The Java™ Platform
Portlet Specification 2.0 (JSR 286)

Stefan Hepper, Portal Architect, IBM

TS-4817
Learn about the new features of the Java™ Platform Portlet Specification V 2.0 and how you can leverage these new features
Agenda

➢ JSR 286 Overview
➢ New Features in JSR 286
  • Coordination
  • Resource Serving
  • Cookies and Headers
  • Portlet Filter
  • Other additions
➢ Using AJAX with Portlets
➢ Compatibility to V 1.0
➢ Relation to Web Services for Remote Portlets (WSRP)
➢ Summary
Agenda

- JSR 286 Overview
- New Features in JSR 286
  - Coordination
  - Resource Serving
  - Cookies and Headers
  - Portlet Filter
  - Other additions
- Using AJAX with Portlets
- Compatibility to V 1.0
- Relation to Web Services for Remote Portlets (WSRP)
- Summary
Overview Portal Model

Integration-at-the-glass – performed by the portal

Components may or may not work on the same backends
Scope of the Java Technology Portlet Specification

- Portlet API and portlet container
- Contract between the API and the container
- Deployment unit: portlet application
- Not
  - Aggregation, layout management
  - Page personalization and configuration engines
  - Portal administration and configuration
Where do we want to go from V1.0?

V 1.0 (JSR 168)
- Provide the programming model for standalone, pluggable UI application components

V 2.0 (JSR 286)
- Enable coordination between portlets and allow building composite applications based on portlet components
- Serving resources
- Allow for a better user experience using AJAX patterns
- Better support for Web Frameworks like JSF and Struts
- Alignment with Web Service for Remote Portlets (WSRP) 2.0
JSR 286 Expert Group

IBM is leading this JSR, all major Java technology portal (commercial and open source) vendors represented in the EG

Expert Group members

Reference implementation will be provided at Apache
- As Apache Pluto 2.0
- http://portals.apache.org/pluto

TCK will be available for free
Will extend the JSR 168 TCK
## JSR 286 Schedule

<table>
<thead>
<tr>
<th>Step</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kick-off</td>
<td>Feb 06</td>
</tr>
<tr>
<td>Early draft 1</td>
<td>Aug 06</td>
</tr>
<tr>
<td>Early draft 2</td>
<td>April 07</td>
</tr>
<tr>
<td>Public draft</td>
<td>July 07</td>
</tr>
<tr>
<td>Final draft</td>
<td>Dec 07</td>
</tr>
<tr>
<td>Approved</td>
<td>Mar 08</td>
</tr>
</tbody>
</table>

*Size of specification and API more than doubled compared to V 1.0*
Agenda

» JSR 286 Overview
» New Features in JSR 286
  • Coordination
  • Resource Serving
  • Cookies and Headers
  • Portlet Filter
  • Other additions
» Using AJAX with Portlets
» Compatibility to V 1.0
» Relation to Web Services for Remote Portlets (WSRP)
» Summary
Why is coordination so important?

- The #1 complaint about v1.0 was the missing capability to send events between portlets
  - v1.0 only has the portlet application session scope for coordination
  - Only usable within the one portlet application, not across portlet applications
- v2.0 will add additional coordination capabilities
  - Eventing
  - Public render parameters across portlets
- Coordination allows business users building composite applications out of portlet components
  - Can be done at runtime, without programming
Demo Events
Events

- JSR 286 introduces a loosely coupled event paradigm
  - A portlet can declare events it wants to receive and events it wants to emit
  - The portal / portlet container will act as broker and distribute the events
  - Allows wiring of portlets at runtime
  - Dynamic event declaration only for sending events

- Event handling will be an additional step in the overall action phase
  - State changes are allowed
  - Event handling must be finished before rendering starts
Events

Event types
• Are defined in the portlet deployment descriptor via the Java class name
• Can be complex, but must be Java platform and Java Architecture for XML Binding (JAXB) software serializable
• String or XML simple types strongly recommended to reduce coupling
• Use complex types only as last resort

Event names
• Are defined as QNames in the DD
• As full QName (namespace + local part)
  <qname xmlns:x="http://www.ibm.com/">x:emailAddress822</qname>
• Only local part of the QName + default namespace
  <event-definition>
    <name>emailAddress822.AddToAddressBook</name>
  </event-definition>

May have additional aliases for automatic event mapping
Event request flow

The Action Phase must be finished before the render phase starts.

Render requests are fired in no specific order. They may be fired one after the other or in parallel.

Not defined by the Java Portlet Specification
Deployment descriptor sample for sending an event

Event defined in the DD:

```xml
<event-definition>
  <qname xmlns:x="http://examples.com/events">
    x:Address
  </qname>
  <value-type>com.examples.Address</value-type>
</event-definition>

<portlet>
  <supported-publishing-event>
    <qname xmlns:x="http://examples.com/events">
      x:Address
    </qname>
  </supported-publishing-event>
</portlet>
```
Event processing in the portlet

```java
@XmlRootElement
public class Address implements Serializable {
    private String street; private String city;
    public void setStreet(String s) {street = s;}
    public String getStreet() { return street;}
    public void setCity(String c) { city = c;}
    public String getCity() { return city;}
}

void processAction(ActionRequest req, ActionResponse resp) {
    Address sampleAddress = new Address();
    sampleAddress.setStreet("myStreet");
    sampleAddress.setCity("myCity");
    QName name = new QName ("http:examples.com/events", "Address");
    resp.setEvent(name, sampleAddress);
```
Deployment descriptor sample for receiving an Event

```xml
<event-definition>
  <qname xmlns:x="http://examples.com/">
    x:AddToAddressBook
  </qname>
  <alias xmlns:x="http://examples.com/">
    x:emailAddress822
  </alias>
  <value-type>java.lang.String</value-type>
</event-definition>

<portlet>
  <supported-processing-event>
    <qname xmlns:x="http://examples.com/">
      x:AddToAddressBook
    </qname>
  </supported-processing-event>
</portlet>
```
Event processing in the portlet

void processEvent(EventRequest req, EventResponse resp) {
    ...
    Event event = req.getEvent();
    if (event.getName().equals("AddToAddressBook")) {
        String emailAddress = event.getValue();
        ...
    }
}
Demo Public Parameters
Public parameters in portal URLs

http://portal.com/...docid=document2...

http://portal.com/...docid=document3...

Portal Page

Navigator Portlet
- document1
- **document2**
- document3

Content portlet
content of document2

Render link to select document3

Portal Page

Navigator Portlet
- document1
- document2
- **document3**

Content portlet
content of document3
Public parameters

- Allow render parameters to be shared across portlets
  - Not restricted to the portlet application
  - May be even across pages
  - Lightweight coordination based on HTTP GET

- Example
  - The zip code of a selected city allowing different portlets (map, tourist information, weather) to display information for this city

- Semantic
  - Parameters are visible to the portal and allowed to be shared with other components
Public Parameters

> Re-use existing render parameter APIs
  > Allows to even enable JSR 168 portlets to use public render params by just giving them an JSR 286 deployment descriptor

> Define in the portlet.xml which render parameters are public
  > Has an simple string ID that the portlet can use in the code
  > Provides a QName and optional alias names for wiring the parameter

> Allow getting all public params via the PortletContext at runtime
Public parameters vs. Events

Different HTTP semantics
- Public parameters: GET semantic, render parameters can be changed via render links and define view state
- Events: POST semantics, part of the action phase, allowed to change persistent state

Advantages of using public parameters
- Less processing overhead, no action phase required
- Parallel rendering of portlets possible

Limitations when using public render parameters
- No active notification that something has changed

Public render parameters are encoded in the URL in some portals, like the RI or IBM WebSphere Portal
- Bookmarkability
- Support of browser back/forward button
- Caching in the browser
Agenda

➢ JSR 286 Overview
➢ New Features in JSR 286
  • Coordination
  • Resource Serving
  • Cookies and Headers
  • Portlet Filter
  • Other additions
➢ Using AJAX with Portlets
➢ Compatibility to V 1.0
➢ Relation to Web Services for Remote Portlets (WSRP)
➢ Summary
Resource serving

Resource serving in JSR 168: direct serving via the portal/portlet container

- Via using encodeURL(resourceURL)
- No portlet runtime context available

Additional in JSR 286: resource serving via the portlet

- New ResourceURLs that trigger a new lifecycle method serveResource
- Portlet context available (render params, portlet mode, window state, preferences...)
- No state changes on navigation state (render params, portlet mode, window state) or shared state allowed
- Protected via the portal access control

Resource Ids

- You can set a specific resource ID on a resource URL
- Default behavior of GenericPortlet is to try to forward the resource serving to the resource ID specified
Resource serving

Different cache levels of resource URLs
- For supporting caching of the resource at the browser
- Three types introduced: FULL, PORTLET, PAGE
- FULL
  - Resource content is fully cacheable in the browser
  - No access to the portlet mode, window state, render parameters
  - No Portlet URLs besides Resource URLs of type FULL
  - May even be shared across portlets using the property ResourceURL.SHARED and a QName
- PORTLET
  - Resource content is cacheable on a portlet level
  - Can be cached in the browser as long the portlet state does not change
  - No Portlet URLs besides Resource URLs of type FULL and PORTLET
- PAGE
  - Resource content is cacheable on the portal page level
  - Any change of state on the portal page requires a reload of the resource
  - All Portlet URLs are allowed
Resource serving API

- **ResourceURL**
  - setResourceID(String id)
  - setCacheability(String cacheLevel)
  - cacheLevels: full, portlet, page

- **resourceURL tag**

- **New lifecycle interface ResourceServingPortlet**
  - void serveResource (ResourceRequest req, ResourceResponse resp)
  - Support for different HTTP methods (GET, POST…)
    - Via getMethod()
  - ResourceRequest
    - Like render request + ability to get uploaded data
  - ResourceResponse
    - Like render response + full control over the output stream
  - Allow changing preferences
Resource serving request flow

© 2007 IBM Corporation

Client

Portlet container

Portlets A B C

Page request

New Page with ResourceURL in Markup of Portlet C

Resource request

serveResource

Resource markup
Code sample resource serving

- JavaServer Pages™ (JSP™) technology generating the resourceURL

```<portlet:resourceURL var="searchUrl"/>
<input id="combobox" dojoType="StateSelector"
    value="Start typing"
    dataUrl="<%=searchUrl%>search=%{searchString}"
    mode="remote">
```

- Portlet code

```java
public void serveResource(ResourceRequest request,
    ResourceResponse response) 
    throws PortletException, IOException {
    if (request.getParameter("search") != null) {
        // do search, e.g. return JSON fragment for Dojo widget
    }
```
Agenda

▷ JSR 286 Overview
▷ New Features in JSR 286
  • Coordination
  • Resource Serving
  • Cookies and Headers
  • Portlet Filter
  • Other additions
▷ Using AJAX with Portlets
▷ Compatibility to V 1.0
▷ Relation to Web Services for Remote Portlets (WSRP)
▷ Summary
Setting cookies and HTTP headers / HTML head attributes

> Portlets in v1.0 could not set cookies and HTTP headers
  - Portal has the control over the output stream to the client and body content may already be written

> Portlets in v2.0 can set cookies and HTTP headers / HTML head attributes
  - In render and resource life cycle methods
  - Restrictions for render response
    - Need to be done in the RENDER_HEADERS part of the render phase
    - May be overridden by the portal or other portlets
  - Restrictions on cookies
    - Cookies may be stored on the portal or get re-written and thus not accessible on the client
API

➢ HTTP headers
  • Setting: via the set / add property methods on the response
  • Retrieving: via the getProperty methods on the request

➢ HTML head attributes
  • Setting: via addProperty(String, org.w3c.dom.Element)

➢ Cookies
  • Setting: addProperty(javax.servlet.http.Cookie)
  • Retrieving: javax.servlet.http.Cookie[] getCookies()
Supporting setting Headers / Head elements / Cookies in render

- Headers / Cookies needs to be set before the document body starts
  - Buffer all output and at the end create the response to the client
    - Very inefficient
  - Split render into two parts: headers and markup

- JSR 286 allows portals to set a render request attribute RENDER_PART with values
  - RENDER_HEADERS for setting headers, cookies, title
  - RENDER_MARKUP for rendering the markup

- GenericPortlet takes care of this request attribute
  - for RENDER_HEADERS calls
    - doHeaders,
    - setNextPossiblePortletModes
    - setTitle
  - Calls dispatch to doXYZ or RenderMode annotation for RENDER_MARKUP
Code sample for setting cookies

Setting cookies

```java
public class MyPortlet extends GenericPortlet {
    protected doHeaders(RenderRequest req, RenderResponse resp) {
        Cookie cookie = new Cookie("myCookie", "42");
        resp.setProperty(cookie);
    }
}
```

Retrieving cookies

```java
protected doView(RenderRequest req, RenderResponse resp) {
    Cookie[] cookies = req.getCookies();
    if ( cookies != null ) {
        // find my cookie in the array and retrieve
        // value with cookie.getValue()
    }
```
Agenda

➢ JSR 286 Overview
➢ New Features in JSR 286
  • Coordination
  • Resource Serving
  • Cookies and Headers
  • Portlet Filter
  • Other additions
➢ Using AJAX with Portlets
➢ Compatibility to V 1.0
➢ Relation to Web Services for Remote Portlets (WSRP)
➢ Summary
Portlet filters

- Portlet filters and request / response wrappers available
  - Similar in large parts to the servlet filter model
  - Filters are declared in the DD via the filter and filter-mapping element
  - Filters can be restricted to specific portlet lifecycle methods in the DD via the filter-mapping element
  - One filter interface per portlet lifecycle
    - But implementations can implement multiple interfaces
    - Filter chain that gets called by the portlet container

- Available via new javax.portlet.filter package
Code sample Portlet Filters

Portlet Deployment Descriptor

// filter declaration
<filter>
    <filter-name>PortletFilter</filter-name>
    <filter-class>com.example.PortletFilter</filter-class>
    <lifecycle>RENDER</lifecycle>
</filter>

// filter mapping
<filter-mapping>
    <filter-name>PortletFilter</filter-name>
    <portlet-name>MyPortlet</portlet-name>
</filter-mapping>
Code sample Portlet Filters

> Portlet Filter Code

```java
public class PortletFilter implements RenderFilter {

    public void doFilter(RenderRequest req, RenderResponse resp, FilterChain chain) throws .. {
        PrintWriter pw = resp.getWriter();
        pw.write("Pre-processing");

        MyRenderResponseWrapper resWrapper =
            new MyRenderResponseWrapper(res);
        chain.doFilter(req, resWraper);

        pw.write("Post-processing");
    }
}
```
Agenda

- JSR 286 Overview
- New Features in JSR 286
  - Coordination
  - Resource Serving
  - Cookies and Headers
  - Portlet Filter
  - Other additions
- Using AJAX with Portlets
- Compatibility to V 1.0
- Relation to Web Services for Remote Portlets (WSRP)
- Summary
Extended Request Dispatcher capabilities

➢ Goal: Better support servlet-based frameworks
➢ Request dispatch includes now allowed for all life cycle methods
  • For action / event no markup can be returned
  • Allows servlet-based bridges to handle controller logic via servlets
➢ Request dispatcher forward allowed for all life cycle methods
  • Delegate to servlets for action handling
  • Delegate to JSP™ for complete markup generation
  • Leveraged by GenericPortlet for resource serving by forwarding to the specified resource ID if that reflects the path of the resource
Caching

➢ New API CacheControl
   • Allows to get and set cache settings

➢ Shared cache entries
   • Response can be cached across users
     => large performance gain for non-customizable portlets
   • Default defined in deployment descriptor via the cache-scope tag
   • Can be set on each response

➢ Validation based caching
   • In addition to the expiry time the portlet can provide a token for the currently returned markup
   • When content is expired portlet gets called with the content token
   • The portlet either
     • re-validates the content and set a new expiry time for the token or
     • create new content with a new token and a new expiry time
   • Based on the HTTP ETag validation caching scheme
Tag lib additions

- New tag for creating resource URLs
- New variables available via defineObjects
  - portletSession
  - portletSessionScope: Map of portlet session attributes
  - portletPreferences
  - portletPreferencesValues: Map of portlet preference values
- New attributes for the URL tags
  - escapeXml for turning of XML escaping, like in JSTL
    - Per default URLs are XML escaped
    - Default can be change in the portlet.xml via a new container-runtime-option element and setting javax.portlet.escapeXml to false
  - copyCurrentRenderParameters for copying the current render parameters
Portlet managed modes

- Allow portlets to specify their own application-specific portlet modes
  - From the portal point of view they are treated like the View mode
  - Portlet can specify a text and description for that mode
  - Portal may include this additional mode in the navigation area
- Portlet can register for portlet mode change events
  - preset render params or data in prefs/backend systems
- Portlets can return a list of meaningful new portlet modes in each render
  - Hint for the portal to render the appropriate controls
- Example
  - ShowShoppingCart mode that displays the shopping cart content
Java 5 features

> Annotations
  - Events handling methods can be annotated with `@ProcessEvent`
  - Action handling methods can be annotated with `@ProcessAction`
    - action name must be set on the ActionURL as value of the parameter `javax.portlet.action`
  - Render handling methods for a specific portlet mode can be annotated with `@RenderMode`

> Enums
  - Enum for user info attributes

> Generics
  - Leveraged Generics in the interfaces

> Support for Java 1.4-based implementations
  - There is a jar file for 1.4-based implementations with the Java 5 features removed
  - Allows to write portlets that run on Java 1.4-based containers
Misc

➢ Extended runtime Ids
  • Namespace is now valid for the lifetime of the portlet window
  • Portlet can access the portlet window ID at the request
    • Use this ID if a per portlet window cache key is needed
➢ PortletURL now accepts a writer
  • Much more efficient than creating Strings
➢ CC/PP support (JSR 188)
  • Available as attribute on the request
➢ Restrict the custom window states for a given markup
➢ Additional listener support
  • URL listeners
  • Servlet 2.4 listeners
➢ Resource bundle for portlet application level texts
  • In 1.0 it needed to be inline in the portlet.xml
Agenda

➤ JSR 286 Overview

➤ New Features in JSR 286
  • Coordination
  • Resource Serving
  • Cookies and Headers
  • Portlet Filter
  • Other additions

➤ Using AJAX with Portlets

➤ Compatibility to V 1.0

➤ Relation to Web Services for Remote Portlets (WSRP)

➤ Summary
AJAX usages in a Portal environment – JSR 168

Portal level
- Portal does an aggregation on the client (browser)
- Portal manages AJAX interaction
- Transparent to the portlet
- Possible with JSR 168 portlets
- Example:
  - Client-side aggregation theme of the IBM WebSphere Portal 6.1

Portlet level
- Portlet brings its favorite AJAX library
- Portlet has to manages end-point on its own
  - Provide servlet for serving the AJAX request
- JSR 168 is limited
  - response without portlet context (served via servlet)
  - no state changes for state managed by the portlet container(portlet mode, window state, render parameters, portlet preferences, portlet session attributes)
AJAX usage pattern with JSR 168

- Portlet WAR contains one portlet and one servlet
- Portlet is embedded into markup like any other portlet
- Servlet is directly accessed through JavaScript™ programming language in portlet markup
JSR 286 – leveraging resource serving

> Portlet owned AJAX calls
  * Full access to the portlet state
    - Via XmlHttpRequest and ResourceURLs
    - Allowed to change portlet preferences and portlet-scoped session attributes
  * Functionality restricted:
    - No state changes for navigational state (portlet mode, window state, render parameters)
    - No support for events

> Coordinated between portal and portlet
  * Not covered by JSR 286, but vendor specific solutions exist
  * Needs to include a client side library that the portlet can leverage
    - E.g. for allowing the portal to update the URLs on the page if the portlet sets new render parameters
  * There may a some help from groups like OpenAjax in the future
    - Defining a generic client side gadget programming model
AJAX usage pattern with JSR 168

- JavaScript programming language call points to the Portal Application
- Portal Application dispatches resource serving call to portlet
Agenda

- JSR 286 Overview
- New Features in JSR 286
  - Coordination
  - Resource Serving
  - Cookies and Headers
  - Portlet Filter
  - Other additions
- Using AJAX with Portlets
- Compatibility to V 1.0
- Relation to Web Services for Remote Portlets (WSRP)
- Summary
Portlet compatibility

- Interfaces are binary compatible between 1.0 and 2.0
  - JSR 168 portlets can run unchanged on a JSR 286 container

- Source compatibility achieved besides the following changes
  - RenderResponse.setContentType is no longer required before calling getWriter or getOutputStream.
    - Helps frameworks like JSF
  - getProtocol for included servlets / JSPs no longer returns null, but ‘HTTP/1.1’ in V2.0
    - Make behavior compatible with GenericServlet

- V 2.0 portlet deployment descriptor is an extension of V 1.0
  - You get a JSR 286 portlet by just changing the line in the portlet.xml that references the 2.0 xsd

- V 2.0 tag lib has a new namespace
  - Avoids clashes for portlets that get migrated from JSR 168
Important clarifications

- Come into play when migrating a JSR 168 portlet to a JSR 286 portlet
- XML escaping of portlet URLs produced via the portlet tag library
  - V 2.0 clarifies that the default is all portlet URLs are XML escaped
  - Default can be changed with the portlet container runtime option `javax.portlet.escapeXml`
- Defining multiple values for the same parameter name in the Portlet param tag
  - V 2.0 clarifies that if the same name of a parameter occurs more than once within PortletURLs the values must be delivered as parameter value array with the values in the order of the declaration within the URL tag
- Parameters set on the portlet URL and the post body are aggregated into the request parameter set
Agenda

- JSR 286 Overview
- New Features in JSR 286
  - Coordination
  - Resource Serving
  - Cookies and Headers
  - Portlet Filter
  - Other additions
- Using AJAX with Portlets
- Compatibility to V 1.0
- Relation to Web Services for Remote Portlets (WSRP)
- Summary
Web Services for Remote Portlets V2.0

- Standard protocol for accessing portlets as web service
- Defined at OASIS, chaired by IBM
- Members of the TC
  - BEA, R. Brooks, CA, NetUnity Software, Novell, Microsoft, Oracle, Tibco, SAP, Sun, Vignette…
- Schedule
  - Final since March 2008
- Closely aligned with JSR 286
  - Enables writing JSR 286 portlets and publish them as WSRP services at runtime
  - Contains all the coordination features and resource serving of JSR 286
  - Portlet developer does not need to know anything about WSRP
  - Portal administrators can simply decide to run portlets on remote servers
Agenda

- JSR 286 Overview
- New Features in JSR 286
  - Coordination
  - Resource Serving
  - Cookies and Headers
  - Portlet Filter
  - Other additions
- Using AJAX with Portlets
- Compatibility to V 1.0
- Relation to Web Services for Remote Portlets (WSRP)
- Summary
Summary

- JSR 286 is a large step forward
  - The size of the specification and the API more than doubled compared to V 1.0
- JSR 286 supports building composite application
  - via coordination means events and public parameters
- JSR 286 allows for additional AJAX use cases
  - Using resource serving through the portlet
- Better integration of portlets with servlet-based frameworks
  - Setting HTTP headers / cookies, filters, request dispatching
- Better scalability
  - Different cache levels for resource serving, shared cache entries, validation-based caching, PortletURL.write
- WSRP 2.0 and JSR 286 are closely aligned
  - publish a JSR 286 portlet as WSRP service
For More Information

➤ JSR 286
  • http://jcp.org/en/jsr/detail?id=286

➤ Reference Implementation
  • http://portals.apache.org/pluto/

➤ WSRP
  • http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrp
THANK YOU

Stefan Hepper