Microsoft Visual Studio (July 2023)

Basic Editing

Visual Studio Code is an editor first and foremost, and includes the features you need for highly productive source code editing. This topic takes you through the basics of the editor and helps you get moving with your code.

Keyboard shortcuts

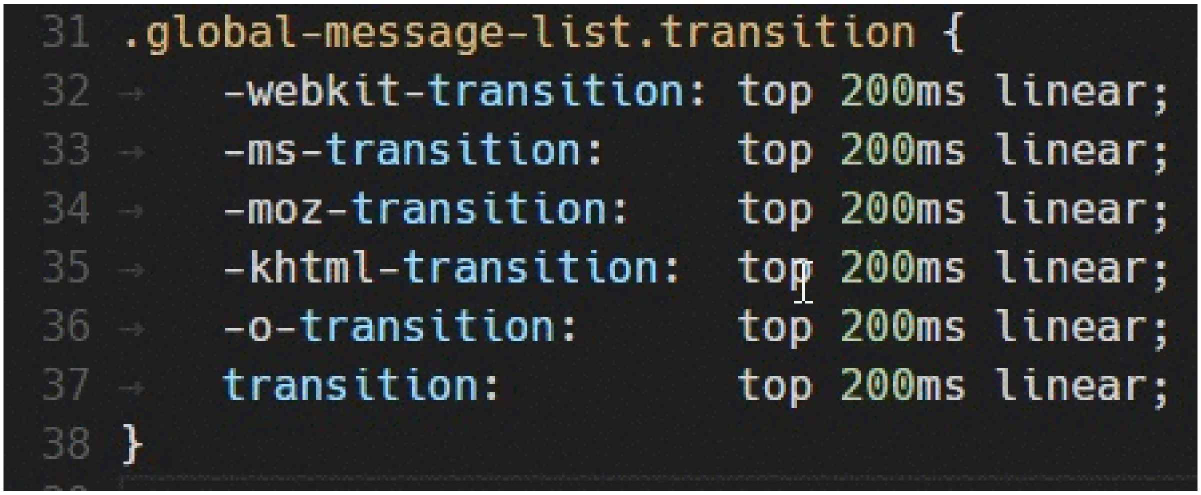
Being able to keep your hands on the keyboard when writing code is crucial for high productivity. VS Code has a rich set of default keyboard shortcuts as well as allowing you to customize them.

- Keyboard Shortcuts Reference - Learn the most commonly used and popular keyboard shortcuts by downloading the reference sheet.
- Install a Keymap extension - Use the keyboard shortcuts of your old editor (such as Sublime Text, Atom, and Vim) in VS Code by installing a Keymap extension.
- Customize Keyboard Shortcuts - Change the default keyboard shortcuts to fit your style.

Multiple selections (multi-cursor)

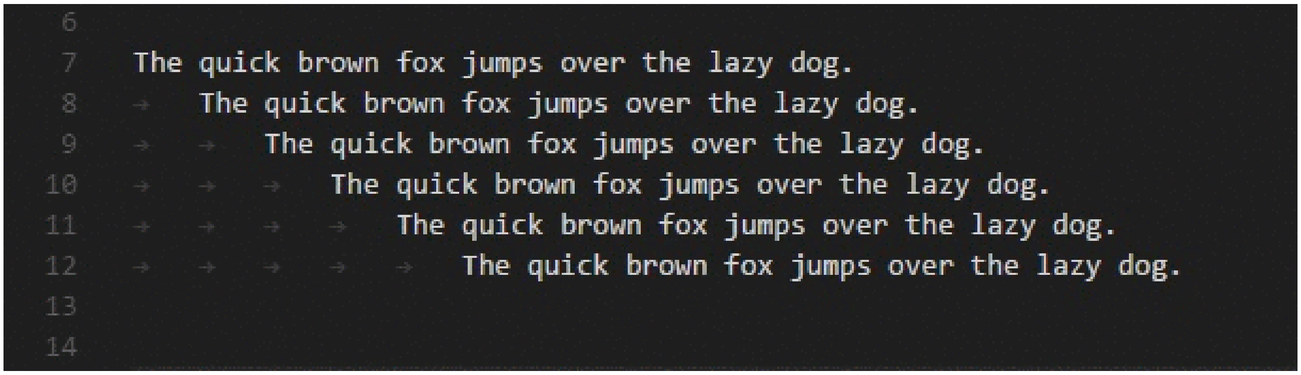
VS Code supports multiple cursors for fast simultaneous edits. You can add secondary cursors (rendered thinner) with `Alt+Click`. Each cursor operates independently based on the context it sits in. A common way to add more cursors is with `Ctrl+Alt+Down` or `Ctrl+Alt+Up` that insert cursors below or above.

Note: Your graphics card driver (for example NVIDIA) might overwrite these default shortcuts.



```
.global-message-list.transition {
    -webkit-transition: top 200ms linear;
    -ms-transition:     top 200ms linear;
    -moz-transition:    top 200ms linear;
    -khtml-transition:  top 200ms linear;
    -o-transition:      top 200ms linear;
    transition:         top 200ms linear;
}
```

`Ctrl+D` selects the word at the cursor, or the next occurrence of the current selection.



```

The quick brown fox jumps over the lazy dog.
→   The quick brown fox jumps over the lazy dog.
→   →   The quick brown fox jumps over the lazy dog.
→   →   →   The quick brown fox jumps over the lazy dog.
→   →   →   →   The quick brown fox jumps over the lazy dog.
→   →   →   →   →   The quick brown fox jumps over the lazy dog.


```

Tip: You can also add more cursors with `Ctrl+Shift+L`, which will add a selection at each occurrence of the current selected text.

Multi-cursor modifier

If you'd like to change the modifier key for applying multiple cursors to `Cmd+Click` on macOS and `Ctrl+Click` on Windows and Linux, you can do so with the `editor.multiCursorModifier` setting. This lets users coming from other editors such as Sublime Text or Atom continue to use the keyboard modifier they are familiar with.

The setting can be set to:

- `ctrlCmd` - Maps to `Ctrl` on Windows and `Cmd` on macOS.
- `alt` - The existing default `Alt`.

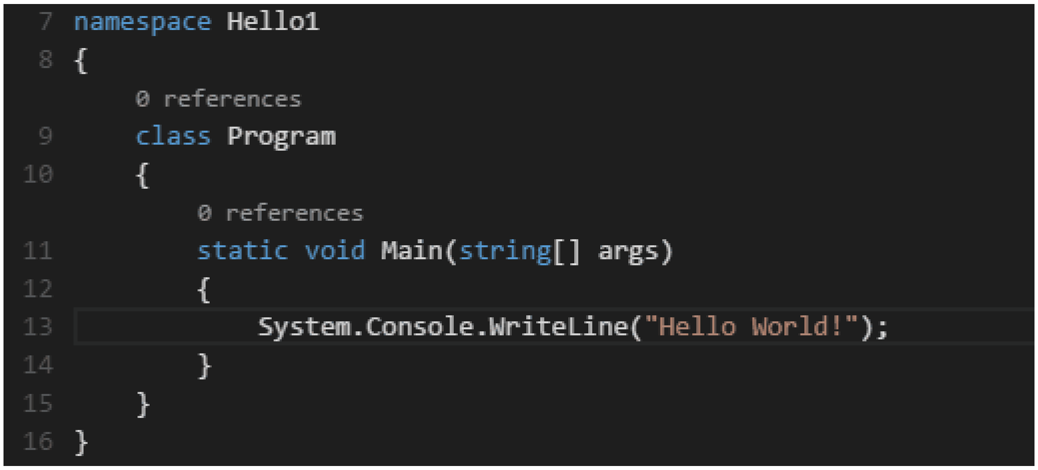
There's also a menu item **Use Ctrl+Click for Multi-Cursor** in the **Selection** menu to quickly toggle this setting.

The **Go to Definition** and **Open Link** gestures will also respect this setting and adapt such that they do not conflict. For example, when the setting is `ctrlCmd`, multiple cursors can be added with `Ctrl/Cmd+Click`, and opening links or going to definition can be invoked with `Alt+Click`.

Shrink/expand selection

Quickly shrink or expand the current selection. Trigger it with `Shift+Alt+Left` and `Shift+Alt+Right`.

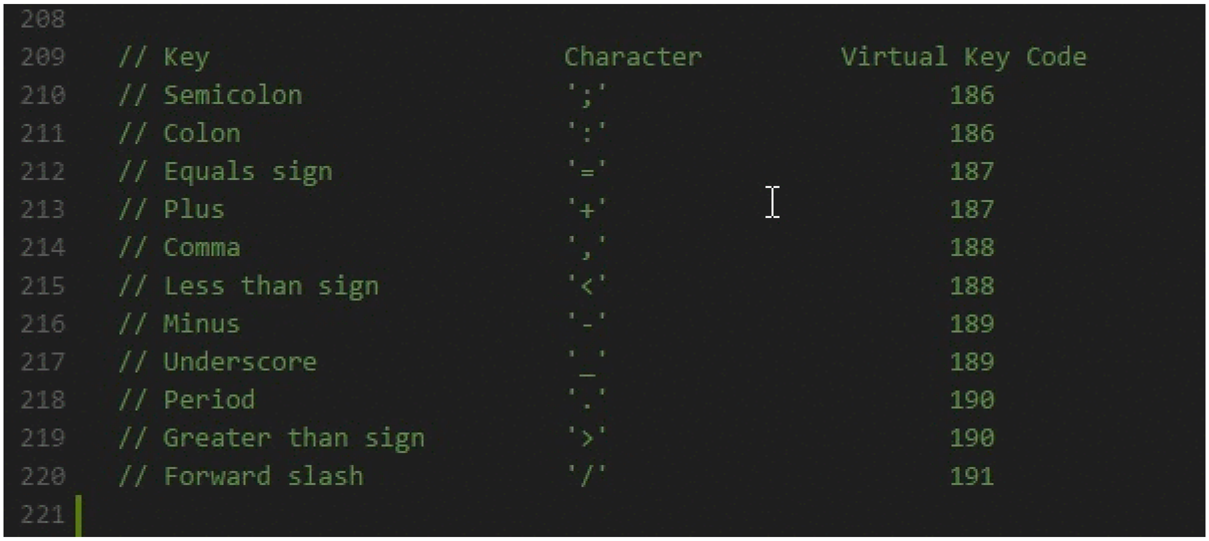
Here's an example of expanding the selection with `Shift+Alt+Right`:



```
namespace Hello1
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            System.Console.WriteLine("Hello World!");
        }
    }
}
```

Column (box) selection

Place the cursor in one corner and then hold `Shift+Alt` while dragging to the opposite corner:



```

// Key                        Character         Virtual Key Code
// Semicolon                  ';'                      186
// Colon                      ':'                      186
// Equals sign                '='                      187
// Plus                       '+'                      187
// Comma                      ','                      188
// Less than sign             '<'                      188
// Minus                      '-'                      189
// Underscore                 '_'                      189
// Period                     '.'                      190
// Greater than sign          '>'                      190
// Forward slash              '/'                      191

```

Note: This changes to `Shift+Ctrl/Cmd` when using `Ctrl/Cmd` as multi-cursor modifier.

There are also default key bindings for column selection on macOS and Windows, but not on Linux.

Key	**Command**	**Command ID**
`Ctrl+Shift+Alt+Down`	Column Select Down	`cursorColumnSelectDown`
`Ctrl+Shift+Alt+Up`	Column Select Up	`cursorColumnSelectUp`
`Ctrl+Shift+Alt+Left`	Column Select Left	`cursorColumnSelectLeft`
`Ctrl+Shift+Alt+Right`	Column Select Right	`cursorColumnSelectRight`
`Ctrl+Shift+Alt+PageDown`	Column Select Page Down	`cursorColumnSelectPageDown`
`Ctrl+Shift+Alt+PageUp`	Column Select Page Up	`cursorColumnSelectPageUp`

You can edit your `keybindings.json` to bind them to something more familiar if you want.

Column Selection mode

The user setting **Editor: Column Selection** controls this feature. Once this mode is entered, as indicated in the Status bar, the mouse gestures and the arrow keys will create a column selection by default. This global toggle is also accessible via the **Selection** > **Column Selection Mode** menu item. In addition, one can also disable Column Selection mode from the Status bar.

Save / Auto Save

By default, VS Code requires an explicit action to save your changes to disk, `Ctrl+S`.

However, it's easy to turn on `Auto Save`, which will save your changes after a configured delay or when focus leaves the editor. With this option turned on, there is no need to explicitly save the file. The easiest way to turn on `Auto Save` is with the **File** > **Auto Save** toggle that turns on and off save after a delay.

For more control over `Auto Save`, open User or Workspace settings and find the associated settings:

- `files.autoSave`: Can have the values:
  - `off` - to disable auto save.
  - `afterDelay` - to save files after a configured delay (default 1000 ms).
  - `onFocusChange` - to save files when focus moves out of the editor of the dirty file.
  - `onWindowChange` - to save files when the focus moves out of the VS Code window.
- `files.autoSaveDelay`: Configures the delay in milliseconds when `files.autoSave` is configured to `afterDelay`. The default is 1000 ms.

Hot Exit

VS Code will remember unsaved changes to files when you exit by default. Hot exit is triggered when the application is closed via **File** > **Exit** (**Code** > **Quit** on macOS) or when the last window is closed.

You can configure hot exit by setting `files.hotExit` to the following values:

- `"off"`: Disable hot exit.
- `"onExit"`: Hot exit will be triggered when the application is closed, that is when the last window is closed on Windows/Linux or when the `workbench.action.quit` command is triggered (from the **Command Palette**, keyboard shortcut or menu). All windows without folders opened will be restored upon next launch.
- `"onExitAndWindowClose"`: Hot exit will be triggered when the application is closed, that is when the last window is closed on Windows/Linux or when the `workbench.action.quit` command is triggered (from the **Command Palette**, keyboard shortcut or menu), and also for any window with a folder opened regardless of whether it is the last window. All windows without folders opened will be restored upon next launch. To restore folder windows as they were before shutdown, set `window.restoreWindows` to `all`.

If something happens to go wrong with hot exit, all backups are stored in the following folders for standard install locations:

- **Windows** `%APPDATA%\Code\Backups`
- **macOS** `$HOME/Library/Application Support/Code/Backups`

- **Linux** `$HOME/.config/Code/Backups`

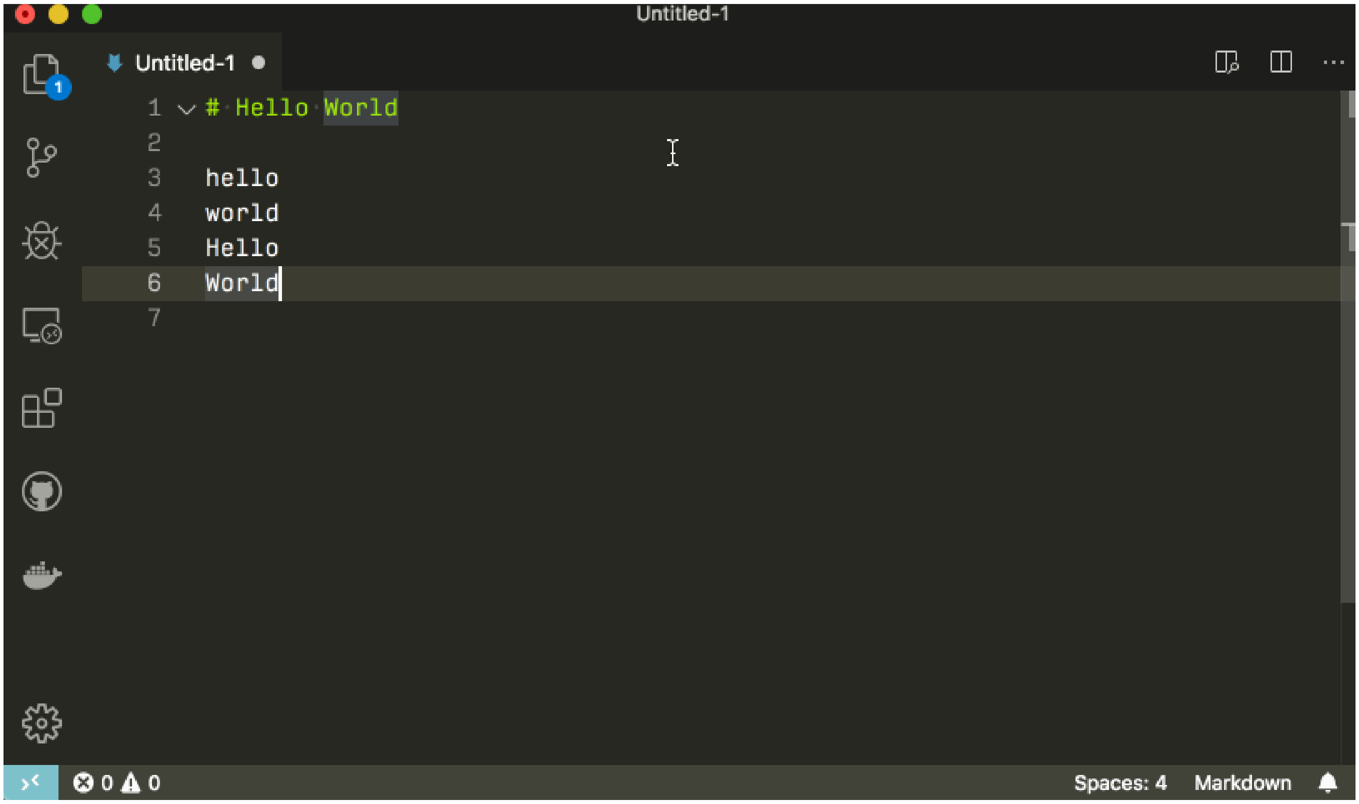
Find and Replace

VS Code allows you to quickly find text and replace in the currently opened file. Press `Ctrl+F` to open the Find Widget in the editor, search results will be highlighted in the editor, overview ruler and minimap.

If there are more than one matched result in the current opened file, you can press `Enter` and `Shift+Enter` to navigate to next or previous result when the find input box is focused.

Seed Search String From Selection

When the Find Widget is opened, it will automatically populate the selected text in the editor into the find input box. If the selection is empty, the word under the cursor will be inserted into the input box instead.

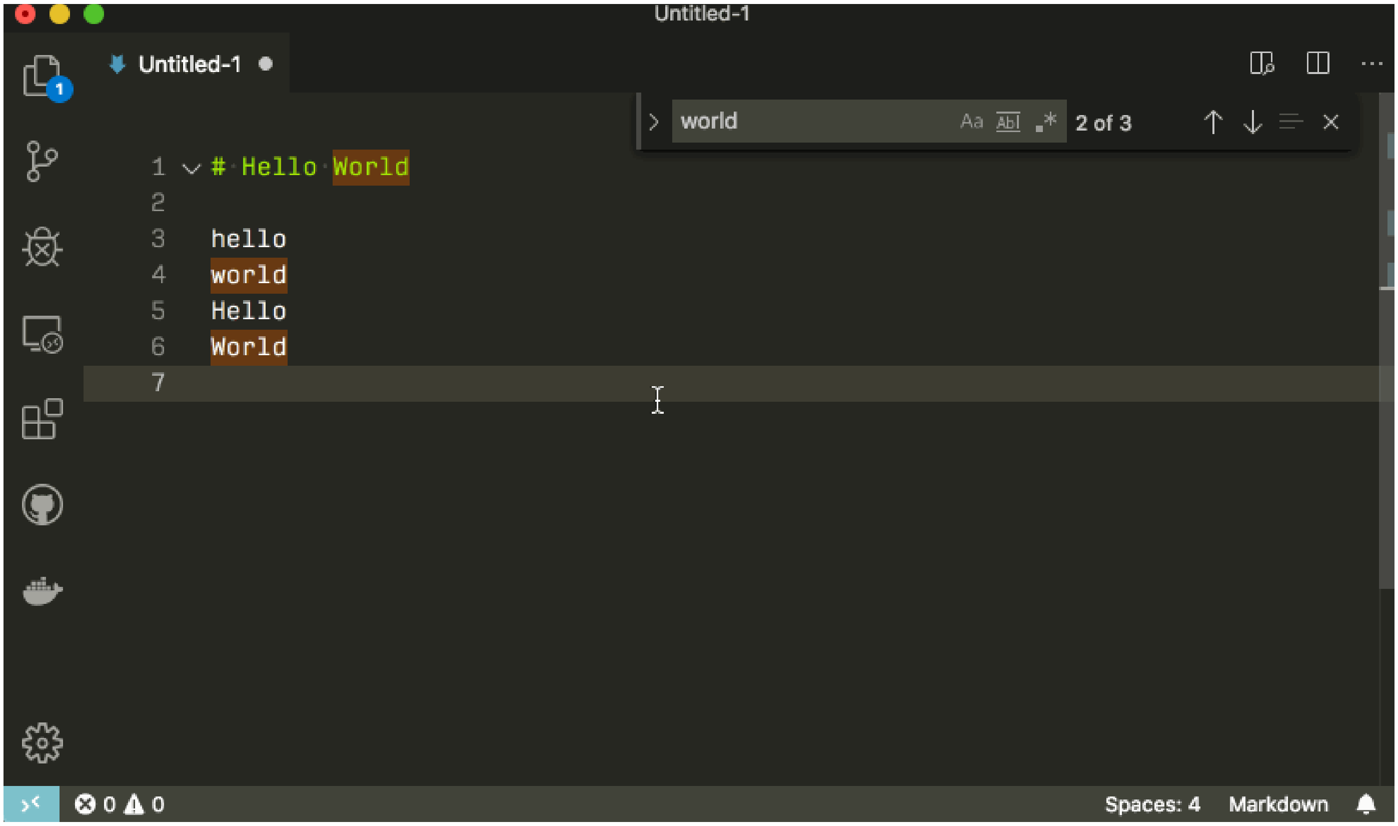


```
# Hello World

hello
world
Hello
World
```

This feature can be turned off by setting `editor.find.seedSearchStringFromSelection` to `false`.

Find In Selection

By default, the find operations are run on the entire file in the editor. It can also be run on selected text. You can turn this feature on by clicking the hamburger icon on the Find Widget.



```
# Hello World

hello
world
Hello
World

```

If you want it to be the default behavior of the Find Widget, you can set `editor.find.autoFindInSelection` to `always`, or to `multiline`, if you want it to be run on selected text only when multiple lines of content are selected.

Advanced find and replace options

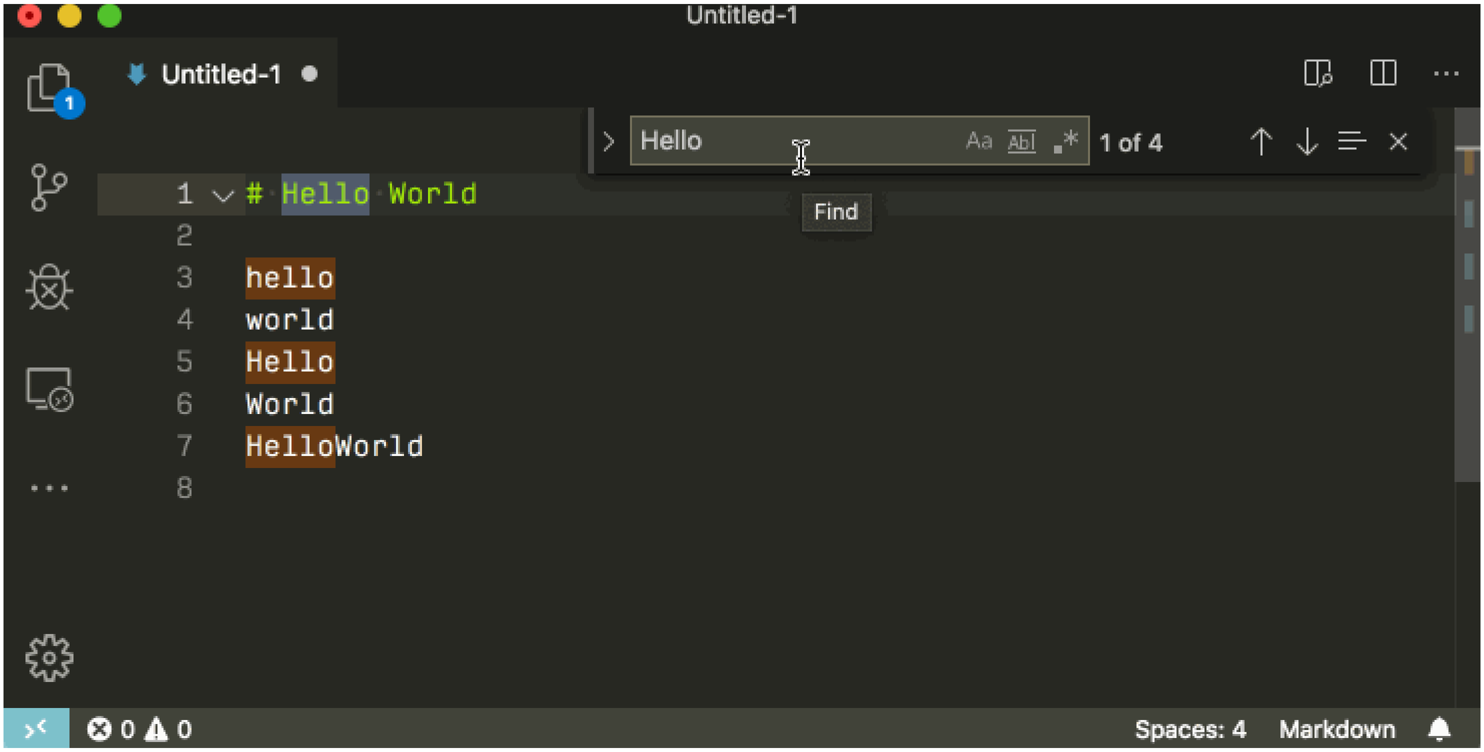
In addition to find and replace with plain text, the Find Widget also has three advanced search options:

- Match Case
- Match Whole Word
- Regular Expression

The replace input box support case preserving, you can turn it on by clicking the Preserve Case (**AB**) button.

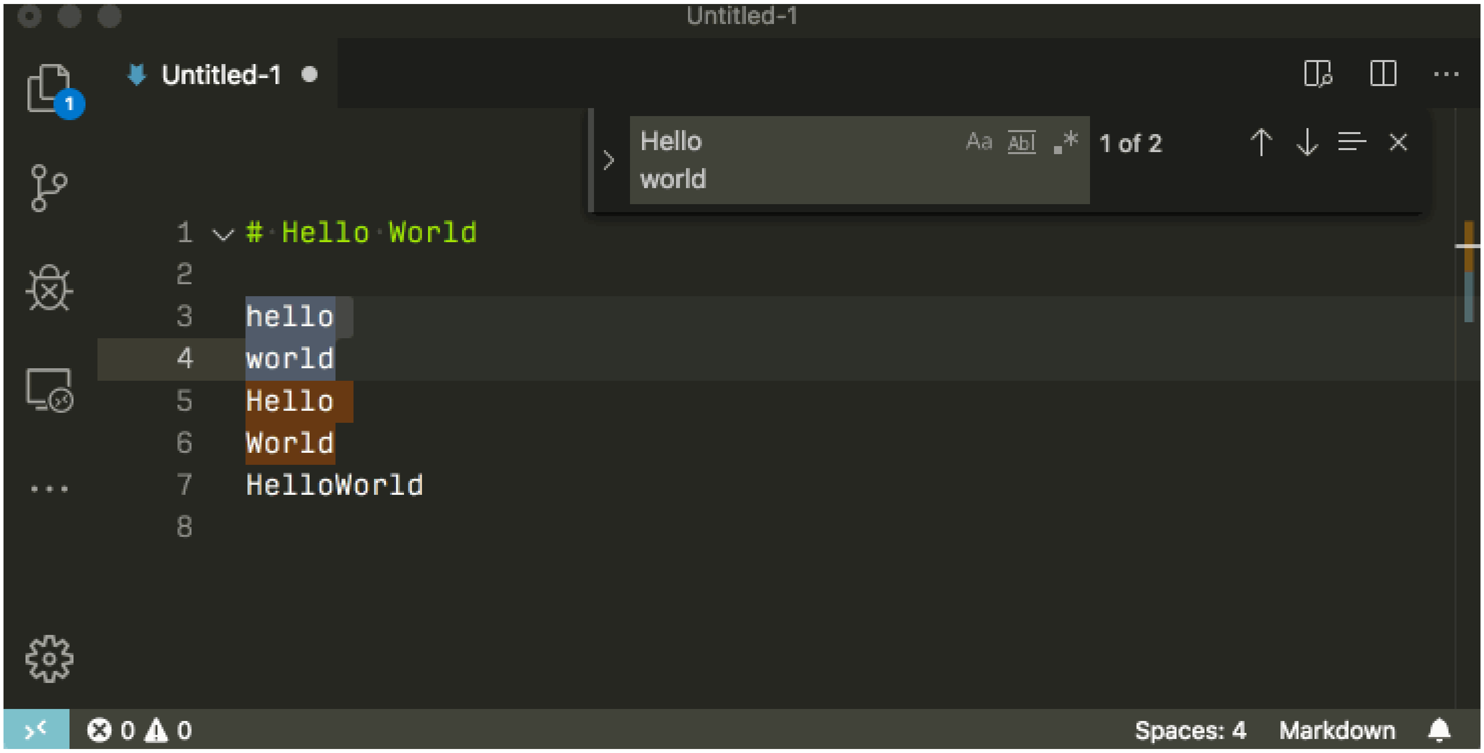
Multiline support and Find Widget resizing

You can search multiple line text by pasting the text into the Find input box and Replace input box. Pressing `Ctrl+Enter` inserts a new line in the input box.



```
# Hello World

hello
world
Hello
World
HelloWorld
```

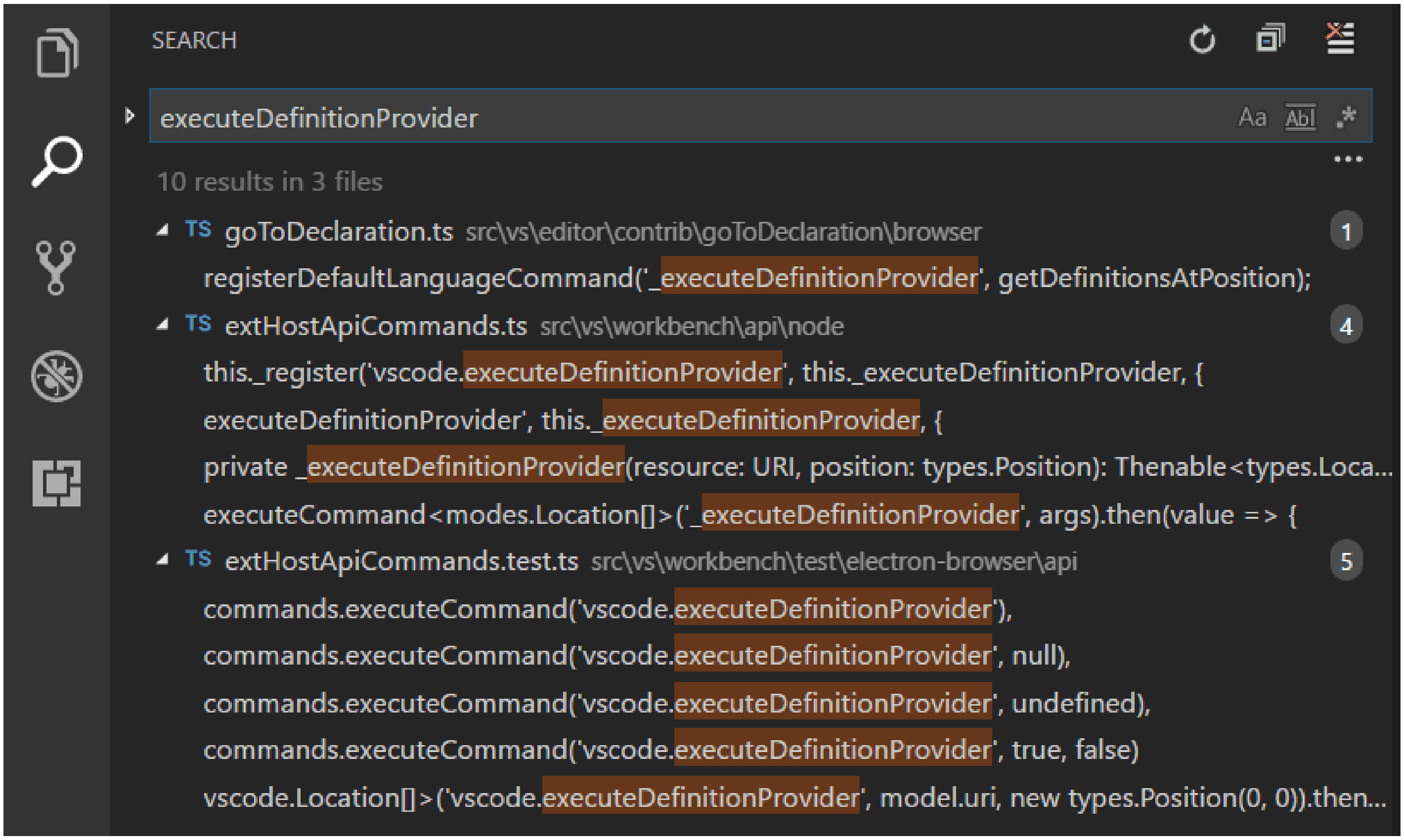
While searching long text, the default size of Find Widget might be too small. You can drag the left sash to enlarge the Find Widget or double click the left sash to maximize it or shrink it to its default size.



```
# Hello World

hello
world
Hello
World
HelloWorld
```

Search across files

VS Code allows you to quickly search over all files in the currently opened folder. Press `Ctrl+Shift+F` and enter your search term. Search results are grouped into files containing the search term, with an indication of the hits in each file and its location. Expand a file to see a preview of all of the hits within that file. Then single-click on one of the hits to view it in the editor.

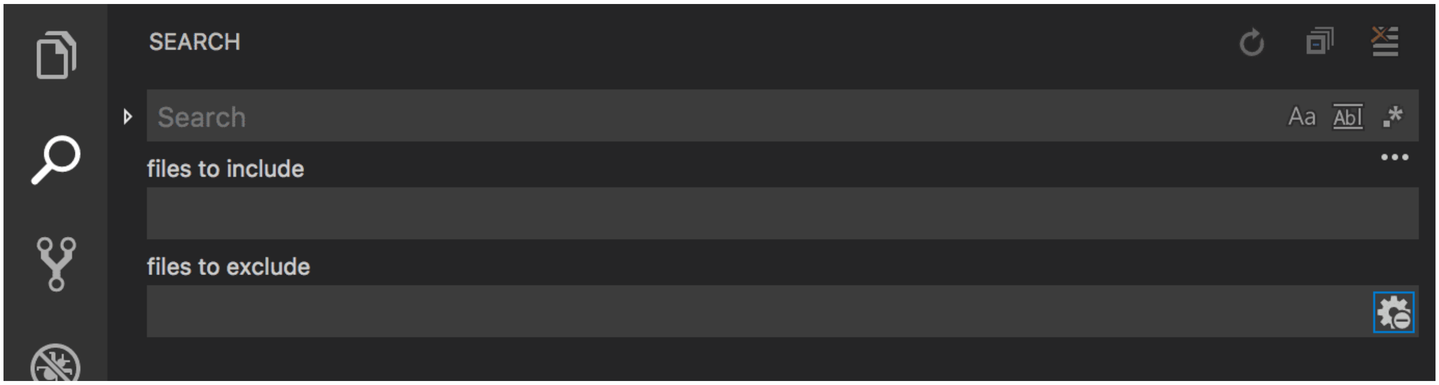


```
executeDefinitionProvider
10 results in 3 files
TS goToDeclaration.ts src\vs\editor\contrib\goToDeclaration\browser 1
    registerDefaultLanguageCommand('_executeDefinitionProvider', getDefinitionsAtPosition);
TS extHostApiCommands.ts src\vs\workbench\api\node 4
    this._register('vscode.executeDefinitionProvider', this._executeDefinitionProvider, {
    executeDefinitionProvider', this._executeDefinitionProvider, {
    private _executeDefinitionProvider(resource: URI, position: types.Position): Thenable<types.Loca...
    executeCommand<modes.Location[]>('_executeDefinitionProvider', args).then(value => {
TS extHostApiCommands.test.ts src\vs\workbench\test\electron-browser\api 5
    commands.executeCommand('vscode.executeDefinitionProvider'),
    commands.executeCommand('vscode.executeDefinitionProvider', null),
    commands.executeCommand('vscode.executeDefinitionProvider', undefined),
    commands.executeCommand('vscode.executeDefinitionProvider', true, false)
    vscode.Location[]>('vscode.executeDefinitionProvider', model.uri, new types.Position(0, 0)).then...
```

Tip: We support regular expression searching in the search box, too.

You can configure advanced search options by clicking the ellipsis (**Toggle Search Details**) below the search box on the right (or press `Ctrl+Shift+J`). This will show additional fields to configure the search.

Advanced search options



In the two input boxes below the search box, you can enter patterns to include or exclude from the search. If you enter `example`, that will match every folder and file named `example` in the workspace. If you enter `./example`, that will match the folder `example/` at the top level of your workspace. Use `,` to separate multiple patterns. Paths must use forward slashes. You can also use glob pattern syntax, for example:

- `*` to match zero or more characters in a path segment
- `?` to match on one character in a path segment
- `**` to match any number of path segments, including none
- `{}` to group conditions (for example `{**/*.html,**/*.txt}` matches all HTML and text files)
- `[]` to **declare** a range of characters to match (`example.[0-9]` to match on `example.0`, `example.1`, …)
- `[!...]` to negate a range of characters to match (`example.[!0-9]` to match on `example.a`, `example.b`, but not `example.0`)

VS Code excludes some folders by default to reduce the number of search results that you are not interested in (for example: `node_modules`). Open settings to change these rules under the `files.exclude` and `search.exclude` section.

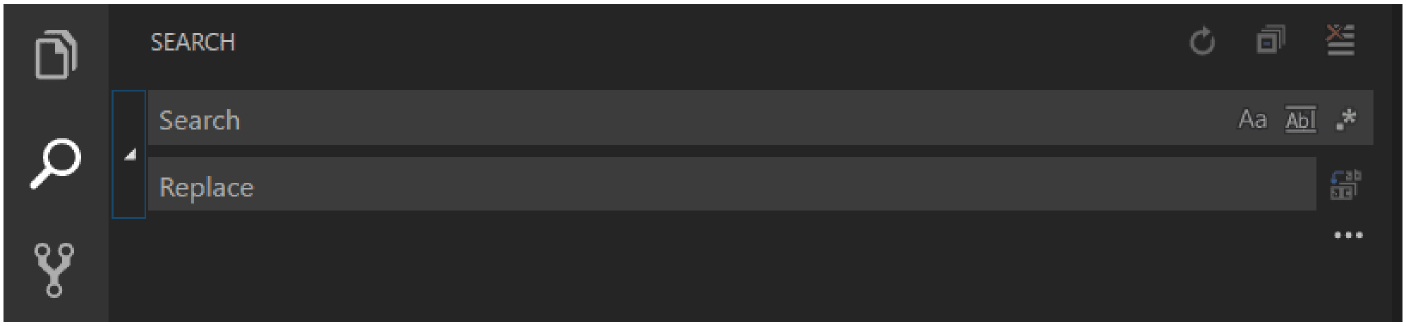
Note that glob patterns in the Search view work differently than in settings such as `files.exclude` and `search.exclude`. In the settings, you must use `**/example` to match a folder named `example` in subfolder `folder1/example` in your workspace. In the Search view, the `**` prefix is assumed. The glob patterns in these settings are always evaluated relative to the path of the workspace folder.

Also note the **Use Exclude Settings and Ignore Files** toggle button in the **files to exclude** box. The toggle determines whether to exclude files that are ignored by your `.gitignore` files and/or matched by your `files.exclude` and `search.exclude` settings.

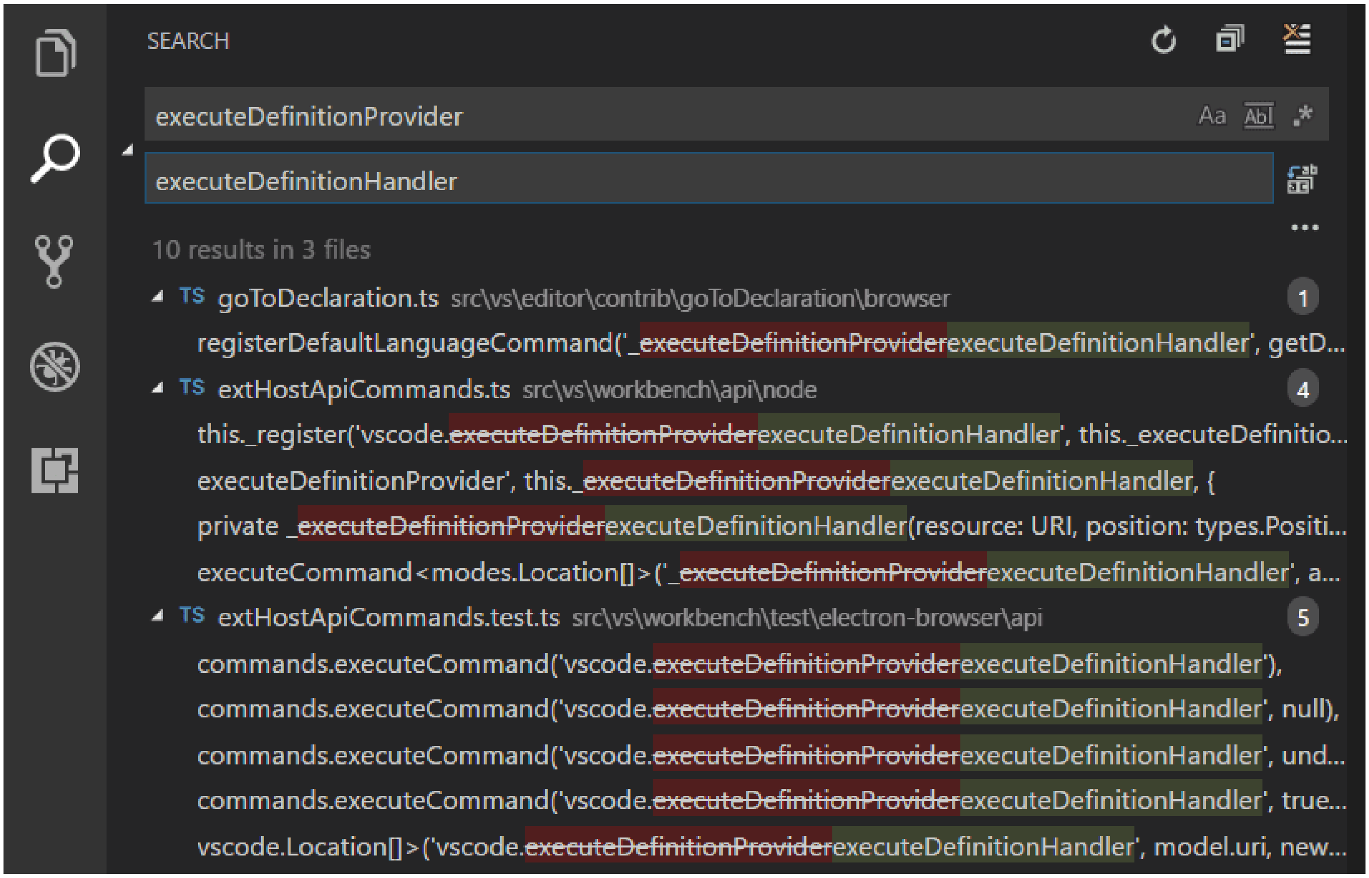
Tip: From the Explorer, you can right-click on a folder and select **Find in Folder** to search inside a folder only.

Search and replace

You can also Search and Replace across files. Expand the Search widget to display the Replace text box.



When you type text into the Replace text box, you will see a diff display of the pending changes. You can replace across all files from the Replace text box, replace all in one file or replace a single change.



executeDefinitionProvider

executeDefinitionHandler

10 results in 3 files

TS goToDeclaration.ts src\vs\editor\contrib\goToDeclaration\browser 1
registerDefaultLanguageCommand('~~_executeDefinitionProvider~~executeDefinitionHandler', getD...

TS extHostApiCommands.ts src\vs\workbench\api\node 4
this._register('vscode.~~executeDefinitionProvider~~executeDefinitionHandler', this._executeDefinitio...
executeDefinitionProvider', this._~~executeDefinitionProvider~~executeDefinitionHandler, {
private _~~executeDefinitionProvider~~executeDefinitionHandler(resource: URI, position: types.Positi...
executeCommand<modes.Location[]>('_~~executeDefinitionProvider~~executeDefinitionHandler', a...

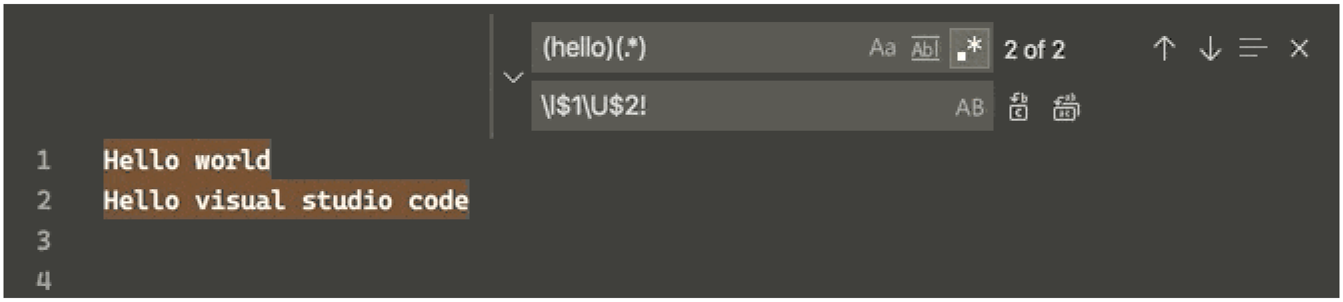
TS extHostApiCommands.test.ts src\vs\workbench\test\electron-browser\api 5
commands.executeCommand('vscode.~~executeDefinitionProvider~~executeDefinitionHandler'),
commands.executeCommand('vscode.~~executeDefinitionProvider~~executeDefinitionHandler', null),
commands.executeCommand('vscode.~~executeDefinitionProvider~~executeDefinitionHandler', und...
commands.executeCommand('vscode.~~executeDefinitionProvider~~executeDefinitionHandler', true...
vscode.Location[]>('vscode.~~executeDefinitionProvider~~executeDefinitionHandler', model.uri, new...

Tip: You can quickly reuse a previous search term by using `Down` and `Up` to navigate through your search term history.

Case changing in regex replace

VS Code supports changing the case of regex matching groups while doing Search and Replace in the editor or globally. This is done with the modifiers `\u\U\l\L`, where `\u` and `\l` will upper/lowercase a single character, and `\U` and `\L` will upper/lowercase the rest of the matching group.

Example:



(hello)(.*)

\l$1\U$2!

2 of 2

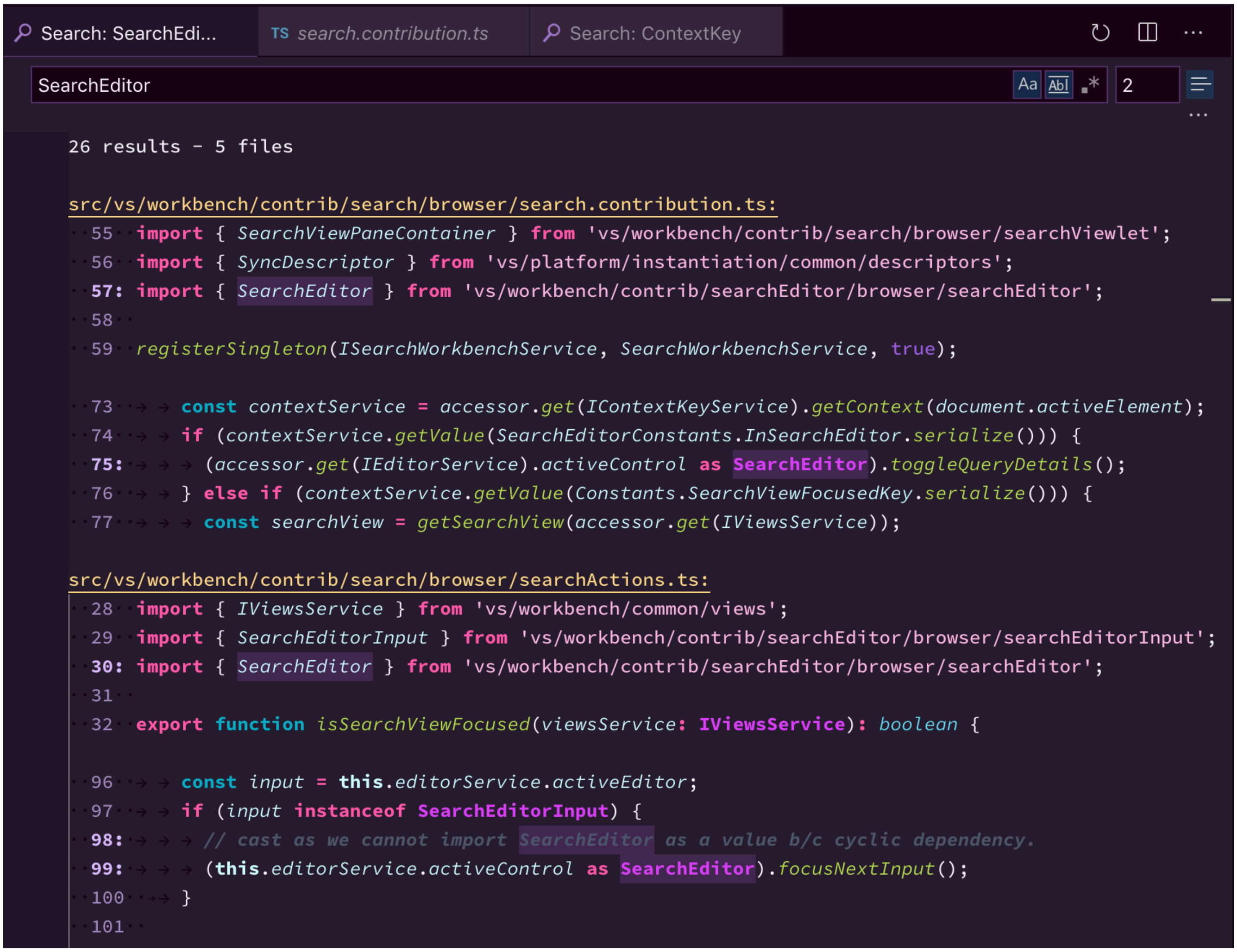
```
Hello world
Hello visual studio code
```

The modifiers can also be stacked - for example, `\u\u\u$1` will uppercase the first three characters of the group, or `\l\U$1` will lowercase the first character, and uppercase the rest. The capture group is referenced by `$n` in the replacement string, where `n` is the order of the capture group.

Search Editor

Search Editors let you view workspace search results in a full-sized editor, complete with syntax highlighting and optional lines of surrounding context.

Below is a search for the word 'SearchEditor' with two lines of text before and after the match for context:



SearchEditor

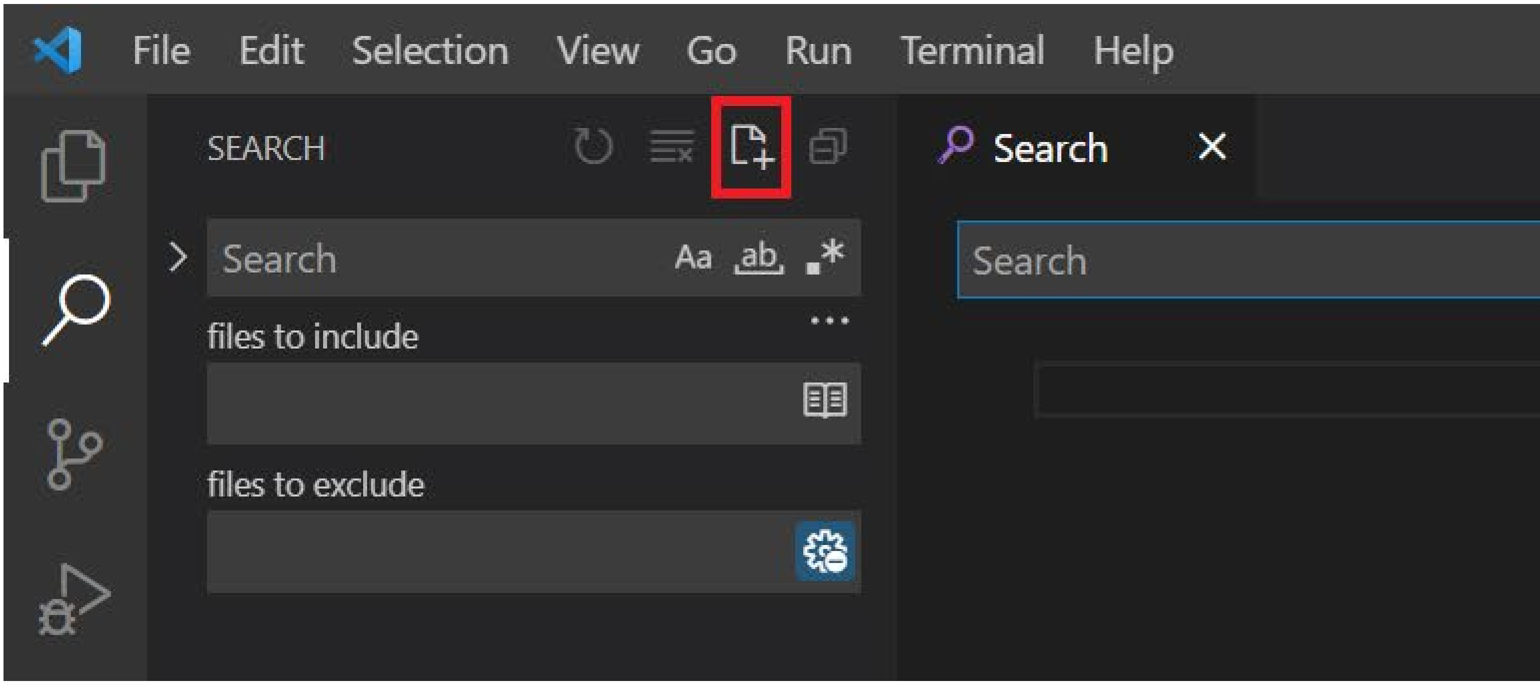
2

```
26 results - 5 files

src/vs/workbench/contrib/search/browser/search.contribution.ts:
  55  import { SearchViewPaneContainer } from 'vs/workbench/contrib/search/browser/searchViewlet';
  56  import { SyncDescriptor } from 'vs/platform/instantiation/common/descriptors';
  57: import { SearchEditor } from 'vs/workbench/contrib/searchEditor/browser/searchEditor';
  58
  59  registerSingleton(ISearchWorkbenchService, SearchWorkbenchService, true);

  73          const contextService = accessor.get(IContextKeyService).getContext(document.activeElement);
  74          if (contextService.getValue(SearchEditorConstants.InSearchEditor.serialize())) {
  75:             (accessor.get(IEditorService).activeControl as SearchEditor).toggleQueryDetails();
  76          } else if (contextService.getValue(Constants.SearchViewFocusedKey.serialize())) {
  77              const searchView = getSearchView(accessor.get(IViewsService));

src/vs/workbench/contrib/search/browser/searchActions.ts:
  28  import { IViewsService } from 'vs/workbench/common/views';
  29  import { SearchEditorInput } from 'vs/workbench/contrib/searchEditor/browser/searchEditorInput';
  30: import { SearchEditor } from 'vs/workbench/contrib/searchEditor/browser/searchEditor';
  31
  32  export function isSearchViewFocused(viewsService: IViewsService): boolean {

  96          const input = this.editorService.activeEditor;
  97          if (input instanceof SearchEditorInput) {
  98:             // cast as we cannot import SearchEditor as a value b/c cyclic dependency.
  99:             (this.editorService.activeControl as SearchEditor).focusNextInput();
  100         }
  101
```

The **Open Search Editor** command opens an existing Search Editor if one exists, or to otherwise create a new one. The **New Search Editor** command will always create a new Search Editor.

In the Search Editor, results can be navigated to using **Go to Definition** actions, such as `F12` to open the source location in the current editor group, or `Ctrl+K F12` to open the location in an editor to the side. Additionally, double-clicking can optionally open the source location, configurable with the `search.searchEditor.doubleClickBehaviour` setting.

You can also use the **Open New Search Editor** button at the top of the Search view, and can copy your existing results from a Search view over to a Search Editor with the **Open in editor** link at the top of the results tree, or the **Search Editor: Open Results in Editor** command.



The Search Editor above was opened by selecting the **Open New Search Editor** button (third button) on the top of the Search view.

Search Editor commands and arguments

- `search.action.openNewEditor` - Opens the Search Editor in a new tab.
- `search.action.openInEditor` - Copy the current Search results into a new Search Editor.
- `search.action.openNewEditorToSide` - Opens the Search Editor in a new window next to the window you currently have opened.

There are two arguments that you can pass to the Search Editor commands (`search.action.openNewEditor`, `search.action.openNewEditorToSide`) to allow keybindings to configure how a new Search Editor should behave:

- `triggerSearch` - Whether a search be automatically run when a Search Editor is opened. Default is true.
- `focusResults` - Whether to put focus in the results of a search or the query input. Default is true.

For example, the following keybinding runs the search when the Search Editor is opened but leaves the focus in the search query control.

```
{

  "key": "ctrl+o",

  "command": "search.action.openNewEditor",

  "args": { "query": "VS Code", "triggerSearch": true, "focusResults": false }
}
```

Search Editor context default

The `search.searchEditor.defaultNumberOfContextLines` setting has a default value of 1, meaning one context line will be shown before and after each result line in the Search Editor.

Reuse last Search Editor configuration

The `search.searchEditor.reusePriorSearchConfiguration` setting (default is `false`) lets you reuse the last active Search Editor's configuration when creating a new Search Editor.

IntelliSense

We'll always offer word completion, but for the rich languages, such as JavaScript, JSON, HTML, CSS, SCSS, Less, C# and TypeScript, we offer a true IntelliSense experience. If a language service knows possible completions, the IntelliSense suggestions will pop up as you type. You can always manually trigger it with `Ctrl+Space`. By default, `Tab` or `Enter` are the accept keyboard triggers but you can also customize these key bindings.

Tip: The suggestions filtering supports CamelCase so you can type the letters which are upper cased in a method name to limit the suggestions. For example, "cra" will quickly bring up "createApplication".

Tip: IntelliSense suggestions can be configured via the `editor.quickSuggestions` and `editor.suggestOnTriggerCharacters` settings.

JavaScript and TypeScript developers can take advantage of the npmjs type declaration (typings) file repository to get IntelliSense for common JavaScript libraries (Node.js, React, Angular). You can find a good explanation on using type declaration files in the JavaScript language topic and the Node.js tutorial.

Learn more in the IntelliSense document.

Formatting

VS Code has great support for source code formatting. The editor has two explicit format actions:

- **Format Document** (`Shift+Alt+F`) - Format the entire active file.
- **Format Selection** (`Ctrl+K Ctrl+F`) - Format the selected text.

You can invoke these from the **Command Palette** (`Ctrl+Shift+P`) or the editor context menu.

VS Code has default formatters for JavaScript, TypeScript, JSON, HTML, and CSS. Each language has specific formatting options (for example, `html.format.indentInnerHtml`) which you can tune to your preference in your user or workspace settings. You can also disable the default language formatter if you have another extension installed that provides formatting for the same language.

```
"html.format.enable": false
```

Along with manually invoking code formatting, you can also trigger formatting based on user gestures such as typing, saving or pasting. These are off by default but you can enable these behaviors through the following settings:

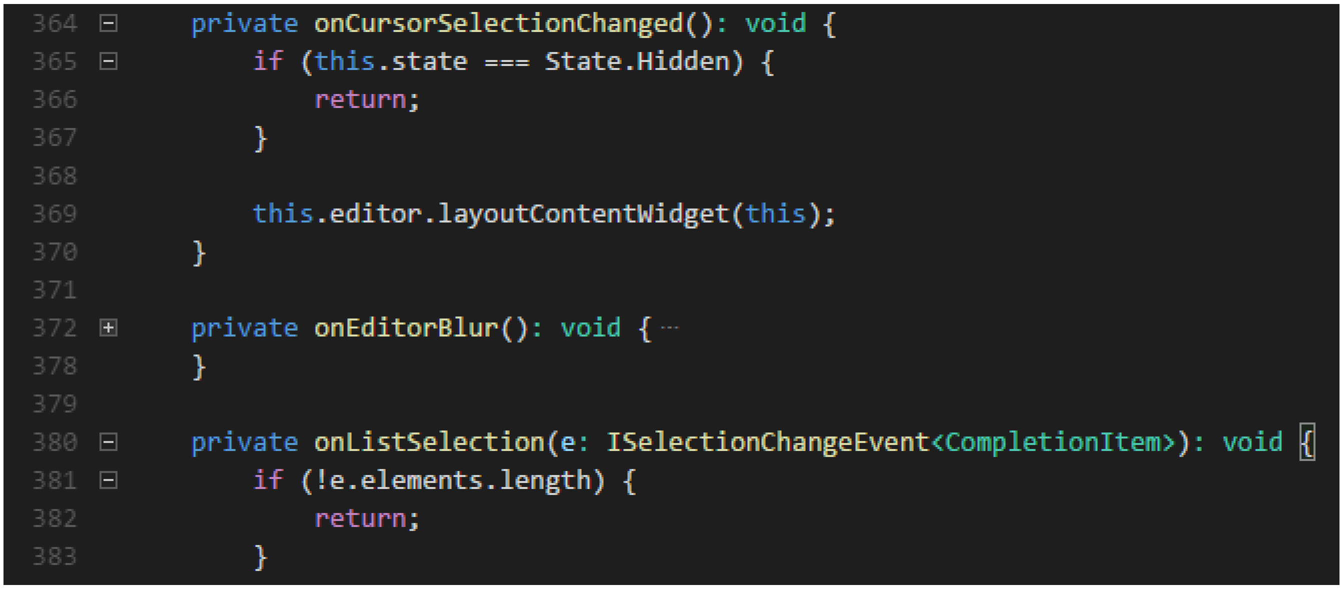
- `editor.formatOnType` - Format the line after typing.
- `editor.formatOnSave` - Format a file on save.
- `editor.formatOnPaste` - Format the pasted content.

Note: Not all formatters support format on paste as to do so they must support formatting a selection or range of text.

In addition to the default formatters, you can find extensions on the Marketplace to support other languages or formatting tools. There is a `Formatters` category so you can easily search and find formatting extensions. In the **Extensions** view search box, type 'formatters' or 'category:formatters' to see a filtered list of extensions within VS Code.

Folding

You can fold regions of source code using the folding icons on the gutter between line numbers and line start. Move the mouse over the gutter and click to fold and unfold regions. Use `Shift + Click` on the folding icon to fold or unfold the region and all regions inside.



```
    private onCursorSelectionChanged(): void {
        if (this.state === State.Hidden) {
            return;
        }

        this.editor.layoutContentWidget(this);
    }

    private onEditorBlur(): void { ···
    }

    private onListSelection(e: ISelectionChangeEvent<CompletionItem>): void {
        if (!e.elements.length) {
            return;
        }
```

You can also use the following actions:

- Fold (`Ctrl+Shift+[`) folds the innermost uncollapsed region at the cursor.
- Unfold (`Ctrl+Shift+]`) unfolds the collapsed region at the cursor.
- Toggle Fold (`Ctrl+K Ctrl+L`) folds or unfolds the region at the cursor.
- Fold Recursively (`Ctrl+K Ctrl+[`) folds the innermost uncollapsed region at the cursor and all regions inside that region.

- Unfold Recursively (`Ctrl+K Ctrl+]`) unfolds the region at the cursor and all regions inside that region.
- Fold All (`Ctrl+K Ctrl+0`) folds all regions in the editor.
- Unfold All (`Ctrl+K Ctrl+J`) unfolds all regions in the editor.
- Fold Level X (`Ctrl+K Ctrl+2` for level 2) folds all regions of level X, except the region at the current cursor position.
- Fold All Block Comments (`Ctrl+K Ctrl+/`) folds all regions that start with a block comment token.

Folding regions are by default evaluated based on the indentation of lines. A folding region starts when a line has a smaller indent than one or more following lines, and ends when there is a line with the same or smaller indent.

Folding regions can also be computed based on syntax tokens of the editor's configured language. The following languages already provide syntax aware folding: Markdown, HTML, CSS, LESS, SCSS, and JSON.

If you prefer to switch back to indentation-based folding for one (or all) of the languages above, use:

```
"[html]": {

  "editor.foldingStrategy": "indentation"

},
```

Regions can also be defined by markers defined by each language. The following languages currently have markers defined:

Language	Start region	End region
Bat	`::#region` or `REM #region`	`::#endregion` or `REM #endregion`
C#	`#region`	`#endregion`
C/C++	`#pragma region`	`#pragma endregion`
CSS/Less/SCSS	`/*#region*/`	`/*#endregion*/`
Coffeescript	`#region`	`#endregion`
F#	`//#region` or `(#_region)`	`//#endregion` or `(#_endregion)`

Language	Start region	End region
Java	`//#region` or `//<editor-fold>`	`// #endregion` or `//</editor-fold>`
Markdown	`<!-- #region -->`	`<!-- #endregion -->`
Perl5	`#region` or `=pod`	`#endregion` or `=cut`
PHP	`#region`	`#endregion`
PowerShell	`#region`	`#endregion`
Python	`#region` or `# region`	`#endregion` or `# endregion`
TypeScript/JavaScript	`//#region`	`//#endregion`
Visual Basic	`#Region`	`#End Region`

To fold and unfold only the regions defined by markers use:

- Fold Marker Regions (`Ctrl+K Ctrl+8`) folds all marker regions.
- Unfold Marker Regions (`Ctrl+K Ctrl+9`) unfolds all marker regions.

Fold selection

The command **Create Manual Folding Ranges from Selection** (`Ctrl+K Ctrl+,`) creates a folding range from the currently selected lines and collapses it. That range is called a **manual** folding range that goes on top of the ranges computed by folding providers.

Manual folding ranges can be removed with the command **Remove Manual Folding Ranges** (`Ctrl+K Ctrl+.`).

Manual folding ranges are especially useful for cases when there isn't programming language support for folding.

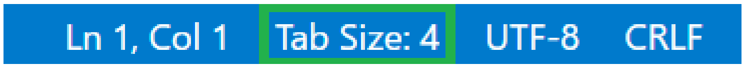
Indentation

VS Code lets you control text indentation and whether you'd like to use spaces or tab stops. By default, VS Code inserts spaces and uses 4 spaces per `Tab` key. If you'd like to use another default, you can modify the `editor.insertSpaces` and `editor.tabSize` settings.

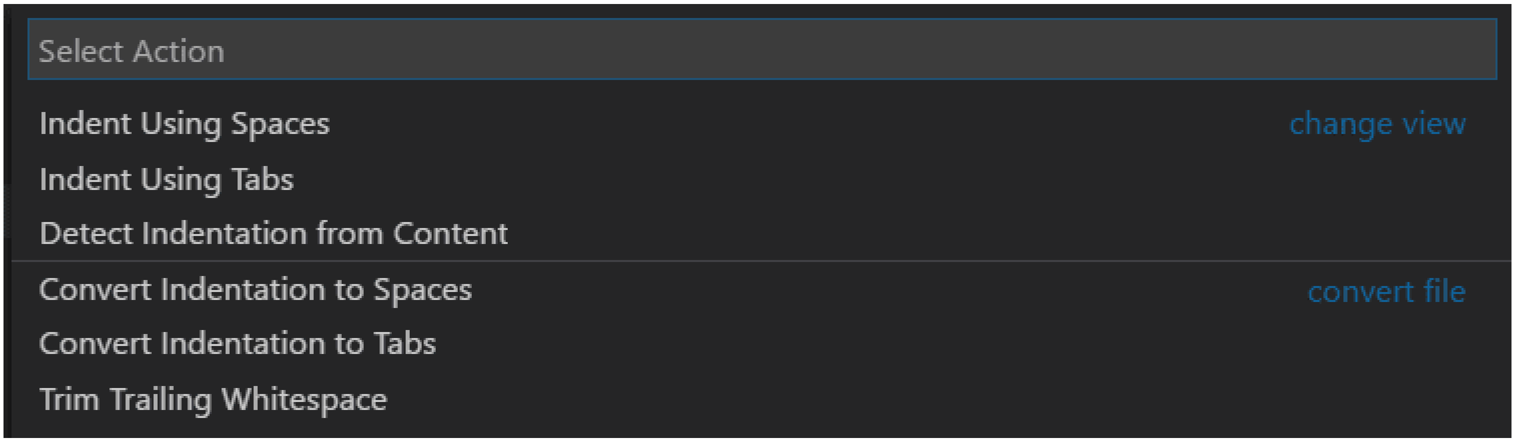
```
    "editor.insertSpaces": true,

    "editor.tabSize": 4,
```

Auto-detection

VS Code analyzes your open file and determines the indentation used in the document. The auto-detected indentation overrides your default indentation settings. The detected setting is displayed on the right side of the Status Bar:



You can click on the Status Bar indentation display to bring up a dropdown with indentation commands allowing you to change the default settings for the open file or convert between tab stops and spaces.

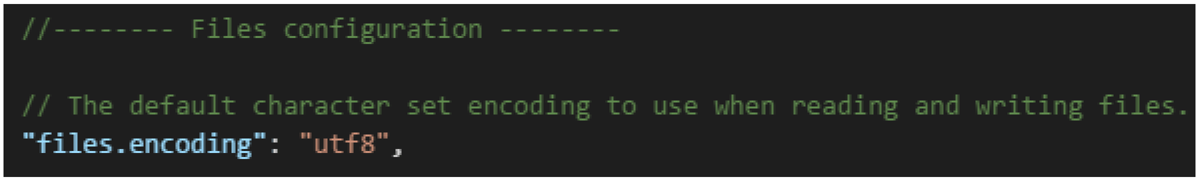


Note: VS Code auto-detection checks for indentations of 2, 4, 6 or 8 spaces. If your file uses a different number of spaces, the indentation may not be correctly detected. For example, if your convention is to indent with 3 spaces, you may want to turn off `editor.detectIndentation` and explicitly set the tab size to 3.

```
    "editor.detectIndentation": false,

    "editor.tabSize": 3,
```

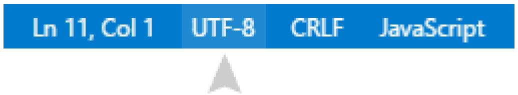
File encoding support

Set the file encoding globally or per workspace by using the `files.encoding` setting in **User Settings** or **Workspace Settings**.

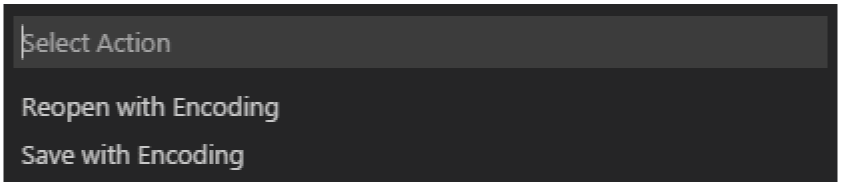


```
//-------- Files configuration --------

// The default character set encoding to use when reading and writing files.
"files.encoding": "utf8",
```

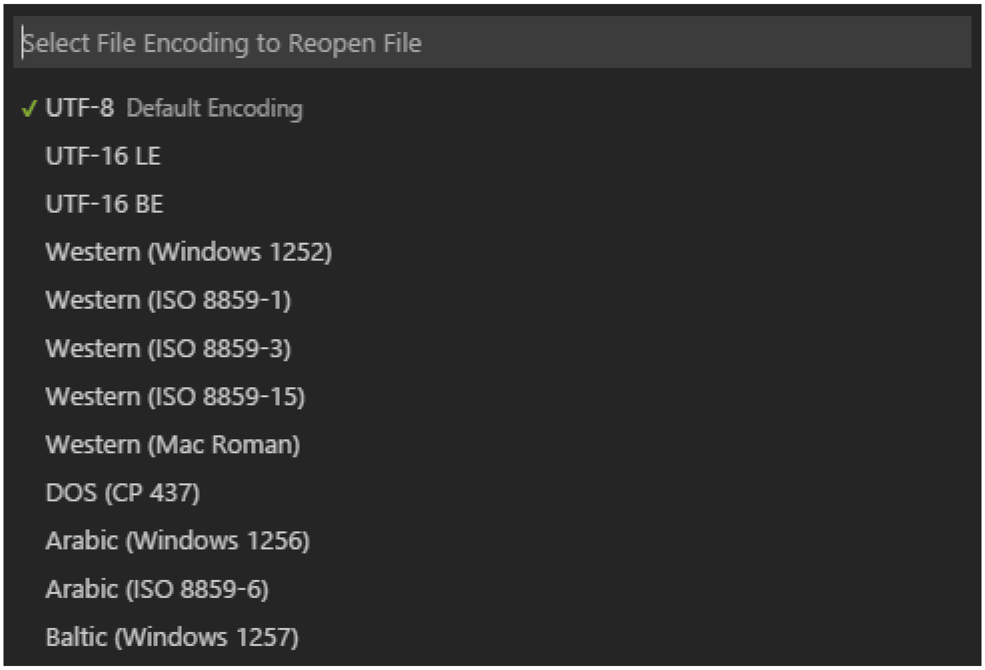
You can view the file encoding in the status bar.



Click on the encoding button in the status bar to reopen or save the active file with a different encoding.



Then choose an encoding.



Next steps

You've covered the basic user interface - there is a lot more to VS Code. Read on to find out about:

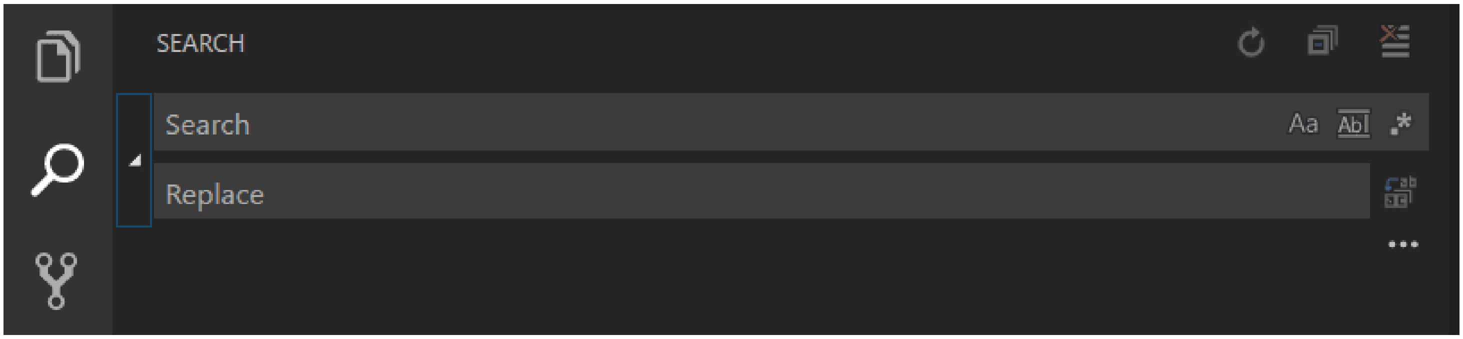
- Intro Video - Setup and Basics - Watch a tutorial on the basics of VS Code.
- User/Workspace Settings - Learn how to configure VS Code to your preferences through user and workspace settings.
- Code Navigation - Peek and Goto Definition, and more.
- Integrated Terminal - Learn about the integrated terminal for quickly performing command-line tasks from within VS Code.
- IntelliSense - VS Code brings smart code completions.

- Debugging - This is where VS Code really shines.

Common questions

Is it possible to globally search and replace?

Yes, expand the Search view text box to include a replace text field. You can search and replace across all the files in your workspace. Note that if you did not open VS Code on a folder, the search will only run on the currently open files.



How do I turn on word wrap?

You can control word wrap through the `editor.wordWrap` setting. By default, `editor.wordWrap` is `off` but if you set to it to `on`, text will wrap on the editor's viewport width.

```
    "editor.wordWrap": "on"
```

You can toggle word wrap for the VS Code session with `Alt+Z`.

You can also add vertical column rulers to the editor with the `editor.rulers` setting, which takes an array of column character positions where you'd like vertical rulers.

How can I avoid placing extra cursors in word wrapped lines?

If you'd like to ignore line wraps when adding cursors above or below your current selection, you can pass in `{ "logicalLine": true }` to `args` on the keybinding like this:

```
{

  "key": "shift+alt+down",

  "command": "editor.action.insertCursorBelow",

  "when": "textInputFocus",

  "args": { "logicalLine": true },

},

{
  "key": "shift+alt+up",
  "command": "editor.action.insertCursorAbove",
  "when": "textInputFocus",
  "args": { "logicalLine": true },
},
```

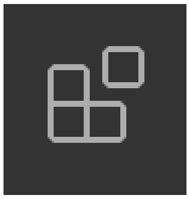
Extension Marketplace

Increase the power of Visual Studio Code through Extensions

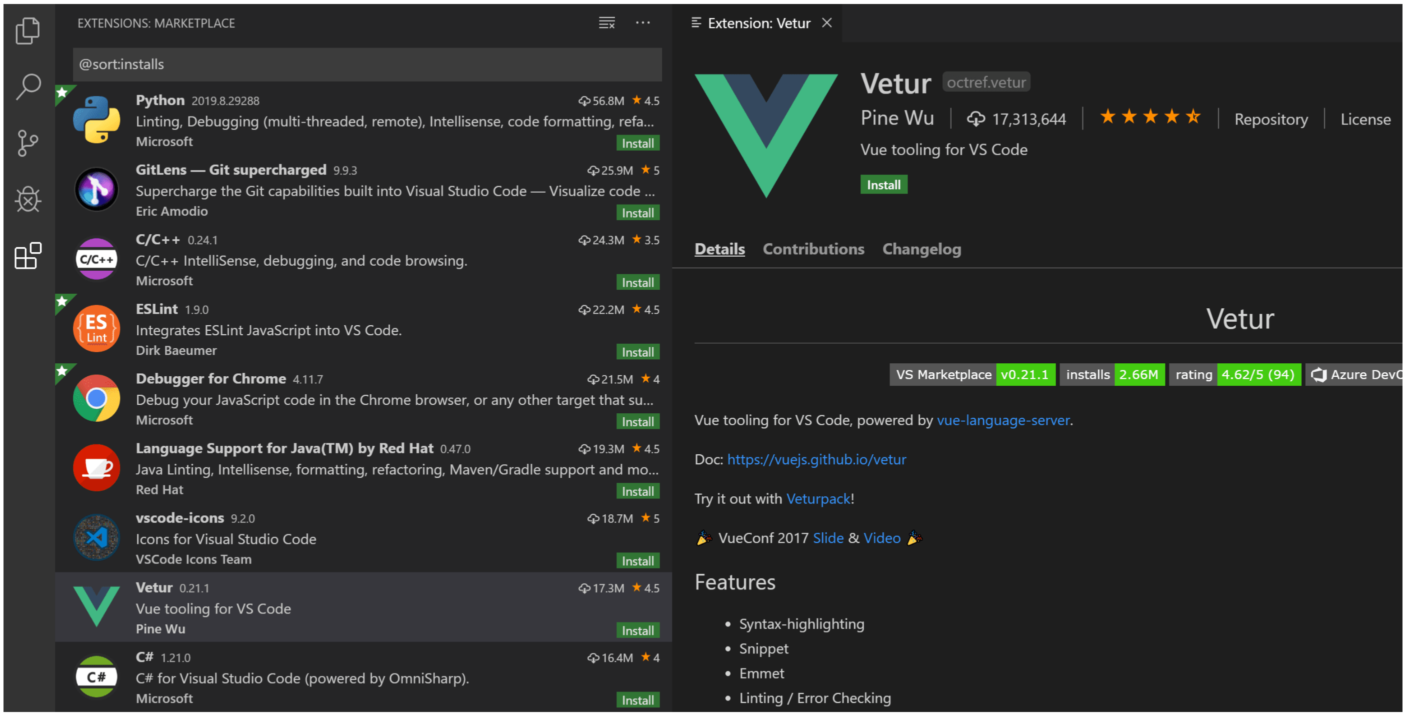
The features that Visual Studio Code includes out-of-the-box are just the start. VS Code extensions let you add languages, debuggers, and tools to your installation to support your development workflow. VS Code's rich extensibility model lets extension authors plug directly into the VS Code UI and contribute functionality through the same APIs used by VS Code. This article explains how to find, install, and manage VS Code extensions from the [Visual Studio Code Marketplace](https://marketplace.visualstudio.com/VSCode).

Browse for extensions

You can browse and install extensions from within VS Code. Bring up the Extensions view by clicking on the Extensions icon in the **Activity Bar** on the side of VS Code or the **View: Extensions** command (`Ctrl+Shift+X`).



This will show you a list of the most popular VS Code extensions on the [VS Code Marketplace](https://marketplace.visualstudio.com/VSCode).



Python 2019.8.29288 56.8M ★4.5
Linting, Debugging (multi-threaded, remote), Intellisense, code formatting, refa...
Microsoft Install

GitLens — Git supercharged 9.9.3 25.9M ★5
Supercharge the Git capabilities built into Visual Studio Code — Visualize code ...
Eric Amodio Install

C/C++ 0.24.1 24.3M ★3.5
C/C++ IntelliSense, debugging, and code browsing.
Microsoft Install

ESLint 1.9.0 22.2M ★4.5
Integrates ESLint JavaScript into VS Code.
Dirk Baeumer Install

Debugger for Chrome 4.11.7 21.5M ★4
Debug your JavaScript code in the Chrome browser, or any other target that su...
Microsoft Install

Language Support for Java(TM) by Red Hat 0.47.0 19.3M ★4.5
Java Linting, Intellisense, formatting, refactoring, Maven/Gradle support and mo...
Red Hat Install

vscode-icons 9.2.0 18.7M ★5
Icons for Visual Studio Code
VSCode Icons Team Install

Vetur 0.21.1 17.3M ★4.5
Vue tooling for VS Code
Pine Wu Install

C# 1.21.0 16.4M ★4
C# for Visual Studio Code (powered by OmniSharp).
Microsoft Install

Vetur octref.vetur
Pine Wu | 17,313,644 | ★★★★★ | Repository | License
Vue tooling for VS Code
Install

Vetur

Vue tooling for VS Code, powered by vue-language-server.

Doc: https://vuejs.github.io/vetur

Try it out with Veturpack!

🎉 VueConf 2017 Slide & Video 🎉

Features

- Syntax-highlighting
- Snippet
- Emmet
- Linting / Error Checking

Each extension in the list includes a brief description, the publisher, the download count, and a five star rating. You can select the extension item to display the extension's details page where you can learn more.

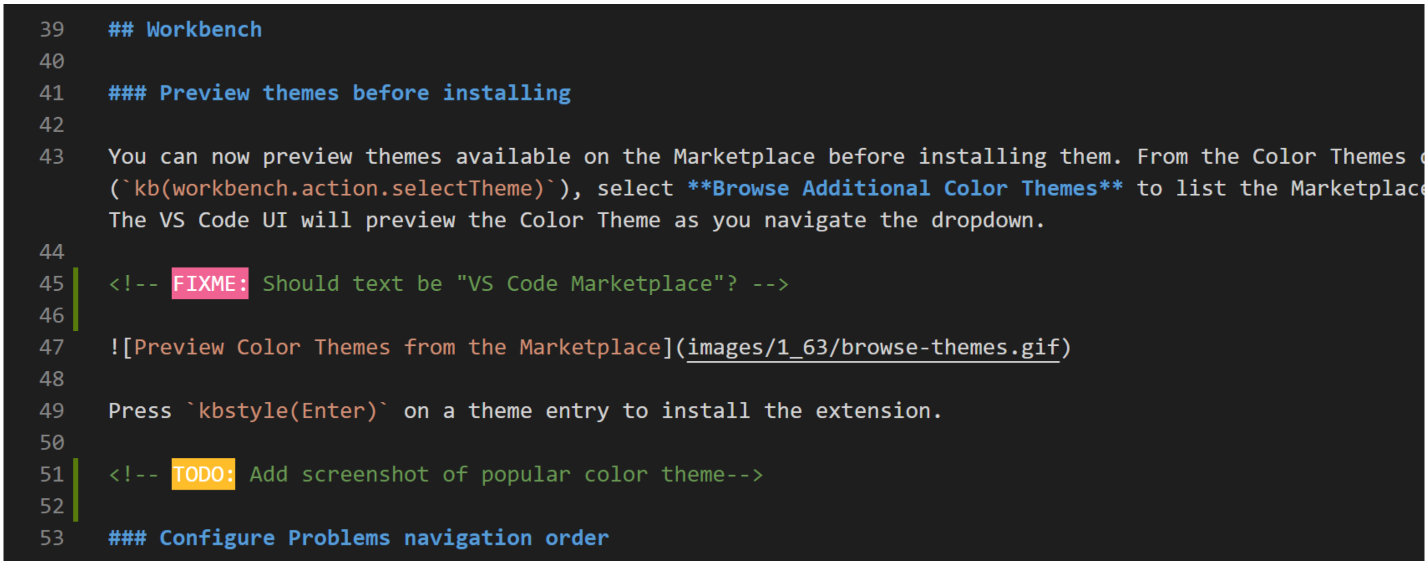
Note: If your computer's Internet access goes through a proxy server, you will need to configure the proxy server. See [Proxy server support]() for details.

Install an extension

To install an extension, select the **Install** button. Once the installation is complete, the **Install** button will change to the **Manage** gear button.

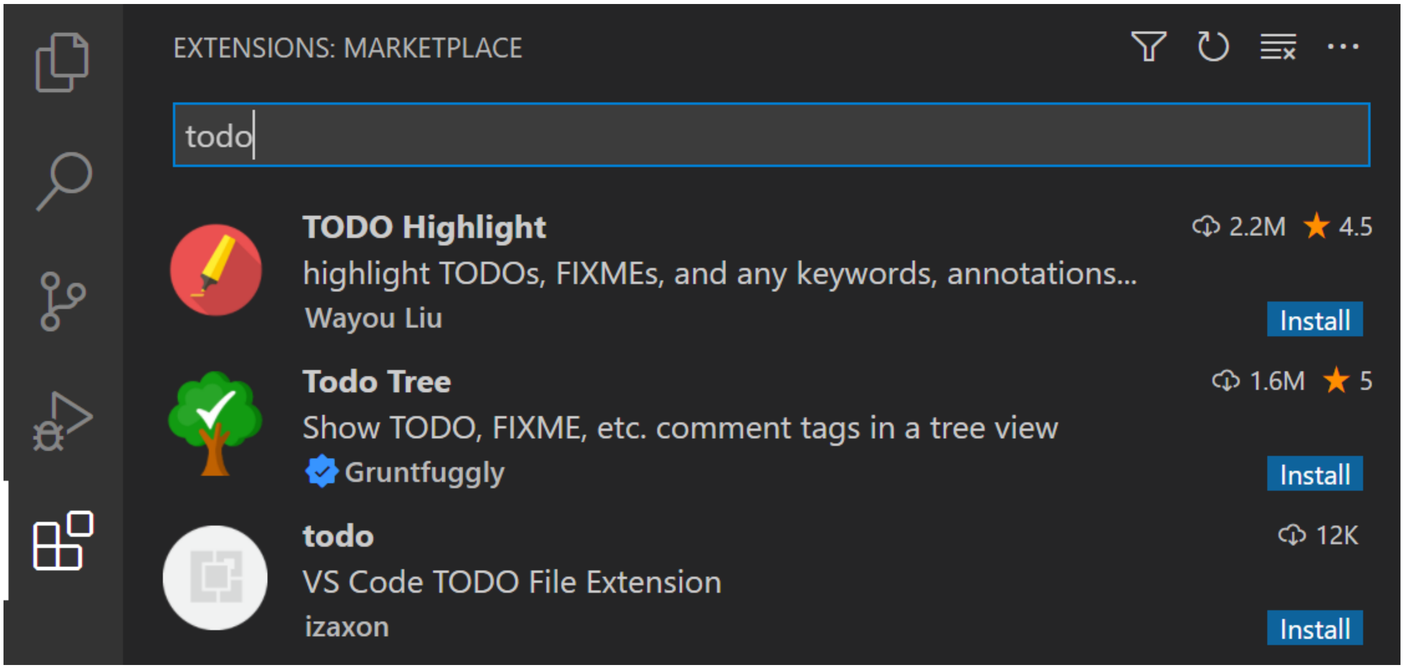
Find and install an extension

For example, let's install the popular [TODO Highlight]() extension. This extension highlights text like 'TODO:' and 'FIXME:' in your source code so you can quickly find undone sections.



```
## Workbench

### Preview themes before installing

You can now preview themes available on the Marketplace before installing them. From the Color Themes
(`kb(workbench.action.selectTheme)`), select **Browse Additional Color Themes** to list the Marketplace
The VS Code UI will preview the Color Theme as you navigate the dropdown.

<!-- FIXME: Should text be "VS Code Marketplace"? -->

![Preview Color Themes from the Marketplace](images/1_63/browse-themes.gif)

Press `kbstyle(Enter)` on a theme entry to install the extension.

<!-- TODO: Add screenshot of popular color theme-->

### Configure Problems navigation order
```

In the Extensions view (`Ctrl+Shift+X`), type 'todo' in the search box to filter the Marketplace offerings to extensions with 'todo' in the title or metadata. You should see the **TODO Highlight** extension in the list.



EXTENSIONS: MARKETPLACE

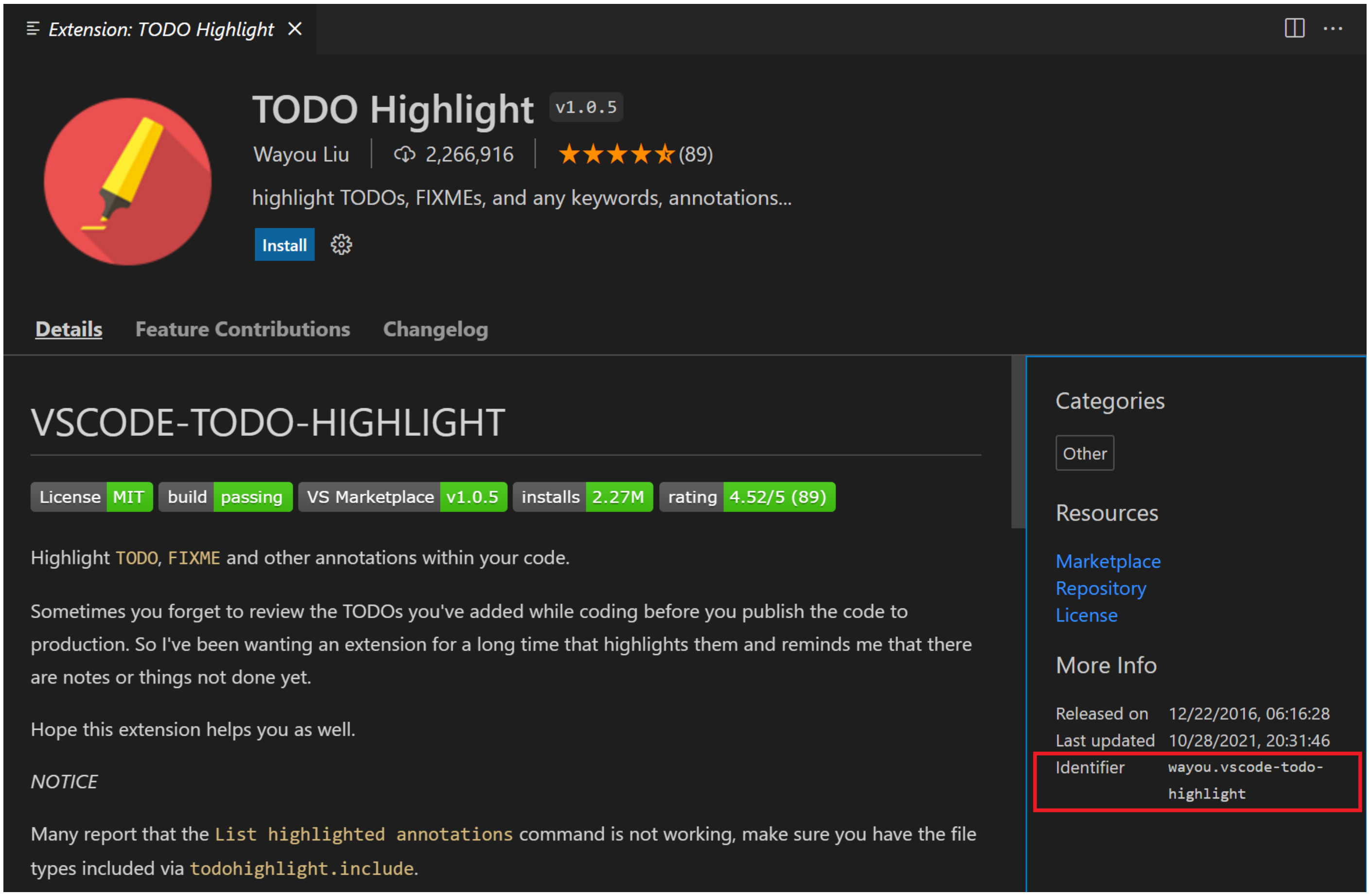
todo

TODO Highlight 2.2M ★ 4.5
highlight TODOs, FIXMEs, and any keywords, annotations...
Wayou Liu

Todo Tree 1.6M ★ 5
Show TODO, FIXME, etc. comment tags in a tree view
Gruntfuggly

todo 12K
VS Code TODO File Extension
izaxon

An extension is uniquely identified by its publisher and extension IDs. If you select the **TODO Highlight** extension, you will see the Extension details page, where you can find the extension ID, in this case, `wayou.vscode-todo-highlight`. Knowing the extension ID can be helpful if there are several similarly named extensions.



highlight TODOs, FIXMEs, and any keywords, annotations...

VSCODE-TODO-HIGHLIGHT

Highlight `TODO`, `FIXME` and other annotations within your code.

Sometimes you forget to review the TODOs you've added while coding before you publish the code to production. So I've been wanting an extension for a long time that highlights them and reminds me that there are notes or things not done yet.

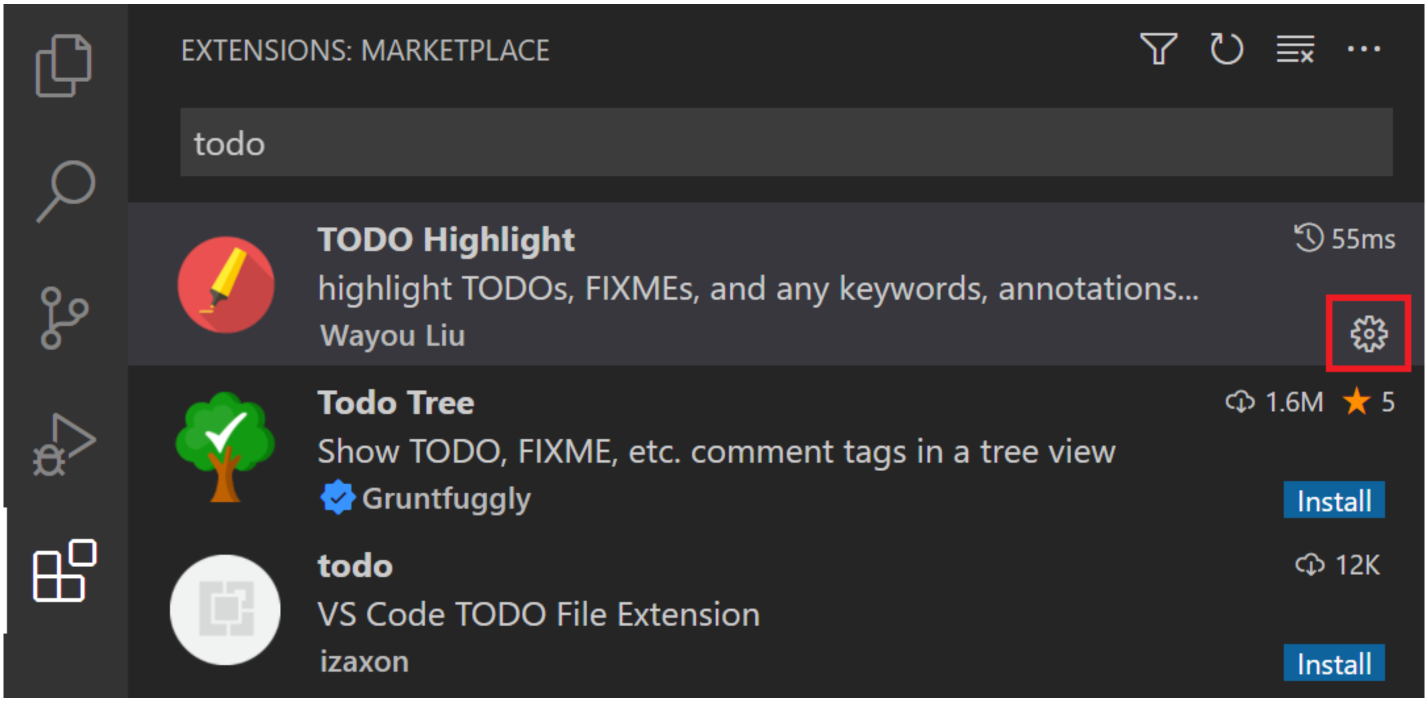
Hope this extension helps you as well.

NOTICE

Many report that the `List highlighted annotations` command is not working, make sure you have the file types included via `todohighlight.include`.

Released on 12/22/2016, 06:16:28
Last updated 10/28/2021, 20:31:46
Identifier `wayou.vscode-todo-highlight`

Select the **Install** button, and VS Code will download and install the extension from the Marketplace. When the installation is complete, the **Install** button will be replaced with a **Manage** gear button.



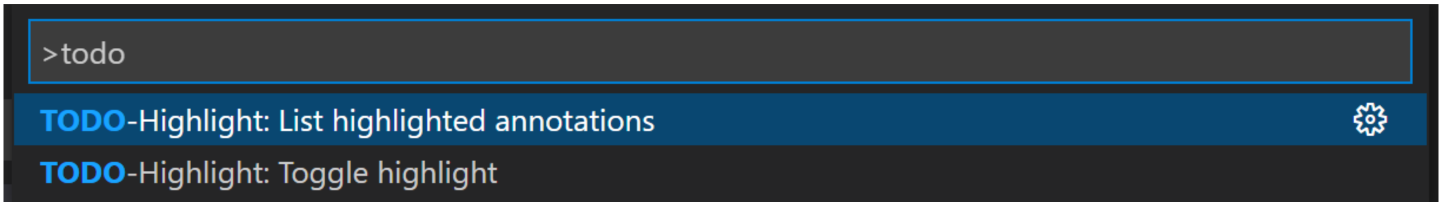
TODO Highlight
highlight TODOs, FIXMEs, and any keywords, annotations...
Wayou Liu

Todo Tree
Show TODO, FIXME, etc. comment tags in a tree view
Gruntfuggly

todo
VS Code TODO File Extension
izaxon

To see the TODO Highlight extension in action, open any source code file and add the text 'TODO:' and you will see the text highlighted.

The TODO Highlight extension contributes the commands, **TODO-Highlight: List highlighted annotations** and **TODO-Highlight: Toggle highlight**, that you can find in the Command Palette (`Ctrl+Shift+P`). The **TODO-Highlight: Toggle highlight** command lets you quickly disable or enable highlighting.

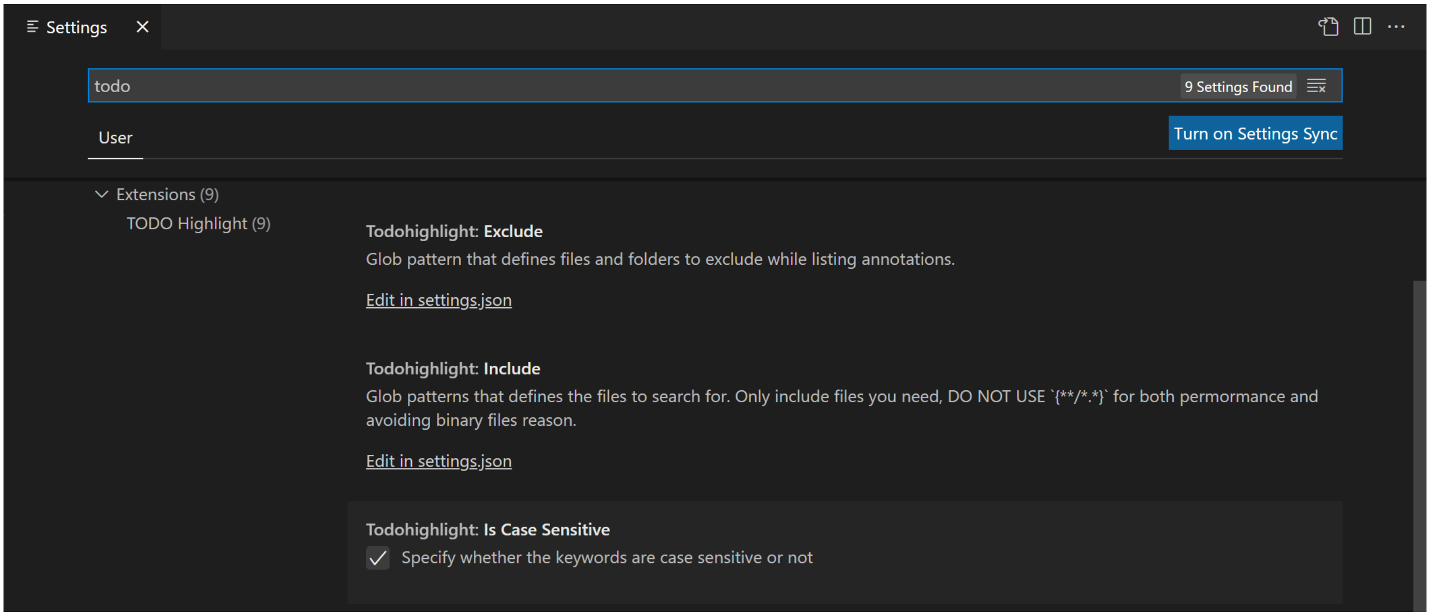


>todo

TODO-Highlight: List highlighted annotations

TODO-Highlight: Toggle highlight

The extension also provides settings for tuning its behavior, which you can find in the Settings editor (`Ctrl+,`). For example, you might want the text search to be case insensitive and you can uncheck the **Todohighlight: Is Case Sensitive** setting.

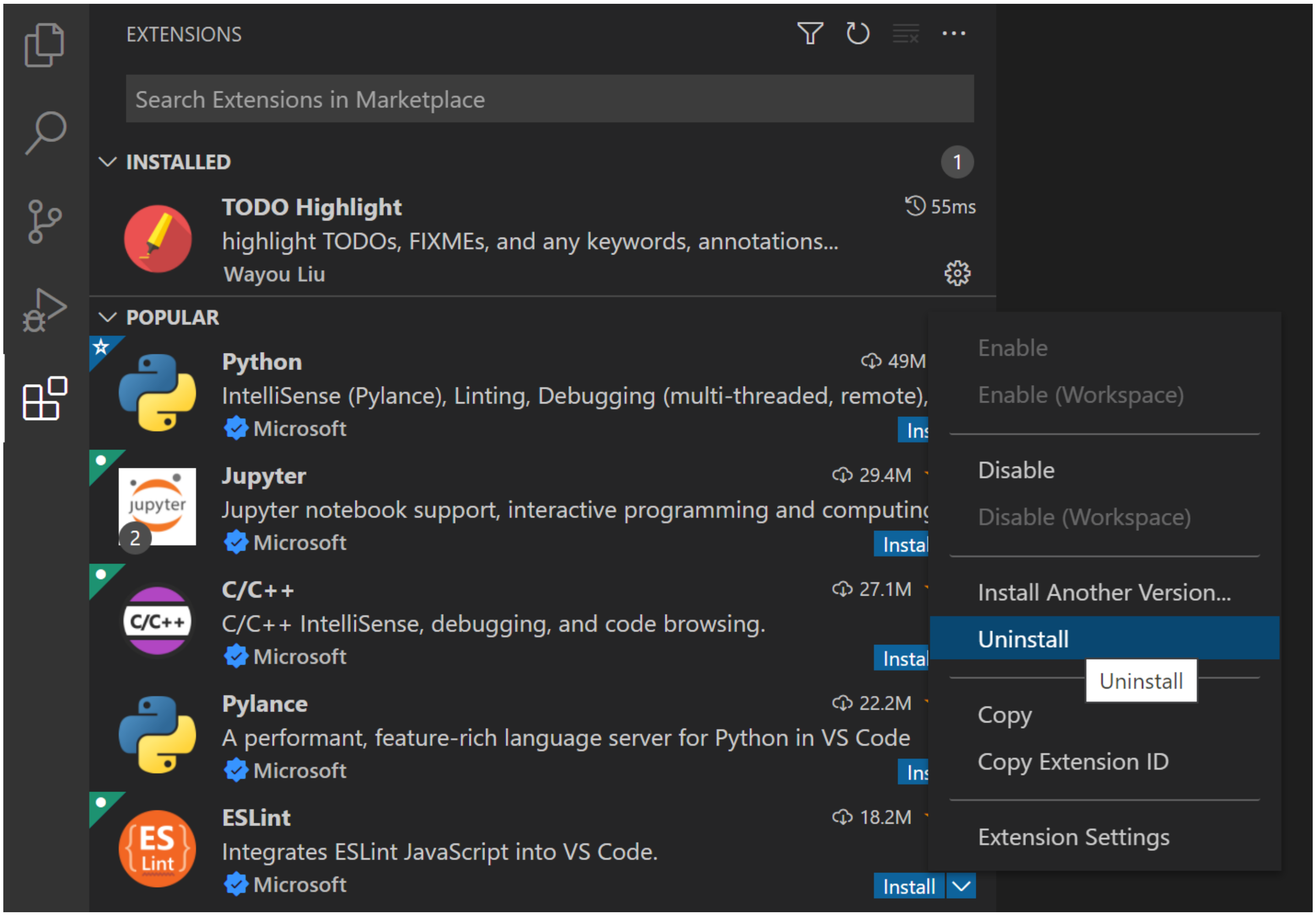


Todohighlight: Exclude
Glob pattern that defines files and folders to exclude while listing annotations.
Edit in settings.json

Todohighlight: Include
Glob patterns that defines the files to search for. Only include files you need, DO NOT USE `{**/*.*}` for both permormance and avoiding binary files reason.
Edit in settings.json

Todohighlight: Is Case Sensitive
☑ Specify whether the keywords are case sensitive or not

If an extension doesn't provide the functionality you want, you can always **Uninstall** the extension from the **Manage** button context menu.

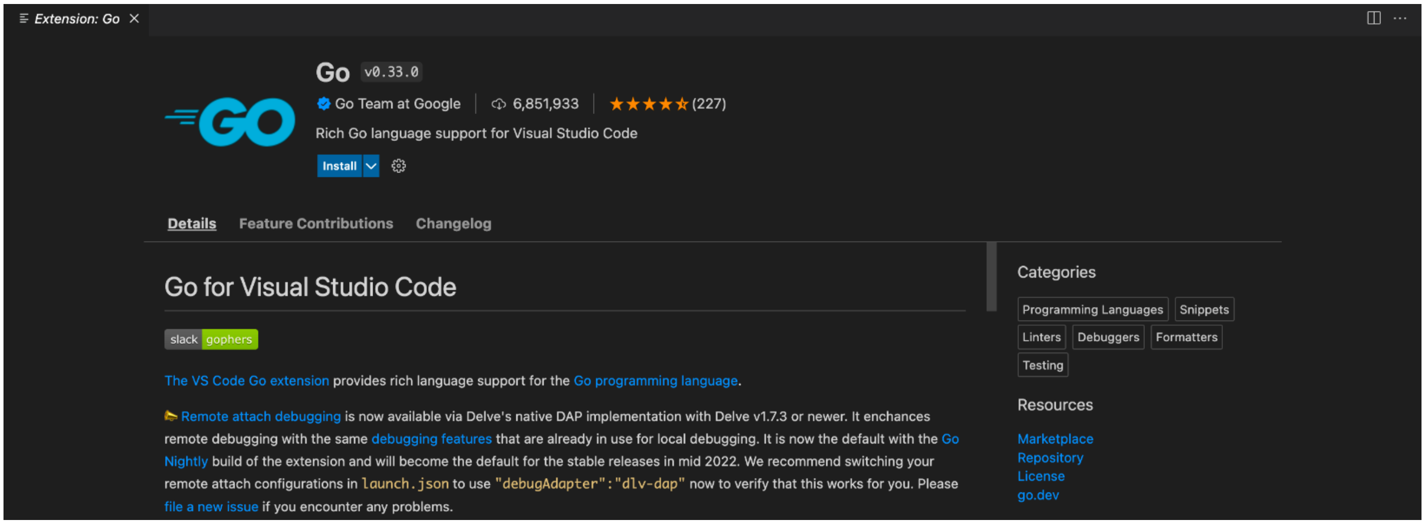

EXTENSIONS
Search Extensions in Marketplace
INSTALLED
1
TODO Highlight
55ms
highlight TODOs, FIXMEs, and any keywords, annotations...
Wayou Liu
POPULAR
Python
49M
IntelliSense (Pylance), Linting, Debugging (multi-threaded, remote),
Microsoft
Jupyter
29.4M
Jupyter notebook support, interactive programming and computing
Microsoft
C/C++
27.1M
C/C++ IntelliSense, debugging, and code browsing.
Microsoft
Pylance
22.2M
A performant, feature-rich language server for Python in VS Code
Microsoft
ESLint
18.2M
Integrates ESLint JavaScript into VS Code.
Microsoft
Install
Enable
Enable (Workspace)
Disable
Disable (Workspace)
Install Another Version...
Uninstall
Uninstall
Copy
Copy Extension ID
Extension Settings

This has been just one example of how to install and use an extension. The VS Code Marketplace has thousands of extensions supporting hundreds of programming languages and tasks. Everything from full featured language support for Java, Python, Go, and C++ to simple extensions that create GUIDs, change the color theme, or add virtual pets to the editor.

Extension details

On the extension details page, you can read the extension's README and review the extension's:

- **Feature Contributions** - The extension's additions to VS Code such as settings, commands and keyboard shortcuts, language grammars, debugger, etc.
- **Changelog** - The extension repository CHANGELOG if available.
- **Dependencies** - Lists if the extension depends on any other extensions.

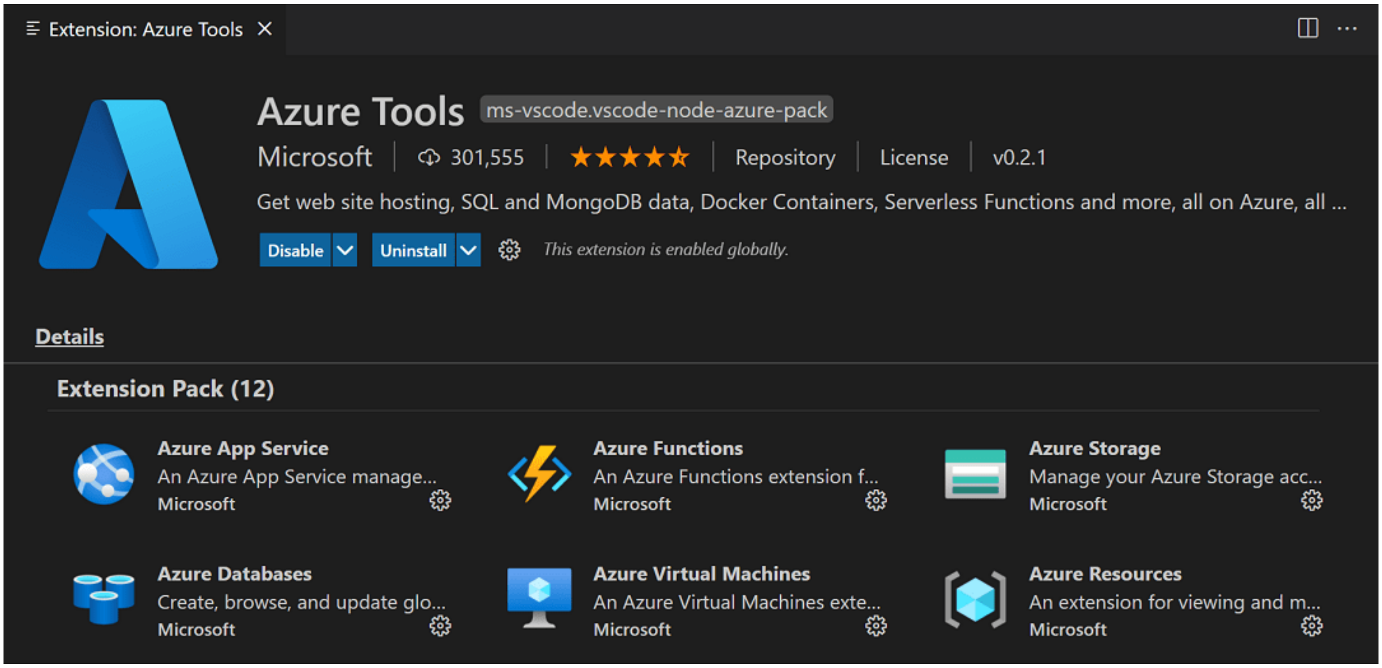


Go for Visual Studio Code

The VS Code Go extension provides rich language support for the Go programming language.

Remote attach debugging is now available via Delve's native DAP implementation with Delve v1.7.3 or newer. It enchances remote debugging with the same debugging features that are already in use for local debugging. It is now the default with the Go Nightly build of the extension and will become the default for the stable releases in mid 2022. We recommend switching your remote attach configurations in `launch.json` to use `"debugAdapter":"dlv-dap"` now to verify that this works for you. Please file a new issue if you encounter any problems.

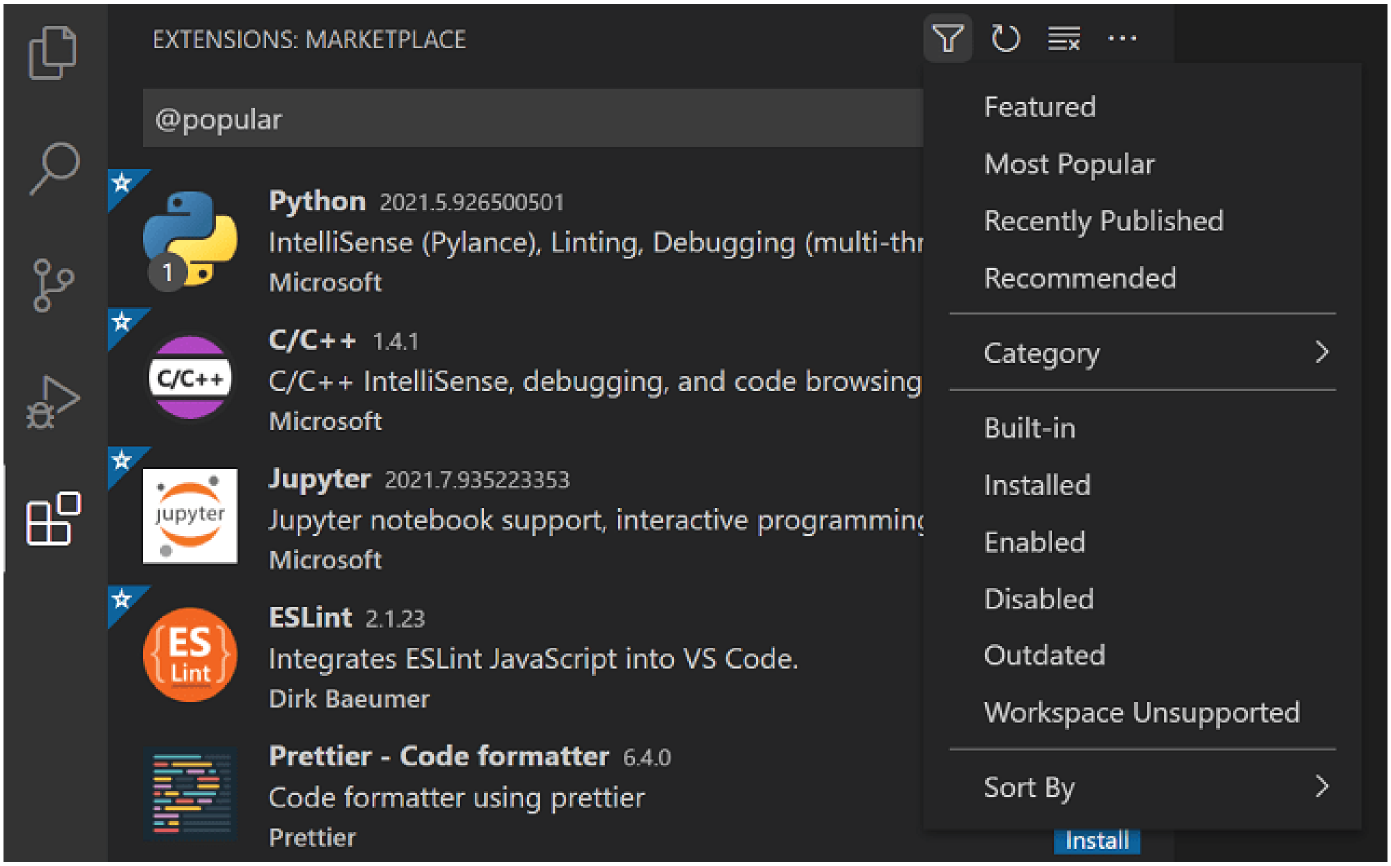
If an extension is an Extension Pack, the **Extension Pack** section will display which extensions will be installed when you install the pack. Extension Packs bundle separate extensions together so they can be easily installed at one time.



Get web site hosting, SQL and MongoDB data, Docker Containers, Serverless Functions and more, all on Azure, all ...

Extensions view filter and commands

You can filter the Extensions view with the **Filter Extensions** context menu.



Python 2021.5.926500501
IntelliSense (Pylance), Linting, Debugging (multi-thr
Microsoft

C/C++ 1.4.1
C/C++ IntelliSense, debugging, and code browsing
Microsoft

Jupyter 2021.7.935223353
Jupyter notebook support, interactive programming
Microsoft

ESLint 2.1.23
Integrates ESLint JavaScript into VS Code.
Dirk Baeumer

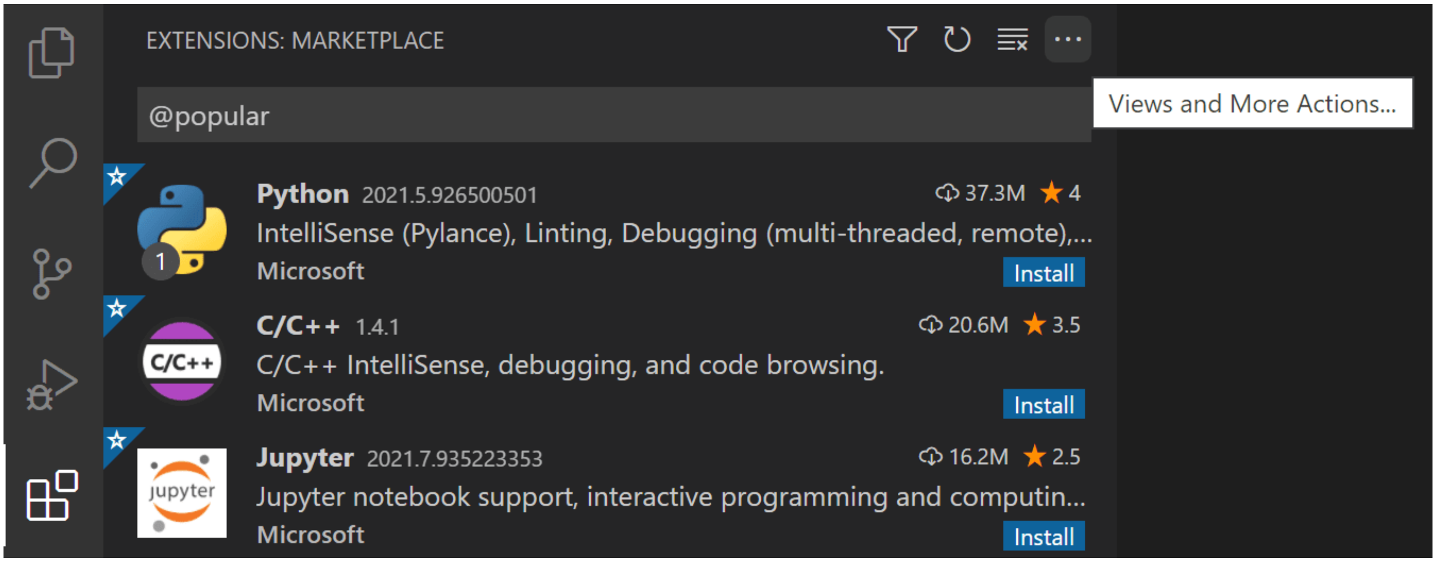
Prettier - Code formatter 6.4.0
Code formatter using prettier
Prettier

There are filters to show:

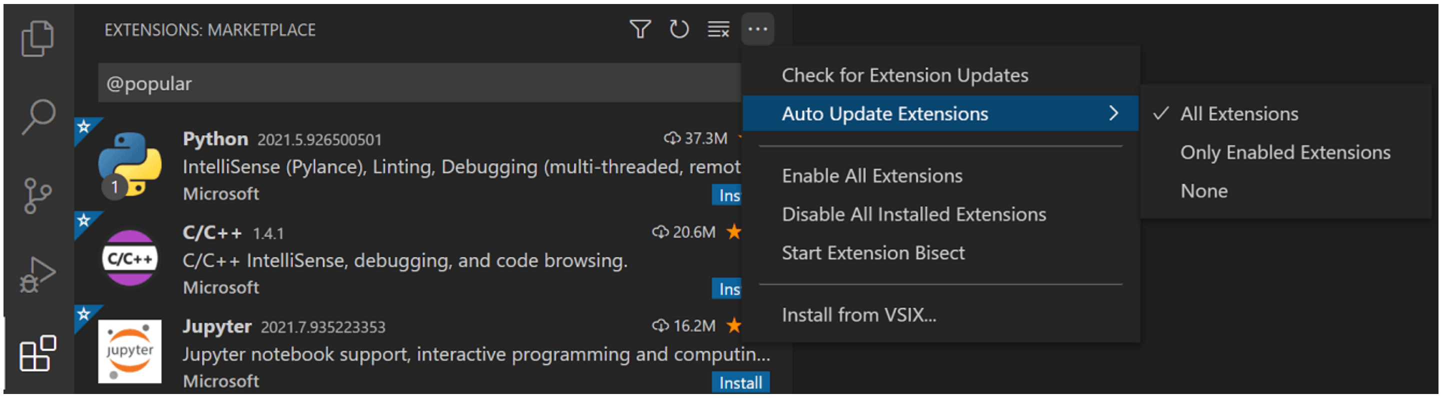
- The list of currently installed extensions
- The list of outdated extensions that can be updated
- The list of currently enabled/disabled extensions
- The list of recommended extensions based on your workspace
- The list of globally popular extensions

You can sort the extension list by **Install Count** or **Rating** in either ascending or descending order. You can learn more about extension search filters below.

You can run additional Extensions view commands via the `...` **View and More Actions** button.


EXTENSIONS: MARKETPLACE
Views and More Actions...
@popular
Python 2021.5.926500501
37.3M ★ 4
IntelliSense (Pylance), Linting, Debugging (multi-threaded, remote),...
Microsoft
Install
C/C++ 1.4.1
20.6M ★ 3.5
C/C++ IntelliSense, debugging, and code browsing.
Microsoft
Install
Jupyter 2021.7.935223353
16.2M ★ 2.5
Jupyter notebook support, interactive programming and computin...
Microsoft
Install

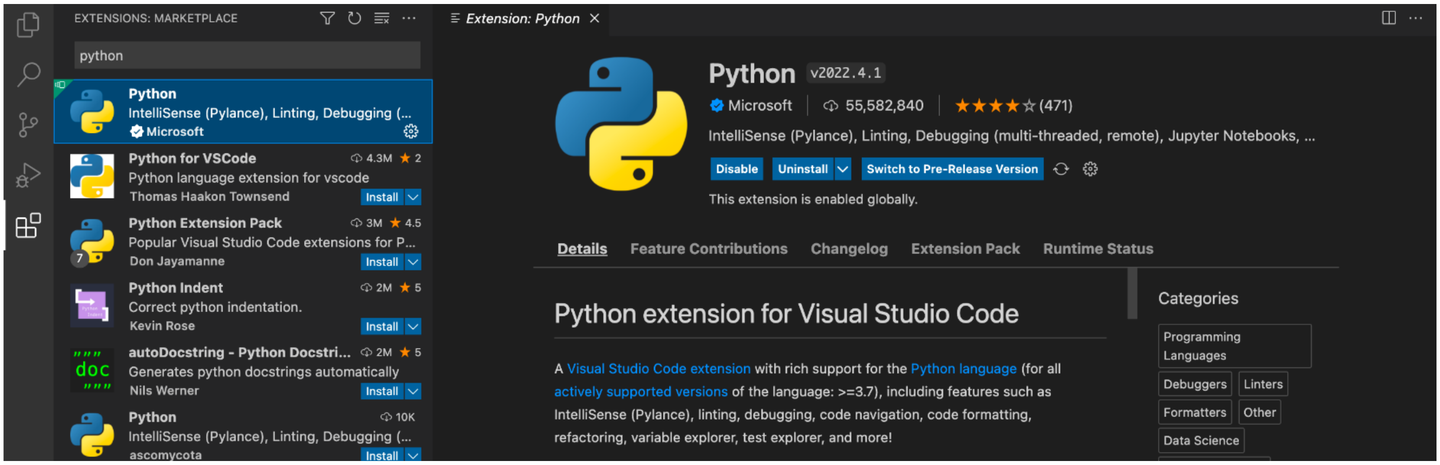
Through this context menu you can control extension updates, enable or disable all extensions, and use the [Extension Bisect](#) utility to isolate problematic extension behavior.



Search for an extension

You can clear the Search box at the top of the Extensions view and type in the name of the extension, tool, or programming language you're looking for.

For example, typing 'python' will bring up a list of Python language extensions:


EXTENSIONS: MARKETPLACE
python
Python
IntelliSense (Pylance), Linting, Debugging (...
Microsoft
Python for VSCode
4.3M ★ 2
Python language extension for vscode
Thomas Haakon Townsend
Install
Python Extension Pack
3M ★ 4.5
Popular Visual Studio Code extensions for P...
Don Jayamanne
Install
Python Indent
2M ★ 5
Correct python indentation.
Kevin Rose
Install
autoDocstring - Python Docstri...
2M ★ 5
Generates python docstrings automatically
Nils Werner
Install
Python
10K
IntelliSense (Pylance), Linting, Debugging (...
ascomycota
Install
Extension: Python
Python v2022.4.1
Microsoft
55,582,840
★★★★☆ (471)
IntelliSense (Pylance), Linting, Debugging (multi-threaded, remote), Jupyter Notebooks, ...
Disable
Uninstall
Switch to Pre-Release Version
This extension is enabled globally.
Details
Feature Contributions
Changelog
Extension Pack
Runtime Status
Python extension for Visual Studio Code
Categories
Programming Languages
Debuggers
Linters
Formatters
Other
Data Science

A Visual Studio Code extension with rich support for the Python language (for all actively supported versions of the language: >=3.7), including features such as IntelliSense (Pylance), linting, debugging, code navigation, code formatting, refactoring, variable explorer, test explorer, and more!

If you know the exact identifier for an extension you're looking for, you can use the `@id:` prefix, for example `@id:octref.vetur`. Additionally, to filter or sort results, you can use the filter and sort commands, detailed below.

Manage extensions

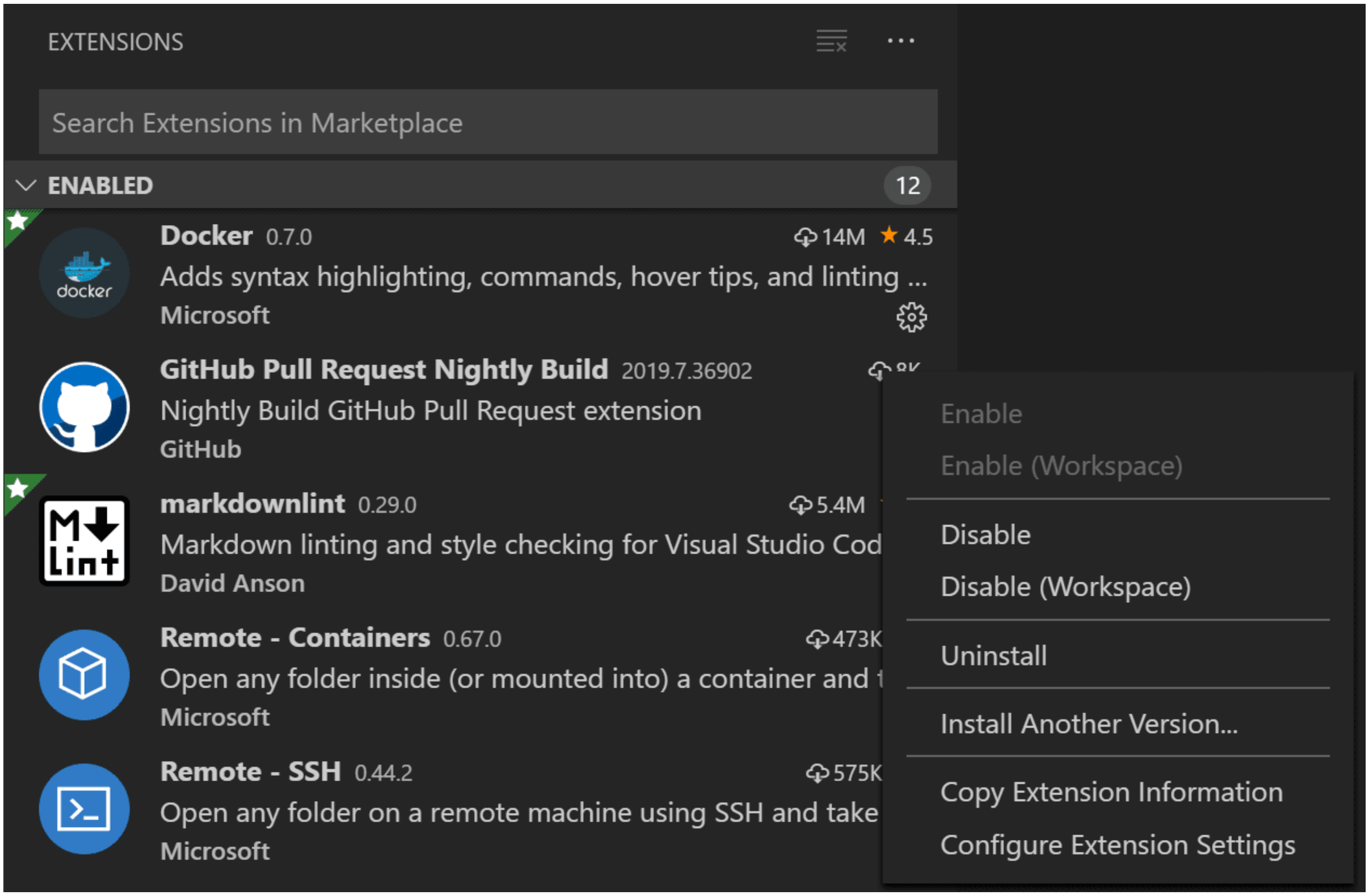
VS Code makes it easy to manage your extensions. You can install, disable, update, and uninstall extensions through the Extensions view, the **Command Palette** (commands have the **Extensions:** prefix) or command-line switches.

List installed extensions

By default, the Extensions view will show the extensions you currently have enabled, all extensions that are recommended for you, and a collapsed view of all extensions you have disabled. You can use the **Show Installed Extensions** command, available in the **Command Palette** (`Ctrl+Shift+P`) or the **More Actions** (`...`) dropdown menu, to clear any text in the search box and show the list of all installed extensions, which includes those that have been disabled.

Uninstall an extension

To uninstall an extension, select the **Manage** gear button at the right of an extension entry and then choose **Uninstall** from the dropdown menu. This will uninstall the extension and prompt you to reload VS Code.


EXTENSIONS
Search Extensions in Marketplace
ENABLED
12
Docker 0.7.0
14M
4.5
Adds syntax highlighting, commands, hover tips, and linting ...
Microsoft
GitHub Pull Request Nightly Build 2019.7.36902
Nightly Build GitHub Pull Request extension
GitHub
markdownlint 0.29.0
5.4M
Markdown linting and style checking for Visual Studio Cod
David Anson
Remote - Containers 0.67.0
473K
Open any folder inside (or mounted into) a container and t
Microsoft
Remote - SSH 0.44.2
575K
Open any folder on a remote machine using SSH and take
Microsoft
Enable
Enable (Workspace)
Disable
Disable (Workspace)
Uninstall
Install Another Version...
Copy Extension Information
Configure Extension Settings

Disable an extension

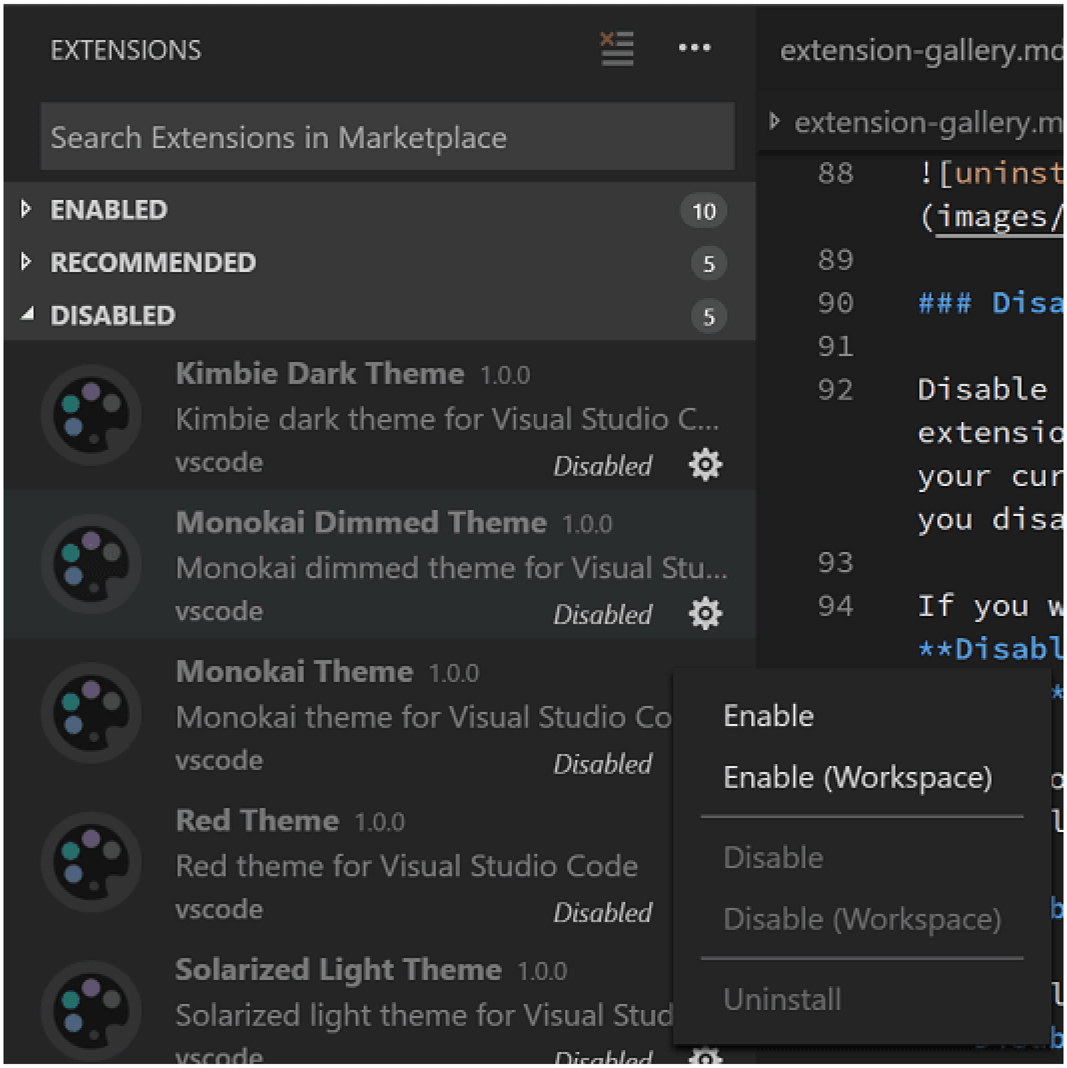
If you don't want to permanently remove an extension, you can instead temporarily disable the extension by clicking the gear button at the right of an extension entry. You can disable an extension globally or just for your current Workspace. You will be prompted to reload VS Code after you disable an extension.

If you want to quickly disable all installed extensions, there is a **Disable All Installed Extensions** command in the **Command Palette** and **More Actions** (`...`) dropdown menu.

Extensions remain disabled for all VS Code sessions until you re-enable them.

Enable an extension

Similarly if you have disabled an extension (it will be in the **Disabled** section of the list and marked ***Disabled***), you can re-enable it with the **Enable** or **Enable (Workspace)** commands in the dropdown menu.



There is also an **Enable All Extensions** command in the **More Actions** (`...`) dropdown menu.

Extension auto-update

VS Code checks for extension updates and installs them automatically. After an update, you will be prompted to reload VS Code. If you'd rather update your extensions manually, you can disable auto-update with the **Disable Auto Updating Extensions** command that sets the `extensions.autoUpdate` setting to `false`. If you don't want VS Code to even check for updates, you can set the `extensions.autoCheckUpdates` setting to false.

Update an extension manually

If you have extensions auto-update disabled, you can quickly look for extension updates by using the **Show Outdated Extensions** command that uses the `@outdated` filter. This will display any available updates for your currently installed extensions. Select the **Update** button for the outdated extension and the update will be installed and you'll be prompted to reload VS Code. You can also update all your outdated extensions at one time with the **Update All Extensions** command. If you also have automatic checking for updates disabled, you can use the **Check for Extension Updates** command to check which of your extensions can be updated.

Recommended extensions

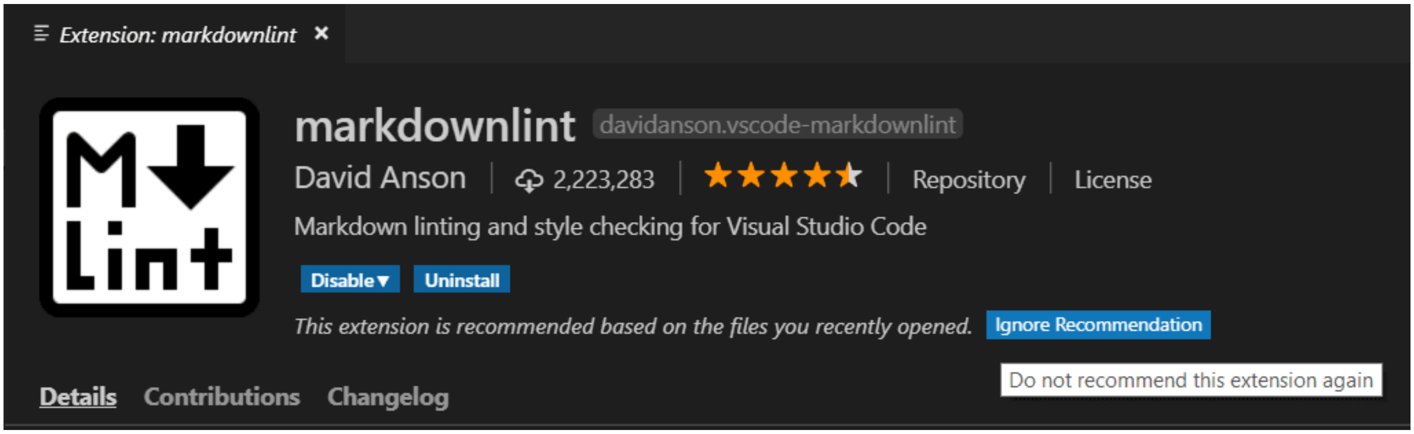
You can see a list of recommended extensions using **Show Recommended Extensions**, which sets the `@recommended` filter. Extension recommendations can either be:

- **Workspace Recommendations** - Recommended by other users of your current workspace.
- **Other Recommendations** - Recommended based on recently opened files.

See the section below to learn how to contribute recommendations for other users in your project.

Ignoring recommendations

To dismiss a recommendation, select on the extension item to open the Details page and then select the **Manage** gear button to display the context menu. Select the **Ignore Recommendation** menu item. Ignored recommendations will no longer be recommended to you.



Markdown linting and style checking for Visual Studio Code

This extension is recommended based on the files you recently opened.

Configuring extensions

VS Code extensions may have very different configurations and requirements. Some extensions contribute settings to VS Code, which can be modified in the Settings editor. Other extensions may have their own configuration files. Extensions may also require installation and setup of additional components like compilers, debuggers, and command-line tools. Consult the extension's README (visible in the Extensions view details page) or go to the extension page on the VS Code Marketplace (click on the extension name in the details page). Many extensions are open source and have a link to their repository on their Marketplace page.

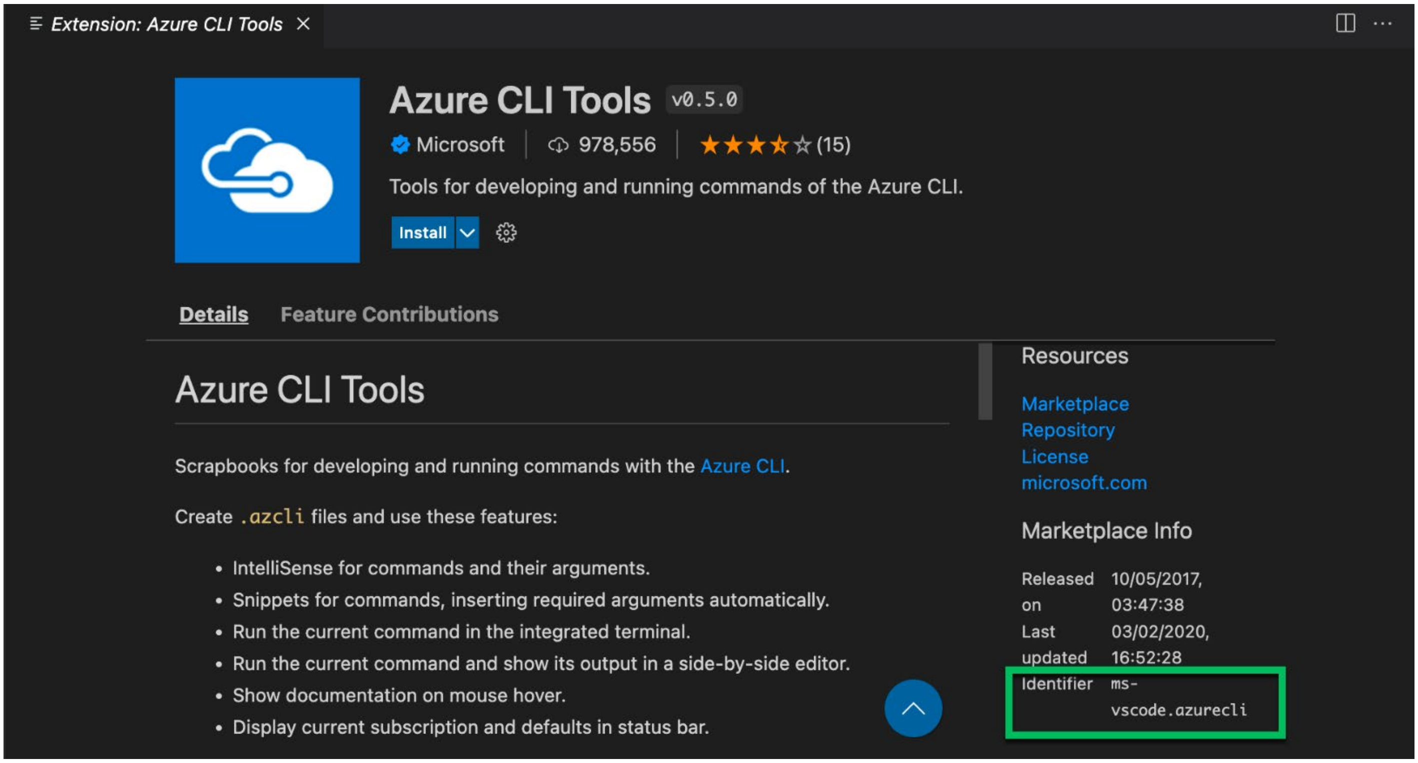
Command line extension management

To make it easier to automate and configure VS Code, it is possible to list, install, and uninstall extensions from the command line. When identifying an extension, provide the full name of the form `publisher.extension`, for example `ms-python.python`.

Example:

```
code --extensions-dir <dir>

    Set the root path for extensions.

code --list-extensions

    List the installed extensions.

code --show-versions

    Show versions of installed extensions, when using --list-extension.

code --install-extension (<extension-id> | <extension-vsix-path>)

    Installs an extension.

code --uninstall-extension (<extension-id> | <extension-vsix-path>)

    Uninstalls an extension.

code --enable-proposed-api (<extension-id>)

    Enables proposed API features for extensions. Can receive one or more extension IDs to enable
individually.
```

You can see the extension ID on the extension details page under the Marketplace Info.



Tools for developing and running commands of the Azure CLI.

Azure CLI Tools

Scrapbooks for developing and running commands with the Azure CLI.

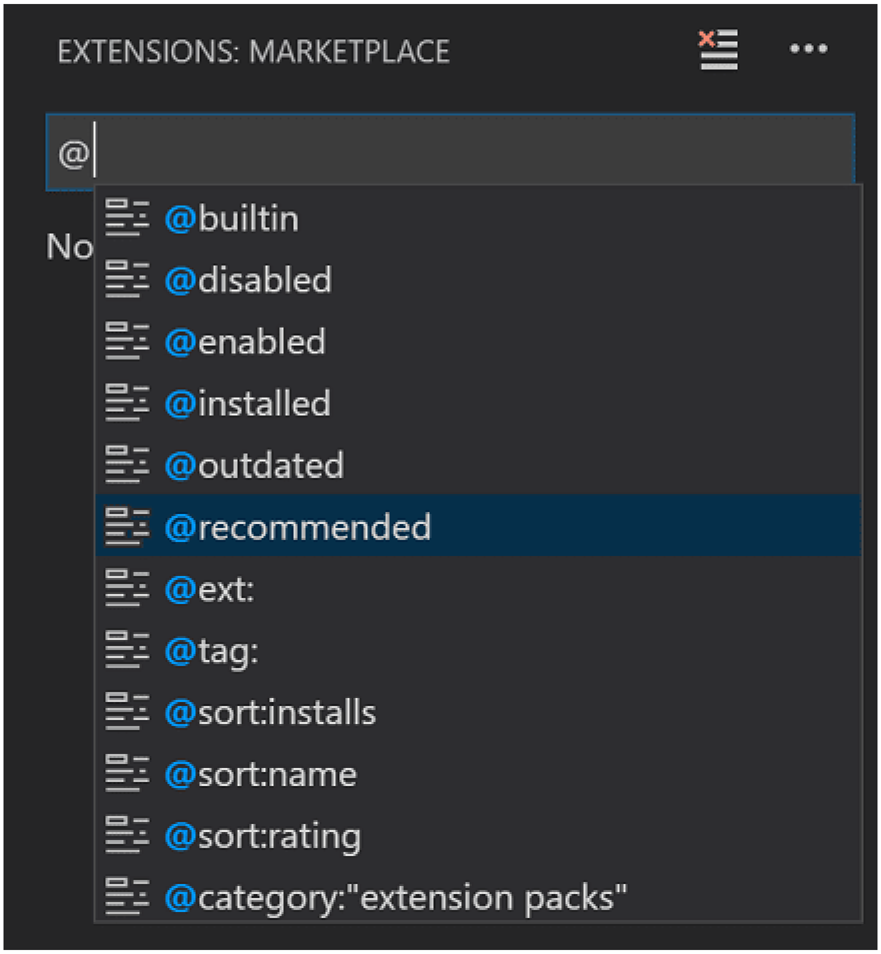
Create `.azcli` files and use these features:

- IntelliSense for commands and their arguments.
- Snippets for commands, inserting required arguments automatically.
- Run the current command in the integrated terminal.
- Run the current command and show its output in a side-by-side editor.
- Show documentation on mouse hover.
- Display current subscription and defaults in status bar.

Released on	10/05/2017, 03:47:38
Last updated	03/02/2020, 16:52:28
Identifier	ms-vscode.azurecli

Extensions view filters

The Extensions view search box supports filters to help you find and manage extensions. You may have seen filters such as `@installed` and `@recommended` if you used the commands **Show Installed Extensions** and **Show Recommended Extensions**. Also, there are filters available to let you sort by popularity or ratings and search by category (for example 'Linters') and tags (for example 'node'). You can see a complete listing of all filters and sort commands by typing `@` in the extensions search box and navigating through the suggestions:



Here are the Extensions view filters:

- `@builtin` - Show extensions that come with VS Code. Grouped by type (Programming Languages, Themes, etc.).
- `@disabled` - Show disabled installed extensions.
- `@installed` - Show installed extensions.
- `@outdated` - Show outdated installed extensions. A newer version is available on the Marketplace.
- `@enabled` - Show enabled installed extensions. Extensions can be individually enabled/disabled.
- `@recommended` - Show recommended extensions. Grouped as Workspace specific or general use.
- `@category` - Show extensions belonging to specified category. Below are a few of supported categories. For a complete list, type `@category` and follow the options in the suggestion list:
    - `@category:themes`
    - `@category:formatters`
    - `@category:linters`
    - `@category:snippets`

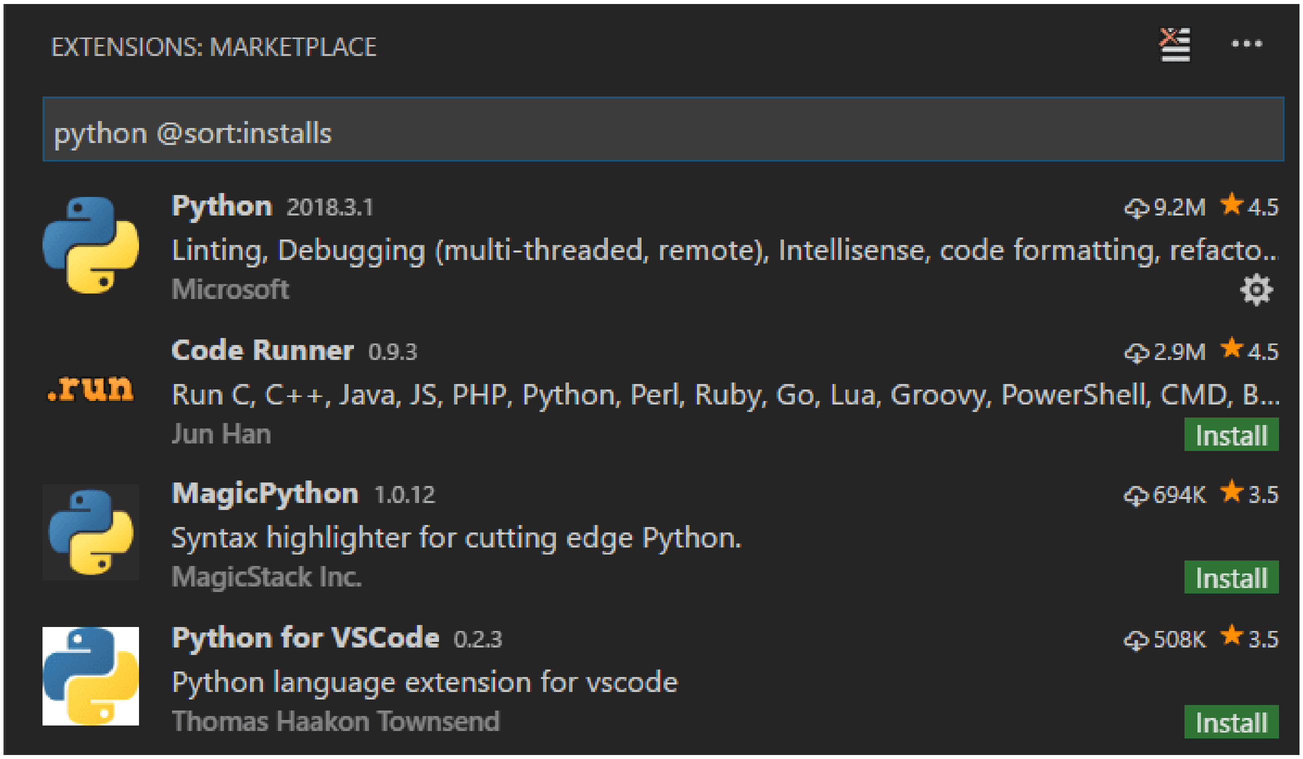
These filters can be combined as well. For example: Use `@installed @category:themes` to view all installed themes.

If no filter is provided, the Extensions view displays the currently installed and recommended extensions.

Sorting

You can sort extensions with the `@sort` filter, which can take the following values:

- `installs` - Sort by Marketplace installation count, in descending order.
- `rating` - Sort by Marketplace rating (1-5 stars), in descending order.
- `name` - Sort alphabetically by extension name.



python @sort:installs

Python 2018.3.1 9.2M ★4.5
Linting, Debugging (multi-threaded, remote), Intellisense, code formatting, refacto..
Microsoft

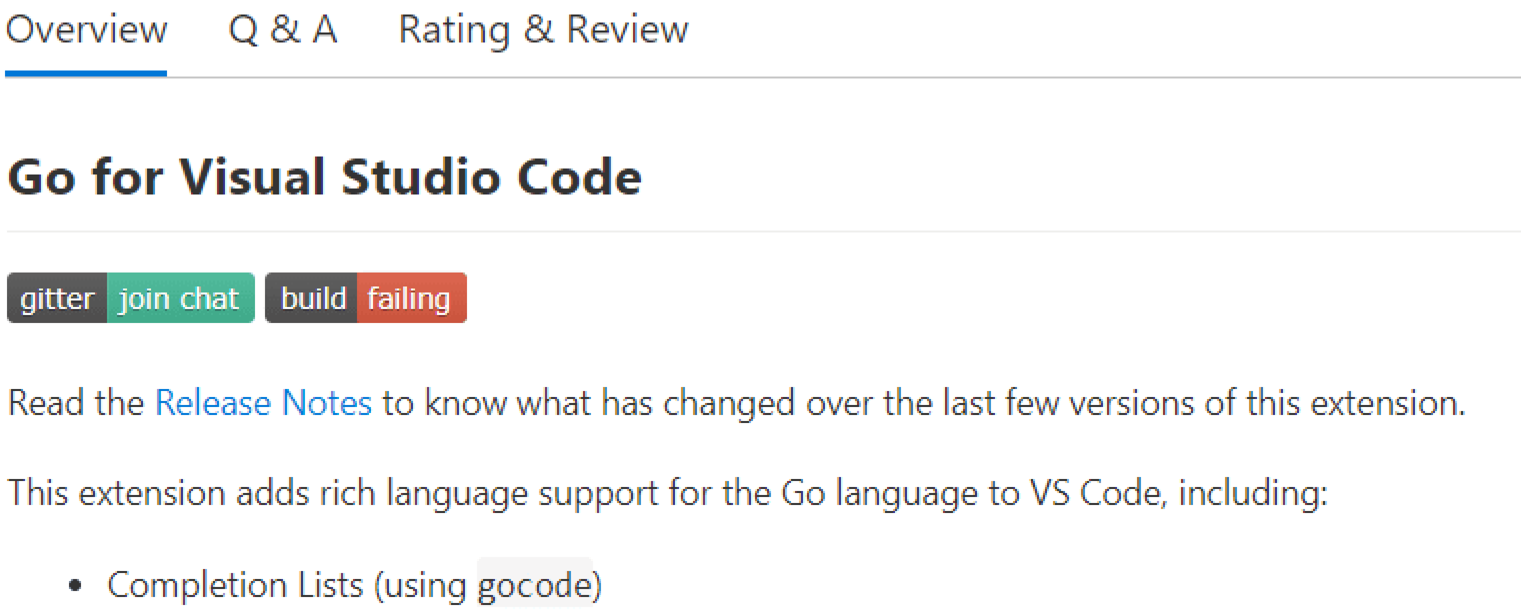
Code Runner 0.9.3 2.9M ★4.5
Run C, C++, Java, JS, PHP, Python, Perl, Ruby, Go, Lua, Groovy, PowerShell, CMD, B...
Jun Han

MagicPython 1.0.12 694K ★3.5
Syntax highlighter for cutting edge Python.
MagicStack Inc.

Python for VSCode 0.2.3 508K ★3.5
Python language extension for vscode
Thomas Haakon Townsend

Categories and tags

Extensions can set **Categories** and **Tags** describing their features.



Go for Visual Studio Code

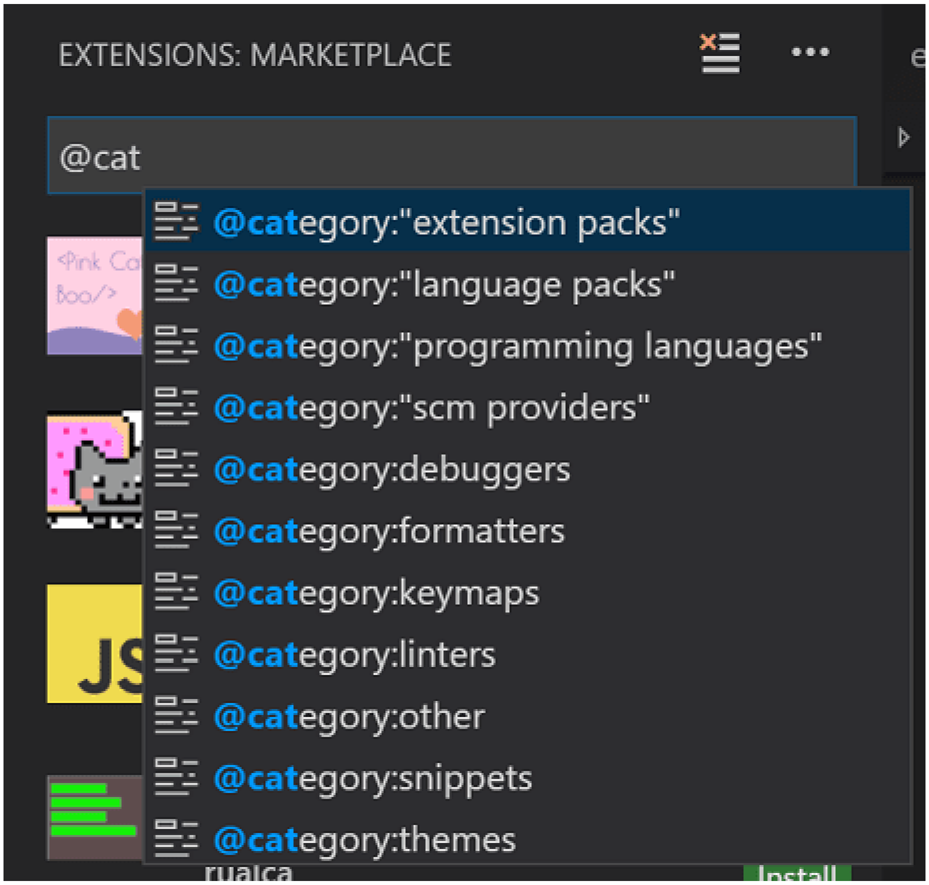
Read the Release Notes to know what has changed over the last few versions of this extension.

This extension adds rich language support for the Go language to VS Code, including:

- Completion Lists (using `gocode`)

You can filter on category and tag by using `category:` and `tag:`.

Supported categories are: `[Programming Languages, Snippets, Linters, Themes, Debuggers, Formatters, Keymaps, SCM Providers, Other, Extension Packs, Language Packs, Data Science, Machine Learning, Visualization, Notebooks, Education, Testing]`. They can be accessed through IntelliSense in the extensions search box:



Note that you must surround the category name in quotes if it is more than one word (for example, `category:"SCM Providers"`).

Tags may contain any string and are not provided by IntelliSense, so review the Marketplace to find helpful tags.

Install from a VSIX

You can manually install a VS Code extension packaged in a `.vsix` file. Using the **Install from VSIX** command in the Extensions view command dropdown, or the **Extensions: Install from VSIX** command in the **Command Palette**, point to the `.vsix` file.

You can also install using the VS Code `--install-extension` command-line switch providing the path to the `.vsix` file.

```
code --install-extension myextension.vsix
```

You may provide the `--install-extension` multiple times on the command line to install multiple extensions at once.

If you'd like to learn more about packaging and publishing extensions, see our Publishing Extensions article in the Extension API.

Workspace recommended extensions

A good set of extensions can make working with a particular workspace or programming language more productive and you'd often like to share this list with your team or colleagues. You can create a recommended list of extensions for a workspace with the **Extensions: Configure Recommended Extensions (Workspace Folder)** command.

In a single folder workspace, the command creates an `extensions.json` file located in the workspace `.vscode` folder where you can add a list of extensions identifiers ({publisherName}.{extensionName}).

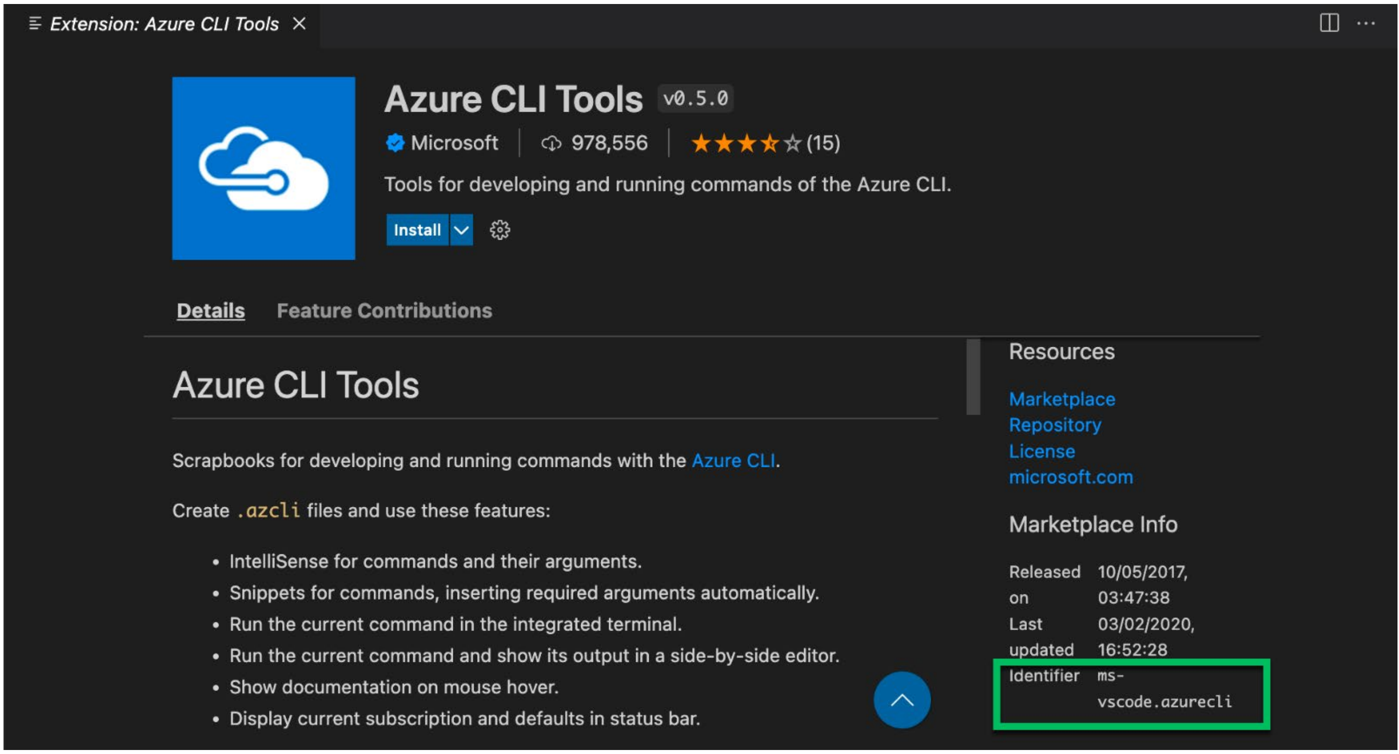
In a multi-root workspace, the command will open your `.code-workspace` file where you can list extensions under `extensions.recommendations`. You can still add extension recommendations to individual folders in a multi-root workspace by using the **Extensions: Configure Recommended Extensions (Workspace Folder)** command.

An example `extensions.json` could be:

```
{
  "recommendations": ["dbaeumer.vscode-eslint", "esbenp.prettier-vscode"]
}
```

which recommends a linter extension and a code formatter extension.

An extension is identified using its publisher name and extension identifier `publisher.extension`. You can see the name on the extension's detail page. VS Code will provide you with auto-completion for installed extensions inside these files.



Azure CLI Tools v0.5.0

Microsoft | 978,556 | ★★★★☆ (15)

Tools for developing and running commands of the Azure CLI.

Azure CLI Tools

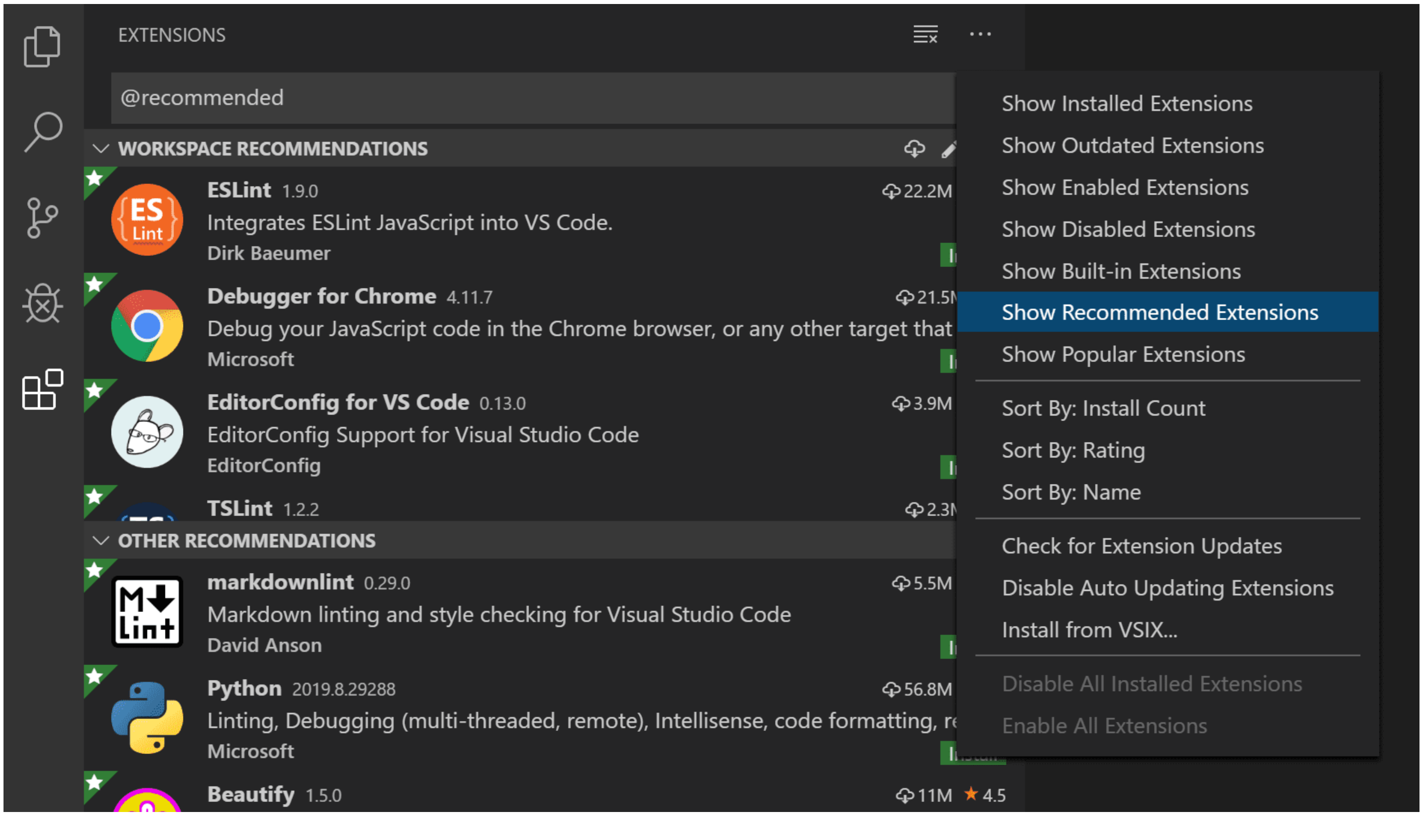
Scrapbooks for developing and running commands with the Azure CLI.

Create `.azcli` files and use these features:

- IntelliSense for commands and their arguments.
- Snippets for commands, inserting required arguments automatically.
- Run the current command in the integrated terminal.
- Run the current command and show its output in a side-by-side editor.
- Show documentation on mouse hover.
- Display current subscription and defaults in status bar.

Released on	10/05/2017, 03:47:38
Last updated	03/02/2020, 16:52:28
Identifier	ms-vscode.azurecli

VS Code prompts a user to install the recommended extensions when a workspace is opened for the first time. The user can also review the list with the **Extensions: Show Recommended Extensions** command.



ESLint 1.9.0 22.2M
Integrates ESLint JavaScript into VS Code.
Dirk Baeumer

Debugger for Chrome 4.11.7 21.5M
Debug your JavaScript code in the Chrome browser, or any other target that
Microsoft

EditorConfig for VS Code 0.13.0 3.9M
EditorConfig Support for Visual Studio Code
EditorConfig

TSLint 1.2.2 2.3M

markdownlint 0.29.0 5.5M
Markdown linting and style checking for Visual Studio Code
David Anson

Python 2019.8.29288 56.8M
Linting, Debugging (multi-threaded, remote), Intellisense, code formatting, r
Microsoft

Beautify 1.5.0 11M ★ 4.5

Next steps

Here are a few topics you may find interesting...

- Extension API - Start learning about the VS Code extension API.
- Your First Extension - Try creating a simple Hello World extension.
- Publishing to the Marketplace - Publish your own extension to the VS Code Marketplace.

Common questions

Where are extensions installed?

Extensions are installed in a per user extensions folder. Depending on your platform, the location is in the following folder:

- **Windows** `%USERPROFILE%\.vscode\extensions`
- **macOS** `~/.vscode/extensions`
- **Linux** `~/.vscode/extensions`

You can change the location by launching VS Code with the `--extensions-dir <dir>` command-line option.

Whenever I try to install any extension, I get a connect ETIMEDOUT error

You may see this error if your machine is going through a proxy server to access the Internet. See the Proxy server support section in the setup topic for details.

Can I download an extension directly from the Marketplace?

Some users prefer to download an extension once from the Marketplace and then install it multiple times from a local share. This is useful when there are connectivity concerns or if your development team wants to use a fixed set of extensions.

To download an extension, navigate to the details page for the specific extension within the Marketplace. On that page, there is a **Download Extension** link in the **Resources** section, which is located on the right-hand side of the page.

Once downloaded, you can then install the extension via the **Install from VSIX** command in the Extensions view command dropdown.

Can I stop VS Code from providing extension recommendations?

Yes, if you would prefer to not have VS Code display extension recommendations in the Extensions view or through notifications, you can modify the following settings:

- `extensions.showRecommendationsOnlyOnDemand` - Set to true to remove the **RECOMMENDED** section.
- `extensions.ignoreRecommendations` - Set to true to silence extension recommendation notifications.

The **Show Recommended Extensions** command is always available if you want to see recommendations.

Can I trust extensions from the Marketplace?

The Marketplace runs a virus scan on each extension package that's published to ensure its safety. The virus scan is run for each new extension and for each extension update. Until the scan is all clear, the extension won't be published in the Marketplace for public usage.

The Marketplace also prevents extension authors from name-squatting on official publishers such as Microsoft and RedHat.

If a malicious extension is reported and verified, or a vulnerability is found in an extension dependency:

1. The extension is removed from the Marketplace.
2. The extension is added to a kill list so that if it has been installed, it will be automatically uninstalled by VS Code.

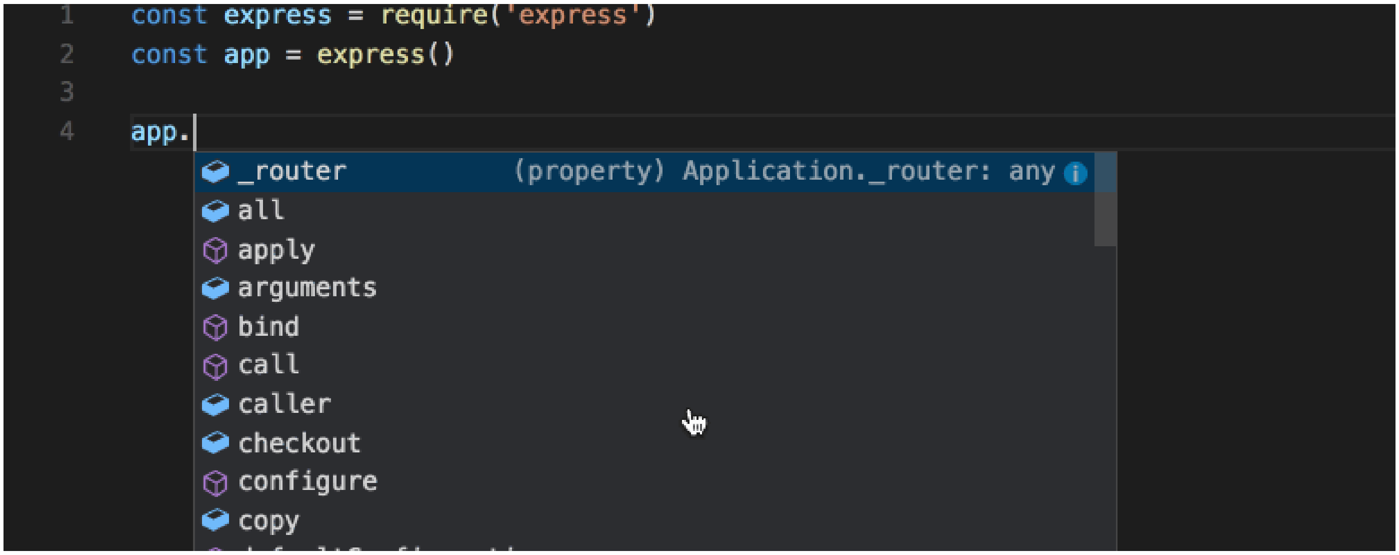
The Marketplace also provides you with resources to make an informed decision about the extensions you install:

- **Ratings & Review** - Read what others think about the extension.
- **Q & A** - Review existing questions and the level of the publisher's responsiveness. You can also engage with the extension's publisher(s) if you have concerns.
- **Issues, Repository, and License** - Check if the publisher has provided these and if they have the support you expect.

If you do see an extension that looks suspicious, you can report the extension to the Marketplace with the **Report Abuse** link at the bottom of the extension **More Info** section.

IntelliSense

IntelliSense is a general term for various code editing features including: code completion, parameter info, quick info, and member lists. IntelliSense features are sometimes called by other names such as "code completion", "content assist", and "code hinting."



```
const express = require('express')
const app = express()

app.
```

IntelliSense for your programming language

Visual Studio Code IntelliSense is provided for JavaScript, TypeScript, JSON, HTML, CSS, SCSS, and Less out of the box. VS Code supports word based completions for any programming language but can also be configured to have richer IntelliSense by installing a language extension.

Below are the most popular language extensions in the Marketplace. Select an extension tile below to read the description and reviews to decide which extension is best for you.

Python
90.7M
ms-python

C/C++
49.1M
ms-vscode

C#
22.5M
ms-dotnettools

Extension Pack for Java
20.4M
vscjava

Go
10.0M
golang

Dart
7.1M
Dart-Code

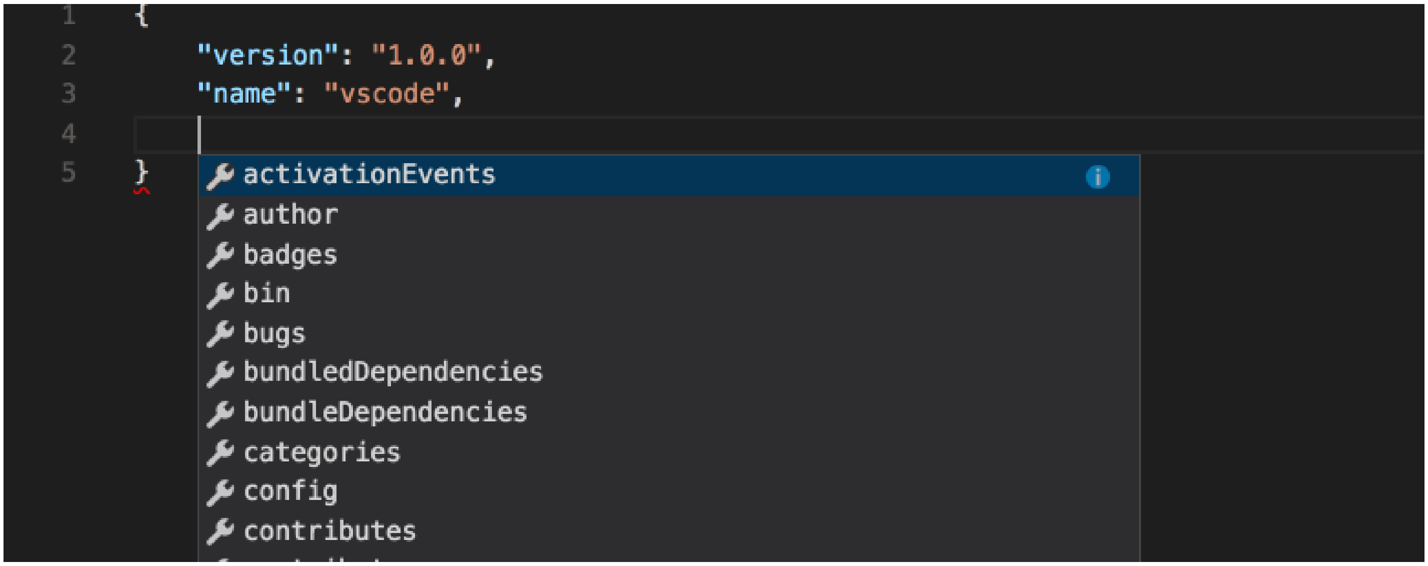
PHP Extension Pack
4.2M
xdebug

Ruby LSP
162.7K
Shopify

IntelliSense features

VS Code IntelliSense features are powered by a language service. A language service provides intelligent code completions based on language semantics and an analysis of your source code. If a language service knows possible completions, the IntelliSense suggestions will pop up as you type. If you continue typing characters, the list of members (variables, methods, etc.) is filtered to only include members containing your typed characters. Pressing `Tab` or `Enter` will insert the selected member.

You can trigger IntelliSense in any editor window by typing `Ctrl+Space` or by typing a trigger character (such as the dot character (`.`) in JavaScript).

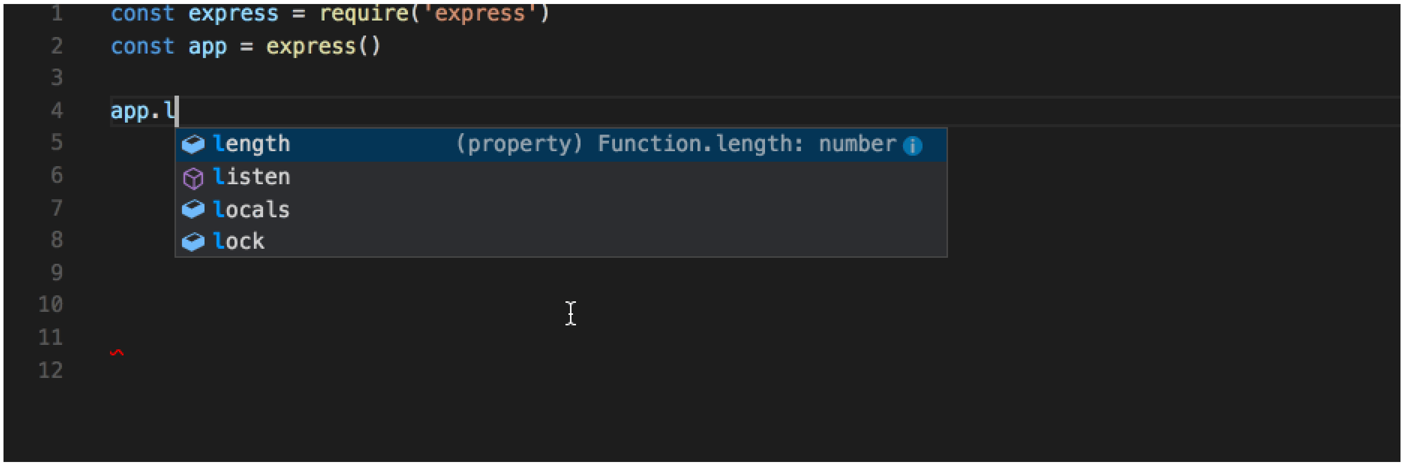


```
{
    "version": "1.0.0",
    "name": "vscode",
    
}
```

Tip: The suggestions widget supports CamelCase filtering, meaning you can type the letters which are upper cased in a method name to limit the suggestions. For example, "cra" will quickly bring up "createApplication".

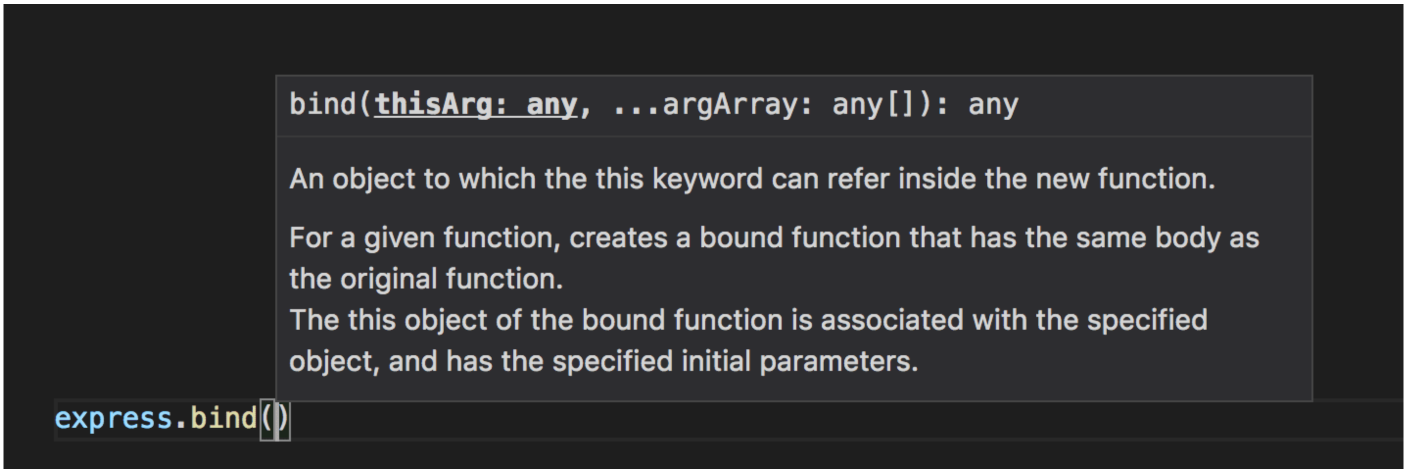
If you prefer, you can turn off IntelliSense while you type. See Customizing IntelliSense below to learn how to disable or customize VS Code's IntelliSense features.

As provided by the language service, you can see **quick info** for each method by either pressing `Ctrl+Space` or clicking the info icon. The accompanying documentation for the method will now expand to the side. The expanded documentation will stay so and will update as you navigate the list. You can close this by pressing `Ctrl+Space` again or by clicking on the close icon.



```
const express = require('express')
const app = express()

app.l
```

After choosing a method you are provided with **parameter info**.



```
bind(thisArg: any, ...argArray: any[]): any
```

An object to which the this keyword can refer inside the new function.

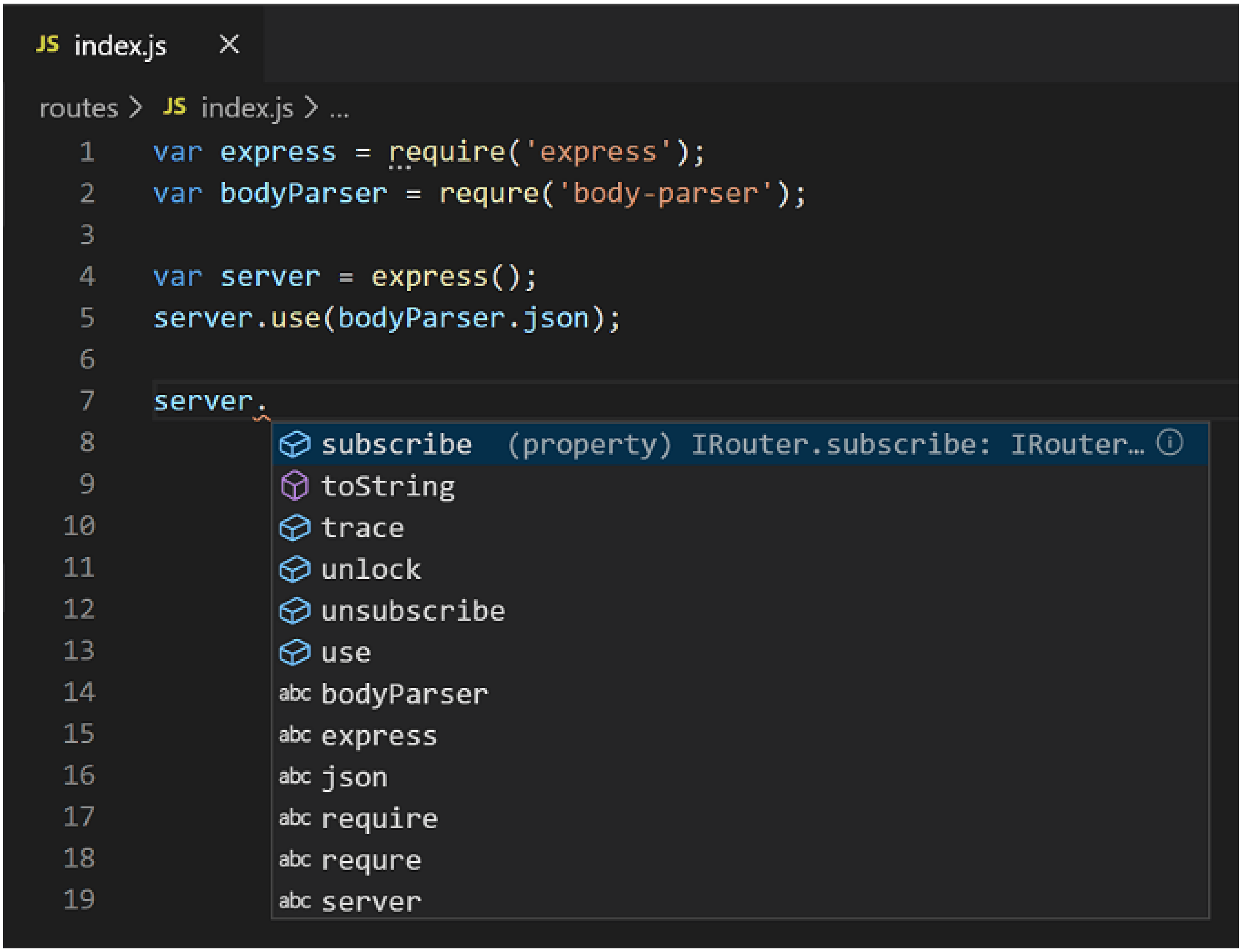
For a given function, creates a bound function that has the same body as the original function.
The this object of the bound function is associated with the specified object, and has the specified initial parameters.

```
express.bind()
```

When applicable, a language service will surface the underlying types in the quick info and method signatures. In the image above, you can see several `any` types. Because JavaScript is dynamic and doesn't need or enforce types, `any` suggests that the variable can be of any type.

Types of completions

The JavaScript code below illustrates IntelliSense completions. IntelliSense gives both inferred proposals and the global identifiers of the project. The inferred symbols are presented first, followed by the global identifiers (shown by the Word icon).



```
var express = require('express');
var bodyParser = requre('body-parser');

var server = express();
server.use(bodyParser.json);

server.
```

VS Code IntelliSense offers different types of completions, including language server suggestions, snippets, and simple word based textual completions.

Icon	Name	Symbol type
	Methods and Functions	`method`, `function`, `constructor`
	Variables	`variable`
	Fields	`field`
	Type parameters	`typeParameter`

Icon	Name	Symbol type
	Constants	`constant`
	Classes	`class`
	Interfaces	`interface`
	Structures	`struct`
	Events	`event`
	Operators	`operator`
	Modules	`module`
	Properties and Attributes	`property`
	Values and Enumerations	`value`, `enum`
	References	`reference`
	Keywords	`keyword`
	Files	`file`
	Folders	`folder`

Icon	Name	Symbol type
	Colors	`color`
	Unit	`unit`
	Snippet prefixes	`snippet`
	Words	`text`

Customizing IntelliSense

You can customize your IntelliSense experience in settings and key bindings.

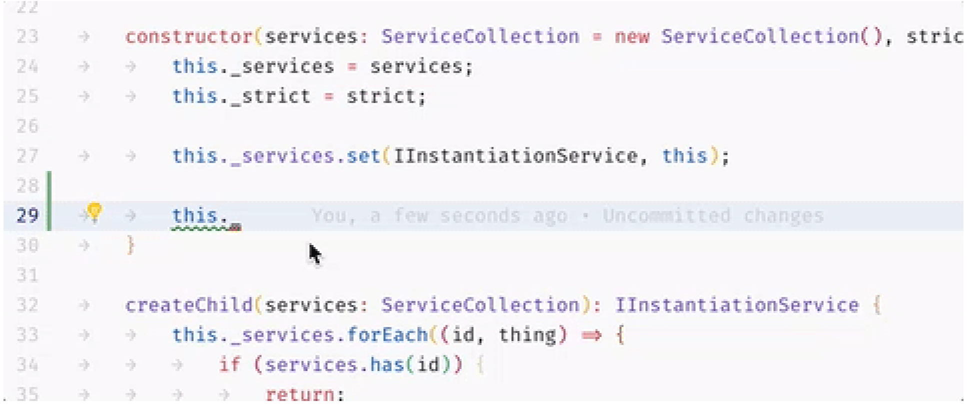
Settings

The settings shown below are the default settings. You can change these settings in your `settings.json` file as described in User and Workspace Settings.

```
{
    // Controls if quick suggestions should show up while typing
    "editor.quickSuggestions": {
        "other": true,
        "comments": false,
        "strings": false
    },

     // Controls whether suggestions should be accepted on commit characters. For example, in Java
Script, the semi-colon (`;`) can be a commit character that accepts a suggestion and types that ch
aracter.
    "editor.acceptSuggestionOnCommitCharacter": true,
    // Controls if suggestions should be accepted on 'Enter' - in addition to 'Tab'. Helps to avoid ambiguity between inserting new lines or accepting suggestions. The value 'smart' means only accept a suggestion with Enter when it makes a textual change

    "editor.acceptSuggestionOnEnter": "on",



    // Controls the delay in ms after which quick suggestions will show up.

    "editor.quickSuggestionsDelay": 10,



    // Controls if suggestions should automatically show up when typing trigger characters

    "editor.suggestOnTriggerCharacters": true,



    // Controls if pressing tab inserts the best suggestion and if tab cycles through other suggestions

    "editor.tabCompletion": "off",



    // Controls whether sorting favours words that appear close to the cursor

    "editor.suggest.localityBonus": true,



    // Controls how suggestions are pre-selected when showing the suggest list

    "editor.suggestSelection": "first",



    // Enable word based suggestions

    "editor.wordBasedSuggestions": true,



    // Enable parameter hints

    "editor.parameterHints.enabled": true,

}
```

Tab Completion

The editor supports "tab completion" which inserts the best matching completion when pressing `Tab`. This works regardless of the suggest widget showing or not. Also, pressing `Tab` after inserting a suggestions will insert the next best suggestion.



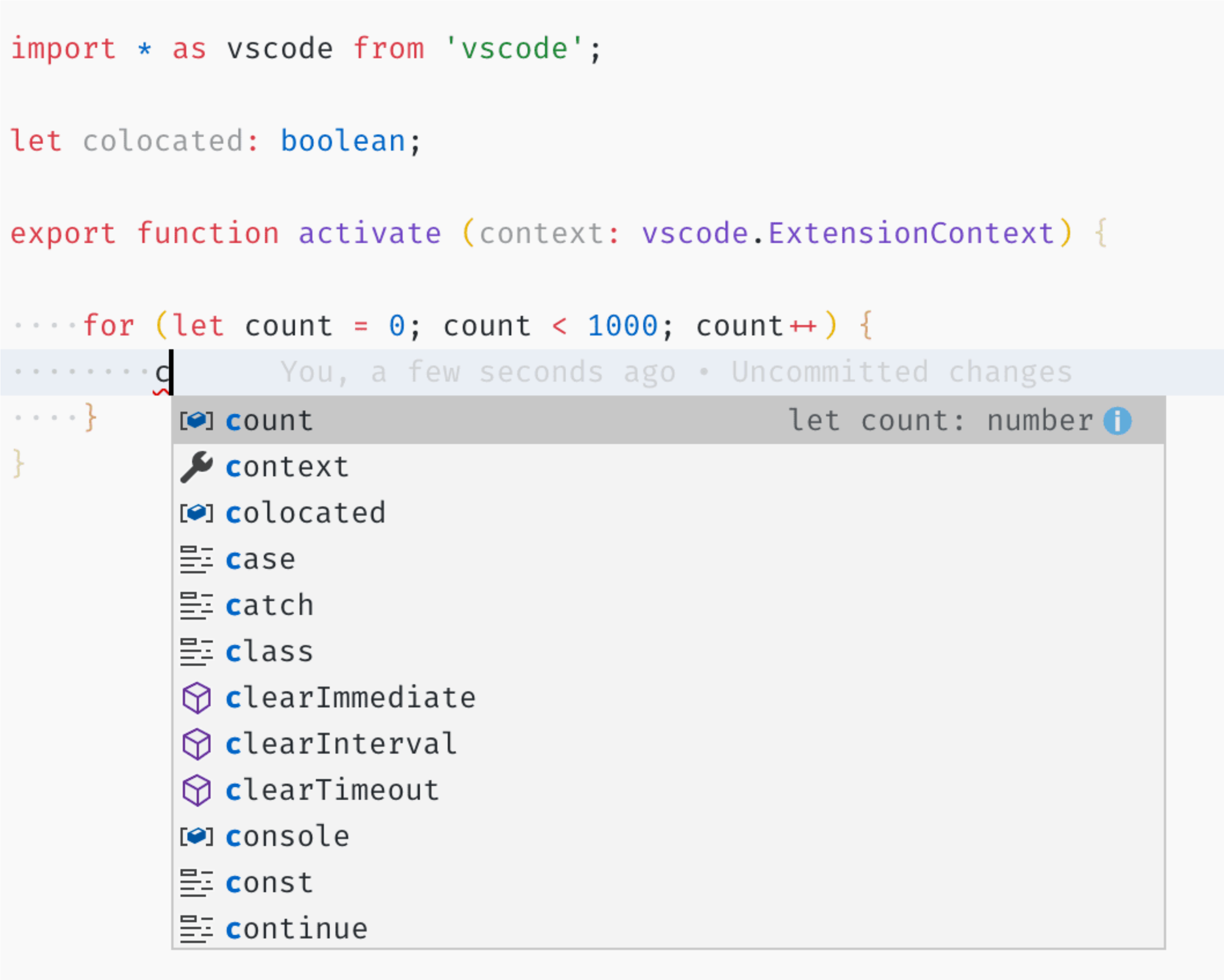
```
    constructor(services: ServiceCollection = new ServiceCollection(), stric
        this._services = services;
        this._strict = strict;

        this._services.set(IInstantiationService, this);

        this.        You, a few seconds ago • Uncommitted changes
    }

    createChild(services: ServiceCollection): IInstantiationService {
        this._services.forEach((id, thing) => {
            if (services.has(id)) {
                return;
```

By default, tab completion is disabled. Use the `editor.tabCompletion` setting to enable it. These values exist:

- `off` - (default) Tab completion is disabled.
- `on` - Tab completion is enabled for all suggestions and repeated invocations insert the next best suggestion.
- `onlySnippets` - Tab completion only inserts static snippets which prefix match the current line prefix.

Locality Bonus

Sorting of suggestions depends on extension information and on how well they match the current word you are typing. In addition, you can ask the editor to boost suggestions that appear closer to the cursor position, using the `editor.suggest.localityBonus` setting.



```
import * as vscode from 'vscode';

let colocated: boolean;

export function activate (context: vscode.ExtensionContext) {

····for (let count = 0; count < 1000; count++) {
········c        You, a few seconds ago • Uncommitted changes
····}
}
```

In above images you can see that `count`, `context`, and `colocated` are sorted based on the scopes in which they appear (loop, function, file).

Suggestion selection

By default, VS Code pre-selects the first suggestion in the suggestion list. If you'd like different behavior, for example, to always select the most recently used item in the suggestion list, you can use the `editor.suggestSelection` setting.

The available `editor.suggestSelection` values are:

- `first` - (default) Always select the top list item.
- `recentlyUsed` - The previously used item is selected unless a prefix (type to select) selects a different item.
- `recentlyUsedByPrefix` - Select items based on previous prefixes that have completed those suggestions.

Selecting the most recently used item is very useful as you can quickly insert the same completion multiple times.

"Type to select" means that the current prefix (roughly the text left of the cursor) is used to filter and sort suggestions. When this happens and when its result differs from the result of `recentlyUsed`, it will be given precedence.

When using the last option, `recentlyUsedByPrefix`, VS Code remembers which item was selected for a specific prefix (partial text). For example, if you typed `co` and then selected `console`, the next time you typed `co`, the suggestion `console` would be pre-selected. This lets you quickly map various prefixes to different suggestions, for example `co` -> `console` and `con` -> `const`.

Snippets in suggestions

By default, VS Code shows snippets and completion proposals in one widget. You can control the behavior with the `editor.snippetSuggestions` setting. To remove snippets from the suggestions widget, set the value to `"none"`. If you'd like to see snippets, you can specify the order relative to suggestions; at the top (`"top"`), at the bottom (`"bottom"`), or inline ordered alphabetically (`"inline"`). The default is `"inline"`.

Key bindings

The key bindings shown below are the default key bindings. You can change these in your `keybindings.json` file as described in Key Bindings.

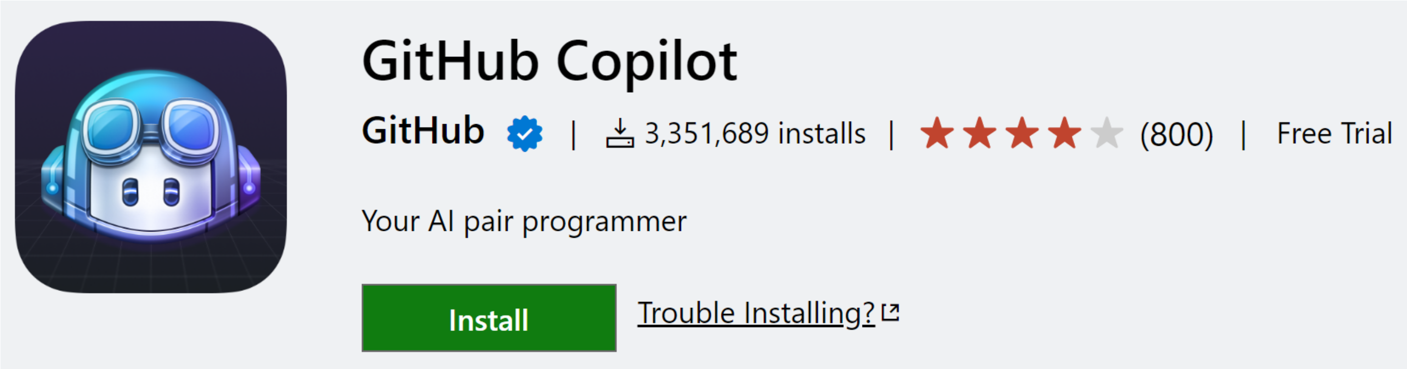
Note: There are many more key bindings relating to IntelliSense. Open the **Default Keyboard Shortcuts** (**File** > **Preferences** > **Keyboard Shortcuts**) and search for "suggest".

```
[
  {
    "key": "ctrl+space",
    "command": "editor.action.triggerSuggest",
    "when": "editorHasCompletionItemProvider && editorTextFocus && !editorReadonly"
  },
  {
    "key": "ctrl+space",
    "command": "toggleSuggestionDetails",
    "when": "editorTextFocus && suggestWidgetVisible"
  },
  {
    "key": "ctrl+alt+space",
    "command": "toggleSuggestionFocus",
    "when": "editorTextFocus && suggestWidgetVisible"
  }
]
```

Enhance completions with AI

In VS Code, you can enhance your coding with artificial intelligence (AI), such as suggestions for lines of code or entire functions, fast documentation creation, and help creating code-related artifacts like tests.

GitHub Copilot is an AI-powered code completion tool that helps you write code faster and smarter. You can use the GitHub Copilot extension in VS Code to generate code, or to learn from the code it generates.



GitHub Copilot

GitHub | 3,351,689 installs | ★★★★☆ (800) | Free Trial

Your AI pair programmer

You can learn more about how to get started with Copilot in the Copilot documentation.

Troubleshooting

If you find IntelliSense has stopped working, the language service may not be running. Try restarting VS Code and this should solve the issue. If you are still missing IntelliSense features after installing a language extension, open an issue in the repository of the language extension.

Tip: For configuring and troubleshooting JavaScript IntelliSense, see the JavaScript documentation.

A particular language extension may not support all the VS Code IntelliSense features. Review the extension's README to find out what is supported. If you think there are issues with a language extension, you can usually find the issue repository for an extension through the VS Code Marketplace. Navigate to the extension's Details page and select the **Support** link.

Next steps

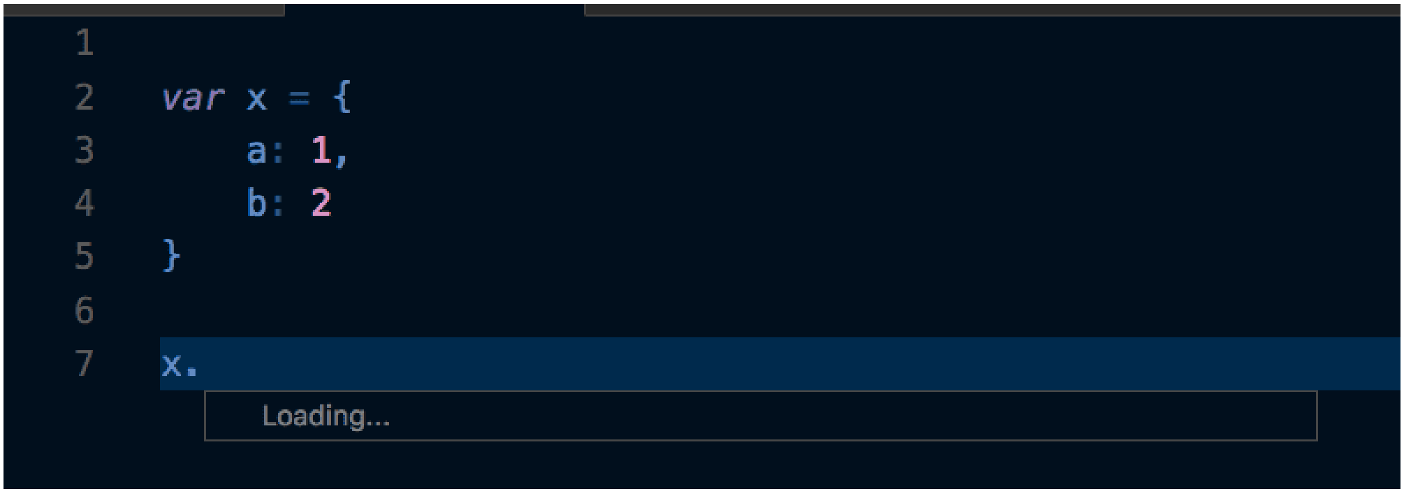
IntelliSense is just one of VS Code's powerful features. Read on to learn more:

- JavaScript - Get the most out of your JavaScript development, including configuring IntelliSense.
- Node.js - See an example of IntelliSense in action in the Node.js walkthrough.
- Debugging - Learn how to set up debugging for your application.

- Creating Language extensions - Learn how to create extensions that add IntelliSense for new programming languages.
- Artificial Intelligence - Learn how to use AI with GitHub Copilot to enhance your coding.

Common questions

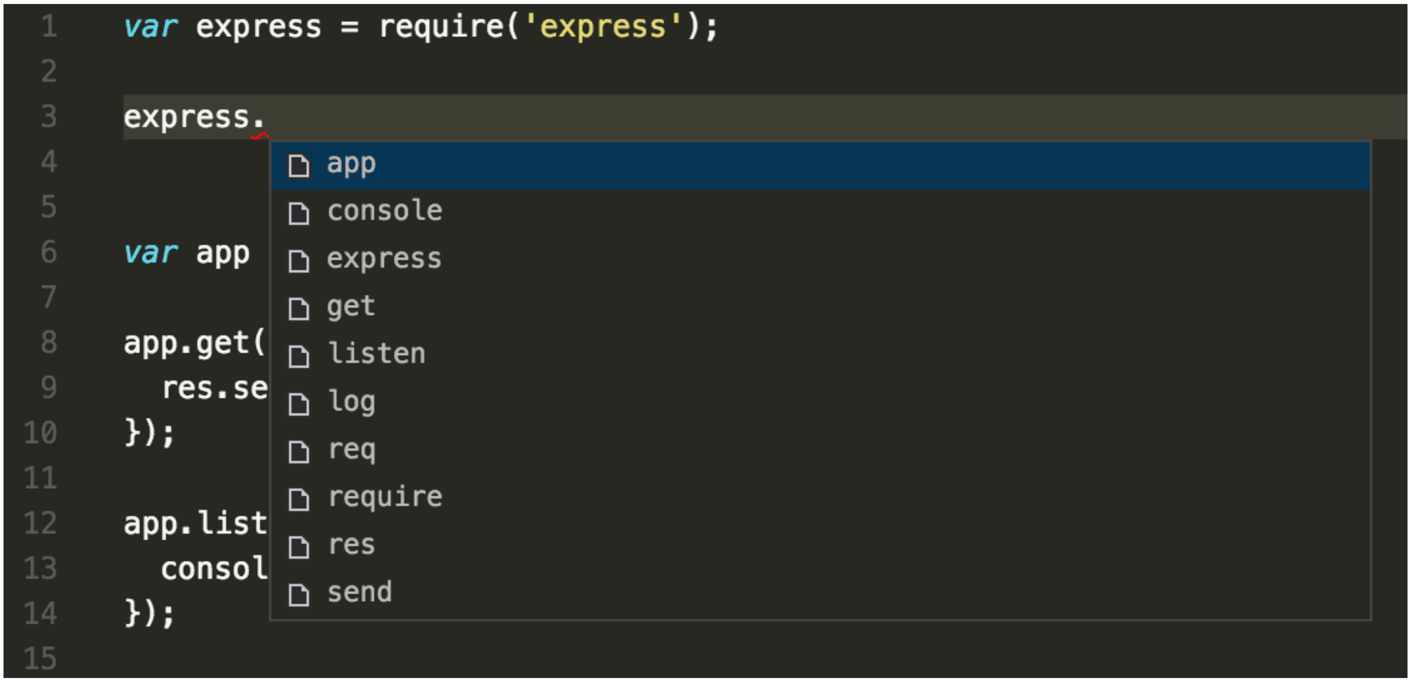
Why am I not getting any suggestions?



```
var x = {
    a: 1,
    b: 2
}

x.
```

This can be caused by a variety of reasons. First, try restarting VS Code. If the problem persists, consult the language extension's documentation. For JavaScript specific troubleshooting, please see the JavaScript language topic.

Why am I not seeing method and variable suggestions?



```
var express = require('express');

express.


var app

app.get(
  res.se
});

app.list
  consol
});
```

This issue is caused by missing type declaration (typings) files in JavaScript. You can check if a type declaration file package is available for a specific library by using the TypeSearch site. There is more information about this issue in the JavaScript language topic. For other languages, please consult the extension's documentation.

Code Navigation

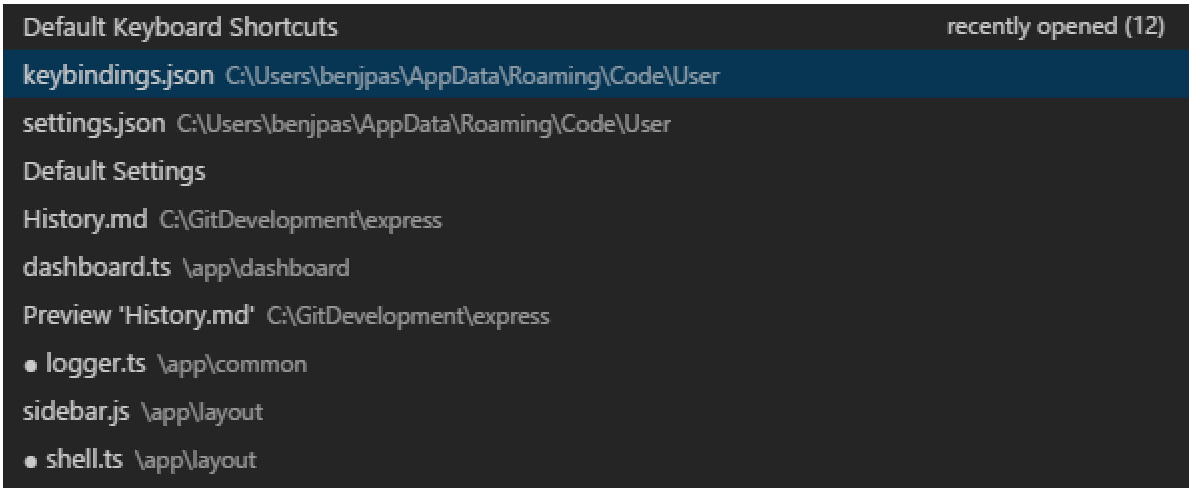
Visual Studio Code has a high productivity code editor which, when combined with programming language services, gives you the power of an IDE and the speed of a text editor. In this topic, we'll first describe VS Code's language intelligence features (suggestions, parameter hints, smart code navigation) and then show the power of the core text editor.

Quick file navigation

Tip: You can open any file by its name when you type `Ctrl+P` (**Quick Open**).

The Explorer is great for navigating between files when you are exploring a project. However, when you are working on a task, you will find yourself quickly jumping between the same set of files. VS Code provides two powerful commands to navigate in and across files with easy-to-use key bindings.

Hold `Ctrl` and press `Tab` to view a list of all files open in an editor group. To open one of these files, use `Tab` again to pick the file you want to navigate to, then release `Ctrl` to open it.

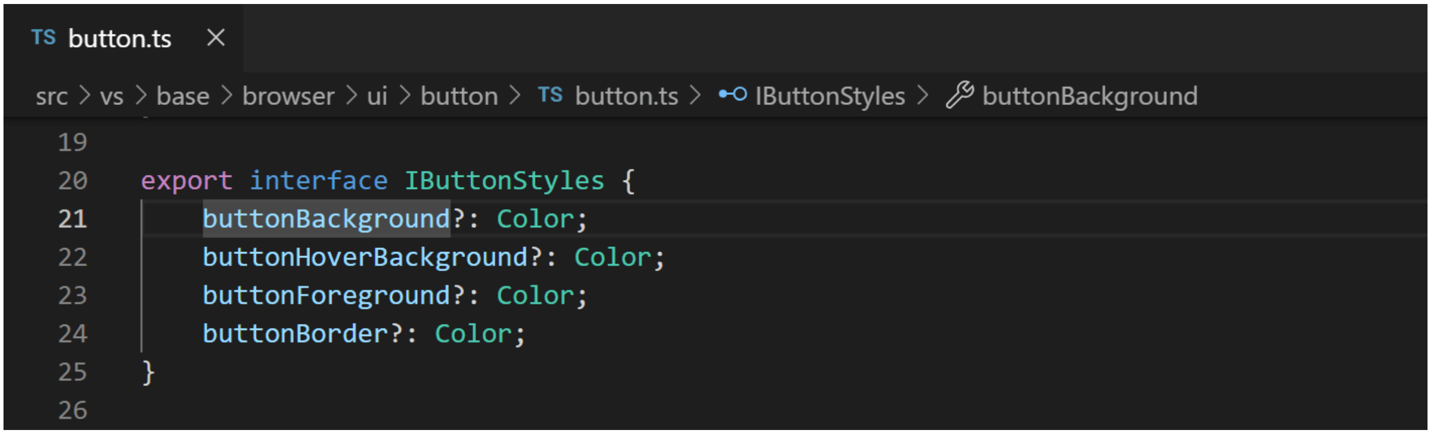


Default Keyboard Shortcuts recently opened (12)
keybindings.json C:\Users\benjpas\AppData\Roaming\Code\User
settings.json C:\Users\benjpas\AppData\Roaming\Code\User
Default Settings
History.md C:\GitDevelopment\express
dashboard.ts \app\dashboard
Preview 'History.md' C:\GitDevelopment\express
● logger.ts \app\common
sidebar.js \app\layout
● shell.ts \app\layout

Alternatively, you can use `Alt+Left` and `Alt+Right` to navigate between files and edit locations. If you are jumping around between different lines of the same file, these shortcuts allow you to navigate between those locations easily.

Breadcrumbs

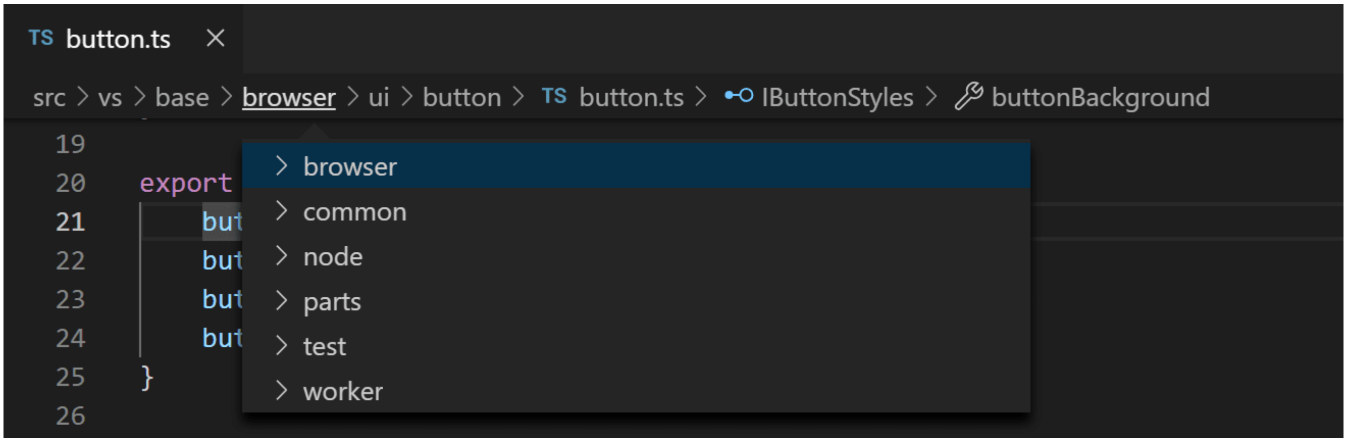
The editor has a navigation bar above its contents called Breadcrumbs. It shows the current location and allows you to quickly navigate between folders, files, and symbols.


TS button.ts
src > vs > base > browser > ui > button > TS button.ts > IButtonStyles > buttonBackground

```
export interface IButtonStyles {
    buttonBackground?: Color;
    buttonHoverBackground?: Color;
    buttonForeground?: Color;
    buttonBorder?: Color;
}
```

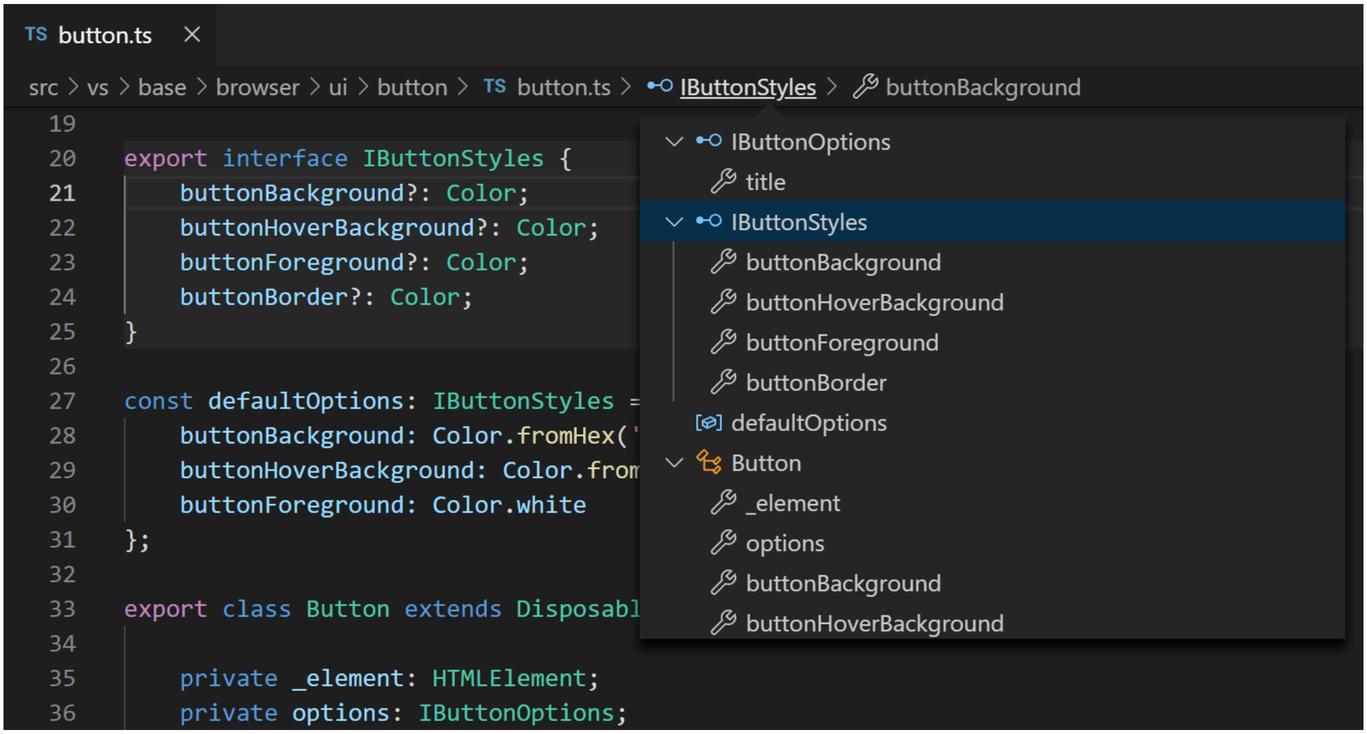
Breadcrumbs always show the file path and, with the help of language extensions, the symbol path up to the cursor position. The symbols shown are the same as in Outline view and Go to Symbol.

Selecting a breadcrumb in the path displays a dropdown with that level's siblings so you can quickly navigate to other folders and files.


TS button.ts
src > vs > base > browser > ui > button > TS button.ts > IButtonStyles > buttonBackground
> browser
> common
> node
> parts
> test
> worker

```
export
    but
    but
    but
    but
}
```

If the current file type has language support for symbols, you will see the current symbol path and a dropdown of other symbols at the same level and below.



```
export interface IButtonStyles {
    buttonBackground?: Color;
    buttonHoverBackground?: Color;
    buttonForeground?: Color;
    buttonBorder?: Color;
}

const defaultOptions: IButtonStyles =
    buttonBackground: Color.fromHex('
    buttonHoverBackground: Color.from
    buttonForeground: Color.white
};

export class Button extends Disposabl

    private _element: HTMLElement;
    private options: IButtonOptions;
```

You can turn off breadcrumbs with the **View** > **Show Breadcrumbs** toggle or with the `breadcrumbs.enabled` setting.

Breadcrumb customization

The appearance of breadcrumbs can be customized. If you have very long paths or are only interested in either file paths or symbols paths, you can use the `breadcrumbs.filePath` and `breadcrumbs.symbolPath` settings. Both support `on`, `off`, and `last` and they define if or what part of the path you see. By default, breadcrumbs show file and symbol icons to the left of the breadcrumb but you can remove the icons by setting `breadcrumbs.icons` to false.

Symbol order in Breadcrumbs

You can control how symbols are ordered in the Breadcrumbs dropdown with the `breadcrumbs.symbolSortOrder` settings.

Allowed values are:

- `position` - position in the file (default)
- `name` - alphabetical order
- `type` - symbol type order

Breadcrumb keyboard navigation

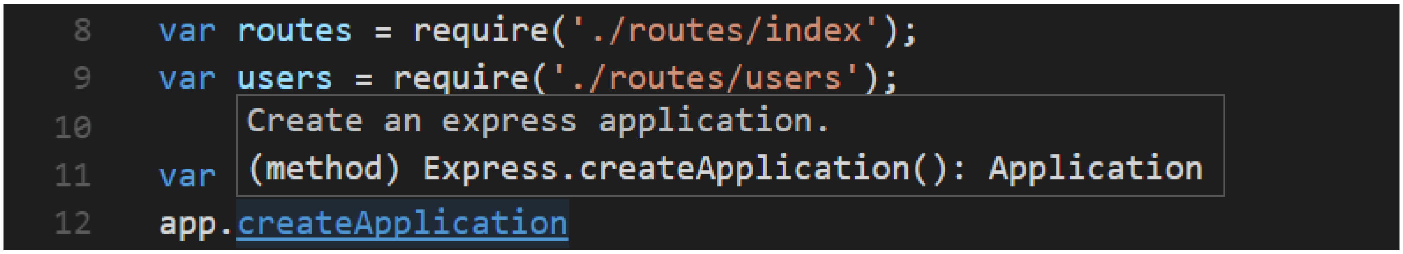
To interact with breadcrumbs, use the **Focus Breadcrumbs** command or press `Ctrl+Shift+.`. It will select that last element and open a dropdown that allows you to navigate to a sibling file or symbol. Use the `Left` and `Right` keyboard shortcuts to go to elements before or after the current element. When the dropdown appears, start typing - all matching elements will be highlighted and the best match will be selected for quick navigation.

You can also interact with breadcrumbs without the dropdown. Press `Ctrl+Shift+;` to focus the last element, use `Left` and `Right` to navigate, and use `Space` to reveal the element in the editor.

Go to Definition

If a language supports it, you can go to the definition of a symbol by pressing `F12`.

If you press `Ctrl` and hover over a symbol, a preview of the declaration will appear:



```
var routes = require('./routes/index');
var users = require('./routes/users');

var
app.createApplication
```

Create an express application.
(method) Express.createApplication(): Application

Tip: You can jump to the definition with `Ctrl+Click` or open the definition to the side with `Ctrl+Alt+Click`.

Go to Type Definition

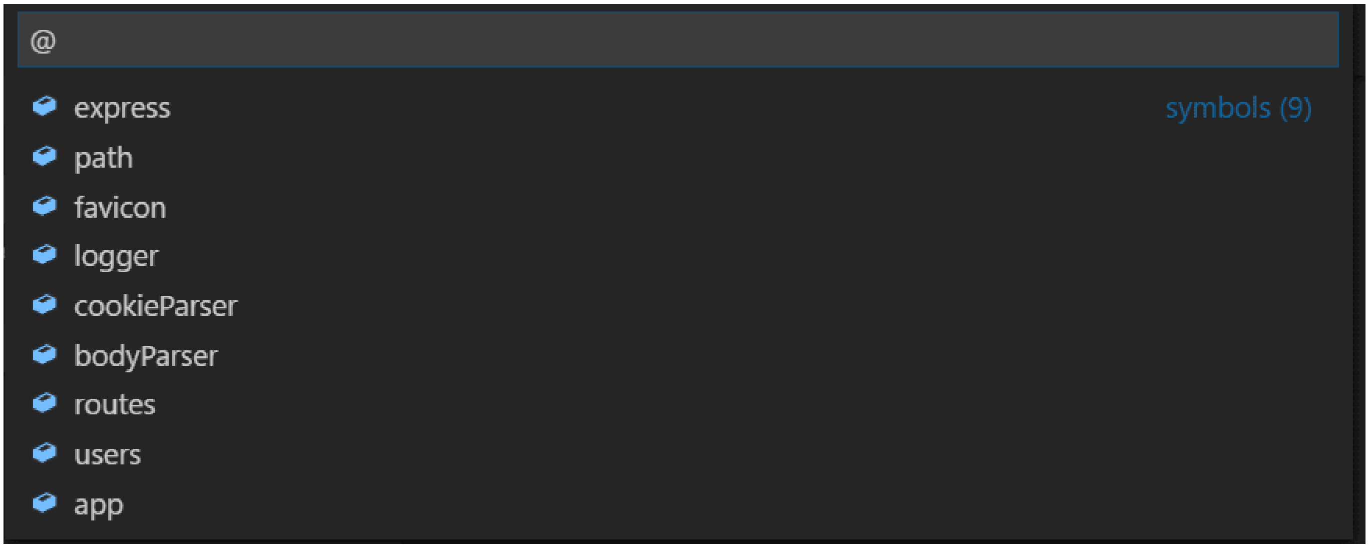
Some languages also support jumping to the type definition of a symbol by running the **Go to Type Definition** command from either the editor context menu or the **Command Palette**. This will take you to the definition of the type of a symbol. The command `editor.action.goToTypeDefinition` is not bound to a keyboard shortcut by default but you can add your own custom keybinding.

Go to Implementation

Languages can also support jumping to the implementation of a symbol by pressing `Ctrl+F12`. For an interface, this shows all the implementors of that interface and for abstract methods, this shows all concrete implementations of that method.

Go to Symbol

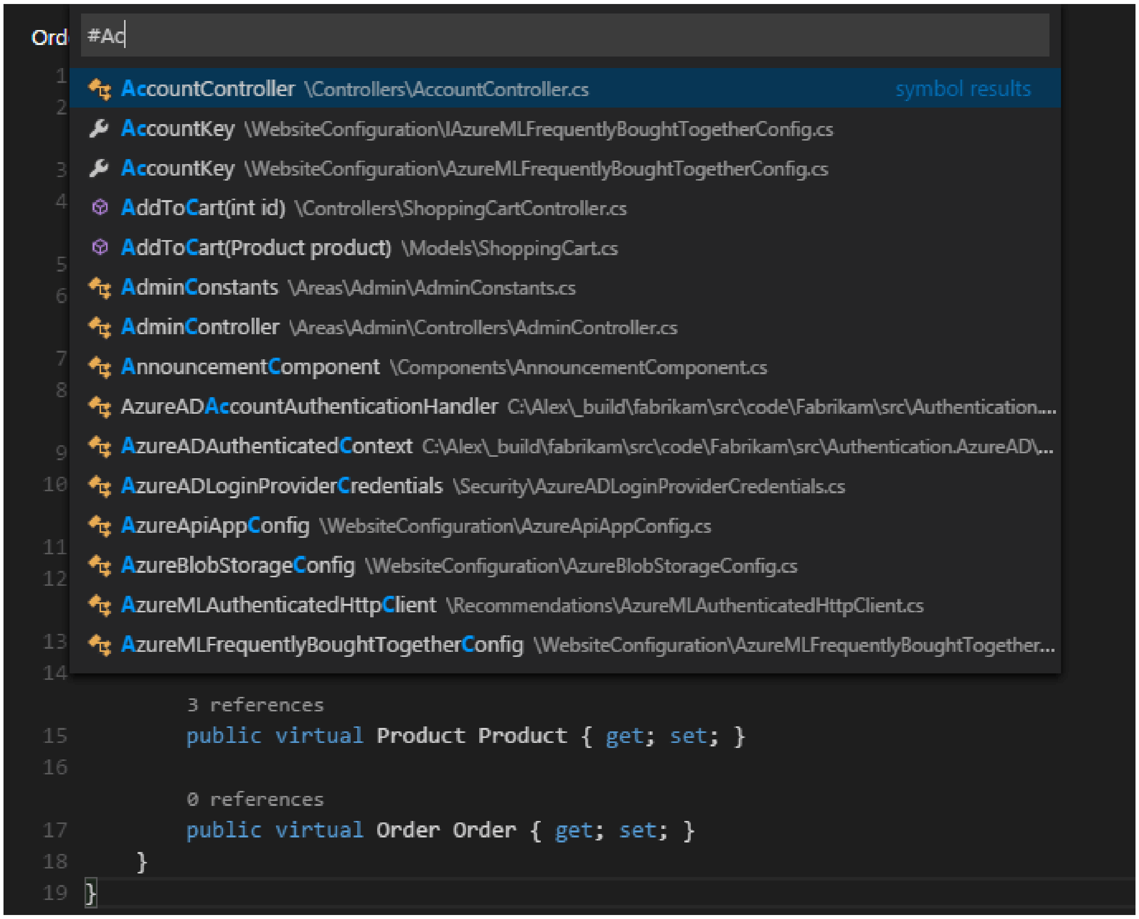
You can navigate symbols inside a file with `Ctrl+Shift+O`. By typing `:` the symbols will be grouped by category. Press `Up` or `Down` and navigate to the place you want.



```
@
express                                   symbols (9)
path
favicon
logger
cookieParser
bodyParser
routes
users
app
```

Open symbol by name

Some languages support jumping to a symbol across files with `Ctrl+T`. Type the first letter of a type you want to navigate to, regardless of which file contains it, and press `Enter`.

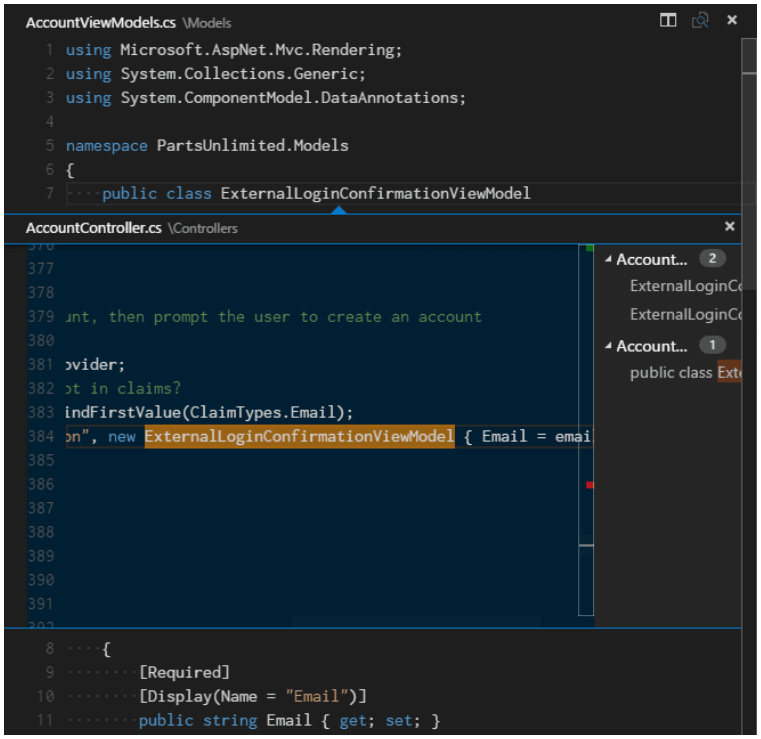


```
Ord  #Ac
AccountController \Controllers\AccountController.cs          symbol results
AccountKey \WebsiteConfiguration\IAzureMLFrequentlyBoughtTogetherConfig.cs
AccountKey \WebsiteConfiguration\AzureMLFrequentlyBoughtTogetherConfig.cs
AddToCart(int id) \Controllers\ShoppingCartController.cs
AddToCart(Product product) \Models\ShoppingCart.cs
AdminConstants \Areas\Admin\AdminConstants.cs
AdminController \Areas\Admin\Controllers\AdminController.cs
AnnouncementComponent \Components\AnnouncementComponent.cs
AzureADAccountAuthenticationHandler C:\Alex\_build\fabrikam\src\code\Fabrikam\src\Authentication....
AzureADAuthenticatedContext C:\Alex\_build\fabrikam\src\code\Fabrikam\src\Authentication.AzureAD\...
AzureADLoginProviderCredentials \Security\AzureADLoginProviderCredentials.cs
AzureApiAppConfig \WebsiteConfiguration\AzureApiAppConfig.cs
AzureBlobStorageConfig \WebsiteConfiguration\AzureBlobStorageConfig.cs
AzureMLAuthenticatedHttpClient \Recommendations\AzureMLAuthenticatedHttpClient.cs
AzureMLFrequentlyBoughtTogetherConfig \WebsiteConfiguration\AzureMLFrequentlyBoughtTogether...
```

```
        3 references
        public virtual Product Product { get; set; }

        0 references
        public virtual Order Order { get; set; }
    }
}
```

Peek

We think there's nothing worse than a big context switch when all you want is to quickly check something. That's why we support peeked editors. When you execute a **Go to References** search (via `Shift+F12`), or a **Peek Definition** (via `Alt+F12`), we embed the result inline:



```
using Microsoft.AspNet.Mvc.Rendering;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace PartsUnlimited.Models
{
    public class ExternalLoginConfirmationViewModel
```

```


unt, then prompt the user to create an account

ovider;
ot in claims?
indFirstValue(ClaimTypes.Email);
on", new ExternalLoginConfirmationViewModel { Email = emai
```

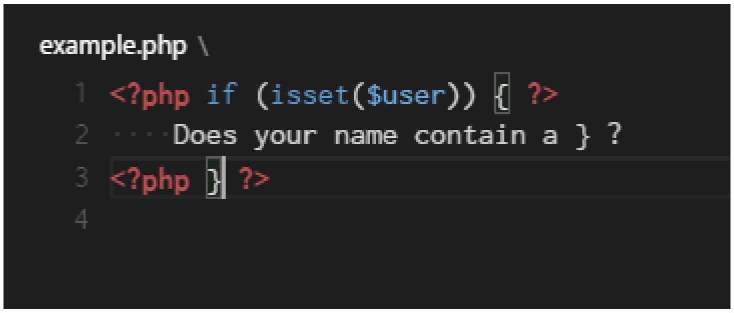
```
    {
        [Required]
        [Display(Name = "Email")]
        public string Email { get; set; }
```

You can navigate between different references in the peeked editor and make quick edits right there. Clicking on the peeked editor filename or double-clicking in the result list will open the reference in the outer editor.

Tip: Additionally, the peek window is closed if you press `Escape` or double-click in the peek editor region. You can disable this behavior with the `editor.stablePeek` setting.

Bracket matching

Matching brackets will be highlighted as soon as the cursor is near one of them.

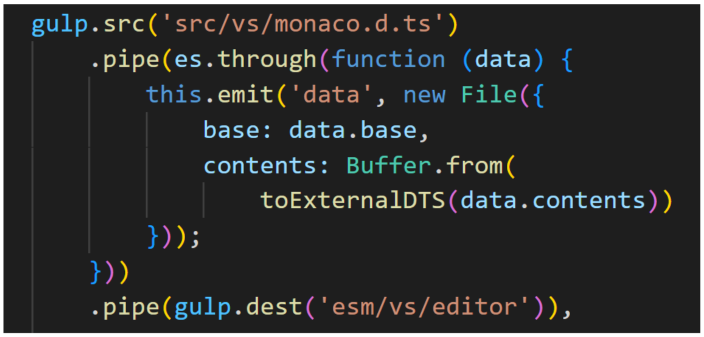


```
<?php if (isset($user)) { ?>
    Does your name contain a } ?
<?php } ?>

```

Tip: You can jump to the matching bracket with `Ctrl+Shift+\`

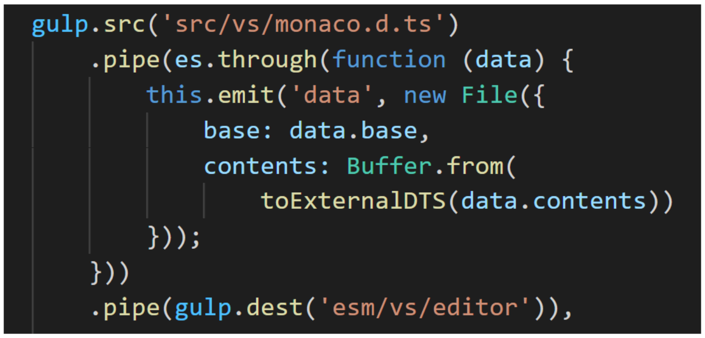
Bracket Pair Colorization

Matching bracket pairs can also be colorized by setting `editor.bracketPairColorization.enabled` to `true`.



```
gulp.src('src/vs/monaco.d.ts')
    .pipe(es.through(function (data) {
        this.emit('data', new File({
            base: data.base,
            contents: Buffer.from(
                toExternalDTS(data.contents))
        }));
    }))
    .pipe(gulp.dest('esm/vs/editor')),
```

Bracket Pair Colorization On



```
gulp.src('src/vs/monaco.d.ts')
    .pipe(es.through(function (data) {
        this.emit('data', new File({
            base: data.base,
            contents: Buffer.from(
                toExternalDTS(data.contents))
        }));
    }))
    .pipe(gulp.dest('esm/vs/editor')),
```

Bracket Pair Colorization Off

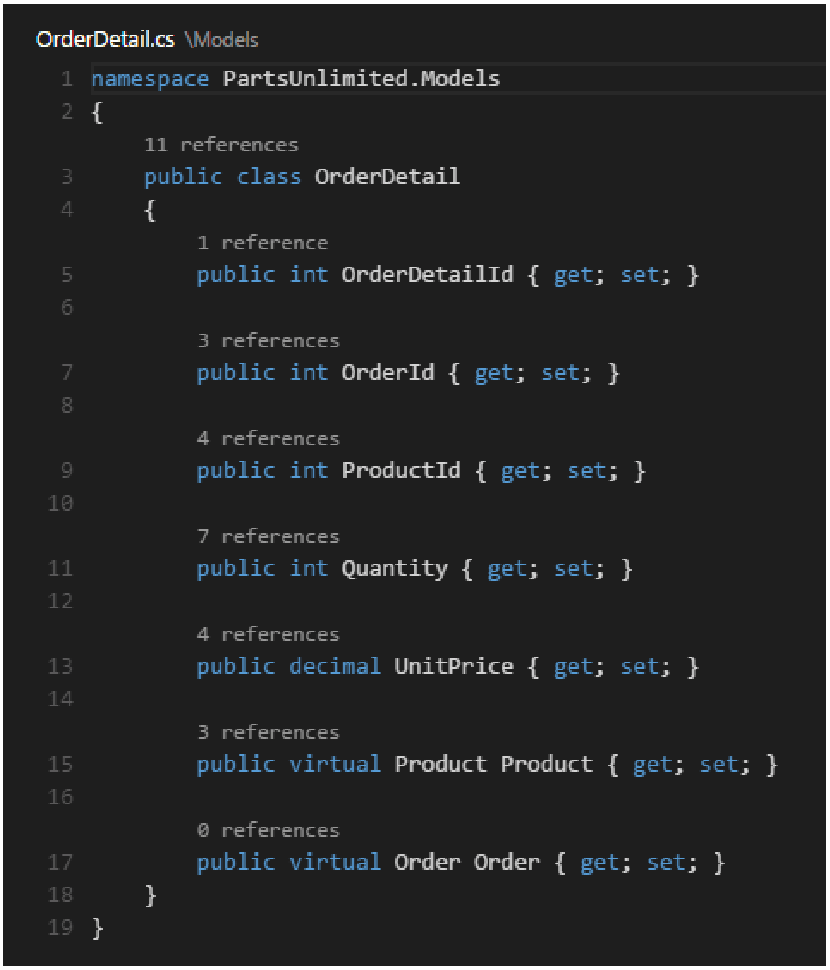
All colors are themeable and up to six colors can be configured.

You can use `workbench.colorCustomizations` to override these theme-contributed colors in your settings:

```
"workbench.colorCustomizations": {

    "editorBracketHighlight.foreground1": "#FFD700",

    "editorBracketHighlight.foreground2": "#DA70D6",

    "editorBracketHighlight.foreground3": "#179fff",

},
```

Reference information

Some languages like C# support inline reference information, that is updated live. This allows you to quickly analyze the impact of your edit or the popularity of your specific method or property throughout your project:



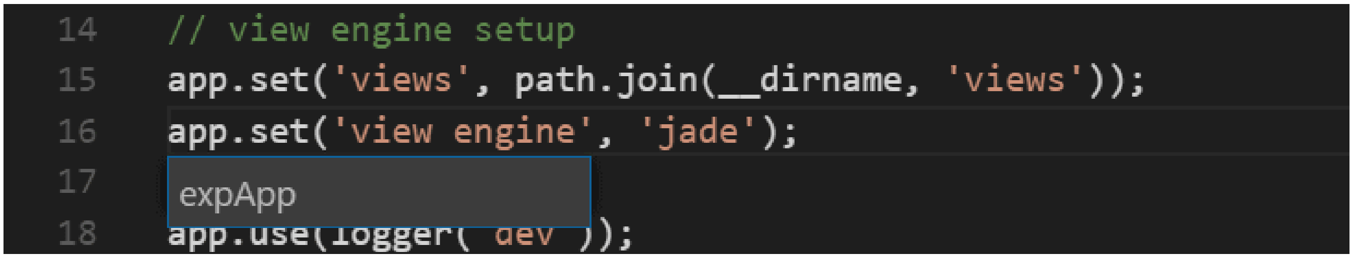
```
namespace PartsUnlimited.Models
{
    11 references
    public class OrderDetail
    {
        1 reference
        public int OrderDetailId { get; set; }

        3 references
        public int OrderId { get; set; }

        4 references
        public int ProductId { get; set; }

        7 references
        public int Quantity { get; set; }

        4 references
        public decimal UnitPrice { get; set; }

        3 references
        public virtual Product Product { get; set; }

        0 references
        public virtual Order Order { get; set; }
    }
}
```

Tip: Directly invoke the **Peek References** action by clicking on these annotations.

Tip: Reference information shown in CodeLens can be turned on or off through the `editor.codeLens` setting.

Rename symbol

Some languages support rename symbol across files. Press `F2` and then type the new desired name and press `Enter`. All usages of the symbol will be renamed, across files.

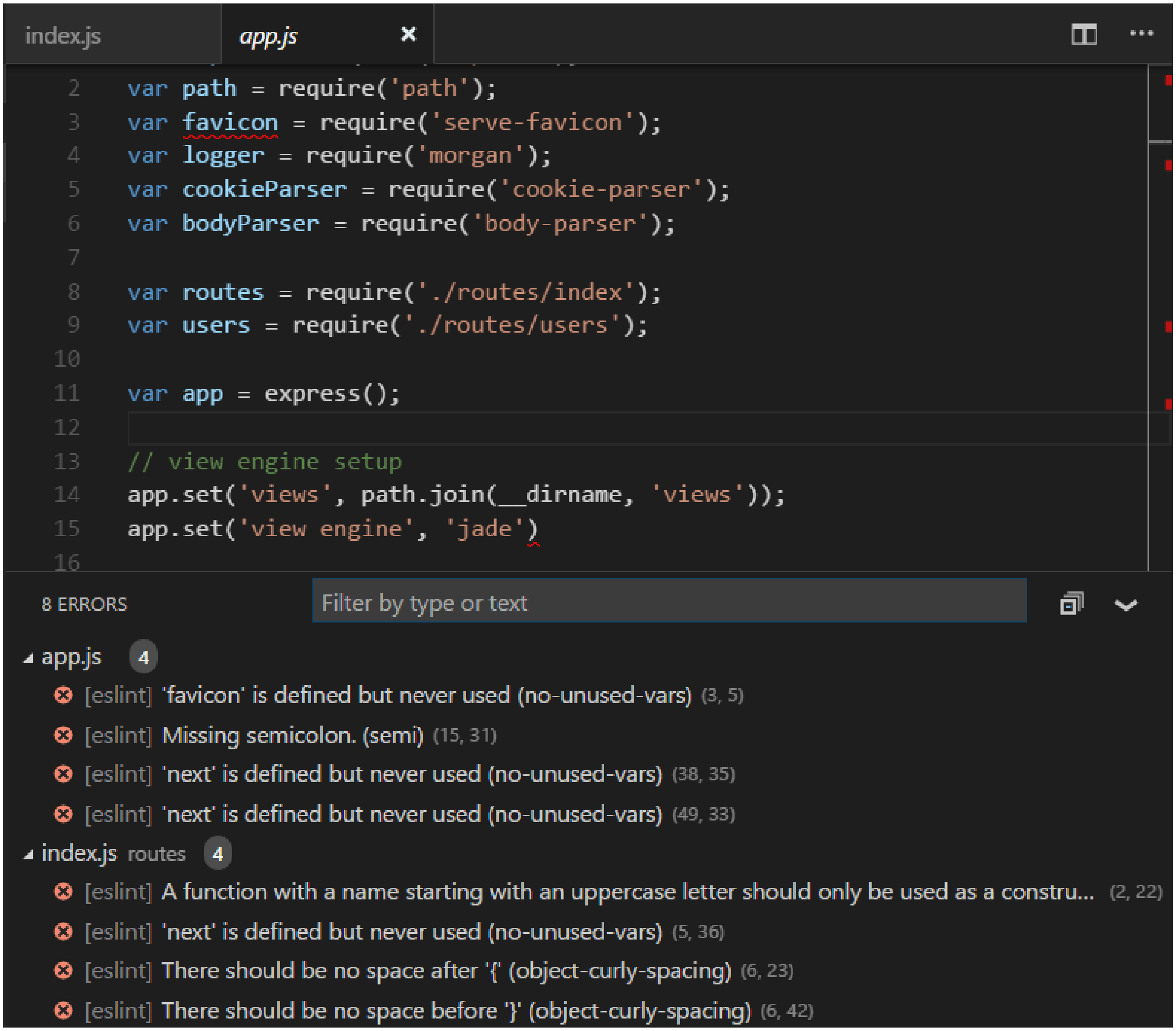


```
// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'jade');
expApp
app.use(logger('dev'));
```

Errors & warnings

Warnings or Errors can be generated either via configured tasks, by rich language services, or by linters, that constantly analyze your code in the background. Since we love bug-free code, warnings and errors show up in multiple places:

- In the Status Bar, there is a summary of all errors and warnings counts.
- You can click on the summary or press `Ctrl+Shift+M` to display the **PROBLEMS** panel with a list of all current errors.
- If you open a file that has errors or warnings, they will be rendered inline with the text and in the overview ruler.



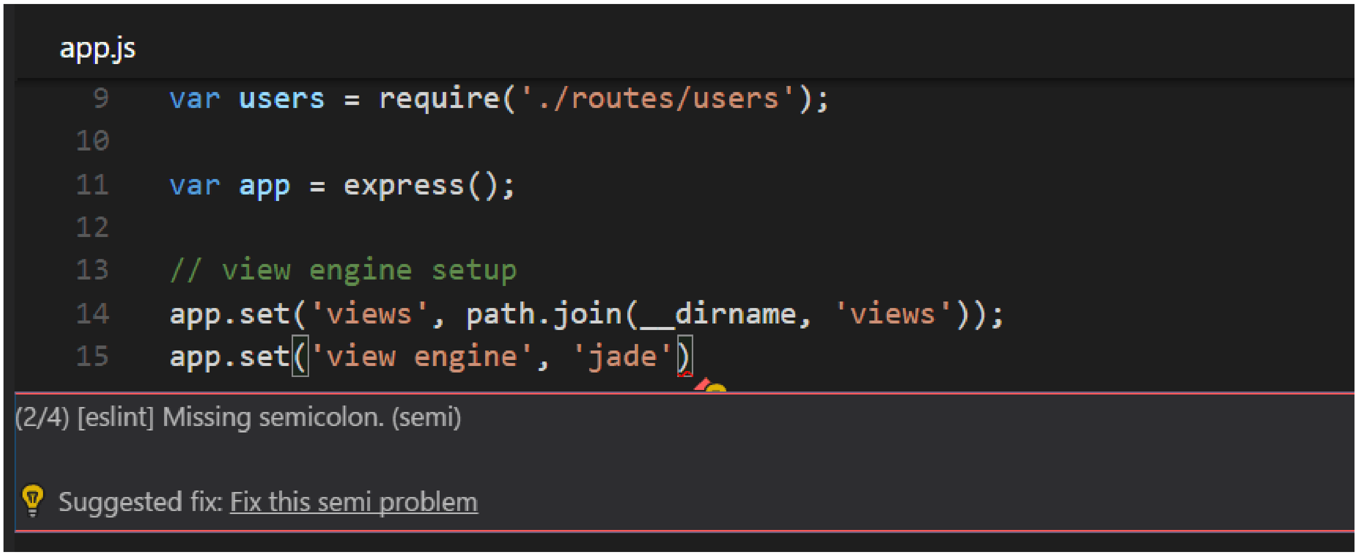
```
var path = require('path');
var favicon = require('serve-favicon');
var logger = require('morgan');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');

var routes = require('./routes/index');
var users = require('./routes/users');

var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'jade')

```

8 ERRORS

app.js 4
- [eslint] 'favicon' is defined but never used (no-unused-vars) (3, 5)
- [eslint] Missing semicolon. (semi) (15, 31)
- [eslint] 'next' is defined but never used (no-unused-vars) (38, 35)
- [eslint] 'next' is defined but never used (no-unused-vars) (49, 33)

index.js routes 4
- [eslint] A function with a name starting with an uppercase letter should only be used as a constru... (2, 22)
- [eslint] 'next' is defined but never used (no-unused-vars) (5, 36)
- [eslint] There should be no space after '{' (object-curly-spacing) (6, 23)
- [eslint] There should be no space before '}' (object-curly-spacing) (6, 42)

Tip: To loop through errors or warnings in the current file, you can press `F8` or `Shift+F8` which will show an inline zone detailing the problem and possible Code Actions (if available):



```
var users = require('./routes/users');

var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'jade')
```

(2/4) [eslint] Missing semicolon. (semi)

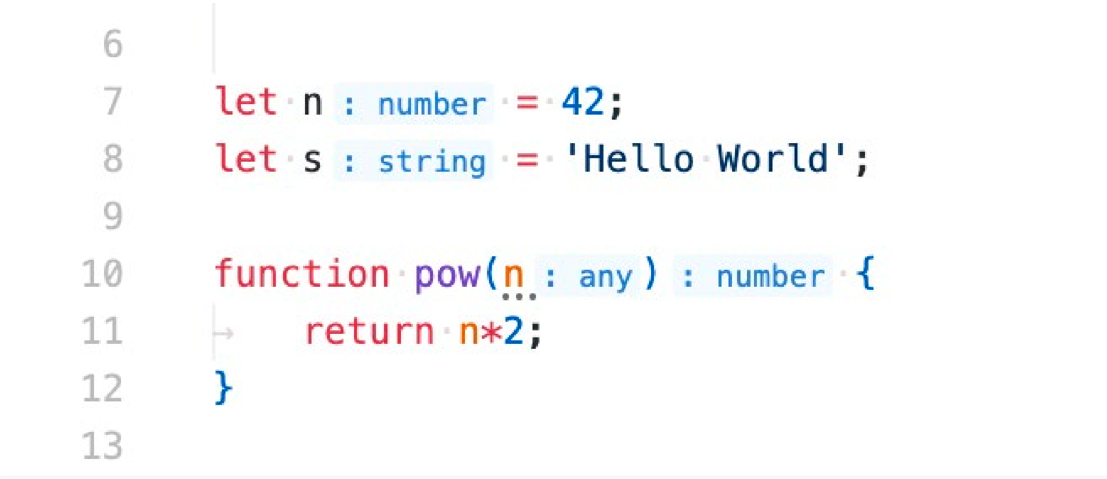
Suggested fix: Fix this semi problem

Code Action

Warnings and Errors can provide Code Actions (also known as Quick Fixes) to help fix issues. These will be displayed in the editor in the left margin as a lightbulb. Clicking on the lightbulb will either display the Code Action options or perform the action.

Inlay Hints

Some languages provide inlay hints: that is additional information about source code that is rendered inline. This is usually used to show infered types. The sample below shows inlay hints that display the inferred types of JavaScript variables and function return types.

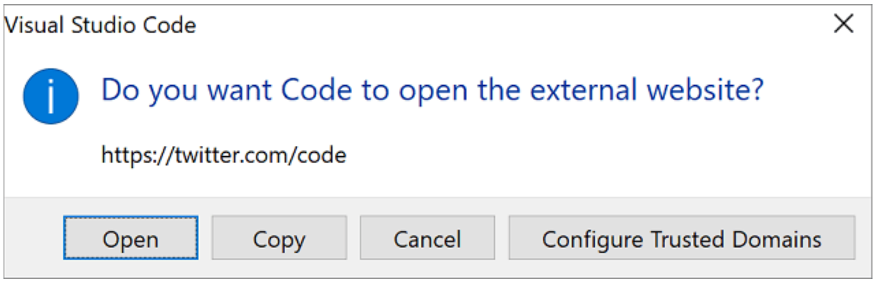


```

let n : number = 42;
let s : string = 'Hello World';

function pow(n : any) : number {
    return n*2;
}

```

Inlay hints can be enabled/disabled with the `editor.inlayHints.enabled` setting, the default is enabled. Extensions, like TypeScript or Rust, are needed to provide the actual inlay hint information.

Outgoing link protection

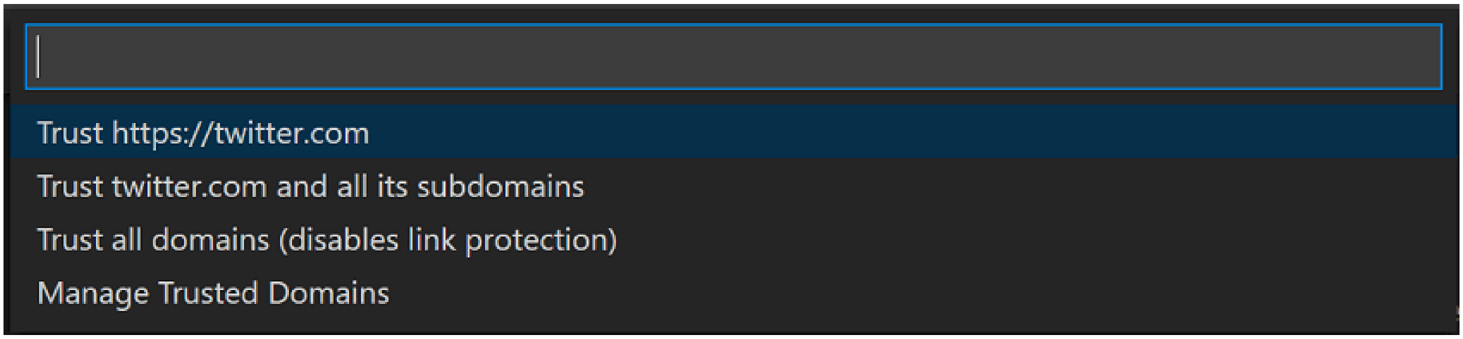
For your protection, VS Code displays a prompt before opening an outgoing website link from the editor.


Visual Studio Code

Do you want Code to open the external website?

https://twitter.com/code

You can proceed to the external website in your browser or have the options to copy the link or cancel the request. If you choose **Configure Trusted Domains**, a dropdown lets you trust the exact URL, trust the URL domain and subdomains, or trust all domains to disable outgoing link protection.


Trust https://twitter.com
Trust twitter.com and all its subdomains
Trust all domains (disables link protection)
Manage Trusted Domains

The option to **Manage Trusted Domains**, also available at any time from the Command Palette, brings up the **Trusted Domains** JSON file, where you can add, remove, or modify trusted domains.

```
// You can use the "Manage Trusted Domains" command to open this file.

// Save this file to apply the trusted domains rules.

[

  "*.twitter.com"

]
```

The **Trusted Domains** JSON file also has comments with examples of the supported domain formats and a list of the domains trusted by default, such as `https://*.visualstudio.com` and `https://*.microsoft.com`.

Next steps

Now that you know how the editor works, time to try a few other things...

- Intro Video - Code Editing - Watch an introductory video on code editing features.
- User Interface - In case you missed a basic orientation around VS Code.
- Key Bindings - Learn how to modify key bindings to your preference.
- Debugging - This is where VS Code really shines.

Common questions

How can I automatically select the second entry in Quick Open instead of the first?

With the command `workbench.action.quickOpenPreviousEditor`, you can have the second entry automatically selected in Quick Open. This can be useful if you want to select the previous entry from the list without having to invoke another keybinding:

```
[
  {
    "key": "ctrl+p",
    "command": "workbench.action.quickOpenPreviousEditor"
  },
  {
    "key": "ctrl+p",
    "command": "-workbench.action.quickOpen"
  }
]
```

How can I configure Ctrl+Tab to navigate across all editors of all groups

By default, `Ctrl+Tab` navigates between editors of the same editor group. If you want to navigate across all opened editors in all groups, you can create keyboard shortcuts for the `workbench.action.quickOpenPreviousRecentlyUsedEditor` and `workbench.action.quickOpenLeastRecentlyUsedEditor` commands:

```
[
  {
    "key": "ctrl+tab",
    "command": "workbench.action.quickOpenPreviousRecentlyUsedEditor",
    "when": "!inEditorsPicker"
  },
  {
    "key": "ctrl+shift+tab",
    "command": "workbench.action.quickOpenLeastRecentlyUsedEditor",
    "when": "!inEditorsPicker"
  }
]
```

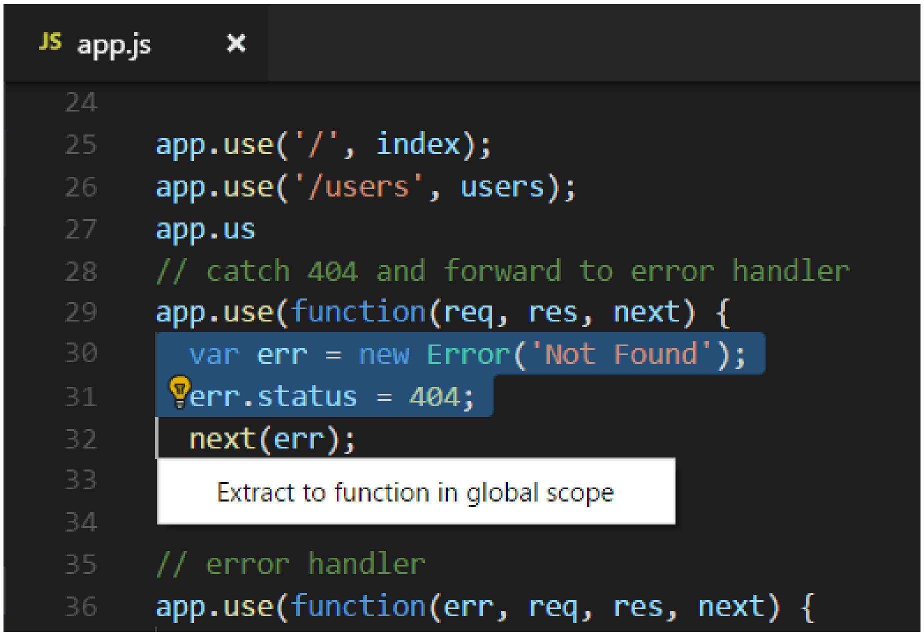
How can I navigate between recently used editors without a picker

Here is a list of commands you can use to navigate in editors without opening a picker:

Key	Command	Command ID
`unassigned`	Open Next Recently Used Editor	`workbench.action.openNextRecentlyUsedEditor`
`unassigned`	Open Previously Used Editor	`workbench.action.openPreviousRecentlyUsedEditor`
`unassigned`	Open Next Recently Used Editor in Group	`workbench.action.openNextRecentlyUsedEditorInGroup`
`unassigned`	Open Previously Used Editor in Group	`workbench.action.openPreviousRecentlyUsedEditorInGroup`

Refactoring

[Source code refactoring]() can improve the quality and maintainability of your project by restructuring your code while not modifying the runtime behavior. Visual Studio Code supports refactoring operations (refactorings) such as [Extract Method]() and [Extract Variable]() to improve your code base from within your editor.



```

app.use('/', index);
app.use('/users', users);
app.us
// catch 404 and forward to error handler
app.use(function(req, res, next) {
  var err = new Error('Not Found');
  err.status = 404;
  next(err);


// error handler
app.use(function(err, req, res, next) {
```

For example, a common refactoring used to avoid duplicating code (a maintenance headache) is the [Extract Method]() refactoring, where you select source code that you'd like to reuse elsewhere and pull it out into its own shared method.

Refactorings are provided by a language service and VS Code has built-in support for TypeScript and JavaScript refactoring through the [TypeScript]() language service. Refactoring support for other programming languages is provided through VS Code [extensions]() that contribute language services. The UI and commands for refactoring are the same across languages, and in this topic we'll demonstrate refactoring support with the TypeScript language service.

Code Actions = Quick Fixes and refactorings

In VS Code, Code Actions can provide both refactorings and Quick Fixes for detected issues (highlighted with green squiggles). An available Code Action is announced by a lightbulb near the source code when the cursor is on a squiggle or selected text region. Clicking on the Code Action lightbulb or using the **Quick Fix** command `Ctrl+.` will display Quick Fixes and refactorings.

If you'd just like to see refactorings without Quick Fixes, you can use the **Refactor** command (`Ctrl+Shift+R`).

Note: If you prefer to not see the Code Action lightbulb in your editor, you can disable lightbulbs with the `editor.lightbulb.enable` [setting](). You can still open Quick Fixes through **Quick Fix** command and `Ctrl+.` keyboard shortcut.

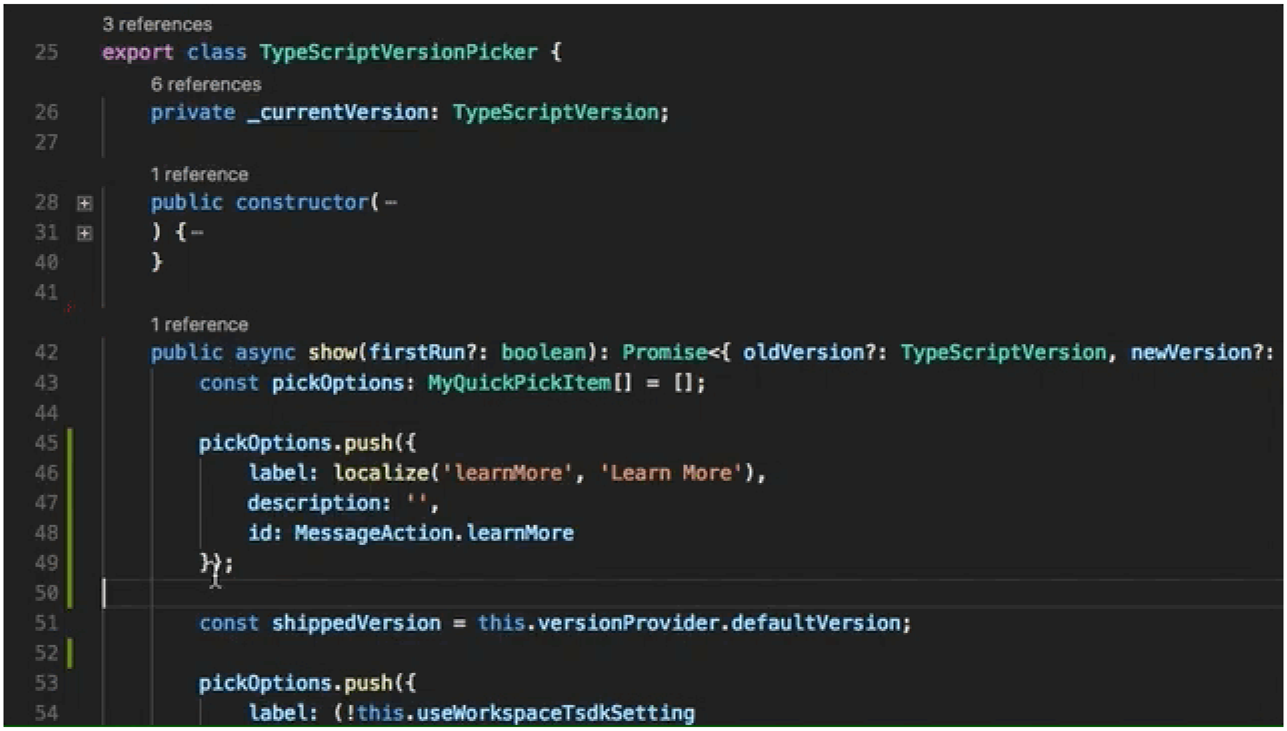
Refactoring actions

Extract Method

Select the source code you'd like to extract and then click on the lightbulb in the gutter or press (`Ctrl+.`) to see available refactorings. Source code fragments can be extracted into a new method, or into a new function at various different scopes. During the extract refactoring, you will be prompted to provide a meaningful name.

Extract Variable

TypeScript language service provides **Extract to const** refactoring to create a new local variable for the currently selected expression:

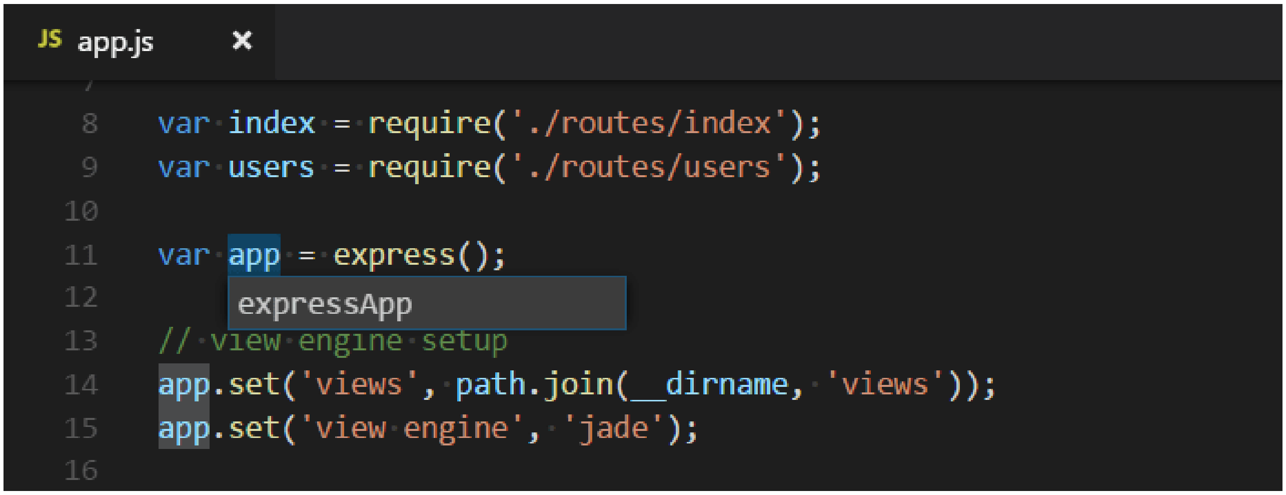


```
3 references
export class TypeScriptVersionPicker {
    6 references
    private _currentVersion: TypeScriptVersion;

    1 reference
    public constructor(⋯
    ) {⋯
    }

    1 reference
    public async show(firstRun?: boolean): Promise<{ oldVersion?: TypeScriptVersion, newVersion?:
        const pickOptions: MyQuickPickItem[] = [];

        pickOptions.push({
            label: localize('learnMore', 'Learn More'),
            description: '',
            id: MessageAction.learnMore
        });

        const shippedVersion = this.versionProvider.defaultVersion;

        pickOptions.push({
            label: (!this.useWorkspaceTsdkSetting
```

When working with classes, you can also extract a value to a new property.

Rename symbol

Renaming is a common operation related to refactoring source code and VS Code has a separate **Rename Symbol** command (`F2`). Some languages support rename symbol across files. Press `F2` and then type the new desired name and press `Enter`. All usages of the symbol will be renamed, across files.



```
var index = require('./routes/index');
var users = require('./routes/users');

var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'jade');
```

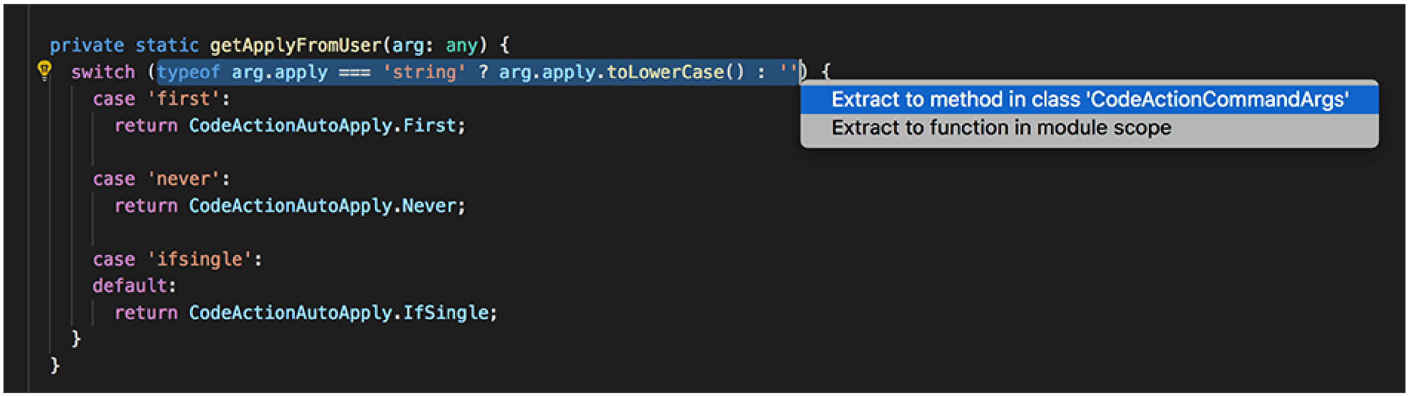
Keybindings for Code Actions

The `editor.action.codeAction` command lets you configure keybindings for specific Code Actions. This keybinding, for example, triggers the **Extract function** refactoring Code Actions:

```
{
  "key": "ctrl+shift+r ctrl+e",
  "command": "editor.action.codeAction",
  "args": {
    "kind": "refactor.extract.function"
  }
}
```

Code Action kinds are specified by extensions using the enhanced `CodeActionProvided` API. Kinds are hierarchical, so `"kind": "refactor"` will show all refactoring Code Actions, whereas `"kind": "refactor.extract.function"` will only show **Extract function** refactorings.

Using the above keybinding, if only a single `"refactor.extract.function"` Code Action is available, it will be automatically applied. If multiple **Extract function** Code Actions are available, we bring up a context menu to select them:


Extract to method in class 'CodeActionCommandArgs'
Extract to function in module scope

```
private static getApplyFromUser(arg: any) {
  switch (typeof arg.apply === 'string' ? arg.apply.toLowerCase() : '') {
    case 'first':
      return CodeActionAutoApply.First;

    case 'never':
      return CodeActionAutoApply.Never;

    case 'ifsingle':
    default:
      return CodeActionAutoApply.IfSingle;
  }
}
```

You can also control how/when Code Actions are automatically applied using the `apply` argument:

```
{

  "key": "ctrl+shift+r ctrl+e",

  "command": "editor.action.codeAction",

  "args": {

    "kind": "refactor.extract.function",

    "apply": "first"

  }

}
```

Valid values for `"apply"`:

- `"first"` - Always automatically apply the first available Code Action.
- `"ifSingle"` - Default. Automatically apply the Code Action if only one is available. Otherwise, show the context menu.
- `"never"` - Always show the Code Action context menu, even if only a single Code Action is available.

When a Code Action keybinding is configured with `"preferred": true`, only preferred Quick Fixes and refactorings are shown. A preferred Quick Fix addresses the underlying error, while a preferred refactoring is the most common refactoring choice. For example, while multiple `refactor.extract.constant` refactorings may exist, each extracting to a different scope in the file, the preferred `refactor.extract.constant` refactoring is the one that extracts to a local variable.

This keybinding uses `"preferred": true` to create a refactoring that always tries to extract the selected source code to a constant in the local scope:

```
{

  "key": "shift+ctrl+e",

  "command": "editor.action.codeAction",

  "args": {

    "kind": "refactor.extract.constant",

    "preferred": true,

    "apply": "ifSingle"

  }

}
```

Extensions with refactorings

You can find extensions that support refactoring by looking in the VS Code Marketplace. You can go to the Extensions view (`Ctrl+Shift+X`) and type 'refactor' in the search box. You can then sort by install count or ratings to see which extensions are popular.

Python
90.7M
ms-python

Language Support for Java(TM) by Red Hat
25.8M
redhat

GitHub Copilot
7.3M
GitHub

PHP IntelliSense
5.8M
felixfbecker

Tip: The extensions shown above are dynamically queried. Click on an extension tile above to read the description and reviews to decide which extension is best for you.

Next steps

- Intro Video - Code Editing - Watch an introductory video on code editing features.
- Code Navigation - VS Code lets you move quickly through your source code.
- Debugging - Learn about debugging with VS Code.

Common questions

Why don't I see any lightbulbs when there are errors in my code?

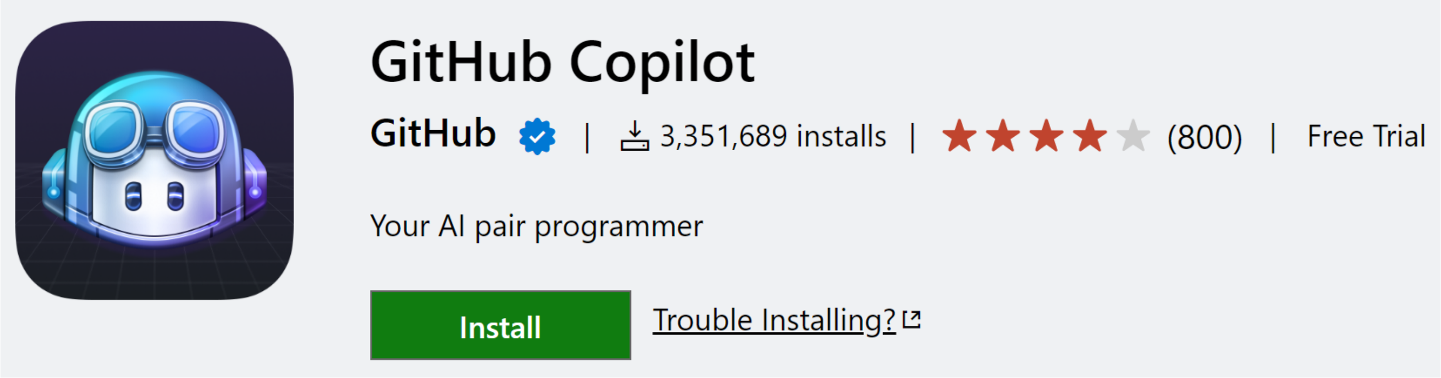
Lightbulbs (Code Actions) are only shown when your cursor hovers over the text showing the error. Hovering over the text will show the error description, but you need to move the cursor or select text to see lightbulbs for Quick Fixes and refactorings.

AI Tools in VS Code

The GitHub Copilot extension is an AI pair programmer tool that helps you write code faster and smarter. You can use the Copilot extension in VS Code to generate code, learn from the code it generates, and even configure your editor.

Prerequisites

You'll use the GitHub Copilot extension to power your AI suggestions in VS Code.



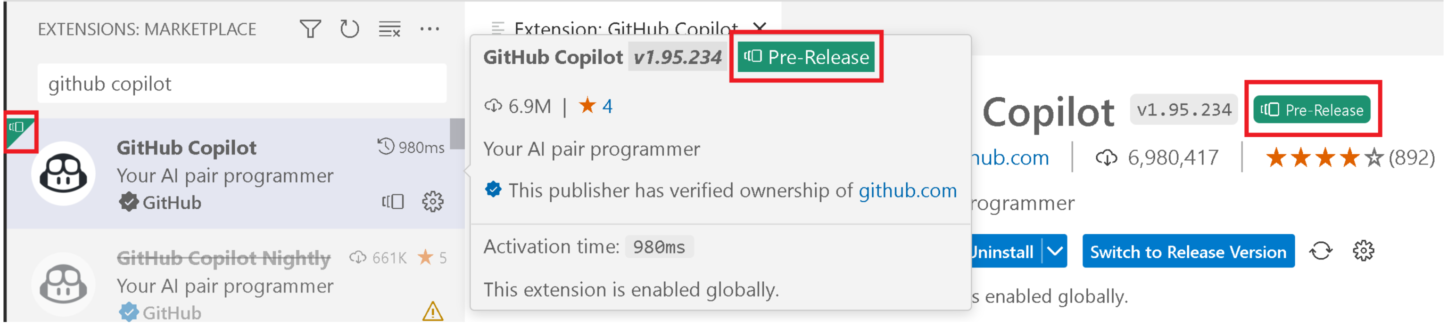
GitHub Copilot

GitHub | 3,351,689 installs | ★★★★☆ (800) | Free Trial

Your AI pair programmer

To use GitHub Copilot, you need an active GitHub Copilot subscription. In the content below, you'll learn how VS Code will help you activate your free trial directly from VS Code. You can also activate your trial starting from the GitHub Copilot signup page.

Note: For some of the latest features we'll explore below, you'll need to use the pre-release version of the GitHub Copilot extension, which will provide you the latest updates in Copilot.



EXTENSIONS: MARKETPLACE

github copilot

GitHub Copilot — Your AI pair programmer — GitHub — 980ms

~~GitHub Copilot Nightly~~ — 661K ★ 5 — Your AI pair programmer — GitHub

GitHub Copilot v1.95.234 Pre-Release

6.9M | ★ 4

Your AI pair programmer

This publisher has verified ownership of github.com

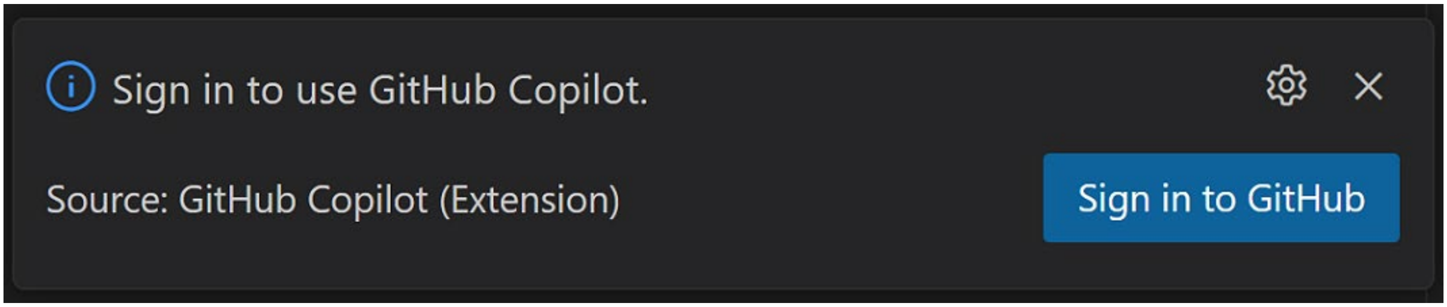
Activation time: 980ms

This extension is enabled globally.

Extension: GitHub Copilot
Copilot v1.95.234 Pre-Release
hub.com
6,980,417
★★★★☆ (892)
rogrammer
Uninstall
Switch to Release Version
s enabled globally.

Sign in and sign up

If you have not previously authorized VS Code in your GitHub account, you will be prompted to sign in to GitHub in VS Code:

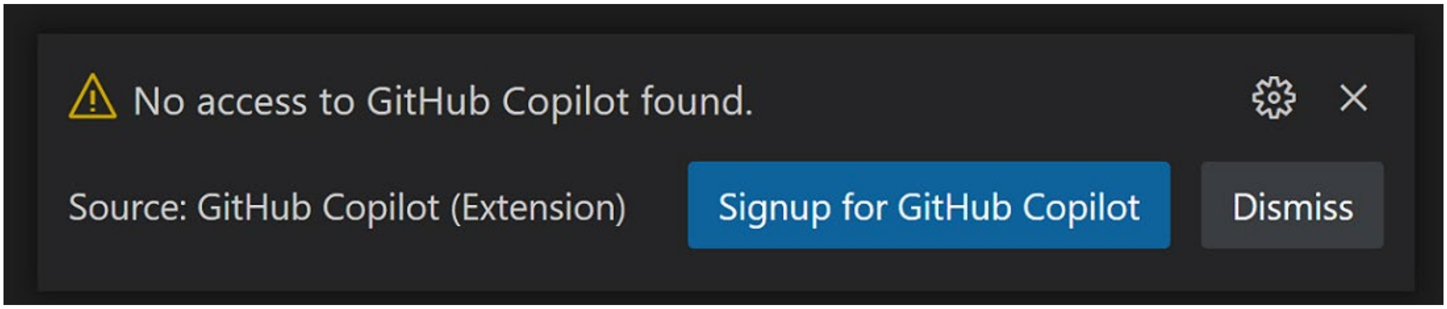

Sign in to use GitHub Copilot.

Source: GitHub Copilot (Extension)

In your browser, GitHub will request the necessary permissions for GitHub Copilot. To approve these permissions, select **Authorize Visual Studio Code**.

Activate your free trial

If you haven't yet activated your free trial for Copilot, the extension will notify you in VS Code. Select **Signup for GitHub Copilot** to activate your trial.


No access to GitHub Copilot found.

Source: GitHub Copilot (Extension)

You can learn more about billing for Copilot in the GitHub Copilot documentation.

Using Copilot

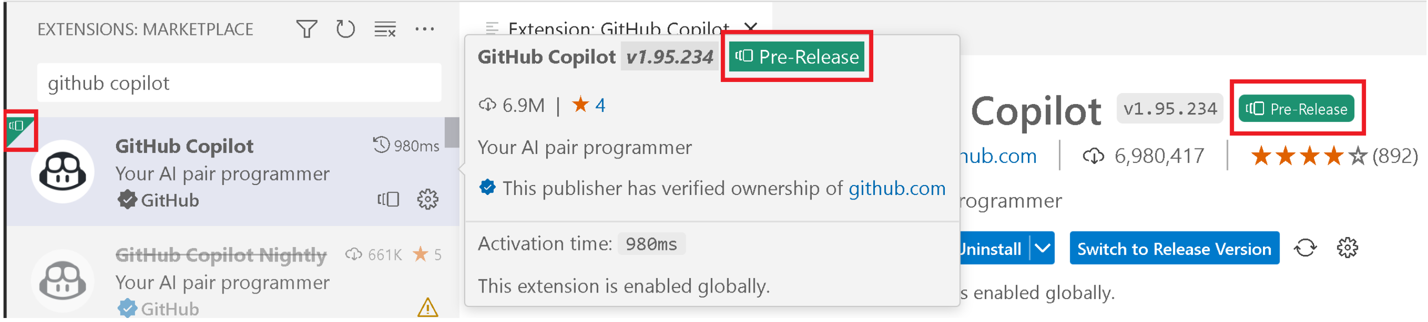
Now that you've signed up for Copilot and activated the extension, let's see its assistance in action!

GitHub Copilot provides suggestions for numerous languages and a wide variety of frameworks, and it works especially well for Python, JavaScript, TypeScript, Ruby, Go, C# and C++.

There are three main ways to get assistance from Copilot:

- **Inline suggestions:** Harness Copilot's help automatically through suggestions it provides directly inline as you work in your code.
- **Chat view:** Ask Copilot for help with any task or question in the GitHub Copilot Chat view.
- **Inline chat:** Talk with Copilot while writing code, inline in your files.

Note: To get access to the chat view and inline chat, you'll need to sign up for the GitHub Copilot chat waitlist. You'll also need to use pre-release version of the GitHub Copilot extension.



Inline suggestions

Copilot presents suggestions automatically to help you code more efficiently. There are just 3 steps to harnessing these suggestions:

1. Start writing code (or code-related items, like comments or tests).

Copilot provides suggestions for a variety of languages and frameworks. For any given input, Copilot may offer multiple suggestions. You can select which suggestion to use or reject all suggestions.

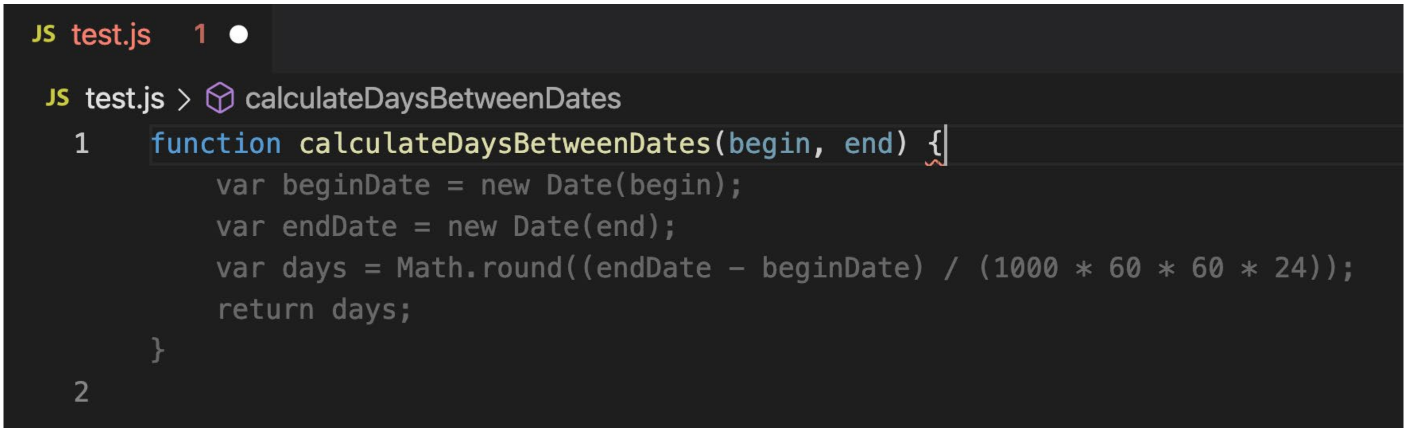
2. Receive a Copilot suggestion in gray ghost (faded) text.

Ghost text is placeholder text that will be replaced by input you type or select from Copilot.

As an example, a JavaScript file, you can type the following function header:

```
function calculateDaysBetweenDates(begin, end) {
```

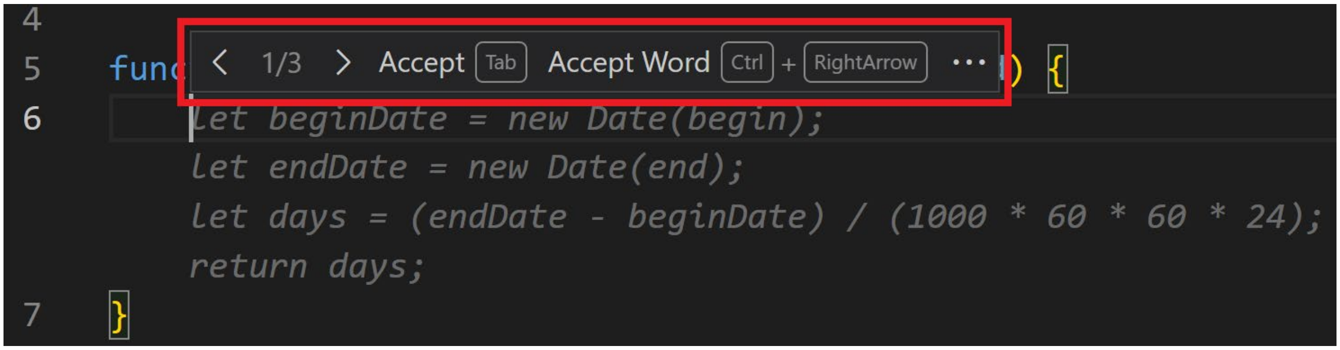
Copilot will provide a suggestion like the following:



```
function calculateDaysBetweenDates(begin, end) {
    var beginDate = new Date(begin);
    var endDate = new Date(end);
    var days = Math.round((endDate - beginDate) / (1000 * 60 * 60 * 24));
    return days;
}
```

3. Choose to accept Copilot's suggestion.

For any given input, Copilot may offer multiple suggestions. When Copilot offers a suggestion, you can use accept it with the `Tab` key, or hover over the suggestion to see the inline suggestion toolbar:



```
func                                                            ) {
    let beginDate = new Date(begin);
    let endDate = new Date(end);
    let days = (endDate - beginDate) / (1000 * 60 * 60 * 24);
    return days;
}
```

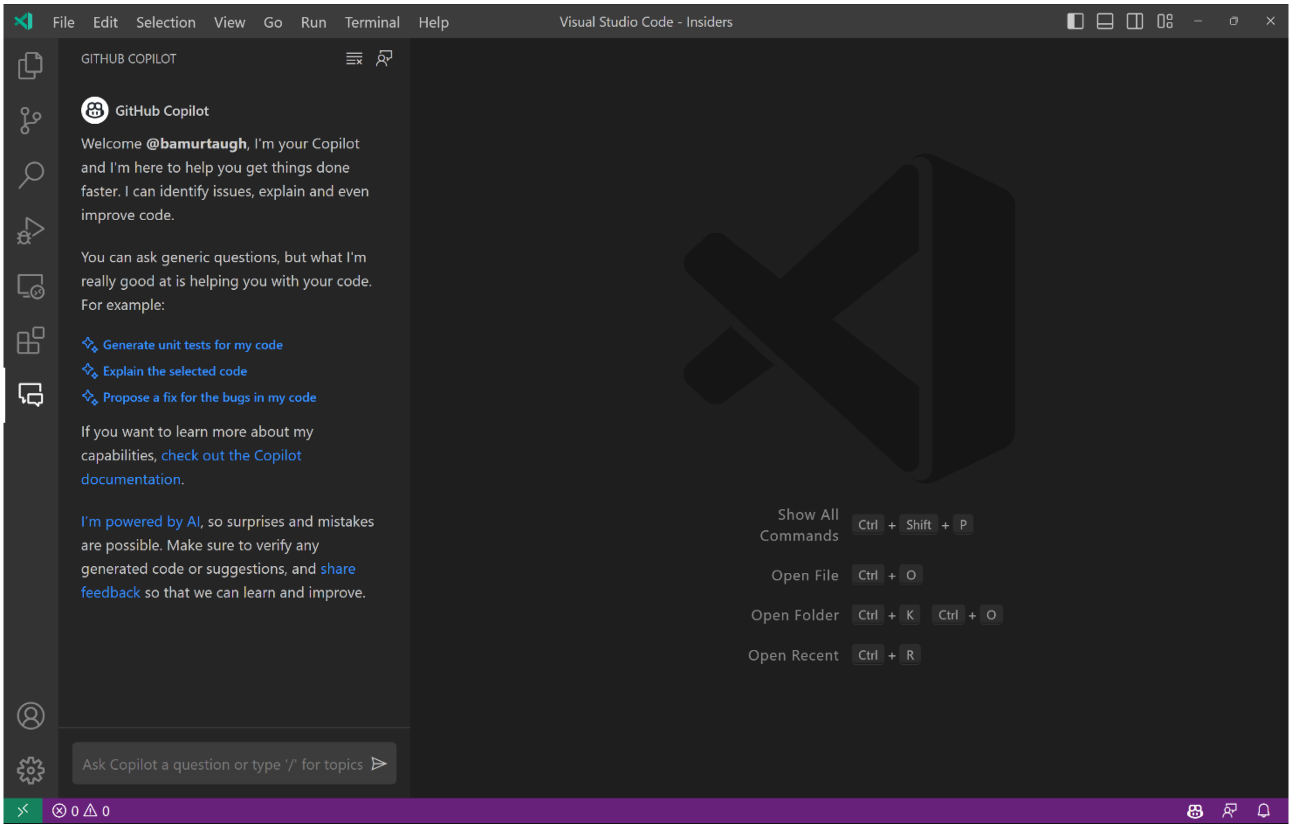
In the image above, Copilot presents three suggestions. You can accept the entire suggestion with `Tab`, or only part of the suggestion with `Ctrl+RightArrow`. You can switch between suggestions in the suggestion toolbar, or use the keyboard shortcut `Alt+]` instead.

If you don't want to accept any of the suggestions, you can continue typing, and Copilot will continue providing suggestions as you work.

Chat view

When developing a project or learning something new, it can be a big help to get AI assistance on your questions, big or small. Copilot enables an interactive Chat experience that understands the context of your code, workspace, extensions, settings, and more.

Once you've signed up and been granted access to Copilot chat through the chat waitlist, install the pre-release version of the GitHub Copilot extension in VS Code. You'll be presented a new GitHub Copilot chat view in the Activity Bar:



GitHub Copilot

Welcome **@bamurtaugh**, I'm your Copilot and I'm here to help you get things done faster. I can identify issues, explain and even improve code.

You can ask generic questions, but what I'm really good at is helping you with your code. For example:

Generate unit tests for my code

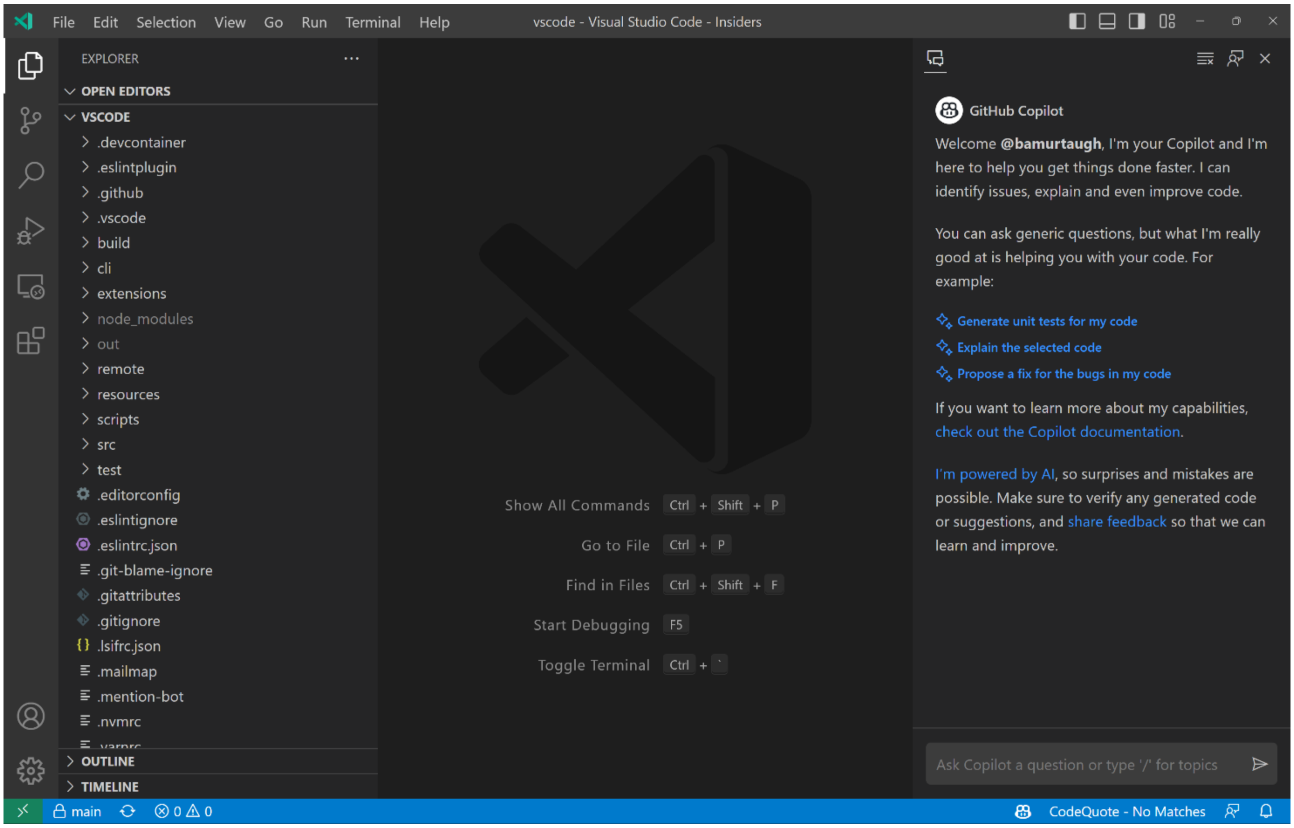
Explain the selected code

Propose a fix for the bugs in my code

If you want to learn more about my capabilities, check out the Copilot documentation.

I'm powered by AI, so surprises and mistakes are possible. Make sure to verify any generated code or suggestions, and share feedback so that we can learn and improve.

Like other views in VS Code, you can move it anywhere. For example, you can move it to the secondary sidebar so that you can use other views like the Explorer at the same time:



GitHub Copilot

Welcome **@bamurtaugh**, I'm your Copilot and I'm here to help you get things done faster. I can identify issues, explain and even improve code.

You can ask generic questions, but what I'm really good at is helping you with your code. For example:

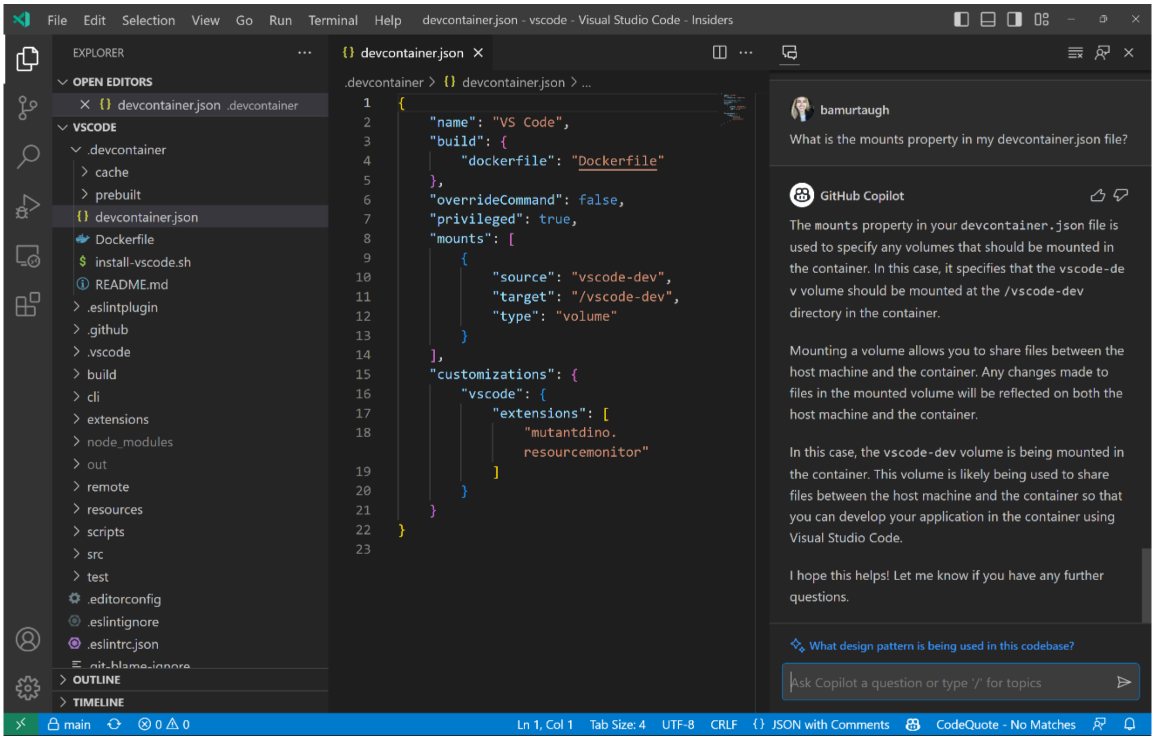
- Generate unit tests for my code
- Explain the selected code
- Propose a fix for the bugs in my code

If you want to learn more about my capabilities, check out the Copilot documentation.

I'm powered by AI, so surprises and mistakes are possible. Make sure to verify any generated code or suggestions, and share feedback so that we can learn and improve.

Ask Copilot a question or type '/' for topics

Copilot will suggest potential questions to get started. You can select any of these questions or use the chat box to type your own:



```json
{
    "name": "VS Code",
    "build": {
        "dockerfile": "Dockerfile"
    },
    "overrideCommand": false,
    "privileged": true,
    "mounts": [
        {
            "source": "vscode-dev",
            "target": "/vscode-dev",
            "type": "volume"
        }
    ],
    "customizations": {
        "vscode": {
            "extensions": [
                "mutantdino.resourcemonitor"
            ]
        }
    }
}
```

bamurtaugh

What is the mounts property in my devcontainer.json file?

GitHub Copilot

The `mounts` property in your `devcontainer.json` file is used to specify any volumes that should be mounted in the container. In this case, it specifies that the `vscode-dev` volume should be mounted at the `/vscode-dev` directory in the container.

Mounting a volume allows you to share files between the host machine and the container. Any changes made to files in the mounted volume will be reflected on both the host machine and the container.

In this case, the `vscode-dev` volume is being mounted in the container. This volume is likely being used to share files between the host machine and the container so that you can develop your application in the container using Visual Studio Code.

I hope this helps! Let me know if you have any further questions.

What design pattern is being used in this codebase?

Ask Copilot a question or type '/' for topics

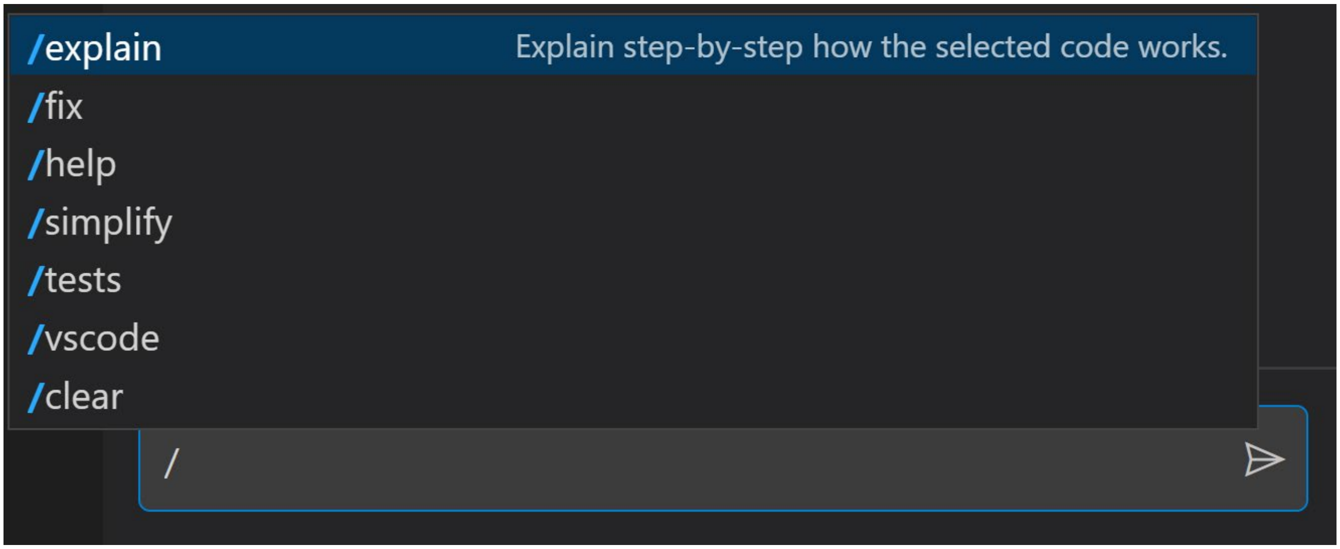
As you continue asking questions, Copilot maintains the history of your conversation, and it'll provide related follow-up questions or commands in its response too.

You can help Copilot provide better answers by upvoting or downvoting responses with the thumbs up and down icons in the upper right of its response. This provides Copilot feedback on how much it helped with your scenario so that it can help you even better in the future.

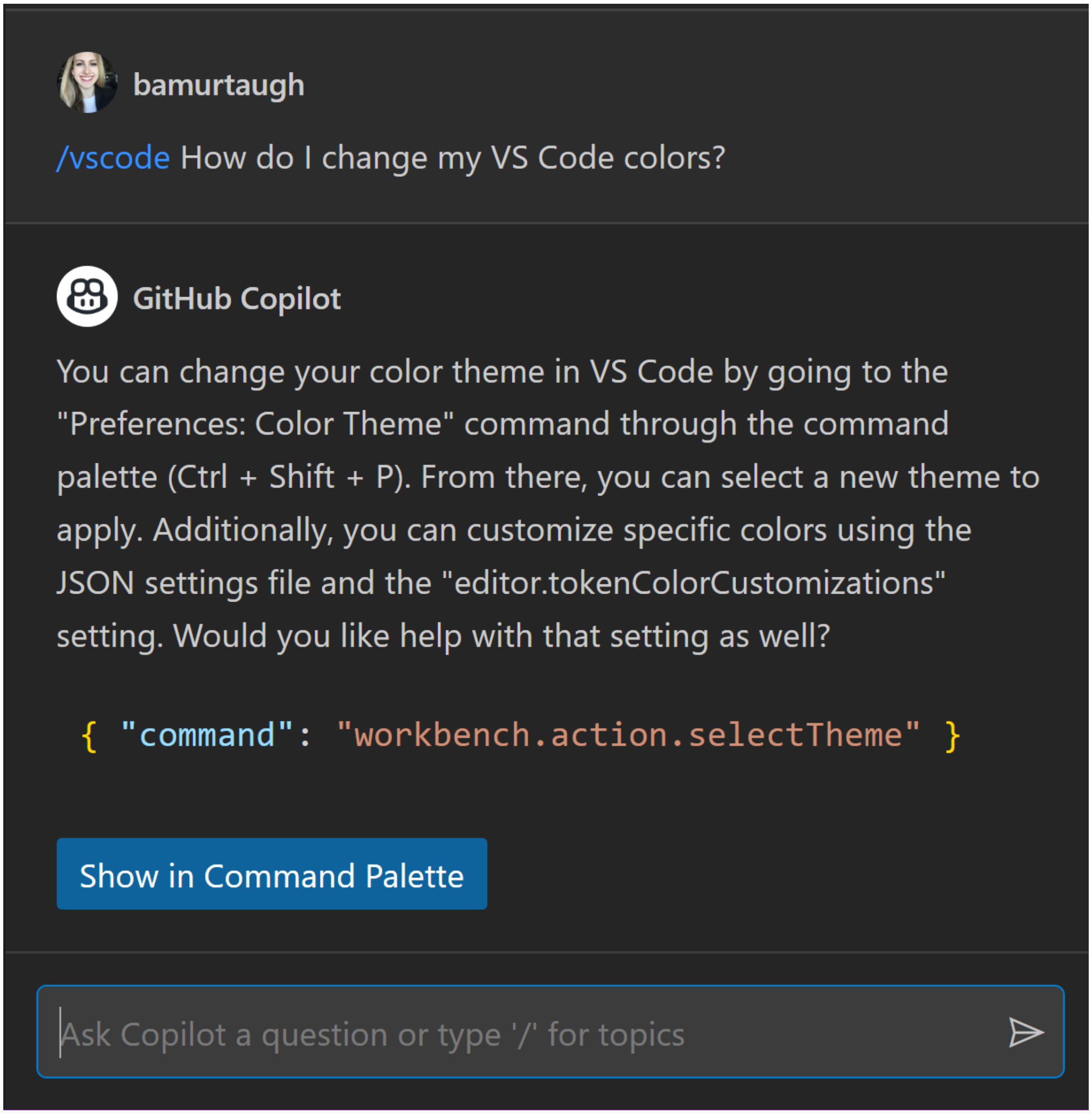
Slash commands

To further help Copilot give you more relevant answers, you can choose a topic for your questions through "slash commands."

You can prepend your chat inputs with a specific topic name to help Copilot give you a more relevant response. When you start typing `/`, you'll see the list of possible topics:



- /explain: Explain step-by-step how the selected code works.
- /fix: Propose a fix for the bugs in the selected code.
- /help: Prints general help about GitHub Copilot.
- /tests: Generate unit tests for the selected code.
- /vscode: Questions about VS Code commands and settings.
- /clear: Clear the session.



bamurtaugh

/vscode How do I change my VS Code colors?

GitHub Copilot

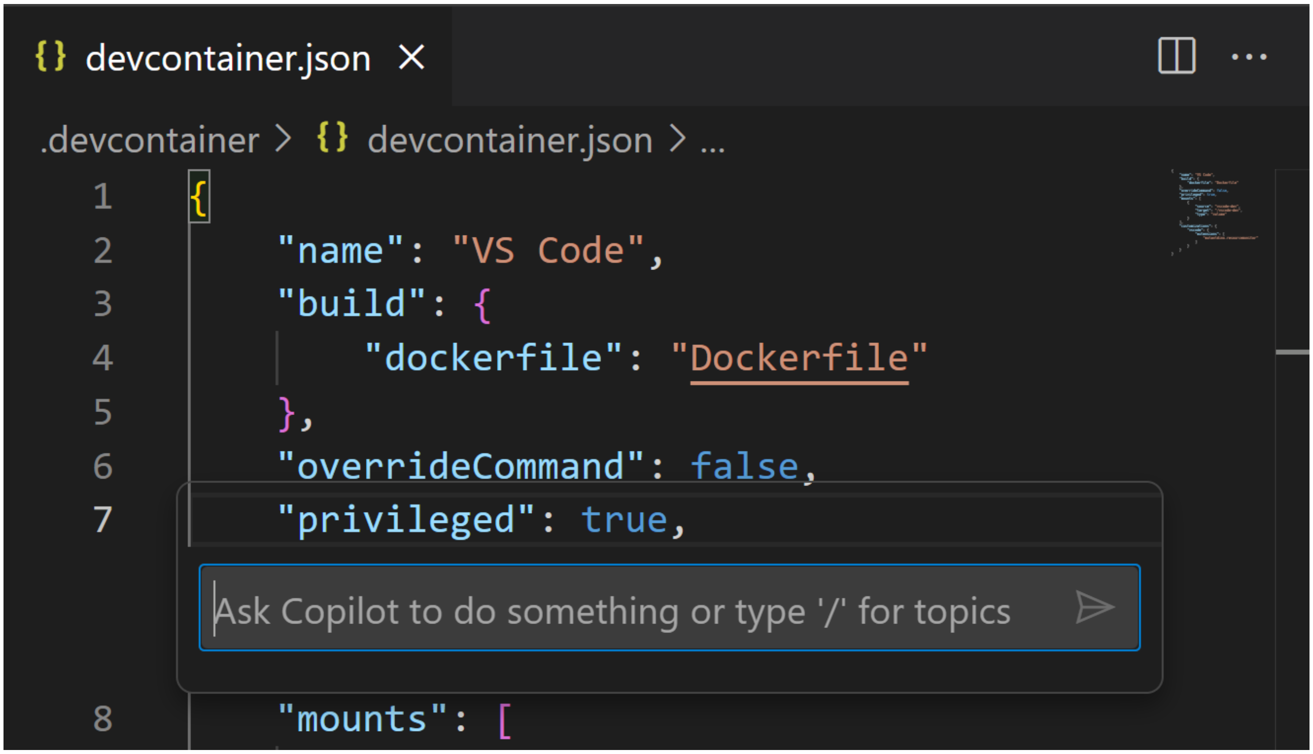
You can change your color theme in VS Code by going to the "Preferences: Color Theme" command through the command palette (Ctrl + Shift + P). From there, you can select a new theme to apply. Additionally, you can customize specific colors using the JSON settings file and the "editor.tokenColorCustomizations" setting. Would you like help with that setting as well?

```
{ "command": "workbench.action.selectTheme" }
```

Inline chat

An additional key functionality of Copilot is answering questions inline as you're coding. This allows you to harness the power of AI while staying in your existing editor workflow.

In any file, you can press `Ctrl+I` on your keyboard to bring up Copilot inline chat:


{} devcontainer.json
.devcontainer > {} devcontainer.json > ...
Ask Copilot to do something or type '/' for topics

```
{
    "name": "VS Code",
    "build": {
        "dockerfile": "Dockerfile"
    },
    "overrideCommand": false,
    "privileged": true,
    "mounts": [
```

You can ask Copilot questions that emerge as you write and iterate on code, such as "Explain this piece of code" or "How do I add functionality to do X?" Several slash commands also work in inline chat.

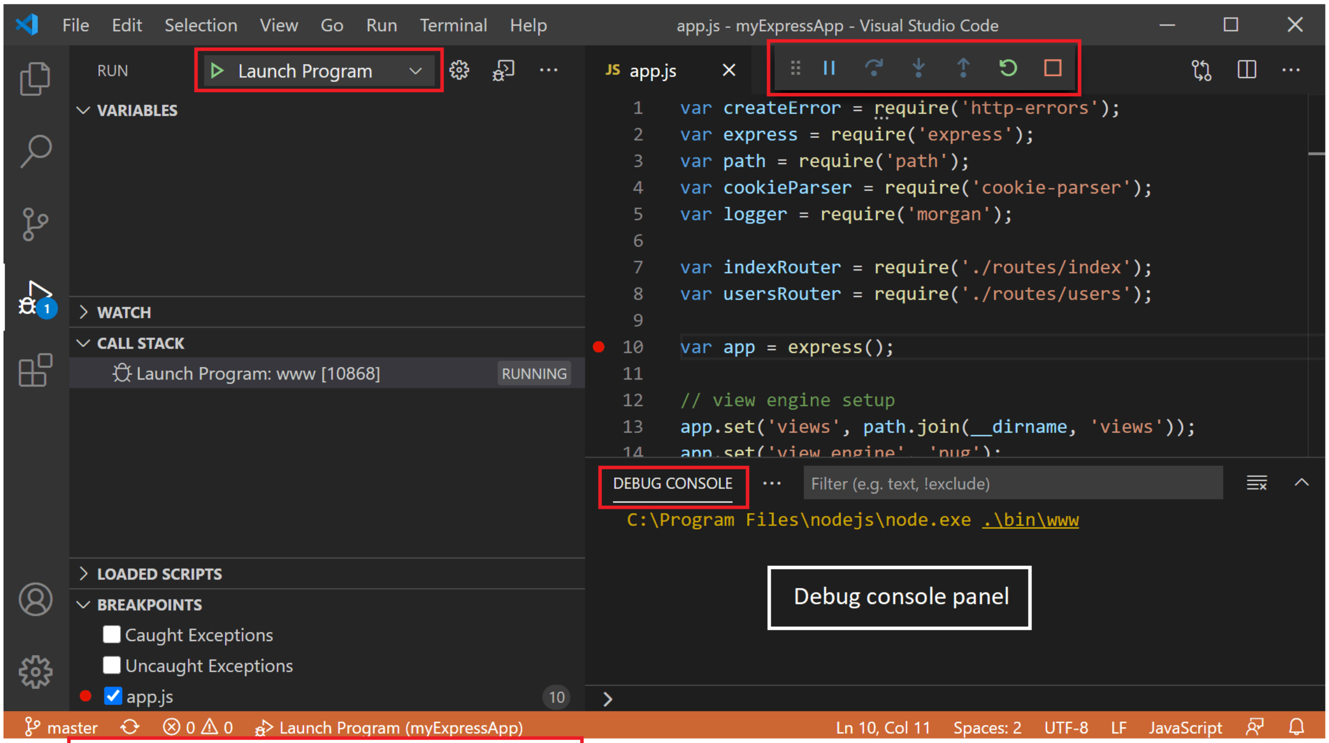
Additional resources

Congratulations, you've now used artificial intelligence to enhance your coding!

You can read more about Copilot and how to use it in VS Code in the GitHub Copilot documentation.

Debugging

One of the key features of Visual Studio Code is its great debugging support. VS Code's built-in debugger helps accelerate your edit, compile, and debug loop.



Start debugging

Pause, step over, step in/out, restart, stop

Debug console panel

Debug side bar

```javascript
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');

var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');

var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');
```

```
C:\Program Files\nodejs\node.exe .\bin\www
```

Debugger extensions

VS Code has built-in debugging support for the Node.js runtime and can debug JavaScript, TypeScript, or any other language that gets transpiled to JavaScript.

For debugging other languages and runtimes (including PHP, Ruby, Go, C#, Python, C++, PowerShell and many others), look for `Debuggers` extensions in the VS Code Marketplace or select **Install Additional Debuggers** in the top-level Run menu.

Below are several popular extensions which include debugging support:

Python
90.7M
ms-python

C/C++
49.1M
ms-vscode

Debugger for Java
23.3M
vscjava

C#
22.5M
ms-dotnettools

Tip: The extensions shown above are dynamically queried. Select an extension tile above to read the description and reviews to decide which extension is best for you.

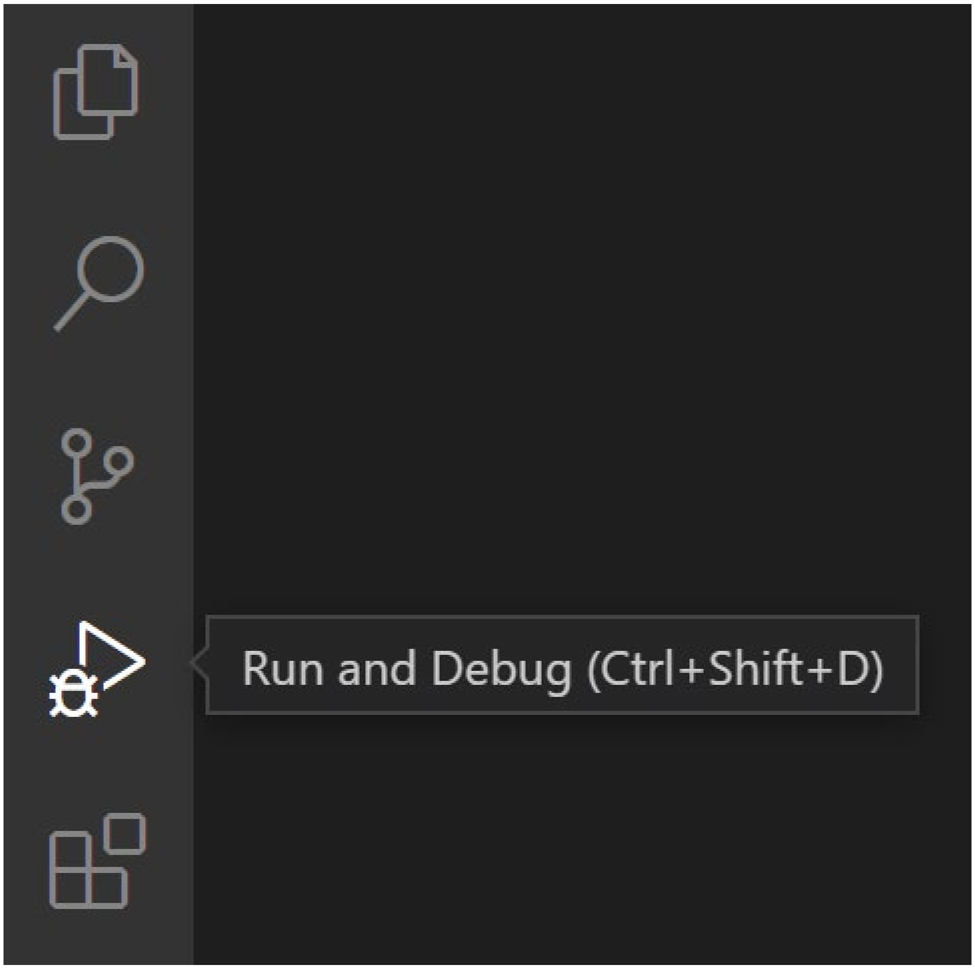
Start debugging

The following documentation is based on the built-in Node.js debugger, but most of the concepts and features are applicable to other debuggers as well.

It is helpful to first create a sample Node.js application before reading about debugging. You can follow the Node.js walkthrough to install Node.js and create a simple "Hello World" JavaScript application (`app.js`). Once you have a simple application set up, this page will take you through VS Code debugging features.

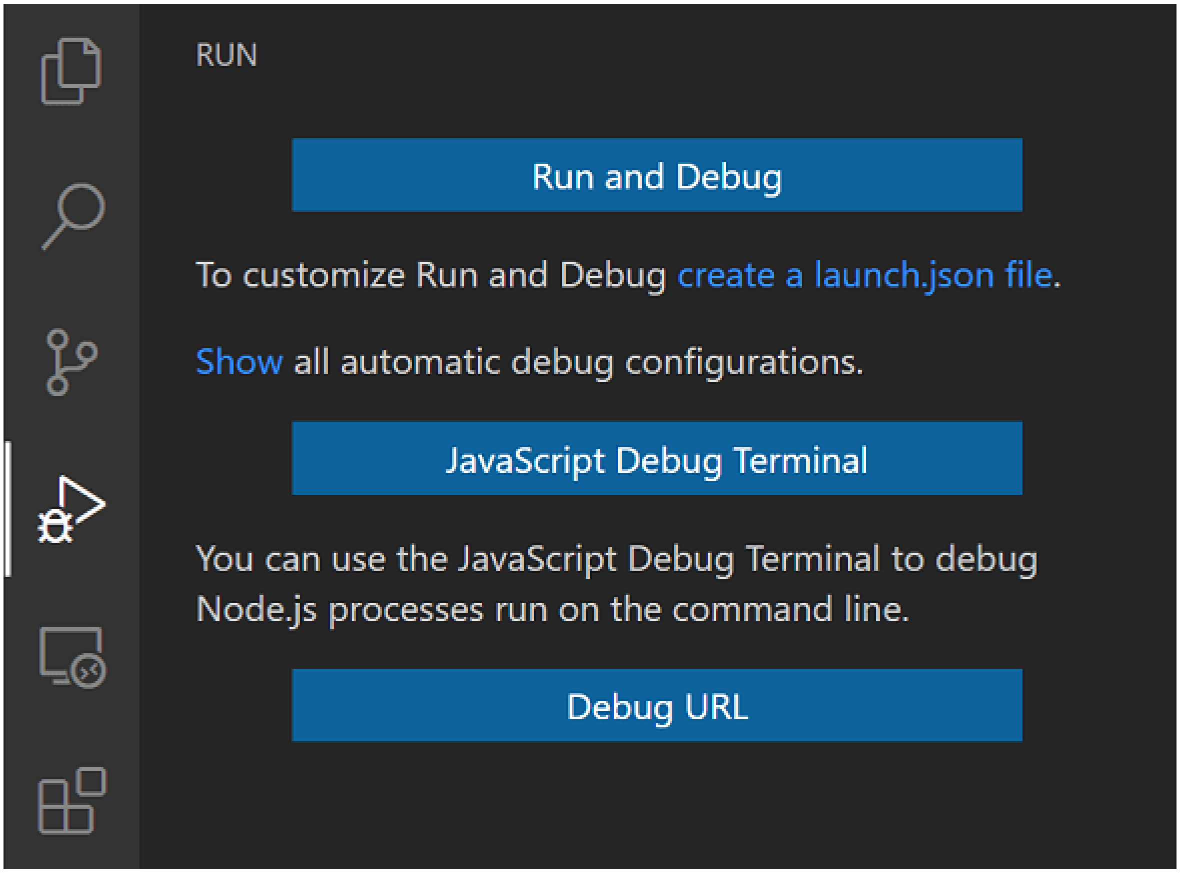
Run and Debug view

To bring up the **Run and Debug** view, select the **Run and Debug** icon in the **Activity Bar** on the side of VS Code. You can also use the keyboard shortcut `Ctrl+Shift+D`.



The **Run and Debug** view displays all information related to running and debugging and has a top bar with debugging commands and configuration settings.

If running and debugging is not yet configured (no `launch.json` has been created), VS Code shows the Run start view.



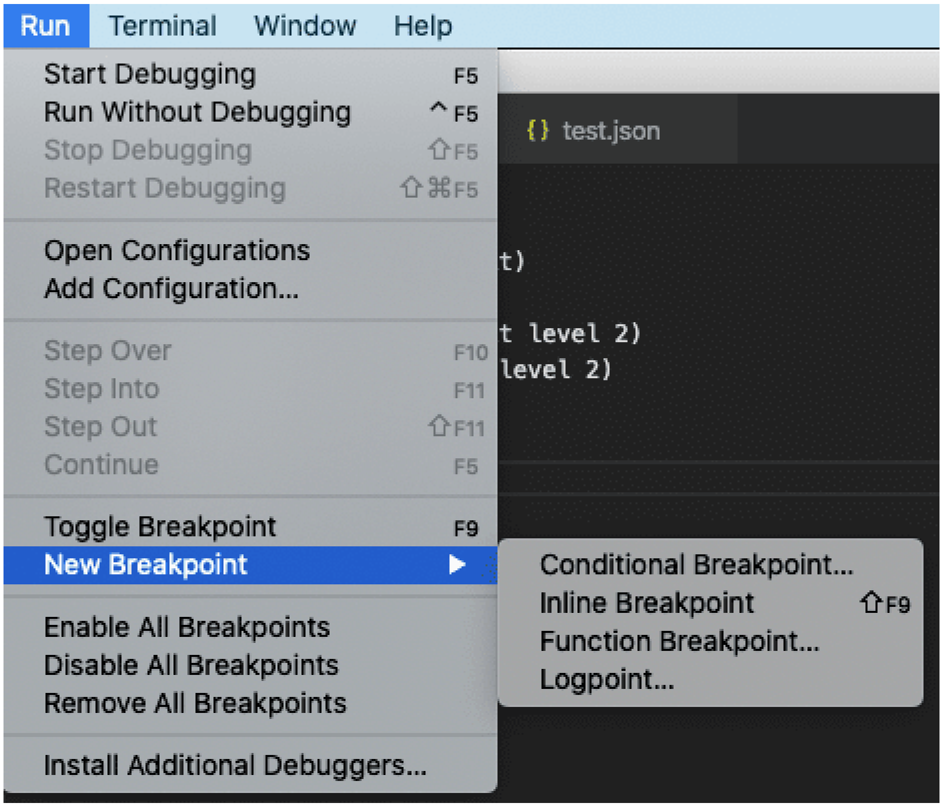
To customize Run and Debug create a launch.json file.

Show all automatic debug configurations.

You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.

Run menu

The top-level **Run** menu has the most common run and debug commands:

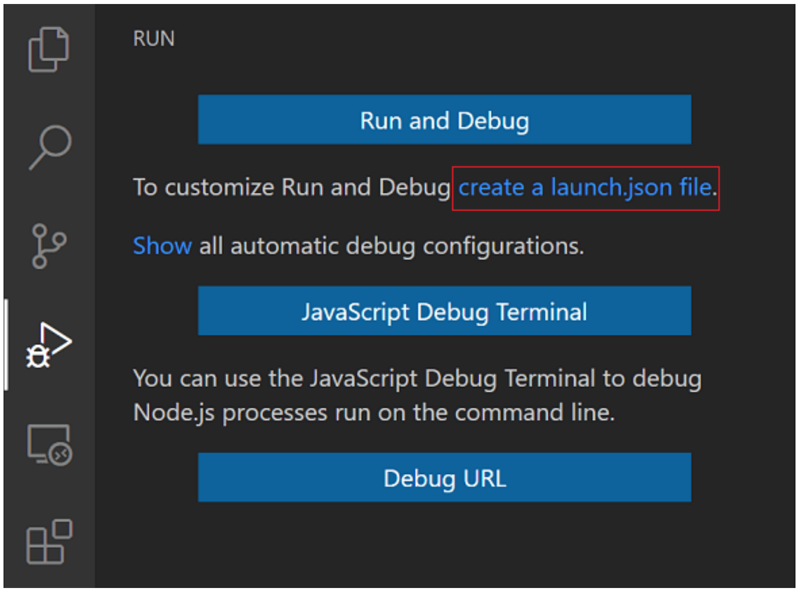


Launch configurations

To run or debug a simple app in VS Code, select **Run and Debug** on the Debug start view or press `F5` and VS Code will try to run your currently active file.

However, for most debugging scenarios, creating a launch configuration file is beneficial because it allows you to configure and save debugging setup details. VS Code keeps debugging configuration information in a `launch.json` file located in a `.vscode` folder in your workspace (project root folder) or in your user settings or workspace settings.

To create a `launch.json` file, click the **create a launch.json file** link in the Run start view.

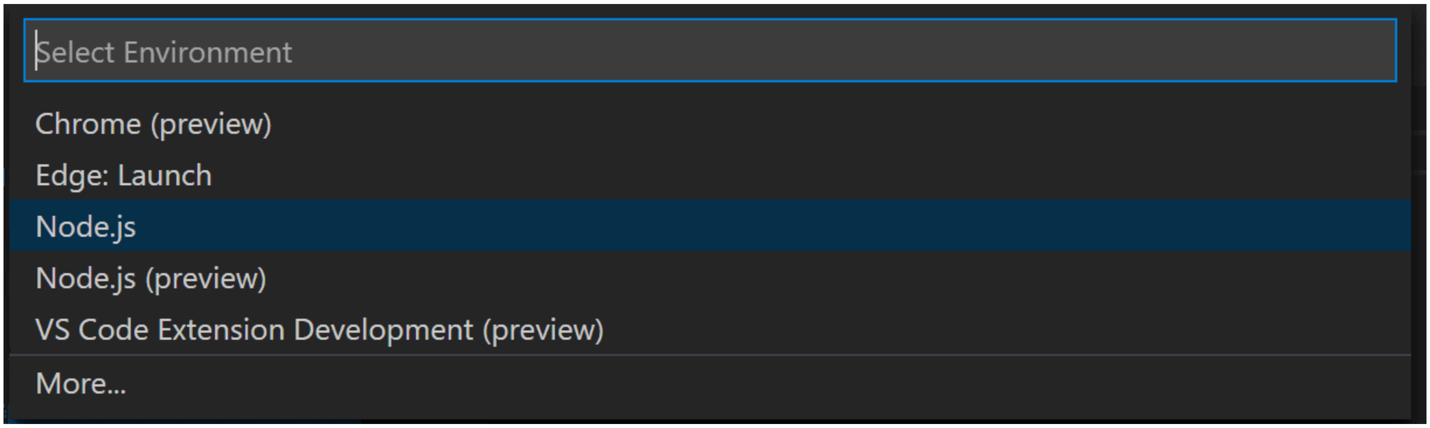


To customize Run and Debug create a launch.json file.

Show all automatic debug configurations.

You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.

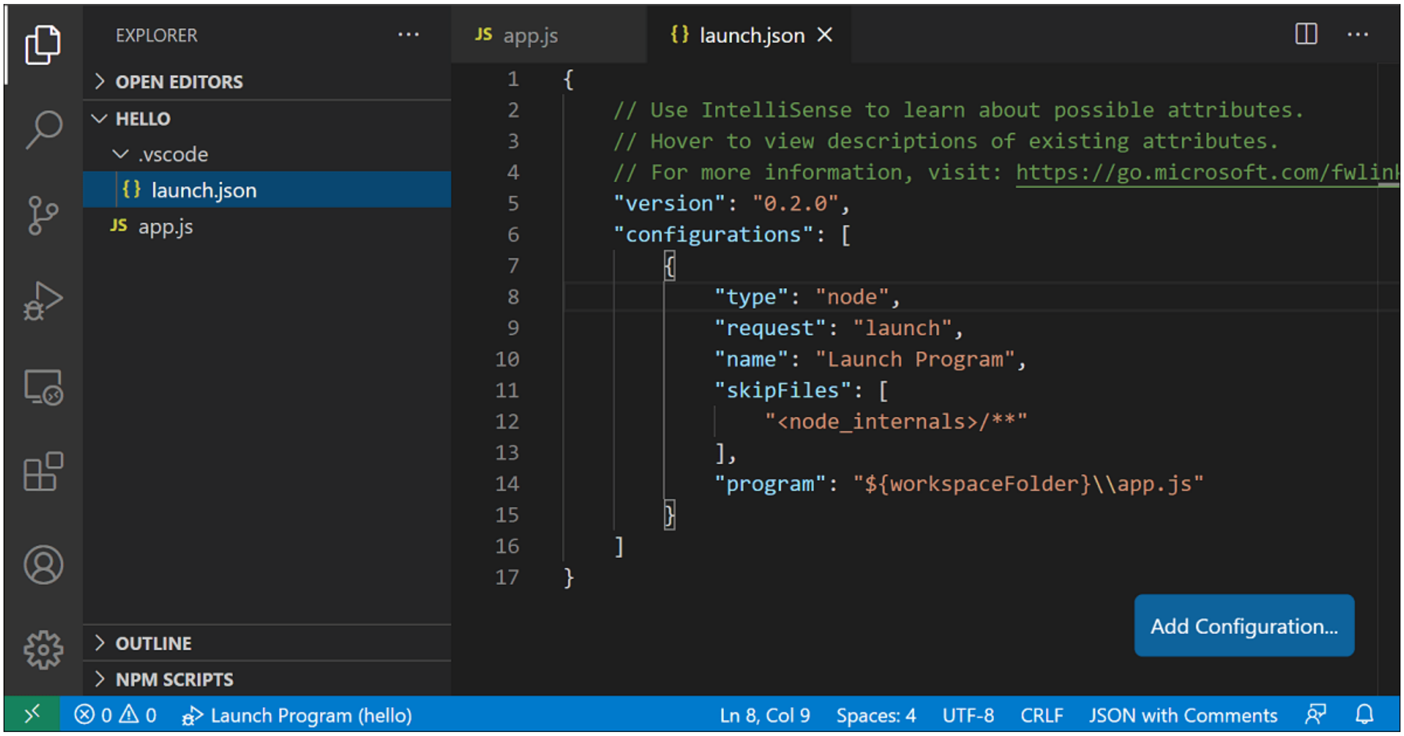
VS Code will try to automatically detect your debug environment, but if this fails, you will have to choose it manually:



Here is the launch configuration generated for Node.js debugging:

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "node",
      "request": "launch",
      "name": "Launch Program",
      "skipFiles": ["<node_internals>/**"],
      "program": "${workspaceFolder}\\app.js"
    }
  ]
}
```

If you go back to the File Explorer view (`Ctrl+Shift+E`), you'll see that VS Code has created a `.vscode` folder and added the `launch.json` file to your workspace.

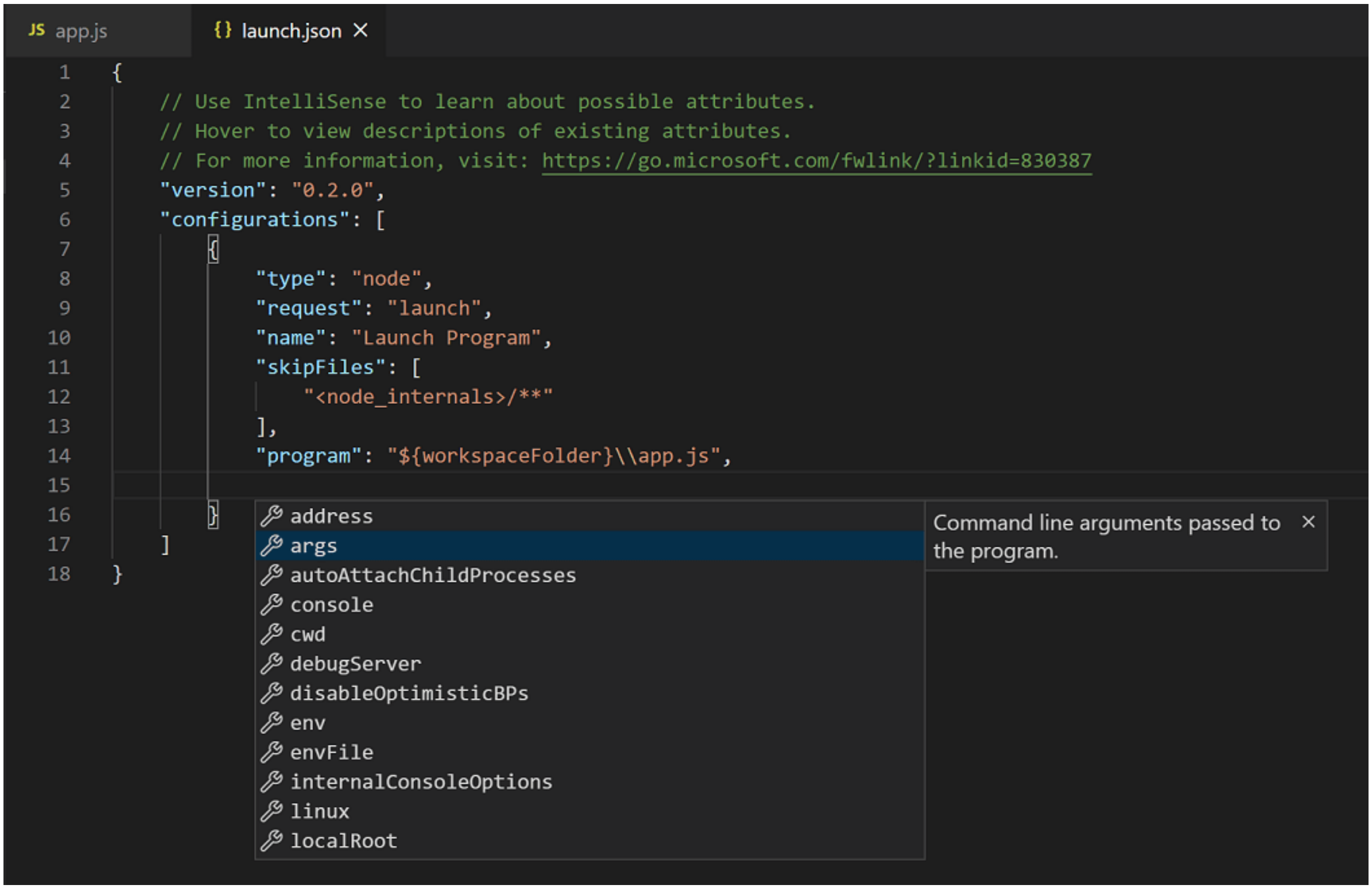


```
{
    // Use IntelliSense to learn about possible attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit: https://go.microsoft.com/fwlin
    "version": "0.2.0",
    "configurations": [
        {
            "type": "node",
            "request": "launch",
            "name": "Launch Program",
            "skipFiles": [
                "<node_internals>/**"
            ],
            "program": "${workspaceFolder}\\app.js"
        }
    ]
}
```

Note: You can debug a simple application even if you don't have a folder open in VS Code, but it is not possible to manage launch configurations and set up advanced debugging. The VS Code Status Bar is purple if you do not have a folder open.

Note that the attributes available in launch configurations vary from debugger to debugger. You can use IntelliSense suggestions (`Ctrl+Space`) to find out which attributes exist for a specific debugger. Hover help is also available for all attributes.

Do not assume that an attribute that is available for one debugger automatically works for other debuggers too. If you see green squiggles in your launch configuration, hover over them to learn what the problem is and try to fix them before launching a debug session.



```
{
    // Use IntelliSense to learn about possible attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
    "version": "0.2.0",
    "configurations": [
        {
            "type": "node",
            "request": "launch",
            "name": "Launch Program",
            "skipFiles": [
                "<node_internals>/**"
            ],
            "program": "${workspaceFolder}\\app.js",

        }
    ]
}
```

Command line arguments passed to the program.

Review all automatically generated values and make sure that they make sense for your project and debugging environment.

Launch versus attach configurations

In VS Code, there are two core debugging modes, **Launch** and **Attach**, which handle two different workflows and segments of developers. Depending on your workflow, it can be confusing to know what type of configuration is appropriate for your project.

If you come from a browser Developer Tools background, you might not be used to "launching from your tool," since your browser instance is already open. When you open DevTools, you are simply **attaching** DevTools to your open browser tab. On the other hand, if you come from a server or desktop background, it's quite normal to have your editor **launch** your process for you, and your editor automatically attaches its debugger to the newly launched process.

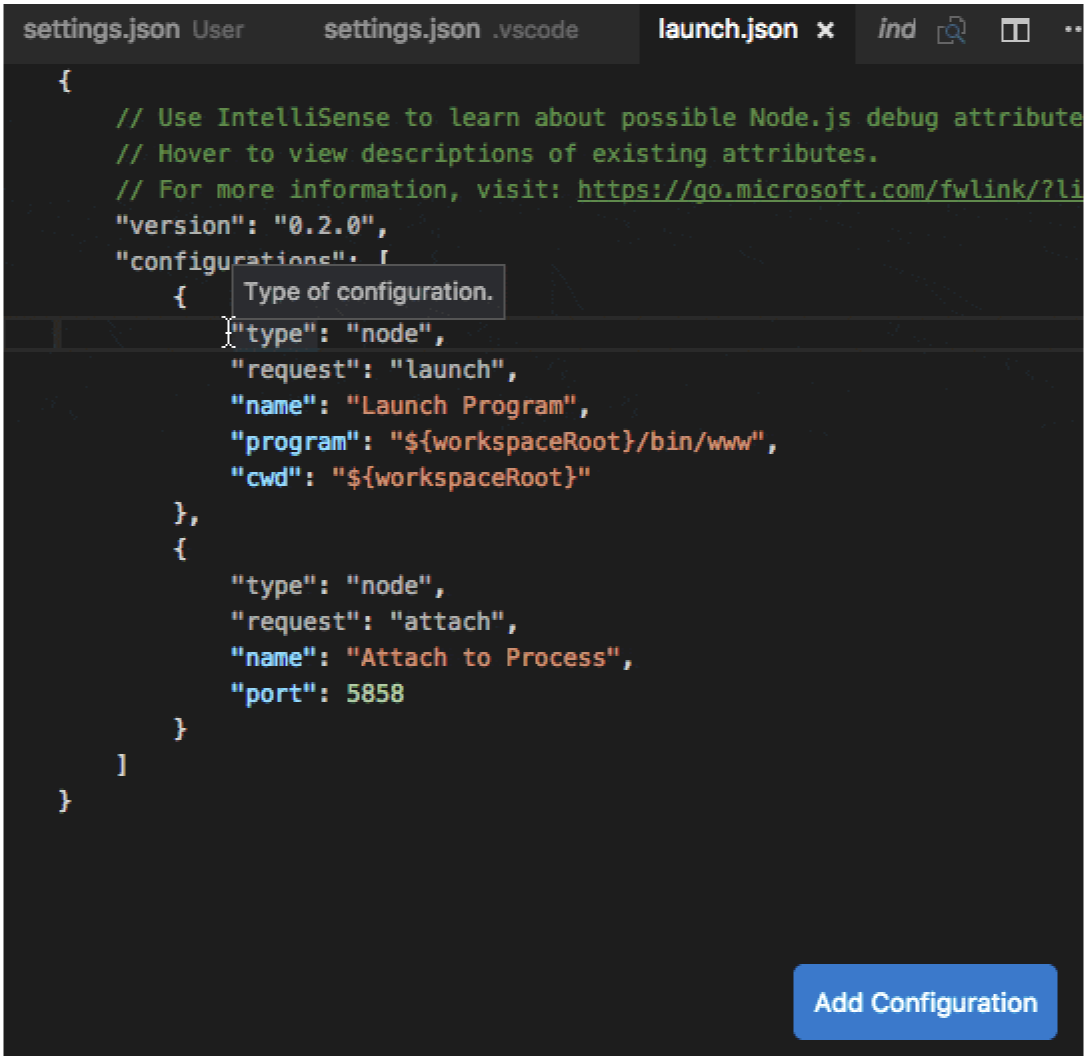
The best way to explain the difference between **launch** and **attach** is to think of a **launch** configuration as a recipe for how to start your app in debug mode **before** VS Code attaches to it, while an **attach** configuration is a recipe for how to connect VS Code's debugger to an app or process that's **already** running.

VS Code debuggers typically support launching a program in debug mode or attaching to an already running program in debug mode. Depending on the request (`attach` or `launch`), different attributes are required, and VS Code's `launch.json` validation and suggestions should help with that.

Add a new configuration

To add a new configuration to an existing `launch.json`, use one of the following techniques:

- Use IntelliSense if your cursor is located inside the configurations array.
- Press the **Add Configuration** button to invoke snippet IntelliSense at the start of the array.
- Choose **Add Configuration** option in the Run menu.



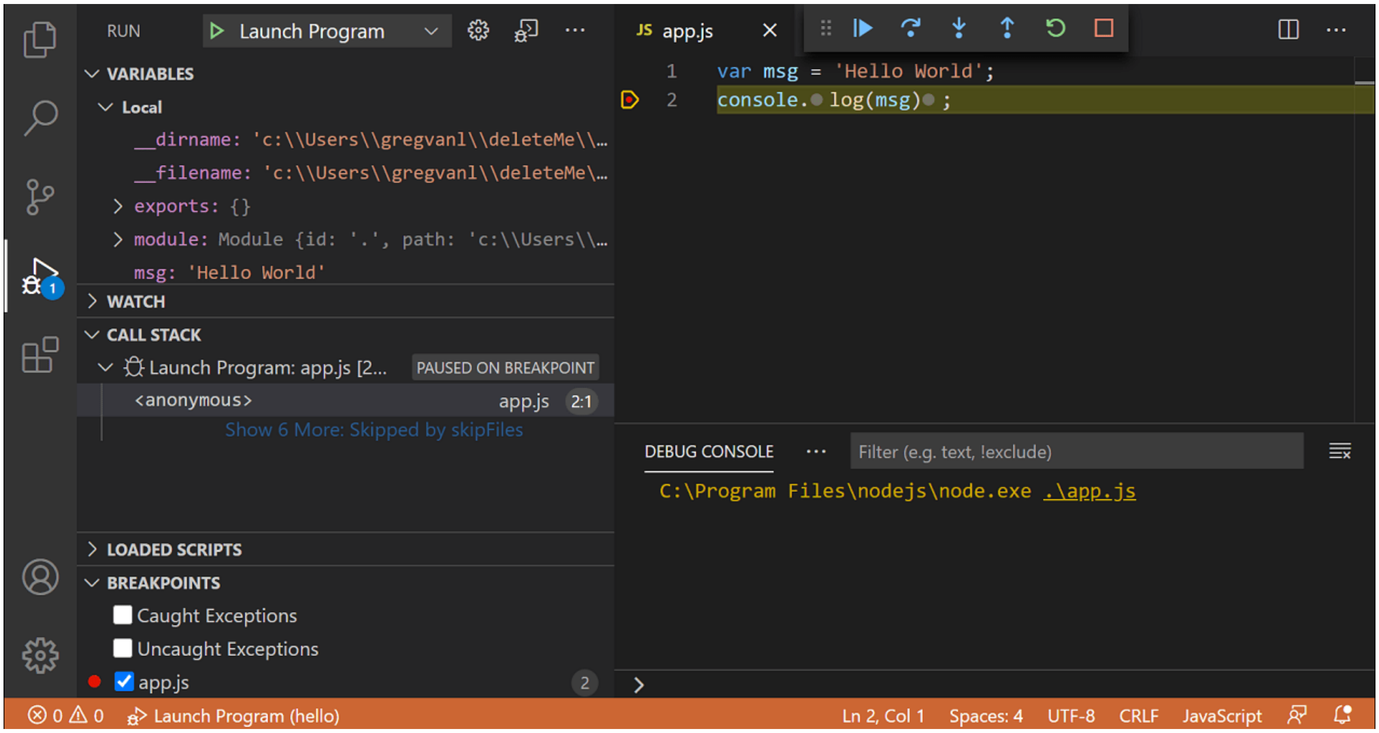
```
{
    // Use IntelliSense to learn about possible Node.js debug attribute
    // Hover to view descriptions of existing attributes.
    // For more information, visit: https://go.microsoft.com/fwlink/?li
    "version": "0.2.0",
    "configurations": [
        {
            "type": "node",
            "request": "launch",
            "name": "Launch Program",
            "program": "${workspaceRoot}/bin/www",
            "cwd": "${workspaceRoot}"
        },
        {
            "type": "node",
            "request": "attach",
            "name": "Attach to Process",
            "port": 5858
        }
    ]
}
```

VS Code also supports compound launch configurations for starting multiple configurations at the same time; for more details, please read this section.

In order to start a debug session, first select the configuration named **Launch Program** using the **Configuration dropdown** in the **Run and Debug** view. Once you have your launch configuration set, start your debug session with `F5`.

Alternatively, you can run your configuration through the **Command Palette** (`Ctrl+Shift+P`) by filtering on **Debug: Select and Start Debugging** or typing `'debug '` and selecting the configuration you want to debug.

As soon as a debugging session starts, the **DEBUG CONSOLE** panel is displayed and shows debugging output, and the Status Bar changes color (orange for default color themes):

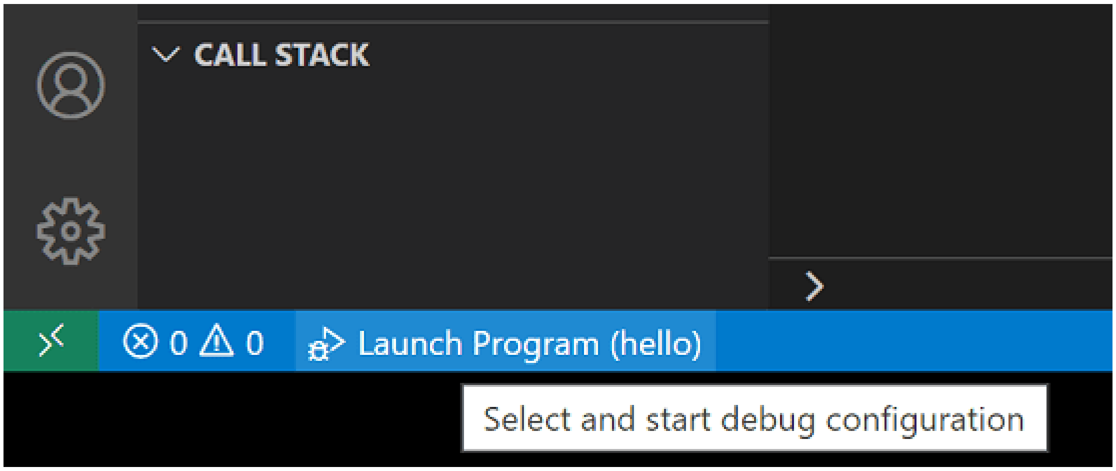


```
__dirname: 'c:\\Users\\gregvanl\\deleteMe\\…
__filename: 'c:\\Users\\gregvanl\\deleteMe\\…
> exports: {}
> module: Module {id: '.', path: 'c:\\Users\\…
msg: 'Hello World'
```

```javascript
var msg = 'Hello World';
console.log(msg);
```

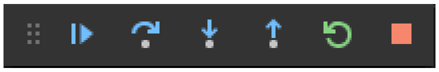
```
C:\Program Files\nodejs\node.exe .\app.js
```

In addition, the **debug status** appears in the Status Bar showing the active debug configuration. By selecting the debug status, a user can change the active launch configuration and start debugging without needing to open the **Run and Debug** view.



Debug actions

Once a debug session starts, the **Debug toolbar** will appear on the top of the editor.



Action	Explanation
Continue / Pause `F5`	**Continue**: Resume normal program/script execution (up to the next breakpoint). **Pause**: Inspect code executing at the current line and debug line-by-line.
Step Over `F10`	Execute the next method as a single command without inspecting or following its component steps.
Step Into `F11`	Enter the next method to follow its execution line-by-line.
Step Out `Shift+F11`	When inside a method or subroutine, return to the earlier execution context by completing remaining lines of the current method as though it were a single command.
Restart `Ctrl+Shift+F5`	Terminate the current program execution and start debugging again using the current run configuration.
Stop `Shift+F5`	Terminate the current program execution.

Tip: Use the setting `debug.toolBarLocation` to control the location of the debug toolbar. It can be the default `floating`, `docked` to the **Run and Debug** view, or `hidden`. A `floating` debug toolbar can be dragged horizontally and also down to the editor area.

Run mode

In addition to debugging a program, VS Code supports **running** the program. The **Debug: Run (Start Without Debugging)** action is triggered with `Ctrl+F5` and uses the currently selected launch configuration. Many of the launch configuration attributes are supported in 'Run' mode. VS Code maintains a debug session while the program is running, and pressing the **Stop** button terminates the program.

Tip: The **Run** action is always available, but not all debugger extensions support 'Run'. In this case, 'Run' will be the same as 'Debug'.

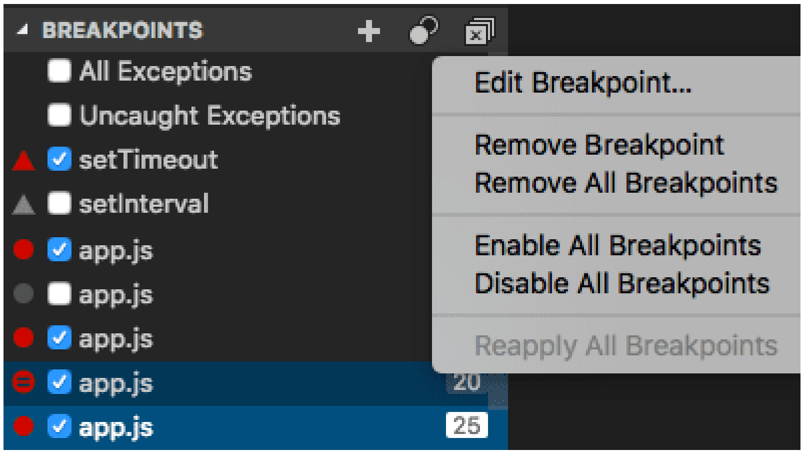
Breakpoints

Breakpoints can be toggled by clicking on the **editor margin** or using `F9` on the current line. Finer breakpoint control (enable/disable/reapply) can be done in the **Run and Debug** view's **BREAKPOINTS** section.

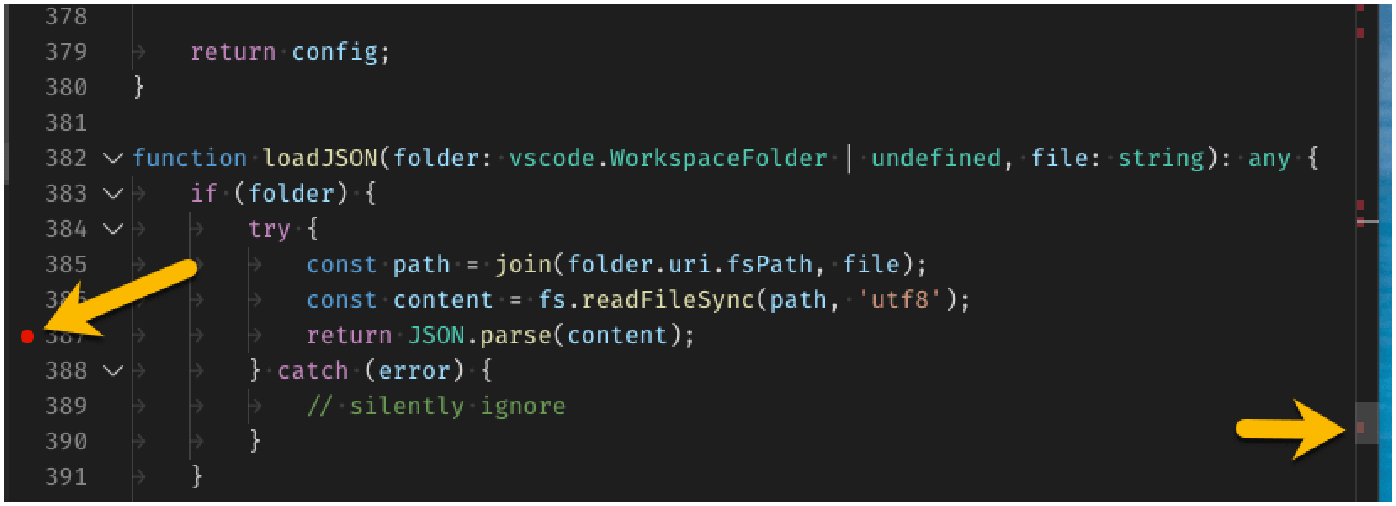
- Breakpoints in the editor margin are normally shown as red filled circles.
- Disabled breakpoints have a filled gray circle.
- When a debugging session starts, breakpoints that cannot be registered with the debugger change to a gray hollow circle. The same might happen if the source is edited while a debug session without live-edit support is running.

If the debugger supports breaking on different kinds of errors or exceptions, those will also be available in the **BREAKPOINTS** view.

The **Reapply All Breakpoints** command sets all breakpoints again to their original location. This is helpful if your debug environment is "lazy" and "misplaces" breakpoints in source code that has not yet been executed.



Optionally, breakpoints can be shown in the editor's overview ruler by enabling the setting `debug.showBreakpointsInOverviewRuler`:

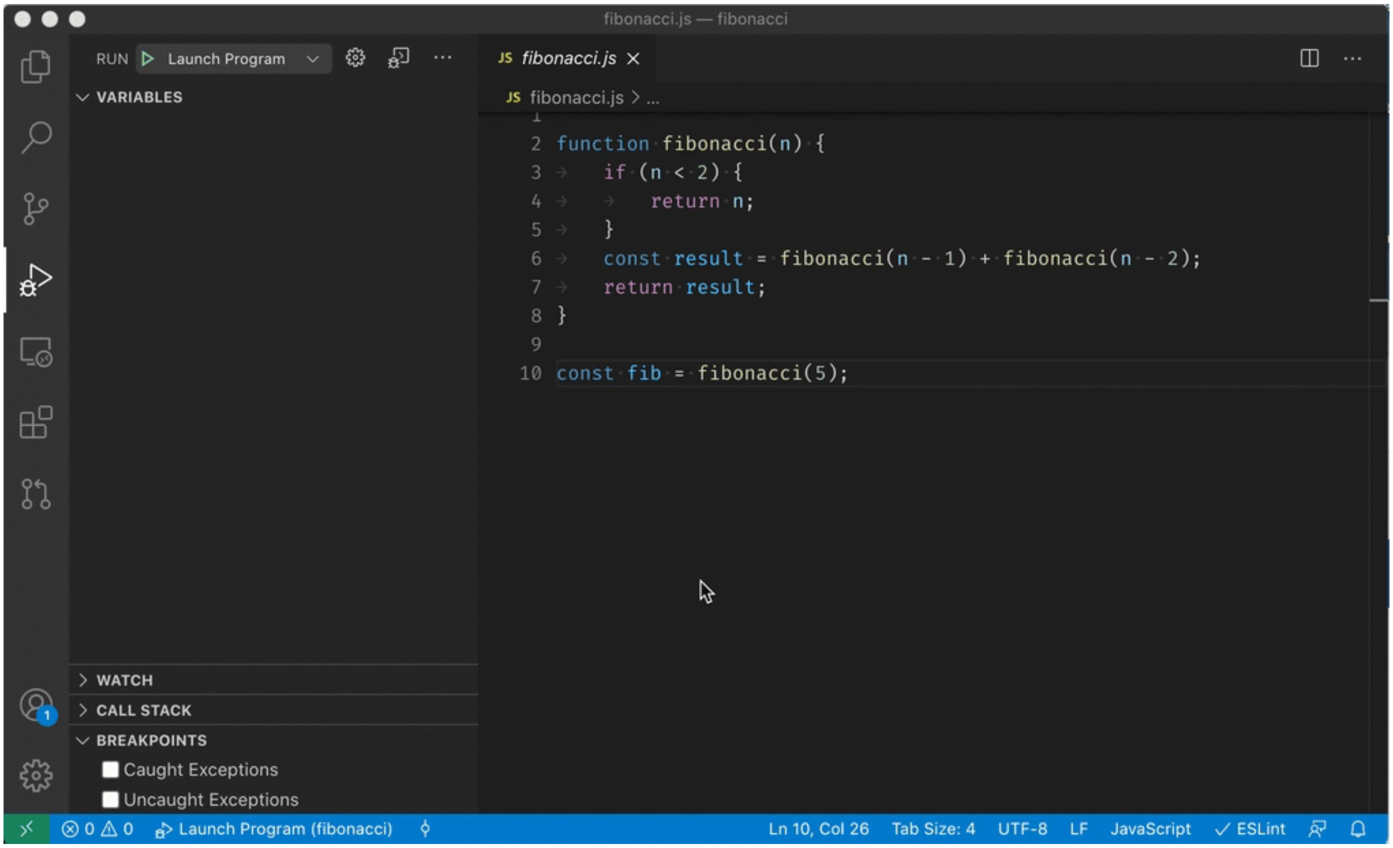


```
    return config;
}

function loadJSON(folder: vscode.WorkspaceFolder | undefined, file: string): any {
    if (folder) {
        try {
            const path = join(folder.uri.fsPath, file);
            const content = fs.readFileSync(path, 'utf8');
            return JSON.parse(content);
        } catch (error) {
            // silently ignore
        }
    }
```

Logpoints

A Logpoint is a variant of a breakpoint that does not "break" into the debugger but instead logs a message to the console. Logpoints are especially useful for injecting logging while debugging production servers that cannot be paused or stopped.

A Logpoint is represented by a "diamond" shaped icon. Log messages are plain text but can include expressions to be evaluated within curly braces ('{}').



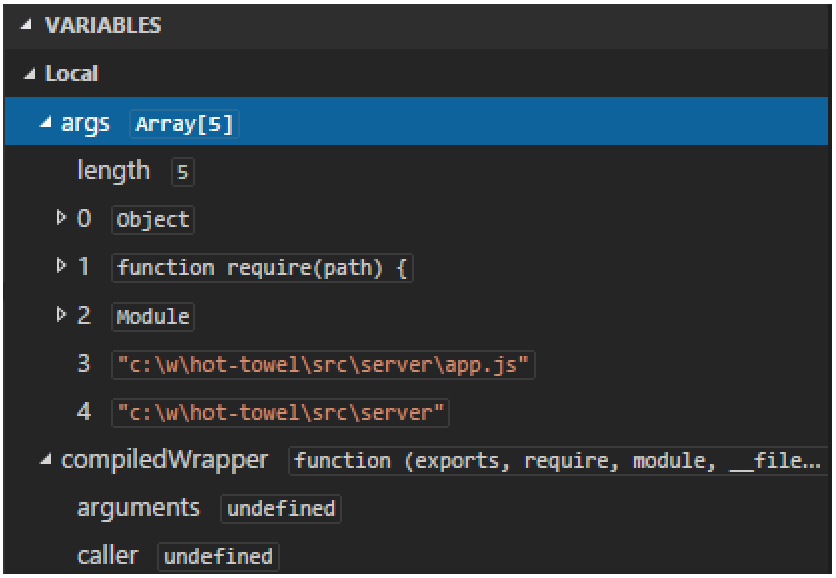
```
function fibonacci(n) {
    if (n < 2) {
        return n;
    }
    const result = fibonacci(n - 1) + fibonacci(n - 2);
    return result;
}

const fib = fibonacci(5);
```

Just like regular breakpoints, Logpoints can be enabled or disabled and can also be controlled by a condition and/or hit count.

Note: Logpoints are supported by VS Code's built-in Node.js debugger, but can be implemented by other debug extensions. The Python and Java extensions, for example, support Logpoints.

Data inspection

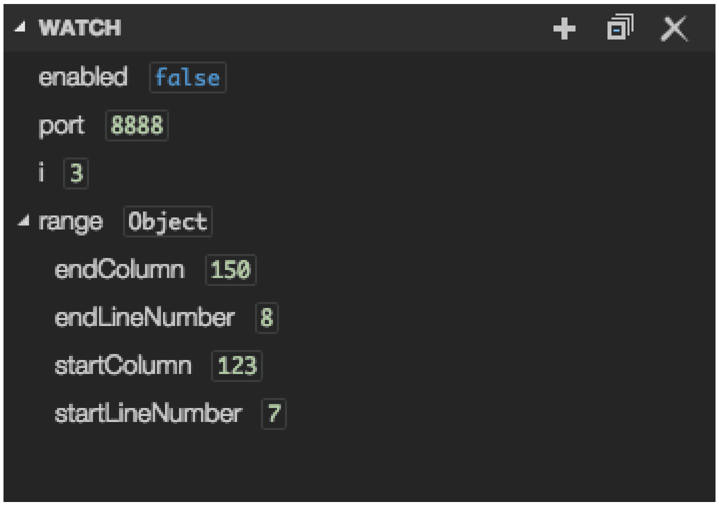
Variables can be inspected in the **VARIABLES** section of the **Run and Debug** view or by hovering over their source in the editor. Variable values and expression evaluation are relative to the selected stack frame in the **CALL STACK** section.



```
VARIABLES
  Local
    args  Array[5]
      length  5
      0  Object
      1  function require(path) {
      2  Module
      3  "c:\w\hot-towel\src\server\app.js"
      4  "c:\w\hot-towel\src\server"
    compiledWrapper  function (exports, require, module, __file…
      arguments  undefined
      caller  undefined
```

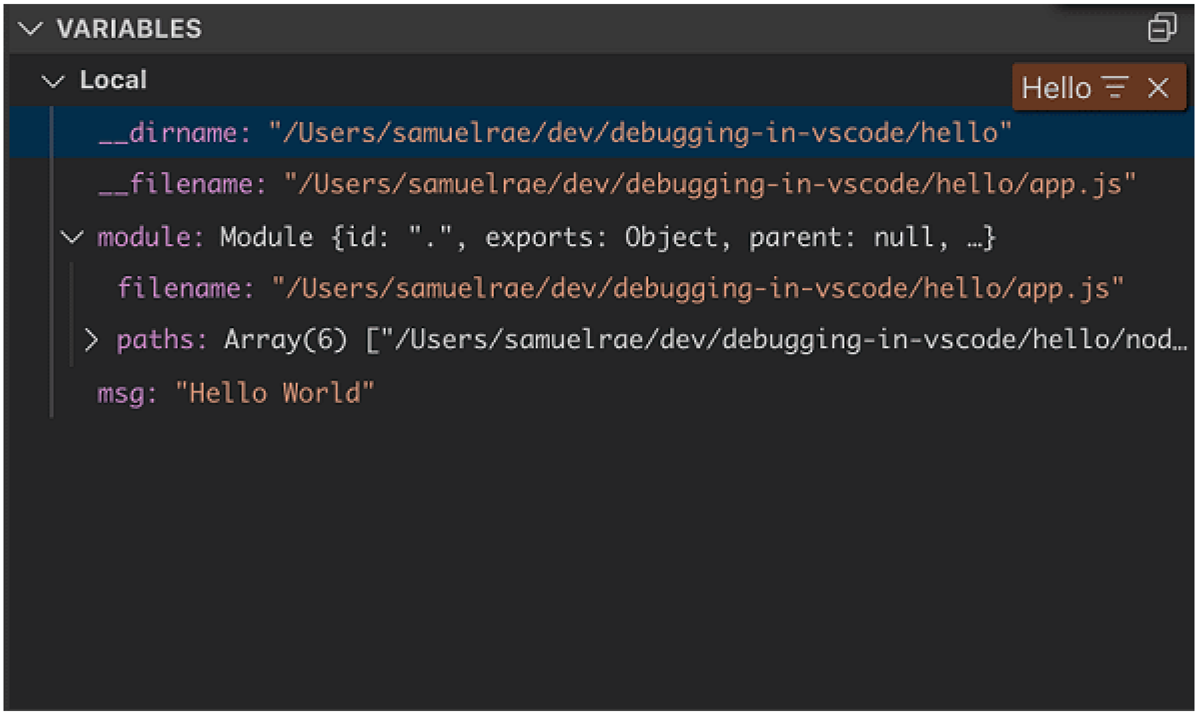
Variable values can be modified with the **Set Value** action from the variable's context menu. Additionally, you can use the **Copy Value** action to copy the variable's value, or **Copy as Expression** action to copy an expression to access the variable.

Variables and expressions can also be evaluated and watched in the **Run and Debug** view's **WATCH** section.



```
WATCH
  enabled  false
  port  8888
  i  3
  range  Object
    endColumn  150
    endLineNumber  8
    startColumn  123
    startLineNumber  7
```

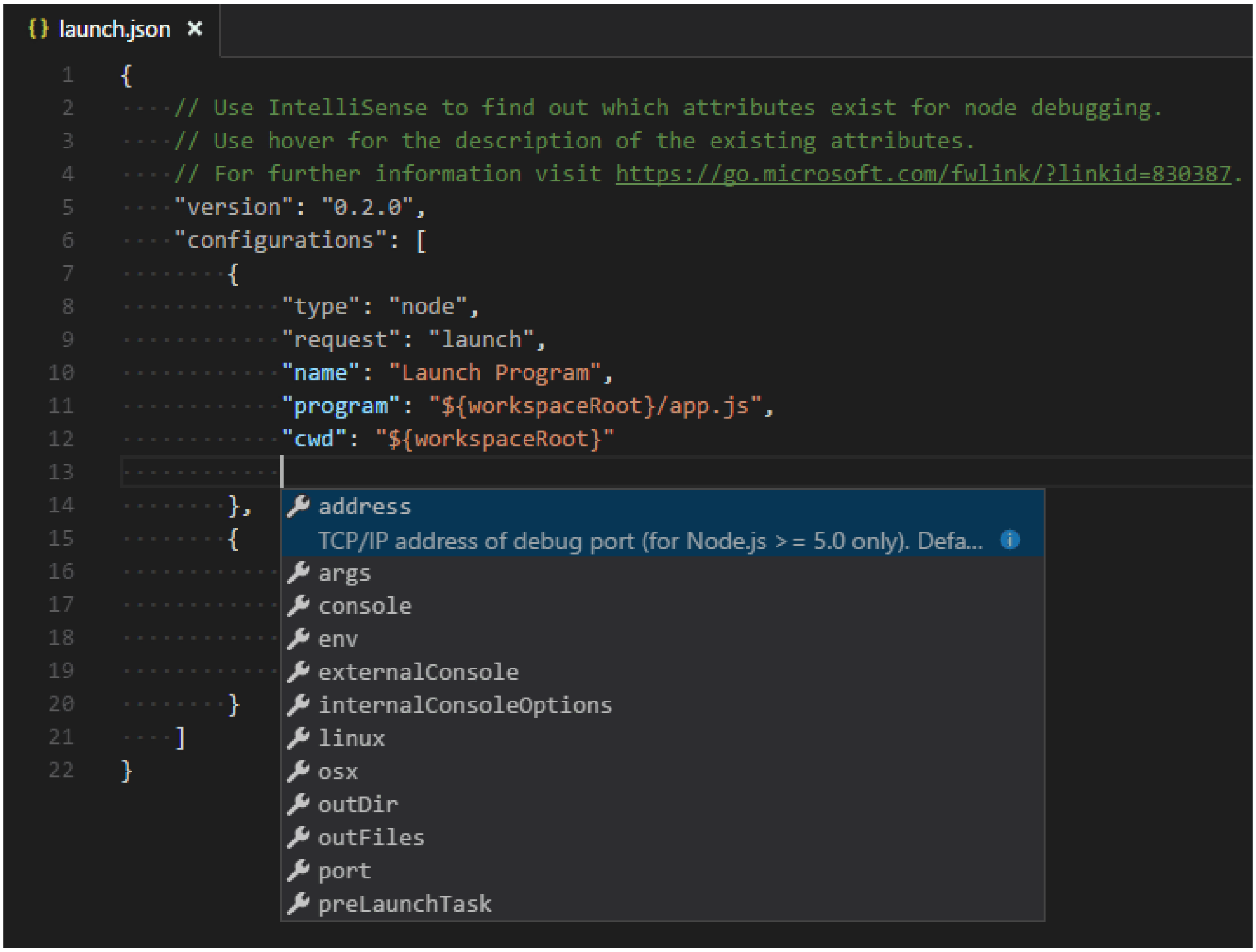
Variable names and values can be filtered by typing while the focus is on the **VARIABLES** section.



```
__dirname: "/Users/samuelrae/dev/debugging-in-vscode/hello"
__filename: "/Users/samuelrae/dev/debugging-in-vscode/hello/app.js"
module: Module {id: ".", exports: Object, parent: null, …}
  filename: "/Users/samuelrae/dev/debugging-in-vscode/hello/app.js"
  paths: Array(6) ["/Users/samuelrae/dev/debugging-in-vscode/hello/nod…
msg: "Hello World"
```

Launch.json attributes

There are many `launch.json` attributes to help support different debuggers and debugging scenarios. As mentioned above, you can use IntelliSense (`Ctrl+Space`) to see the list of available attributes once you have specified a value for the `type` attribute.



```
{
    // Use IntelliSense to find out which attributes exist for node debugging.
    // Use hover for the description of the existing attributes.
    // For further information visit https://go.microsoft.com/fwlink/?linkid=830387.
    "version": "0.2.0",
    "configurations": [
        {
            "type": "node",
            "request": "launch",
            "name": "Launch Program",
            "program": "${workspaceRoot}/app.js",
            "cwd": "${workspaceRoot}"
            
        },
        {




        }
    ]
}
```

The following attributes are mandatory for every launch configuration:

- `type` - the type of debugger to use for this launch configuration. Every installed debug extension introduces a type: `node` for the built-in Node debugger, for example, or `php` and `go` for the PHP and Go extensions.
- `request` - the request type of this launch configuration. Currently, `launch` and `attach` are supported.
- `name` - the reader-friendly name to appear in the Debug launch configuration dropdown.

Here are some optional attributes available to all launch configurations:

- `presentation` - using the `order`, `group`, and `hidden` attributes in the `presentation` object, you can sort, group, and hide configurations and compounds in the Debug configuration dropdown and in the Debug quick pick.
- `preLaunchTask` - to launch a task before the start of a debug session, set this attribute to the label of a task specified in tasks.json (in the workspace's `.vscode` folder). Or, this can be set to `${defaultBuildTask}` to use your default build task.
- `postDebugTask` - to launch a task at the very end of a debug session, set this attribute to the name of a task specified in tasks.json (in the workspace's `.vscode` folder).
- `internalConsoleOptions` - this attribute controls the visibility of the Debug Console panel during a debugging session.

- `debugServer` - **for debug extension authors only**: this attribute allows you to connect to a specified port instead of launching the debug adapter.
- `serverReadyAction` - if you want to open a URL in a web browser whenever the program under debugging outputs a specific message to the debug console or integrated terminal. For details see section Automatically open a URI when debugging a server program below.

Many debuggers support some of the following attributes:

- `program` - executable or file to run when launching the debugger
- `args` - arguments passed to the program to debug
- `env` - environment variables (the value `null` can be used to "undefine" a variable)
- `envFile` - path to dotenv file with environment variables
- `cwd` - current working directory for finding dependencies and other files
- `port` - port when attaching to a running process
- `stopOnEntry` - break immediately when the program launches
- `console` - what kind of console to use, for example, `internalConsole`, `integratedTerminal`, or `externalTerminal`

Variable substitution

VS Code makes commonly used paths and other values available as variables and supports variable substitution inside strings in `launch.json`. This means that you do not have to use absolute paths in debug configurations. For example, `${workspaceFolder}` gives the root path of a workspace folder, `${file}` the file open in the active editor, and `${env:Name}` the environment variable 'Name'. You can see a full list of predefined variables in the Variables Reference or by invoking IntelliSense inside the `launch.json` string attributes.

```json
{

  "type": "node",

  "request": "launch",

  "name": "Launch Program",

  "program": "${workspaceFolder}/app.js",

  "cwd": "${workspaceFolder}",

  "args": ["${env:USERNAME}"]

}
```

Platform-specific properties

`Launch.json` supports defining values (for example, arguments to be passed to the program) that depend on the operating system where the debugger is running. To do so, put a platform-specific literal into the `launch.json` file and specify the corresponding properties inside that literal.

Below is an example that passes `"args"` to the program differently on Windows:

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "node",
      "request": "launch",
      "name": "Launch Program",
      "program": "${workspaceFolder}/node_modules/gulp/bin/gulpfile.js",
      "args": ["myFolder/path/app.js"],
      "windows": {
        "args": ["myFolder\\path\\app.js"]
      }
    }
  ]
}
```

Valid operating properties are `"windows"` for Windows, `"linux"` for Linux, and `"osx"` for macOS. Properties defined in an operating system specific scope override properties defined in the global scope.

Please note that the `type` property cannot be placed inside a platform-specific section, because `type` indirectly determines the platform in remote debugging scenarios, and that would result in a cyclic dependency.

In the example below, debugging the program always **stops on entry** except on macOS:

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "node",
      "request": "launch",

      "name": "Launch Program",

      "program": "${workspaceFolder}/node_modules/gulp/bin/gulpfile.js",

      "stopOnEntry": true,

      "osx": {

        "stopOnEntry": false

      }

    }

  ]

}
```

Global launch configuration

VS Code supports adding a `"launch"` object inside your User settings. This `"launch"` configuration will then be shared across your workspaces. For example:

```
"launch": {

    "version": "0.2.0",

    "configurations": [{

        "type": "node",

        "request": "launch",

        "name": "Launch Program",

        "program": "${file}"

    }]

}
```

Advanced breakpoint topics

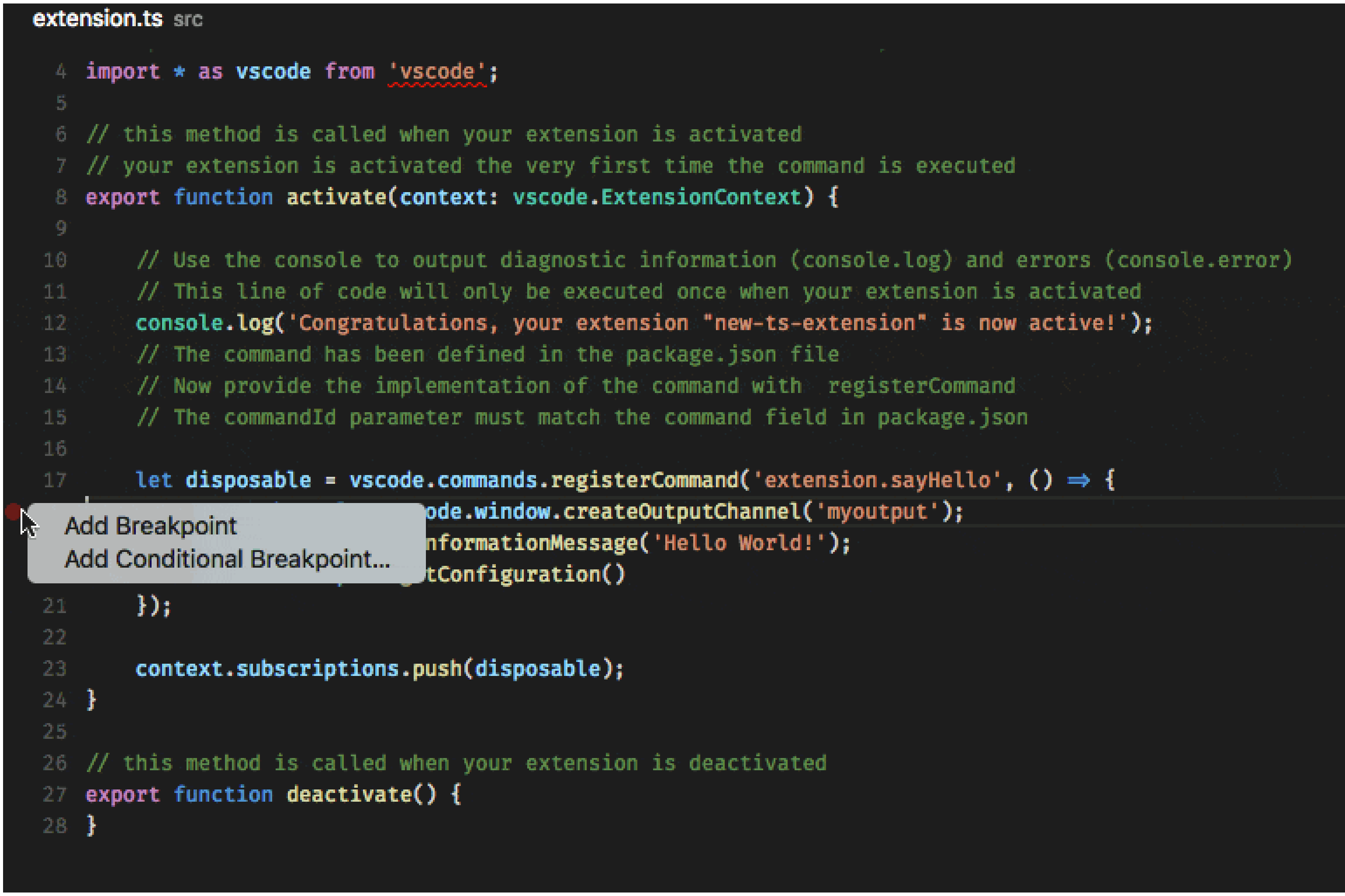
Conditional breakpoints

A powerful VS Code debugging feature is the ability to set conditions based on expressions, hit counts, or a combination of both.

- **Expression condition**: The breakpoint will be hit whenever the expression evaluates to `true`.

- **Hit count**: The 'hit count' controls how many times a breakpoint needs to be hit before it will 'break' execution. Whether a 'hit count' is respected and the exact syntax of the expression vary among debugger extensions.

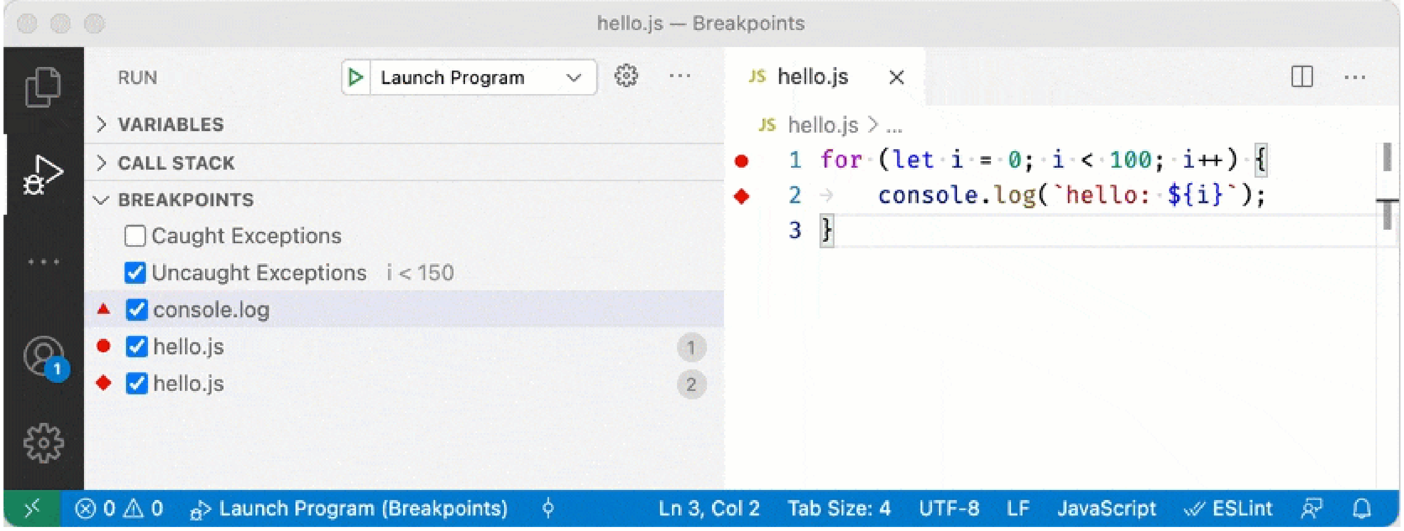
You can add a condition and/or hit count when creating a source breakpoint (with the **Add Conditional Breakpoint** action) or when modifying an existing one (with the **Edit Condition** action). In both cases, an inline text box with a dropdown menu opens where you can enter expressions:



```
import * as vscode from 'vscode';

// this method is called when your extension is activated
// your extension is activated the very first time the command is executed
export function activate(context: vscode.ExtensionContext) {

    // Use the console to output diagnostic information (console.log) and errors (console.error)
    // This line of code will only be executed once when your extension is activated
    console.log('Congratulations, your extension "new-ts-extension" is now active!');
    // The command has been defined in the package.json file
    // Now provide the implementation of the command with  registerCommand
    // The commandId parameter must match the command field in package.json

    let disposable = vscode.commands.registerCommand('extension.sayHello', () => {
                          ode.window.createOutputChannel('myoutput');
                          nformationMessage('Hello World!');
                          tConfiguration()
    });

    context.subscriptions.push(disposable);
}

// this method is called when your extension is deactivated
export function deactivate() {
}
```

Condition and hit count editing support is also supported for **function** and **exception** breakpoints. You can initiate condition editing from the context menu or the new inline **Edit Condition** action.

An example of condition editing in the **BREAKPOINTS** view:



☐ Caught Exceptions
☑ Uncaught Exceptions i < 150
☑ console.log
☑ hello.js 1
☑ hello.js 2

```
for (let i = 0; i < 100; i++) {
    console.log(`hello: ${i}`);
}
```

If a debugger does not support conditional breakpoints, the **Add Conditional Breakpoint** and **Edit Condition** actions will be missing.

Inline breakpoints

Inline breakpoints will only be hit when the execution reaches the column associated with the inline breakpoint. This is particularly useful when debugging minified code which contains multiple statements in a single line.

An inline breakpoint can be set using `Shift+F9` or through the context menu during a debug session. Inline breakpoints are shown inline in the editor.

Inline breakpoints can also have conditions. Editing multiple breakpoints on a line is possible through the context menu in the editor's left margin.

Function breakpoints

Instead of placing breakpoints directly in source code, a debugger can support creating breakpoints by specifying a function name. This is useful in situations where source is not available but a function name is known.

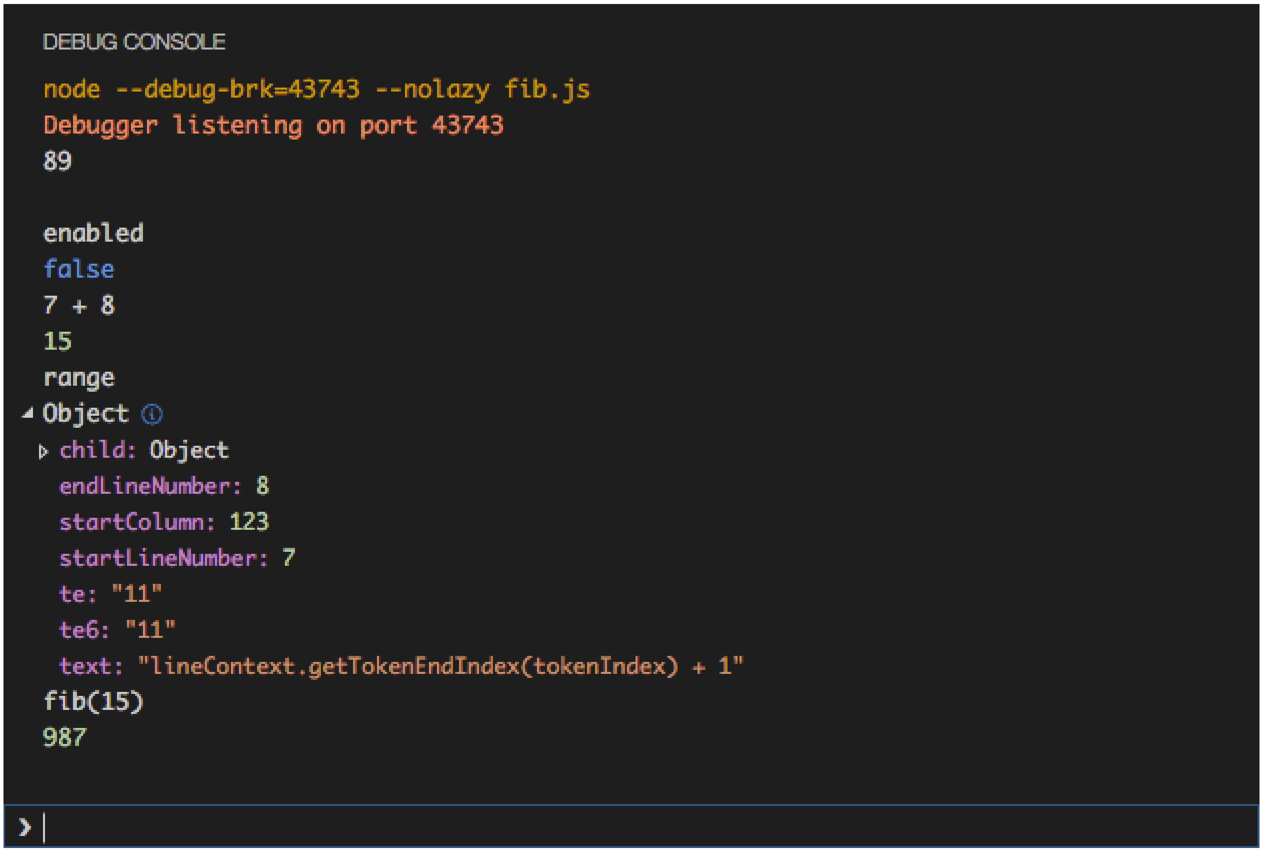
A function breakpoint is created by pressing the **+** button in the **BREAKPOINTS** section header and entering the function name. Function breakpoints are shown with a red triangle in the **BREAKPOINTS** section.

Data breakpoints

If a debugger supports data breakpoints, they can be set from the context menu in the **VARIABLES** view. The **Break on Value Change/Read/Access** commands will add a data breakpoint that is hit when the value of the underlying variable changes/is read/is accessed. Data breakpoints are shown with a red hexagon in the **BREAKPOINTS** section.

Debug Console REPL

Expressions can be evaluated with the **Debug Console** REPL ([Read-Eval-Print Loop](https://en.wikipedia.org/wiki/Read%E2%80%93eval%E2%80%93print_loop)) feature. To open the Debug Console, use the **Debug Console** action at the top of the Debug pane or use the **View: Debug Console** command (`Ctrl+Shift+Y`). Expressions are evaluated after you press `Enter` and the Debug Console REPL shows suggestions as you type. If you need to enter multiple lines, use `Shift+Enter` between the lines and then send all lines for evaluation with `Enter`. Debug Console input uses the mode of the active editor, which means that the Debug Console input supports syntax coloring, indentation, auto closing of quotes, and other language features.



```
node --debug-brk=43743 --nolazy fib.js
Debugger listening on port 43743
89

enabled
false
7 + 8
15
range
Object
 child: Object
 endLineNumber: 8
 startColumn: 123
 startLineNumber: 7
 te: "11"
 te6: "11"
 text: "lineContext.getTokenEndIndex(tokenIndex) + 1"
fib(15)
987

>
```

Note: You must be in a running debug session to use the Debug Console REPL.

Redirect input/output to/from the debug target

Redirecting input/output is debugger/runtime specific, so VS Code does not have a built-in solution that works for all debuggers.

Here are two approaches you might want to consider:

1. Launch the program to debug ("debug target") manually in a terminal or command prompt and redirect input/output as needed. Make sure to pass the appropriate command line options to the debug target so that a debugger can attach to it. Create and run an "attach" debug configuration that attaches to the debug target.
2. If the debugger extension you are using can run the debug target in VS Code's Integrated Terminal (or an external terminal), you can try to pass the shell redirect syntax (for example, "<" or ">") as arguments.

Here's an example `launch.json` configuration:

```
{
  "name": "launch program that reads a file from stdin",
  "type": "node",
  "request": "launch",
  "program": "program.js",
  "console": "integratedTerminal",
  "args": ["<", "in.txt"]
}
```

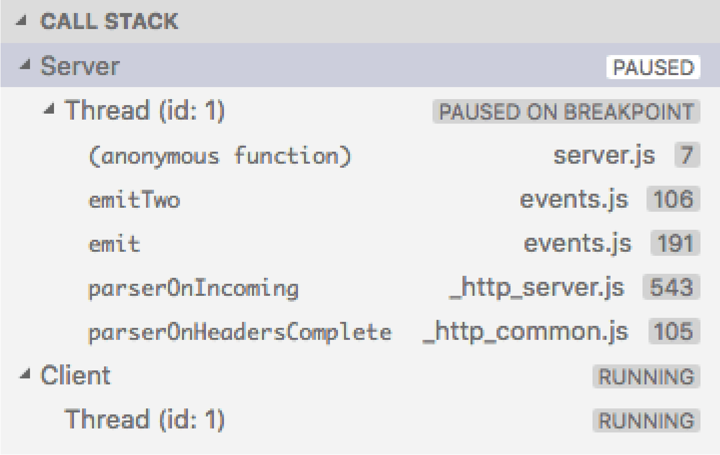
This approach requires that the "<" syntax is passed through the debugger extension and ends up unmodified in the Integrated Terminal.

Multi-target debugging

For complex scenarios involving more than one process (for example, a client and a server), VS Code supports multi-target debugging.

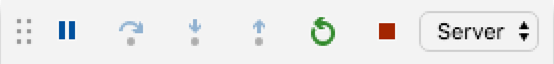
Using multi-target debugging is simple: after you've started a first debug session, you can just launch another session. As soon as a second session is up and running, the VS Code UI switches to *multi-target mode*:

- The individual sessions now show up as top-level elements in the **CALL STACK** view.



```
CALL STACK
  Server                                          PAUSED
    Thread (id: 1)                PAUSED ON BREAKPOINT
      (anonymous function)                 server.js 7
      emitTwo                           events.js 106
      emit                              events.js 191
      parserOnIncoming             _http_server.js 543
      parserOnHeadersComplete      _http_common.js 105
  Client                                          RUNNING
    Thread (id: 1)                                RUNNING
```

- The debug toolbar shows the currently **active session** (and all other sessions are available in a dropdown menu).



- Debug actions (for example, all actions in the debug toolbar) are performed on the active session. The active session can be changed either by using the dropdown menu in the debug toolbar or by selecting a different element in the **CALL STACK** view.

Compound launch configurations

An alternative way to start multiple debug sessions is by using a **compound** launch configuration. A compound launch configuration lists the names of two or more launch configurations that should be launched in parallel. Optionally a `preLaunchTask` can be specified that is run before the individual debug sessions are started. The boolean flag `stopAll` controls whether manually terminating one session will stop all of the compound sessions.

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "node",
      "request": "launch",
      "name": "Server",
      "program": "${workspaceFolder}/server.js"
    },
    {
      "type": "node",
      "request": "launch",
      "name": "Client",
      "program": "${workspaceFolder}/client.js"
    }
  ],
  "compounds": [
    {
      "name": "Server/Client",
      "configurations": ["Server", "Client"],
      "preLaunchTask": "${defaultBuildTask}",
      "stopAll": true
    }
  ]
}
```

Compound launch configurations are displayed in the launch configuration dropdown menu.

Remote debugging

VS Code does not itself support remote debugging: this is a feature of the debug extension you are using, and you should consult the extension's page in the Marketplace for support and details.

There is, however, one exception: the Node.js debugger included in VS Code supports remote debugging. See the Node.js Debugging topic to learn how to configure this.

Automatically open a URI when debugging a server program

Developing a web program typically requires opening a specific URL in a web browser in order to hit the server code in the debugger. VS Code has a built-in feature "**serverReadyAction**" to automate this task.

Here is an example of a simple Node.js Express application:

```
var express = require('express');
var app = express();

app.get('/', function(req, res) {
  res.send('Hello World!');
});

app.listen(3000, function() {
  console.log('Example app listening on port 3000!');
});
```

This application first installs a "Hello World" handler for the "/" URL and then starts to listen for HTTP connections on port 3000. The port is announced in the Debug Console, and typically, the developer would now type `http://localhost:3000` into their browser application.

The **serverReadyAction** feature makes it possible to add a structured property `serverReadyAction` to any launch config and select an "action" to be performed:

```
{
  "type": "node",
  "request": "launch",
  "name": "Launch Program",
  "program": "${workspaceFolder}/app.js",

  "serverReadyAction": {
    "pattern": "listening on port ([0-9]+)",
    "uriFormat": "http://localhost:%s",
    "action": "openExternally"
  }
}
```

Here the `pattern` property describes the regular expression for matching the program's output string that announces the port. The pattern for the port number is put into parenthesis so that it is available as a regular expression capture group. In this example, we are extracting only the port number, but it is also possible to extract a full URI.

The `uriFormat` property describes how the port number is turned into a URI. The first `%s` is substituted by the first capture group of the matching pattern.

The resulting URI is then opened outside of VS Code ("externally") with the standard application configured for the URI's scheme.

Trigger Debugging via Edge or Chrome

Alternatively, the `action` can be set to `debugWithEdge` or `debugWithChrome`. In this mode, a `webRoot` property can be added that is passed to the Chrome or Edge debug session.

To simplify things a bit, most properties are optional and we use the following fallback values:

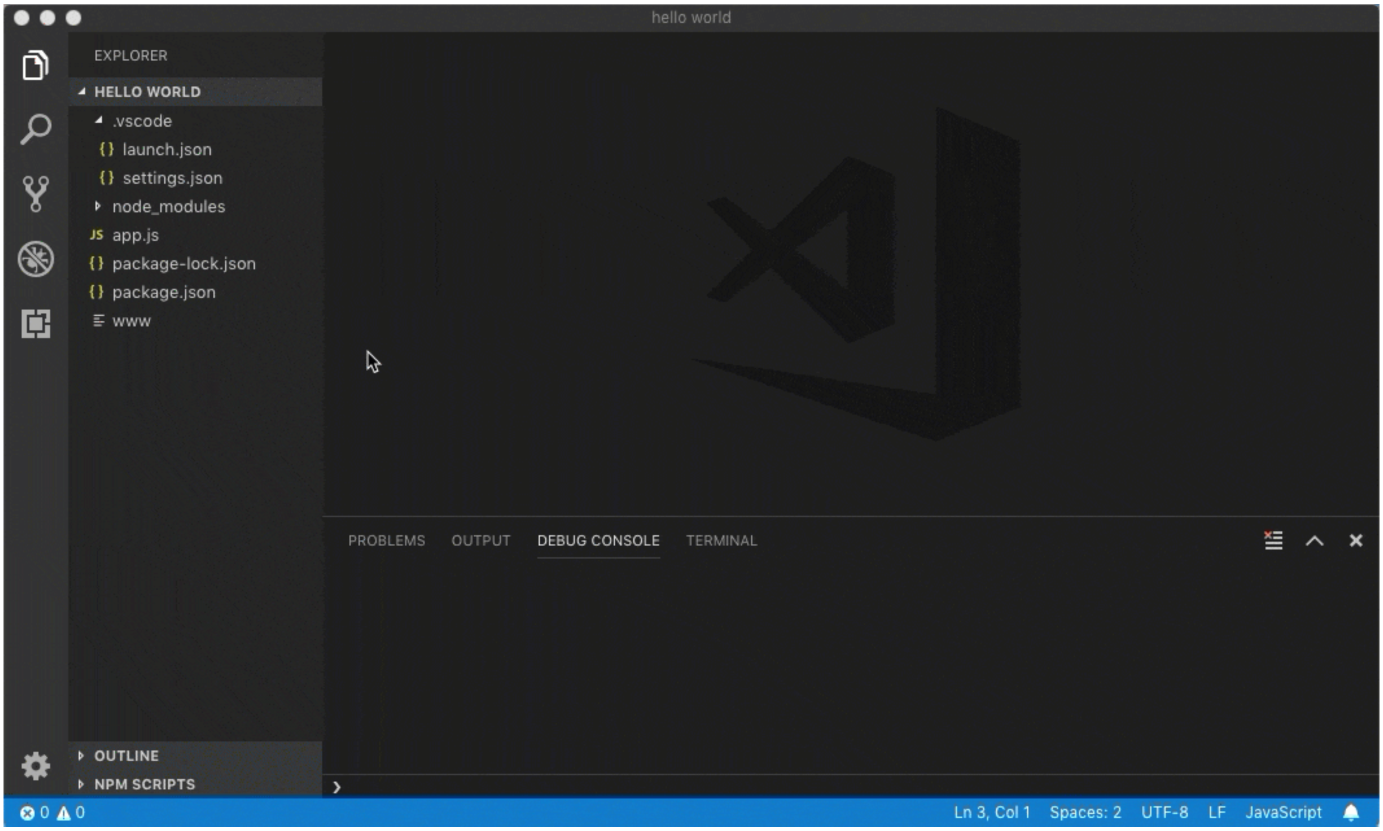
- **pattern**: `"listening on.* (https?://\\S+|[0-9]+)"` which matches the commonly used messages "listening on port 3000" or "Now listening on: https://localhost:5001".
- **uriFormat**: `"http://localhost:%s"`
- **webRoot**: `"${workspaceFolder}"`

Triggering an Arbitrary Launch Config

In some cases, you may need to configure additional options for the browser debug session--or use a different debugger entirely. You can do this by setting `action` to `startDebugging` with a `name` property set to the name of the launch configuration to start when the `pattern` is matched.

The named launch configuration must be in the same file or folder as the one with the `serverReadyAction`.

Here the **serverReadyAction** feature in action:



Next steps

To learn about VS Code's Node.js debugging support, take a look at:

- Node.js - Describes the Node.js debugger, which is included in VS Code.
- TypeScript - The Node.js debugger also supports TypeScript debugging.

To see tutorials on the basics of Node.js debugging, check out these videos:

- Intro Video - Debugging - Showcases the basics of debugging.
- Getting started with Node.js debugging - Shows how to attach a debugger to a running Node.js process.

To learn about debugging support for other programming languages via VS Code extensions:

- C++
- Python
- Java

To learn about VS Code's task running support, go to:

- Tasks - Describes how to run tasks with Gulp, Grunt, and Jake and how to show errors and warnings.

To write your own debugger extension, visit:

- Debugger Extension - Uses a mock sample to illustrate the steps required to create a VS Code debug extension.

Common questions

What are the supported debugging scenarios?

Debugging of Node.js-based applications is supported on Linux, macOS, and Windows out of the box with VS Code. Many other scenarios are supported by VS Code extensions available in the Marketplace.

I do not see any launch configurations in the Run and Debug view dropdown. What is wrong?

The most common problem is that you did not set up `launch.json` or there is a syntax error in that file. Alternatively, you might need to open a folder, since no-folder debugging does not support launch configurations.

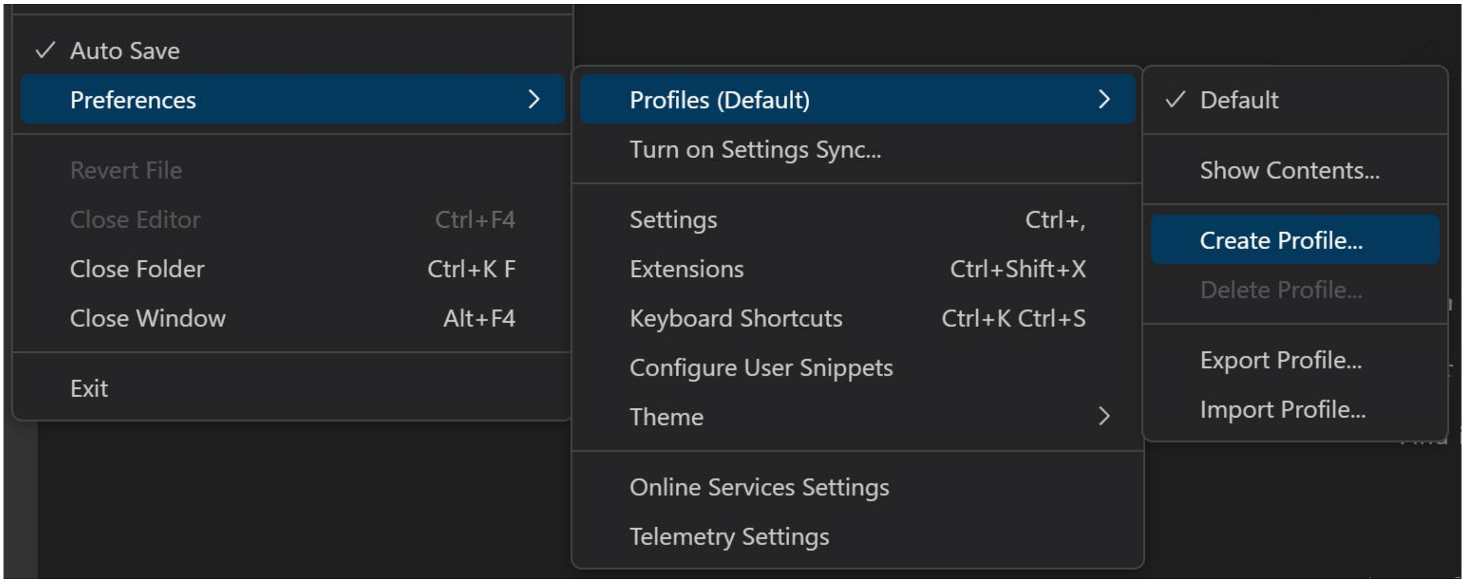
Profiles in Visual Studio Code

Visual Studio Code has hundreds of settings, thousands of extensions, and innumerable ways to adjust the UI layout to customize the editor. VS Code **Profiles** let you create sets of customizations and quickly switch between them or share them with others. This topic explains how to create, modify, export, and import profiles.

Create a Profile

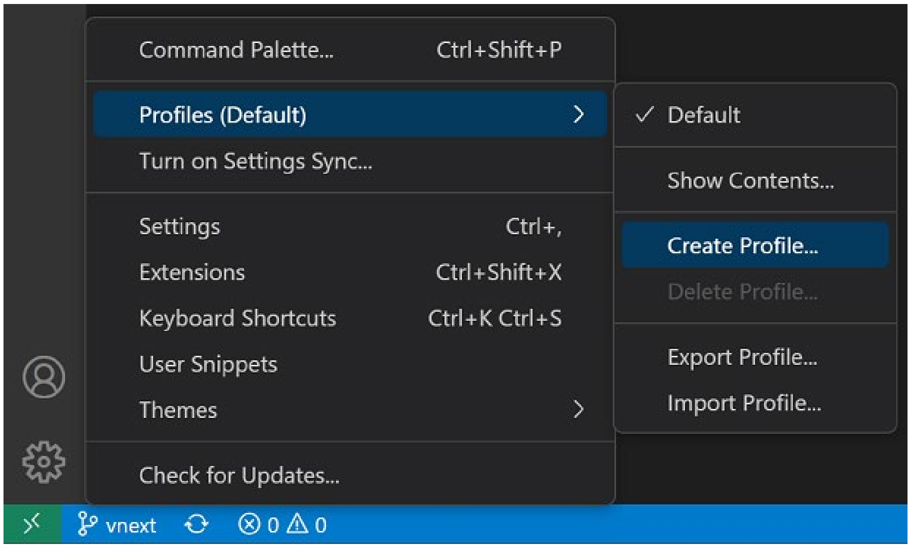
VS Code treats your current configuration as the **Default Profile**. As you modify settings, install extensions, or change UI layout by moving views, these customizations are tracked in the Default Profile.

To create a new profile, you can use the **File** > **Preferences** > **Profiles** > **Create Profile** menu item (**Code** > **Preferences** > **Profiles** > **Create Profile** on macOS).



You can create a new profile based on the current profile (**Profiles: Create from Current Profiles**) or create an Empty Profile. An Empty Profile includes no user customizations (settings, extensions, snippets, etc.).

You can also access the Profile command menu via the **Manage** gear button in the lower right of the Activity bar.



The **Profiles: Create Profile** command is also available in the Command Palette (`Ctrl+Shift+P`).

Once you choose whether to create a new profile based on the current profile or an empty profile, you are prompted to enter a name for the new profile.

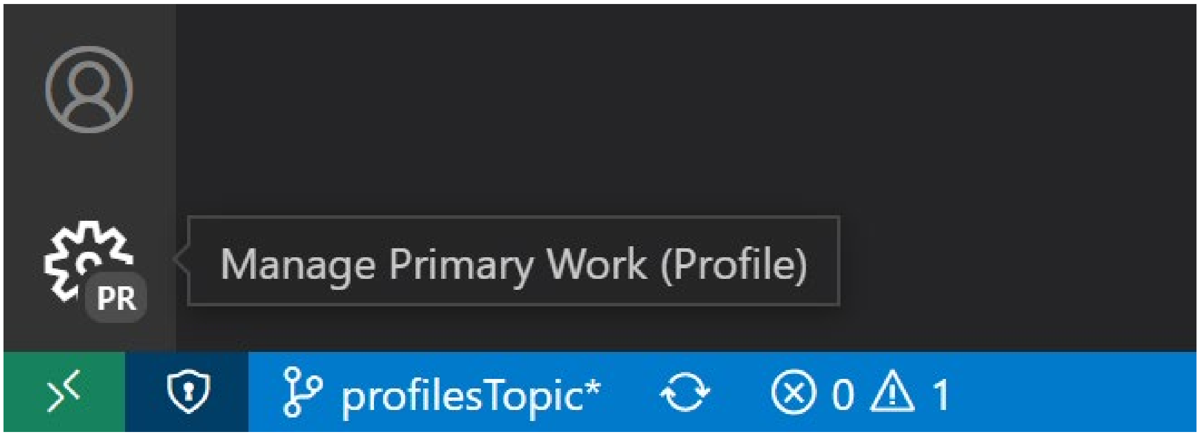
Check the current profile

The current profile name is displayed in several places in the VS Code UI:

- Title bar
- **File** > **Preferences** > **Profiles**
- **Manage** gear button hover

If you are still using the Default Profile, no profile name is displayed.

The **Manage** gear button displays a badge with the first two letters of the active profile so you can quickly check which profile you are running.



Edit a profile

You can edit a profile just as you would normally change any VS Code configuration. You can install/uninstall/disable extensions, change settings, and adjust the editor's UI layout (for example, moving and hiding views) like normal. These changes are stored in your currently active profile.

Workspace associations

When you create or select an existing profile, it is associated with the current workspace and whenever you open that folder, the workspace's profile is active. If you open another folder, the profile switches to that folder's associated profile if one has been set or remains on the last used profile.

Managing profiles

Switch profiles

You can quickly switch between profiles with the **Profiles: Switch Profile** command in the Command Palette, which presents a dropdown listing your available profiles.

You can also switch profiles by selecting a profile from the list displayed in the Profiles menus, available via the **Manage** gear button or **File** > **Preferences** > **Profiles**.

Rename a profile

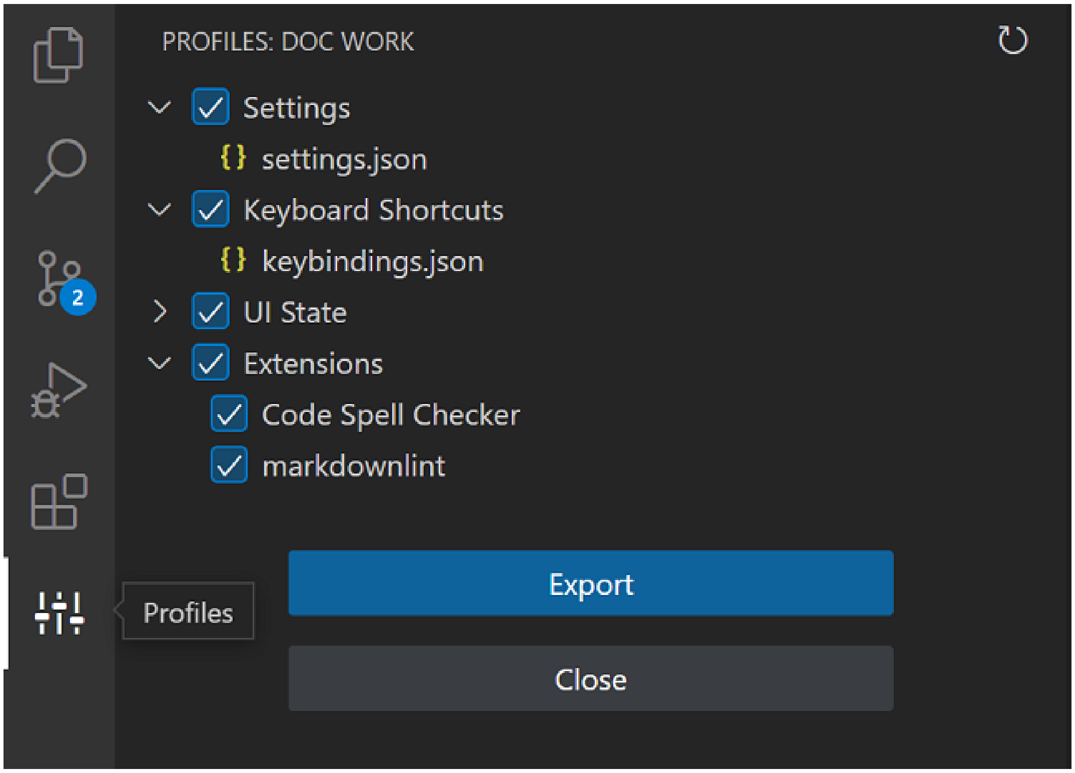
You can rename an existing profile via the **Rename** command in the Profiles menu.

Delete a profile

You can delete a profile via the **Delete Profile** command. The **Delete Profile** drop down lets you select which profile(s) to delete.

Profile contents

The **Profiles: Show Contents** command (available in the Command Palette or Profiles menus) brings up the Profiles view, where you can review the customizations for the profile.



- ☑ Settings
  - {} settings.json
- ☑ Keyboard Shortcuts
  - {} keybindings.json
- ☑ UI State
- ☑ Extensions
  - ☑ Code Spell Checker
  - ☑ markdownlint

A profile can include:

- Settings - In a profile-specific `settings.json` file.
- Extensions - The list of extensions included in the current profile.
- UI state - View layout (positions), visible views and actions.
- Keybindings - In a profile-specific `keybindings.json` file.
- Snippets - In a profile-specific `{language}.json` files.
- User Tasks - In a profile-specific `tasks.json` file.

When you create a new profile based on the Default Profile, the profile-specific configuration files are populated from your user configuration files. Workspace-specific settings are not automatically included in a new profile.

Note: Unchecking an extension in the Profiles view does not remove or disable the extension from the current profile but does remove the extension if you export the profile.

Share Profiles

Export

You can export a profile in order to save it or share it with others. The **Export Profile** command displays the Profiles view with the contents of the active profile and an **Export** button. You can unselect various elements of the profile such as extensions or configuration files before you export the profile.

When you select **Export**, you are prompted for the profile name and whether you want to export to a GitHub gist or your local file system.

Save as a GitHub gist

After you save a profile to GitHub (you'll be prompted to log into GitHub), a dialog gives you the option to **Copy Link** so you can share your profile gist URL with others. The URL includes an autogenerated GUID and has the format `https://vscode.dev/profile/github/{GUID}`. The GitHub gist is marked as **Secret**, so only those with the link can see the gist.

If you launch the profile URL, it opens VS Code for the Web (vscode.dev) with the Profiles view open and the imported profile contents displayed. You can unselect profile elements if you wish and you need to manually **Install Extensions** (via the download cloud button) if you want to continue using that profile in vscode.dev.

You also have the option to **Import Profile in Visual Studio Code**, which opens VS Code Desktop with the profile's contents displayed and an **Import Profile** button.

You can review your gists at `https://gist.github.com/{username}`. From your GitHub gist page you can rename, delete, or copy the GUID of a gist.

Save as a local file

If you chose to save the profile as a local file, a **Save Profile** dialog lets you place the file on your local machine. A profile is persisted in a file with the extension `.code-profile`.

Import

To import an existing profile, run the **Import Profiles** command. You are prompted for the URL of a GitHub gist or the file location of a profile via an **Import Profile** dialog. Once you have selected the profile, the Profiles view opens and displays the profile to import. You can unselect some profile elements if you don't want to import them. Select the **Import Profile** button and you will now be using the imported profile.

Uses for Profiles

Profiles are a great way to customize VS Code to better fit your needs. In this section, we look at some common use cases for profiles.

Since profiles are remembered per workspace, they are a great way to customize VS Code for a specific programming language. For example, you can create a JavaScript frontend profile that includes the extensions, settings, and customizations you use for JavaScript development in one workspace, and have a Python backend profile that includes the extensions, settings, and customizations you use for Python development in another workspace. Using this approach, you can easily switch between workspaces and always have VS Code configured the right way.

Demos

When doing a demo, you can use a profile to set up a specific configuration for your demo. For example, you can create a profile with a specific set of extensions and settings like zoom level, font size, and color theme. By doing this, a demo will not mess up your normal VS Code setup and you can customize VS Code for better visibility during your presentation.

Education

Profiles can be used to customize VS Code for students to ease the use in a classroom setting. Profiles allow educators to quickly share a customized VS Code setup with students. For example, educators can create a profile with a specific set of extensions and settings needed for a computer science class and then share that profile with students.

Report VS Code issues

One use of an Empty Profile is to reset your editor when you want to report an issue with VS Code. An Empty Profile disables all extensions and modified settings so you can quickly see if the issue is due to an extension, a setting, or is in VS Code core.

Profile Templates

VS Code comes with a predefined set of profile templates that you can use to customize VS Code for your specific workflow. To create a new profile based on a template, select a Profile Template when going through the **Create Profile** flow.

Python Profile Template

The Python profile is a good starting point for Python development. It comes with Python specific snippets and has the following extensions:

- autoDocstring - Generate Python docstrings automatically.
- Black Formatter - Formatting support using the black formatter.
- Docker - Create, manage, and debug containerized applications.
- Even Better TOML - Fully-featured TOML support.
- Python - IntelliSense, linting, formatting, debugging, refactoring.
- Python Environment Manager - Manage Python environments and packages.
- Remote Development extension pack - Supports SSH, WSL, and Dev Containers.
- Ruff - Integrates the Ruff Python linter.

This profile also sets the following settings:

```
    "python.analysis.autoImportCompletions": true,

    "python.analysis.fixAll": ["source.unusedImports"],

    "workbench.colorTheme": "Default Dark+ Experimental",

    "editor.defaultFormatter": "ms-python.black-formatter"
```

Data Science Profile Template

The Data Science profile is a good starting point for all data and notebook work. It comes with specific snippets and has the following extensions:

- Data Wrangler - Data cleaning and preparation for tabular datasets.
- Black Formatter - Formatting support using the black formatter.
- Jupyter - Use Jupyter notebooks within VS Code.
- Python - IntelliSense, linting, formatting, debugging, refactoring.
- Dev Containers - Create custom development environments inside a Docker container.
- GitHub Copilot - Your AI pair programmer.

This profile also sets the following settings:

```
    "[python]": {
        "editor.defaultFormatter": "ms-python.black-formatter",
        "editor.formatOnType": true,
        "editor.formatOnSave": true
    },
    "editor.inlineSuggest.enabled": true,
    "editor.lineHeight": 17,
    "breadcrumbs.enabled": false,
    "files.autoSave": "afterDelay",
    "jupyter.themeMatplotlibPlots": true,
    "jupyter.widgetScriptSources": [
        "unpkg.com",
        "jsdelivr.com"
    ],
    "notebook.experimental.outputScrolling": true,
    // "notebook.outline.showCodeCells": true,
    "workbench.colorTheme": "Default Dark+ Experimental",
    "files.exclude": {
        "**/.csv": true,
        "**/.parquet": true,
        "**/.pkl": true,
        "**/.xls": true
    }
```

Doc Writer Profile Template

The Doc Writer profile is a good lightweight setup for writing documentation. It comes with the following extensions:

- Code Spell Checker - Spelling checker for source code.
- Markdown Checkboxes - Adds checkbox support to the VS Code built-in Markdown Preview.
- Markdown Emoji - Adds emoji syntax support to Markdown Preview and notebook Markdown cells.
- Markdown Footnotes - Adds ^footnote syntax support to the Markdown Preview.
- Markdown Preview GitHub Styling - Use GitHub styling in the Markdown Preview.
- Markdown Preview Mermaid Support - Mermaid diagrams and flowcharts.
- Markdown yaml Preamble - Renders YAML front matter as a table.
- markdownlint - Markdown linting and style checking for Visual Studio Code.

This profile also sets the following settings:

```
    "workbench.colorTheme": "Default Light+ Experimental",

    "editor.minimap.enabled": false,

    "breadcrumbs.enabled": false,

    "editor.glyphMargin": false,

    "explorer.decorations.badges": false,

    "explorer.decorations.colors": false,

    "editor.fontLigatures": true,

    "files.autoSave": "afterDelay",

    "git.enableSmartCommit": true,

    "window.commandCenter": true,

    "editor.renderWhitespace": "none",

    "workbench.editor.untitled.hint": "hidden",

    "markdown.validate.enabled": true,

    "markdown.updateLinksOnFileMove.enabled": "prompt",

    "workbench.startupEditor": "none"
```

Node.js Profile Template

The Node.js profile is a good starting point for all Node.js work. It comes with the following extensions:

- ESLint - Integrates ESLint JavaScript into VS Code.
- Dev Containers - Create custom development environments inside a Docker container.
- Docker - Create, manage, and debug containerized applications.
- DotENV - Support for dotenv file syntax.
- EditorConfig for VS Code - EditorConfig Support for Visual Studio Code.
- JavaScript (ES6) code snippets - Code snippets for JavaScript in ES6 syntax.
- Jest - Use Facebook's jest testing framework.
- Microsoft Edge Tools for VS Code - Use the Microsoft Edge Tools from within VS Code.
- npm Intellisense - Autocomplete npm modules in import statements.
- Prettier - Code formatter - Code formatter using Prettier.
- Rest Client - REST Client for Visual Studio Code.
- YAML - YAML language support with built-in Kubernetes syntax.

This profile comes with the following settings:

```
    "editor.formatOnPaste": true,

    "git.autofetch": true,

    "[markdown]":  {

        "editor.wordWrap": "on"

    },

    "[json]": {

        "editor.defaultFormatter": "esbenp.prettier-vscode"

    },

    "[jsonc]": {

        "editor.defaultFormatter": "vscode.json-language-features"

    },

    "[html]": {

        "editor.defaultFormatter": "esbenp.prettier-vscode"

    },

    "[javascript]": {

        "editor.defaultFormatter": "esbenp.prettier-vscode"

    },

    "[typescript]": {

        "editor.defaultFormatter": "esbenp.prettier-vscode"

    },
    "workbench.colorTheme": "Default Dark+ Experimental"
```

Angular Profile Template

The Angular profile is a good starting point for all Angular work. It comes with the following extensions:

- Angular Language Service - Editor services for Angular templates.
- Angular Schematics - Integrate Angular schematics (CLI commands).
- angular2-switcher - Easily navigate to `typescript`|`template`|`style` in angular2 project.
- Dev Containers - Create custom development environments inside a Docker container.
- EditorConfig for VS Code - EditorConfig Support for Visual Studio Code.
- ESLint - Integrates ESLint JavaScript into VS Code.
- JavaScript (ES6) code snippets - Code snippets for JavaScript in ES6 syntax.
- Jest - Use Facebook's jest testing framework.
- Material Icon Theme - Material Design Icons for Visual Studio Code.
- Microsoft Edge Tools for VS Code - Use the Microsoft Edge Tools from within VS Code.
- Playwright Test for VSCode - Run Playwright tests in Visual Studio Code.
- Prettier - Code formatter - Code formatter using Prettier.
- Rest Client - REST Client for Visual Studio Code.
- YAML - YAML language support with built-in Kubernetes syntax.

This profile sets the following settings:

```
    "editor.formatOnPaste": true,

    "git.autofetch": true,

    "[markdown]":  {

        "editor.wordWrap": "on"

    },

    "[json]": {

        "editor.defaultFormatter": "esbenp.prettier-vscode"

    },

    "[jsonc]": {

        "editor.defaultFormatter": "vscode.json-language-features"

    },

    "[html]": {

        "editor.defaultFormatter": "esbenp.prettier-vscode"

    },
    "[javascript]": {

        "editor.defaultFormatter": "esbenp.prettier-vscode"

    },

    "[typescript]": {

        "editor.defaultFormatter": "esbenp.prettier-vscode"

    },

    "workbench.iconTheme": "material-icon-theme",

    "workbench.colorTheme": "Default Dark+ Experimental",
```

Java General Profile Template

The Java General profile is a good starting point for all Java work. It customizes the layout to improve the Java experience and comes with the following extensions from the Extension Pack for Java:

- Debugger for Java - A lightweight Java debugger.
- IntelliCode - AI-assisted development.
- IntelliCode API Usage Examples - Provides code examples for over 100K different APIs.
- Language Support for Java(TM) by Red Hat - Fundamental Java language support, Linting, Intellisense, formatting, refactoring.
- Maven for Java - Manage Maven projects and builds.
- Project Manager for Java - Manage Java projects within VS Code.
- Test Runner for Java - Run and debug JUnit or TestNG test cases.

Java Spring Profile Template

The Java Spring profile is a good starting point for all Java and Spring developers. It builds on the Java General profile and add the following extensions from the Spring Boot Extension Pack:

- Spring Boot Dashboard - Provides Spring Boot live data visualization and observation in your running Spring applications.
- Spring Boot Tools - Rich language support for Spring Boot files.
- Spring Initializr Java Support - Scaffold and generate Spring Boot Java projects.

This profile sets the following settings:

```
    "[java]": {

        "editor.defaultFormatter": "redhat.java"

    },

    "boot-java.rewrite.reconcile": true,

    "workbench.colorTheme": "Default Dark+ Experimental"
```

Command line

You can launch VS Code with a specific profile via the `--profile` command-line interface option. You pass the name of the profile after the `--profile` argument and open a folder or a workspace using that profile. The command line below opens the `web-sample` folder with the "Web Development" profile:

```
code ~/projects/web-sample --profile "Web Development"
```

If the profile specified does not exist, a new empty profile with the given name is created.

Common Questions

Where are profiles kept?

Profiles are stored under your User configurations similar to your user settings and keybindings.

- **Windows** `%APPDATA%\Code\User\profiles`
- **macOS** `$HOME/Library/Application\ Support/Code/User/profiles`
- **Linux** `$HOME/.config/Code/User/profiles`

If you are using the Insiders version, the intermediate folder name is `Code - Insiders`.

Where is the UI State globalState.json file?

If you expand the **UI State** node in the Profiles view, there is a `globalState.json` entry. This is an in-memory JSON representation of your profile's UI State, describing the visibility and layout of various VS Code UI elements. The file does not actually exist on disk and is just a JSON view of the underlying global state storage.

What is a Temporary Profile?

A Temporary Profile is a profile that is not saved across VS Code sessions. You create a Temporary Profile via the **Profiles: Create a Temporary Profile** command in the Command Palette. The Temporary Profile starts as an Empty Profile and has an automatically generated name (such as **Temp 1**). You can modify the profile settings and extensions, use the profile for the lifetime of your VS Code session, but it will be deleted once you close VS Code.

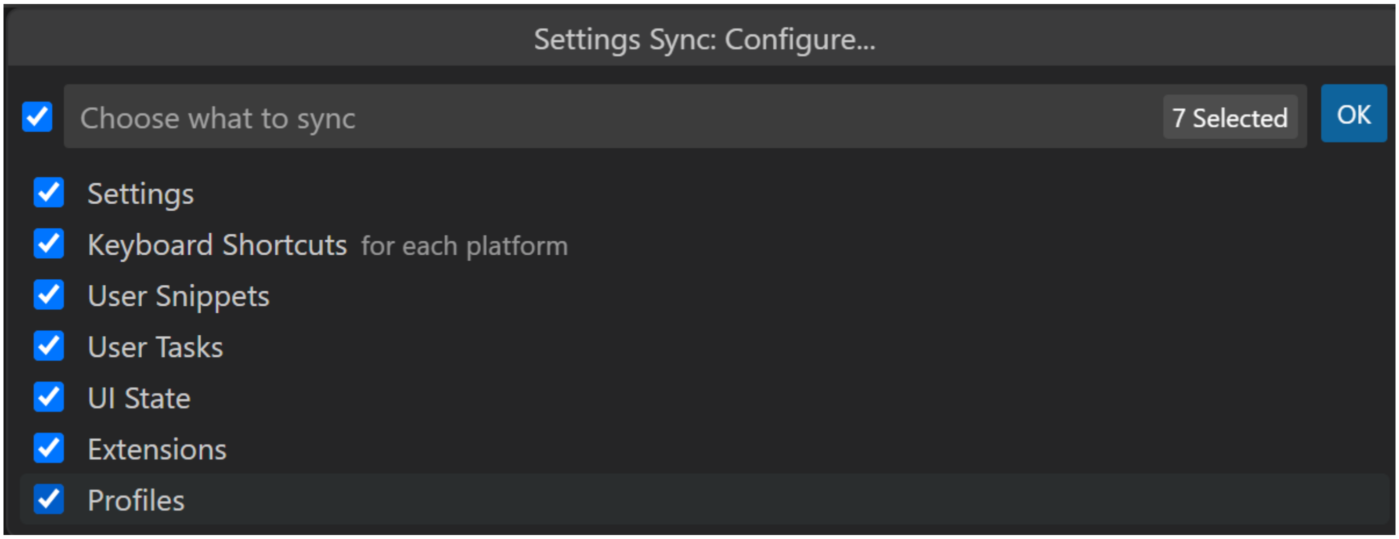
Temporary Profiles are useful if you want to try a new configuration or test an extension without modifying your default or existing profile. Restarting VS Code reenables the current profile for your workspace.

How can I remove the profile from my project?

You can set your project back to the Default Profile. If you'd like to remove all profile workspace associations, you can use the **Developer: Reset Workspace Profiles Associations**, which will set all local folders currently assigned a profile back to the Default Profile. **Reset Workspace Profiles Associations** does not delete any existing profiles.

Do profiles sync across machines (via Settings Sync)?

Yes, you can use Settings Sync to move your profiles across various machines. With Setting Sync enabled and **Profiles** checked in the **Settings Sync: Configure** drop down, all your created profiles are available.



Settings Sync: Configure...

☑ Choose what to sync 7 Selected OK

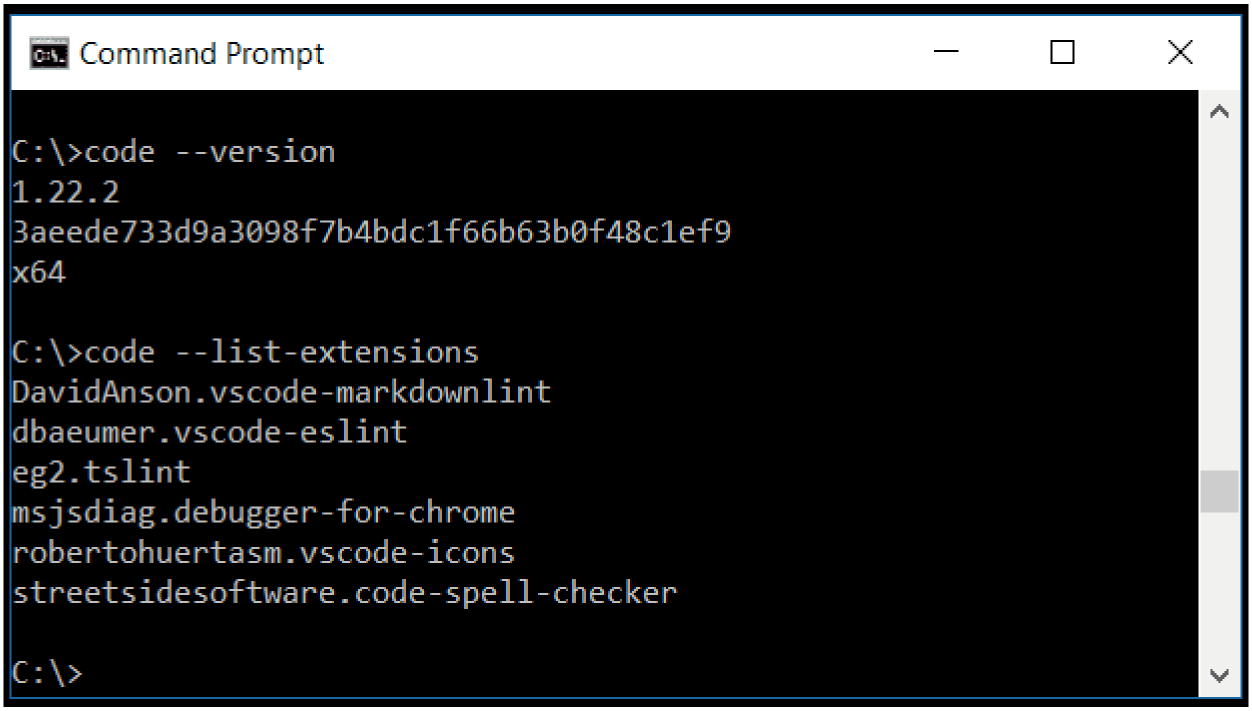
- ☑ Settings
- ☑ Keyboard Shortcuts for each platform
- ☑ User Snippets
- ☑ User Tasks
- ☑ UI State
- ☑ Extensions
- ☑ Profiles

Why are some settings not exported when exporting a profile?

When exporting profiles, machine-specific settings are not included because these setting would not be applicable on another machine. For example, settings that point to local paths are not included.

Command Line Interface (CLI)

Visual Studio Code has a powerful command-line interface built-in that lets you control how you launch the editor. You can open files, install extensions, change the display language, and output diagnostics through command-line options (switches).

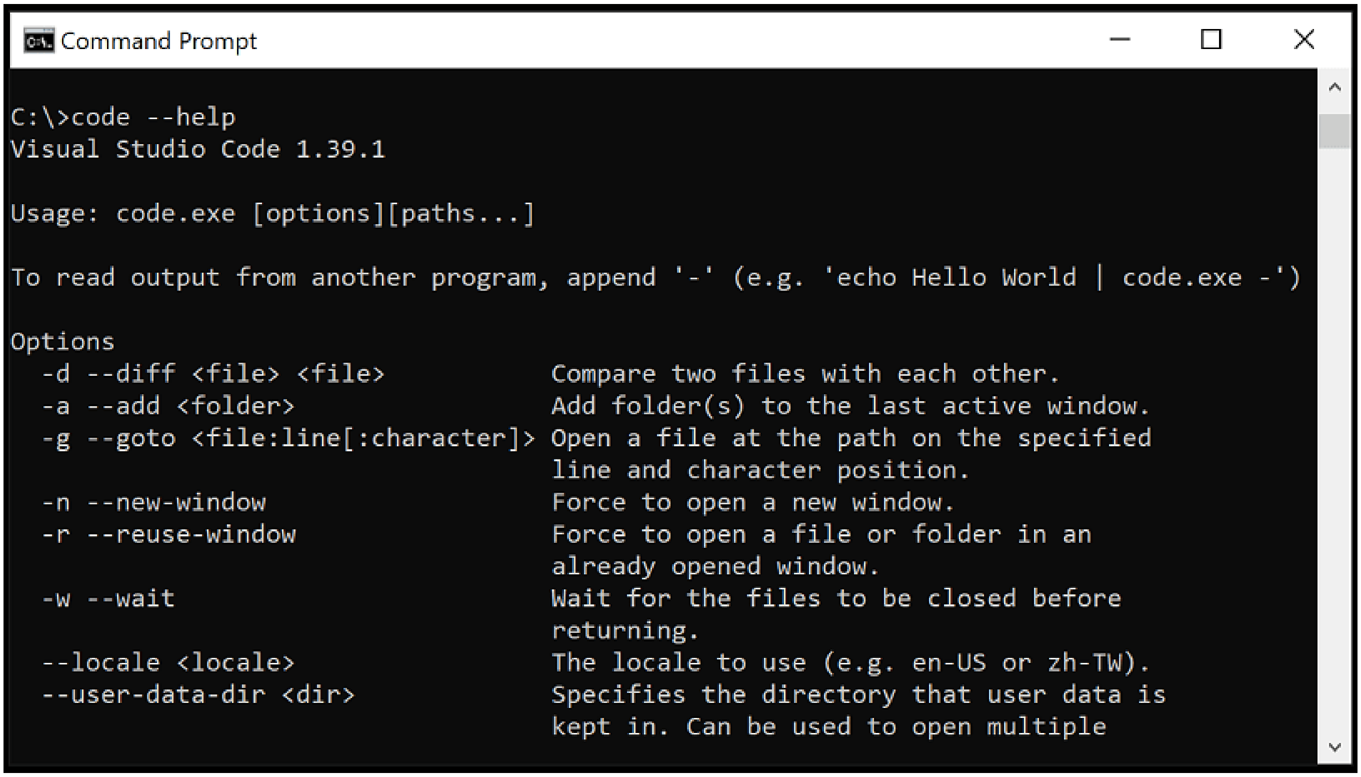


```
C:\>code --version
1.22.2
3aeede733d9a3098f7b4bdc1f66b63b0f48c1ef9
x64

C:\>code --list-extensions
DavidAnson.vscode-markdownlint
dbaeumer.vscode-eslint
eg2.tslint
msjsdiag.debugger-for-chrome
robertohuertasm.vscode-icons
streetsidesoftware.code-spell-checker

C:\>
```

If you are looking for how to run command-line tools inside VS Code, see the Integrated Terminal.

Command line help

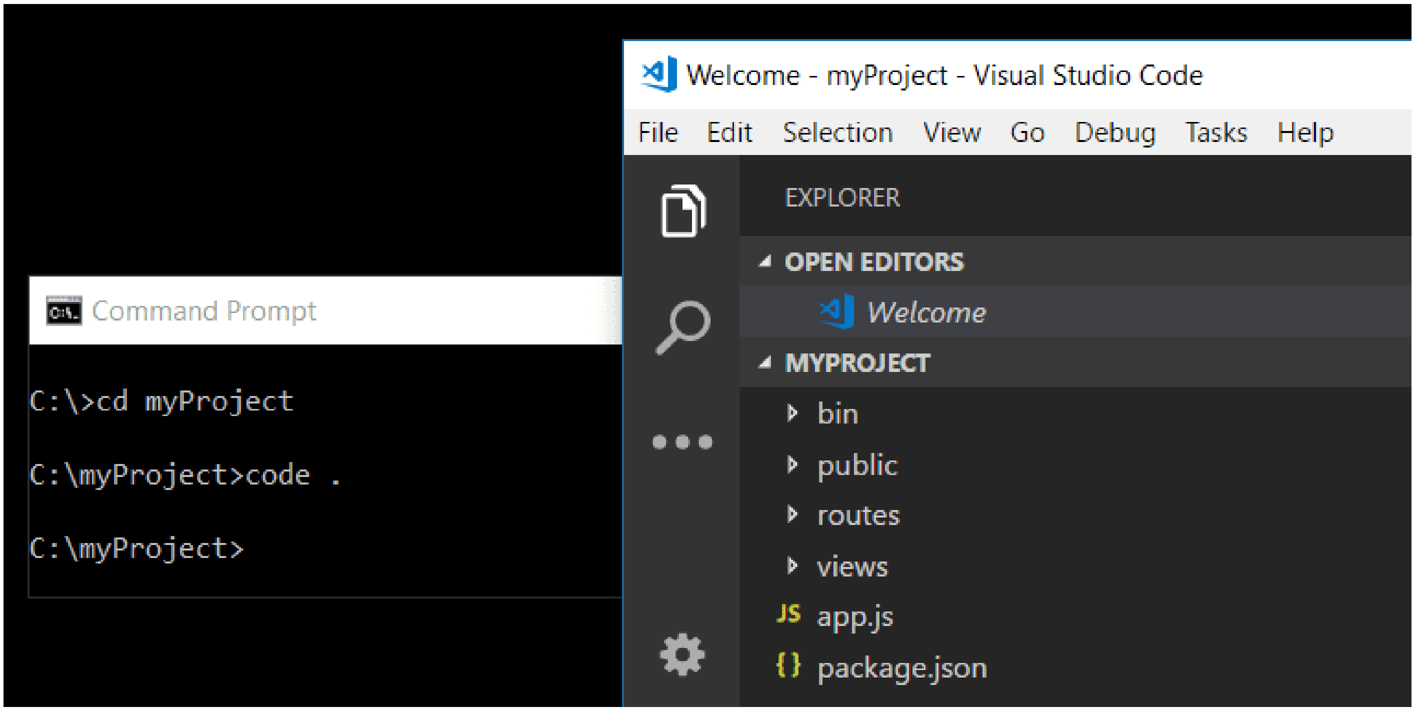
To get an overview of the VS Code command-line interface, open a terminal or command prompt and type `code --help`. You will see the version, usage example, and list of command line options.



```
C:\>code --help
Visual Studio Code 1.39.1

Usage: code.exe [options][paths...]

To read output from another program, append '-' (e.g. 'echo Hello World | code.exe -')

Options
  -d --diff <file> <file>           Compare two files with each other.
  -a --add <folder>                 Add folder(s) to the last active window.
  -g --goto <file:line[:character]> Open a file at the path on the specified
                                    line and character position.
  -n --new-window                   Force to open a new window.
  -r --reuse-window                 Force to open a file or folder in an
                                    already opened window.
  -w --wait                         Wait for the files to be closed before
                                    returning.
  --locale <locale>                 The locale to use (e.g. en-US or zh-TW).
  --user-data-dir <dir>             Specifies the directory that user data is
                                    kept in. Can be used to open multiple
```

Launching from command line

You can launch VS Code from the command line to quickly open a file, folder, or project. Typically, you open VS Code within the context of a folder. To do this, from an open terminal or command prompt, navigate to your project folder and type `code .`:



```
C:\>cd myProject

C:\myProject>code .

C:\myProject>
```

Note: Users on macOS must first run a command (**Shell Command: Install 'code' command in PATH**) to add VS Code executable to the `PATH` environment variable. Read the macOS setup guide for help.

Windows and Linux installations should add the VS Code binaries location to your system path. If this isn't the case, you can manually add the location to the `Path` environment variable (`$PATH` on Linux). For example, on Windows, VS Code is installed under `AppData\Local\Programs\Microsoft VS Code\bin`. To review platform-specific setup instructions, see Setup.

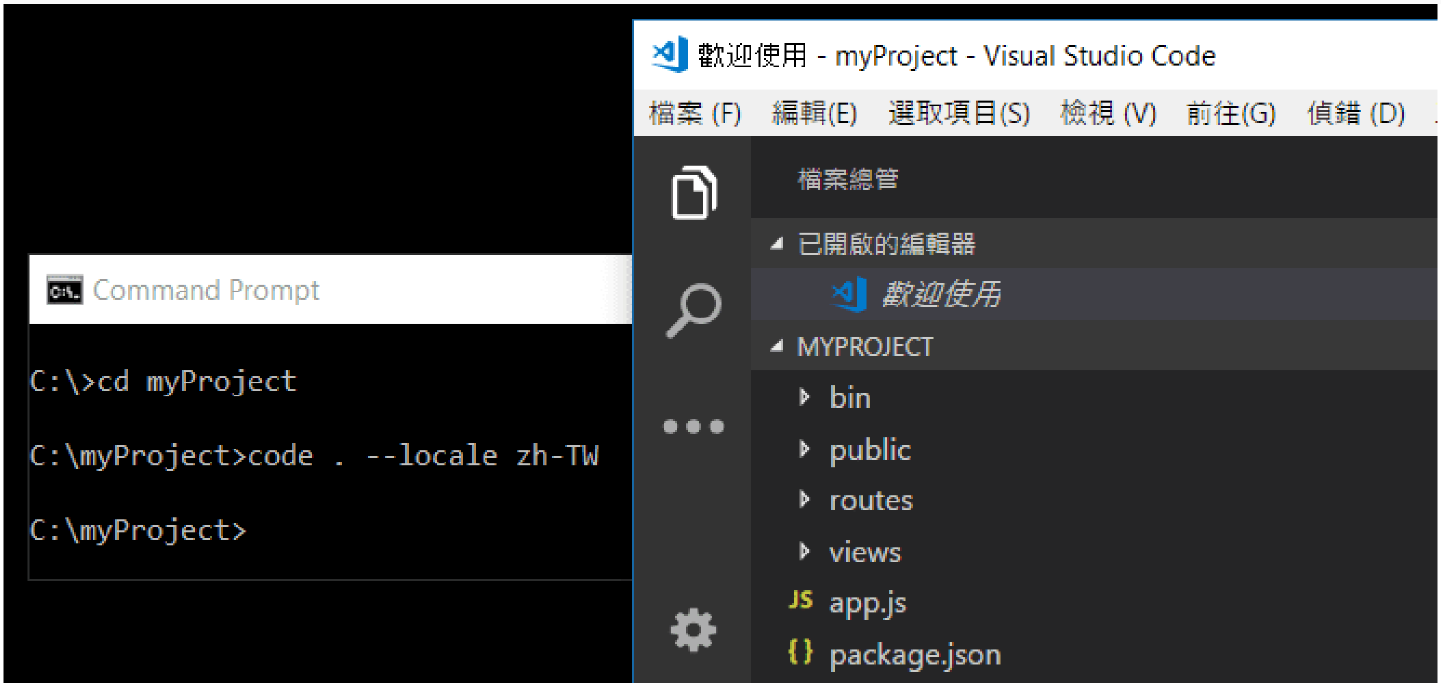
Insiders: If you are using the VS Code Insiders preview, you launch your Insiders build with `code-insiders`.

Core CLI options

Here are optional arguments you can use when starting VS Code at the command line via `code`:

Argument	Description
`-h` or `--help`	Print usage
`-v` or `--version`	Print VS Code version (for example, 1.22.2), GitHub commit ID, and architecture (for example, x64).
`-n` or `--new-window`	Opens a new session of VS Code instead of restoring the previous session (default).
`-r` or `--reuse-window`	Forces opening a file or folder in the last active window.
`-g` or `--goto`	When used with *file:line{:character}*, opens a file at a specific line and optional character position. This argument is provided since some operating systems permit `:` in a file name.
`-d` or `--diff <file1> <file2>`	Open a file difference editor. Requires two file paths as arguments.
`-m` or `--merge <path1> <path2> <base> <result>`	Perform a three-way merge by providing paths for two modified versions of a file, the common origin of both modified versions, and the output file to save merge results.
`-w` or `--wait`	Wait for the files to be closed before returning.

Argument	Description
`--locale <locale>`	Set the display language (locale) for the VS Code session. (for example, `en-US` or `zh-TW`)



```
C:\>cd myProject

C:\myProject>code . --locale zh-TW

C:\myProject>
```

Opening Files and Folders

Sometimes you will want to open or create a file. If the specified file does not exist, VS Code will create them for you along with any new intermediate folders:

```
code index.html style.css documentation\readme.md
```

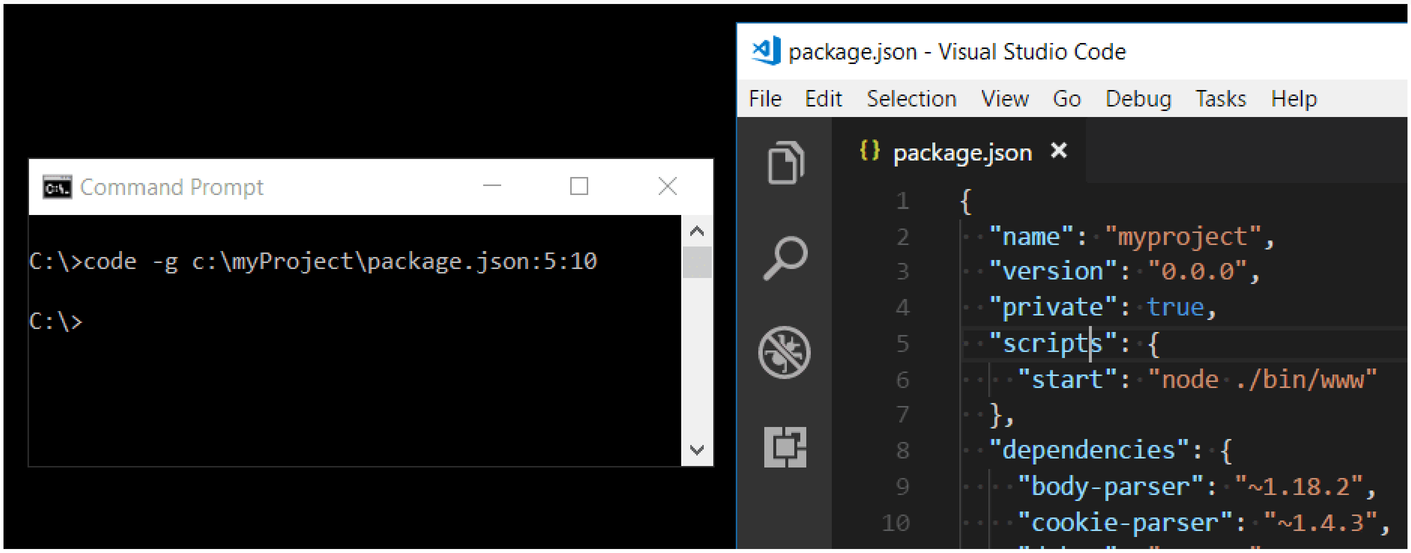
For both files and folders, you can use absolute or relative paths. Relative paths are relative to the current directory of the command prompt where you run `code`.

If you specify more than one file at the command line, VS Code will open only a single instance.

If you specify more than one folder at the command line, VS Code will create a Multi-root Workspace including each folder.

Argument	Description
`file`	Name of a file to open. If the file doesn't exist, it will be created and marked as edited. You can specify multiple files by separating each file name with a space.

Argument	Description
`file:line[:character]`	Used with the `-g` argument. Name of a file to open at the specified line and optional character position.
`folder`	Name of a folder to open. You can specify multiple folders and a new [Multi-root Workspace](#) is created.


Command Prompt
package.json - Visual Studio Code
File Edit Selection View Go Debug Tasks Help
package.json

```
C:\>code -g c:\myProject\package.json:5:10

C:\>
```

```
{
  "name": "myproject",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "body-parser": "~1.18.2",
    "cookie-parser": "~1.4.3",
```

Select a profile

You can launch VS Code with a specific [profile](#) via the `--profile` command-line interface option. You pass the name of the profile after the `--profile` argument and open a folder or a workspace using that profile. The command line below opens the `web-sample` folder with the "Web Development" profile:

`code ~/projects/web-sample --profile "Web Development"`

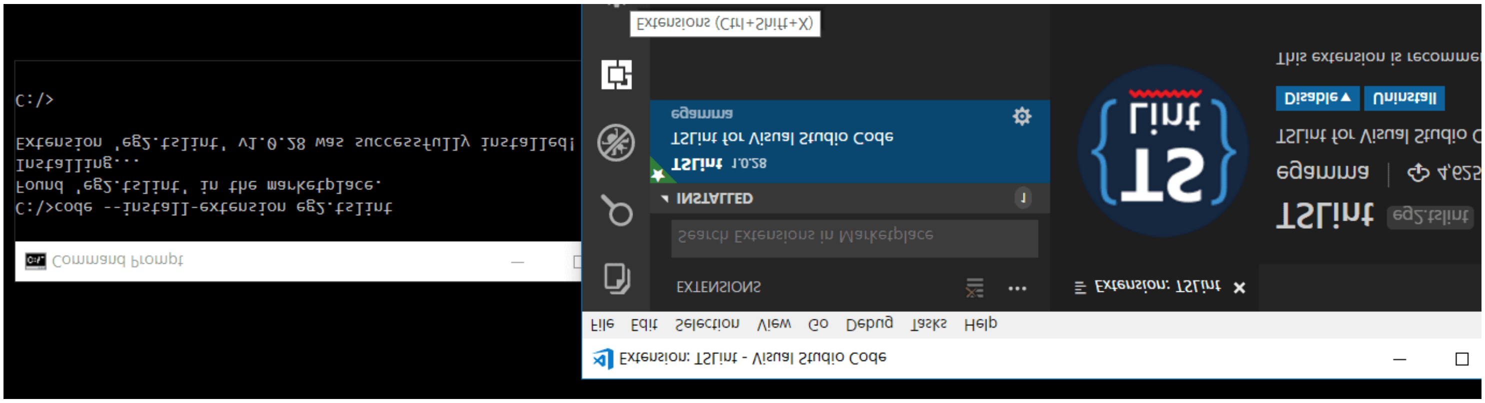
If the profile specified does not exist, a new empty profile with the given name is created.

Working with extensions

You can install and manage VS Code [extensions](#) from the command line.

Argument	Description
`--install-extension <ext>`	Install an extension. Provide the full extension name `publisher.extension` as an argument. Use `--force` argument to avoid prompts.

Argument	Description
`--uninstall-extension <ext>`	Uninstall an extension. Provide the full extension name `publisher.extension` as an argument.
`--disable-extensions`	Disable all installed extensions. Extensions will still be visible in the **Disabled** section of the Extensions view but they will never be activated.
`--list-extensions`	List the installed extensions.
`--show-versions`	Show versions of installed extensions, when using `--list-extensions`
`--enable-proposed-api <ext>`	Enables proposed api features for an extension. Provide the full extension name `publisher.extension` as an argument.


Command Prompt
Extension: TSLint - Visual Studio Code
File Edit Selection View Go Debug Tasks Help
EXTENSIONS
Extension: TSLint
Search Extensions in Marketplace
INSTALLED
TSLint 1.0.28
TSLint for Visual Studio Code
egamma
Extensions (Ctrl+Shift+X)
TSLint
eg2.tslint
egamma
4,625
TSLint for Visual Studio C
Disable ▾
Uninstall
This extension is recomme
{ TS Lint }

```
C:\>code --install-extension eg2.tslint
Found 'eg2.tslint' in the marketplace.
Installing...
Extension 'eg2.tslint' v1.0.28 was successfully installed!
C:\>
```

Advanced CLI options

There are several CLI options that help with reproducing errors and advanced setup.

Argument	Description
`--extensions-dir <dir>`	Set the root path for extensions. Has no effect in Portable Mode.
`--user-data-dir <dir>`	Specifies the directory that user data is kept in, useful when running as root. Has no effect in Portable Mode.

Argument	Description
`-s, --status`	Print process usage and diagnostics information.
`-p, --performance`	Start with the **Developer: Startup Performance** command enabled.
`--disable-gpu`	Disable GPU hardware acceleration.
`--verbose`	Print verbose output (implies `--wait`).
`--prof-startup`	Run CPU profiler during startup.
`--upload-logs`	Uploads logs from current session to a secure endpoint.
Multi-root	
`--add <dir>`	Add folder(s) to the last active window for a multi-root workspace.

Create remote tunnel

VS Code integrates with other remote environments to become even more powerful and flexible. Our goal is to provide a cohesive experience that allows you to manage both local and remote machines from one, unified CLI.

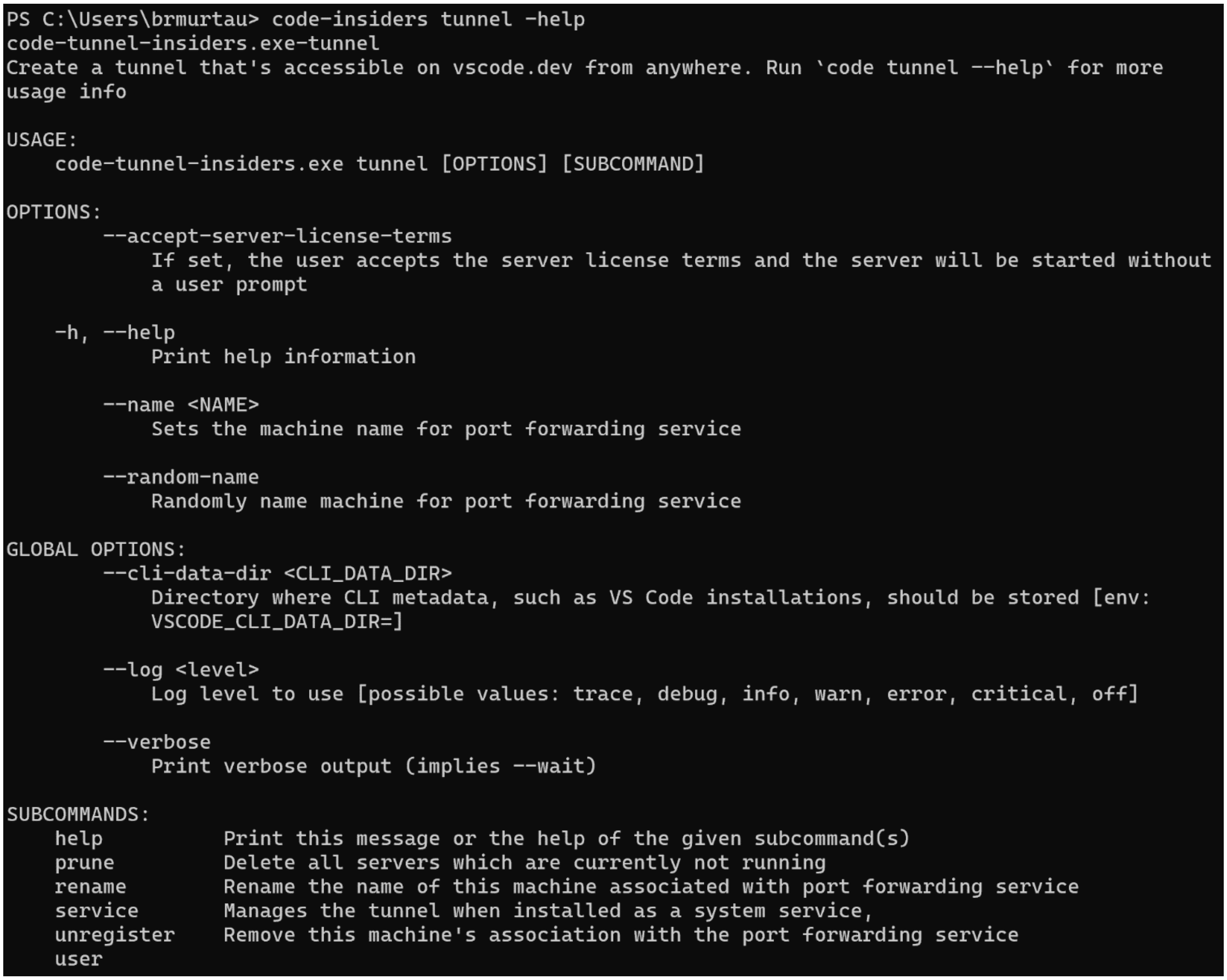
The Visual Studio Code Remote - Tunnels extension lets you connect to a remote machine, like a desktop PC or VM, via a secure tunnel. Tunneling securely transmits data from one network to another. You can then securely connect to that machine from anywhere, without the requirement of SSH.

We've built functionality into the `code` CLI that will initiate tunnels on remote machines. You can run:

```
code tunnel
```

to create a tunnel on your remote machine. You may connect to this machine through a web or desktop VS Code client.

You can review the other tunneling commands by running `code tunnel -help`:



```
PS C:\Users\brmurtau> code-insiders tunnel -help
code-tunnel-insiders.exe-tunnel
Create a tunnel that's accessible on vscode.dev from anywhere. Run `code tunnel --help` for more
usage info

USAGE:
    code-tunnel-insiders.exe tunnel [OPTIONS] [SUBCOMMAND]

OPTIONS:
        --accept-server-license-terms
            If set, the user accepts the server license terms and the server will be started without
            a user prompt

    -h, --help
            Print help information

        --name <NAME>
            Sets the machine name for port forwarding service

        --random-name
            Randomly name machine for port forwarding service

GLOBAL OPTIONS:
        --cli-data-dir <CLI_DATA_DIR>
            Directory where CLI metadata, such as VS Code installations, should be stored [env:
            VSCODE_CLI_DATA_DIR=]

        --log <level>
            Log level to use [possible values: trace, debug, info, warn, error, critical, off]

        --verbose
            Print verbose output (implies --wait)

SUBCOMMANDS:
    help          Print this message or the help of the given subcommand(s)
    prune         Delete all servers which are currently not running
    rename        Rename the name of this machine associated with port forwarding service
    service       Manages the tunnel when installed as a system service,
    unregister    Remove this machine's association with the port forwarding service
    user
```

As you may need to run the CLI on a remote machine that can't install VS Code Desktop, the CLI is also available for standalone install on the [VS Code download page](https://code.visualstudio.com/#alt-downloads).

For more information on Remote Tunnels, you can review the [Remote Tunnels documentation](https://code.visualstudio.com/docs/remote/tunnels).

Opening VS Code with URLs

You can also open projects and files using the platform's URL handling mechanism. Use the following URL formats to:

Open a project

```
vscode://file/{full path to project}/



vscode://file/c:/myProject/
```

Open a file

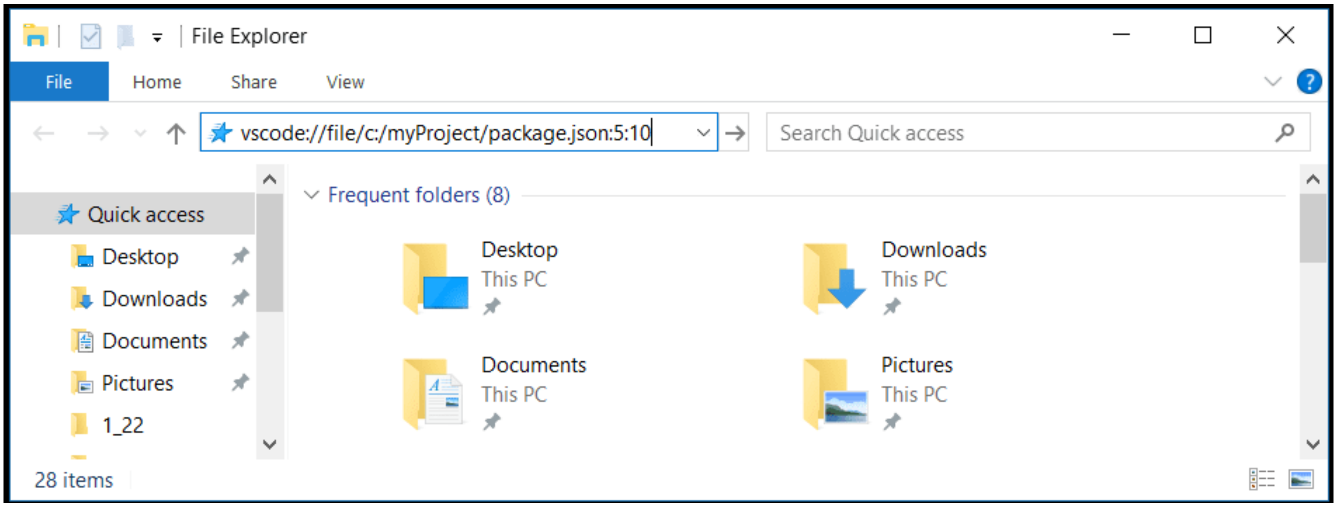
```
vscode://file/{full path to file}
```

```
vscode://file/c:/myProject/package.json
```

Open a file to line and column

```
vscode://file/{full path to file}:line:column


vscode://file/c:/myProject/package.json:5:10
```

You can use the URL in applications such as browsers or file explorers that can parse and redirect the URL. For example, on Windows, you could pass a `vscode://` URL directly to the Windows Explorer or to the command line as `start vscode://{full path to file}`.



vscode://file/c:/myProject/package.json:5:10

Note: If you are using VS Code Insiders builds, the URL prefix is `vscode-insiders://`.

Next steps

Read on to find out about:

- Integrated Terminal - Run command-line tools from inside VS Code.
- Basic Editing - Learn the basics of the VS Code editor.
- Code Navigation - VS Code lets you quickly understand and move through your source code.

Common questions

'code' is not recognized as an internal or external command

Your OS cannot find the VS Code binary `code` on its path. The VS Code Windows and Linux installations should have installed VS Code on your path. Try uninstalling and reinstalling VS Code. If `code` is still not found, consult the platform-specific setup topics for Windows and Linux.

On macOS, you need to manually run the **Shell Command: Install 'code' command in PATH** command (available through the **Command Palette** `Ctrl+Shift+P`). Consult the macOS specific setup topic for details.

How do I get access to a command line (terminal) from within VS Code?

VS Code has an Integrated Terminal where you can run command-line tools from within VS Code.

Can I specify the settings location for VS Code in order to have a portable version?

Not directly through the command line, but VS Code has a Portable Mode, which lets you keep settings and data in the same location as your installation, for example, on a USB drive.

Updates for this guide can be found at:
https://code.visualstudio.com/docs

Updated: July 2023