference (During resource)

Test: No direct way to test from within a resource – so test by ensuring that the actionURL generated during the resource request is the same as the one used during the equivalent action/render tests. I.e. if the URLs are the same AND the other test passes then by transitive behavior the URL is being properly encoded in the resource request (even though we never actually execut the URL).

Note: as not all portlet containers support changing this state, this may not be testable in many environments and hence isn't a required TCK test.

[6.117] (encodeActionURL) A value of `true` or `false` is translated into the `boolean true/false` respectively regardless of case. Any other value is ignored. (During resource)

Test: No direct way to test from within a resource – so test by ensuring that the actionURL generated during the resource request is the same as the one used during the equivalent action/render tests. I.e. if the URLs are the same AND the other test passes then by transitive behavior the URL is being properly encoded in the resource request (even though we never actually execut the URL).

Note: as not all portlet containers support changing this state, this may not be testable in many environments and hence isn't a required TCK test.

[6.118] (encodeActionURL) All other query string parameters are added to the `PortletURL` as parameters. (During resource)

Test: No direct way to test from within a resource – so test by ensuring that the actionURL generated during the resource request is the same as the one used during the equivalent action/render tests. I.e. if the URLs are the same AND the other test passes then by transitive behavior the URL is being properly encoded in the resource request (even though we never actually execut the URL).

[6.119] (getRequestHeaderMap) Return an immutable `Map` whose keys are the set of request header names included in the current request, and whose values (of type `String`) are the first (or only) value for each header name returned by the underlying request. Within a `EVENT_REQUEST`, the map must exclude the `CONTENT-TYPE` property (if it is present in the underlying request). When executing during a `RENDER_PHASE` the bridge must only ensure that `Accept` and `Accept-Language` exist (and as noted above that `Content-Type` doesn't exist).

Test: During Event: call getRequestHeaderMap – verify map is immutable, contains same values as request.getPropertyXXX, plus the Accept and Accept-Language, minus Content-Type.

[6.120] (getRequestHeaderMap) Return an immutable `Map` whose keys are the set of request header names included in the current request, and whose values (of type `String`) are the first (or only) value for each header name returned by the underlying request. In addition, to provide compatibility with servlets, the bridge must ensure that the following entries exist in the `Map` and the bridge is executing during an `RESOURCE_PHASE`: `Accept`, `Accept-Language`, `Content-Type`, and `Content-Length`

Test: During resource: call getRequestHeaderMap – verify map is immutable, contains same values as request.getPropertyXXX, plus the Accept, Accept-Language, and Content-Type.

[6.121] (getRequestHeaderValuesMap) Return an immutable `Map` whose keys are the set of request header names included in the current request, and whose values (of type `String[]`) are all of the value for each header name returned by the underlying request Within a `EVENT_REQUEST`, the map must exclude the `CONTENT-TYPE` property (if it is present in the underlying request). When executing during a `EVENT_PHASE` the bridge must only ensure that `Accept` and `Accept-Language` exist (and as noted above that `Content-Type` doesn't exist).

Test: During Event: call getRequestHeaderValuesMap – verify map is immutable, contains same values as request.getPropertyXXX, plus the Accept and Accept-Language, minus Content-Type.

[6.122] (getRequestHeaderValuesMap) Return an immutable `Map` whose keys are the set of request header names included in the current request, and whose values (of type `String`) are the first (or only) value for each header name returned by the underlying request. In addition, to provide compatibility with servlets, the bridge must ensure that the following entries exist in the `Map` and the bridge is executing during an `RESOURCE_PHASE`: `Accept`, `Accept-Language`, `Content-Type`, and `Content-Length`

Test: During Resource: call getRequestHeaderValuesMap – verify map is immutable, contains same values as request.getPropertyXXX, plus the Accept, Accept-Language, and Content-Type.

[6.123] (getRequestCharacterEncoding) Return the character encoding currently being used to interpret this request. If called during the `RESOURCE_PHASE`, returns the value from the corresponding resource `request.getCharacterEncoding()`

Test: During resource call method and test that its the same as calling the request method directly.

[6.124] (getRequestCharacterEncoding) Return the character encoding currently being used to interpret this request. If called during the `EVENT_PHASE` it returns `null`.

Test: During event, call method, test that the return is null.

[6.125] (getRequestContentType) Return the MIME Content-Type for this request. If called during the `RESOURCE_PHASE`, returns the value from the corresponding action `request.getContentType()`.

Test: During resource call method and test that its the same as calling the request method directly.

[6.126] (getRequestContentType) Return the MIME Content-Type for this request. If called during the `EVENT_PHASE` it returns `null`

Test: During event, call method, test that the return is null.

[6.127] (getResponseCharacterEncoding) Returns the name of the character encoding (MIME charset) used for the body sent in this response. If called

during the RESOURCE_PHASE, returns the value from the corresponding render response.getCharacterEncoding().

Test: During resource call method and test that its the same as calling the response method directly.

[6.128] (getResponseCharacterEncoding) Returns the name of the character encoding (MIME charset) used for the body sent in this response. If called during the EVENT_PHASE it throws an IllegalStateException.

Test: During event call method, test that the exception is thrown.

[6.129] (getResponseContentType) Return the MIME Content-Type for this response. If called during the RESOURCE_PHASE, returns the value from the corresponding render response.getContentType()

Test: During resource call method and test that its the same as calling the response method directly.

[6.130] (getResponseContentType) Return the MIME Content-Type for this response. If called during the EVENT_PHASE it throws an IllegalStateException

Test: During event, call method, test that the exception is thrown.

[6.131] (redirect) If redirect is called during an event request and the target is within this web application and the query string parameter javax.portlet.faces.DirectLink either isn't present or has a value of false then ensure that the event response is set to cause the subsequent render to target this redirect view.

Test: Do as statement says – in event handler call redirect to new view. In render, verify we are in that view. Also set request attr in action (that raises the event) and verify its not there in render to show that scope not saved.

[6.132] Create an instance of a PortletResponseWrapper object that implements javax.portlet.faces.BridgeWriteBehindResponse and set it in the Faces ExternalContext by calling ExternalContext.setResponse()

Test: In JSP that render the page get the ExternalContext.getResponse and check that it implements BridgeWriteBehindResponse

[6.133] If an instance class is configured, it must be used (render case).

Test: During render, in JSP that renders the page get the ExternalContext.getResponse and check that it is the class that is configured.

[6.134] If an instance class is configured, it must be used (resource case).

Test: During resource, in JSP that renders the page get the ExternalContext.getResponse and check that it is the class that is configured.

[6.135] Specifically, when the bridge is initialized, if the portlet context attribute javax.portlet.faces.<portletName>.defaultRenderKitId is set,

the bridge is responsible for ensuring that in every request the request parameter Map(s) returned from ExternalContext.getRequestParameterMap() and ExternalContext.getRequestParameterValuesMap() and the Iterator returned from the ExternalContext.getRequestParameterNames() contain an entry for ResponseStateManager.RENDER_KIT_ID_PARAM. In the Map(s), the value for this entry must be the value from the underlying request, if it exists, otherwise it must be the value in the javax.portlet.faces.<portletName>.defaultRenderKitId context attribute

Test: Test uses Trinidad – have portlet configure the Trinidad renderkit via the portlet.xml entry. Run and action followed by a render. Ensure that in each request the parameter is present in each of the three items above and when a value is present that it has the correct/configured value.

[6.136-6.152] If executed during the RENDER_PHASE or RESOURCE_PHASE and the target was determined by its url path (not portlet: syntax) and that target is a nonFaces viewId, construct and return a renderURL ….

Test: Reprise of tests 6.118-6.125 except the target is a nonfaces viewId. To accomplish this we need a bean that generates the test url and sets a flag indicating it has done so/rendered. We also provide our own portlet (subclass) that on render checks for the flag – if so this is the first render after the rendering of the test url – so if all things do right its the render of the portlet after the test url has been cliucked/submitted (i.e. spec says this is a renderURL not a portlet actionURL) . Resource tests are the same as the render tests except the testURL is rendered in the jsp that supplies the iFrame target (resource).

Chapter 7 Tests

[7.1] One configures the particular implementation of the RenderResponseWrapper and/or ResourceResponseWrapper the bridge uses as the response object it dispatches to in the bridge's application-extension section of the face-config.xml

Test: Covered by Tests #6.133 and #6.134

Chapter 8 Tests