Forerunner JSDoc Guidelines

Author: Jon Totton, July 22, 2013

# Summary

This guideline is being written to form the basis to standardize the way we create our public documentation for our SDK. It will:

* Give the pertinent links to the JSDoc and associated documentation
* Cover how to use JSDoc
* Defined a proposed set of Guidelines
* Provide concrete examples specific to the forerunner SDK code base
* Defines some next steps we should take

# Links

|  |  |
| --- | --- |
| **Description** | **Link** |
| JSDoc user documentation | http://usejsdoc.org/ |
| github / download site | https://github.com/jsdoc3/jsdoc |
| Markdown | http://en.wikipedia.org/wiki/Markdown |

# How to Use JSDoc

The JSDoc utility automatically generates HTML[[1]](#footnote-1) documentation based upon specifically formed comments in JavaScript files. JSDoc is documented [here](http://usejsdoc.org/). The JSDoc utility / project files are checked into our tree and they live in the …\GitHub\Forerunner\build\tools\JSDoc3 folder. You therefore do not need to install JSDoc.

You run JSDoc from the new script file:

…\ReportManagerMVC\ReportManager\JSdoc\buildDocs.cmd

When buildDocs finishes you can see the generated documentation by opening the file:

ReportManagerMVC\ReportManager\JSdoc\Docs\index.html

JSDoc will run based upon the configuration file conf.json which is in the same folder as buildDocs.cmd. Currently the conf.json file instructs JSDoc to look for all .js files under the following folders:

"../Forerunner/Common/js/",

"../Forerunner/ReportExplorer/js/",

"../Forerunner/ReportViewer/js/"

In this way, any / all JavaScript files add to these folders or their children will be documented.

# Guidelines

* Add JSDoc tags to all public methods and data
* Do NOT add JSDoc tags to private methods and data
* Always generate your documentation and make sure it describes everything a developer would need to know before checking in changes
  + Writing documentation as you go is much easier than trying to go back and add documentation later
* Add Examples wherever needed

# Examples

JSDoc has a rich set of tags. These tags are documented [here](http://usejsdoc.org/#JSDoc3_Tag_Dictionary) and form the basis for the documentation JSDoc generates. Some of the coding conventions we have chosen to use such as widgets require a specific conventions / usage of JSDoc tags. These conventions will be documented here.

Since widgets are defined as children of jquery and our widgets are defined under the forerunner namespace, we need to have the jquery alias (“$”), as well as our forerunner namespace defined. These namespaces are defined at the top of the …\Forerunner\Common\js\forerunner.js file. Since these namespaces are defined we can properly define our widgets as follows:

$.forerunner.widgetname

## Widgets

/\*\*

\* Toobar widget used by the reportViewer

\*

\* @namespace **$.forerunner.toolbar**

\* **@prop** {object} options - The options for toolbar

\* @prop {Object} options.$reportViewer - The report viewer widget

\* @prop {String} options.toolClass - The top level class for this tool (E.g., fr-toolbar)

\* **@example**

\* $("#toolbarId").toolbar({

\* $reportViewer: $viewer,

\* toolClass: "fr-toolbar"

\* });

\*

\* Note:

\* Toolbar can be extended by calling the addTools method defined by {@link $.forerunner.toolBase}

\*/

$.widget(widgets.getFullname(widgets.toolbar), $.forerunner.toolBase, /\*\* **@lends** $.forerunner.toolbar \*/ {

}); // $.widget

All JSDoc comment blocks begin with /\*\*. The important tags / conventions for documenting widgets are marked in **red** above.

**$.forerunner.toolbar**

You must explicitly give the namespace name as JSDoc cannot get it from the widget declaration.

**@prop**

Use property tags so that the options associated with the widget are documented on the same page as the widget.

**@example**

Use examples liberally. Examples are the most useful kind of documentation.

**@lends**

Since widget declarations are not simple class, variable or namespaces (see the JSDoc documentation [here](http://usejsdoc.org/)). The @lends tag helps JSDoc determine what is contained inside of the widget definition object that follows.

## Widget Methods

/\*\*

\* Make all tools hidden

\* @function **$.forerunner.toolBase#hideTools**

\*/

hideTools: function (){

},

**$.forerunner.toolBase#hideTools**

Explicitly use the @function tag along with a fully qualified [name path](http://usejsdoc.org/about-namepaths.html). This will properly document the function as an instance method and not a static method. E.g. if you removed the function tag line above the documentation would list the function as static.

## Enums

/\*\*

\* Tool types used by the Toolbase widget **{@link $.forerunner.toolBase}**

\*

\* @readonly

\* **@enum** {String}

\*/

toolTypes: {

button: "button",

input: "input",

textButton: "textbutton",

plainText: "plaintext",

containerItem: "containeritem",

toolGroup: "toolgroup"

},

**{@link $.forerunner.toolBase}**

Inline tags such as the link tag above are very helpful for developers. Use then freely.

**@enum**

The @enum tag provides a very easy mechanism to document all the enum choices with a single tag.

## Namespaces, Constants and functions

/\*\*

\* Defines the event name constant used to trigger the event as well as the fully qualified event name

\* function (widget + event, lowercase). The fully qualified name is used to bind to the event.

\*

\* @namespace

\*/

events: {

/\*\* @constant \*/

actionStarted: "actionstarted",

/\*\* widget + event, lowercase \*/

toolPaneActionStarted: function () {},

You can see in the example above JSDoc normally can figure out what the name of the namespace is so you only need to use the @namespace tag. Likewise using the @constant tag above is very straight forward. Documenting the function above is the easiest as all you need to do is to put in the comment. Note that the constant and function above are documented as static. This is correct because there is no instance created for events.

# Next Steps

* Everyone should learn the basic JSDoc tags. You can look at forerunner.js, toobase.js and toolbar.js for some good examples.
* We should wright a readme.md file and include it in our documentation. The help topic is [here](http://usejsdoc.org/about-including-readme.html).
* We could add [Tutorials](http://usejsdoc.org/about-tutorials.html) to our documentation.
* We could stylize the output by creating out own template.

1. It can also output xml and json based upon a command line switch; we will use the html option. [↑](#footnote-ref-1)