NAME

javadoc - generate HTML pages of API documentation from Java source files

SYNOPSIS

`javadoc` [*options*] [*packagenames*] [*sourcefiles*] [`@`*files*]

options Specifies command-line options, separated by spaces. See **Standard `javadoc` Options**, **Extra `javadoc` Options**, **Standard Options for the Standard Doclet**, and **Extra Options for the Standard Doclet**.

packagenames

Specifies names of packages that you want to document, separated by spaces, for example `java.lang java.lang.reflect java.awt`. If you want to also document the subpackages, then use the `-subpackages` option to specify the packages.

By default, `javadoc` looks for the specified packages in the current directory and subdirectories. Use the `-sourcepath` option to specify the list of directories where to look for packages.

sourcefiles

Specifies names of Java source files that you want to document, separated by spaces, for example `Class.java Object.java Button.java`. By default, `javadoc` looks for the specified classes in the current directory. However, you can specify the full path to the class file and use wildcard characters, for example `/home/src/java/awt/Graphics*.java`. You can also specify the path relative to the current directory.

`@`*files* Specifies names of files that contain a list of `javadoc` tool options, package names, and source file names in any order.

DESCRIPTION

The `javadoc` tool parses the declarations and documentation comments in a set of Java source files and produces corresponding HTML pages that describe (by default) the public and protected classes, nested and unnamed classes (but not anonymous inner classes), interfaces, constructors, methods, and fields. You can use the`javadoc` tool to generate the API documentation or the implementation documentation for a set of source files.

You can run the `javadoc` tool on entire packages, individual source files, or both. When documenting entire packages, you can use the `-subpackages` option either to recursively traverse a directory and its subdirectories, or to pass in an explicit list of package names. When you document individual source files, pass in a list of Java source file names.

Conformance

The Standard Doclet does not validate the content of documentation comments for conformance, nor does it attempt to correct any errors in documentation comments. Anyone running javadoc is advised to be aware of the problems that may arise when generating non-conformant output or output containing executable content, such as JavaScript. The Standard Doclet does provide the **DocLint** feature to help developers detect common problems in documentation comments; but it is also recommended to check the generated output with any appropriate conformance and other checking tools.

For more details on the conformance requirements for HTML5 documents, see **Conformance requirements** [https://www.w3.org/TR/html5/infrastructure.html#conformance-requirements] in the HTML5 Specification. For more details on security issues related to web pages, see the **Open Web Application Security Project (OWASP)** [https://www.owasp.org] page.

OPTIONS

`javadoc` supports command-line options for both the main `javadoc` tool and the currently selected doclet. The Standard Doclet is used if no other doclet is specified.

GNU-style options (that is, those beginning with `--`) can use an equal sign (=) instead of whitespace characters to separate the name of an option from its value.

Standard `javadoc` Options

The following core `javadoc` options are equivalent to corresponding `javac` options. See *Standard Options* in **javac** for the detailed descriptions of using these options:

- `--add-modules`
- `-bootclasspath`
- `--class-path`, `-classpath`, or `-cp`
- `--enable-preview`
- `-encoding`
- `-extdirs`
- `--limit-modules`
- `--module`
- `--module-path` or `-p`
- `--module-source-path`
- `--release`
- `--source` or `-source`
- `--source-path` or `-sourcepath`
- `--system`
- `--upgrade-module-path`

The following options are the core `javadoc` options that are not equivalent to a corresponding `javac` option:

`-breakiterator`
: Computes the first sentence with `BreakIterator`. The first sentence is copied to the package, class, or member summary and to the alphabetic index. The `BreakIterator` class is used to determine the end of a sentence for all languages except for English.

  - English default sentence-break algorithm --- Stops at a period followed by a space or an HTML block tag, such as `<P>`.
  - Breakiterator sentence-break algorithm --- Stops at a period, question mark, or exclamation point followed by a space when the next word starts with a capital letter. This is meant to handle most abbreviations (such as "The serial no. is valid", but will not handle "Mr. Smith"). The `-breakiterator` option doesn't stop at HTML tags or sentences that begin with numbers or symbols. The algorithm stops at the last period in `../filename`, even when embedded in an HTML tag.

`-doclet` *class*
: Generates output by using an alternate doclet. Use the fully qualified name. This doclet defines the content and formats the output. If the `-doclet` option isn't used, then the `javadoc` tool uses the standard doclet for generating the default HTML format. This class must contain the `start(Root)` method. The path to this starting class is defined by the `-docletpath` option.

`-docletpath` *path*
: Specifies where to find doclet class files (specified with the `-doclet` option) and any JAR files it depends on. If the starting class file is in a JAR file, then this option specifies the path to that JAR file. You can specify an absolute path or a path relative to the current directory. If `classpath-list` contains multiple paths or JAR files, then they should be separated with a colon (`:`) on Linux and a semi-colon (`;`) on Windows. This option isn't necessary when the `doclet` starting class is already in the search path.

`-exclude` *pkglist*
: Unconditionally, excludes the specified packages and their subpackages from the list formed by `-subpackages`. It excludes those packages even when they would otherwise be included by some earlier or later `-subpackages` option.

  The following example would include `java.io`, `java.util`, and `java.math` (among others), but would exclude packages rooted at `java.net` and `java.lang`. Notice that these examples exclude `java.lang.ref`, which is a subpackage of `java.lang`.

  - **Linux and macOS:**

    ```
    javadoc -sourcepath /home/user/src -subpackages java -exclude java.net
    ```

  - **Windows:**

    ```
    javadoc -sourcepath \user\src -subpackages java -exclude java.net:java
    ```

`--expand-requires` *value*
: Instructs the javadoc tool to expand the set of modules to be documented. By default, only the modules given explicitly on the command line are documented. Supports the following values:

  - `transitive`: additionally includes all the required transitive dependencies of those modules.
  - `all`: includes all dependencies.

`--help`, `-help`, `-h`, or `-?`
: Prints a synopsis of the standard options.

`--help-extra` or `-X`
: Prints a synopsis of the set of extra options.

`-J`*flag*
: Passes *flag* directly to the Java Runtime Environment (JRE) that runs the `javadoc` tool. For example, if you must ensure that the system sets aside 32 MB of memory in which to process the generated documentation, then you would call the `-Xmx` option as follows: `javadoc -J-Xmx32m -J-Xms32m com.mypackage`. Be aware that `-Xms` is optional because it only sets the size of initial memory, which is useful when you know the minimum amount of memory required.

  There is no space between the `J` and the `flag`.

  Use the `-version` option to report the version of the JRE being used to run the `javadoc` tool.

  ```
  javadoc -J-version
  java version "17" 2021-09-14 LTS
  Java(TM) SE Runtime Environment (build 17+35-LTS-2724)
  Java HotSpot(TM) 64-Bit Server VM (build 17+35-LTS-2724, mixed mode, sha
  ```

`-locale` *name*
: Specifies the locale that the `javadoc` tool uses when it generates documentation. The argument is the name of the locale, as described in `java.util.Locale` documentation, such as `en_US` (English, United States) or `en_US_WIN` (Windows variant).

  Specifying a locale causes the `javadoc` tool to choose the resource files of that locale for messages such as strings in the navigation bar, headings for lists and tables, help file contents, comments in the `stylesheet.css` file, and so on. It also specifies the sorting order for lists sorted alphabetically, and the sentence separator to determine the end of the first sentence. The `-locale` option doesn't determine the locale of the documentation comment text specified in the source files of the documented classes.

`-package`
: Shows only package, protected, and public classes and members.

`-private`
: Shows all classes and members.

`-protected`
: Shows only protected and public classes and members. This is the default.

`-public`
: Shows only the public classes and members.

`-quiet`
: Shuts off messages so that only the warnings and errors appear to make them easier to view. It also suppresses the `version` string.

`--show-members` *value*
: Specifies which members (fields or methods) are documented, where *value* can be any of the following:

  - `public` --- shows only public members
  - `protected` --- shows public and protected members; this is the default
  - `package` --- shows public, protected, and package members
  - `private` --- shows all members

`--show-module-contents` *value*
: Specifies the documentation granularity of module declarations, where *value* can be `api` or `all`.

`--show-packages` *value*
: Specifies which modules packages are documented, where *value* can be `exported` or `all` packages.

`--show-types` *value*
: Specifies which types (classes, interfaces, etc.) are documented, where *value* can be any of the following:

  - `public` --- shows only public types
  - `protected` --- shows public and protected types; this is the default
  - `package` --- shows public, protected, and package types
  - `private` --- shows all types

`-subpackages` *subpkglist*
: Generates documentation from source files in the specified packages and recursively in their subpackages. This option is useful when adding new subpackages to the source code because they are automatically included. Each package argument is any top-level subpackage (such as `java`) or fully qualified package (such as `javax.swing`) that doesn't need to contain source files. Arguments are separated by colons on all operating systems. Wild cards aren't allowed. Use `-sourcepath` to specify where to find the packages. This option doesn't process source files that are in the source tree but don't belong to the packages.

  For example, the following commands generates documentation for packages named `java` and `javax.swing` and all of their subpackages.

  - **Linux and macOS:**

    ```
    javadoc -d docs -sourcepath /home/user/src -subpackages java:javax.swi
    ```

  - **Windows:**

    ```
    javadoc -d docs -sourcepath \user\src -subpackages java:javax.swing
    ```

`-verbose`
: Provides more detailed messages while the `javadoc` tool runs. Without the `-verbose` option, messages appear for loading the source files, generating the documentation (one message per source file), and sorting. The `-verbose` option causes the printing of additional messages that specify the number of milliseconds to parse each Java source file.

`--version`
Prints version information.

`-Werror`
Reports an error if any warnings occur.

Extra `javadoc` Options

Note: The additional options for `javadoc` are subject to change without notice.

The following additional `javadoc` options are equivalent to corresponding `javac` options. See *Extra Options* in **javac** for the detailed descriptions of using these options:

- `--add-exports`
- `--add-reads`
- `--patch-module`
- `-Xmaxerrs`
- `-Xmaxwarns`

Standard Options for the Standard Doclet

The following options are provided by the standard doclet.

`--add-script` *file*
Adds *file* as an additional JavaScript file to the generated documentation. This option can be used one or more times to specify additional script files.

Command-line example:

```
javadoc --add-script first_script.js --add-script second_script.js pkg_foo
```

`--add-stylesheet` *file*
Adds *file* as an additional stylesheet file to the generated documentation. This option can be used one or more times to specify additional stylesheets included in the documentation.

Command-line example:

```
javadoc --add-stylesheet new_stylesheet_1.css --add-stylesheet new_style
```

`--allow-script-in-comments`
Allow JavaScript in options and comments.

`-author`
Includes the `@author` text in the generated docs.

`-bottom` *html-code*
Specifies the text to be placed at the bottom of each output file. The text is placed at the bottom of the page, underneath the lower navigation bar. The text can contain HTML tags and white space, but when it does, the text must be enclosed in quotation marks. Use escape characters for any internal quotation marks within text.

`-charset` *name*
Specifies the HTML character set for this document. The name should be a preferred MIME name as specified in the **IANA Registry, Character Sets** [http://www.iana.org/assignments/character-sets].

For example:

```
javadoc -charset "iso-8859-1" mypackage
```

This command inserts the following line in the head of every generated page:

```
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
```

The `meta` tag is described in the **HTML standard (4197265 and 4137321), HTML Document Representation** [http://www.w3.org/TR/REC-html40/charset.html#h-5.2.2].

`-d` *directory*

Specifies the destination directory where the `javadoc` tool saves the generated HTML files. If you omit the `-d` option, then the files are saved to the current directory. The `directory` value can be absolute or relative to the current working directory. The destination directory is automatically created when the `javadoc` tool runs.

- **Linux and macOS:** For example, the following command generates the documentation for the package `com.mypackage` and saves the results in the `/user/doc/` directory:

  `javadoc -d /user/doc/ com.mypackage`

- **Windows:** For example, the following command generates the documentation for the package `com.mypackage` and saves the results in the `\user\doc\` directory:

  `javadoc -d \user\doc\ com.mypackage`

`-docencoding` *name*

Specifies the encoding of the generated HTML files. The name should be a preferred MIME name as specified in the **IANA Registry, Character Sets** [http://www.iana.org/assignments/character-sets].

Three options are available for use in a `javadoc` encoding command. The `-encoding` option is used for encoding the files read by the `javadoc` tool, while the `-docencoding` and `-charset` options are used for encoding the files written by the tool. Of the three available options, at most, only the input and an output encoding option are used in a single encoding command. If you specify both input and output encoding options in a command, they must be the same value. If you specify neither output option, it defaults to the input encoding.

For example:

`javadoc -docencoding "iso-8859-1" mypackage`

`-docfilessubdirs`

Recursively copies doc-file subdirectories. Enables deep copying of doc-files directories. Subdirectories and all contents are recursively copied to the destination. For example, the directory `doc-files/example/images` and all of its contents are copied. The **`-excludedocfilessubdir`** option can be used to exclude specific subdirectories.

`-doctitle` *html-code*

Specifies the title to place near the top of the overview summary file. The text specified in the `title` tag is placed as a centered, level-one heading directly beneath the top navigation bar. The `title` tag can contain HTML tags and white space, but when it does, you must enclose the title in quotation marks. Additional quotation marks within the `title` tag must be escaped. For example, `javadoc -doctitle "<b>My Library</b><br>v1.0" com.mypackage`.

`-excludedocfilessubdir` *name1*`,` *name2...*

Excludes any subdirectories with the given names when recursively copying doc-file subdirectories. See **`-docfilessubdirs`**. For historical reasons, `:` can be used anywhere in the argument as a separator instead of `,`.

`-footer` *html-code*

Specifies the footer text to be placed at the bottom of each output file. The`html-code` value is placed to the right of the lower navigation bar. The `html-code` value can contain HTML tags and white space, but when it does, the `html-code` value must be enclosed in quotation marks. Use escape characters for any internal quotation marks within a footer.

`-group` *name p1*`,` *p2...*

Group the specified packages together in the Overview page. For historical reasons, `:` can be used as a separator anywhere in the argument instead of `,`.

`-header` *html-code*

Specifies the header text to be placed at the top of each output file. The header is placed to the right of the upper navigation bar. The `header` can contain HTML tags and white space, but

when it does, the `header` must be enclosed in quotation marks. Use escape characters for internal quotation marks within a header. For example, `javadoc -header "<b>My Library</b><br>v1.0" com.mypackage`.

`-helpfile` *filename*

Includes the file that links to the **HELP** link in the top and bottom navigation bars . Without this option, the `javadoc` tool creates a help file `help-doc.html` that is hard-coded in the `javadoc` tool. This option lets you override the default. The *filename* can be any name and isn't restricted to `help-doc.html`. The `javadoc` tool adjusts the links in the navigation bar accordingly. For example:

- **Linux and macOS:**

```
javadoc -helpfile /home/user/myhelp.html java.awt
```

- **Windows:**

```
javadoc -helpfile C:\user\myhelp.html java.awt
```

`-html5`

This option is a no-op and is just retained for backwards compatibility.

`--javafx` or `-javafx`

Enables JavaFX functionality. This option is enabled by default if the JavaFX library classes are detected on the module path.

`-keywords`

Adds HTML keyword `<meta>` tags to the generated file for each class. These tags can help search engines that look for `<meta>` tags find the pages. Most search engines that search the entire Internet don't look at `<meta>` tags, because pages can misuse them. Search engines offered by companies that confine their searches to their own website can benefit by looking at `<meta>` tags. The `<meta>` tags include the fully qualified name of the class and the unqualified names of the fields and methods. Constructors aren't included because they are identical to the class name. For example, the class `String` starts with these keywords:

```
<meta name="keywords" content="java.lang.String class">
<meta name="keywords" content="CASE_INSENSITIVE_ORDER">
<meta name="keywords" content="length()">
<meta name="keywords" content="charAt()">
```

`-link` *url*

Creates links to existing `javadoc` generated documentation of externally referenced classes. The *url* argument is the absolute or relative URL of the directory that contains the external `javadoc` generated documentation. You can specify multiple `-link` options in a specified `javadoc` tool run to link to multiple documents.

Either a `package-list` or an `element-list` file must be in this *url* directory (otherwise, use the `-linkoffline` option).

Note: The `package-list` and `element-list` files are generated by the `javadoc` tool when generating the API documentation and should not be modified by the user.

When you use the `javadoc` tool to document packages, it uses the `package-list` file to determine the packages declared in an API. When you generate API documents for modules, the `javadoc` tool uses the `element-list` file to determine the modules and packages declared in an API.

The `javadoc` tool reads the names from the appropriate list file and then links to the packages or modules at that URL.

When the `javadoc` tool runs, the *url* value is copied into the `<A HREF>` links that are created. Therefore, *url* must be the URL to the directory and not to a file.

You can use an absolute link for *url* to enable your documents to link to a document on any web

site, or you can use a relative link to link only to a relative location. If you use a relative link, then the value you pass in should be the relative path from the destination directory (specified with the `-d` option) to the directory containing the packages being linked to. When you specify an absolute link, you usually use an HTTP link. However, if you want to link to a file system that has no web server, then you can use a file link. Use a file link only when everyone who wants to access the generated documentation shares the same file system. In all cases, and on all operating systems, use a slash as the separator, whether the URL is absolute or relative, and `https:`, `http:`, or `file:` as specified in the **URL Memo: Uniform Resource Locators** [http://www.ietf.org/rfc/rfc1738.txt].

```
-link https://<host>/<directory>/<directory>/.../<name>
-link http://<host>/<directory>/<directory>/.../<name>
-link file://<host>/<directory>/<directory>/.../<name>
-link <directory>/<directory>/.../<name>
```

`--link-modularity-mismatch` (`warn`|`info`)

Specifies whether external documentation with wrong modularity (e.g. non-modular documentation for a modular library, or the reverse case) should be reported as a warning (`warn`) or just a message (`info`). The default behavior is to report a warning.

`-linkoffline` *url1 url2*

This option is a variation of the `-link` option. They both create links to `javadoc` generated documentation for externally referenced classes. You can specify multiple `-linkoffline` options in a specified `javadoc` tool run.

Use the `-linkoffline` option when:

- Linking to a document on the web that the `javadoc` tool can't access through a web connection
- The `package-list` or `element-list` file of the external document either isn't accessible or doesn't exist at the URL location, but does exist at a different location and can be specified by either the `package-list` or `element-list` file (typically local).

Note: The `package-list` and `element-list` files are generated by the `javadoc` tool when generating the API documentation and should not be modified by the user.

If *url1* is accessible only on the World Wide Web, then the `-linkoffline` option removes the constraint that the `javadoc` tool must have a web connection to generate documentation.

Another use of the `-linkoffline` option is as a work-around to update documents. After you have run the `javadoc` tool on a full set of packages or modules, you can run the `javadoc` tool again on a smaller set of changed packages or modules, so that the updated files can be inserted back into the original set.

For example, the `-linkoffline` option takes two arguments. The first is for the string to be embedded in the `<a href>` links, and the second tells the `javadoc` tool where to find either the `package-list` or `element-list` file.

The *url1* or *url2* value is the absolute or relative URL of the directory that contains the external `javadoc` generated documentation that you want to link to. When relative, the value should be the relative path from the destination directory (specified with the `-d` option) to the root of the packages being linked to. See *url* in the `-link` option.

`--link-platform-properties` *url*

Specifies a properties file used to configure links to platform documentation.

The *url* argument is expected to point to a properties file containing one or more entries with the following format, where `<version>` is the platform version as passed to the `--release` or `--source` option and `<url>` is the base URL of the corresponding platform API documentation:

doclet.platform.docs.<version>=<url>

For instance, a properties file containing URLs for releases 15 to 17 might contain the following lines:

```
doclet.platform.docs.15=https://example.com/api/15/
doclet.platform.docs.16=https://example.com/api/16/
doclet.platform.docs.17=https://example.com/api/17/
```

If the properties file does not contain an entry for a particular release no platform links are generated.

`-linksource`
: Creates an HTML version of each source file (with line numbers) and adds links to them from the standard HTML documentation. Links are created for classes, interfaces, constructors, methods, and fields whose declarations are in a source file. Otherwise, links aren't created, such as for default constructors and generated classes.

  This option exposes all private implementation details in the included source files, including private classes, private fields, and the bodies of private methods, regardless of the `-public`, `-package`, `-protected`, and `-private` options. Unless you also use the `-private` option, not all private classes or interfaces are accessible through links.

  Each link appears on the name of the identifier in its declaration. For example, the link to the source code of the `Button` class would be on the word `Button`:

  public class Button extends Component implements Accessible

  The link to the source code of the `getLabel` method in the `Button` class is on the word `getLabel`:

  public String getLabel()

`--main-stylesheet` *file* or `-stylesheetfile` *file*
: Specifies the path of an alternate stylesheet file that contains the definitions for the CSS styles used in the generated documentation. This option lets you override the default. If you do not specify the option, the `javadoc` tool will create and use a default stylesheet. The file name can be any name and isn't restricted to `stylesheet.css`. The `--main-stylesheet` option is the preferred form.

  Command-line example:

  ```
  javadoc --main-stylesheet main_stylesheet.css pkg_foo
  ```

`-nocomment`
: Suppresses the entire comment body, including the main description and all tags, and generate only declarations. This option lets you reuse source files that were originally intended for a different purpose so that you can produce skeleton HTML documentation during the early stages of a new project.

`-nodeprecated`
: Prevents the generation of any deprecated API in the documentation. This does what the `-nodeprecatedlist` option does, and it doesn't generate any deprecated API throughout the rest of the documentation. This is useful when writing code when you don't want to be distracted by the deprecated code.

`-nodeprecatedlist`
: Prevents the generation of the file that contains the list of deprecated APIs (`deprecated-list.html`) and the link in the navigation bar to that page. The `javadoc` tool continues to generate the deprecated API throughout the rest of the document. This is useful when your source code contains no deprecated APIs, and you want to make the navigation bar cleaner.

`-nohelp`
: Omits the HELP link in the navigation bar at the top of each page of output.

`-noindex`
: Omits the index from the generated documents. The index is produced by default.

`-nonavbar`
: Prevents the generation of the navigation bar, header, and footer, that are usually found at the top and bottom of the generated pages. The `-nonavbar` option has no effect on the `-bottom` option. The `-nonavbar` option is useful when you are interested only in the content and have no need for navigation, such as when you are converting the files to PostScript or PDF for printing only.

`--no-platform-links`
: Prevents the generation of links to platform documentation. These links are generated by default.

`-noqualifier` *name1*`,`*name2...*
: Excludes the list of qualifiers from the output. The package name is removed from places where class or interface names appear. For historical reasons, `:` can be used anywhere in the argument as a separator instead of `,`.

  The following example omits all package qualifiers: `-noqualifier all`.

  The following example omits `java.lang` and `java.io` package qualifiers: `-noqualifier java.lang:java.io`.

  The following example omits package qualifiers starting with `java` and `com.sun` subpackages, but not `javax`: `-noqualifier java.*:com.sun.*`.

  Where a package qualifier would appear due to the previous behavior, the name can be suitably shortened. This rule is in effect whether or not the `-noqualifier` option is used.

`-nosince`
: Omits from the generated documents the `Since` sections associated with the `@since` tags.

`-notimestamp`
: Suppresses the time stamp, which is hidden in an HTML comment in the generated HTML near the top of each page. The `-notimestamp` option is useful when you want to run the `javadoc` tool on two source bases and get the differences between `diff` them, because it prevents time stamps from causing a `diff` (which would otherwise be a `diff` on every page). The time stamp includes the `javadoc` tool release number.

`-notree`
: Omits the class and interface hierarchy pages from the generated documents. These are the pages you reach using the Tree button in the navigation bar. The hierarchy is produced by default.

`--override-methods` (`detail`|`summary`)
: Documents overridden methods in the detail or summary sections. The default is `detail`.

`-overview` *filename*
: Specifies that the `javadoc` tool should retrieve the text for the overview documentation from the source file specified by `filename` and place it on the Overview page (`overview-summary.html`). A relative path specified with the file name is relative to the current working directory.

  While you can use any name you want for the `filename` value and place it anywhere you want for the path, it is typical to name it `overview.html` and place it in the source tree at the directory that contains the topmost package directories. In this location, no path is needed when documenting packages, because the `-sourcepath` option points to this file.

  - **Linux and macOS:** For example, if the source tree for the `java.lang` package is `src/classes/java/lang/`, then you could place the overview file at src/classes/overview.html.

- **Windows:** For example, if the source tree for the `java.lang` package is `src\classes\java\lang\`, then you could place the overview file at `src\classes\overview.html`

The overview page is created only when you pass two or more package names to the `javadoc` tool. The title on the overview page is set by `-doctitle`.

`-serialwarn`
: Generates compile-time warnings for missing `@serial` tags. By default, Javadoc generates no serial warnings. Use this option to display the serial warnings, which helps to properly document default serializable fields and `writeExternal` methods.

`--since` *release*(`,` *release*)*
: Generates documentation for APIs that were added or newly deprecated in the specified *release*s.

  If the `@since` tag in the `javadoc` comment of an element in the documented source code matches a *release* passed as option argument, information about the element and the release it was added in is included in a "New API" page.

  If the "Deprecated API" page is generated and the `since` element of the `java.lang.Deprecated` annotation of a documented element matches a *release* in the option arguments, information about the release the element was deprecated in is added to the "Deprecated API" page.

  Releases are compared using case-sensitive string comparison.

`--since-label` *text*
: Specifies the *text* to use in the heading of the "New API" page. This may contain information about the releases covered in the page, e.g. "New API in release 2.0", or "New API since release 1".

`--snippet-path` *snippetpathlist*
: Specifies the search paths for finding files for external snippets. The *snippetpathlist* can contain multiple paths by separating them with the platform path separator (`;` on Windows; `:` on other platforms.) The Standard Doclet first searches the `snippet-files` subdirectory in the package containing the snippet, and then searches all the directories in the given list.

`-sourcetab` *tab-length*
: Specifies the number of spaces each tab uses in the source.

`--spec-base-url` *url*
: Specifies the base URL for relative URLs in `@spec` tags, to be used when generating links to any external specifications. It can either be an absolute URL, or a relative URL, in which case it is evaluated relative to the base directory of the generated output files. The default value is equivalent to `{@docRoot}/../specs`.

`-splitindex`
: Splits the index file into multiple files, alphabetically, one file per letter, plus a file for any index entries that start with non-alphabetical symbols.

`-tag` *name*:*locations*:*header*
: Specifies single argument custom tags. For the `javadoc` tool to spell-check tag names, it is important to include a `-tag` option for every custom tag that is present in the source code, disabling (with `X`) those that aren't being output in the current run. The colon (`:`) is always the separator. To include a colon in the tag name, escape it with a backward slash (`\`). The `-tag` option outputs the tag heading, *header*, in bold, followed on the next line by the text from its single argument. Similar to any block tag, the argument text can contain inline tags, which are also interpreted. The output is similar to standard one-argument tags, such as the `@return` and `@author` tags. Omitting a *header* value causes the *name* to be the heading. *locations* is a list of characters specifying the kinds of declarations in which the tag may be used. The following characters may be used, in either uppercase or lowercase:

- `A`: all declarations
- `C`: constructors
- `F`: fields
- `M`: methods
- `O`: the overview page and other documentation files in `doc-files` subdirectories
- `P`: packages
- `S`: modules
- `T`: types (classes and interfaces)
- `X`: nowhere: the tag is disabled, and will be ignored

The order in which tags are given on the command line will be used as the order in which the tags appear in the generated output. You can include standard tags in the order given on the command line by using the `-tag` option with no *locations* or *header*.

`-taglet` *class*

Specifies the fully qualified name of the taglet used in generating the documentation for that tag. Use the fully qualified name for the *class* value. This taglet also defines the number of text arguments that the custom tag has. The taglet accepts those arguments, processes them, and generates the output.

Taglets are useful for block or inline tags. They can have any number of arguments and implement custom behavior, such as making text bold, formatting bullets, writing out the text to a file, or starting other processes. Taglets can only determine where a tag should appear and in what form. All other decisions are made by the doclet. A taglet can't do things such as remove a class name from the list of included classes. However, it can execute side effects, such as printing the tag's text to a file or triggering another process. Use the `-tagletpath` option to specify the path to the taglet. The following example inserts the To Do taglet after Parameters and ahead of Throws in the generated pages.

```
-taglet com.sun.tools.doclets.ToDoTaglet
-tagletpath /home/taglets
-tag return
-tag param
-tag todo
-tag throws
-tag see
```

Alternately, you can use the `-taglet` option in place of its `-tag` option, but that might be difficult to read.

`-tagletpath` *tagletpathlist*

Specifies the search paths for finding taglet class files. The *tagletpathlist* can contain multiple paths by separating them with the platform path separator (`;` on Windows; `:` on other platforms.) The `javadoc` tool searches all subdirectories of the specified paths.

`-top` *html-code*

Specifies the text to be placed at the top of each output file.

`-use` Creates class and package usage pages. Includes one Use page for each documented class and package. The page describes what packages, classes, methods, constructors and fields use any API of the specified class or package. Given class C, things that use class C would include subclasses of C, fields declared as C, methods that return C, and methods and constructors with parameters of type C. For example, you can look at the Use page for the `String` type. Because the `getName` method in the `java.awt.Font` class returns type `String`, the `getName` method uses `String` and so the `getName` method appears on the Use page for `String`. This documents only uses of the API, not the implementation. When a method uses `String` in its implementation,

but doesn't take a string as an argument or return a string, that isn't considered a use of `String`.To access the generated Use page, go to the class or package and click the **Use link** in the navigation bar.

`-version`
Includes the version text in the generated docs. This text is omitted by default. To find out what version of the `javadoc` tool you are using, use the `-J-version` option.

`-windowtitle` *title*
Specifies the title to be placed in the HTML `<title>` tag. The text specified in the `title` tag appears in the window title and in any browser bookmarks (favorite places) that someone creates for this page. This title should not contain any HTML tags because a browser will not interpret them correctly. Use escape characters on any internal quotation marks within the `title` tag. If the `-windowtitle` option is omitted, then the `javadoc` tool uses the value of the `-doctitle` option for the `-windowtitle` option. For example, `javadoc -windowtitle "My Library" com.mypackage`.

Extra Options for the Standard Doclet

The following are additional options provided by the Standard Doclet and are subject to change without notice. Additional options are less commonly used or are otherwise regarded as advanced.

`--date` *date-and-time*
Specifies the value to be used to timestamp the generated pages, in **ISO 8601** [https://www.iso.org/iso-8601-date-and-time-format.html] format. The specified value must be within 10 years of the current date and time. It is an error to specify both `-notimestamp` and `--date`. Using a specific value means the generated documentation can be part of a **reproducible build** [https://reproducible-builds.org/]. If the option is not given, the default value is the current date and time. For example:

```
javadoc --date 2022-02-01T17:41:59-08:00 mypackage
```

`--legal-notices` (`default`|`none`|*directory*)
Specifies the location from which to copy legal files to the generated documentation. If the option is not specified or is used with the value `default`, the files are copied from the default location. If the argument is used with value `none`, no files are copied. Every other argument is interpreted as directory from which to copy the legal files.

`--no-frames`
This option is a no-op and is just retained for backwards compatibility.

`-Xdoclint`
Enables recommended checks for problems in documentation comments.

By default, the `-Xdoclint` option is enabled. Disable it with the option `-Xdoclint:none`.

For more details, see **DocLint**.

`-Xdoclint:`*flag*,*flag*,...
Enable or disable specific checks for different kinds of issues in documentation comments.

Each *flag* can be one of `all`, `none`, or `[-]`*group* where *group* has one of the following values: `accessibility`, `html`, `missing`, `reference`, `syntax`. For more details on these values, see **DocLint Groups**.

When specifying two or more flags, you can either use a single `-Xdoclint:...` option, listing all the desired flags, or you can use multiple options giving one or more flag in each option. For example, use either of the following commands to check for the HTML, syntax, and accessibility issues in the file `MyFile.java`.

```
javadoc -Xdoclint:html -Xdoclint:syntax -Xdoclint:accessibility MyFile.j
javadoc -Xdoclint:html,syntax,accessibility MyFile.java
```

The following examples illustrate how to change what DocLint reports:

- `-Xdoclint:none` --- disables all checks
- `-Xdoclint:`*group* --- enables *group* checks
- `-Xdoclint:all` --- enables all groups of checks
- `-Xdoclint:all,`*-group* --- enables all checks except *group* checks

For more details, see **DocLint**.

`-Xdoclint/package:[-]`*packages*
: Enables or disables checks in specific packages. *packages* is a comma separated list of package specifiers. A package specifier is either a qualified name of a package or a package name prefix followed by `*`, which expands to all subpackages of the given package. Prefix the package specifier with − to disable checks for the specified packages.

  For more details, see **DocLint**.

`-Xdocrootparent` *url*
: Replaces all `@docRoot` items followed by `/..` in documentation comments with *url*.

DOCLINT

DocLint provides the ability to check for possible problems in documentation comments. Problems may be reported as warnings or errors, depending on their severity. For example, a missing comment may be bad style that deserves a warning, but a link to an unknown Java declaration is more serious and deserves an error. Problems are organized into **groups**, and options can be used to enable or disable messages in one or more groups. Within the source code, messages in one or more groups can be **suppressed** by using `@SuppressWarnings` annotations.

When invoked from `javadoc`, by default DocLint checks all comments that are used in the generated documentation. It thus relies on other command-line options to determine which declarations, and which corresponding documentation comments will be included. *Note:* this may mean that even comments on some private members of serializable classes will also be checked, if the members need to be documented in the generated `Serialized Forms` page.

In contrast, when DocLint is invoked from `javac`, DocLint solely relies on the various `-Xdoclint...` options to determine which documentation comments to check.

DocLint doesn't attempt to fix invalid input, it just reports it.

Note: DocLint doesn't guarantee the completeness of these checks. In particular, it isn't a full HTML compliance checker. The goal is to just report common errors in a convenient manner.

Groups

The checks performed by DocLint are organized into groups. The warnings and errors in each group can be enabled or disabled with command-line options, or suppressed with `@SuppressWarnings` annotations.

The groups are as follows:

- `accessibility` --- Checks for issues related to accessibility. For example, no `alt` attribute specified in an `<img>` element, or no caption or summary attributes specified in a `<table>` element.

  Issues are reported as errors if a downstream validation tool might be expected to report an error in the files generated by `javadoc`.

  For reference, see the **Web Content Accessibility Guidelines** [https://www.w3.org/WAI/standards-guidelines/wcag/].

- `html` --- Detects common high-level HTML issues. For example, putting block elements inside inline elements, or not closing elements that require an end tag.

  Issues are reported as errors if a downstream validation tool might be expected to report an error in the files generated by `javadoc`.

  For reference, see the **HTML Living Standard** [https://html.spec.whatwg.org/multipage/].

- `missing` --- Checks for missing documentation comments or tags. For example, a missing comment on a class declaration, or a missing `@param` or `@return` tag in the comment for a method declaration.

  Issues related to missing items are typically reported as warnings because they are unlikely to be reported as errors by downstream validation tools that may be used to check the output generated by `javadoc`.

- `reference` --- Checks for issues relating to the references to Java API elements from documentation comment tags. For example, the reference in `@see` or `{@link ...}` cannot be found, or a bad name is given for `@param` or `@throws`.

  Issues are typically reported as errors because while the issue may not cause problems in the generated files, the author has likely made a mistake that will lead to incorrect or unexpected documentation.

- `syntax` --- Checks for low-level syntactic issues in documentation comments. For example, unescaped angle brackets (< and >) and ampersands (`&`) and invalid documentation comment tags.

  Issues are typically reported as errors because the issues may lead to incorrect or unexpected documentation.

Suppressing Messages

DocLint checks for and recognizes two strings that may be present in the arguments for an `@SuppressWarnings` annotation.

- `doclint`
- `doclint:`*LIST*

where *LIST* is a comma-separated list of one or more of `accessibility`, `html`, `missing`, `syntax`, `reference`.

The names in *LIST* are the same **group** names supported by the command-line `-Xdoclint` option for `javac` and `javadoc`. (This is the same convention honored by the `javac -Xlint` option and the corresponding names supported by `@SuppressWarnings`.)

The names in *LIST* can equivalently be specified in separate arguments of the annotation. For example, the following are equivalent:

- `@SuppressWarnings("doclint:accessibility,missing")`
- `@SuppressWarnings("doclint:accessibility", "doclint:missing")`

When DocLint detects an issue in a documentation comment, it checks for the presence of `@SuppressWarnings` on the associated declaration and on all lexically enclosing declarations. The issue will be ignored if any such annotation is found containing the simple string `doclint` or the longer form `doclint:LIST` where *LIST* contains the name of the group for the issue.

Note: as with other uses of `@SuppressWarnings`, using the annotation on a module or package declaration only affects that declaration; it does not affect the contents of the module or package in other source files.

All messages related to an issue are suppressed by the presence of an appropriate `@SuppressWarnings` annotation: this includes errors as well as warnings.

Note: It is only possible to *suppress* messages. If an annotation of `@SuppressWarnings("doclint")` is given on a top-level declaration, all DocLint messages for that declaration and any enclosed declarations will be suppressed; it is not possible to selectively re-enable messages for issues in enclosed declarations.

Comparison with downstream validation tools

DocLint is a utility built into `javac` and `javadoc` that checks the content of documentation comments, as found in source files. In contrast, downstream validation tools can be used to validate the output generated from those documentation comments by `javadoc` and the Standard Doclet.

Although there is some overlap in functionality, the two mechanisms are different and each has its own strengths and weaknesses.

- Downstream validation tools can check the end result of any generated documentation, as it will be seen by the end user. This includes content from all sources, including documentation comments, the Standard Doclet itself, user-provided taglets, and content supplied via command-line options. Because such tools are analyzing complete HTML pages, they can do more complete checks than can DocLint. However, when a problem is found in the generated pages, it can be harder to track down exactly where in the build pipeline the problem needs to be fixed.
- DocLint checks the content of documentation comments, in source files. This makes it very easy to identify the exact position of any issues that may be found. DocLint can also detect some semantic errors in documentation comments that downstream tools cannot detect, such as missing comments, using an `@return` tag in a method returning `void`, or an `@param` tag describing a non-existent parameter. But by its nature, DocLint cannot report on problems such as missing links, or errors in user-provided custom taglets, or problems in the Standard Doclet itself. It also cannot reliably detect errors in documentation comments at the boundaries between content in a documentation comment and content generated by a custom taglet.