Best Practices for Javascript

> **Note**
>
> This is WIP (Work In Progress)

Introduction

This page is attempting to capture JavaScript best practices used when creating widgets for Sakai. It includes a set of good general practices as well as references to external sources. Feel free to add your own.

Links to resources

- Sakai 3 UX Development Guidelines and Information
- Setting up Aptana Studio for Sakai3 UX development
- Tips for High Quality Javascript (aaronz-sakai blog)
- DHTML Developer Checklist from the Fluid project - includes recommendations on Javascript packaging as well as markup and event handlers for accessibility

Best Practices

Hide the container tag in widget templates

The containing tag for widget templates should always be hidden using style="display: none;". This will keep the user from seeing the static template content while the page is still loading. It also fixes an issue with CSS rendering when CSS is placed into the page dynamically.

JSDoc

When you are writing code, it should always be well documented. The syntax we are using is the one from JSDoc, which is very similar to Javadoc.
It's also embeded in Aptana Studio. So if you put you cursor just above a function and type "/**" and hit Enter, some JSDoc will already be shown.

```
/**
*
* @param {Boolean} addToAll
* @param {Function} functionOnComplete
*/
var getAllPollwow = function(addToAll, functionOnComplete) {
...
}
```

Of course you should fill in the necessary information:

```
/**
* Get all the pollwows from the current site
* @param {Boolean} addToAll Add to all the pollwows
* @param {Function} functionOnComplete Function to be executed on completion
*/
var getAllPollwow = function(addToAll, functionOnComplete) {
...
}
```

Url:
http://jsdoc.sourceforge.net/
http://en.wikipedia.org/wiki/JSDoc

Use Trimpath JST (Javascript Templates)

Trimpath JST is an open-source component that lets you have template-based programming. We mainly use this because with IE6.0 some DOM manipulations cause the page to reload. When you do a lot of those manipulations, the website loads very slow. Although you should do as much of the processing inside your JavaScript files, it is also possible to use some functionality (like if/for /var) inside the templates.

Some useful things to keep in mind:

1. When you write a template, make sure the comment tags are directly next to a div tag.

   ```
   <div> <!--
   ```

   will not work, but

   ```
   <div><!--
   ```

   will work.
2. Although the founders of JST always use templates inside a textarea, it's possible in Sakai to use them inside a div.
3. First render the template and then show the container div.

Example:

twitter.html

```
<div id="twitter_container" style="display: none;">

<!-- TWITTER CONTAINER -->
<div id="twitter_sub_container"></div>
<div id="twitter_template_get_status"><!--
        <label class="twitter_input_label" for="twitter_input_screen_name">Screen name:</label>
        <input class="twitter_input_text" id="twitter_input_screen_name" type="text" value="${screen_name}"/>
        <div class="buttonBar twitter_buttonBar">
                <a class="button twitter_button" href="javascript:;" id="twitter_link_get_status">OK</a>
        </div>
--></div>
</div>
```

twitter.js

```
$("#twitter_sub_container", rootel).html($.TemplateRenderer($('#twitter_template_get_status'), json));
$("#twitter_container", rootel).show();
```

Url:
http://code.google.com/p/trimpath/wiki/JavaScriptTemplates

Use a $.parseJSON function instead of eval

To convert a JSON text into an object, you can use the eval() function. eval() invokes the JavaScript compiler. Since JSON is a proper subset of JavaScript, the compiler will correctly parse the text and produce an object structure. The text must be wrapped in parenthesis to avoid tripping on an ambiguity in JavaScript's syntax.

```
var myObject = eval('(' + myJSONtext + ')');
```

The eval() function is very fast. However, it can compile and execute any JavaScript program, so there can be security issues. The use of eval is indicated when the source is trusted and competent. It is much safer to use a JSON parser. In web applications over XMLHttpRequest, communication is permitted only to the same origin that provide that page, so it is trusted. But it might not be competent. If the server is not rigorous in its JSON encoding, or if it does not scrupulously validate all of its inputs, then it could deliver invalid JSON text that could be carrying dangerous script. The eval() function would execute the script, unleashing its malice.

To defend against this, a JSON parser should be used. A JSON parser will recognize only JSON text, rejecting all scripts. In browsers that provide native JSON support, JSON parsers are also much faster than eval. It is expected that native JSON support will be included in the next ECMAScript standard.

```
var myObject = $.parseJSON(myJSONtext);
```

Source:
http://www.json.org/js.html

Naming convention

To avoid ambiguity in naming there is one main rule you always should apply when writing names in html: always start the name of a class, id or name attribute with widgetname and then an underscore. So if the name of your widget (is the same as the id of the widget in dev/configuration/widgets.js) is twitter; your names always should start with twitter_.

```
<div id="twitter_container"></div>
```

Coding tips

Initializing An Array

Always use brackets when initializing an array, like this

```
var myArray = [];
```

Creating an Array with brackets instead of with the "new" constructor avoids a bit of confusion where you want to initialize only one integer. For instance:

```
var badArray = new Array(10); // Creates an empty Array that's sized for 10 elements.
var goodArray = [10];          // Creates an Array with 10 as the first element.
```

Also the tool we use for correctness and valid code practices (jsLint) doesn't allow "new Array()"

Initializing an object

```
var goodObject = {};
var badObject = new Object();
```

JSON

We use JSON instead of XML to store data in JCR because it is easier to handle.

live() instead of bind()

live() binds a handler to an event, just like bind, for all current - and future - matched element(s). The main words in the last sentence are "future element(s)". Once you bound the event to the element, it will be bound during the whole lifetime of a page.

It is very handy when you use it in combination with Trimpath JST (JavaScript Templates). You should also notice in the examples that it is not possible to use $("a", someElement).live(...) but this would work: $("li a").live(...).

With live()

```
$("#" + tuid + ".pollwow_close").live("click", function(e, ui) {
```

With bind()

```
$(".pollwow_close", rootel).bind("click", function(e, ui) {
```

Url:
http://docs.jquery.com/Events/live

JSLint

JSLint is a JavaScript program that helps you to solve bugs/problems in your JavaScript code. When you use this program it's best to check the following settings:

- Assume a browser (*otherwise functions like 'alert' will cause errors*)
- Disallow undefined variables
- Disallow leading _ in identifiers
- Disallow == and != (*you should use === and !==, which also checks the type of the elements, instead*)

Usually all your code should pass through JSLint, but there is one exception:

> **Be Careful**
>
> Try to avoid if possible

```
/**
 * Add a new option
 */
var addNewOption = function() {
        if (json) {
                bindOptions();
        }
};

/**
 * Bind all the elements in the options div
 */
var bindOptions = function() {
        /** Bind add new option on enter in last box */
        if (addNewOptionOnEnter) {
                if (json.pollwow.options.length > 0) {
                $("#pollwow_question_input" + (json.pollwow.options.length-1), rootel).keypress(function(e) {
                                if (e.which === 13) {
                                        addNewOption();
                                        return false;
                                }
                        });
                }
        }
};
```

In the code above, you should notice that it seems like addNewOption calls the function bindOptions and in reverse (which is a circular dependency and should be avoided). But if you look closer, you'll see that bindOptions only binds the function and not directly executes it.

Url:
http://www.jslint.com/

Script tag

Always close the script with his proper ending tag "</script>"
Your HTML file should not contain any JavaScript or CSS (except for the style="display: none;" for hiding the containers.)

Writing and importing a widget

Your html-file should look like this

```
<!-- STYLESHEETS -->
<link rel="stylesheet" type="text/css" href="/devwidgets/quiz/css/quiz.css"></link>

<!-- CONTAINER -->
<div>
...
</div>

<!-- JAVASCRIPT -->
<script type="text/javascript" language="JavaScript" src="/devwidgets/quiz/javascript/quiz.js"></script>
```

Your should always start your widget with the following JavaScript-code (example quiz-widget):

```
var sakai = sakai || {};

/**
 * Initialize the quiz widget
 * @param {String} tuid Unique id of the widget
 * @param {String} placement Widget place
 * @param {Boolean} showSettings Show the settings of the widget or not
 */
sakai.quiz = function(tuid, placement, showSettings) {
//Your widget code
}

sakai.api.Widgets.widgetLoader.informOnLoad("quiz");
```

Now to register your code as a widget you should add the following code to the config.json file (located at devwidgets/widgetname /config.json)

```
quiz :
{
    "ca":true,
    "description":"Quiz widget",
    "id":"quiz",
    "img":"/devwidgets/quiz/images/quiz.png",
    "name":"Quiz",
    "showinsakaigoodies":true,
    "url":"/devwidgets/quiz/quiz.html"
}
```

Key/value pairs for config.json

property	category	type	value(s)	used where	comments
i18n	i18n	object	key value pair about which translations there are for the widgets		
height	iframe	number	height of an iframe	sites, *	Use it if iframe: true
iframe	iframe	boolean	is it an iframe or not	/	
scroll	iframe	boolean	add scrollbars for an iframe	/	Should be added (just like height) if you want scrollbars for an iframe
description	info	string	info about the widget	portal	
hasSettings	info	boolean	does the widget has a settings screen	portal	If this is true and you hover over a portal widget, you see a settings link when you click on the pencil icon
id	info	string	id of the widget, which should be the same as the above one	portal, sites, *	
img	info	string	url for the image of the widget	sites	
name	info	string	small info about the widget	portal, sites	Used in portal to show a name of each box + this is the name of the widget when you click on it
url	info	string	url for the widget (for internal widgets it starts with /devwidgets, for iframe widgets probably http:// or https://)	portal, sites, *	

multipleinstance	info	boolean	allow multiple instances of a widget	sites	
ca	info - sites	boolean	is in a category in sites or not	sites	This has to be true if one of the showin... variables is true
showinmedia	info - sites	boolean	show in the media category	sites	
showinsakaigoodies	info - sites	boolean	show in the sakai goodies	sites	
showinsidebar	info - sites	boolean	show in the sitebar for a site	sites	
personalportal	location	boolean	add the widget to the portal page or not	portal, sites	
siteportal	location	boolean	show the widget on site-dashboard pages	/	

Adding external JavaScript libraries

If you would like to add an extra JavaScript file you should always add the following line at the bottom of that file.

```
sakai.api.Widgets.widgetLoader.informOnLoad("quiz");
```

And of course add the library in the html file.

```
<script type="text/javascript" language="JavaScript" src="/devwidgets/quiz/lib/rijndael.js"></script>
```

Keeping the number of requests to a minimum

If you are doing a GET request to the REST services, try to use the browsers cache as much as possible.
However, if you do want to fetch unique data, you should set the cache setting to false.
ex:

```
$.ajax({
    url: "/rest/sites",
    cache: false,
    success: function(data) {
        // Do something with the data.
    },
    error: function(data) {
        // Do something because the request failed.
    }
});
```

Prototype methods

Do not just add extra functions to standard JavaScript classes.
The following code might seem very useful but could actually be very harmful, it might break other people their code.

```
String.prototype.trim = function() { return this.replace(/^\s+|\s+$/, ''); };

..

var s = str.trim();
```

If you want to add extra functionality to the build-in classes in JavaScript, be sure to check sakai_magic.js (under /dev/_lib/sakai_util /.)

Helpful Notes

Cross server AJAX workarounds

AJAX requests cannot be sent to a domain that is not the same as the one the page originated from (i.e. I got a page from server1.com so I can send ajax requests to server1.com but not to server2.com or other domains). This is in place as a security measure and is unlikely to change in browsers any time soon. There are a few ways to get around this so that content from other sites can be loaded into pages (in order of common usage):

Use an iframe

This is how google calendar and many of the google widgets are embedded in pages. Basically the iframe points to the server and loads the entire page at the address into the iframe. Here is a sample embedded calendar tag.

```
<iframe src="http://www.google.com/calendar/embed?src=p%23weather%40group.v.calendar.google.com
&ctz=Europe/London" style="border: 0" width="800" height="600" frameborder="0" scrolling="no"></iframe>
```

The disadvantages (or advantages?) here are that this is really just dropping content into the page and does not actually make it part of the page. There is normally no way to interact with the content or to have the content interact with the current server (it is effectively blocked from access your server by the same rules).

Create a proxy on your server to point to the other server

In this case you are basically setting up a URL on your server which actually redirects to the other server. It allows the content to be embedded in the page as if it were part of your server and to make requests to your server so this should only be done for trusted sites.

Use a JavaScript library

There are libraries like ACD (http://www.ajax-cross-domain.com/) which provide the ability to retrieve content from external sites. These libraries allow direct calls to be made to other server as if those servers were yours. These typically take advantage of the script tag hack.

```
<script type="text/javascript" src="http://www.ajax-cross-domain.com/cgi-bin/ACD/ACD.js
?uri=(http://216.92.176.52/?name=john)"></script>
<script type="text/javascript"> alert(ACD.responseText); </script>
```

References:
http://en.wikipedia.org/wiki/Same_origin_policy
http://developer.yahoo.com/javascript/howto-proxy.html
http://www.ajax-cross-domain.com/
http://www.domscripting.com/blog/display/91
http://snook.ca/archives/javascript/cross_domain_aj/