

Using The Template With Visual Studio Code

Abstract

This documentation provides a quick reference to the core features of [Visual Studio Code's Editor](#). It focusses on aspects that are expected to be used in the context of the present template. Reading these pages is not a prerequisite to use the template. However, as it points out the way the template was designed in the aim of exploiting certain of VSCode's features and extensions, a quick walk through before starting to use the template for the first time should be of interest even to those who are already familiar with VSCode and is recommended to everyone.

▼ TABLE OF CONTENTS

- [About This Documentation](#)
- [VSCode's Editor Overview](#)
 - [VSCode's Workspaces](#)
 - [VSCode Configuration](#)
- [The Admonitions Extension](#)
 - [Admonitions Overview](#)
 - [Admonitions' Usage](#)
 - [Creating Admonitions](#)
 - [Default Types Of Admonitions](#)
 - [Customizing Admonitions](#)
 - [Transpiled HTML](#)
 - [Custom Types Of Admonitions](#)
 - [Customizing Admonitions' Styles](#)
- [The Todo-Tree Extension](#)
 - [Todo-Tree Overview](#)
 - [Todo-Tree's Tag Categories](#)
 - [Todo-Tree's Tags Description](#)
 - [Issues Tag Category](#)
 - [Alerts Tag Category](#)
 - [Records Tag Category](#)
 - [Hunches Tag Category](#)
 - [Markers Tag Category](#)
 - [Helpers Tag Category](#)
 - [Dividers Tag Category](#)

About This Documentation

Visual Studio Code or, for short *VSCode*, is a cross-platform source code editor developed by Microsoft. It includes support for debugging, embedded Git control and GitHub, syntax highlighting, intelligent code completion, snippets, and code refactoring. It is highly customizable, allowing users to change the theme, keyboard shortcuts, preferences, and install extensions that add additional functionality. The source code is free and open source and released under the permissive MIT License. The compiled binaries are freeware and free for private or commercial use.

The purpose of the present documentation is not to serve as a tutorial for VSCode, but to provide an insight of the features and extensions that the enclosing template was designed to exploit. It is not a prerequisite to read it before using the template, but it is recommended to do so, even for those who are already familiar with VSCode. The sections dedicated to the extensions, in particular, should be of interest even to VSCode's experts. The latter are also encouraged to do so and to provide feedbacks and suggestions, as knowledgeable opinions are for sure one of the most valuable assets any project can benefit from.



VSCode's Usefull Links

The present documentation is not intended to serve as a reference for VSCode. For any question you may have about VSCode itself, please refer to the following links:

- [Official website](#)
- [Official documentation](#)
- [Getting started with Visual Studio Code](#)

Despite it not being the goal of this document to provide a tutorial for VSCode, a few words about its core concepts features are in order. The next section describe how a project is organized in VSCode and how configuration works.

VSCode's Editor Overview

VSCode's Workspaces

A `workspace` is a collection of files and folders opened in the editor. A `single-root workspace` contains a single folder, while a `multi-root workspace` has several.

The concept of a workspace enables VS Code to:

- Configure settings applying only to specific folders.
- Persist `tasks-link` and `debugger launch` configurations.
- Store and restore UI states.
- Selectively enable or disable extensions only for that workspace.

Creating a workspace is as simple as opening a folder or a set of folders in the editor. Once opened, VSCode will automatically keep track of its state, which includes the last used layout. VSCode saves the configuration of single-root workspaces in a folder named `.vscode` in the workspace root directory. It does in a `<the-project-name>.code-workspace` file for multi-root workspaces; that file may lie wherever convenient, including a remote location.



Note

The syntax used by VSCode in the case of a single-root workspace lacks consistency. Indeed, until a `.code-workspace` file exists, the Settings Panel labels `User` and `Workspace` as the available configuration levels. Then, it surprisingly saves the preferences within a `.vscode/settings.json` file, although such location should be home to nothing else than *folder level settings*! This inconsistency is enhanced when defining and saving some preferences **first** and creating a `.code-workspace` **later**.

One would expect the *workspace* configuration to be available in the `Workspace` tab of the editor's *Settings Panel*, but they are not. Instead, they are now considered a `Folder` configuration. Such behavior makes sense when considering the saving location but makes none when focusing on the editor's UI. Everything happens as if the UI changed the nature of the settings.

It is worth pointing out that multiple versions of a `.code-workspace` file can coexist, which allows one to save different configurations of the same workspace and switch between them easily.

VSCode Configuration

VSCode configuration happens at three levels: `User`, `Workspace`, and `Folder`, which override each other in that order. The Settings Panel of the editor lets the user choose the level to edit and updates the corresponding file:

- **User settings** are (typically) defined in the file `settings.json` located in the `.vscode` folder of the *user's home directory*. They then apply globally to all VSCode instances with the lowest priority. Note that the *user's settings* themselves branch into two categories:
 - **Global settings** are defined in the file `settings.json` located in the `.vscode` folder of the *user's home directory*. They apply to all VSCode instances with the lowest priority.
 - **Profile settings** can be defined only if a *profile* different from the default one is used and apply only when that profile is active. In that case, they override the *global settings*. Switching from one profile to another is done by clicking on the profile's name in the bottom-left corner of the VSCode's window. The same interface is also used to create, edit, and delete profiles.
- **Workspace settings** lie in the `.code-workspace` that VSCode creates when choosing to [save a workspace](#). These settings then apply to the opened workspace, overriding the *user's settings* but not the *folder's settings*.
- **Folder settings** are in the file `settings.json` located in the `.vscode` folder of the selected `<root-folder>`. They apply, with the highest priority, to the whole contents of the enclosing folder.

Going through VSCode's `Settings Panel` is not the only way to configure the editor. An alternative consists in modifying the corresponding `json` files directly. That second method is much faster and, in certain cases, gives access to more options than the graphical interface of the setting panel does. However, it requires a good knowledge of the available settings and/or intensive use of the documentation.

The Admonitions Extension

`markdown.extensions.admonition`

*This is a quick reference for VSCode's **admonition extension**, which allows to display messages in fancy and vibrant colored boxes and use icon + text headers to grab readers' attention. This extension is included in the standard Markdown library. The full documentation is available on [Markdown's Officially Supported Extension's Website](#).*

Admonitions Overview

The typical use case for *admonitions* is to draw the reader's attention to a particular point. They can appear anywhere in an ordinary HTML `body` element, including another *admonition*.

Admonitions consist of a colored box with an optional header and a body. The header is composed of an icon and a title, both of which can be customized. The body is a normal paragraph of text.

Below is a quick example of multiple nested admonitions of different types:

Note

This is the **note** admonition body

Danger Title

This is the **danger** admonition body

This is the **success** admonition body, with stripped out title and icons

As shown below, there are plenty of other types to choose from !

Admonitions' Usage

Creating Admonitions

The extension `markdown.extensions.admonition` provides the following syntax for admonitions:

```
!!! <type> [Alternative Title | "" to hide the header]

<paragraph>

<paragraph>

...
```

, where the `<type>` defines the overall style (icon and color) of the admonition and the `<paragraph>` is a normal paragraph of text. As an example, here is the code that generated the admonitions of the *overview* section:

```
!!! note

    This is the note admonition body

!!! danger Danger Title
    This is the danger admonition body

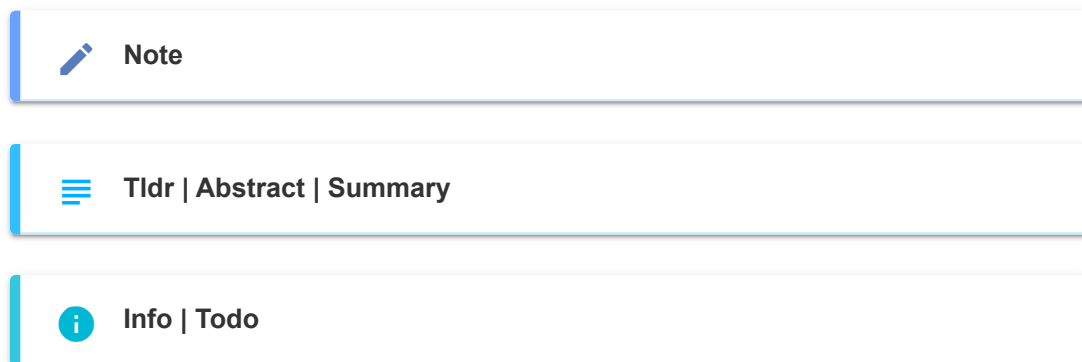
!!! success ""
    This is the success admonition body, with stripped out title and ico













    As shown below, there are plenty of other types to choose from !
```

A detailed description of the HTML code generated by the Markdown snippet above is provided in the last sections, starting from [Customization](#Customizing Admonitions).

Default Types Of Admonitions

Below is a list of the *default admonitions* types as given in the extension's [official documentation](#). Note that each icon is assigned to one and only one color. The converse is no true, though, as each color is assigned to multiple icons. This behavior may be changed by creating your own custom types, as explained in the next section.



	Tip Hint
	Done Check Success
	Faq Help Question
	Warning
	Attention
	Caution
	Fail Missing Failure
	Danger
	Error
	Bug
	Snippet Example
	Cite Quote

Customizing Admonitions

Admonition can be customized using CSS. To understand how, we first look at the HTML structure of an admonition. Then, we show how to implement a custom *type* and how to customize its look using CSS.

Transpiled HTML

The HTML structure of an admonition consists of a *division* container in which the header and every “chunk” of text in the body are considered as two separate *paragraphs*. Only the *division* and the title *paragraph*, are assigned a CSS class; the body *paragraph* is not. For example, the following markdown code :

```
!!! note The Note Title is considered as a paragraph
    And so is also the note's body, with the difference that is not assigned a CSS cl

    It is worth noting that the body can be made of several paragraphs, as shown below
```

will generate *note* admonition with both a title and a body. The corresponding HTML code will then consist three distinct `<p>` elements within a `<div>` . More precisely, the generated HTML code will be:

```
<div class="admonition note">
<p class="admonition-title">The Note Title is considered as a paragraph</p>
<p>And so is also the note's body, with the difference that is not assigned a CSS class</p>
<p>It is worth noting that the body can be made of several paragraphs, as shown below</p>
</div>
```

Custom Types Of Admonitions

According to the previous section, getting your own custom types is then as simple as invoking your own *type* and defining the corresponding CSS class according to the desired look. For example, creating the “foo” *type* simply amounts to using the following markdown code:

```
```markdown
!!! foo The Foo Title This is the foo admonition body
```
```

and defining the corresponding CSS class to take control over the following HTML code:

```
<div class="admonition foo">
<p class="admonition-title">The Foo Title</p>
<p>This is the foo admonition body</p>
</div>
```

The following CSS code snippet shows how to customize the look of the “foo” admonition:

```

.admonition.foo {
  border: 1px solid #000000;
  border-radius: 0.5em;
  padding: 0.5em;
  margin: 1em 0;
  background-color: #f0f0f0;
  color: #000000;
}
.admonition.foo .admonition-title {
  font-weight: bold;
  text-transform: uppercase;
  font-size: 90%;
  margin: 0;
  padding: 0;
  border: 0;
  background-color: transparent;
  color: #000000;
}

```

Customizing Admonitions' Styles

Last but not least, a custom CSS must be defined to take control over the HTML code generated by the extension. Due to the fact that only the `div container` contains and the `title paragraph` are assigned a CSS class, the CSS code to use is not so trivial. Indeed:

- `Div containers` are easy to select as they are simply any `div` elements with the `admonition` class.
- `Title paragraph` are the `p` elements with the `admonition-title` class **that are within the above div container**.
- `Body paragraphs` are the `p` elements that are within the above `div` container but that **do not have the `admonition-title` class**. That last condition is not easy to implement using CSS, since there are no such selectors as “not” or “except”. The easiest way to do it is to make sure that `body paragraphs` are styled **above** the `title paragraph` and that the latter override every style that is not desired.

Here is an example of a working CSS code snippet that implements the above rules:


```

.admonition.foo {
  border: 1px solid gray;
  border-radius: 0.5em;
  padding: 0.5em;
  margin: 1em 0;
  color: #323232;
}
div.admonition.foo p {
  margin: 0;
  padding: 0;
  border: 0;
  background-color: transparent;
  color: #lightyellow;
}
.admonition.foo .admonition-title {
  font-weight: bold;
  text-variant: small-caps;
  font-size: 90%;
  margin: 0;
  padding: 0;
  border: 0;
  background-color: transparent;
  color: #white;
}

```

The Todo-Tree Extension

Abstract

*This documentation provides a quick reference to VSCode's **Todo-Tree** extension. Its primary purpose is to describe the way it is configured in this project's workspace and how it is meant to be used.*

Todo-Tree Overview

The extension **Todo-Tree** uses **ripgrep** to search the workspace for keywords like `TODO`, `FIXME` or `NOTE` and displays the results in an interactive *tree view* that can be sorted, filtered and exported.

The `TAGS` it looks for can be customized, and it can also be configured to further bundle the results into `SUBTAGS`. Those can also be highlighted in the editor according to a customizable color scheme.

The following tables list the tags that are currently made available by the extension's configuration defined in [vscode-index.code-workspace](#).

Note on icons

The icons in use are the ones provided by the [octicons](#) set. Due to poor and unstable support of colors, using alternative sets is not recommended.

Todo-Tree's Tag Categories




The tags defined in this project's workspace are spliced into 7 disjoint categories which definitions are summarized in the table below. More details as well as the tags available in each category are provided in the next sections.

Category	Description
Issues	Tags identifying major issues and calling for immediate action
Alerts	Tags pointing to issues which severity is not yet established
Helpers	Tags which are used to hold information about the project
Markers	Tags identifying editable parts of the project like templates
Dividers	Tags which are used to visually segment the project's file
Records	Tags containing general tasks meant to produce tasks' lists
Hunches	Tags referring to modifications not to be in the next release

Todo-Tree's Tags Description

Issues Tag Category




Tags identifying major issues and calling for immediate action. Freeing the project from all tags falling into that category should be the developers' highest priority. No release should be considered until every single one of them has been addressed and removed from the project, no matter its present state and the location of the tags. For this reason, the extension should be configured to display the tags from this category wherever the tree is generated from.

	TAG	Description
	BUG	Describes an identified bug (still) needing to be addressed.
	FIXME	Formally identifies and describes something needing a fix.
	FINISH	Documents a non-critical task that still needs to be done.

Alerts Tag Category




Tags pointing to issues whose severity is still to be established, but that are potentially damaging enough to be considered as a threat. The tags from this category are typically transitional, meaning that addressing such a tag is expected to yield its requalification as either a *record* or an *issue* tag. Cleaning the project from all *alerts* should be

every developer’s 2nd highest priority, right after clearing all tags marked as *issues*. Again, no release should be considered as long as the todo-tree displays any of these tags.

	TAG	Description
	CHECK	Points out that something needs to be checked or tested.
	TEMP	A temporary piece of code to be deleted in the short term.
	REVIEW	Calls for something’s critical in an improvement aim.




Records Tag Category

Tags that are used to produce to-do lists of different natures. Tags from this category are mostly used to keep track of the project’s progress and to make sure that none of the features expected to be part of the next release is forgotten. For the project to be considered as ready for release, all tags from this category should be either addressed or moved to the *hunches* category. As a result, while this category is expected to be the most densely populated one among the generated tree at the heart of every development phase, it should vanish completely when approaching a release.

	TAG	Description
	TEST	Documents a non-critical task that still needs to be done.
	TODO	Documents a non-critical task that still needs to be done.
	IMPROVE	To Be Done: something that has not even been started yet




Hunches Tag Category

Low-priority tags refer to modifications that are not meant to be part of the next release, whether because they are considered to be non-essential, or because the underlying features are not advanced enough to qualify for inclusion. This category also includes all the tags referencing changes that have been planned (independently of their status) as well as the eventual mentions of ideas or proposals that have not yet been evaluated. Tags from this category have no impact on the project’s readiness for release, the reason why the extension may be safely configured to hide them whenever the tree is generated from outside the development environment.

	TAG	Description
	IDEA	Keeps track of a suggestion seemingly worth a consideration.
	LEARN	A topic of interest that should be studied, an article to read, etc ...
	ROADMAP	To Be Done: something that has not even been started yet



Markers Tag Category

Tags identifying editable parts of the project, either by delimiting customizable areas or by being themselves placeholders meant to be replaced. Tags from this category typically call for actions on a template that are meant to happen at a pre-determined point in the project's life cycle. Depending on its nature, each of these tags may be repeatedly needed and, thus, persist throughout all project's life cycles, or serve only once and then be removed. As a result, whether or not their presence impacts the project's release readiness must be evaluated on a case-by-case basis. It is however very unlikely that any of them should be part of the project's final distribution.

	TAG	Description
	@@	Prefixes keywords that are placeholders for customizable values
	OPTION	Highlights a chunk of code that is one among other alternatives
	EDIT	Highlights the beginning of an editable chunk of code like a template's customizable area.







Helpers Tag Category

Tags which are used to hold information about the project that is central enough to its development that it is worth using the tree to keep track of them. In the same aim, tags from this category may also be used to bookmark locations in the project that are of particular interest and, thus, need to be accessed frequently and/or simply as a reminder for some easily forgettable action that is important to be taken. These tags' lifetime typically spans over the whole development and has no impact on whether the project can be released or not. As a consequence, the extension should be configured to hide this category's tags whenever generating the tree from the release or the deployment environment.

	TAG	Description
	LINK	Serves as shortcut to an editable region within, typically, templates
	NOTE	A general note or comment calling for no action of any kind
	CAUTION	A general note or comment calling for no action of any kind

Dividers Tag Category

Tags that are used to visually segment the project's file. The extension is configured to provide 6 levels of sectioning, starting with *parts* and *chapters* and ending with *sub-sub-sub-sections*. Obviously, tags from this category call for no action at all and are intrinsically meant to never be deleted; their number even has a vocation to increase with the size of the project and, thus, over time. It is worth noting that the case of *dividers* is in complete contrast to this of *helpers*. Indeed, unlike *helpers*, *dividers* have no interest at all in the listing ability of the extension. Instead, they benefit from the fact that tags can be highlighted within the project's source files.

	TAG	Description
	PART	Whole-line speparator.
	CHAPTER	A chapter using the <i>line</i> setting.
	SECTION	A section using the <i>text-and-comment</i> setting.
	SUBSECTION	A sub-section using the <i>tag-and-comment</i> setting.
	BLOCK	A paragraph or any enumeration like, typically, a list.
	ITEM	Highlights a given element in a block like, typically a list entry.