Docs

Tutorials and Labs

Themes

Script Portlet

Inter-Portlet Communication in IBM Script Portlet Using Public Render

Script Portlet Development Best Practices

MICHAEL BURATI / APRIL 14, 2015 / 0 COMMENTS

Script Portlet Development Best Practices, Tips and

Parameters

Business Process Manager Sample for IBM Script Portlet in WebSphere Portal

Using Node.js Build Tools with Script Portlet

Using an AngularJS Single Page Application as a Script Portlet

Using a Backbone.js Single Page Application as a Script Portlet

Accessing Server-Side Data in Script Portlet Using a Web Content Manager JSP Component

Getting Started with Script Portlet

Restrictions

Whether you are developing applications for use with the Script Portlet, or a portlet built by other means (RAD Portal Toolkit, Web Experience Factory, manual java coding etc), there are many choices you can make that may affect the performance, scalability, stability and efficiency of your application. The following are some suggested best practices when developing applications with Javascript, HTML and CSS for use with Script Portlet and IBM WebSphere Portal.

Follow existing JS, HTML, and CSS web application development Best Practices

One of the primary goals of Script Portlet is to allow developers to use their existing Javascript, HTML and CSS skills, and knowledge, to develop script based applications for WebSphere Portal.

Developers should follow many of the best practices that already apply to such types of web application development (Javascript coding practices, choosing a framework or Javascript API appropriate for the particular application use case, framework usage etc).

Script Portlet and Source Code Management across Environments

Using Single Page Applications with Script Portlet

Script Portlet Development Best Practices

Using Command Line Push to Deploy Script Portlet Applications

Adding a Script Portlet Application to the Site Toolbar in WebSphere Portal 8.5

Script Portlet Support for External Editors

Responsive, Mobile, and

Javascript Framework Selection

A goal of Script Portlet is to be Javascript framework agnostic, and not to dictate or recommend any single framework(s) as the one you must or should use. That being said, there are aspects of Javascript libraries and frameworks that may make them more or less suitable for use in a multi-portlet portal page environment, whether built with Script Portlet or any other portlet development framework. You may want to avoid or be extra careful with frameworks or aspects of frameworks that try to control or manipulate the entire <html> or <body> elements of a browser page, as that element is owned by the portal page when building portal applications, not by individual portlets. Some frameworks may manipulate the “body” element by default, but many provide a mechanism to specify the container element which the framework should operate within.

Well behaved portlets render, manipulate and affect only the container (e.g., <div>) that wraps the portlet markup, and avoid doing things that can adversely affect the portal page, it’s navigation elements and layout, and/or other portlets on the page. Not all frameworks and/or styles were developed with that sharing of the page (with other frameworks and/or other applications leveraging the same framework) in mind.

For example, a framework that only allows extensions and overrides to the core

Worklight

Administration

Web Content

Application Development

Integration

Product Documentation Resources

framework itself in a static manner, could be overridden in one way by one application imported to script portlet A and in another way by another application leveraging that same framework, imported to script portlet B, in a way that could cause collisions if combined in a single portal page.   When evaluating frameworks for use in a multi-application multi-portlet environment, consider how well (if at all) the framework allows per application module settings. Another consideration to pay close attention to is which features of a framework work well, are supported and a best practice when used multiple times within a single browser page, versus which features are not allowed more than once across a browser page.

Avoid large numbers of http requests

- If all editing of the script based application will be via external tools prior to pushing the application artifacts to the server then:
  - Leverage JS layer building and minification tools to reduce the number and size of your application Javascript artifacts prior to pushing the application to the server, for a more efficient application (at the minor expense of the resulting minified and combined Javascript not being editable via the browser based editor).
  - Provide common CSS via Portal Theme module.
  - Combine and minify application specific CSS into a single application stylesheet

to reduce http requests in order to avoid issues on slower networks.

- If you cannot combine application level Javascript modules into a single layer via external Javascript tools, and your application includes multiple JS files:
  - Leverage the Script Portlet 1.3 data-scriptportlet-combine-urls=”true” attribute on the main <html> element of your script application
  - This feature allows runtime combining of sibling local application script references in your main application page
  - This feature requires WebSphere Portal 8.5 CF03 and later releases (although CF05 or later is recommended, for proper use of the content toolbar functionality)

Choose styles that behave well in a multi-portlet portal page

- Avoid CSS that may manipulate and cause visual errors for the portal page and/or other portlets on the page
- For example, styles that force a banner to an absolute position (0,0 or other) on the page, end up overwriting and hiding portal page navigation controls.

- Styles that affect the entire html or body of a page or general div containers or commonly used classnames may adversely affect the portal page or other portlets in a way that the developers of those components did not intend.
- Styles should be scoped to uniquely named (by unique id or unique class) islands within the markup for that application or at a minimum be consistent across the portlets that will be used on a given portal page.
- Common branding across an enterprise portal is best provided by a portal theme module rather than developers trying to mimic or load the same common styles across multiple portlet based applications

Javascript Framework Loading

For production deployment use, Javascript frameworks (jQuery, Dojo, and similar) should be loaded by the Portal page using its modularized Theme support and not by each individual portlet, to avoid collisions, framework resets by mulitple portlets and inefficient re-loading of the frameworks.

Note: For initial prototype and single portlet demos, some sample apps do load framework JS directly from the portlet app during development, but this should not be used for production and multiple portlets on a page should not both load the same *framework, even for sample or demo purposes.*

In order to allow local development and testing of script portlet applications, Script Portlet 1.3 allows the use of a new html directive attribute “data-scriptportlet-theme-capability” that may be applied to script reference tags, to indicate that such script references should be commented out when the application is pushed or imported to the portal server.

For example, the following script reference to a hosted copy of jquery may be used locally, but will be commented out when pushed or imported to the server as a script portlet.
<script src=”<URL to hosted copy of jQuery>” data-scriptportlet-theme-capability=”jquery”/>

Portal provides an out of the box module for jquery and documentation for how to add it to a theme’s contributions folder and then to a profile associated with that theme.

Application Folder Layout

A properly laid out application tree (folder structure on disk and pushed to the server, and in compressed zip files used with import) should follow the following structure:

- index.html (or another file specified as the primary page for the application with mainHtmlFile option to sp command)

    - should be at the root of the folder structure

- optional preview-image.jpg at the root (sibling of primary html page) will be used as the application icon when the app is pushed to a site area enabled for use with the WebSphere Portal 8.5 CF05 or later content toolbar.
- create subfolders of the root, for images, js, css, json, etc.

Use of Source Code Control to Manage Complex Application Projects

IBM Web Content Manager provides projects and versions, primarily for content workflow, but is not a full source code control system in that it does not provide branches and merges, automatic association of change sets with work items etc.

For long lived application projects (i.e., as opposed to quick prototypes or demos), you may want to consider using a source control system that provides versioning, branching and merging, association with defect tracking and enhancement change sets etc, as you would any application development project.

The “sp push” command line utility allows publishing versioned releases of applications built with the use of existing external developer tools and source code control

External Development and Publishing Applications

With Script Portlet you can leverage your existing development tools, and push the results to a site area (content folder) within a WCM library.   An out of the box site area is provided with the Script Portlet installation, with the name: “Script Portlet Library / Script Portlet Applications” and applications may be pushed to this out of the box site area or custom site area(s) enabled for use with the content toolbar may be created as described in the Script Portlet documentation.

Pushing an application to a site area instead of to an existing portal instance on a page, allows line of business page authors to leverage that portlet app on multiple portal pages, and for the application to continue to be available for use on other portal pages, even if the initial portal page is removed.   In addition, using Portal 8.5 CF05 and later, adding the application from the content toolbar to the page creates a reference to the application in the site area, such that updates to the application pushed from the development release process become available to instances added to portal pages immediately. Prior to CF05, the content toolbar would “copy” the script portlet application from the site area to a portal page, and the instance(s) on portal page(s) could become out of synch with the latest version in the site area, as updates and fixes are published there.

You may use the Portal Content Toolbar feature which allows a preview-image.jpg in the root of an application (eg, sibling of main index.html) to be used as an icon for that application when viewed alongside other portlet applications from that content toolbar. A default generic icon will be provided if no preview-image.jpg is provided, but having a custom preview-image icon per application will help distinguish the application from others in the hopefully growing list of application icons that your page developers will see in the toolbar's content palette for your script application site area(s).

Application Storage in Web Content Manager

It is a good practice to store your apps in the "Script Portlet Applications" site area or a custom site area and not in the portal pages site area under the Portal Site library. As described in the previous section, pushing applications from an external location makes it easy to store your application(s) under such application site areas and then to leverage them from the content toolbar. Adding an empty "Script Portlet" from the Application toolbar adds an empty content item to that page under the portal site area, and is less desirable as the content item storing the application artifacts are then tied to that portal page by default, rather than in a location that can be leveraged by page authors creating or updating other portal pages. You may use the Editor’s “Save As” functionality to move a portal page specific application instance from the portal page site area to an appropriate application site area.

Using Portal Projects to stage Deployment on Live Servers

On servers with existng live applications and users, you may use the Portal “Project” feature when making updates to those applications (whether via Push, Import or Editor) into a “Draft” copy of a page (by setting the portal page view to that project prior to clicking Import) to allow for review of the updates in the portal page context prior to publishing that project.   This can avoid breaking existing users of the page with updates that may not have been qualified in a portal page context yet.

When using command line push to upload an application to a draft content item associated with a project, use the -projectID command line argument or setting in sp-config.json to specify the corresponding project name or id.

The command line tool supports “sp list projects” to obtain a list of current project names and ids, to verify that you are using the correct name or id when pushing the application.

When using Export Config and command line “Push” to update a portlet on a project based portal page, export sp-config when “in” the project, to get appropriate config settings such that the application is pushed back to a draft in the corresponding project.

Troubleshooting and Debugging Techniques

The best place to start looking when script based applications fail, is typically to use your browsers’ Javascript console, network tab and debugger to diagnose and determine the cause of application specific issues.   Those browser based diagnostic tools are often the best starting point for figuring out what’s going wrong with your custom application (missing artifact from your zip causing an HTTP 404 error, dynamic URL couldn’t be found or fixed up with WCM Plugin markup to generate an actual WCM URL to the imported element etc).

For issues that only occur when an externally pushed application is running on a portal page as a script portlet (not standalone), you may use the browser based editor to try out changes and fixes that help the application behave better within the portal page and then apply the change to the external source control version once the issue is resolved.

In addition to using the Script Portlet Editor to view what was imported, you may use the WCM Authoring Portlet to drill down to the Script Portlet Content Item (typically under the

Portal Site WCM library, and associated with the page where you Script Portlet instance is located) to see the individual WCM elements created for the imported or pushed applications.

For applications that can be run locally (no portal or portlet dependencies) it is a good practice to test problematic applications external to portal with browser pointed at the local filesystem version of the application to isolate whether problem(s) are related to the app itself or integration of the application with portal and/or portlets on the page. If it runs fine locally, try it on a portal page by itself next, to determine if other portlet(s) on the page are causing the issue(s).

Leverage Available Samples, Articles, and the Script Portlet Forum

- Samples of various techniques
- Articles describing various features and techniques
- Forum discussions, Q&A
- Follow documented practices and procedures
    - e.g. upgrade requires an uninstall then install set of steps
    - additional steps for virtual portal and additional steps to enable non-default

security settings

Some samples are intended only to show a single feature or technique and don’t necessarily leverage all best practices and techniques available and thus are not intended to be copied and used as is. For example, for ease of setup and trial use, some samples include all employee or contact images within the portlet app itself. It is not necessarily a best practice to be storing all of your contacts or employee images directly with your script portlet app instance. The lists of employees/contacts may come from a back-end REST service and include absolute URLs to where those images are stored in a static artifact serving environment.

Learn the individual techniques from the samples and then build your application(s) leveraging those techniques and these and other best practices, to build and test production ready applications.

Unique Names and IDs

- Avoid collisions across portlets by using unique names for JS variables and HTML id values.
- Manually specified unique names may not be enough (eg, if the same portlet is used multiple times), in which case you may leverage portlet namespaces

- Leverage the Script Portlet 1.3 feature that provides ability to replace __SPNS__ token with Script Portlet Namespace plugin when app is imported/pushed
  - The Script Portlet Namespace plugin renders the portlet namespace when the portlet artifact renders to the browser
- Try multiple copies of the application (as multiple script portlet instances) on a portal page during FVT, to verify lack of collisions

Use a Four (or more) Tier Architecture where appropriate

- Portal and WCM should be providing the content and site management of your applications, not all the data manipulation and logic
- A best practice is to retrieve only the minimal data necessary for each application from back ends and not pulling back more than you need and then filtering and manipulating it in portal or browser.
- For back ends where some data manipulation is necessary, leverage a data tier between portal and the backend to do such manipulation and filtering, rather than pulling more data and work into portal or the application than is necessary.

Application State

Many script based applications are stateless and just fetch the current value(s) from some back end for display, but there are also applications that require maintaining some amount of state information.

- For minimal state (a few bytes) you could use a cookie but remember the cookie will flow on each request, so you don’t want to transfer much data, so only do this if very small, and also look at the security aspect of the data you want to store there. If it is, for example the current selected, or index, or such, those are good things to use.
- For HTML5 browsers/apps, then you may use local storage. There are limits, but you can store megabytes of state here. What is good, is you have full control over the serialization of this data, and the size and it isn’t transferred to the server. Don’t use this though if you need some of this state server side for some decision.
- If not HTML5, there are polyfills for local storage, some fall back to sqlite, others to userData, and others to cookies. Look at the data you want in the storage, and browser level to find out if one of these provides a reasonable fallback. Overall most aren’t bad, but the cookie fallback is not ideal, so stay away from that when possible.
- If you need this state for other server decisions, or if it is large, them we typically suggest creating a servlet that uses dynacache to save the state. Some benefits here

include the ability transform this state, so the browser only sees the state it really needs and backends can access this same state for their decisions. It is independently scaleable(you could farm or cluster this servlet) and you have a caching layer for the data. If you need this data to persist(meaning it isn't transient), then you could back this dynacache up to a database or a webservice for persisting the state.

- As a last resort, if none of the above options suffice and you have no similar alternatives, then you could possibly use a WCM JSP tag in your script portlet html file, to access the server side state. This has access to a session, can call EJBs, dynacache, etc… We suggest this as a last recourse as it is better for this state be outside of portal/WCM's jvm and independently cacheable.

Migrating Applications from Staging to Production

- Store your apps in "Script Portlet Applications" site area or custom site area, not in Portal Site library under the portal pages site area
- Leverage portal's "Managed Pages" feature
  - to maintain unique copies of Script Portlet library configuration and site area across virtual portals
  - to make migrating pages between portal environments easier and more

automated.

- Leverage WCM “Syndication” to migrate between staging and production servers

Use the Latest Version of Portal and Script Portlet Available (including CFs)

While Script Portlet supports WebSphere Portal 8001 and 8.5, you get the most features by moving to the latest available versions

- 8001CF11 and 8.5CF1 and later required for full Script Portlet Virtual Portal support
- 8.5 and later required for Editor Save-As support of images and binary files
- 8.5 CF03 and later are required for use of the resource aggregation Javascript combiner feature mentioned above
- **8.5 CF05 and later is the recommended minimum level for the best experience with the most features provided with Script Portlet 1.3** and is required for “reference” to site area copy of applications added via the Content Toolbar (prior to CF5 the content toolbar would “copy” the application to the page, thus missing out on updates to the application later pushed to the site area)

Leave a comment

Name *

Email *

Comment

Submit Comment