

# JSDoc Tags

Command Name	Description
@param @argument	Describes a function parameter by specifying the parameter name and description.
@return @returns	Describes the return value of the function.
@author	Indicates the author of the code.
@deprecated	Indicates that a function is deprecated and may be removed from future versions of the code. You should avoid using this particular piece of code.
@see	Creates an HTML link to the description of the specified class.
@version	Specifies the release version.
@requires	Creates an HTML link to the specified class that is required for this class.
@throws @exception	Describes the type of exception that a function may throw.
{@link}	Creates an HTML link to the specified class. This is similar to @see but can be embedded inside comment text.
@author	Indicates the author of the code.
@fileoverview	A special tag that when used in the first block of documentation in a file, specifies that the rest of the documentation block will be used to provide an overview of the file.
@class	Provides information about the class and is used in the constructor's documentation.
@constructor	Identifies a function as the constructor for a class.
@type	Indicates the return type of a function.
@extends	Indicates that a class subclasses another class. JSDoc can often detect this information on its own, but in some instances using this tag is required.
@private	Signifies that a class or function is private. Private classes and functions will not be available in the HTML documentation unless JSDoc is run with the --private command-line option.
@final	Indicates that a value is a constant value. Keep in mind that JavaScript can't actually enforce a value as being constant.
@ignore	JSDoc ignores functions that are labeled with this tag.

```
C:\Outils\JSDoc-1.9.9.2>perl jsdoc.pl -h
Usage: jsdoc [OPTIONS] <js sourcefiles and/or directories>+

-h ! --help          Show this message and exit
-r ! --recursive    Recurse through given directories
-p ! --private      Show private methods and fields
-d ! --directory    Specify output directory (defaults to js_docs_out)
-q ! --quiet        Suppress normal output

--page-footer       Specify (html) footer string that will be added to
                    all docs
--project-name      Specify project name for that will be added to docs
--logo              Specify a path to a logo to be used in the docs
--project-summary   Specify a path to a text file that contains an
                    overview summary of the project

--no-sources        Don't include the source code view
--extensions        Provide a comma-separated list of file extensions
                    to be considered as JavaScript source files

--package-naming    Use package-style naming (i.e. keep directory names
                    in the file path). This is useful if you have multiple
                    files with the same name, but in different directories.
                    This option is only useful if --recursive is also used.

--globals-name      Specify a 'class name' under which all unattached
                    methods will be classified. The defaults to GLOBALS

--format            Set the output format. The options are html, xml
                    and xml, defaulting to html. The others are currently
                    alpha software.

--template-dir      Provide another directory containing HTML templates

--no-lexical-privates Ignore "private" variables and functions that are
                    lexically defined within constructors
```

```
/**
 * @fileoverview This file is an example of how JSDoc can be used to document
 * JavaScript.
 *
 *
 * @author Ryan Asleson
 * @version 1.0
 */

/**
 * Construct a new Person class.
 * @class This class represents an instance of a Person.
 * @constructor
 * @param {String} name The name of the Person.
 * @return A new instance of a Person.
 */
function Person(name) {
  /**
   * The Person's name
   * @type String
   */
  this.name = name;
  /**
   * Return the Person's name. This function is assigned in the class
   * constructor rather than using the prototype keyword.
   * @returns The Person's name
   * @type String
   */
  this.getName = function() {
    return name;
  }
}

/**
 * Construct a new Employee class.
 * @extends Person
 * @class This class represents an instance of an Employee.
 * @constructor
 * @return A new instance of a Person.
 */
function Employee(name, title, salary) {
  this.name = name;

  /**
   * The Employee's title
   * @type String
   */
  this.title = title;

  /**
   * The Employee's salary
   * @type int
   */
  this.salary = salary;
}

/* Employee extends Person */
Employee.prototype = new Person();

/**
 * An example of function assignment using the prototype keyword.
 * This method returns a String representation of the Employee's data.
 * @returns The Employee's name, title, and salary
 * @type String
 */
Employee.prototype.getDescription = function() {
  return this.name + " - "
    + this.title + " - "
    + "$" + this.salary;
}
```