PHP Documentation Standards

- What Should Be Documented
  - Documenting Tips
    - Language
    - Grammar
    - Miscellaneous
  - Formatting Guidelines
    - General
    - Summary
    - Description
    - @since Section (Changelogs)
    - Other Descriptions
    - Line wrapping
- DocBlock Formatting
  - 1. Functions & Class Methods
    - 1.1 Parameters That Are Arrays
    - 1.2 Deprecated Functions
  - 2. Classes
    - 2.1 Class Members
      - 2.1.1 Properties
      - 2.1.2 Constants
  - 3. Requires and Includes
  - 4. Hooks (Actions and Filters)
    - 4.1 Duplicate Hooks
  - 5. Inline Comments
    - 5.1 Single line comments
    - 5.2 Multi-line comments

- 6. File Headers
- 7. Constants
- PHPDoc Tags
  - Deprecated Tags
  - Other Tags
- Resources

WordPress uses a customized documentation schema that draws inspiration from PHPDoc, an evolving standard for providing documentation to PHP code, which is maintained by phpDocumentor.

What Should Be Documented

PHP documentation in WordPress mostly takes the form of either formatted blocks of documentation or inline comments.

The following is a list of what should be documented in WordPress files:

- Functions and class methods
- Classes
- Class members (including properties and constants)
- Requires and includes
- Hooks (actions and filters)
- Inline comments
- File headers
- Constants

Documenting Tips

Language

Summaries should be clear, simple, and brief. Avoid describing “why” an element exists, rather, focus on documenting “what” and “when” it does something.

A function, hook, class, or method is a *third-person singular* element, meaning that *third-person singular verbs* should be used to describe what each does.

Need help remembering how to conjugate for third-person singular verbs? Imagine prefixing the function, hook, class, or method summary with “It”:

- *Good*: “(It) Does something.”
- *Bad*: “(It) Do something.”

Summary examples:

- **Functions**: *What* does the function do?
  - Good: *Displays the last modified date for a post.*
  - Bad: *Display the date on which the post was last modified.*
- **Filters**: *What* is being filtered?
  - Good: *Filters the post content.*
  - Bad: *Lets you edit the post content that is output in the post template.*
- **Actions:** *When* does an action fire?
  - Good: *Fires after most of core is loaded, and the user is authenticated.*
  - Bad: *Allows you to register custom post types, custom taxonomies, and other general housekeeping tasks after a lot of WordPress core has loaded.*

Grammar

Descriptive elements should be written as complete sentences. The one exception to this standard is file header summaries, which are intended as file “titles” more than sentences.

The serial (Oxford) comma should be used when listing elements in summaries, descriptions, and parameter or return descriptions.

Miscellaneous

`@since`: The recommended tool to use when searching for the version something was added to WordPress is `svn blame`. An additional resource for older hooks is the WordPress Hooks Database.

If the version number cannot be determined after using these tools, use `@since Unknown`.

Anything ported over from WPMU should use `@since MU (3.0.0)`. Existing `@since MU (3.0.0)` tags should not be changed.

Code Refactoring: It is permissible to space out the specific action or filter lines being documented to meet the coding standards, but do not refactor other code in the file.

Formatting Guidelines

WordPress' inline documentation standards for PHP are specifically tailored for optimum output on the official Code Reference. As such, following the standards in core and formatting as described below are *extremely* important to ensure expected output.\

General

DocBlocks should directly precede the hook, action, function, method, or class line. There should not be any opening/closing tags or other things between the DocBlock and the declarations to prevent the parser becoming confused.

Summary

No HTML markup or Markdown of any kind should be used in the summary. If the text refers to an HTML element or tag, then it should be written as "image tag" or "img" element, not "`<img>`". For example:

- Good: *Fires when printing the link tag in the header.*
- Bad: *Fires when printing the* `<link>` *tag in the header.*

Inline PHPDoc tags may be used.

Description

HTML markup should never be used outside of code examples, though Markdown can be used, as needed, in the description.

1. Lists:
   Use a hyphen (-) to create an unordered list, with a blank line before and after.

   ```
   * Description which includes an unordered list:
   *
   * - This is item 1.
   * - This is item 2.
   * - This is item 3.
   *
   * The description continues on ...
   ```

   Use numbers to create an ordered list, with a blank line before and after.

   ```
   * Description which includes an ordered list:
   *
   * 1. This is item 1.
   ```

```
* 2. This is item 2.
* 3. This is item 3.
*
* The description continues on ...
```

2. Code samples would be created by indenting every line of the code by 4 spaces, with a blank line before and after. Blank lines in code samples also need to be indented by four spaces. Note that examples added in this way will be output in `<pre>` tags and *not* syntax-highlighted.

```
* Description including a code sample:
*
*    $status = array(
*        'draft'   => __( 'Draft' ),
*        'pending' => __( 'Pending Review' ),
*        'private' => __( 'Private' ),
*        'publish' => __( 'Published' )
*    );
*
* The description continues on ...
```

3. Links in the form of URLs, such as related Trac tickets or other documentation, should be added in the appropriate place in the DocBlock using the `@link` tag:

```
* Description text.
*
* @link https://core.trac.wordpress.org/ticket/20000
```

`@since` Section (Changelogs)

Every function, hook, class, and method should have a corresponding `@since` version associated with it (more on that below).

No HTML should be used in the descriptions for `@since` tags, though limited Markdown can be used as necessary, such as for adding backticks around variables, arguments, or parameter names, e.g. `$variable`.

Versions should be expressed in the 3-digit `x.x.x` style:

```
* @since 4.4.0
```

If significant changes have been made to a function, hook, class, or method, additional `@since` tags, versions, and descriptions should be added to provide a changelog for that function.

“Significant changes” include but are not limited to:

- Adding new arguments or parameters.
- Required arguments becoming optional.
- Changing default/expected behaviors.
- Functions or methods becoming wrappers for new APIs.
- Parameters which have been renamed (once PHP 8.0 support has been announced).

PHPDoc supports multiple `@since` versions in DocBlocks for this explicit reason. When adding changelog entries to the `@since` block, a version should be cited, and a description should be added in sentence case and form and end with a period:

```
 * @since 3.0.0
 * @since 3.8.0 Added the `post__in` argument.
 * @since 4.1.0 The `$force` parameter is now optional.
```

Other Descriptions

`@param`, `@type`, `@return`: No HTML should be used in the descriptions for these tags, though limited Markdown can be used as necessary, such as for adding backticks around variables, e.g. `$variable` .

- Inline `@see` tags can also be used to auto-link hooks in core:
  - Hooks, e.g. `'save_post'`
  - Dynamic hooks, e.g. `'$old_status_to_$new_status'` (Note that any extra curly braces have been removed inside the quotes)
- Default or available values should use single quotes, e.g. ‘draft’. Translatable strings should be identified as such in the description.
- HTML elements and tags should be written as “audio element” or “link tag”.

Line wrapping

DocBlock text should wrap to the next line after 80 characters of text. If the DocBlock itself is indented on the left 20 character positions, the wrap could occur at position 100, but should not extend beyond a total of 120 characters wide.

DocBlock Formatting

The examples provided in each section below show the expected DocBlock content and tags, as well as the exact formatting. Use spaces inside the DocBlock, not tabs, and ensure that items in each tag group are aligned according to the examples.

1. Functions & Class Methods

Functions and class methods should be formatted as follows:

- **Summary**: A brief, one sentence explanation of the purpose of the function spanning a maximum of two lines. Use a period at the end.
- **Description**: A supplement to the summary, providing a more detailed description. Use a period at the end of sentences.
- `@ignore`: Used when an element is meant only for internal use and should be skipped from parsing.
- `@since x.x.x`: Should always be 3-digit (e.g. `@since 3.9.0`). Exception is `@since MU (3.0.0)`.
- `@access`: Only used for core-only functions or classes implementing “private” core APIs. If the element is private it will be output with a message stating its intention for internal use.
- `@see`: Reference to a function, method, or class that is heavily-relied on within. See the note above about inline `@see` tags for expected formatting.
- `@link`: URL that provides more information. This should never be used to reference another function, hook, class, or method, see `@see`.
- `@global`: List PHP globals that are used within the function or method, with an optional description of the global. If multiple globals are listed, they should be aligned by type, variable, and description, with each other as a group.
- `@param`: Note if the parameter is *Optional* before the description, and include a period at the end. The description should mention accepted values as well as the default. For example: *Optional. This value does something. Accepts ‘post’, ‘term’, or empty. Default empty.*
- `@return`: Should contain all possible return types and a description for each. Use a period at the end. Note: `@return void` should not be used outside the default bundled themes and the PHP compatibility shims included in WordPress Core.

```
/**
 * Summary.
 *
 * Description.
 *
 * @since x.x.x
 *
 * @see Function/method/class relied on
 * @link URL
 * @global type $varname Description.
 * @global type $varname Description.
 *
 * @param type $var Description.
```

```
 * @param type $var Optional. Description. Default.
 * @return type Description.
 */
```

1.1 Parameters That Are Arrays

Parameters that are an array of arguments should be documented in the “originating” function only, and cross-referenced via an `@see` tag in corresponding DocBlocks.

Array values should be documented using WordPress’ flavor of hash notation style similar to how Hooks can be documented, each array value beginning with the `@type` tag, and taking the form of:

```
*     @type type $key Description. Default 'value'. Accepts 'value', 'value'.
*                     (aligned with Description, if wraps to a new line)
```

An example of an “originating” function and re-use of an argument array is `wp_remote_request|post|get|head()`.

```
/**
 * Summary.
 *
 * Description.
 *
 * @since x.x.x
 *
 * @param type  $var Description.
 * @param array $args {
 *     Optional. An array of arguments.
 *
 *     @type type $key Description. Default 'value'. Accepts 'value', 'value'.
 *                     (aligned with Description, if wraps to a new line)
 *     @type type $key Description.
 * }
 * @param type  $var Description.
 * @return type Description.
 */
```

In most cases, there is no need to mark individual arguments in a hash notation as *optional*, as the entire array is usually optional. Specifying “Optional.” in the hash notation description should suffice. In the case where the array is NOT optional, individual key/value pairs may be optional and should be marked as such as necessary.

1.2 Deprecated Functions

If the function is deprecated and should not be used any longer, the `@deprecated` tag, along with the version and description of what to use instead, should be added. Note the additional use of an `@see` tag – the Code Reference uses this information to attempt to link to the replacement function.

```
/**
 * Summary.
 *
 * Description.
 *
 * @since x.x.x
 * @deprecated x.x.x Use new_function_name()
 * @see new_function_name()
 *
 * @param type $var Optional. Description.
 * @param type $var Description.
 * @return type Description.
 */
```

2. Classes

Class DocBlocks should be formatted as follows:

- **Summary**: A brief, one sentence explanation of the **purpose** of the class spanning a maximum of two lines. Use a period at the end.
- **Description**: A supplement to the summary, providing a more detailed description. Use a period at the end.
- **`@since x.x.x`**: Should always be 3-digit (e.g. `@since 3.9.0`). Exception is `@since MU (3.0.0)`.

```
/**
 * Summary.
 *
 * Description.
 *
 * @since x.x.x
 */
```

If documenting a sub-class, it's also helpful to include an `@see` tag reference to the super class:

```
/**
 * Summary.
 *
 * Description.
 *
 * @since x.x.x
 *
 * @see Super_Class
 */
```

2.1 Class Members

2.1.1 Properties

Class properties should be formatted as follows:

- **Summary**: Use a period at the end.
- `@since x.x.x`: Should always be 3-digit (e.g. `@since 3.9.0`). Exception is `@since MU (3.0.0)`.
- `@var`: Formatted the same way as `@param`, though the description may be omitted.

```
/**
 * Summary.
 *
 * @since x.x.x
 * @var type $var Description.
 */
```

2.1.2 Constants

Summary: Use a period at the end.

`@since x.x.x`: Should always be 3-digit (e.g. `@since 3.9.0`). Exception is `@since MU (3.0.0)`.

`@var`: Formatted the same way as `@param`, though the description may be omitted.

```
/**
 * Summary.
 *
 * @since x.x.x
 * @var type $var Description.
 */
```

```
const NAME = value;
```

3. Requires and Includes

Files required or included should be documented with a summary description DocBlock. Optionally, this may apply to inline `get_template_part()` calls as needed for clarity.

```
/**
 * Summary.
 */
require_once( ABSPATH . WPINC . '/filename.php' );
```

4. Hooks (Actions and Filters)

Both action and filter hooks should be documented on the line immediately preceding the call to `do_action()` or `do_action_ref_array()`, or `apply_filters()` or `apply_filters_ref_array()`, and formatted as follows:

- **Summary**: A brief, one line explanation of the purpose of the hook. Use a period at the end.
- **Description**: A supplemental description to the summary, if warranted.
- `@ignore`: Used when a hook is meant only for internal use and should be skipped from parsing.
- `@since x.x.x`: Should always be 3-digit (e.g. `@since 3.9.0`). Exception is `@since MU (3.0.0)`.
- `@param`: If the parameter is an array of arguments, document each argument using a hash notation (described above in the *Parameters That Are Arrays* section), and include a period at the end of each line.

Note that `@return` is *not* used for hook documentation, because action hooks return nothing, and filter hooks always return their first parameter.

```
/**
 * Summary.
 *
 * Description.
 *
 * @since x.x.x
 *
 * @param type  $var Description.
 * @param array $args {
```

```
 *     Short description about this hash.
 *
 *     @type type $var Description.
 *     @type type $var Description.
 * }
 * @param type  $var Description.
 */
```

If a hook is in the middle of a block of HTML or a long conditional, the DocBlock should be placed on the line immediately before the start of the HTML block or conditional, even if it means forcing line-breaks/PHP tags in a continuous line of HTML.

Tools to use when searching for the version a hook was added are `svn blame`, or the WordPress Hooks Database for older hooks. If, after using these tools, the version number cannot be determined, use `@since Unknown`.

4.1 Duplicate Hooks

Occasionally, hooks will be used multiple times in the same or separate core files. In these cases, rather than list the entire DocBlock every time, only the first-added or most logically-placed version of an action or filter will be fully documented. Subsequent versions should have a single-line comment.

For actions:

```
/** This action is documented in path/to/filename.php */
```

For filters:

```
/** This filter is documented in path/to/filename.php */
```

To determine which instance should be documented, search for multiples of the same hook tag, then use `svn blame` to find the first use of the hook in terms of the earliest revision. If multiple instances of the hook were added in the same release, document the one most logically-placed as the “primary”.

5. Inline Comments

Inline comments inside methods and functions should be formatted as follows:

5.1 Single line comments

```
// Allow plugins to filter an array.
```

5.2 Multi-line comments

```
/*
 * This is a comment that is long enough to warrant being stretched over
 * the span of multiple lines. You'll notice this follows basically
 * the same format as the PHPDoc wrapping and comment block style.
 */
```

Important note: Multi-line comments must not begin with `/**` (double asterisk) as the parser might mistake it for a DocBlock. Use `/*` (single asterisk) instead.

6. File Headers

The file header DocBlock is used to give an overview of what is contained in the file.

Whenever possible, **all** WordPress files should contain a header DocBlock, regardless of the file's contents – this includes files containing classes.

```
/**
 * Summary (no period for file headers)
 *
 * Description. (use period)
 *
 * @link URL
 *
 * @package WordPress
 * @subpackage Component
 * @since x.x.x (when the file was introduced)
 */
```

The *Summary* section is meant to serve as a succinct description of **what** specific purpose the file serves.

Examples:

- Good: *"Widgets API: WP_Widget class"*
- Bad: *"Core widgets class"*

The *Description* section can be used to better explain overview information for the file such as how the particular file fits into the overall makeup of an API or component.

Examples:

- Good: *"The Widgets API is comprised of the WP_Widget and WP_Widget_Factory classes in addition to a variety of top-level functionality that implements the Widgets and related sidebar APIs. WordPress registers a number of common widgets by default."*

7. Constants

The constant DocBlock is used to give a description of the constant for better use and understanding.

Constants should be formatted as follows:

- **Summary**: Use a period at the end.
- `@since x.x.x`: Should always be 3-digit (e.g. `@since 3.9.0`). Exception is `@since MU (3.0.0)`.
- `@var`: Formatted the same way as `@param`. The description is optional.

```
/**
 * Summary.
 *
 * @since x.x.x (if available)
 * @var type $var Description.
 */
```

PHPDoc Tags

Common PHPDoc tags used in WordPress include `@since`, `@see`, `@global` `@param`, and `@return` (see table below for full list).

For the most part, tags are used correctly, but not all the time. For instance, sometimes you'll see an `@link` tag inline, linking to a separate function or method. "Linking" to known classes, methods, or functions is not necessary, as the Code Reference automatically links these elements. For "linking" hooks inline, the proper tag to use is `@see` – see the *Other Descriptions* section.

Tag	Usage	Description

`@access`	private	Only used in limited circumstances, like when visibility modifiers cannot be used in the code, and only when private, such as for core-only functions or core classes implementing “private” APIs. Used directly below the `@since` line in block.
`@deprecated`	version x.x.x Use *replacement function name* instead	What version of WordPress the function/method was deprecated. Use 3-digit version number. Should be accompanied by a matching `@see` tag.
`@global`	datatype $variable description	Document global(s) used in the function/method. For boolean and integer types, use `bool` and `int`, respectively.
`@internal`	information string	Typically used wrapped in `{}` for adding notes for internal use only.
`@ignore`	(standalone)	Used to skip parsing of the entire element.
`@link`	URL	Link to additional information for the function/method. For an external script/library, links to source. Not to be used for related functions/methods; use `@see` instead.
`@method`	returntype description	Shows a “magic” method found inside the class.
`@package`	packagename	Specifies package that all functions, includes, and defines in the file belong to. Found in DocBlock at top of the file. For core (and bundled themes), this is always **WordPress**.
`@param`	datatype $variable description	Function/method parameter of the format: parameter type, variable name, description, default behavior. For boolean and integer types, use `bool` and `int`, respectively.
`@return`	datatype description	Document the return value of functions or methods. `@return void` should not be used outside of the default bundled themes. For boolean and integer types, use `bool` and `int` , respectively.
`@see`	elementname	References another function/method/class the function/method relies on. Should only be used inline for “linking” hooks.

`@since`	version x.x.x	Documents release version function/method was added. Use 3-digit version number – this is to aid with version searches, and for use when comparing versions in code. Exception is `@since MU (3.0.0)`.
`@static`	(standalone)	Note: This tag has been used in the past, but should no longer be used. Just using the static keyword in your code is enough for phpDocumentor on PHP5+ to recognize static variables and methods, and PhpDocumentor will mark them as static.
`@staticvar`	datatype $variable description	Note: This tag has been used in the past, but should no longer be used. Document a static variable's use in a function/method. For boolean and integer types, use `bool` and `int`, respectively.
`@subpackage`	subpackagename	For page-level DocBlock, specifies the Component that all functions and defines in file belong to. For class-level DocBlock, specifies the subpackage/component the class belongs to.
`@todo`	information string	Documents planned changes to an element that have not been implemented.
`@type`	datatype description for an argument array value	Used to denote argument array value types. See the **Hooks** or **Parameters That Are Arrays** sections for example syntax.
`@uses`	class::methodname() / class::$variablename / functionname()	Note: This tag has been used in the past, but should no longer be used. References a key function/method used. May include a short description.
`@var`	datatype description	Data type for a class variable and short description. Callbacks are marked callback.

PHPDoc tags work with some text editors/IDEs to display more information about a piece of code. It is useful to developers using those editors to understand what the purpose is, and where they would use it in their code. PhpStorm and Netbeans already support PHPDoc.

The following text editors/IDEs have extensions/bundles you can install that will help you auto-create DocBlocks:

- Notepad++: DocIt for Notepad++ (Windows)
- TextMate: php.tmbundle (Mac)
- SublimeText: sublime packages (Windows, Mac, Linux)

Note: Even with help generating DocBlocks, most code editors don't do a very thorough job – it's likely you'll need to manually fill in certain areas of any generated DocBlocks.\

Deprecated Tags

> **Preface:** For the time being, and for the sake of consistency, WordPress Core will continue to use `@subpackage` tags – both when writing new DocBlocks, and editing old ones.
>
> Only when the new – external – PSR-5 recommendations are finalized, will across-the-board changes be considered, such as deprecating certain tags.

As proposed in the new PSR-5 recommendations, the following PHPDoc tag should be deprecated:

- `@subpackage` (in favor of a unified package tag: `@package Package\Subpackage`)
- `@static` (no longer needed)
- `@staticvar` (no longer needed)

Other Tags

`@package` Tag in Plugins and Themes (bundled themes excluded)

Third-party plugin and theme authors **must not** use `@package WordPress` in their plugins or themes. The `@package` name for plugins should be the plugin name; for themes, it should be the theme name, spaced with underscores: `Twenty_Fifteen`.

`@author` Tag

It is WordPress' policy not to use the `@author` tag, except in the case of maintaining it in external libraries. We do not want to imply any sort of "ownership" over code that might discourage contribution.

`@copyright` and `@license` Tags

The `@copyright` and `@license` tags are used in external libraries and scripts, and should not be used in WordPress core files.

- `@copyright` is used to specify external script copyrights.
- `@license` is used to specify external script licenses.

Resources

- Wikipedia on PHPDoc
- PEAR Standards
- phpDocumentor
- phpDocumentor Tutorial Tags
- Draft PSR-5 recommendations
- Draft PSR-19 recommendations

□□□ #2
□ □□□ □□□ 2026-10-31 18:41:57 UTC
□ □□□ □□□ 2026-10-31 18:42:00 UTC