JavaScript coding standard – 2013-02-10

Why we need a coding standard

- Minimizes the chance of coding errors
- Results in consistent, readable, extensible, and maintainable code
- Encourages code efficiency, effectiveness, and reuse
- Encourages the use of JavaScript's strengths and avoids it weaknesses
- Enhances understanding throughout the coding team

General guidelines

- **Investigate third-party code** like jQuery plugins before building your own - balance the cost of integration and bloat versus benefits of standardization and code consistency
- **Avoid embedding** JavaScript code in HTML; use external libraries instead
- **Minify, obfuscate, and gzip JavaScript and CSS** before go-live (e.g. Uglify, Closure Compiler, YUI Compressor)

Code layout and comments

Use white space to help comprehension

- **Indent two spaces** per code level
- **Use spaces, not tabs** to indent as there is not a standard for the placement of tabs stops
- **Limit code and comment lines to a maximum of 78 characters**
- **Follow a function with no space** and then its opening left parenthesis, **(**.
- **Follow a keyword with a single space** and then its opening left parenthesis, **(**.
- **Each semicolon ;** in the control part of a **for** statement should be followed with a space.
- **Align like elements vertically** to aid comprehension, within reason
- **Prefer single quotes** over double quotes for string delimiters

Organize your code in paragraphs

- **Organize your code in logical paragraphs** and place blank lines between each
- **Each line should contain at most one statement or assignment** although we do allow multiple variable declarations per line
- **Place white space between operators** and variables so that variables are easier to spot
- **Place white space after every comma**
- **Align like operators** within paragraphs
- **Indent comments** the same amount as the code they explain
- **Place a semicolon at the end of every statement**
- **Place braces around all statements in a control structure**. Control structures include `for`, `if`, and `while` constructs, among others.

Break lines consistently

- **Break lines before operators** as one can easily review all operators in the left column.
- **Indent subsequent lines of the statement one level** e.g. two spaces in our case.
- **Break lines after commas separators**.
- **If there is no closing bracket or parenthesis**, place it on its own line. This clearly indicates the conclusion of the statement without forcing the reader to scan horizontally for the semicolon.

Use K&R style bracketing

- **Place the opening** parenthesis, brace or bracket at the end of the opening line
- **Indent the code** inside the delimiters (parenthesis, brace, or bracket) one level - e.g. two spaces
- **Place the closing** parenthesis, brace or bracket on its own line with the same indentation as the opening line

Comment strategically

- **Align comments** to the same level as the code they explain
- **Comment frugally.** Prefer to comment at the paragraph level.
- **Non-trivial functions should explain** the **purpose** of the function, what **arguments** it uses, what **settings** it uses, what it **returns**, and any exceptions it **throws**.
- **If you disable code,** explain why with a comment of the following format: `// TODO <YYYY-MM-DD> <username> - <comment>`.

Example of API documentation for a function

```
// BEGIN DOM Method /toggleSlider/
// Purpose   : Extends and retracts chat slider
// Required Arguments :
//   * do_extend (boolean) true extends slider, false retracts
// Optional Arguments :
//   * callback (function) executed after animation is complete
// Settings  :
//   * chat_extend_time,   chat_retract_time
//   * chat_extend_height, chat_retract_height
// Returns   : boolean
//   * true  - slider animation activated
//   * false - slider animation not activated
// Throws    : none
//
toggleSlider = function( do_extend, callback ){ … };
// END DOM Method /toggleSlider/
```

Variable names

Use common characters

- **Use only a-z, A-Z, 0-9, underscore, or $**
- **Do not begin a variable name with a number**

Communicate variable scope

- **Use camelCase when the variable is full-module scope** (i.e. it can be accessed anywhere in a module namespace).
- **Use under_scores when the variable is not full-module scope** (i.e. variables local to a function within a module namespace).
- **Make sure all module scope variables have at least two syllables** so that the scope is clear. For example, instead of using a variable called **config** we can use the more descriptive and obviously module-scoped **configMap**.

Objects typically have a concrete "real world" analog and we name them accordingly:

- **An object variable name should be a noun** followed by an optional modifier, e.g. **employee** or **receipt**.
- **Make sure a module-scoped object variable name has two syllables** or more so the scope is clear, e.g. **storeEmployee** or **salesReceipt**.
- **Prefix jQuery objects with a $**. This is a pretty common convention these days, and jQuery objects (or collections as they are sometimes called) are quite prevalent in SPAs.

Variable Name Convention (Indicator – Local Scope – Module scope)		
Boolean type		
is (indicates state)	is_retracted	isRetracted
do (requests action)	do_retract	doRetract
has (indicates inclusion)	has_whiskers	hasWhiskers
String type		
string (or str)	direction_string	directionString
name	employee_name	employeeName
msg (or message)	employee_msg	employeeMsg
text	email_text	emailText
html	body_html	bodyHtml
id (identifier)	email_id	emailId
date	email_date	emailDate
Integer type		
int	size_int	sizeInt
--- (convention)	i, j, k	---
count	employee_count	employeeCount

Variable Name Convention (Indicator – Local Scope – Module scope)		
index or idx	employee_index	employeeIndex
Number type		
num	size_num	sizeNum
--- (convention)	x, y, z	---
coord (coordinate)	x_coord	xCoord
ratio	sales_ratio	salesRatio
time	extend_time	extendTime (in seconds or milliseconds)
Regex type		
regex	regex_filter	regexFilter
Array type		
list	timestamp_list	timestampList
Map type		
map	employee_map	employeeMap
map	receipt_timestamp_map	receiptTimestampMap
Object type		
--	employee	storeEmployee
--	receipt	salesReceipt
$ (jquery object)	$area_tabs	$areaTabs
Unknown type		
http_data, socket_data	httpData, socketData	Unknown data type received from an HTTP feed or web socket
arg_data, data	---	Unknown data type received as an argument

Naming functions

- **Function variable names should always start with a verb followed by a noun**
- **Module-scoped functions should always have two syllables** or more so the scope is clear, e.g. **getRecord** or **emptyCacheMap**.

Indicator	Meaning	Local scope	Module scope
fn	Generic function indicator	fn_sync	fnSync
curry	Return a function as specified by argument(s)	curry_make_user	curryMakeUser
destroy / remove	Remove a datastructure, e.g. remove an array. Implies that data references will be tidied up as needed	destroy_entry, remove_element	destroyEntry, removeElement
empty	Remove some or all members of a data structure without removing the container - e.g. remove all elements an array but leave the array intact	empty_cache_map	emptyCacheMap
fetch	Return data fetched from an external source, e.g. from an AJAX or web socket call	fetch_user_list	fetchUserList
get	return data from an object or other internal data structure	get_user_list	getUserList
make	Return newly constructed object (does not use the new operator)	make_user	makeUser
on	Event handler; single word for event	on_mouseover	onMouseover
save	Save data to an object or other internal data structure	save_user_list	saveUserList
set	Initialize or update values as provided by arguments	set_user_name	setUserName
store	Send data to an external source for storage, e.g. via an AJAX call		
update	Similar to set, however, has a "was previously been initialized" nuance	update_user_list	updateUserList

Variable declaration and assignment

- **Use {} or []** instead of `new Object()` or `Array()` to create a new object, map, or array
- **Use utilities to copy objects and arrays.** See jQuery.extend()
- **Explicitly declare all variables first** in the functional scope using a single var keyword
- **Use named arguments** whenever requiring 3 or more arguments in a function, as positional arguments are easy to forget, and are not self-documenting
- **Use one line per variable assignment.** Order alphabetically group logically
- **More than one declaration** may be placed on a single line.

Functions

- **Assign all functions to variables**. This reinforces their first-class status in JS
- **Use functions to provide scope**, the JavaScript 'let' statement has questionable value
- **Declare all functions before they are used**
- **Use the factory pattern for object constructors**, as it better illustrates how JavaScript objects actually works, is very fast, and can be used to provide class-like capabilities.
- **Avoid pseudo classical object constructors** - those that take a **new** keyword. If you must keep such a constructor, capitalize its first letter
- **When a function is to be invoked immediately,** wrap the function in parenthesis so that it is clear that the value being produced is the result of the function
- **Use jQuery** for DOM manipulations

Namespaces and file layout

Namespace basics

- **Claim a single, short name** (2-4 letters) for your application namespace, e.g. spa.
- **Subdivide the namespace per responsibility**, e.g. spa.data, spa.model, spa.shell, etc.

JavaScript files

- **Include third-party JavaScript** files first in our HTML so their functions may be evaluated and made ready to our application.
- **Include our JavaScript** files in order of namespace. You cannot load namespace **spa.shell**, for example, if the root namespace, **spa**, has not yet been loaded.
- **Give all JavaScript files a .js suffix**
- **Store all Static JavaScript files** under a directory called **js**.
- **Use the template** to start any JavaScript module file.
- **Name JavaScript files** according to the namespace they provide, one namespace per file.Examples includie spa.js, spa.shell.js, spa.chat.js

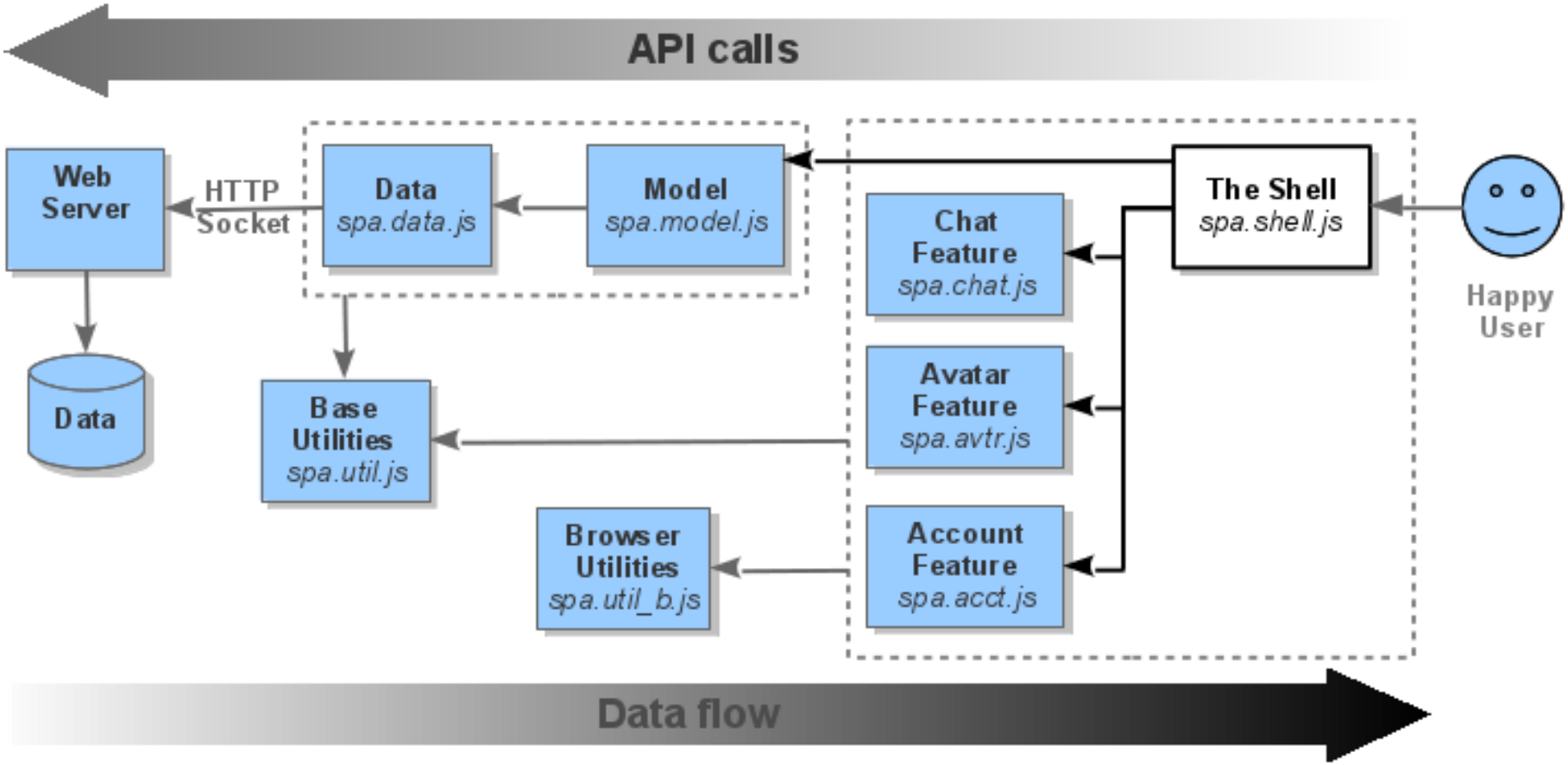
CSS files

- **A CSS file should be created for each JavaScript file that generates HTML.** Examples: spa.css // spa.* namespace spa.shell.css // spa.shell.* namespace spa.slider.css // spa.slider.* namespace
- **Give all CSS files a .css suffix**
- **Store all CSS files** under a directory called **css**
- **CSS id's and class names** should be prefixed according to the name of the module they support. Examples: spa.css defines #spa, .spa-x-clearall spa.shell.css defines #spa-shell-header, #spa-shell-footer, .spa-shell-main
- **Use our application prefix for all classes and id's** to avoid unintended interaction with third-party modules
- **Use <namespace>-x-<descriptor>** for state-indicator and other shared class names Examples might include **spa-x-select** and **spa-x-disabled** and defined in the **spa.css** file.

Validating code

- Always test code with jslint -jslint <filename> and install the jslint commit hook for git
- Always use module_template.js which contains our jslint settings

Dust.js



- **Place templates in modules under configMap.template_map**. Later we may place these in separate dust files (see below)
- **Please namespace template keys**, for example template_map['sl.ibm.array_row'].
- **Use 'error' and 'dust_html'** as the arguments for dust render callback function
- **Use the sl.utill method** to compile dust templates. This should be placed in initModule().
- **Prefix all dust context object with dust_**
- **Later:** When we need localization, we might place dust templates in separate files that use the same namespace and a dust extension (e.g. 'sl.ibm.dust'). However, this may not be necessary
- **Later:** When we need localization, we may place localized strings in a base context object. We think it would be useful to make our local strings uniquely identifiable compared to user input / backend data. To this end, we should like to namespace our label strings, and also use english hints to help when localizing. For example:
  { dust_base_context : { 'p' : { 'pipeline_title' : 'Pipeline Title', … } }