AMA Project Javascript Coding Standards

# Introduction

The purpose of this document is to convey the Javascript coding standards to developers in order to have a more consistent, familiar, maintainable, and extensible code base.

# Source File Basics

[Google Java Style: Source File Basics](https://google.github.io/styleguide/javaguide.html#s2-source-file-basics), although for Java, uses principles that are equally and appropriately applicable to Javascript files as well. The main difference being file types: for Javascript files, the extension should be .js, for CSS files, the extension should be .css, and for HTML files, the extension should be .html. Unless a reason surfaces not to comply with these principles, we should comply with them.

# Source File Structure

The source file structure will depend on the design pattern you utilize with different design patterns being applicable to different cases. All source files should follow these rules:

* Organized as such to pass JSLint
* Minimize, if not completely avoid, use of the global namespace
* Keep files as small as possible but balance with separating concern

# CODE FORMATTING

There is not a good, all purpose formatting tool for Javascript like there is for Java (c.f., Eclipse code formatter). In general we will:

* Follow the Google Javascript Style Formatting: <https://google.github.io/styleguide/javascriptguide.xml>
* Lint everything using jslint (a code quality tool: <http://www.jslint.com/>). If it doesn’t pass jslint, then it shouldn’t be committed. If inconsistency exists between the Google style and jslint, then go with jslint because there is tool support for ensuring code passes jslint, which means we can bring tools to bear on keeping the code consistent. For example, the IDE I use – WebStorm – can be configured (and I have it configured) to automatically check all Javascript files according to jslint and advise/alert when there are discrepancies.
* JSLint configuration settings are stored in the project’s git repo; note that the entire code base does not currently pass JSLint

# Naming conventions

Follow Google’s recommendations: [Google Javascript Style: Naming](https://google.github.io/styleguide/javascriptguide.xml#Naming).

# Documentation

There is no known Javascript documenting system that works well with modern Javascript design patterns (e.g., JSDoc has to be incredibly verbose to work with the module design pattern).

# Programming Practices

Note: These are general principles. If you lint your code using jslint, then that takes care of the specifics.

## Avoid global namespace

Functions and variables should not be declared in the global namespace unless absolutely necessary. The only known exceptions we should have:

* Variables for Underscore templates (underscore requires variables to be in the global space)
* Variables for Javascript data structures, such as Editor, RSuiteApiService, etc.

## Prefer function expressions over function declarations

Because Javascript functions are first class citizens, they can be declared as the value of a variable. This is the preferred means as it makes the code clearer and mitigates unintentional errors. For example, function declarations are “hoisted” by the Javascript interpreter, meaning that you can actually use a declared function before declaring it because the interpreter will “hoist” all declarations to the start of the scope. Function expressions are not hoisted. Furthermore, jslint requires function expressions.

**Use consistent, modern design patterns**

The editor code uses two main design patterns:

* The Module Singleton Design pattern, which essentially emulates Java’s class structure by having public and private members and functions
* What I call the Helper pattern, see RsiDateUtils.js as an example. This more closely emulates Java’s static utility class pattern.

Note: It is fine to use Immediately-Invoked Function Expressions to avoid polluting the global namespace.

For other patterns to consider, this is a good resource:

https://addyosmani.com/resources/essentialjsdesignpatterns/book/

## Follow Common Good Coding Practices

Such as:

* Code for reusability
* Keep code as loosely coupled as possible
* Balance small performance improvements with upgrade ease: One of the ways performance can be improved with CKEditor is to use the CKBuilder tool ([www.ckbuilder.com)](http://www.ckbuilder.com)) to have your core CKEditor files actually embed your plugin files. The downside to doing this is that in order to upgrade a plugin, you also have to upgrade CKEditor core, getting a new combined version out of the CKBuilder tool. Currently, it only uses the CKEditor standard edition (generated from CKBuilder), which embeds several common plugins. Other plugins, such as Track Changes, Footnotes, and Widgets, are installed as traditional plugins, which allow for them to be more easily upgraded. This also allows easier upgrade of CKEditor.