JSDOC Usage (x2o)

Intended use of this document

This document has been authored with the purpose of helping use JSDOC to document common Javascript code. This document does not cover jsdoc definitions. For more about JSDOC definitions and available keywords, see https://git.barco.com/projects/CF/repos/documentation-demo/browse.

Refer to each JS file in the src folder of this project to see how documentation is done for different javascript elements.

JSDoc supports Node.js 4.2.0 and later. You can install JSDoc globally or in your project's node_modules folder.

To install the latest version on npm globally (may require sudo; learn how to fix this):

```
npm install -g jsdoc
```

To install the latest version on npm locally and save it in your package's package.json file:

```
npm install --save-dev jsdoc
```

Note: By default, npm adds your package using the caret operator in

front of the version number (for example, ^3.5.2). We recommend using the tilde operator instead (for example, ~3.5.2), which limits updates to the most recent patch-level version. See this Stack Overflow answer for more information about the caret and tilde operators.

To install the latest development version locally, without updating your project's package.json file:

```
npm install git+https://github.com/jsdoc3/jsdoc.git
```

If you installed JSDoc locally, the JSDoc command-line tool is available in ./node_modules/.bin. To generate documentation for the file yourJavaScriptFile.js:

```
./node_modules/.bin/jsdoc yourJavaScriptFile.js
```

Or if you installed JSDoc globally, simply run the jsdoc command:

```
jsdoc yourJavaScriptFile.js
```

By default, the generated documentation is saved in a directory named out. You can use the --destination (-d) option to specify another directory.

Run jsdoc --help for a complete list of command-line options.

Using Gulp or Grunt

JSOC configurable parameters can be specified in a separate JSON file. Parameters can be defined in this document and called via the gulp or grunt task runner.
Refer to the Gulpfile.js and/or Gruntfile.js files to learn how to initiate JSDOC via these tools.

- To execute the gulp task : `gulp doc`
- To execute the grunt task : `grunt doc`

NOTE: Both gulp and grunt can be configured to execute a watcher that runs the doc script each time a change is encountered.

Integration into an existing project

To integrate the features of this demo project into an existing project, follow the steps below.

Assuming there is an existing package.json file

- Copy the script and devDependendies into your existing package.json file. If these nodes do not exist in your package.json file,
  you will have to create them.

- Run the following commands from terminal or CLI `npm install`

- Copy the Gulpfile or Gruntfile into your project root. Having

run npm install, the correct versions of Gulp and/or Grunt will have
already been installed.

- If necessary, change the path configuration in jsdoc.json

- Run the Gulp or Grunt taskrunner from terminal or CLI to generate the documentation.

JSDoc is all about documenting entities (functions, methods, constructors, etc.). That is achieved via comments that precede the entities and start with /**.

Syntax

Let’s review the comment shown at the beginning:

```
/**
 * Repeat <tt>str</tt> several times.
 * @param {string} str The string to repeat.
 * @param {number} [times=1] How many times to repeat the
 string.
 * @returns {string}
 */
```

This demonstrates some of the JSDoc syntax, which consists of the following pieces:

JSDoc comment

This is a JavaScript block comment whose first character is an asterisk. This creates the illusion that the token /** starts such a comment.

Tags

You structure comments by starting lines with tags, keywords that are prefixed with an @ symbol. @param is an example in the preceding code.

HTML

You can freely use HTML in JSDoc comments. For example, <tt> displays a word in a monospaced font.

Type annotations

You can document the type of an entity via a type name in braces. Variations include:

- Single type: @param {string} name
- Multiple types: @param {string|number} idCode
- Arrays of a type: @param {string[]} names

Namepaths

Inside JSDoc comments, so-called namepaths are used to refer to entities. The syntax of such paths is as follows:

- myFunction
- MyClass
- MyClass.staticMember

- MyClass#instanceMember

Classes are usually (implemented by) constructors. Static members are, for example, properties of constructors. JSDoc has a broad definition of instance member. It means everything that can be accessed via an instance. Therefore, instance members include instance properties and prototype properties.

Naming Types

The types of entities are either primitive types or classes. The names of the former always start with lowercase letters; the names of the latter always start with uppercase letters. In other words, the type names of primitives are boolean, number, and string, just like the results returned by the typeof operator. That way, you cannot confuse strings (primitives) with instances of the constructor String (objects).

Basic Tags

Following are the basic metadata tags:

@fileOverview description

Marks a JSDoc comment that describes the whole file. For example:

```
/**
 * @fileOverview Various tool functions.
 * @author <a href="mailto:jd@example.com">John Doe</a>
 * @version 3.1.2
 */
```

@author

Refers to who has written the entity being documented.

@deprecated

Indicates that the entity is not supported anymore. It is a good practice to document what to use instead.

@example

Contains a code example illustrating how the given entity should be used:

```
/**
 * @example
 * var str = 'abc';
 * console.log(repeat(str, 3)); // abcabcabc
 */
```

Basic tags for linking are as follows:

@see

Points to a related resource:

```
/**
 * @see MyConstructor#myMethod
 * @see The <a href="http://example.com">Example Project<
/a>.
```

```
 */
```

{@link ...}

Works like @see, but can be used inside other tags.

@requires resourceDescription

Indicates a resource that the documented entity needs. The resource description is either a namepath or a natural language description.

Versioning tags include the following:

@version versionNumber

Indicates the version of the documented entity. For example:
`@version 10.3.1`

@since versionNumber

Indicates since which version the documented entity has been available. For example:
`@since 10.2.0`

Classes And Constructors

JSDoc distinguishes between classes and constructors. The former concept is more like a type, while a constructor is one way of implementing a class.
JavaScript’s built-in means for defining classes are limited, which is why there are many APIs that help with this task.

These APIs differ, often radically, so you have to help JSDoc with figuring out what is going on. The following tags let you do that:

@constructor

Marks a function as a constructor.

@class

Marks a variable or a function as a class. In the latter case, @class is a synonym for @constructor.

@constructs

Records that a method sets up the instance data. If such a method exists, the class is documented there.

@lends namePath

Specifies to which class the following object literal contributes. There are two ways of contributing.

- @lends Person#: The object literal contributes instance members to Person.
- @lends Person: The object literal contributes static members to Person.

@memberof parentNamePath

The documented entity is a member of the specified object. @lends MyClass#, applied to an object literal, has the same effect as marking each
property of that literal with @memberof MyClass#.

The most common ways of defining a class are: via a constructor function, via an object literal, and via an object literal that has an @constructs method.

Functions and Methods

For functions and methods, you can document parameters, return values, and exceptions they may throw:

- @param {paramType} paramName description

Describes the parameter whose name is paramName. Type and description are optional. Here are some examples:

- @param str The string to repeat.
- @param {string} str
- @param {string} str The string to repeat.

Advanced features:

Optional parameter:

- @param {number} [times] The number of times is optional.

Optional parameter with default value:

- @param {number} [times=1] The number of times is optional.
- @returns {returnType} description — Describes the return value of the function or method. Either type or description can be omitted.

- @throws {exceptionType} description — Describes an exception that might be thrown during the execution of the function or method. Either type or description can be omitted.

Inline Type Information (“Inline Doc Comments”)

There are two ways of providing type information for parameters and return values. First, you can add type annotations to @param and @returns:

```
/**
 * @param {String} name
 * @returns {Object}
 */
function getPerson(name) {
}
```

Second, you can inline the type information:

```
function getPerson(/**String*/ name) /**Object*/ {
}
```