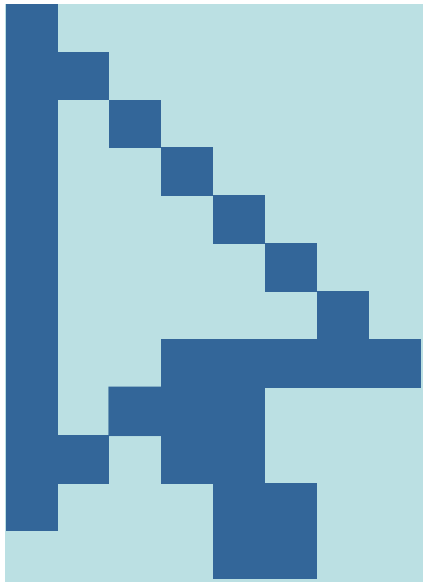
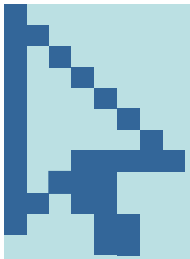


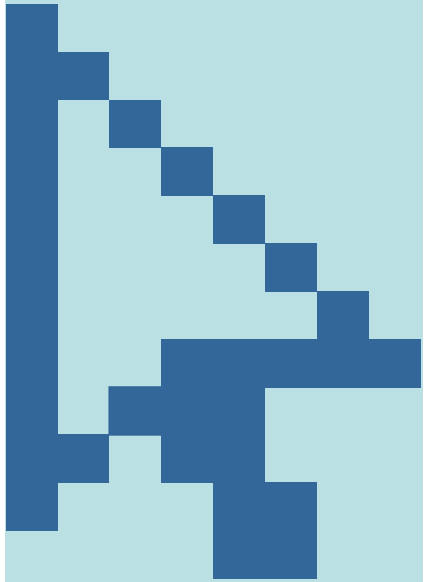
Portlet Standard
JSR 168 / JSR 286

Version 1.0
Martin Weiss

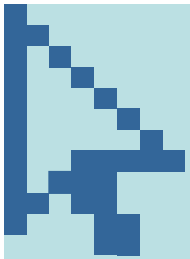
mw-informatik.ch



Agenda

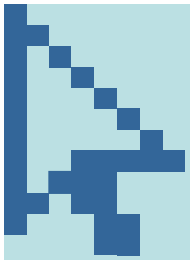


JSR 168



Introduction

- *Different portal server vendors* have defined *different portlet APIs*
- *Incompatible portlet API interfaces* create problems
  - Application and content providers must implement different portlets for different portal servers
  - Portal customers are quickly locked into a particular portal solution
- We need a portlet specification standardization to provide *interoperability between portlets and portals* to overcome these problems

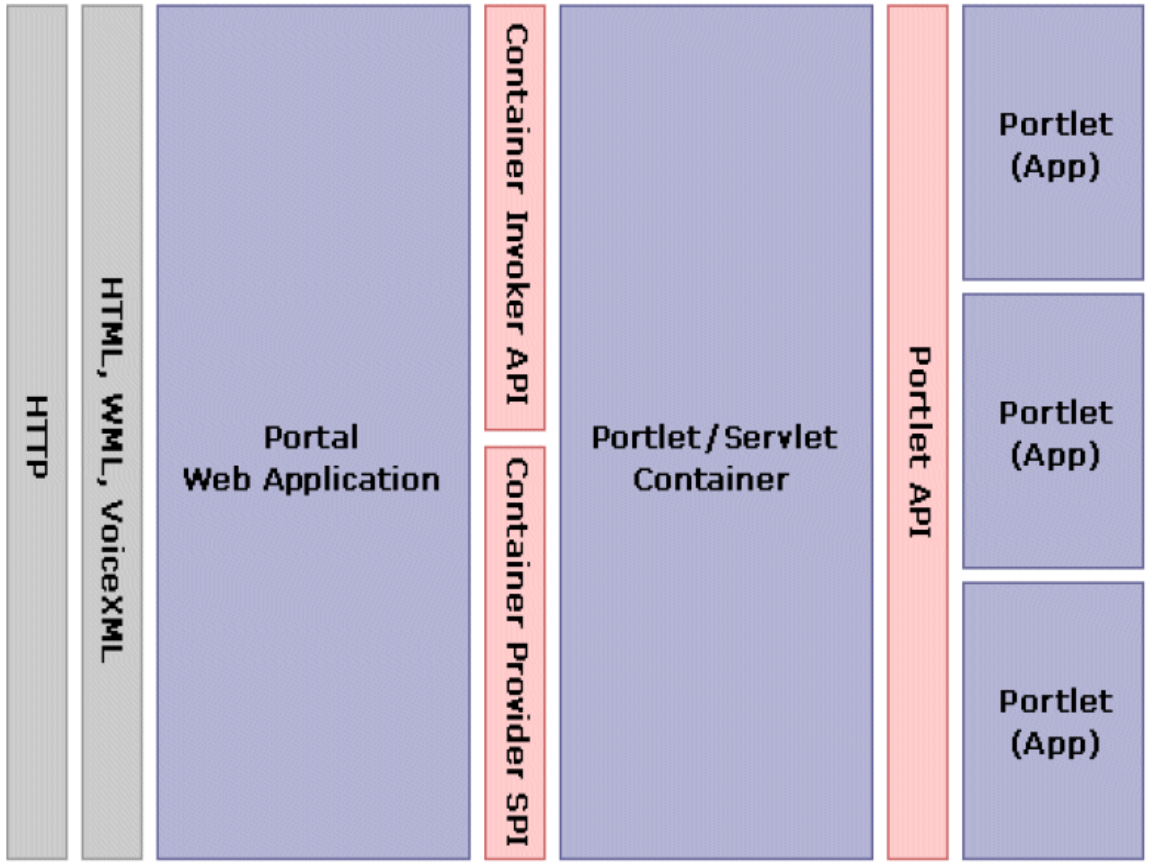


JSR 168 and the Java Community Process

- *Java Standardization Request 168 (JSR 168)* was specified in the Java Community Process (JCP) -> http://jcp.org
  - Expert Group co-led by IBM and Sun
  - JSR 168 v1.0 was approved in *October 2003*
  - Reference implementation of JSR 168 v1.0 provided by IBM, available at Apache -> http://jakarta.apache.org/pluto
  - Compliance test suite available from Sun
  - *Wide market adoption*: supported by many *commercial and open source portals*
    - BEA, IBM, Oracle, Sun, Tibco, Vignette, …
    - Apache, eXo, JBoss, Liferay, uPortal, …
- JSR 168 defines a portlet specification with the following *goals*:
  - Simple programming model
  - Portability
  - Alignment with Servlet Specification v2.3

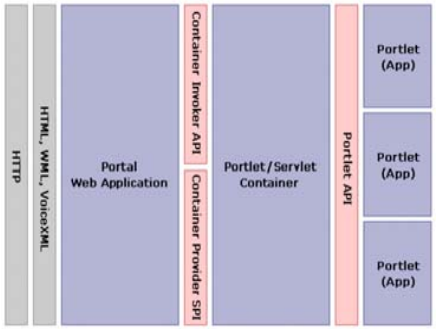
Basic Concepts

- Portal, Portlet Container, Portlets



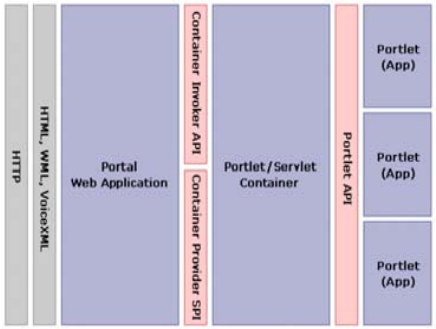
Basic Concepts

- Characteristics of Portal
  - Web application which *aggregates different applications into one page* with a *common look and feel* for the portal user
  - Provides *single sign-on* and sophisticated personalization features which enables *customized content to users*
  - Client request is processed by the Portal Web application, which retrieves the portlets on the current page for the current user. The Portal Web Application, i.e. the *front-end servlet*, then calls the Portlet Container for each portlet to retrieve its content through the *Container Invoker API*



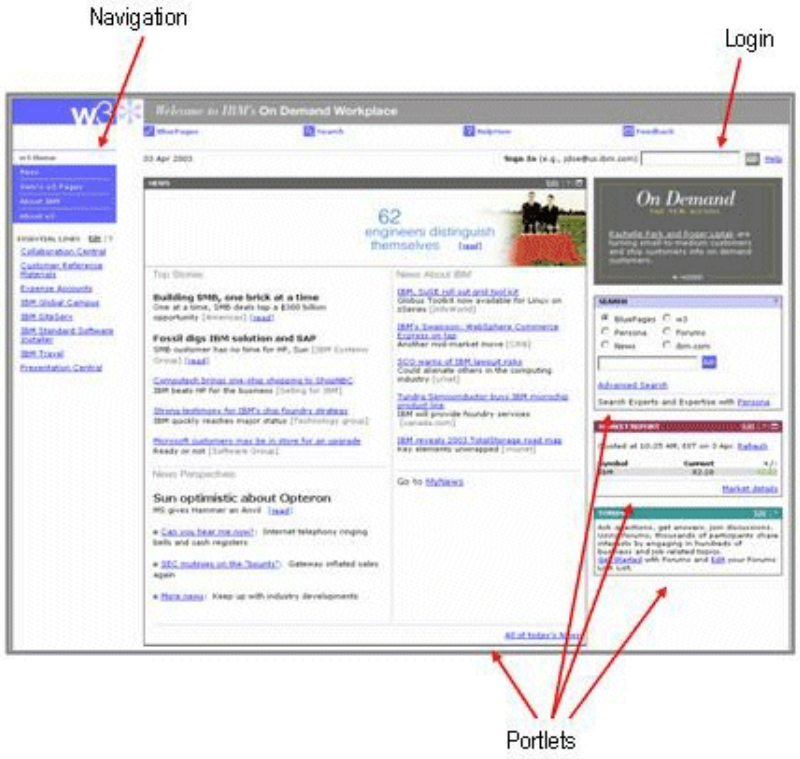
Basic Concepts

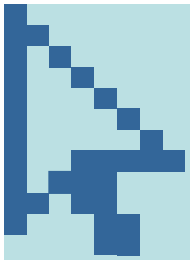
- Characteristics of JSR 168 Portlet Container
  - Extension of the Servlet Container
  - Runs the portlets, provides them with the required runtime environment, and *manages the portlet lifecycle*
  - Provides *persistent storage for portlet preferences* (user specific portlet customization)
  - The Portlet Container calls the portlets through the *Portlet API*
  - The *Container Provider Service Provider Interface (SPI)* enables the Portlet Container to retrieve information from the Portal



Basic Concepts

- Portal page
  - Represents a *complete markup document* and *aggregates several portlet windows* (via Portal Web Application)
  - *Navigation mechanism* to enable the user to navigate to other portal pages





Basic Concepts

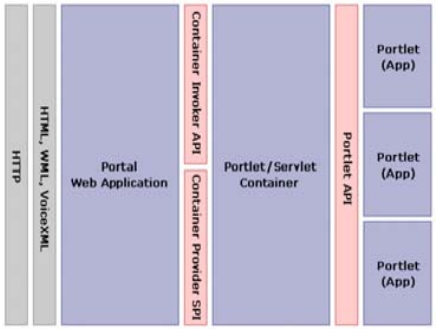
- Characteristics of JSR 168 Portlet API
  - Technical point of view
    - distinct, functional components of a portal
    - Implemented in Java, based on a well-defined portlet API
    - Managed by portal through a portal container
    - Many similarities with the Servlet API (e.g. deployment, class loading, lifecycle)
  - User perspective
    - Functional entities represented on a portal page (Portlet window)

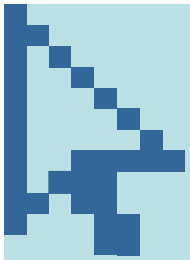
Basic Concepts

- Portlet window
  - Representation of a portlet on a Portal page
  - A portlet window consists of:
    - Title bar, with the title of the portlet
    - Decoration, including buttons to change the *window state* of the portlet (such as maximize or minimize the portlet) and buttons the change the *mode of a portlet* (such as show help or edit the predefined portlet settings)
    - Content produced by the portlet (also called a *markup fragment*)

Scope of JSR 168

- What does JSR 168 define ?
  - Portlet API and Portlet Container
  - Contract between the API and the container
  - Deployment unit: portlet application
- What is out of scope of JSR 168 ?
  - Aggregation, layout management
  - Page personalization and configuration engines
  - Portal administration and configuration



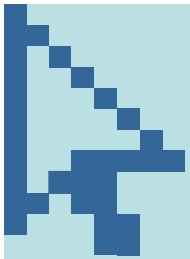


Portlet Modes

JSR 168 defines three categories of portlet modes

- Modes which are required
  - View: Display the portlet output (default mode)
  - Edit: Displays one or more views that let the user *personalize portlet preferences*
  - Help: Display help views

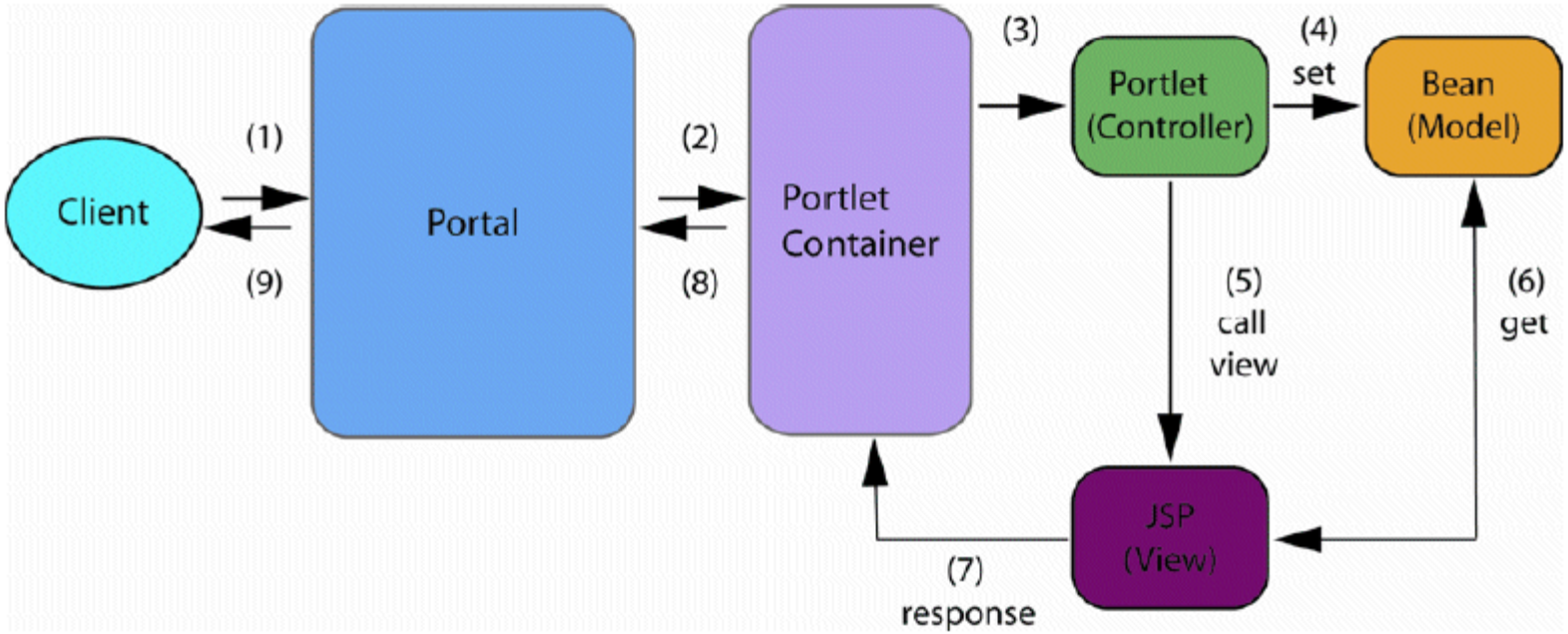
- Optional custom modes
  - About: Display the portlet purpose, origin, version, and other information
  - Config: Display one or more configuration views that let administrators configure portlet preferences which are valid for all users
  - Edit_defaults: Set the default values for the modifiable preferences that are typically changed in the Edit screen
  - Preview: Render output without the need to have back-end connections or user specific data available
  - Print: Display a view which is suitable for printing



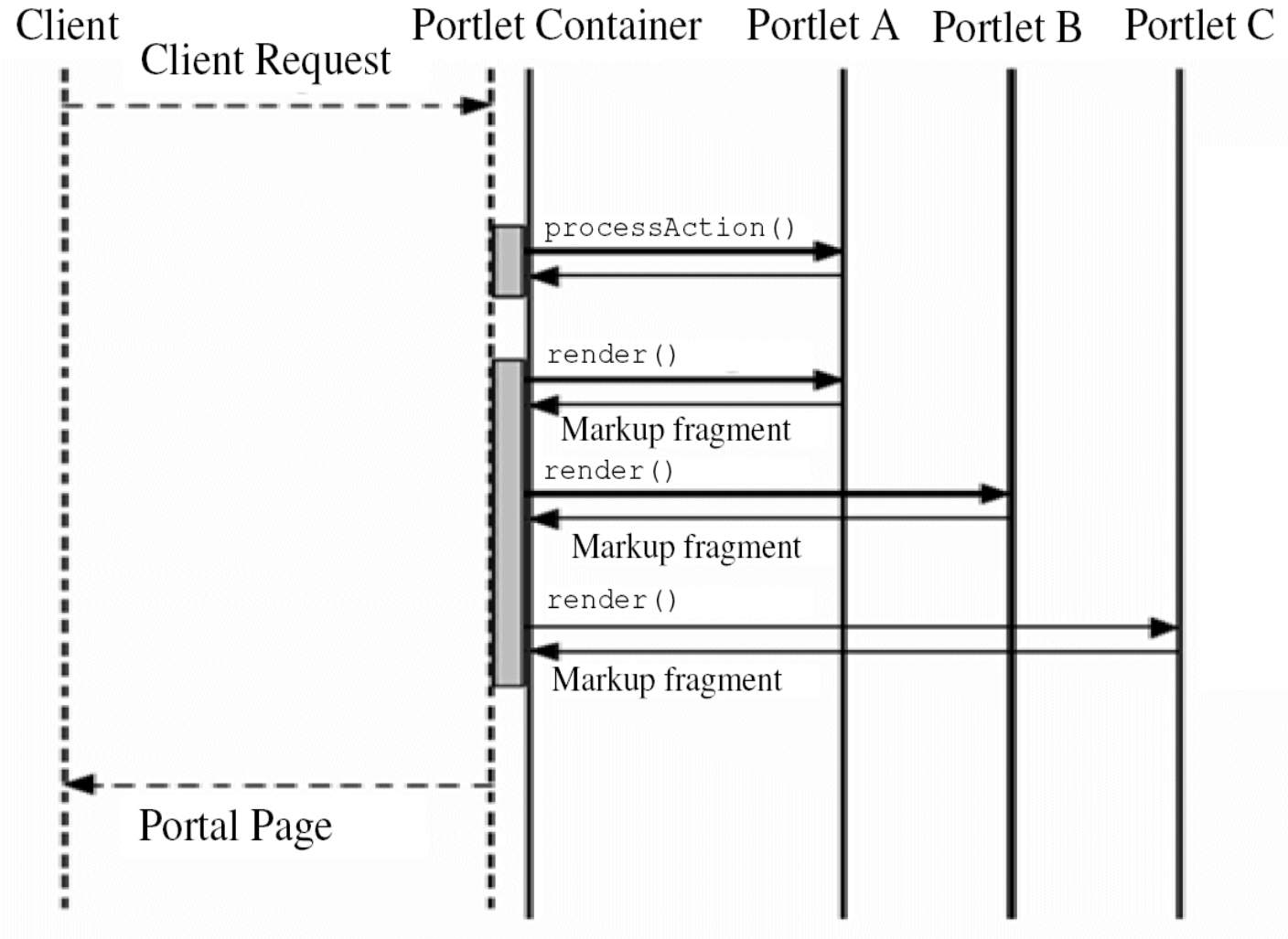
Window States

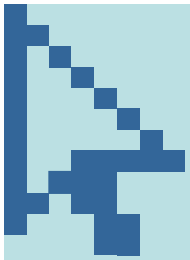
- A window state is an indicator of the amount of portal page space assigned to the content generated by the portlet. The portlet container provides the current window state to the portlet, and the portlet uses the window state to decide how much information it should render. However, portlets can also programmatically change their window state while processing an action event
- JSR 168 defines the following window states:
  - Normal: The portlet shares the space with other portlets and should take this into account when producing its output
  - Maximized: No other portlets are displayed, providing maximum space for the portlet's use
  - Minimized: Only the portlet's title bar is displayed
- In addition the these window states, JSR 168 allows the portal to define custom window states

Portlet MVC architecture



Two Phase Processing in JSR 168

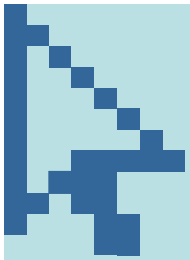




Portlet Class

- The abstract *javax.portlet.GenericPortlet* class provides the default implementation for the *javax.portlet.Portlet* and *javax.portlet.PortletConfig* interface
- User defined portlets should derive from *GenericPortlet* and typically override the following method:
  - processAction, to handle action request
  - doView, to handle render requests when in *VIEW mode*
  - doEdit, to handle render request when in *EDIT mode*
  - doHelp, to handle render request when in *HELP mode*
  - init and destroy, to manage resources that are held for the life of the portlet

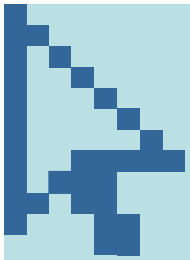
  Normally, there is *no need to override the render method*



Two Types of Portlet Requests / Response

- processAction (ActionRequest, ActionResponse)
  - *Render parameters* are reset
  - Portlet can *set new render parameters via ActionResponse* which are then available in the render request
- render (RenderRequest, RenderResponse)
  - Invoked on every request to the portal to create portlet markup fragments
  - *Portal keeps resending last set of render parameters*

- ***Use render parameters or session to transfer data between the action and render phase***

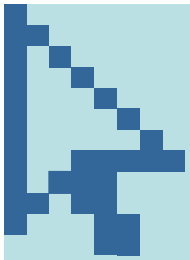


Portlet data model

- JSR 168 defines different mechanisms for the portlet to access *transient* and *persistent* data

- Persistent state
  - *Initialization parameters*, defined in the portlet deployment descriptor (portlet.xml)
  - *Portlet preferences*
    - User-independent (Administrator) preferences; read-only preferences
    - User-dependent preferences

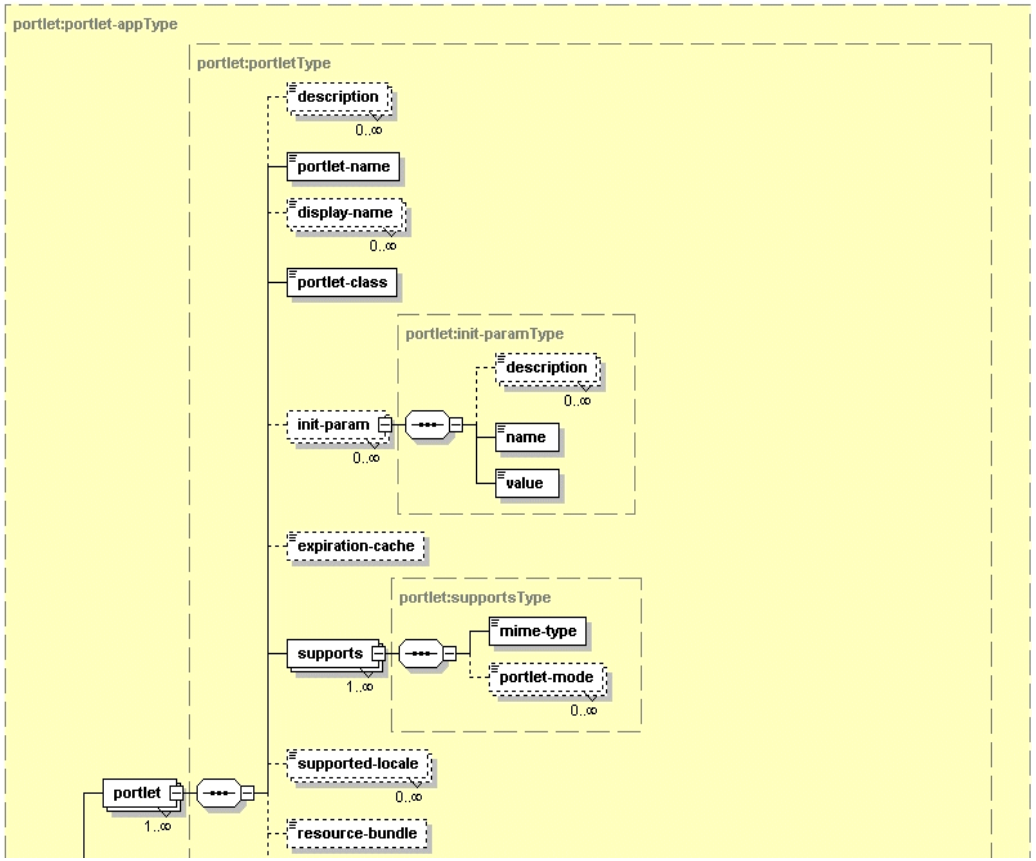
- Transient state
  - *Navigational state* defines how the current view of the portlet is rendered and is specific to the portlet window. Navigational state consist of *portlet mode*, *window state* and *render parameters*
  - *Session*: The portlet can store data in the session with either global scope ("application scope"), to let other components of this Web application access the data, or portlet scope, which is private to the portlet



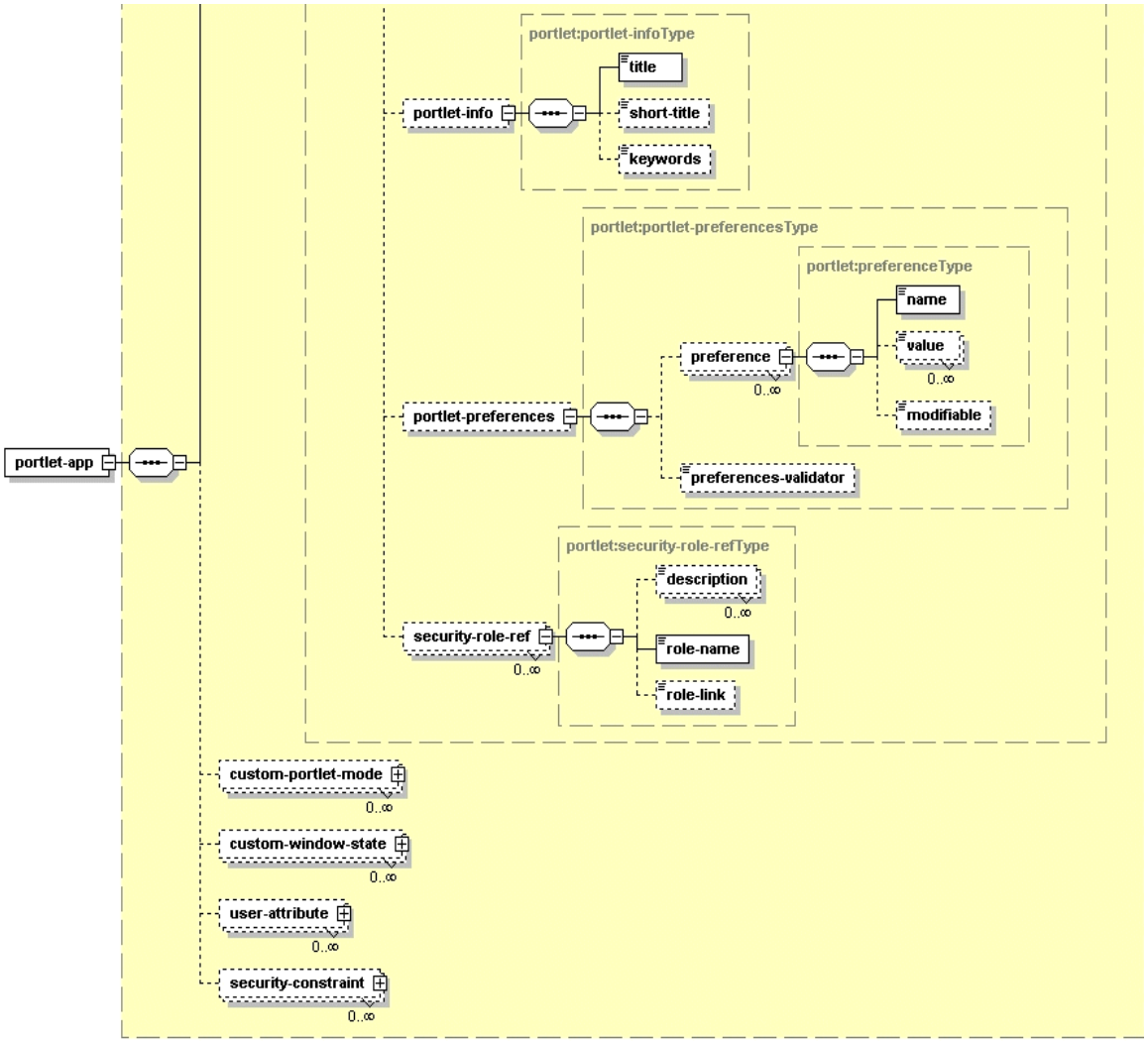
Portlet Packaging and Descriptors

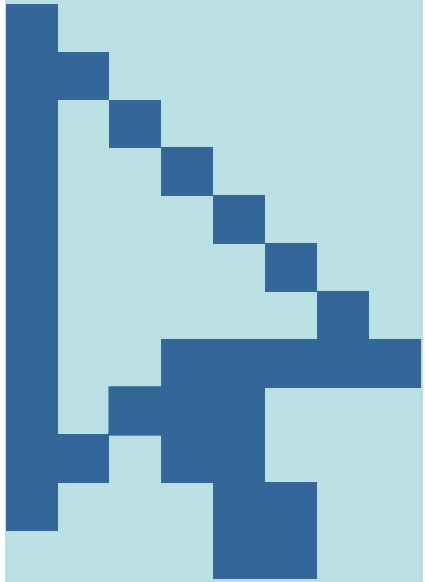
- Portlets are packaged as portlet applications in *WAR files*
- *Portlet deployment description (portlet.xml)* describes the portlet application and its portlets
  - Top element is <portlet-app>; contains <portlet> definitions
- *Web deployment description (web.xml)* describes the web application

Portlet Deployment Descriptor Schema

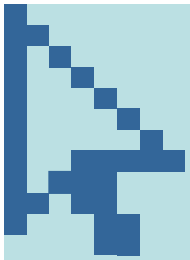


Portlet Deployment Descriptor Schema



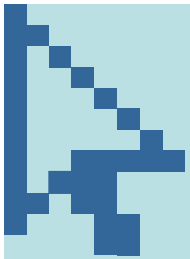


JSR 168 – What Is Missing ?



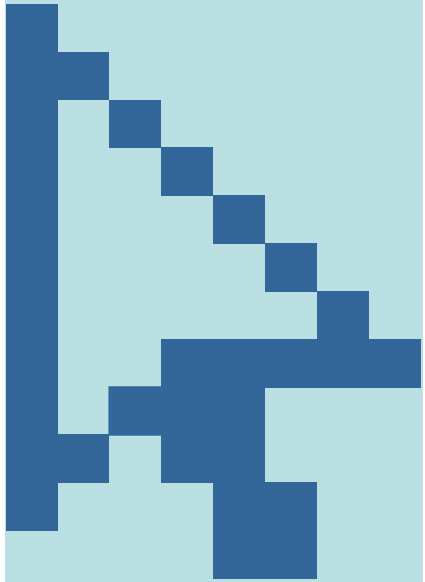
Inter-portlet communication

- Only supported *within the same portlet application* using *session attributes*
- Target portlets will only “see” messages during next render request
- Portlets *should not update their state during a render request*: “event” handling not really possible

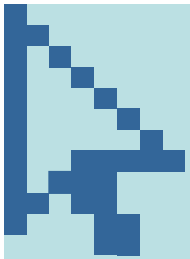


Serving non-html resources

- A portlet can *only render html fragments*
- Have to fallback/delegate to the servlet container
- Requires *coordination between portlet and servlet*

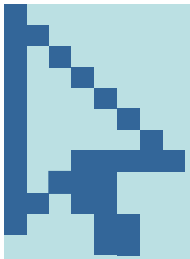


JSR 286



New JSR 286 features

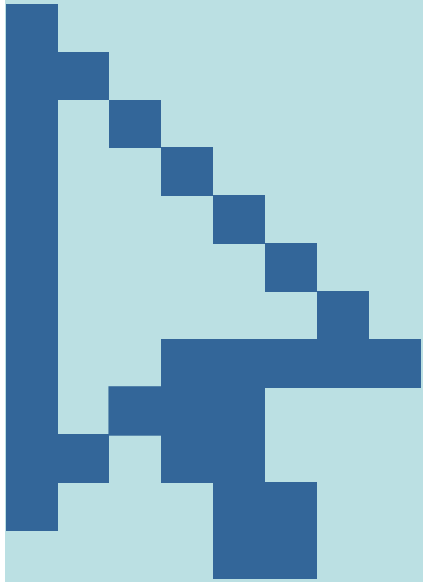
- Portlet coordination
  - Portlet events
  - Public render parameters
- Resource serving
  - Portlet acts as a proxy to a resources, e.g. PDF documents
  - Portlet has full *control over the output stream (no portal aggregation)*
  - Introduction of a new URL type, resource URLs.
    Resource URLs trigger the *serveResource* lifecycle method on the ResourceServingPortlet interface
  - Allows better AJAX support
- Java 5, aligned with J2EE 1.4
  - Java 5 *annotation* support in javax.portlet.GenericPortlet
- Final version submitted in *February 2008*



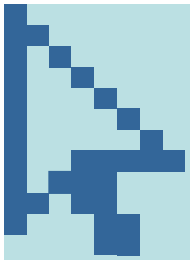
Backward compatibility

- JSR 286 was designed to avoid breaking binary code compatibility with JSR 168 and to maintain compatible behavior for all API methods

- Backward compatibility also includes the deployment descriptor in which JSR 286 added new entries, but did not change existing ones (Migrate JSR 168 portlets by changing the one line in the portlet deployment descriptor that references the schema to the new V2.0 portlet deployment descriptor schema)

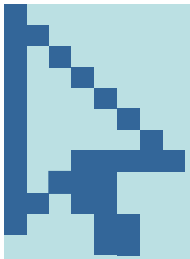


Portlet Events



JSR 286 eventing

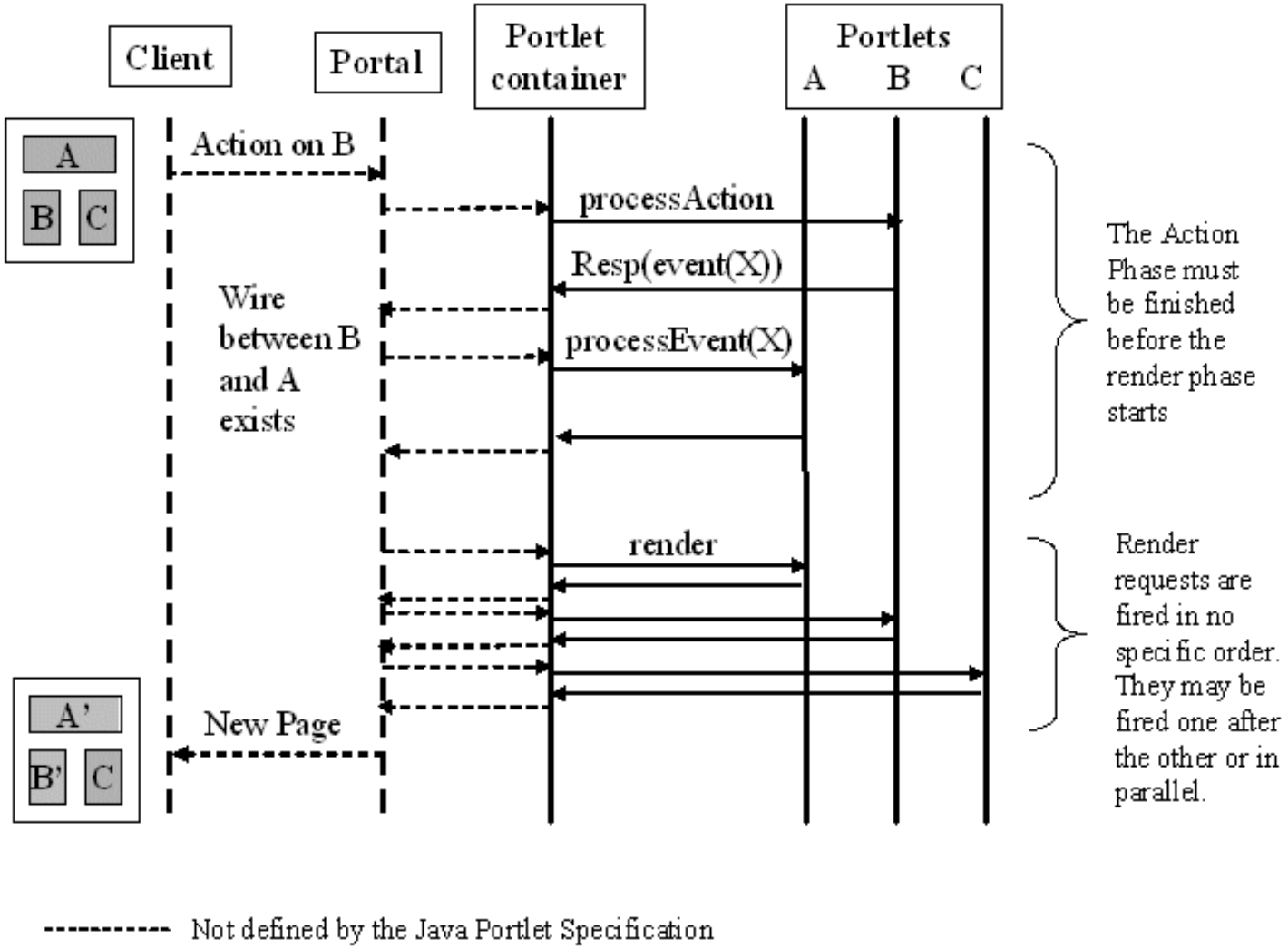
- *Loosely coupled publish/subscribe model*
  - Does not require that the different portlet developers know about each other's work
  - Events are send and received during action phase
- At *development time*, define the date that a portlet understand in the *portlet.xml* and implement the *event specific methods* in the portlet class
- At *deployment time*, create the action connections between the portlets (*not part of the specs*)

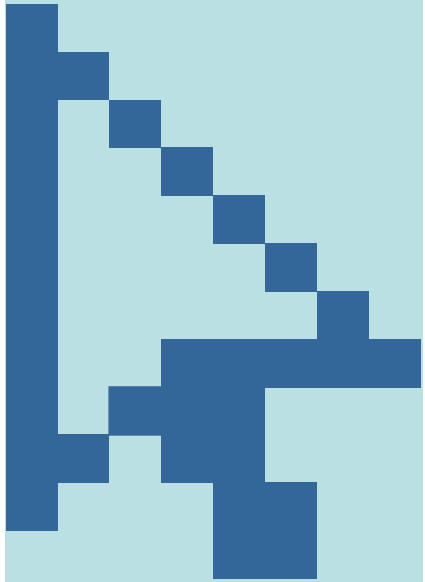


Event declaration

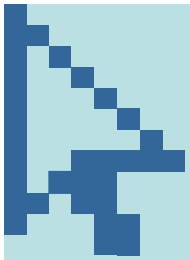
- Portlet events are declared in portlet.xml
  - <event-definition> using Qnames or default namespace
  - A portlet specifies which events it wants to send (<supported-publishing-event>) or receive (<supported-processing-event>) as part of the portlet definition
- Permits portlets to send and receive complex Java objects (payload types)
  - XML serialization mechanism based on JAXB 2.0

Event flow





Public Render Parameters



Public render parameter positioning

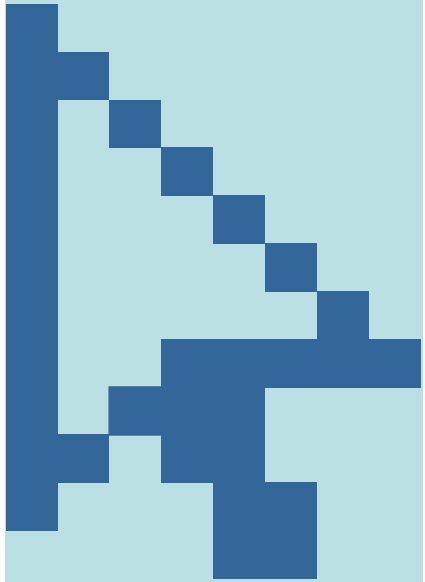
- Besides portlets events, *public render parameter* represents an alternative way for coordination between portlet

  => more *lightweight communication* alternative compared to portlet events

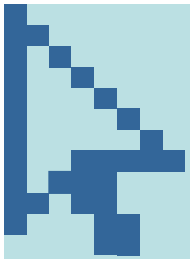
- From a programmer's point of view, *a public render parameter is handled almost identically to an ordinary (private) render parameter*:

  The portlet can set and read this parameter using the same API methods that JSR 168 introduced for private render parameters.

  The important difference is that a *public render parameter is declared in the portlet.xml* deployment descriptor and therefore becomes an external interface of the portlet.

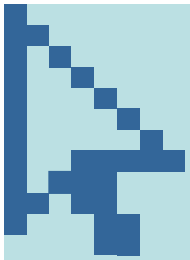


Events vs. Public Render Param: Decison Criteria



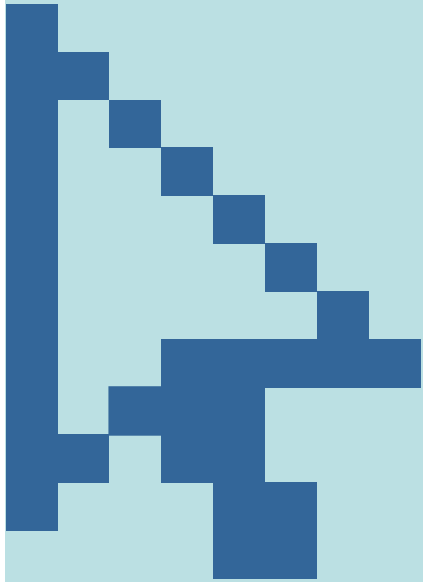
Portlet events features

- They *require explicit portlet code* to send and receive
- They can contain *complex information*
- They allow fine-grained control by setting up different sorts of wires between portlets (on-page or cross-page)
- They can trigger cascaded updates with different information. For example, portlet A can send event X to portlet B, which in turn sends a different event Y to portlet C
- They cause increasing processing overhead as the number of communication links grows
- They must be initiated by some explicit user interaction (normally, by clicking an action link in a portlet), and they cannot be used to set up a coordinated view when first jumping to a page

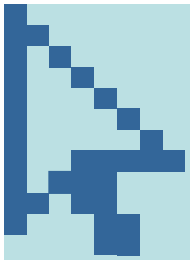


Public render parameter features

- They *do not usually require explicit coding* but only a *declaration in the portlet.xml deployment descriptor*
- They are *limited to simple string values*
- They *do not require explicit administration ("Wiring")* to set up coordination

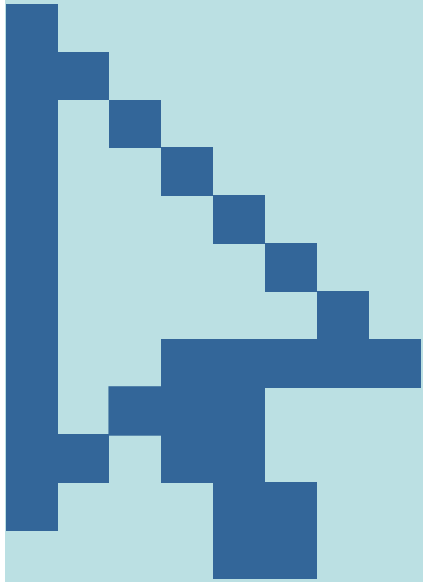


Summary

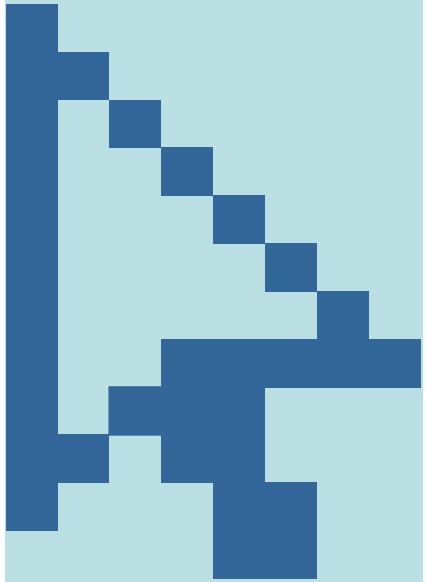


Summary

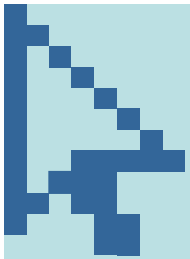
- JSR 168 provides the *overall UI component model*
- JSR 286 focuses on *building integrated composite applications* out of these components
- Portlet technology *similar to Servlet API* (plus extensions)
- Wide market adoptions (commercial and open source portals)



Backup

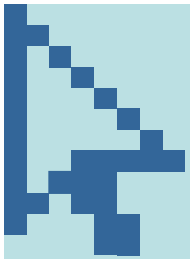


JSR 168



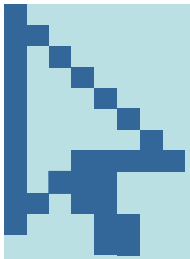
Core interfaces

- javax.portlet.PortletContext
  - The PortletContext interface defines a *portlet view of the portlet container*. Attributes stored in the context are *global for all users and all components in the portlet application*
  - The context provides access to the portlet request dispatcher
  - Interface methods
    - *void setAttribute (String, String)*
    - *void removeAttribute (String)*
    - *String getAttribute (String)*
    - *java.util.Enumeration getAttributeNames ()*
    - *javax.portlet.PortletRequestDispatcher getRequestDispatcher ()*



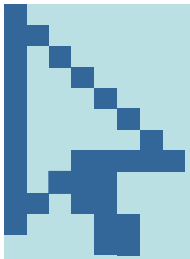
Core interfaces

- javax.portlet.PortletRequestDispatcher
  - The PortletRequestDispatcher interface defines an object that receives requests from the client and sends them to the specified resources (such as a servlet, HTML file, ore JSP file) on the server
  - The portlet container creates the PortletRequestDispatcher object, which is used as a *wrapper around a server resource located at a particular path* or given by a particular name
  - Interface method
    - *void include (RenderRequest, RenderResponse)*



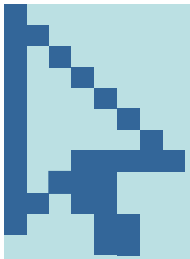
Core interfaces

- javax.portlet.PortletRequest
  - The PortletRequest defines the base interface to provide client request information to a portlet. The portlet container uses two specialized versions of this interface when invoking a portlet, ActionRequest and RenderRequest
  - Interface methods:
    - *String getParameter (String)*
    - *java.util.Enumeration getParameterNames ()*
    - *javax.portlet.PortletSession getPortletSession ()*
    - *javax.portlet.PortletPreferences getPreferences ()*
    - *javax.portlet.PortletMode getPortletMode ()*
    - *javax.portlet.WindowState getWindowState ()*
    - *void setAttribute (String, Object)*
    - *Void removeAttribute (String)*
    - *String getAttribute (String)*



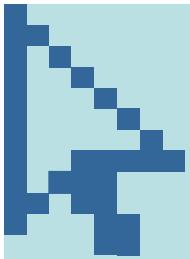
Core interfaces

- javax.portlet.ActionResponse
  - The ActionResponse interface represents the portlet response to an action request. It extends the PortletResponse interface
  - The portlet container creates an ActionResponse object and passes it as argument to the portlet's *processAction* method
  - Interface methods:
    - *void sendRedirect (String)*
    - *void setPortletMode (javax.portlet.PortletMode)*
    - *void setRenderParameter (String, String)*
    - *void setWindowState(javax.portlet.WindowState)*



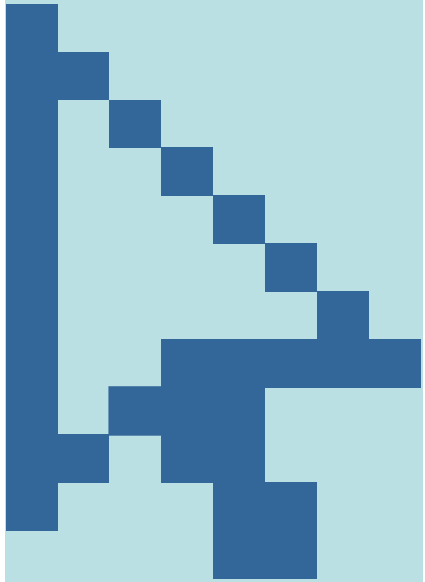
Core interfaces

- javax.portlet.PortletSession
  - The PortletSession interface provides a way to identify a user across more than one request and to store transient information about the user
  - A PortletSession is created *per user client per session scope*
  - The PortletSession interface defines two scopes for storing objects (APPLICATION_SCOPE, PORTLET_SCOPE)
  - Interface methods:
    - *void setAttribute (String, Object)*
    - *void setAttribute (String, Object, int)*
    - *void removeAttribute (String)*
    - *void removeAttribute (String, int)*
    - *Object getAttribute (String)*
    - *Object getAttribute (String, int)*
    - *void invalidate ()*

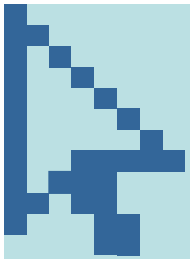


Core interfaces

- javax.portlet.PortletPreferences
  - The PortletPreferences interface allows the portlet to store user specific data
  - There are two different types of preferences:
    - Modifiable preferences – theses preferences can be changed by the portlet in any standard mode (EDIT, HELP, VIEW). Per default every preference is modifiable
    - Read-only preferences – these preferences cannot be changed by the portlet in any standard portlet mode, but may be changed by administrative modes. Preferences are read-only, if they are defined in the deployment descriptor with read-only set to true
  - Interface methods:
    - *String getValue (String, String)*
    - *java.util.Enumeration getNames ()*
    - *void reset (String)*
    - *void setValue (String, String)*
    - *void store ()*

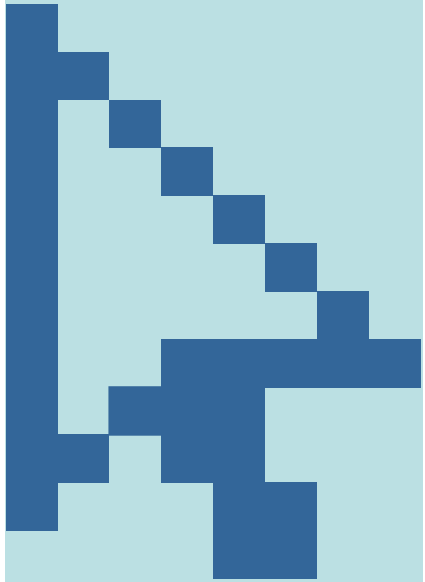


JSR 286

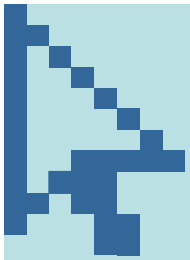


Annotations in javax.portlet.GenericPortlet

- ProcessAction
  - This annotation allows to annotate a method that should process a specific action
- ProcessEvent
  - Allows to annotate methods that should process specific events
- RenderMode
  - Allows to annotate methods that should render a specific portlet mode



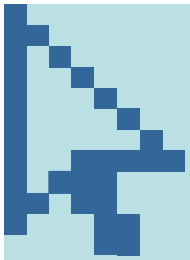
Portlet Events



Sample: portlet.xml

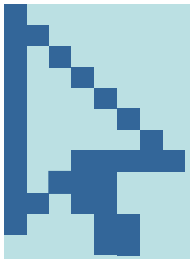
```
<portlet-app>
  <portlet>
    <supported-publishing-event>
      <name>helloworld</name>
    </supported-publishing-event>
  </portlet>
  <portlet>
    <supported-processing-event>
      <qname xmlns:mwi="http://www.mw-informatik.ch/portlet/helloworldConfirmation/ns">
        mwi:helloworldAnswer
      </qname>
    </supported-publishing-event>
  </portlet>

  <default-namespace>http://www.mw-informatik.ch/portlet/helloworld/ns</default-namespace>
  <event-definition>
    <name>helloworld</name>
    <alias>xsd:string</alias>
    <value-type>java.lang.String</value-type>
  </event-definition>
  <event-definition>
    <qname xmlns:mwi="http://www.mw-informatik.ch/portlet/helloworldConfirmation/ns">
      mwi:helloworldAnswer
    </qname>
    <alias>xsd:string</alias>
    <value-type>java.lang.String</value-type>
  </event-definition>
</portlet-app>
```



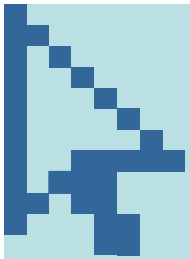
Event programming model

- New core interfaces
  - javax.portlet.Event
  - javax.portlet.EventPortlet
  - javax.portlet.EventRequest
  - javax.portlet.EventRespones
  - javax.portlet.StateAwareResponse



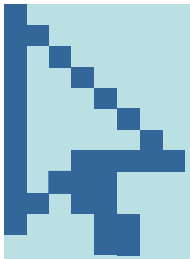
New core interfaces

- javax.portlet.Event
  - Interface methods
    - *String getName ()*
      Get the local part of the event name
    - *javax.xml.namespace.QName getQName ()*
      Get the event QName
    - *java.io.Serializable getValue ()*
      Get the event payload



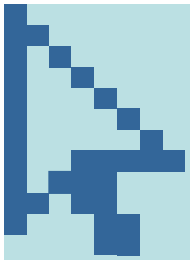
New core interfaces

- javax.portlet.EventPortlet
  - Implementing class: javax.portlet.GenericPortlet
  - Interface methods
    - *void processEvent (javax.portlet.EventRequest, javax.portlet.EventResponse)*
      Method invoked during action phase



New core interfaces

- javax.portlet.EventRequest
  - Superinterface: javax.portlet.PortletRequest
  - The portlet container creates an EventRequest object and passes it as argument to the portlet's *processEvent* method
  - Interface methods
    - *javax.portlet.Event getEvent ()*



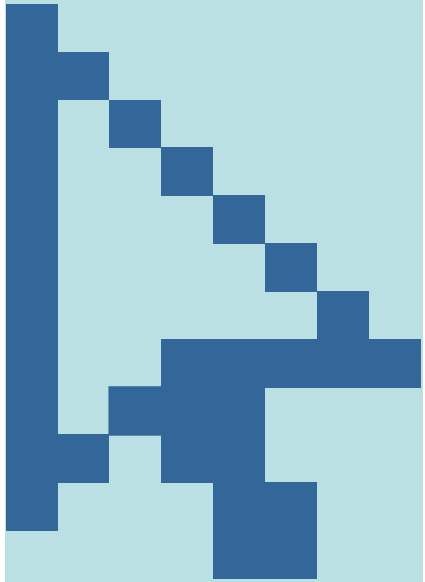
Sample: portlet classes

```
public class HelloworldPortlet extends javax.portlet.GenericPortlet {

   @ProcessAction(name="executeHelloworld")
   public void executeHelloworld(ActionRequest request, ActionResponse response) throws
      PortletException, java.io.IOException {
         String helloworldText = request.getParameter("helloworldText");
         String helloworldAnswer = new Helloworld().answer(helloworldText);
         response.setEvent("helloworld", helloworldAnswer);
         /* response.setEvent(
            new QName("http://www.mw-informatik.ch/portlet/helloworld/ns", "helloworld"),
            helloworldAnswer);  */
}


public class HelloworldConfirmationPortlet extends javax.portlet.GenericPortlet {

   @ProcessEvent(qname="{http://www.mw- informatik.ch/portlet/helloworldConfirmation/ns}
      helloworldAnswer")
   public void processHelloworldAnswer(EventRequest request, EventResponse response)
   throws PortletException, IOException {
      String helloworldAnswer = (String) request.getEvent().getValue();
         response.setRenderParameter("helloworldConfirmation", helloworldAnswer);
   }
}
```



Public Render Parameters

Sample: portlet.xml

```
<portlet-app>

   <portlet>
      <supported-public-render-parameter>helloworldAnswer</supported-public-render-parameter>
   </portlet>

   <public-render-parameter >
      <identifier>helloworldAnswer</identifier>
      <qname xmlns:mwi="http://www.mw-informatik.ch/portlet/helloworld/ns">
         mwi:helloworldAnswer
      </qname>
   </public-render-parameter>

</portlet-app>
```

Sample: portlet classes

```
public class HelloworldPortlet extends javax.portlet.GenericPortlet {

   @ProcessAction(name="executeHelloworld")
   public void executeHelloworld(ActionRequest request, ActionResponse response) throws
      PortletException, java.io.IOException {
         String helloworldText = request.getParameter("helloworldText");
         String helloworldAnswer = new Helloworld().answer(helloworldText);
          response.setRenderParameter("helloworldAnswer", helloworldAnswer);
   }
}


public class HelloworldConfirmationPortlet extends javax.portlet.GenericPortlet {

   @RenderMode(name="view")
   public void showHelloworldConfirmation(RenderRequest request, RenderResponse response)
      throws PortletException, IOException {
         response.setContentType("text/html");
         String helloworldAnswer = request.getParameter("helloworldAnswer");
         request.setAttribute("helloworldConfirmation", helloworldAnswer);
         getPortletContext().getRequestDispatcher(
            "/WEB-INF/jsp/helloworldConfirmationPanel.jsp").include(request, response);
}
```

Contact

Martin Weiss Informatik AG
Untere Roostmatt 8
6300 Zug

Author

Martin Weiss

martin.weiss@mw-informatik.ch