

目錄

简介	1.1
官方文档	1.2
概览	1.3
快速入门(暂无)	1.4
视频介绍	1.4.1
小技巧	1.4.2
用户接口	1.4.3
主题	1.4.4
设置	1.4.5
快捷键绑定	1.4.6
语言区域	1.4.7
编辑器	1.5
基础	1.5.1
安装	1.5.2
扩展市场	1.5.3
任务	1.5.4
调试	1.5.5
为什么选用VSCode	1.5.6
版本控制	1.5.7
易用性	1.5.8
与时俱进的编辑体验	1.5.9
定制化	1.6
概述	1.6.1
用户和工作空间	1.6.2
快捷键绑定	1.6.3
用户定义代码段	1.6.4
调色板	1.6.5
主题	1.6.6
语言区域	1.6.7
工具	1.7

vse命令行工具	1.7.1
yocode扩展生成器	1.7.2
范例	1.7.3
技术支持	1.8
常见问题	1.8.1
错误代码	1.8.2
如何升级	1.8.3
系统要求	1.8.4
扩展	1.9
概述	1.9.1
范例-hello-world	1.9.2
范例-word-count	1.9.3
范例-language-server	1.9.4
范例-调试器	1.9.5
调试-扩展	1.9.6
安装-扩展	1.9.7
范式-原则	1.9.8
测试-扩展	1.9.9
用我们的方法创造扩展	1.9.10
扩展API	1.10
概述	1.10.1
扩展manifest文件	1.10.2
扩展点	1.10.3
激活事件	1.10.4
vscode-api	1.10.5
vscode-api-命令	1.10.6
api调试	1.10.7
语言	1.11
概述	1.11.1
Javascript	1.11.2
JSON	1.11.3
HTML(暂无)	1.11.4
CSS, Sass and Less	1.11.5
TypeScript	1.11.6

Markdown	1.11.7
C++	1.11.8
Java(暂无)	1.11.9
PHP	1.11.10
Python	1.11.11
Go(暂无)	1.11.12
Dockerfile	1.11.13
T-SQL(暂无)	1.11.14
C#	1.11.15
运行时	1.12
nodejs	1.12.1
ASPnet5	1.12.2
unity	1.12.3
office	1.12.4

Microsoft Visual Studio Code 中文手册

DEMO : <https://jeasonstudio.gitbooks.io/vscode-cn-doc/content/>

Visual Studio Code 是微软推出的跨平台编辑器。它采用经典的VS的UI布局，功能强大，扩展性很强。但是 Visual Studio Code 暂时没有中文手册，对于不太熟悉英文的同学会比较吃力。

本项目的初衷是为想使用或者正在使用 Visual Studio Code 的同学提供一个中文手册，方便大家学习使用这个优秀的工具，提高程序开发效率和质量！

翻译流程

第一阶段

先将 [Visual Studio Code Docs](#) 的内容按现有的目录结构翻译成中文，其中：

- 文章正文内容均放在 `md` 目录下，采用 `md` 格式。
- 文章中所用到的图片资源暂时先放在 `images` 目录下，后续图片资源会统一托管到[七牛云存储](#)
- 图片按照文档的 `主目录-副目录-编号` 的格式命名。

文件命名规则

- 文件名为[Visual Studio Code Docs](#) 对应文章标题（即下面列出的目录）的 翻译名称（原英文名）。所有的空格都用 `-` 代替，注意单词首字母大写。
- 对于下级子页面文档，将其放在以父级文档名称命名的文件夹下面。

例如：<https://code.visualstudio.com/docs/editor/whyvscode> 这篇文章，对应 `editor` 这个文件夹下的 `WhyVsCode.md` 文件。

第二阶段

根据翻译文档，制作成类似在线手册或者与官方文档类似的网站，方便大家参阅。

参与项目

欢迎你参与翻译本项目，在翻译的过程中，可以锻炼你的英语能力和 Visual Studio Code 的实际应用能力，同时还为他人提供方便，何乐而不为？

一个个 commit 堆积起来就是一个了不起的 repo，欢迎你 Fork 并提交 Pull Request 或者 Issue，哪怕是改正一个错别字、修正一个病句，我们都会很高兴。

参与方法和步骤如下：

- 登录 <https://github.com>
- Fork `git@github.com:CN-VScode-Docs/CN-VScode-Docs.git` 或者点仓库地址[CN-VScode-Docs](#)
- 创建您的特性分支 (`git checkout -b new-feature`)
- 提交您的改动 (`git commit -m 'Added some features or fixed a bug or change a text'`)
- 将您的改动记录提交到远程 git 仓库 (`git push origin new-feature`)
- 然后到 github 网站的该 git 远程仓库的 new-feature 分支下发起 Pull Request

如果你有任何疑问或者建议、技巧，欢迎提出Issues，大家一起交流。

官方说明文档

- 仓库连接[vscode-docs](#)

正在翻译文章+作者

- AllOfIt + Jeason

项目翻译目录

- [Overview](#)
- EDITOR
 - [Setup](#)
 - [The Basics](#)
 - [Extension Marketplace](#)
 - [Editing Evolved](#)
 - [Version Control](#)
 - [Debugging](#)

- [Tasks](#)
- [Accessibility](#)
- [Why Vs Code](#)
- CUSTOMIZATION
 - [Overview](#)
 - [User and Workspace Settings](#)
 - [Key Bindings](#)
 - [Snippets](#)
 - [Colorizer](#)
 - [Themes](#)
 - [Display Language](#)
- LANGUAGES
 - [Overview](#)
 - [JavaScript](#)
 - [C#](#)
 - [C++](#)
 - [JSON](#)
 - [HTML](#)
 - [PHP](#)
 - [python](#)
 - [Markdown](#)
 - [TypeScript](#)
 - [CSS, Sass and Less](#)
 - [Dockerfile](#)
- RUNTIMES
 - [Node.js](#)
 - [ASP.NET Core](#)
 - [Unity](#)
 - [Office](#)
- EXTENSIONS
 - [Overview](#)
 - [Example - Hello World](#)
 - [Example - Word Count](#)
 - [Example - Language Server](#)
 - [Example - Debuggers](#)
 - [Principles and Patterns](#)
 - [Running and Debugging Your Extension](#)
 - [Installing Extensions](#)

- [Testing Extension](#)
- [\[Our Approach\]\(https://code.visualstudio.com/docs/extensions/our-approach\)](#)
- EXTENSIBILITY REFERENCE
 - [Overview](#)
 - [Extension Manifest](#)
 - [Contribution Points](#)
 - [Activation Events](#)
 - [API vscode namespace](#)
 - [API Complex Commands](#)
 - [API Debugging](#)
- TOOLS
 - [Publishing Tool](#)
 - [Extension Generator](#)
 - [Samples](#)

(翻译完成的，请使用删除线将对应划去,像下面这样)

* ~~[Overview](https://code.visualstudio.com/docs)~~

贡献者（按参与时间排序）

- Jeason
- swizard
- heshenghuan
- Alexi.F
- jinyutao
- yuxuefeng
- chenxinlong
- Cherry Mill Wong
- bjrxyz
- avaicode
- Fallenwood
- ickall
- Albert C.
- Saier
- iskcal
- No.20
- Sophia Woo

- [distantmars](#)
- [mrkou47](#)
- [ichengde](#)
- [bee0060](#)
- [nanci](#)

(Fork 之后自行添加到最后)

开源协议

- [MIT](#)

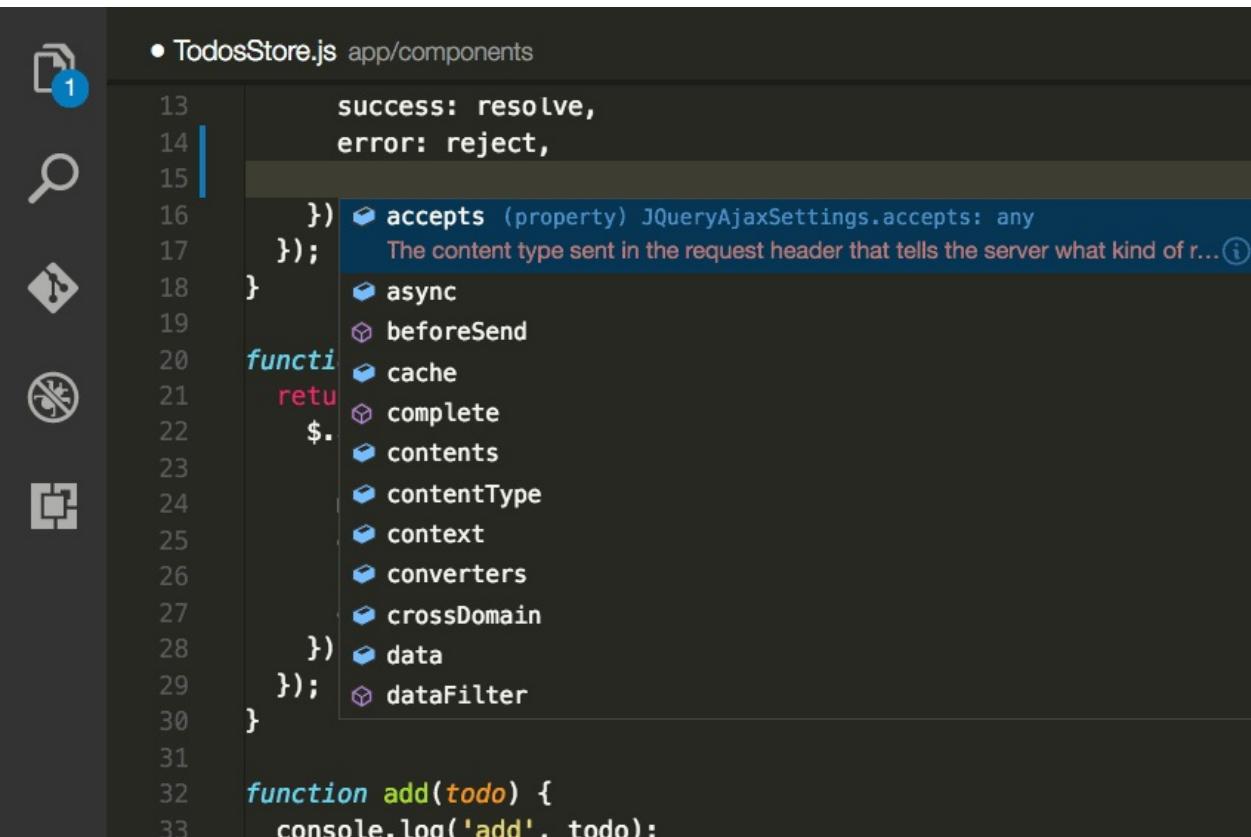
开始 (Getting Started)

Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, OS X and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (C++, C#, Python, PHP) and runtimes.

Visual Studio Code 是一个运行在桌面上，并且可用于Windows，Mac OS X和Linux平台的轻量级且功能强大的源代码编辑器。它配备了内置的JavaScript的，TypeScript和Node.js的支持，并具有其他语言（C ++，C #，Python和PHP）的扩展以及一个丰富的生态系统。

Visual Studio Code的应用 (Visual Studio Code in Action)

智能代码提示和自动补全 (Intelligent Code Completion)



The screenshot shows a code editor window for a file named 'TodosStore.js' located in 'app/components'. The code is part of a function definition, specifically handling an 'error' callback for an Ajax request. A tooltip is displayed over the word 'reject', listing various jQuery method completions such as 'accepts', 'async', 'beforeSend', 'cache', 'complete', 'contents', 'contentType', 'context', 'converters', 'crossDomain', 'data', and 'dataFilter'. The tooltip also includes a brief description of the 'accepts' method: 'The content type sent in the request header that tells the server what kind of r...'. The code editor interface includes a left sidebar with icons for file, search, and refresh, and a status bar at the bottom.

```

• TodosStore.js app/components

13     success: resolve,
14     error: reject,
15   });
16   });
17 });
18 }
19
20 function(todo) {
21   return $.ajax({
22     url: '/api/todos',
23     type: 'POST',
24     data: todo,
25     success: resolve,
26     error: reject
27   });
28 });
29 });
30 }
31
32 function add(todo) {
33   console.log('add', todo);

```

Code smarter with IntelliSense - completions for variables, methods, and imported modules.

更聪明的代码智能感知 - 完备的变量，方法和导入模块。

简化调试过程（Streamlined Debugging）

The screenshot shows the VS Code interface with the 'server.js' file open. The code is as follows:

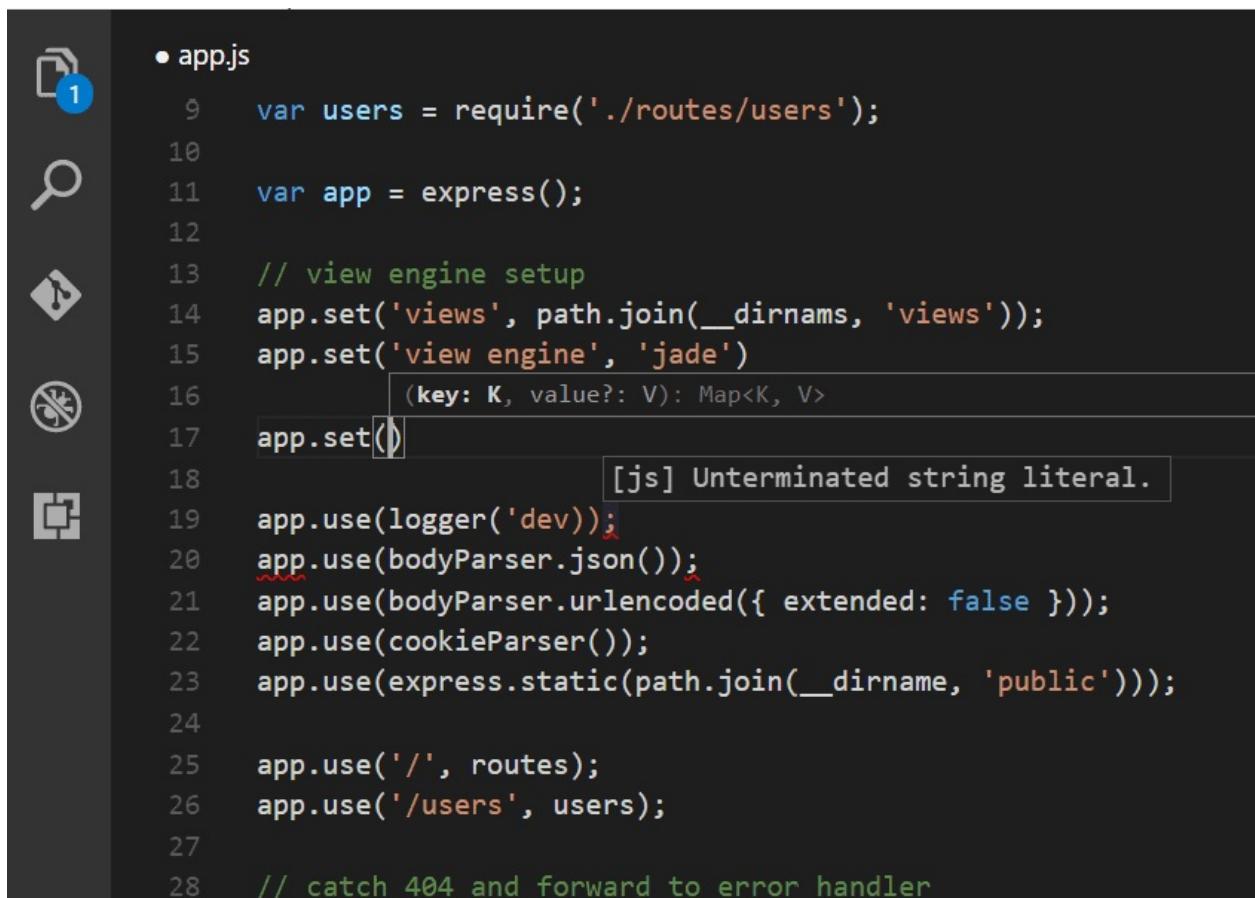
```
37
38     database.del(id, function(todos) {
39         res.send(todos);
40         next();
41     });
42 });
43
44 // Start listening
45 var PORT = 3001;
46 server.listen(PORT, function() {
47     console.log('listening at %s', PORT);
48 });
49
```

The line `server.listen(PORT, function() {` is highlighted in yellow, indicating it is the current line of execution. The 'Variables' sidebar shows local variables like `__dirname`, `__filename`, `bodyParser`, `database`, `exports`, `express`, `module`, and `PORT`. The 'Call Stack' sidebar shows the stack trace: '(anonymous function)', 'Module._compile', 'Module._extensions.js', and 'Module.load'. The 'Debug Console' shows the command `node --debug-brk=23449 server/server.js` and the message 'Debugger listening on port 23449'.

Print debugging is a thing of the past. Debug in VS Code with your terminal tools.

打印 Debug 结果是曾经做的事。在 VS CODE 中将用终端工具进行调试。

快速，强大的编辑能力（Fast, Powerful Editing）



```
• app.js
  9  var users = require('./routes/users');
10
11  var app = express();
12
13  // view engine setup
14  app.set('views', path.join(__dirname, 'views'));
15  app.set('view engine', 'jade')
16      (key: K, value?: V): Map<K, V>
17  app.set()
18          [js] Unterminated string literal.
19  app.use(logger('dev'));
20  app.use(bodyParser.json());
21  app.use(bodyParser.urlencoded({ extended: false }));
22  app.use(cookieParser());
23  app.use(express.static(path.join(__dirname, 'public'))));
24
25  app.use('/', routes);
26  app.use('/users', users);
27
28  // catch 404 and forward to error handler
```

Linting, multi-cursor editing, parameter hints, and other powerful editing features.

静态源代码检查，多光标编辑，参数提示，以及其他强大的编辑功能。

代码导航和重构（Code Navigation and Refactoring）

```
TodosStore.js app/components
1 import _ from 'lodash';
2 import $ from 'jquery';

index.d.ts typings/browser/ambient/jquery - 3 definitions
3198 */
3199     queue(queueName: string, newQueue: Function[]): JQuery;
3200 /**
3201 * Manipulate the queue of functions to be executed, once for
3202 *
3203 * @param queueName A string containing the name of the queue
3204 * @param callback The new function to add to the queue, with
3205 */
3206     queue(queueName: string, callback: Function): JQuery;
3207 }
3208 declare module "jquery" {
3209     export = $;
3210 }
3211 declare var jQuery: JQueryStatic;
3212 declare var $: JQueryStatic;
```

Browse your source code quickly using peek and navigate to definition.

快速浏览你的源代码使用 **peek** 并导航至定义。

对于产品的**Git支持 (In-Product Git Support)**

```

NewTodo.js app/components - Changes on working tree
25
26 yDown(event) {
27 if (event.keyCode === RETURN_KEY_CODE)
28   let text = event.target.value.trim()
29   if (text === '') {
30     return;
31   }
32   TodosStore.add({
33     text: event.target.value.trim()
34     timestamp: new Date().toLocaleString()
35   });
36
37   // clear input
38   event.target.value = '';
39 }
40
41
42 er() {
43 return (
44   <div style={this.styles.spacing} >
45     <p style={this.styles.prompt}>W

```

```

25
26 yDown(event) {
27 if (event.keyCode === RETURN_KEY_CODE)
28   let text = event.target.value.trim()
29   if (text === '') {
30     return;
31   }
32   TodosStore.add(text);
33
34   // clear input
35   event.target.value =
36 }
37
38
39 er() {
40 return (
41   <div style={this.st
42     <p style={this.

```

Speed up your release cycle with Git support inside your editor.

在您的编辑器中加入对 Git 的支持以加快您的研发效率。

常用扩展 (Top Extensions)

Enable additional languages, themes, debuggers, commands, and more. VS Code's growing community shares their secret sauce to improve your workflow.

启用其他语言，主题，调试器，命令等扩展程序。VS CODE 日益增长的社区内有大家分享的私人秘诀，以提高您的工作效率。

第一步 (First Steps)

To get the most out of Visual Studio Code, start by reviewing a few introductory topics:

为了更充分的学习Visual Studio Code，我们将从以下几个方面展开：

Setup - Install VS Code for your platform and configure the tool set for your development needs.

安装 - 为您的平台安装VS CODE并且配置您开发所需求的工具集。

The Basics - Introduction to the basic UI, commands, and features of the VS Code editor.

基础知识 - 介绍基本的UI，命令和VS代码编辑器的功能。

Settings - Customize VS Code for how you like to work.

设置 - 为你喜欢的工作方式自定义 VS CODE。

Languages - Learn about VS Code's support for your favorite programming languages.

语言 - 了解VS CODE对您最喜爱的编程语言的支持。

Node.js - This tutorial gets you quickly running and debugging a Node.js web app.

Node.js - 本教程让你快速运行和调试Node.js编写的web应用程序。

Why VS Code? - Read about the design philosophy and architecture of VS Code.

为什么选择VS CODE？ - 阅读VS CODE的设计理念和架构。

下载 (Downloads)

Download VS Code - Quickly find the appropriate install for your platform (Windows, OS X and Linux).

[下载VS CODE](#) - 快速找到合适你平台的安装程序（Windows，Mac OS X和Linux）。

隐私 (Privacy)

By default, VS Code auto-updates to new versions, and collects usage data and crash report information. You may opt out of these defaults by disabling them as instructed below:

默认情况下，VS CODE自动更新到新版本，并收集使用数据和崩溃报告信息。您可以通过禁用它们更改这些默认值，如下：

How do I disable auto update?

[如何禁用自动更新？](#)

How do I disable crash reporting?

[如何禁用崩溃报告？](#)

How do I disable usage reporting?

[如何禁用使用情况报告？](#)

Visual Studio Code Tips and Tricks

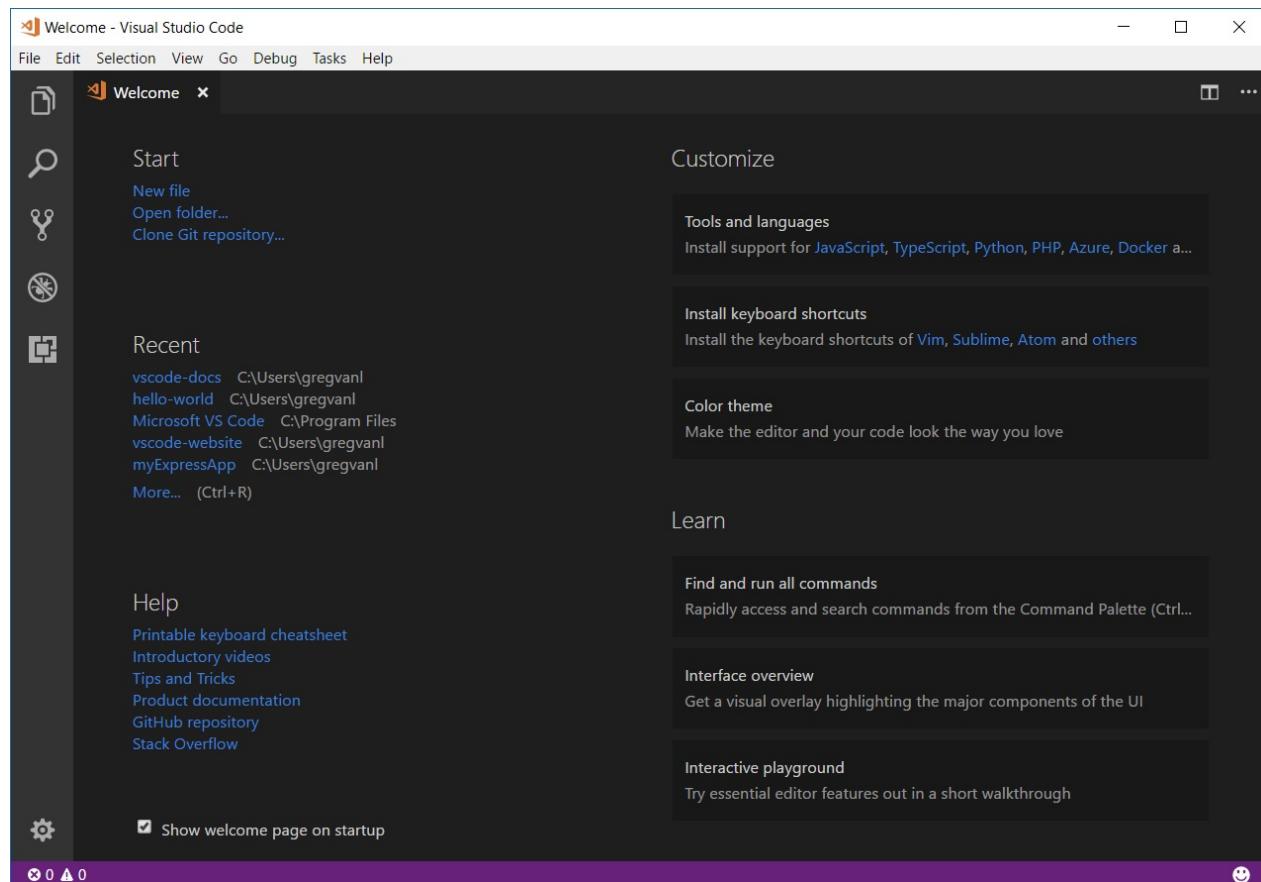
"Tips and Tricks" lets you jump right in and learn how to be productive with Visual Studio Code. You'll become familiar with its powerful editing, code intelligence, and source code control features and learn useful keyboard shortcuts. This topic goes pretty fast and provides a broad overview, so be sure to look at the other in-depth topics in [Getting Started](#) and the [User Guide](#) to learn more.

If you don't have Visual Studio Code installed, go to the [Download](#) page. You can find platform specific setup instructions at [Running VS Code on Linux](#), [macOS](#), and [Windows](#).

Basics

Getting Started

Open the **Welcome** page to get started with the basics of VS Code. **Help > Welcome**.



Includes the **Interactive Playground**.

Interactive Playground - Visual Studio Code

File Edit Selection View Go Debug Tasks Help

Welcome Interactive Playground

Multi-Cursor Editing

Using multiple cursors allows you to edit multiple parts of the document at once, greatly improving your productivity. Try the following actions in the code block below:

1. Box Selection - press any combination of **Ctrl+Shift+Alt+DownArrow**, **Ctrl+Shift+Alt+RightArrow**, **Ctrl+Shift+Alt+UpArrow**, **Ctrl+Shift+Alt+LeftArrow** to select a block of text, you can also press **Shift+Alt** while selecting text with the mouse.
2. Add a cursor - press **Ctrl+Alt+UpArrow** or **Ctrl+Alt+DownArrow** to add a new cursor above or below, you can also use your mouse with **Alt+Click** to add a cursor anywhere.
3. Create cursors on all occurrences of a string - select one instance of a string e.g. `background-color` and press **Ctrl+Shift+L**. Now you can replace all instances by simply typing.

That is the tip of the iceberg for multi-cursor editing have a look at the [selection menu](#) and our handy [keyboard reference guide](#) for additional actions.

```

1  #p1 {background-color: red; /*.red*/
2  #p2 {background-color: green; /*.green*/
3  #p3 {background-color: blue; /*.blue*/

```

CSS Tip: you may have noticed in the example above for CSS we also provide color swatches inline, additionally if you hover over an element such as `#p1` we will show how this is represented in HTML. A simple example of some language specific editor features.

IntelliSense

Visual Studio Code comes with powerful IntelliSense for JavaScript and TypeScript pre-installed. In the below example, position the text cursor in front of the error underline, right after the dot and press **Ctrl+Space** to invoke IntelliSense. Notice how the suggestion comes from the Request API.

```

1  var express = require('express');
2  var app = express();
3
4  app.get('/', function (req, res) {

```

Command Palette

Access all available commands based on your current context.

Keyboard Shortcut: `kb(workbench.action.showCommands)`

App.js - sports-trivia

App.js keybindings.json

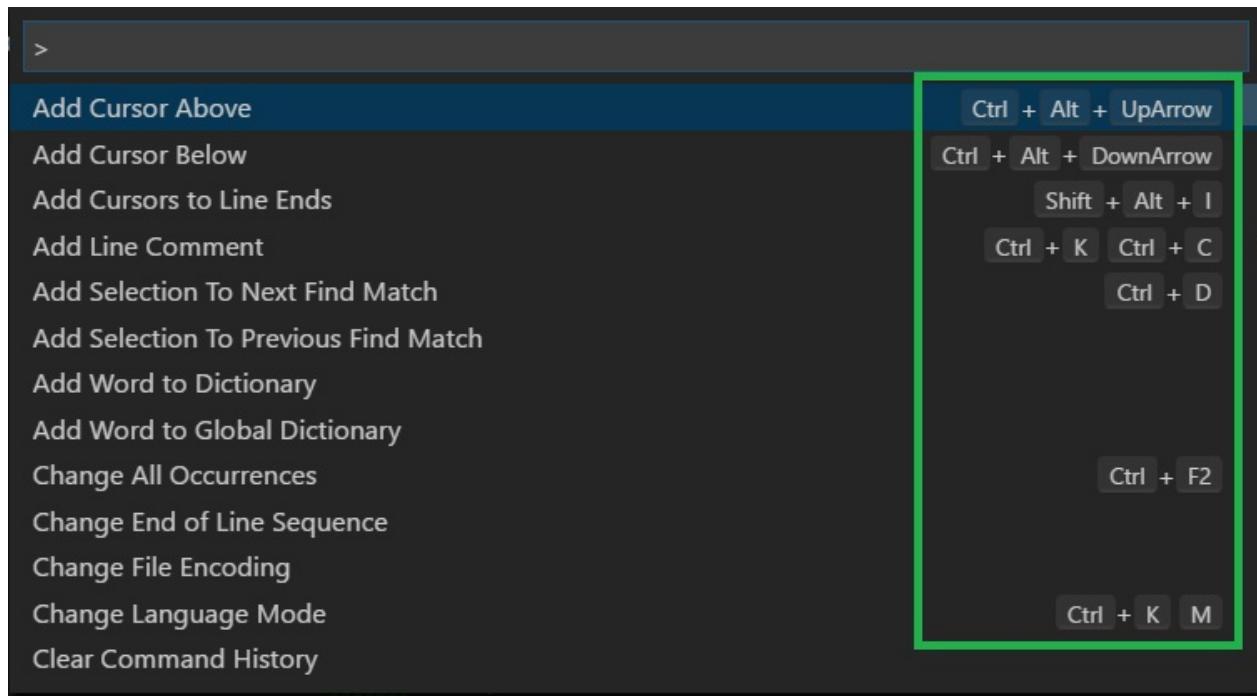
```

1 import React from 'react';
2 import {deepOrange500} from 'material-ui/styles/colors';
3 import AppBar from 'material-ui/AppBar';
4 import getMuiTheme from 'material-ui/styles/getMuiTheme';
5 import MuiThemeProvider from 'material-ui/styles/MuiThemeProvider';
6
7 import QuestionBar from './QuestionBar';
8
9 const muiTheme = getMuiTheme({
10   palette: {
11     accent1Color: deepOrange500,
12   },
13 });
14
15 export default class App extends React.Component {
16   constructor(props, context) {
17     super(props, context);
18
19     this.state = {};
20   }
21

```

Default keyboard shortcuts

All of the commands are in the **Command Palette** with the associated key binding (if it exists). If you forget a keyboard shortcut, use the **Command Palette** to help you out.



Keyboard Reference Sheets

Download the keyboard shortcut reference sheet for your platform ([macOS](#), [Windows](#), [Linux](#)).

 Visual Studio Code	
Keyboard shortcuts for macOS	
General	
⌘F, F1	Show Command Palette
⌘P	Quick Open, Go to File...
⌘N	New window-instance
⌘W	Close window-instance
⌘S	User Settings
⌘K ⌘S	Keyboard Shortcuts
Basic editing	
⌘X	Cut line (empty selection)
⌘C	Copy line (empty selection)
⌃/ ⌄↑	Move line down/up
⌃/ ⌄↓	Move line up/down
⌘K	Delete line
⌘Enter / ⌄⌘Enter	Insert line below/above
⌃[/ ⌄]	Jump to matching bracket
⌃/ ⌄[Indent/outdent line
Home / End	Go to beginning/end of line
⌃Home / ⌄End	Go to beginning/end of file
⌃PgUp / ⌄PgDn	Scroll line up/down
⌃PgUp / ⌄PgDn	Scroll page up/down
⌃⌘[/ ⌄⌘]	Fold/unfold region
⌃⌘[/ ⌄⌘]	Fold/unfold all subregions
⌃⌘0 / ⌄⌘J	Fold/unfold all regions
⌃⌘C	Add line comment
⌃⌘U	Remove line comment
⌃/	Toggle line comment
⌃A	Toggle block comment
⌃Z	Toggle word wrap
Multi-cursor and selection	
⌃+ click	Insert cursor
⌃⌘↑	Insert cursor above
⌃⌘↓	Insert cursor below
⌘U	Undo last cursor operation
⌃cl	Insert cursor at end of each line selected
⌃I	Select current line
⌃⌘L	Select all occurrences of current selection
⌃F2	Select all occurrences of current word
⌃⌘← / ←	Expand / shrink selection
⌃- + drag mouse	Column (box) selection
⌃- ⌄↑ / ↑	Column (box) selection up/down
⌃- ⌄← / ←	Column (box) selection left/right
⌃- PgUp	Column (box) selection page up
⌃- PgDn	Column (box) selection page down
Search and replace	
⌃F	Find
⌃⌘F	Replace
⌃G / ⌄⌘G	Find next/previous
⌃Enter	Select all occurrences of Find match
⌃D	Add selection to next Find match
⌃K ⌄D	Move last selection to next Find match
Rich languages editing	
⌃Space	Trigger suggestion
⌃⌘Space	Trigger parameter hints
⌃- F	Format document
⌃K ⌄F	Format selection
F12	Go to Definition
⌃F12	Peek Definition
⌃K F12	Open Definition to the side
⌃.	Quick Fix
⌃F12	Show References
F2	Rename Symbol
⌃K ⌄X	Trim trailing whitespace
⌃K M	Change file language
Navigation	
⌃T	Show all Symbols
⌃G	Go to Line...
⌃P	Go to File...
⌃O	Go to Symbol...
⌃M	Show Problems panel
F8 / ⌄F8	Go to next/previous error or warning
⌃jTab	Navigate editor group history
⌃- / ⌄-	Go back/forward
⌃M	Toggle Tab moves focus
Editor management	
⌃W	Close editor
⌃K F	Close folder
⌃\	Split editor
⌃1 / ⌄⌘3	Focus into 1st, 2nd, 3rd editor group
⌃K ⌄- / ⌄K ⌄→	Focus into previous/next editor group
⌃K ⌄← / ⌄K ⌄→	Move editor left/right
⌃K - / ⌄K -	Move active editor group
File management	
⌃N	New File
⌃O	Open File...
⌃S	Save
⌃S	Save As...
⌃S	Save All
⌃W	Close
⌃K ⌄W	Close All
Display	
⌃F	Reopen closed editor
⌃K Enter	Keep preview mode editor open
⌃Tab / ⌄Tab	Open next / previous
⌃KP	Copy path of active file
⌃KR	Reveal active file in Explorer
⌃KO	Show active file in new window-instance
Debug	
F8	Toggle breakpoint
F5	Start/Continue
F11 / ⌄F11	Step into/ out
F10	Step over
⌃F5	Stop
⌃K ⌄I	Show hover
Integrated terminal	
⌃`	Show integrated terminal
⌃j`	Create new terminal
⌃C	Copy selection
unassigned	Paste into active terminal
⌃↑ / ↓	Scroll up/down
PgUp / PgDn	Scroll page up/down
⌃Home / End	Scroll to top/bottom
Other operating systems' keyboard shortcuts and additional unassigned shortcuts available at aka.ms/vscodekeybindings	

Quick Open

Quickly open files.

Keyboard Shortcut: kb(workbench.action.quickOpen)



Tip: Type `kbstyle(?)` to view help suggestions.

Navigate between recently opened files

Repeat the **Quick Open** keyboard shortcut to cycle quickly between recently opened files.

Open multiple files from Quick Open

You can open multiple files from **Quick Open** by pressing the Right arrow key. This will open the currently selected file in the background and you can continue selecting files from **Quick Open**.

Command line

VS Code has a powerful command line interface (CLI) to help you customize the editor launch your specific scenarios.

Make sure the VS Code binary is on your path so you can simply type 'code' to launch VS Code. See the platform specific setup topics if VS Code is added to your environment path during installation ([Running VS Code on Linux, macOS, Windows](#)).

```
# open code with current directory
code .

# open the current directory in the most recently used code window
code -r .

# create a new window
code -n

# change the language
code --locale=es

# open diff editor
code --diff <file1> <file2>

# open file at specific line and column <file:line[:character]>
code --goto package.json:10:5

# see help options
code --help

# disable all extensions
code --disable-extensions .
```

.vscode folder

Workspace specific files are in a `.vscode` folder at the root. For example, `tasks.json` for the Task Runner and `launch.json` for the debugger.

Status Bar

Errors and Warnings

Keyboard Shortcut: `kb(workbench.actions.view.problems)`

Quickly jump to errors and warnings in the project.

Cycle through errors with `kb(editor.action.marker.next)` or `kb(editor.action.marker.prev)`

The screenshot shows the VS Code interface. In the center is a code editor window titled "database.js" containing the following code:

```

9     return new Promise(function(resolve, reject) {
10    fs.readFile(DATA, function(err, data) {
11      if (err) {
12        reject(err);
13      } else {
14        resolve(JSON.parse(data));
15      }
16    });
17  });
18}
19

```

Below the code editor is a "Problems" panel with the title "2 ERRORS". It lists two ESLint errors:

- [eslint] Parsing error: Unexpected token) (16, 10)
- [js] Declaration or statement expected. (16, 10)

You can filter problems by type ('errors', 'warnings') or text matching.

Change language mode

Keyboard Shortcut: `kb(workbench.action.editor.changeLanguageMode)`

The screenshot shows the VS Code interface with the Explorer sidebar open. The "WORKING FILES" section is expanded, showing the file structure:

- NewNote.js** (selected)
- App.js
- .eslintrc
- package.json
- NewNote.js
- TODO**
 - dist
 - node_modules
 - src
 - components
 - App
 - App.js
 - App.scss
 - package.json
 - NewNote
 - NewNote.js
 - package.json
 - Notes
 - Notes.js
 - package.json
- Notes
- README.md
- client.js
- index.html
- variables.scss

If you want to persist the new language mode for that file type, you can use the **Configure File Association for ...** command to associate the current file extension with an installed language.

Customization

There are many things you can do to customize VS Code.

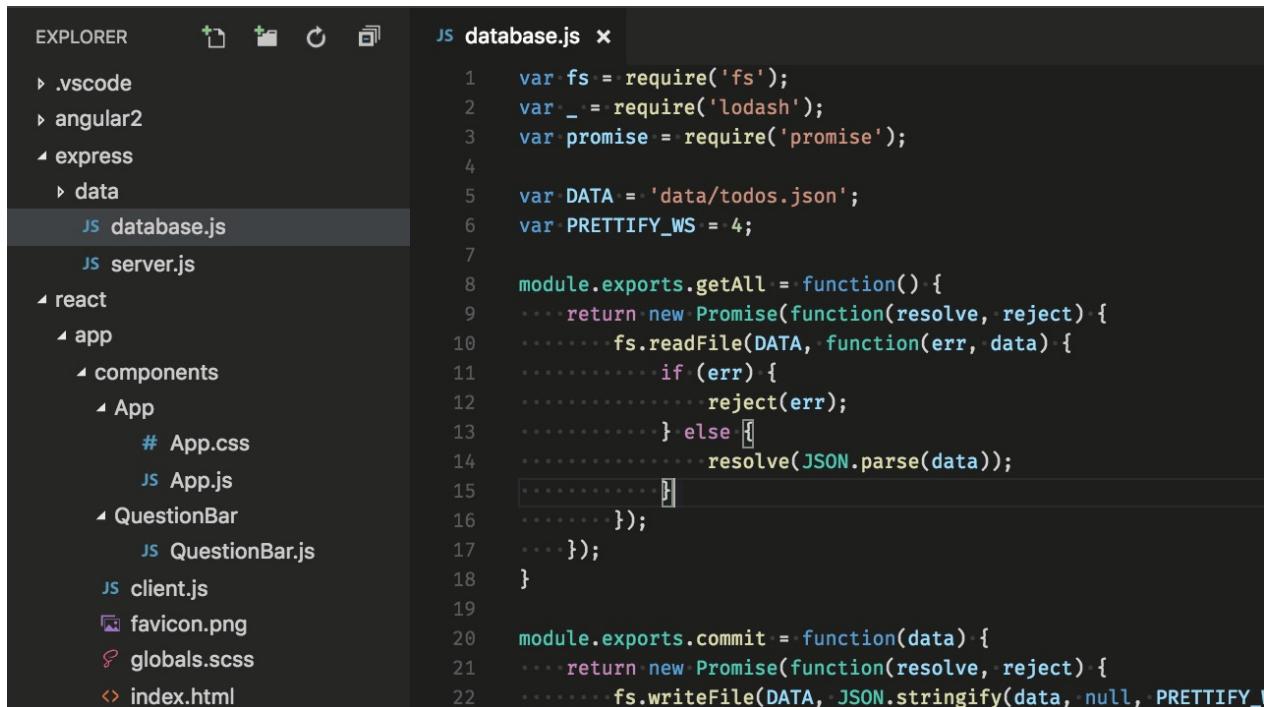
- Change your theme
- Change your keyboard shortcuts
- Tune your settings
- Add JSON validation
- Create snippets
- Install extensions

Check out the full [Settings](#) documentation.

Change your theme

Keyboard Shortcut: `kb(workbench.action.selectTheme)`

You can install more themes from the extension Marketplace.



```

EXPLORER      + - 📁 ⚙️ 🔍 🖼
▸ .vscode
▸ angular2
▸ express
  ▾ data
    JS database.js
    JS server.js
▸ react
  ▾ app
    ▾ components
      ▾ App
        # App.css
        JS App.js
    ▾ QuestionBar
      JS QuestionBar.js
    JS client.js
    🖼 favicon.png
    ❀ globals.scss
    <> index.html

JS database.js ✘
1  var fs = require('fs');
2  var _ = require('lodash');
3  var promise = require('promise');
4
5  var DATA = 'data/todos.json';
6  var PRETTIFY_WS = 4;
7
8  module.exports.getAll = function() {
9    return new Promise(function(resolve, reject) {
10      fs.readFile(DATA, function(err, data) {
11        if (err) {
12          reject(err);
13        } else {
14          resolve(JSON.parse(data));
15        }
16      });
17    });
18  }
19
20 module.exports.commit = function(data) {
21   return new Promise(function(resolve, reject) {
22     fs.writeFile(DATA, JSON.stringify(data, null, PRETTIFY_WS));
23   });
}

```

Additionally, you can install and change your File Icon themes.

```

1  var fs = require('fs');
2  var _ = require('lodash');
3  var promise = require('promise');
4
5  var DATA = 'data/todos.json';
6  var PRETTIFY_WS = 4;
7
8  module.exports.getAll = function() {
9      return new Promise(function(resolve, reject) {
10         fs.readFile(DATA, function(err, data) {
11             if (err) {
12                 reject(err);
13             } else {
14                 resolve(JSON.parse(data));
15             }
16         });
17     }
18 }
19
20 module.exports.commit = function(data) {
21     return new Promise(function(resolve, reject) {
22         fs.writeFile(DATA, JSON.stringify(data, null, PRETTIFY_WS), function(err) {
23             if (err) {
24                 reject(err);
25             } else {
26                 resolve();
27             }
28         });
29     });
30 }

```

Keymaps

Are you used to keyboard shortcuts from another editor? You can install a Keymap extension that brings the keyboard shortcuts from your favorite editor to VS Code. Go to **Preferences > Keymap Extensions** to see the current list on the [Marketplace](#). Some of the more popular ones:

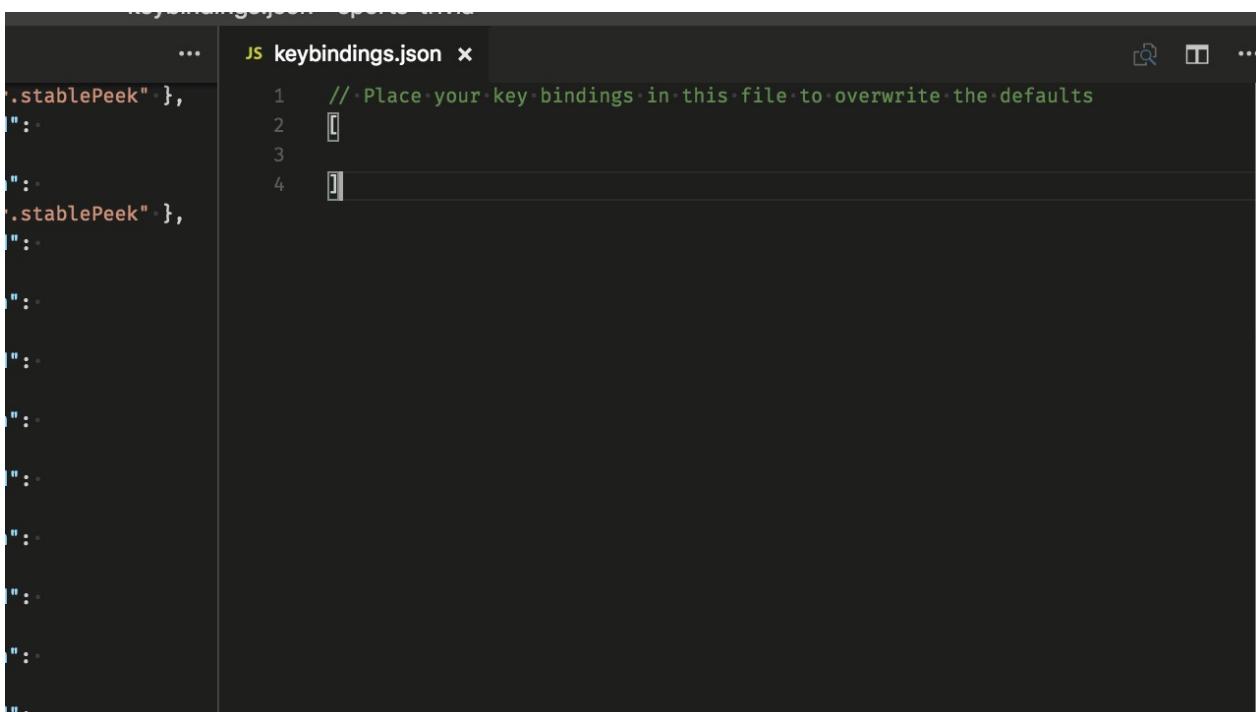
- [Vim](#)
- [Sublime Text Keymap](#)
- [Emacs Keymap](#)
- [Atom Keymap](#)

Customize your keyboard shortcuts

Keyboard Shortcut: `kb(workbench.action.openGlobalKeybindings)`

Command	Keybinding	Source	When
Add Cursor Above	Ctrl + Alt + UpArrow	Default	editorTextFocus
Add Cursor Below	Ctrl + Alt + DownArrow	Default	editorTextFocus
Add Cursors to Line Ends	Shift + Alt + I	Default	editorTextFocus
Add Line Comment	Ctrl + K Ctrl + C	Default	editorTextFocus && !editorReadOnly
Add Selection To Next Find Match	Ctrl + D	Default	editorFocus
Change All Occurrences	Ctrl + F2	Default	editorTextFocus && !editorReadOnly
Change Language Mode	Ctrl + K M	Default	—
Clear	Ctrl + K	Default	terminalFocus

You can search for shortcuts and add your own keybindings to the `keybindings.json` file.



```
... JS keybindings.json ...
1 // Place your key bindings in this file to overwrite the defaults
2 []
3
4 ]]
```

See more in [Key Bindings for Visual Studio Code](#).

Tune your settings

Open User Settings `settings.json`

Keyboard Shortcut: `kb(workbench.action.openGlobalSettings)`

Format on paste

```
"editor.formatOnPaste": true
```

Change the font size

```
"editor.fontSize": 18
```

Change the zoom level

```
"window.zoomLevel": 5
```

Font ligatures

```
"editor.fontFamily": "Fira Code",
"editor.fontLigatures": true
```

Tip: You will need to have a font installed that supports font ligatures. [FiraCode](#) is a popular font on the VS Code team.

```
export default (n) => {
  if (n >= 1000000) {
    return -1;
  }

  var fibs = [];
  fibs[0] = 0;
  fibs[1] = 1;

  for (var i = 2; i <= n; i++) {
    fibs[i] = fibs[i-1] + fibs[i-2];
  }

  return fibs[n];
}
```

Auto Save

```
"files.autoSave": "afterDelay"
```

You can also toggle Auto Save from the top-level menu with the **File > Auto Save**.

Format on save

```
"editor.formatOnSave": true,
```

Change the size of Tab characters

```
"editor.tabSize": 4
```

Spaces or Tabs

```
"editor.insertSpaces": true
```

Render whitespace

```
"editor.renderWhitespace": "all"
```

Ignore files / folders

Removes these files / folders from your editor window.

```
"files.exclude": {  
    "somefolder/": true,  
    "somefile": true  
}
```

Remove these files / folders from search results.

```
"search.exclude": {  
    "someFolder/": true,  
    "somefile": true  
}
```

And many, many [other customizations](#).

Language specific settings

For those settings you only want for specific languages, you can scope the settings by the language identifier. You can find a list of commonly used language ids in the [Language Identifiers](#) reference.

```
"[languageid)": {  
}
```

Tip: You can also create language specific settings with the **Configure Language Specific Settings...** command.

```
// Place your settings in this file to overwrite the default settings
{
  "[typescript]": {
    "editor.fontSize": 12,
    "files.autoSave": "off",
    "editor.fontFamily": "Menlo, Monaco, 'Courier New', monospace",
    "editor.tabSize": 4
  }
}
```

The screenshot shows the VS Code interface with the settings.json file open. A tooltip is displayed over the code editor, listing various editor-related configuration options such as diffEditor.ignoreTrimWhitespace, diffEditor.renderIndicators, and others. The status bar at the bottom indicates the file is master*, tab size is 4, and the JSON file is being edited.

Add JSON validation

Enabled by default for many file types. Create your own schema and validation in `settings.json`

```
"json.schemas": [
  {
    "fileMatch": [
      "/bower.json"
    ],
    "url": "http://json.schemastore.org/bower"
  }
]
```

or for a schema defined in your workspace

```
"json.schemas": [
  {
    "fileMatch": [
      "/foo.json"
    ],
    "url": "./myschema.json"
  }
]
```

or a custom schema

```
"json.schemas": [
  {
    "fileMatch": [
      "./myconfig"
    ],
    "schema": {
      "type": "object",
      "properties": {
        "name" : {
          "type": "string",
          "description": "The name of the entry"
        }
      }
    }
  },
]
```

See more in the [JSON](#) documentation.

Extensions

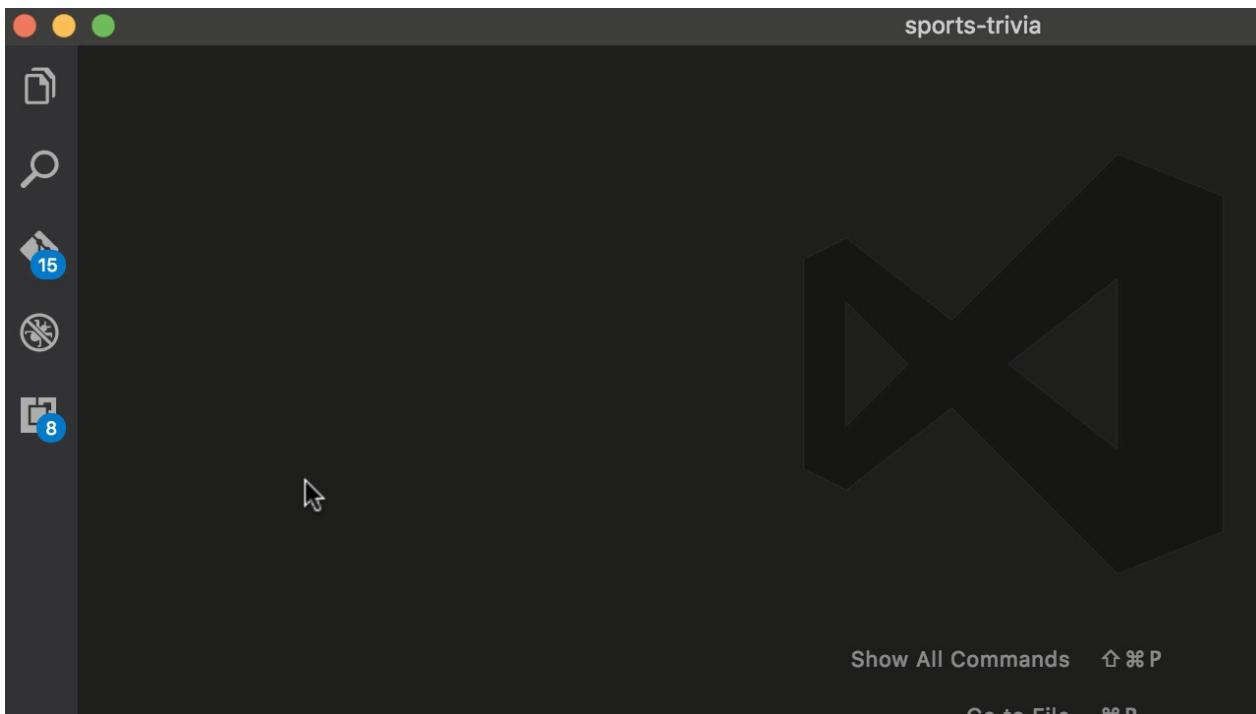
Keyboard Shortcut: `kb(workbench.view.extensions)`

Find extensions

1. In the VS Code [Marketplace](#).
2. Search inside VS Code in the **Extensions** view.
3. View extension recommendations
4. Community curated extension lists, such as [awesome-vscode](#).

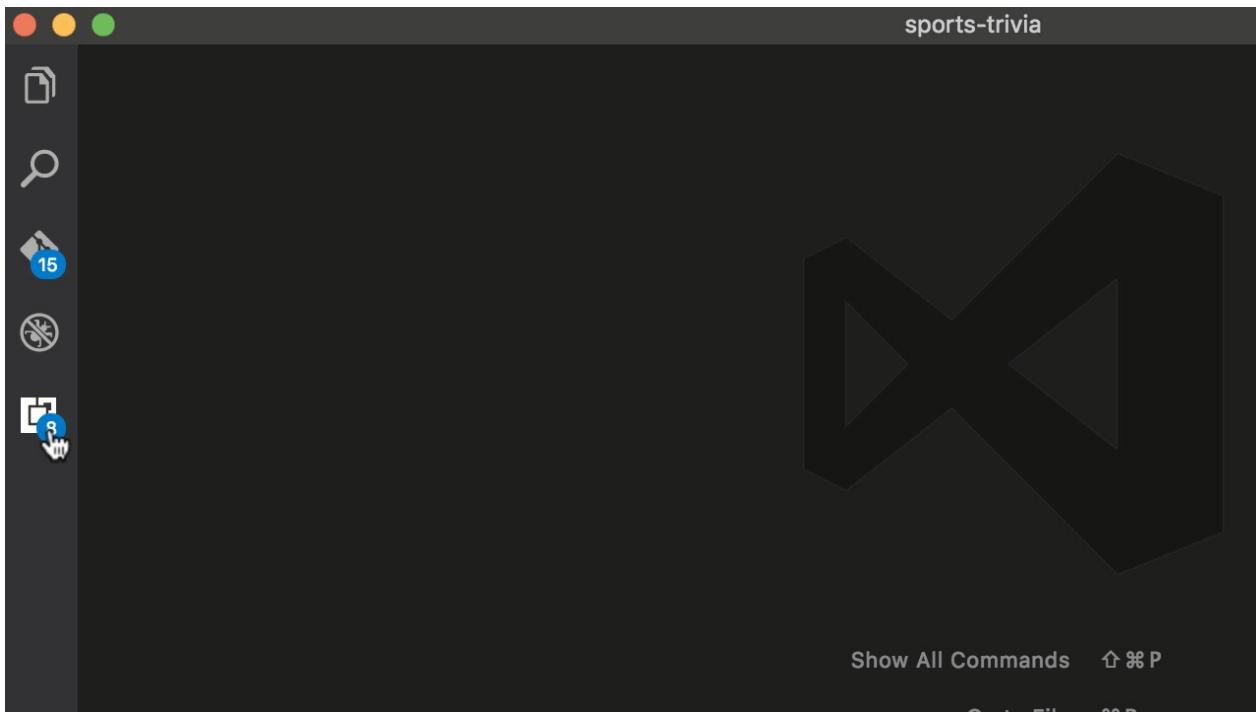
Install extensions

In the **Extensions** view, you can search via the search bar or click the **More (...)** button to filter and sort by install count.



Extension recommendations

In the **Extensions** view, click **Show Recommended Extensions** in the **More (...)** button menu.



Creating my own extension

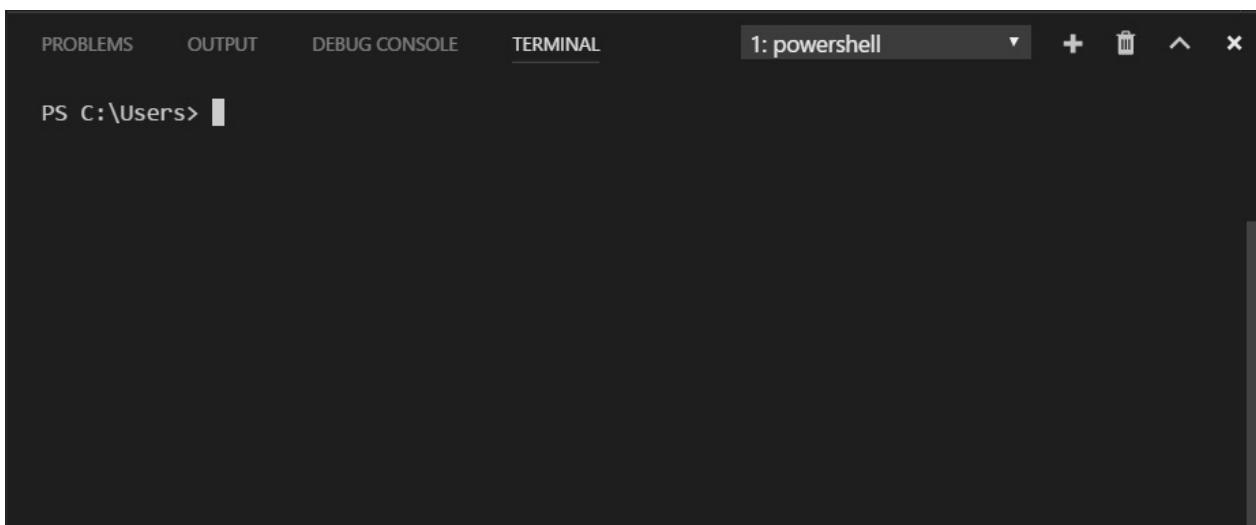
Are you interested in creating your own extension? You can learn how to do this in the documentation, specifically check out the [documentation on contribution points](#).

- configuration
- commands
- keybindings
- languages
- debuggers
- grammars
- themes
- snippets
- jsonValidation

Files and Folders

Integrated Terminal

Keyboard Shortcut: `kb(workbench.action.terminal.toggleTerminal)`



Further reading:

- [Integrated Terminal documentation](#)
- [Mastering VS Code's Terminal article](#)

Auto Save

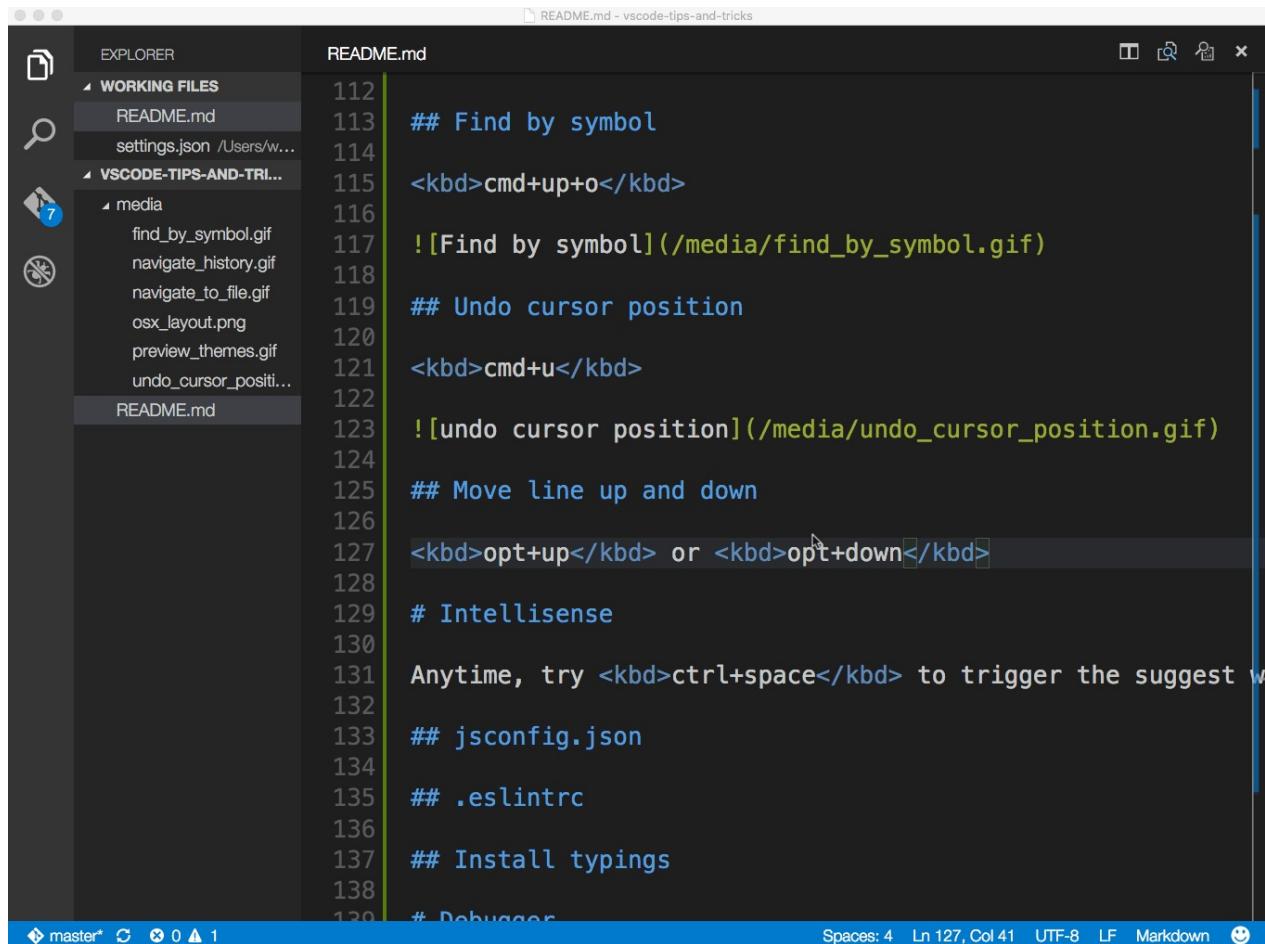
Open User Settings `settings.json` with `kb(workbench.action.openGlobalSettings)`

```
"files.autoSave": "afterDelay"
```

You can also toggle Auto Save from the top-level menu with the **File > Auto Save**.

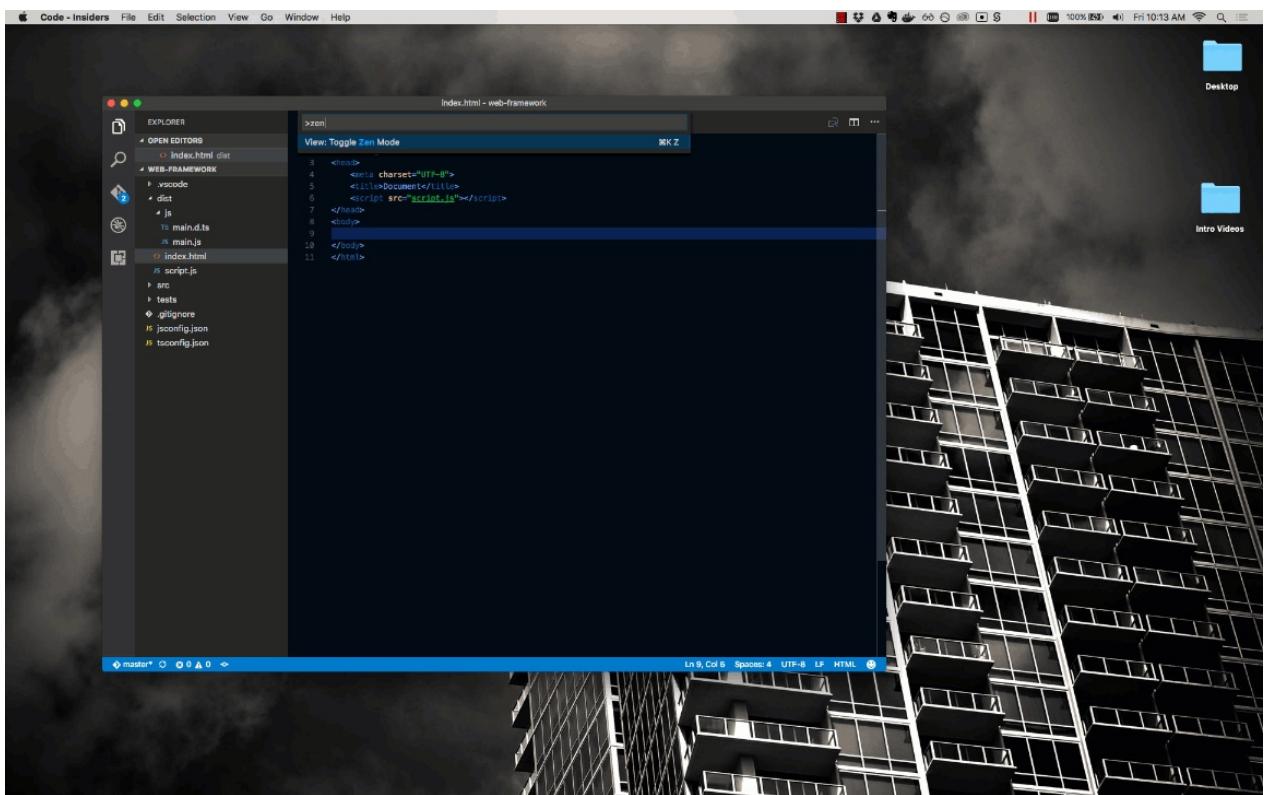
Toggle Sidebar

Keyboard Shortcut: kb(workbench.action.toggleSidebarVisibility)



Zen Mode

Keyboard Shortcut: kb(workbench.action.toggleZenMode)



Enter distraction free Zen mode.

Side by side editing

Keyboard Shortcut: `kb(workbench.action.splitEditor)`

You can also use `kbstyle(Ctrl)` then click a file from the File Explorer (`kbstyle(Cmd+click)` on macOS).

```

82 ````json
83 "files.autoSave": "on"
84 ````

85 ## Toggle sidebar
86 <kbd>cmd+B</kbd>
87 ! [toggle side bar] (/media/toggle_side_bar.gif)

88 ## Side by side editing
89 <kbd>cmd+1</kbd>, <kbd>cmd+2</kbd>, <kbd>cmd+3</kbd>

90 ## Switch between editors
91 <kbd>cmd+</kbd> or <kbd>cmd</kbd> then click a file from the file browser.

92 ## History
93
94 Navigate entire history with <kbd>ctrl+tab</kbd>
95 Navigate back with <kbd>ctrl+-</kbd>.
96 Navigate forward with <kbd>ctrl+shift+up</kbd>
97
98 ! [navigate history] (/media/navigate_history.gif)

99 ## Navigate to a file
100
101
102
103
104
105
106
107
108

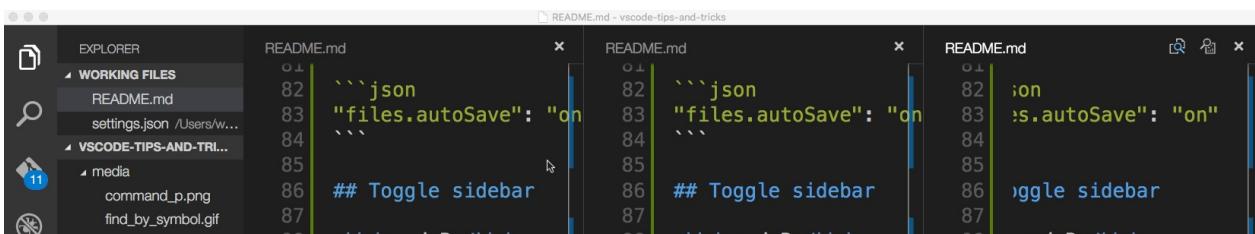
```

Spaces: 4 Ln 92, Col 24 UTF-8 LF Markdown

You can use drag and drop editors to create new editor groups and move editors between groups.

Switch between editors

Keyboard Shortcut: `kb(workbench.action.focusFirstEditorGroup)`,
`kb(workbench.action.focusSecondEditorGroup)`, `kb(workbench.action.focusThirdEditorGroup)`



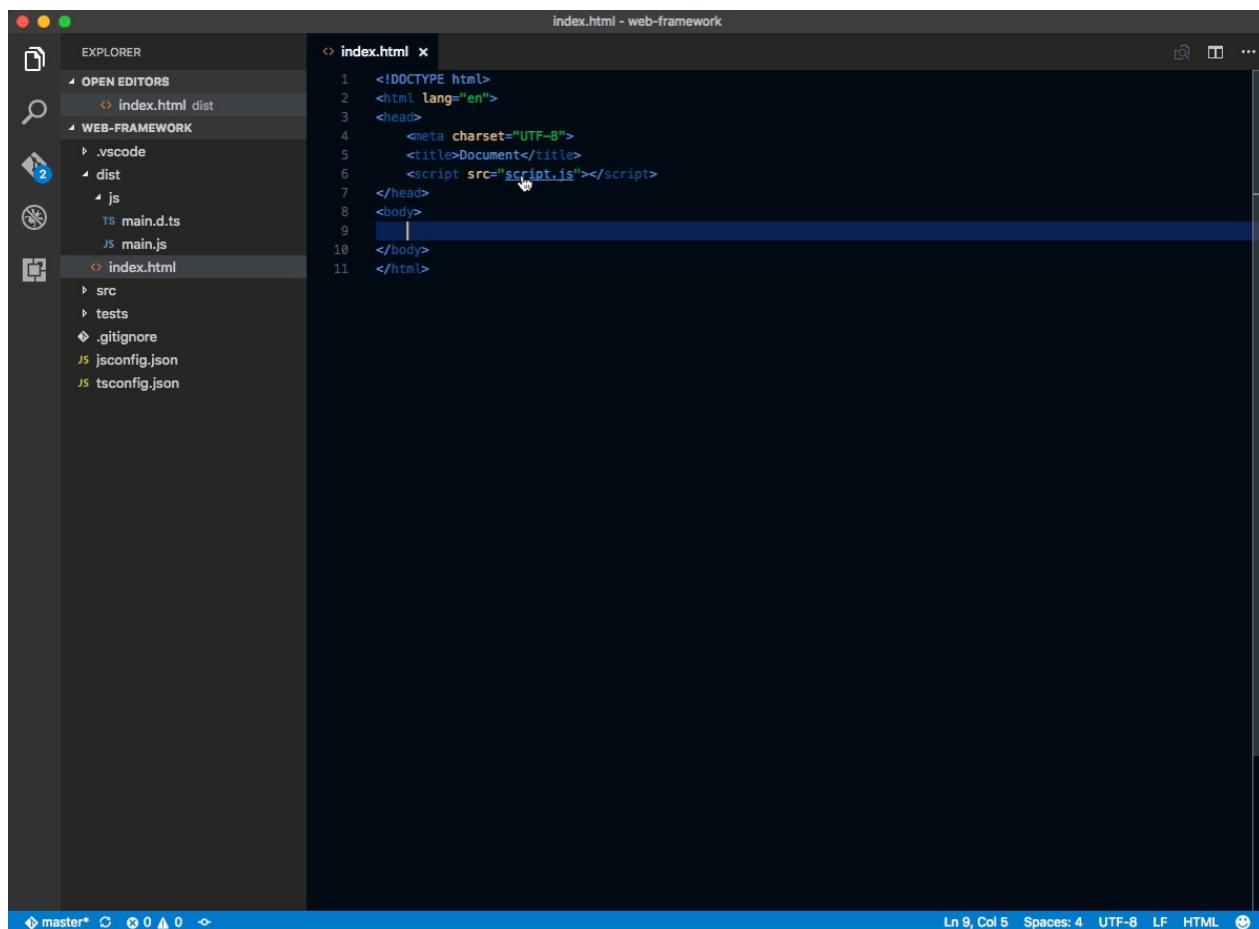
Move to Explorer window

Keyboard Shortcut: `kb(workbench.view.explorer)`

Create or open a file

Keyboard Shortcut: `kbstyle(Ctrl+click)` (`kbstyle(Cmd+click)` on macOS)

You can quickly open a file or image or create a new file by moving the cursor to the file link and using `kbstyle(Ctrl+click)`.



Close the currently opened folder

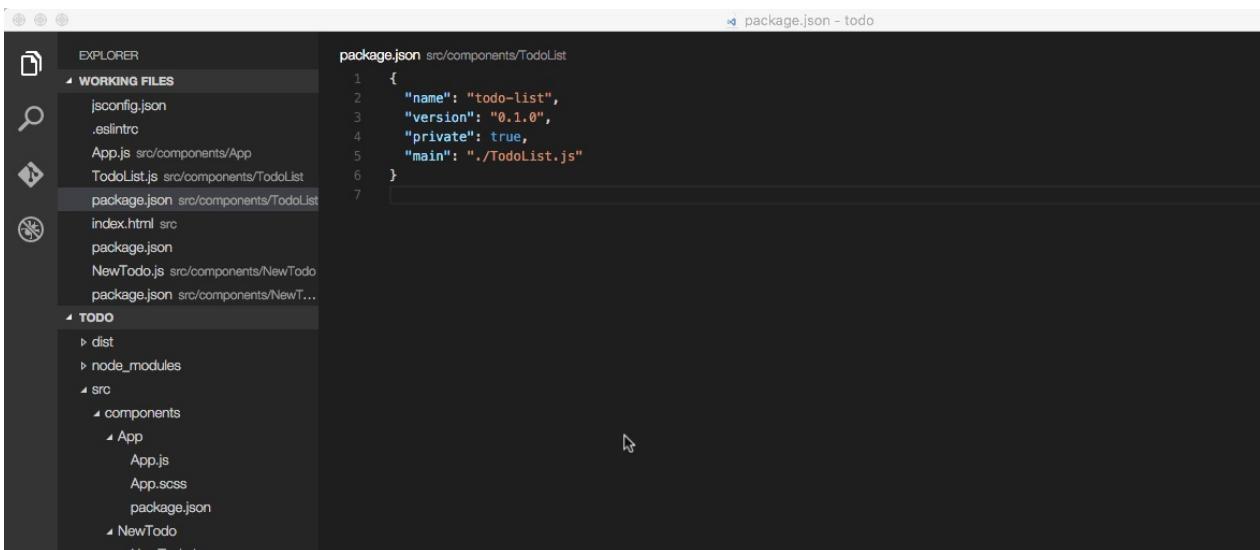
Keyboard Shortcut: `kb(workbench.action.closeActiveEditor)`

Navigation history

Navigate entire history: `kb(workbench.action.openNextRecentlyUsedEditorInGroup)`

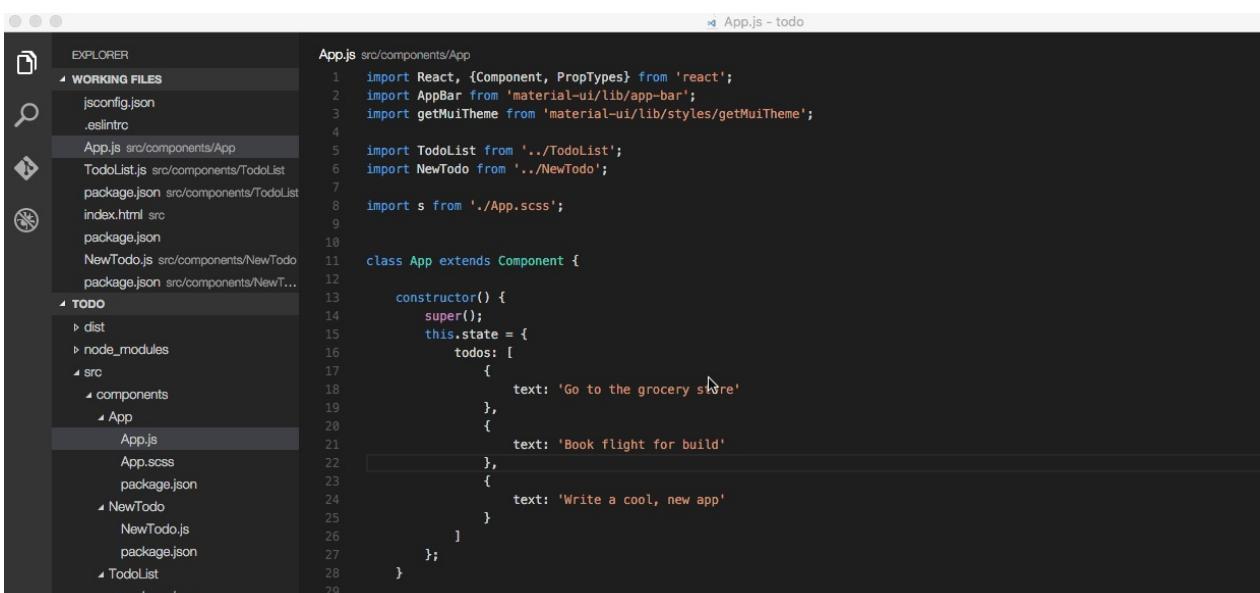
Navigate back: `kb(workbench.action.navigateBack)`

Navigate forward: `kb(workbench.action.navigateForward)`



Navigate to a file

Keyboard Shortcut: kb(workbench.action.quickOpen)



File associations

Create language associations for files that aren't detected correctly. For example, many configuration files with custom file extensions are actually JSON.

```
"files.associations": {  
    ".database": "json"  
}
```

Editing Hacks

Here are a selection of common features for editing code. If the keyboard shortcuts aren't comfortable for you, consider installing a [keymap extension](#) for your old editor.

Tip: You can see recommended keymap extensions in the **Extensions** view with

```
kb(workbench.extensions.action.showRecommendedKeymapExtensions) which filters the search to  
@recommended:keymaps .
```

Multi cursor selection

Keyboard Shortcut: `kb(editor.action.insertCursorAbove)` or

```
kb(editor.action.insertCursorBelow)
```

```
6   .app {  
7     margin: 0;  
8     width: 100%;  
9     height: 100%;  
10    }  
11
```

```
31 .global-message-list.transition {  
32 → -webkit-transition: top 200ms linear;  
33 → -ms-transition: top 200ms linear;  
34 → -moz-transition: top 200ms linear;  
35 → -khtml-transition: top 200ms linear;  
36 → -o-transition: top 200ms linear;  
37 → transition: top 200ms linear;  
38 }
```

Add more cursors to current selection.

```
render() {  
  return (  
    <div className={s.app}>  
      <AppBar  
        title="Todo"  
        iconClassNameRight="muidocs-icon-navigation-expand-more"  
      />  
      <div style={{  
        marginTop: 20,  
        marginLeft: 20  
      }}>  
        <NewNote />  
        <Notes items={this.state.notes}/>  
      </div>  
    </div>  
  );  
}
```

Note: You can also change the modifier to `kbstyle(Ctrl/Cmd)` for applying multiple cursors with the `editor.multiCursorModifier` setting. See [Multi-cursor Modifier](#) for details.

Join line

Keyboard Shortcut: `kb(editor.action.joinLines)`

Windows / Linux: Not bound by default. Open [Keyboard Shortcuts](#) (`kb(workbench.action.openGlobalKeybindings)`) and bind `editor.action.joinLines` to a shortcut of your choice.

```

JS App.js      JS util.js      X   ⌂ keybindings.json
1
2
3
4     function foo(a,
5         b,
6         c,
7         d,
8         e) []
9
10    }

```

Copy line up / down

Keyboard Shortcut: `kb(editor.action.copyLinesUpAction)` or
`kb(editor.action.copyLinesDownAction)`

The commands **Copy Line Up/Down** are unbound on Linux because the VS Code default keybindings would conflict with Ubuntu keybindings, see [Issue #509](#). You can still set the commands `editor.action.copyLinesUpAction` and `editor.action.copyLinesDownAction` to your own preferred keyboard shortcuts.

server.js

```
1  var express = require('express');
2  var bodyParser = require('body-parser')
3
4  var database = require('./database');
5
```

Shrink / expand selection

Keyboard Shortcut: kb(editor.action.smartSelect.shrink) or
kb(editor.action.smartSelect.grow)



```
constructor() {
  super();
  this.state = {
    notes: [
      {
        text: 'Book flight for build'
      },
      {
        text: 'Write a cool, new app'
      },
      {
        text: 'build a react app'
      }
    ];
}
```

You can learn more in the [Basic Editing](#) documentation.

Go to Symbol in File

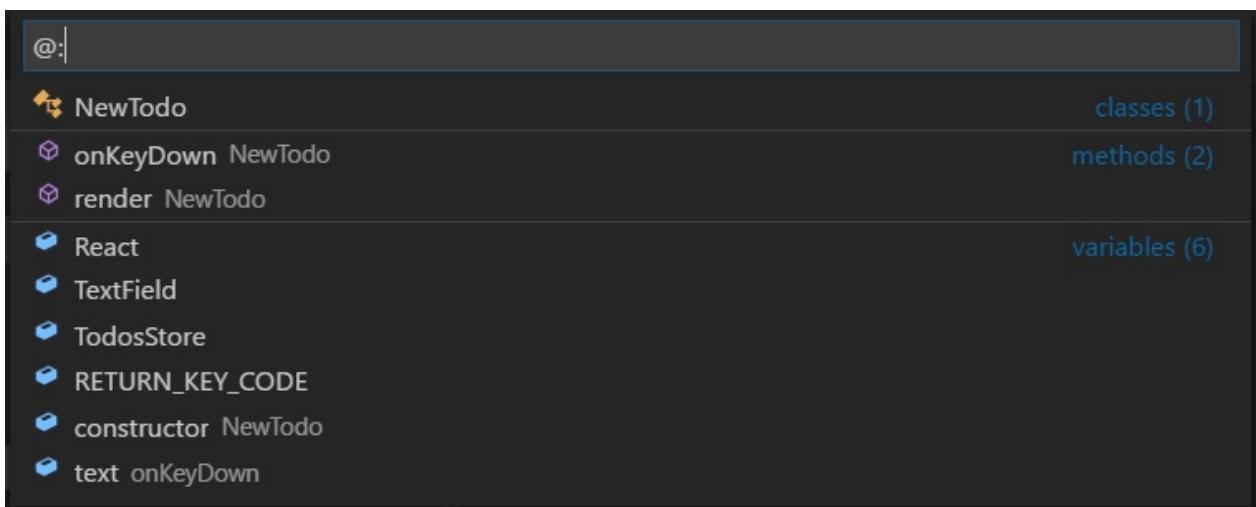
Keyboard Shortcut: kb(workbench.action.gotoSymbol)

```
Default Settings - vscode-tips-and-tricks

EXPLORER
  WORKING FILES
    README.md
    settings.json /Users/wangzhiqiang/...
  VS CODE TIPS AND TRICKS
    media
      osx_layout.png
      preview_themes.gif
    README.md

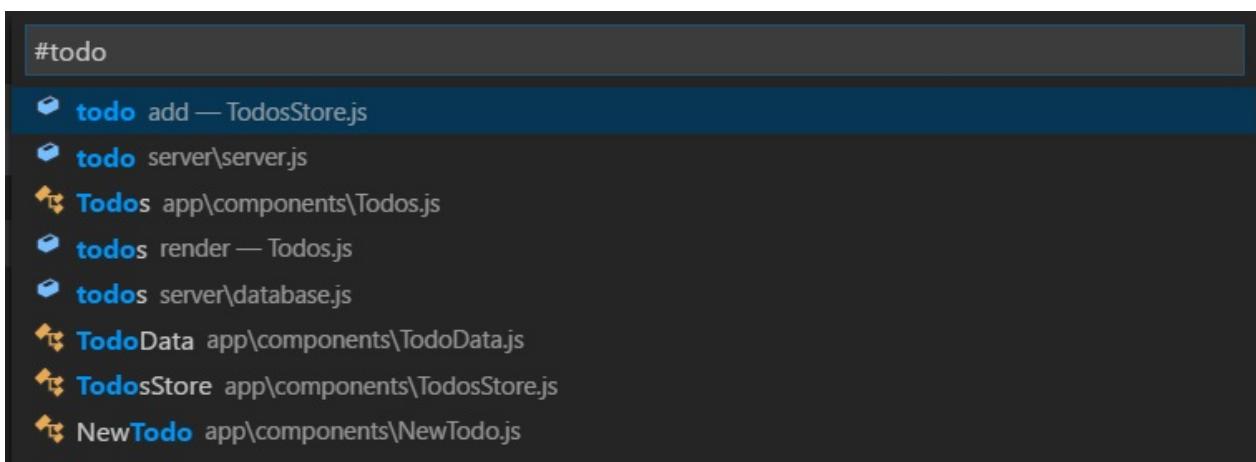
Default Settings
119
120      // The default end of line character.
121      "files.eol": "\n",
122
123      // When enabled, will trim trailing whites
124      "files.trimTrailingWhitespace": false,
125
126      // Controls auto save of dirty files. Acces
127      "files.autoSave": "off",
128
129      // Controls the delay in ms after which a
130      "files.autoSaveDelay": 1000,
131
132
133      //----- File Explorer configuration -----
134
135      // Maximum number of working files to show
136      "explorer.workingFiles.maxVisible": 9,
137
138      // Controls if the height of the working f
139      "explorer.workingFiles.dynamicHeight": tru
140
141
142      //----- HTTP configuration -----
143
144      // The proxy setting to use. If not set wi
145      "http.proxy": "",
146
147      // Whether the proxy server certificate sh
148      "http.proxyStrictSSL": true,
149
150
151      //----- Update configuration -----
152
153      // Configure the update channel to receive
154      "update.channel": "default",
155
156
```

You can group the symbols by kind by adding a colon, `@:`.



Go to Symbol in Workspace

Keyboard Shortcut: `kb(workbench.action.showAllSymbols)`



Navigate to a specific line

Keyboard Shortcut: `kb(workbench.action.gotoLine)`

```

TodoList.js src/components/TodoList
1 import React, {Component} from 'react';
2 import List from 'material-ui/lib/lists/list';
3 import ListItem from 'material-ui/lib/lists/list-item';
4
5
6 class TodoList extends Component {
7
8   createItem(item) {
9     return <ListItem key={item.text}>{item.text}</ListItem>;
10 }
11
12   render() {
13
14     return (
15       <List>{this.props.items.map(this.createItem)}</List>
16     );
17   }
18 }
19
20 export default TodoList;
21

```

Undo cursor position

Keyboard Shortcut: kb(cursorUndo)

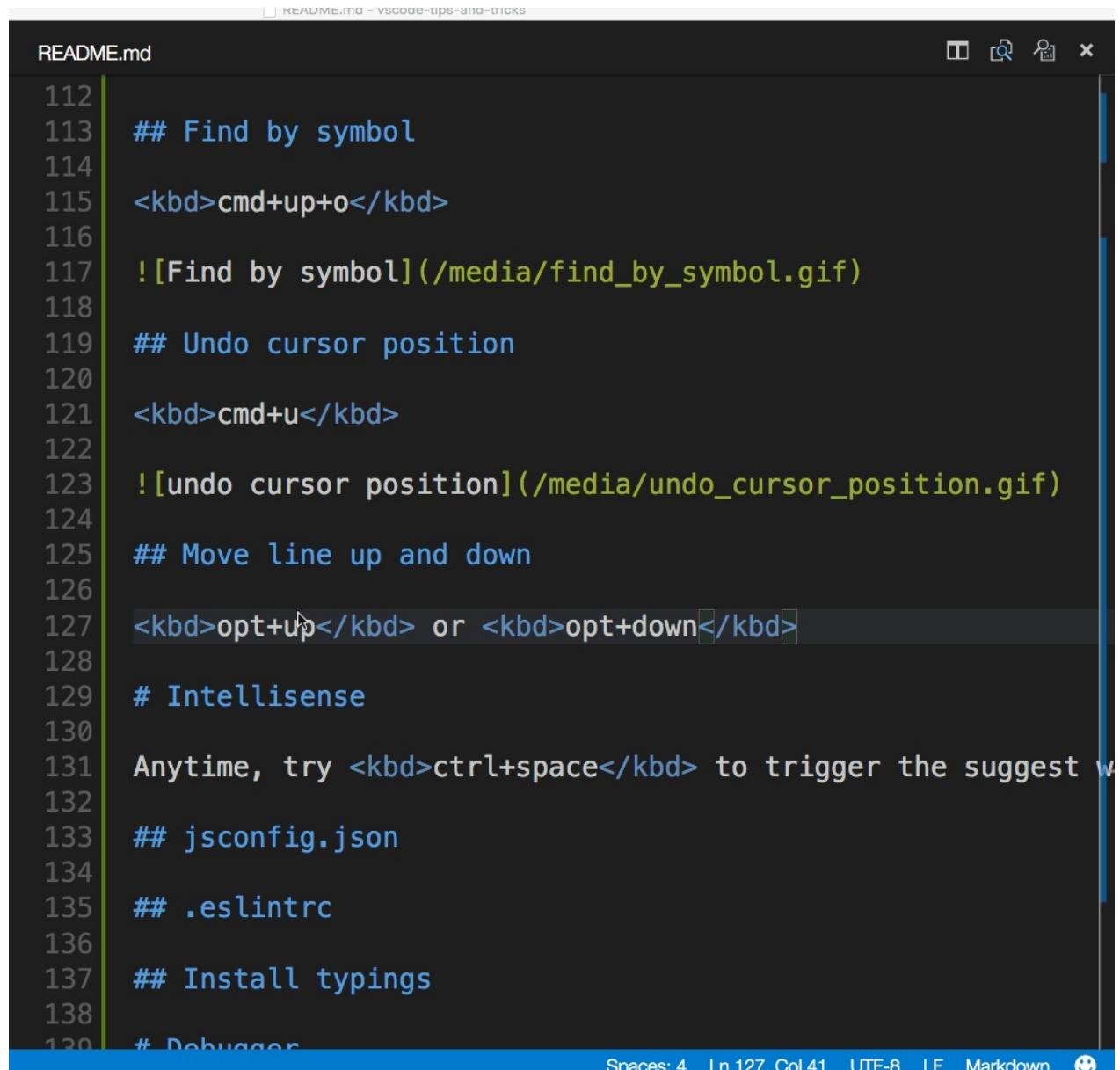
```

110
111 # Editor hacks
112
113 ## Find by symbol
114
115 <kbd>cmd+up+o</kbd>
116
117 ! [Find by symbol] (/media/find_by_symbol.gif)
118
119 ## Undo cursor position
120
121 <kbd>cmd+u</kbd>
122
123 ## Move line up and down
124
125 <kbd>opt+up</kbd> or <kbd>opt+down</kbd>
126
127 # Intellisense
128

```

Move line up and down

Keyboard Shortcut: `kb(editor.action.moveLinesUpAction)` or
`kb(editor.action.moveLinesDownAction)`



The screenshot shows the VS Code interface with the file 'README.md' open. The code editor displays several keyboard shortcuts:

- Line 112: `## Find by symbol`
- Line 115: `<kbd>cmd+up+o</kbd>`
- Line 117: `![Find by symbol](/media/find_by_symbol.gif)`
- Line 119: `## Undo cursor position`
- Line 121: `<kbd>cmd+u</kbd>`
- Line 123: `![undo cursor position](/media/undo_cursor_position.gif)`
- Line 125: `## Move line up and down`
- Line 127: `<kbd>opt+up</kbd> or <kbd>opt+down</kbd>`
- Line 129: `# Intellisense`
- Line 131: `Anytime, try <kbd>ctrl+space</kbd> to trigger the suggest w`
- Line 133: `## jsconfig.json`
- Line 135: `## .eslintrc`
- Line 137: `## Install typings`
- Line 138: `# Debugger`

The status bar at the bottom of the screen shows: Spaces: 4 Ln 127, Col 41 UTF-8 LF Markdown 😊

Trim trailing whitespace

Keyboard Shortcut: `kb(editor.action.trimTrailingWhitespace)`

```
render() {  
  
    return (  
        <List>{this.props.items.map(this.createItem)}</List>  
    );  
}  
}
```

Code formatting

Currently selected source code: kb(editor.action.formatSelection)

Whole document format: kb(editor.action.formatDocument)

```
render() {  
    return (  
        <div className={s.app}>  
            <AppBar  
                title="Notes"  
                iconClassNameRight="muidocs-icon-navigation-expandmore" />  
            <div style={{  
                marginTop: 20,  
                marginLeft: 20  
            }}>  
                <NewNote />  
                <Notes items={this.state.notes}/>  
            </div>  
        </div>  
    );  
}
```

Code folding

Keyboard Shortcut: kb(editor.fold) and kb(editor.unfold)

```
class App extends Component {  
  
  constructor() {  
    super();  
    this.state = {  
      notes: [  
        {  
          text: "go to the grocery store"  
        },  
        {  
          text: 'read medium article about engineering'  
        },  
        {  
          text: 'create build session'  
        },  
        {  
          text: 'fix bug #232'  
        }  
      ]  
    };  
  }  
}
```

Select current line

Keyboard Shortcut: kb(expandLineSelection)

```
render() {
  return (
    <div className={s.app}>
      <AppBar
        title="Notes"
        iconClassNameRight="muidocs-icon-navigation-expand-right"
      />
      <div style={{marginTop: 20, marginLeft: 20}}>
        <NewNote />
        <Notes items={this.state.notes}/>
      </div>
    </div>
  );
}
```

Navigate to beginning and end of file

Keyboard Shortcut: `kb(cursorTop)` and `kb(cursorBottom)`

The screenshot shows a code editor window with the following details:

- Title Bar:** App.js src/components/App
- Code Content:**

```
1 import React, {Component, PropTypes} from 'react';
2 import AppBar from 'material-ui/lib/app-bar';
3 import getMuiTheme from 'material-ui/lib/styles/getMuiTheme';
4
5 import Notes from '../Notes';
6 import NewNote from '../NewNote';
7
8 import s from './App.scss';
9
10
11 class App extends Component {
12
13     constructor() {
14         super();
15         this.state = {
16             notes: [
17                 {
18                     text: "go to the grocery store"
19                 },
20                 {
21                     text: 'read medium article about engineering'
22                 },
23                 {
24                     text: 'create build session'
25                 },
26                 {
27                     text: 'fix bug #232'
28                 }
29             ]
30         };
31     }
32
33     getChildContext(){
34         return {

```
- Status Bar:** Ln 1, Col 1 Spaces: 4 UTF-8 LF JavaScript 😊

Open Markdown Preview

In a Markdown file, use

Keyboard Shortcut: kb(markdown.showPreview)

The screenshot shows a dark-themed code editor window for a file named 'README.md'. The file contains several lines of code, each preceded by a line number. The code includes sections for branches, staging, committing, and git output. It also provides instructions for viewing git commands and styling README pages. A section on toggle preview is present, along with keyboard shortcuts for both preview and output.

```
194  **INLINE VIEW**  
195  
196  ## Branches  
197  
198  ## Staging  
199  
200  ## Committing  
201  
202  ## See Git output  
203  
204  Sometimes I want to see what my tool is doing. Visual Studio Code ma  
205  what git commands are running. This is helpful when learning git or  
206  source control issue.  
207  
208  <kbd>shift+cmd+u</kbd> to run `toggleOutput`. Select Git in the drop  
209  
210  # README tips  
211  
212  ## Styling README pages  
213  ↴  
214  ## Toggle preview  
215  
216  <kbd>shift+cmd+v</kbd> to `Toggle Preview`  
217  
218  
219  
220  
221  
222
```

Side by Side Markdown Edit and Preview

In a Markdown file, use

Keyboard Shortcut: `kb(markdown.showPreviewToSide)`

Special bonus: The preview will now sync.

```

1  ---
2  Order: 9
3  Area: languages
4  TOCTitle: Markdown
5  ContentId:
6  47A8BA5A-A103-4B61-B5FB-185C15E54C52
7  PageTitle: Markdown editing with
8  Visual Studio Code
9  DateApproved: 12/14/2016
10 MetaDescription: Get the best out of
11 Visual Studio Code for Markdown
12 ---
13 # Markdown and VS Code
14
15 Working with Markdown files in Visual
16 Studio Code is simple, straightforward,
17 and fun. Besides VS Code's basic
18 editing, there are a number of
19 Markdown specific features that will
20 help you be more productive.

```

Working with Markdown files in Visual Studio Code is simple, straightforward, and fun. Besides VS Code's basic editing, there are a number of Markdown specific features that will help you be more productive.

Markdown Extensions

In addition to the functionality VS Code provides out of the box, you can install an extension for greater functionality.

Tip: The extensions shown above are

Ln 10, Col 17 Spaces: 4 UTF-8 CRLF Markdown

IntelliSense

`kb(editor.action.triggerSuggest)` to trigger the Suggestions widget.

```

18 // Routes
19
20 // Retrieve all todos
21 server.get('/todos', function(req, res, next) {
22   database.getAll().then(function(todos) {
23     ...
24   });
25 });
26
27 // Add a new todo
28 server.post('/todos', function(req, res, next) {
29   var todo = req.body;
30   database.add(todo).then(function(todos) {
31     res.send(todos);
32     next();
33   });

```

You can view available methods, parameter hints, short documentation, etc.

Peek

Select a symbol then type `kb(editor.action.peekImplementation)` . Alternatively, you can use the context menu.

```
22      });
23  }
24
25  handleClick(note) {
26    NotesStore.remove(note);
27  }
28
29  create(note) {
30    return (<ListItem onMouseDown={this.handleClick.bind(null, note)}>
31          {note.text}
32        </ListItem>
33      );
34  }
35
36  render() {
37    const todos = this.state.notes.map(this.create.bind(this));
38    return (
39      <List style={{width: 400}}>
```

Go to Definition

Select a symbol then type `kb(editor.action.goToDeclaration)` . Alternatively, you can use the context menu or `kbstyle(Ctrl+click)` (`kbstyle(Cmd+click)` on macOS).

The screenshot shows a code editor window with the title "Notes.js - todo". The code is written in JavaScript and uses Material-UI components. The status bar at the bottom shows "DO:s", "Ln 12, Col 21", "Spaces: 2", "UTF-8", "LF", "JavaScript", and a smiley face icon.

```

1 import React, {Component} from 'react';
2 import List from 'material-ui/lib/lists/list';
3 import ListItem from 'material-ui/lib/lists/list-item';
4 import DoneIcon from 'material-ui/lib/svg-icons/action/done';
5
6 import NotesStore from './NotesStore';
7
8 class Notes extends Component {
9
10    constructor() {
11        super();
12        this.state = [
13            notes: NotesStore.getAll()
14        ];
15    }
16
17    componentDidMount() {
18        NotesStore.subscribe((action) => {
19            this.setState({
20                notes: action.notes
21            });
22        });
23    }
24
25    handleClick(note) {
26        NotesStore.remove(note);

```

You can go back to your previous location with the **Go > Back** command or

```
kb(workbench.action.navigateBack) .
```

You can also see the type definition if you press `kbstyle(Ctrl)` (`kbstyle(Cmd)` on macOS) when you are hovering over the type.

Find All References

Select a symbol then type `kb(editor.action.referenceSearch.trigger)`. Alternatively, you can use the context menu.

```
5
6 import NotesStore from './NotesStore';
7
8 class Todo extends Component {
9
10    constructor() {
11        super();
12        this.state = {
13            notes: NotesStore.getAll()
14        };
15    }
16
17    componentDidMount() {
18        NotesStore.subscribe((action) => {
19            this.setState({
20                notes: action.notes
21            });
22        });
23    }
24
25    handleClick(note) {
26        NotesStore.remove(note);
27    }
28
29    create(note) {
30        return (<ListItem onMouseDown={this.handleClick.bind(null, note)} key={note.id}>
31            {note.text}
32        </ListItem>
33    }
34}
```

O:s

Ln 8, Col 10 Spaces: 2 UTF-8 LF JavaScript 😊

Rename Symbol

Select a symbol then type `kb(editor.action.rename)`. Alternatively, you can use the context menu.

```
7
8     class Notes extends Component { ↵
9
10    constructor() {
11        super();
12        this.state = {
13            notes: NotesStore.getAll()
14        };
15    }
16
17    componentDidMount() {
18        NotesStore.subscribe((action) => {
19            this.setState({
20                notes: action.notes
21            });
22        });
23    }
24
25    handleClick(note) {
26        NotesStore.remove(note);
27    }
28
29    create(note) {
30        return (<ListItem onMouseDown={this.handleClick.bind(null, note)} key={note.id}
31                {note.text}>
32        </ListItem>
33    );
34}
```

D:s Ln 16, Col 1 Spaces: 2 UTF-8 LF JavaScript

.eslintrc.json

Install the [ESLint extension](#). Configure your linter however you'd like. Consult the [ESLint specification](#) for details on it's linting rules and options.

Here is configuration to use ES6.

```
{  
  "env": {  
    "browser": true,  
    "commonjs": true,  
    "es6": true,  
    "node": true  
  },  
  "parserOptions": {  
    "ecmaVersion": 6,  
    "sourceType": "module",  
    "ecmaFeatures": {  
      "jsx": true,  
      "classes": true,  
      "defaultParams": true  
    }  
  },  
  "rules": {  
    "no-const-assign": 1,  
    "no-extra-semi": 0,  
    "semi": 0,  
    "no-fallthrough": 0,  
    "no-empty": 0,  
    "no-mixed-spaces-and-tabs": 0,  
    "no-redeclare": 0,  
    "no-this-before-super": 1,  
    "no-undef": 1,  
    "no-unreachable": 1,  
    "no-use-before-define": 0,  
    "constructor-super": 1,  
    "curly": 0,  
    "eqeqeq": 0,  
    "func-names": 0,  
    "valid-typeof": 1  
  }  
}
```

package.json

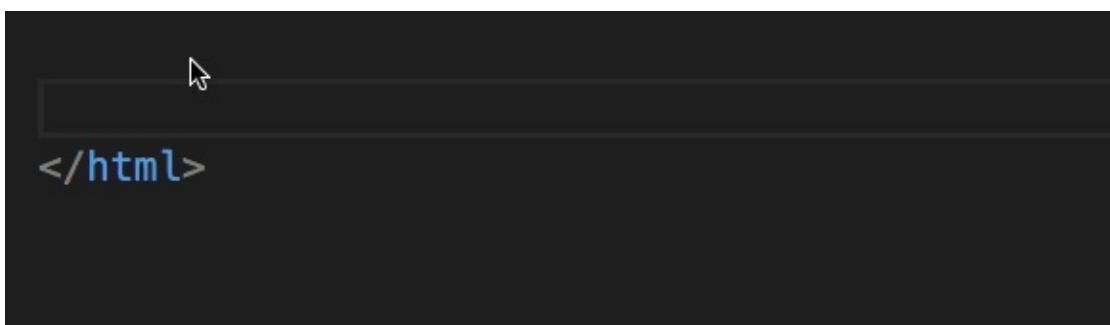
See IntelliSense for your `package.json` file.

package.json

```
1  {
2    "name": "react-todo",
3    "version": "1.0.0",
4    "description": "Simple todo application to demo react and es6",
5    "main": "webpack.config.js",
6    "scripts": {
7      "build": "webpack --config webpack.config.js",
8      "dev": "webpack-dev-server --config webpack.config.js --op
9    },
10   "author": "waderyan",
11   "license": "ISC",
12   "devDependencies": {
13     "babel-core": "^6.5.2",
14     "babel-loader": "^6.2.3",
15     "babel-preset-es2015": "^6.5.0",
16     "babel-preset-react": "^6.5.0",
17     "babel-preset-stage-0": "^6.5.0",
18     "html-webpack-plugin": "^2.8.1",
19     "node-sass": "^3.4.2",
20     "sass-loader": "^3.1.2",
21     "style-loader": "^0.13.0",
22     "webpack": "^1.12.13",
23     "webpack-dev-server": "^1.14.1"
24 }
```

Emmet syntax

[Support for Emmet syntax.](#)



Snippets

Create custom snippets

File > Preferences > User Snippets, select the language, and create a snippet.

```
"create component": {  
  "prefix": "component",  
  "body": [  
    "class $1 extends React.Component {",  
    "",  
    "\trender() {",  
    "\t\treturn ($2);",  
    "\t}",  
    "",  
    "}"  
  ],  
},
```

See more details in [Creating your own Snippets](#).

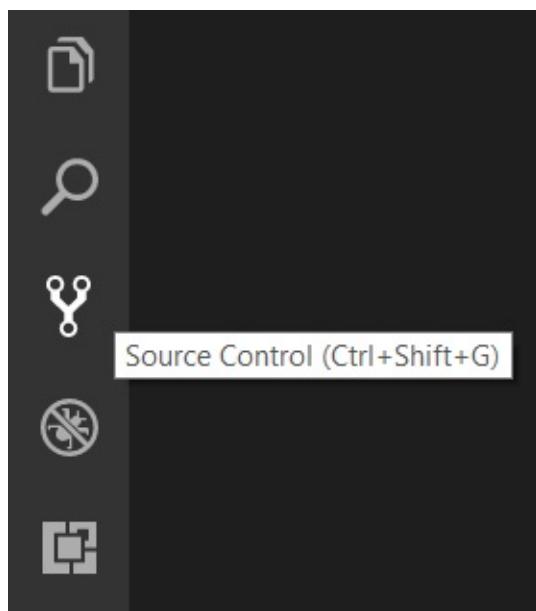
Git integration

Keyboard Shortcut: kb(workbench.view.scm)

Git integration comes with VS Code "in-the-box". You can install other SCM provider from the extension Marketplace. This section describes the Git integration but much of the UI and gestures are shared by other SCM providers.

Diffs

From the **Source Control** view, select the file to diff.



Side by side

Default is side by side diff.

The screenshot shows a code editor window with two panes. The left pane displays the original code, and the right pane shows the modified code with red highlights indicating changes. A green box highlights the text "Click Git icon ![git icon](/media/git_icon.png)". Below the code, the status bar indicates "Ln 188, Col 1 Tab Size: 4 UTF-8 LF Markdown".

```
1 / 1  
172 # Debugging  
173  
174 # Task Runner  
175  
176 ## Auto detect tasks  
177  
178 ## Gulp tasks  
179  
180 ## Mocha  
181  
182 # Git Integration  
183  
184 Excellent integration for entire Git workflow.  
185  
186 ## Diffs  
187  
188 ## Branches  
189  
190 ## Staging  
191  
192 ## Committing  
193  
194 ## See Git output  
195  
196 ## Branches  
197  
198 ## Staging  
199  
200 ## Committing  
201  
202 ## See Git output  
203  
204 Sometimes I want to see what my tool is doing. V  
205 what git commands are running. This is helpful w  
0 Ln 188, Col 1 Tab Size: 4 UTF-8 LF Markdown 😊
```

Inline view

Toggle inline view by clicking the **More (...)** button in the top right and selecting **Switch to Inline View**.

```
11 11 class App extends Component {  
12 12     constructor() {  
13 13         super();  
14 14         this.state = {  
15 15             todos: [  
16 16                 {  
17 17                     text: 'Go to the grocery store'  
18 18                     text: 'Book flight for build'  
19 19                 },  
20 20                 {  
21 21                     text: 'Book flight for build'  
22 22                     text: 'Write a cool, new app'  
23 23                 },  
24 24                     text: 'Write a cool, new app'  
25 25                     text: 'Create a react todo list app'  
26 26                 ]  
27 27             };  
28 28     }  
29 29  
30 30     getChildContext(){  
31 31         return {
```

If you prefer the inline view, you can set `"diffEditor.renderSideBySide": false`.

Review pane

Navigate through diffs with `kb(editor.action.diffReview.next)` and `kb(editor.action.diffReview.prev)`. This will present them in a unified patch format. Lines can be navigated with arrow keys and pressing `kbstyle(Enter)` will jump back in the diff editor and the selected line.

The screenshot shows a code diff interface for a file named 'diff.js' (Index). The interface has a dark theme with color-coded regions for different types of changes:

- Red background (deletions):** Lines 12 and 13 are highlighted in red, indicating they have been deleted.
- Green background (insertions):** Lines 10, 11, and 12 are highlighted in green, indicating they have been inserted.
- Grey background (unchanged):** Lines 7, 8, 9, 14, 15, 16, 17, 18, and 19 are shown in grey, indicating they remain unchanged.

```
1/2: @@ -7,10 +7,13 @@
    7      7  Version: 0.5
    8      8  */
    9      9
   10     +// Here are some inserted lines
   11     +// with some extra comments
   12     +...
   13 (function (global, undefined) {
   14   "use strict";
   15   undefinedVariable = {};
   16   undefinedVariable.prop = 5;
   17
   18   function initializeProperties(target, members) {
   19     var keys = Object.keys(members);
```

Edit pending changes

You can make edits directly in the pending changes of the diff view.

Branches

Easily switch between Git branches via the Status Bar.

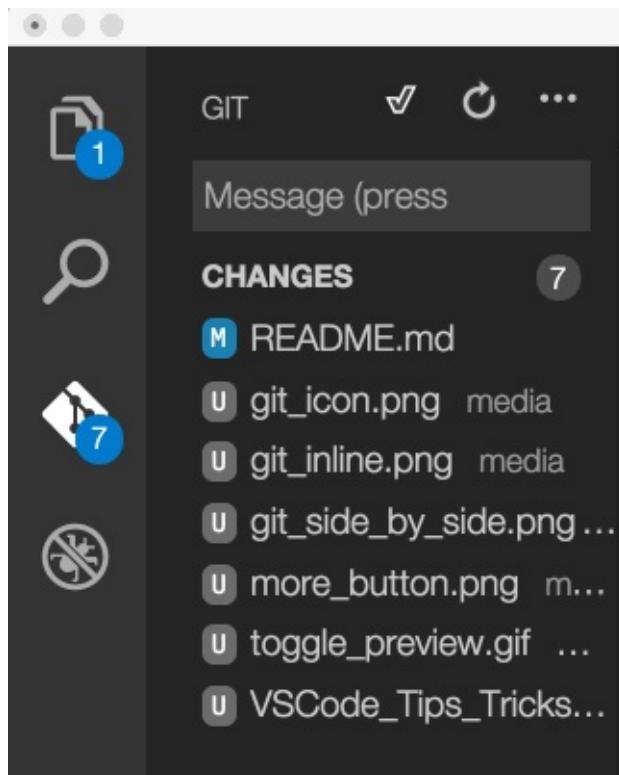
```
App.js src/components/App - Changes on index
1 import React, {Component, PropTypes} from 'react';
2 import AppBar from 'material-ui/lib/app-bar';
3 import getMuiTheme from 'material-ui/lib/styles/getMuiTheme';
4
5 import TodoList from '../TodoList';
6 import NewTodo from '../NewTodo';
7
8 import s from './App.scss';
9
10
11 class App extends Component {
12
13     constructor() {
14         super();
15         this.state = {
16             todos: [
17                 {
18                     text: 'Book flight for build'
19                 },
20                 {
21                     text: 'Write a cool, new app'
22                 },
23                 {
24                     text: 'Create a react todo list app'
25                 }
26             ]
27         };
28     }
29
30     render() {
31         return (
32             <div>
33                 <AppBar title="React Todo" muiTheme={getMuiTheme()} />
34                 <TodoList todos={this.state.todos} onAddTodo={this.addTodo} />
35             </div>
36         );
37     }
38
39     addTodo(text) {
40         const todos = this.state.todos.concat([
41             {
42                 text
43             }
44         ]);
45         this.setState({ todos });
46     }
47
48     deleteTodo(index) {
49         const todos = [...this.state.todos];
50         todos.splice(index, 1);
51         this.setState({ todos });
52     }
53
54     editTodo(index, text) {
55         const todos = [...this.state.todos];
56         todos[index].text = text;
57         this.setState({ todos });
58     }
59
60     handleTextChange(text) {
61         this.setState({ text });
62     }
63
64     handleFormSubmit(event) {
65         event.preventDefault();
66         this.addTodo(this.state.text);
67         this.setState({ text: '' });
68     }
69
70     handleDeleteClick(index) {
71         this.deleteTodo(index);
72     }
73
74     handleEditClick(index) {
75         this.editTodo(index, this.state.text);
76     }
77
78     handleTextChange(index, text) {
79         this.editTodo(index, text);
80     }
81
82     handleDelete(index) {
83         this.deleteTodo(index);
84     }
85
86     handleEdit(index) {
87         this.editTodo(index);
88     }
89
90     handleTextChange(index, text) {
91         this.editTodo(index, text);
92     }
93
94     handleDelete(index) {
95         this.deleteTodo(index);
96     }
97
98     handleEdit(index) {
99         this.editTodo(index);
100    }
101}
```

Ln 39, Col 23 Spaces: 4 JavaScript Configure Excludes 😊

Staging

Stage all

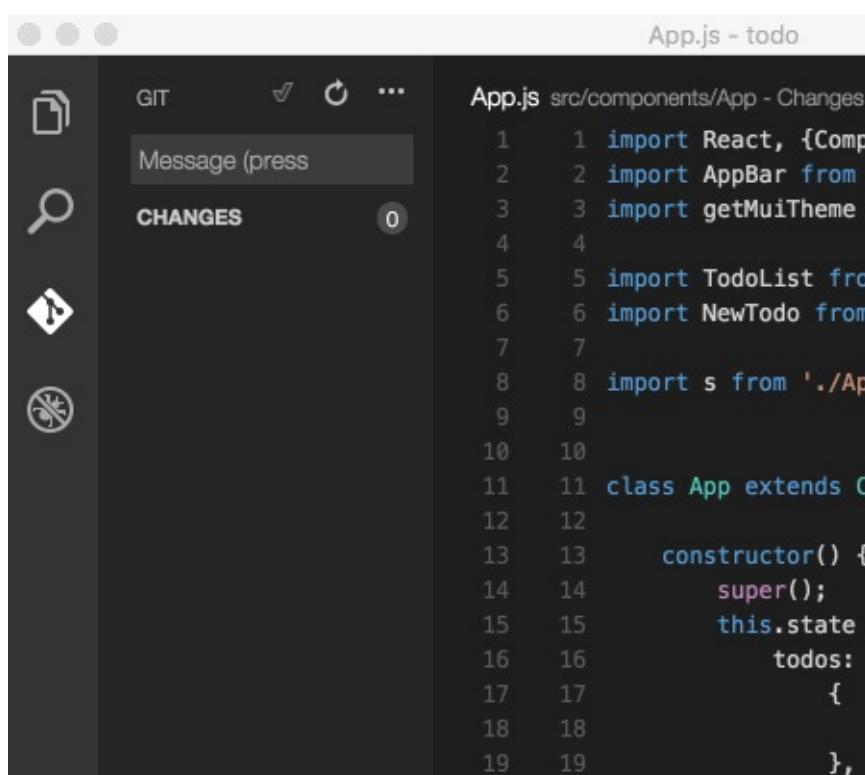
Hover over the number of files and click the plus button.



Stage selected

Stage a portion of a file by selecting that file (using the arrows) and then choosing **Stage Selected Ranges** from the **Command Palette**.

Undo last commit



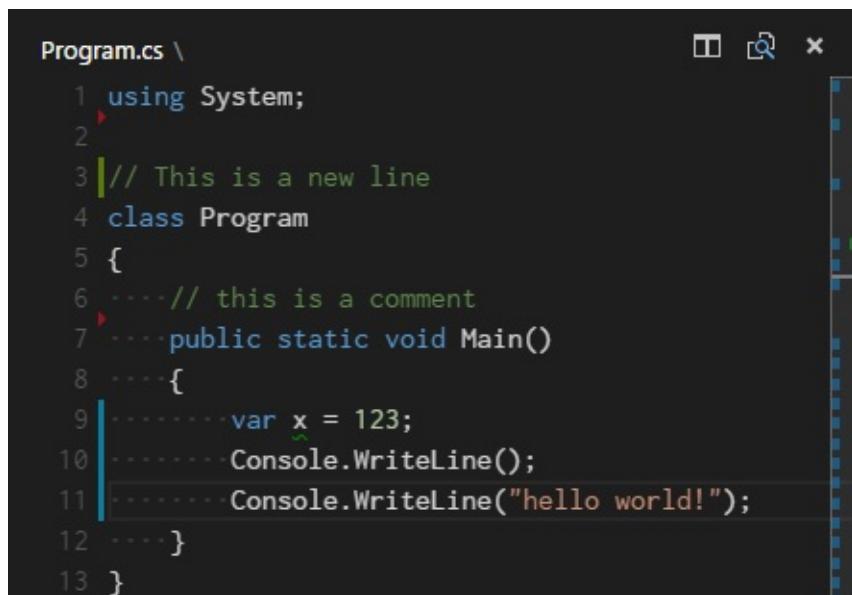
See Git output

VS Code makes it easy to see what Git commands are actually running. This is helpful when learning Git or debugging a difficult source control issue.

Use the **Toggle Output** command (`kb(workbench.action.output.toggleOutput)`) and select **Git** in the drop-down.

Gutter indicators

View diff decorations in editor. See [documentation](#) for more details.



Resolve merge conflicts

During a merge, go to the **Source Control** view (`kb(workbench.view scm)`) and make changes in the diff view.

Setup VS Code as default merge tool

```
git config --global merge.tool code
```

Debugging

Configure debugger

From the **Command Palette** (`kb(workbench.action.showCommands)`) and select **Debug: Open launch.json**, select the environment. This will generate a `launch.json` file. Works out of the box as expected for Node.js and other environments. May need some additional configuration for other languages. See [documentation](#) for more details.

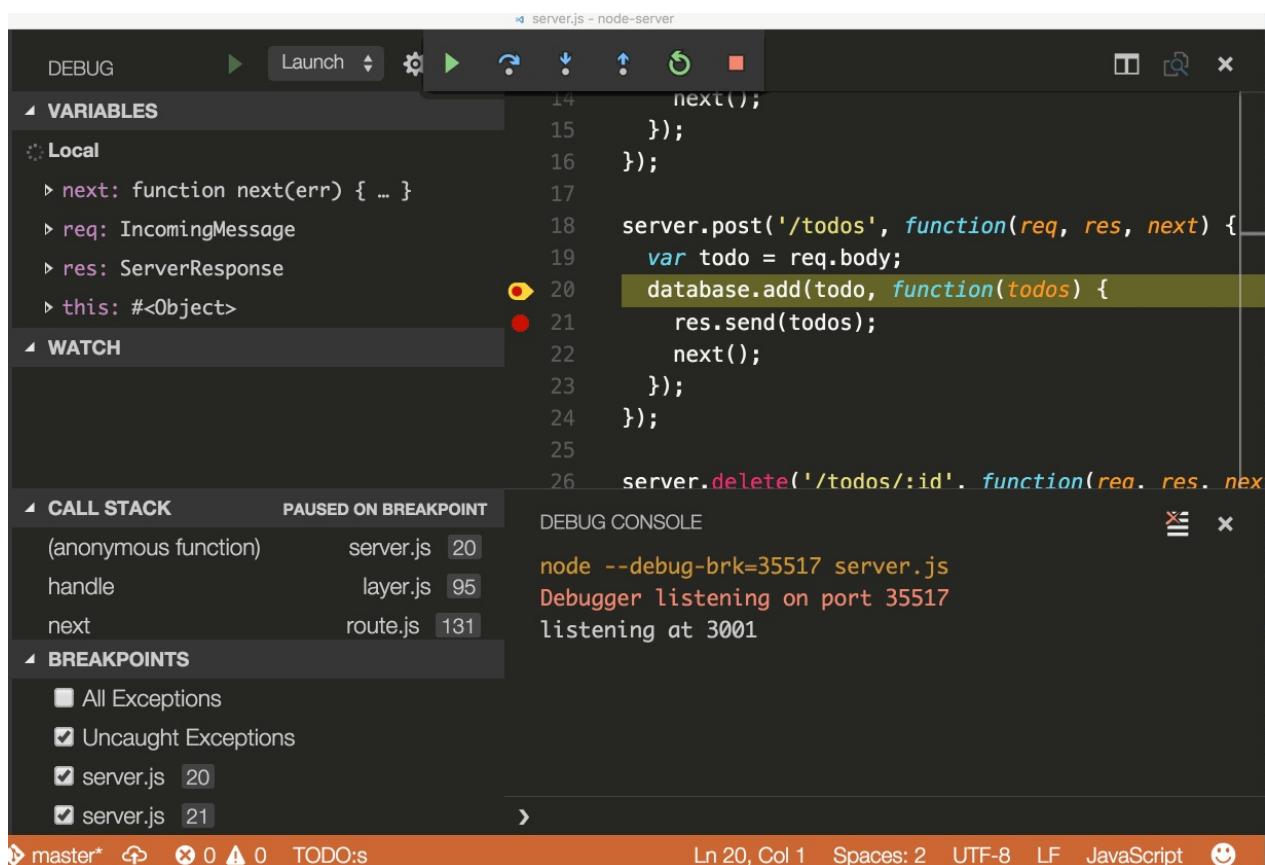
```
nd

ring a merge click the git icon and make changes in the diff view
git icon] (/media/git_icon.png)

Setup VS Code as default merge tool
```

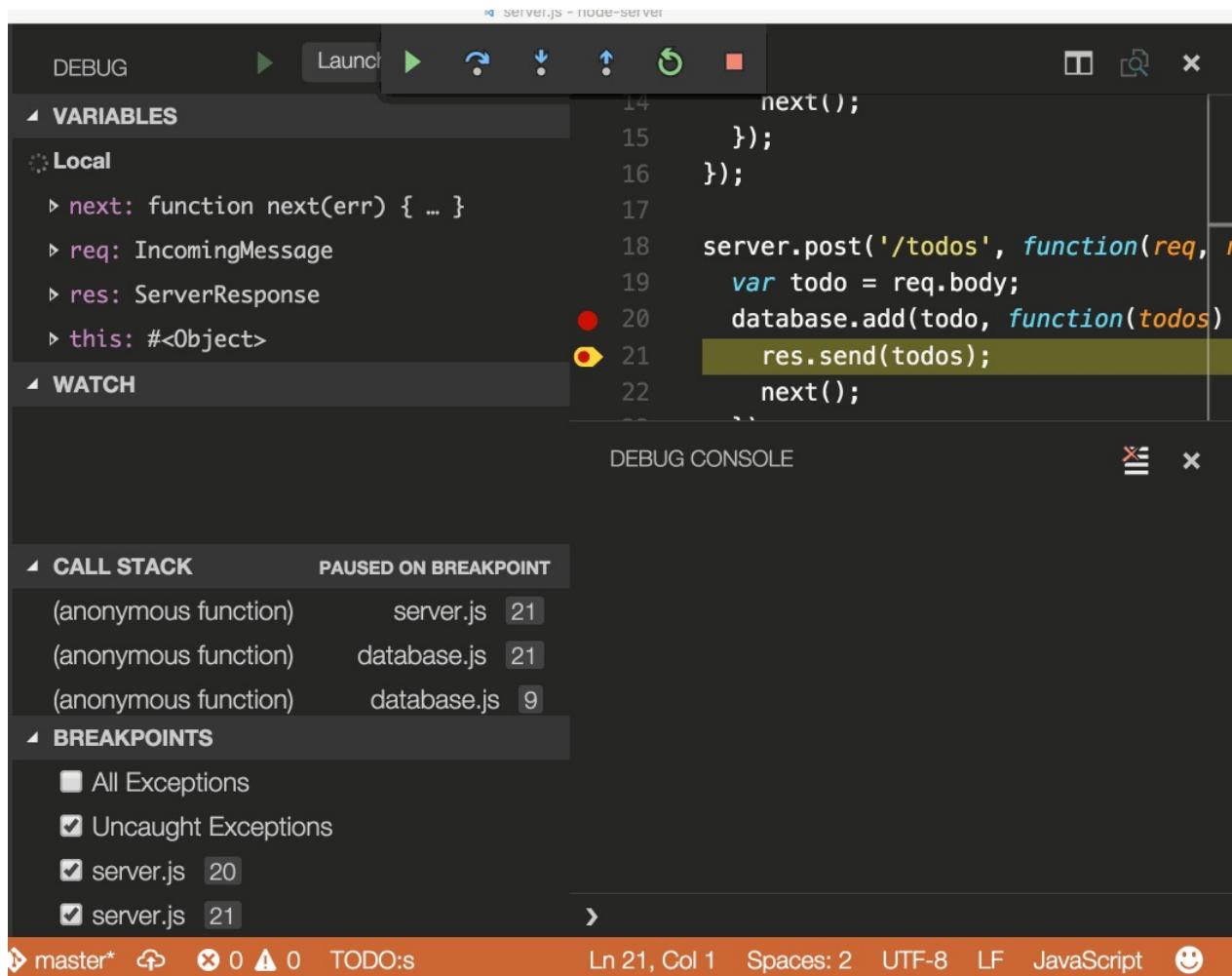
Breakpoints and stepping through

Place breakpoints next to the line number. Navigate forward with the Debug widget.



Data inspection

Inspect variables in the **Debug** panels and in the console.



Inline values

You can set `"debug.inlineValues": true` to see variable values inline in the debugger. This feature is experimental and disabled by default.

Task Runner

Auto detect tasks

Select **Tasks** from the top-level menu, run the command **Configure Tasks...**, then select the type of task you'd like to run. This will generate a `task.json` file with content like the following. See the [Tasks](#) documentation for more details.

```
{  
    // See https://go.microsoft.com/fwlink/?LinkId=733558  
    // for the documentation about the tasks.json format  
    "version": "2.0.0",  
    "tasks": [  
        {  
            "type": "npm",  
            "script": "install",  
            "group": {  
                "kind": "build",  
                "isDefault": true  
            }  
        }  
    ]  
}
```

There are occasionally issues with auto generation. Check out the documentation for getting things to work properly.

Run tasks from the Tasks menu

Select **Tasks** from the top-level menu, run the command **Run Task...**, and select the task you want to run. Terminate the running task by running the command **Terminate Task...**

The screenshot shows the VS Code interface with the file `tasks.json` open. The code defines a task named "install" with argument "install" and another task named "build" with arguments "run" and "dev". The status bar at the bottom shows "master*" and various status icons.

```

1  {
2      // See http://go.microsoft.com/fwlink/?LinkId=733558
3      // for the documentation about the tasks.json format
4      "version": "0.1.0",
5      "command": "npm",
6      "isShellCommand": true,
7      "showOutput": "always",
8      "suppressTaskName": true,
9      "tasks": [
10         {
11             "taskName": "install",
12             "args": ["install"]
13         },
14         {
15             "taskName": "build",
16             "args": ["run", "dev"]
17         }
18     ]
19 }

```

master* 🔍 ✖️ 0 ⚠️ 0 TODO:s Ln 9, Col 15 Tab Size: 4 UTF-8 LF JSON

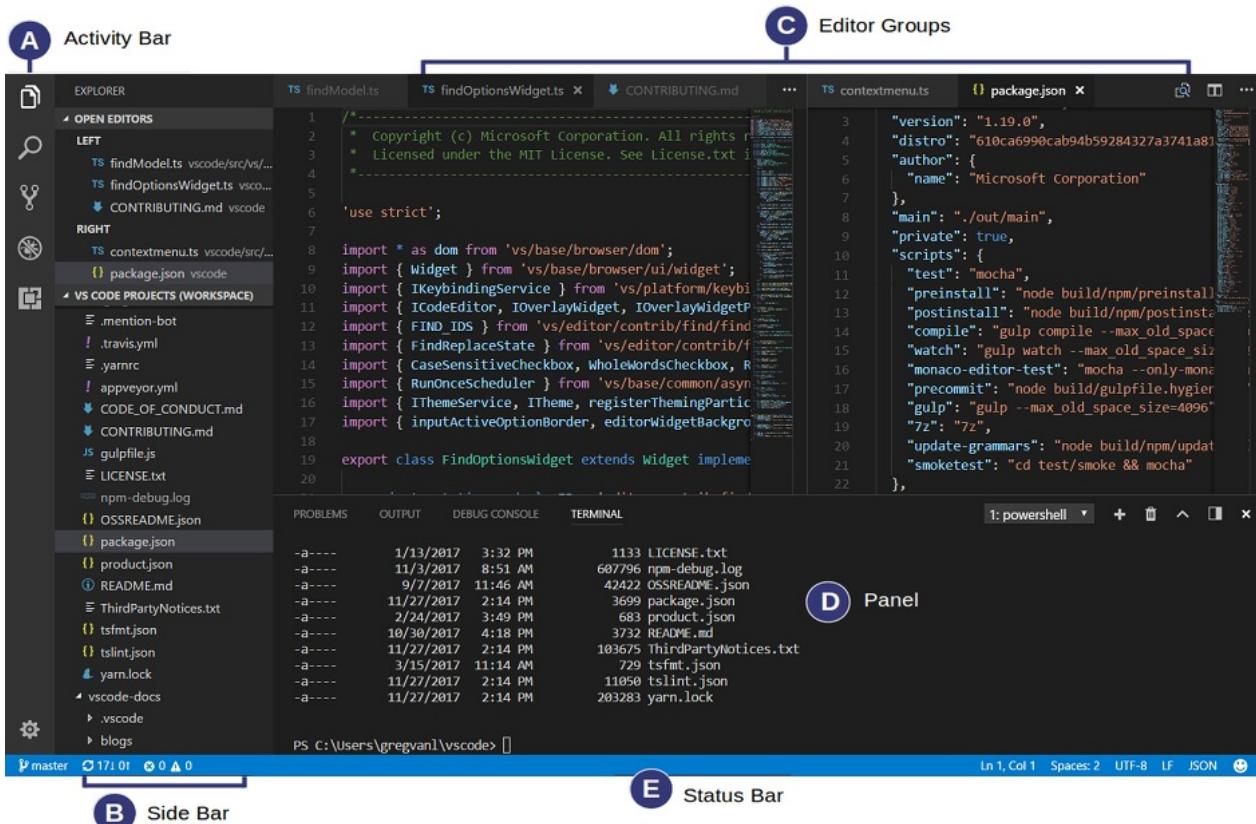
Insiders builds

The Visual Studio Code team uses the Insiders version to test the latest features and bug fixes of VS Code. You can also use the Insiders version by [downloading here](#).

- For Early Adopters - Insiders has the most recent code changes for users and extension authors to try out.
- Frequent Builds - New builds everyday with the latest bug fixes and features.
- Side-by-side install - Insiders installs next to the Stable build allowing you to use either independently.

User Interface

At its heart, Visual Studio Code is a code editor. Like many other code editors, VS Code adopts a common user interface and layout of an explorer on the left, showing all of the files and folders you have access to, and an editor on the right, showing the content of the files you have opened.



Basic Layout

VS Code comes with a simple and intuitive layout that maximizes the space provided for the editor while leaving ample room to browse and access the full context of your folder or project. The UI is divided into five areas:

- **Editor** - The main area to edit your files. You can open up to three editors side by side.
- **Side Bar** - Contains different views like the Explorer to assist you while working on your project.
- **Status Bar** - Information about the opened project and the files you edit.
- **Activity Bar** - Located on the far left-hand side, this lets you switch between views and gives you additional context-specific indicators, like the number of outgoing changes

when Git is enabled.

- **Panels** - You can display different panels below the editor region for output or debug information, errors and warnings, or an integrated terminal. Panel can also be moved to the right for more vertical space.

Each time you start VS Code, it opens up in the same state it was in when you last closed it. The folder, layout, and opened files are preserved.

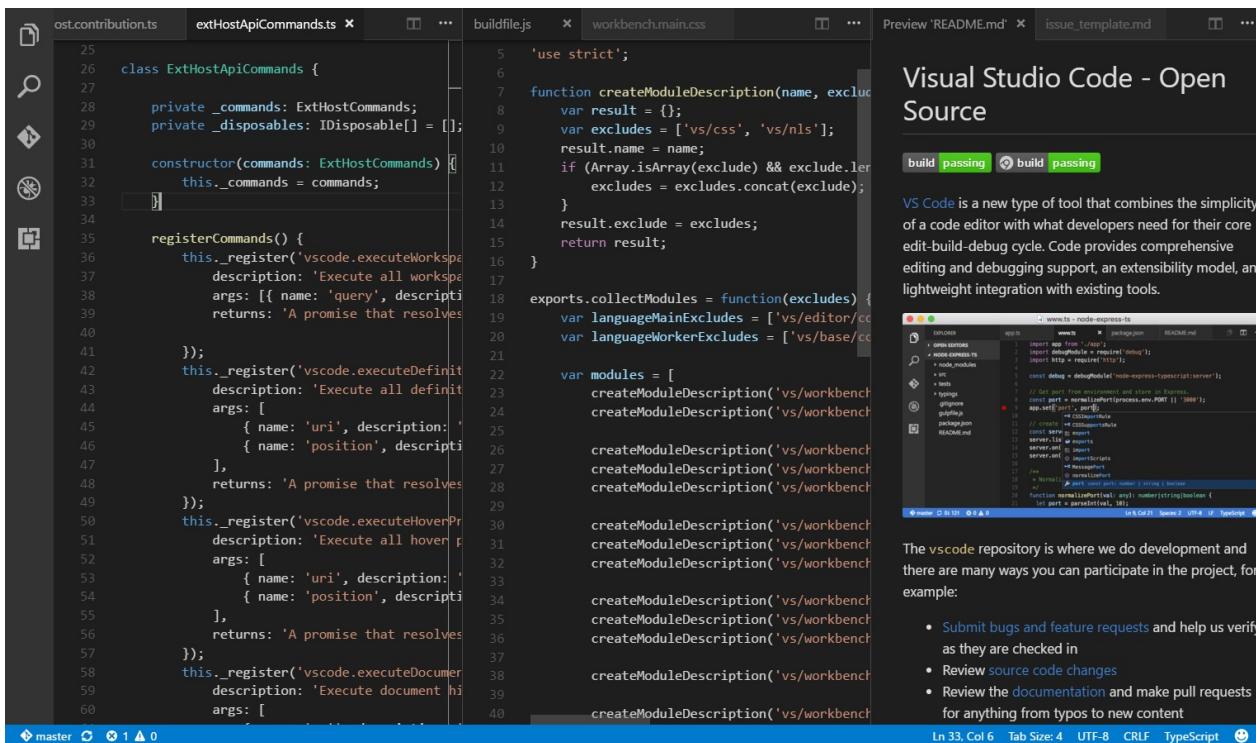
Open files in each editor are displayed with tabbed headers (Tabs) at the top of the editor region. To learn more about tabbed headers, see the [Tabs](#) section below.

Tip: You can move the Side Bar to the right hand side ([View > Move Side Bar Right](#)) or toggle its visibility (`kb(workbench.action.toggleSidebarVisibility)`).

Side by Side Editing

You can have up to three editors open side by side. If you already have one editor open, there are multiple ways of opening another editor to the side of the existing one:

- `kbstyle(Ctrl)` (Mac: `kbstyle(Cmd)`) click on a file in the Explorer.
- `kb(workbench.action.splitEditor)` to split the active editor into two.
- **Open to the Side** from the Explorer context menu on a file.
- Click the **Split Editor** button in the upper right of an editor.
- Drag and drop a file to the either side of the editor region.
- `kbstyle(Ctrl+Enter)` (Mac: `kbstyle(Cmd+Enter)`) in the **Quick Open** (`kb(workbench.action.quickOpen)`) file list.



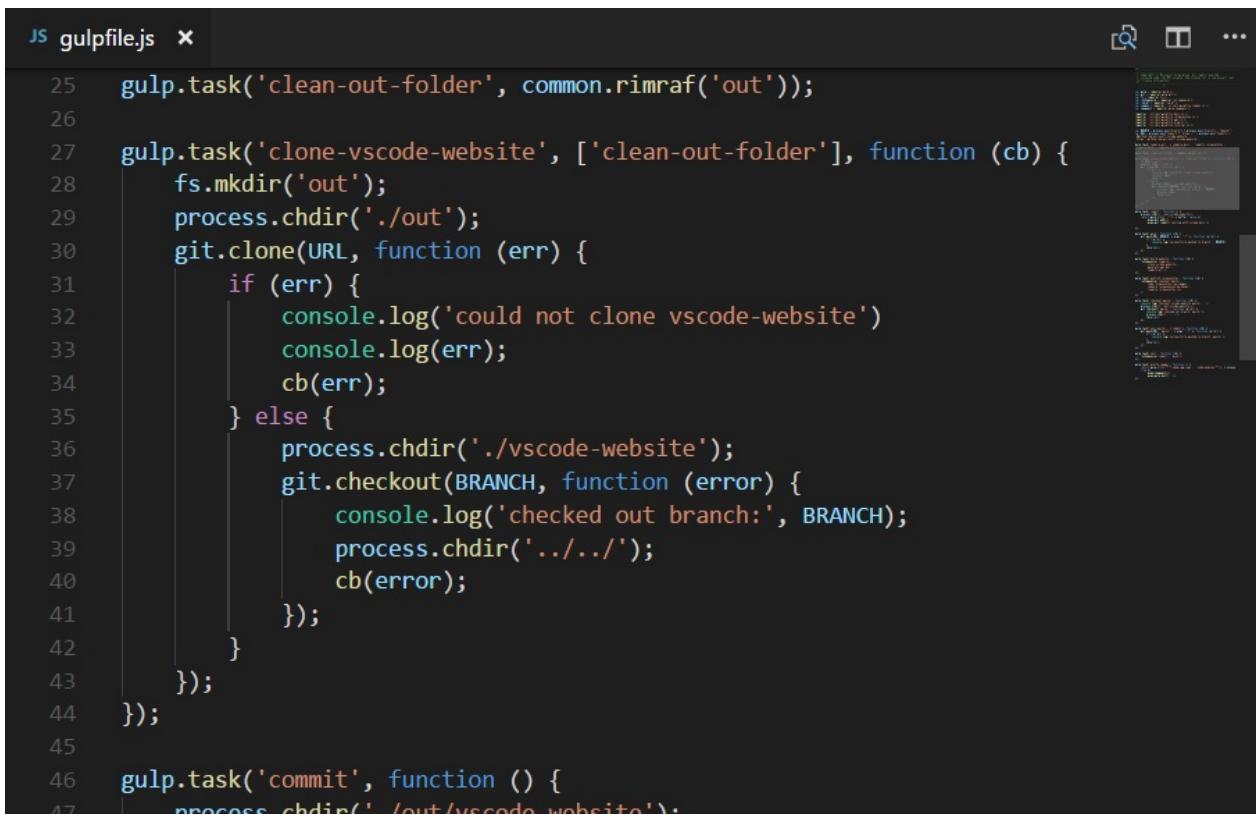
Whenever you open another file, the editor that is active will display the content of that file. So if you have two editors side by side and you want to open file 'foo.cs' into the right hand editor, make sure that editor is active (by clicking inside it) before opening file 'foo.cs'.

When you have more than one editor open you can switch between them quickly by holding the `kbstyle(Ctrl)` (Mac: `kbstyle('Cmd')`) key and pressing `kbstyle(1)`, `kbstyle(2)`, or `kbstyle(3)`.

Tip: You can resize editors and reorder them. Drag and drop the editor title area to reposition or resize the editor.

Minimap - outline view

A Minimap (outline view) gives you a high level overview of your source code which is very useful for quick navigation and code understanding. A file's minimap is shown in the right side of the editor. You can click or drag the shaded area to quickly jump to different sections of your file.



```

JS gulpfile.js ✘

25  gulp.task('clean-out-folder', common.rimraf('out'));
26
27  gulp.task('clone-vscode-website', ['clean-out-folder'], function (cb) {
28    fs.mkdir('out');
29    process.chdir('./out');
30    git.clone(URL, function (err) {
31      if (err) {
32        console.log('could not clone vscode-website')
33        console.log(err);
34        cb(err);
35      } else {
36        process.chdir('./vscode-website');
37        git.checkout(BRANCH, function (error) {
38          console.log('checked out branch:', BRANCH);
39          process.chdir('../..');
40          cb(error);
41        });
42      }
43    });
44  });
45
46  gulp.task('commit', function () {
47    process.chdir('../out/vscode-website').

```

If you would like to disable minimap, you can set `"editor.minimap.enabled": false` in your user or workspace [settings](#).

Indent Guides

The image above also shows indentation guides (vertical lines) which help you quickly see matching indent levels. If you would like to disable indent guides, you can set

`"editor.renderIndentGuides": false` in your user or workspace [settings](#).

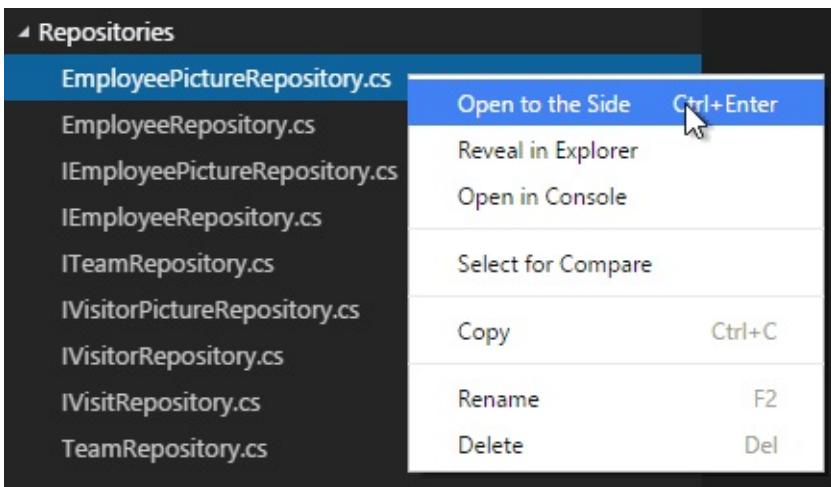
Explorer

The Explorer is used to browse, open, and manage all of the files and folders in your project. VS Code is file and folder based - you can get started immediately by opening a file or folder in VS Code.

After opening a folder in VS Code, the contents of the folder are shown in the Explorer. You can do many things from here:

- Create, delete, and rename files and folders.
- Move files and folders with drag and drop.
- Use the context menu to explore all options.

Tip: You can drag and drop files into the Explorer from outside VS Code to copy them.



VS Code works very well with other tools that you might use, especially command-line tools. If you want to run a command-line tool in the context of the folder you currently have open in VS Code, right-click the folder and select **Open in Command Prompt** (or **Open in Terminal** on Mac or Linux).

You can also navigate to the location of a file or folder in the native Explorer by right-clicking on a file or folder and selecting **Reveal in Explorer** (or **Reveal in Finder** on the Mac or **Open Containing Folder** on Linux).

Tip: Type `kb(workbench.action.quickopen)` (**Quick Open**) to quickly search and open a file by its name.

By default, VS Code excludes some folders from the Explorer (for example, `.git`). Use the `files.exclude` setting to configure rules for hiding files and folders from the Explorer.

Tip: This is really useful to hide derived resources files, like `*.meta` in Unity, or `*.js` in a TypeScript project. For Unity to exclude the `*.cs.meta` files, the pattern to choose would be: `"**/*.cs.meta": true`. For TypeScript, you can exclude generated JavaScript for TypeScript files with: `"**/*.js": {"when": "$(basename).ts"}`.

Open Editors

At the top of the Explorer is a section labeled **OPEN EDITORS**. This is a list of active files or previews. These are files you previously opened in VS Code that you're working on. For example, a file will be listed in the **OPEN EDITORS** section if you:

- Make a change to a file.
- Double-click a file's header.
- Double-click a file in the Explorer.
- Open a file that is not part of the current folder.

Just click an item in the **OPEN EDITORS** section, and it becomes active in VS Code.

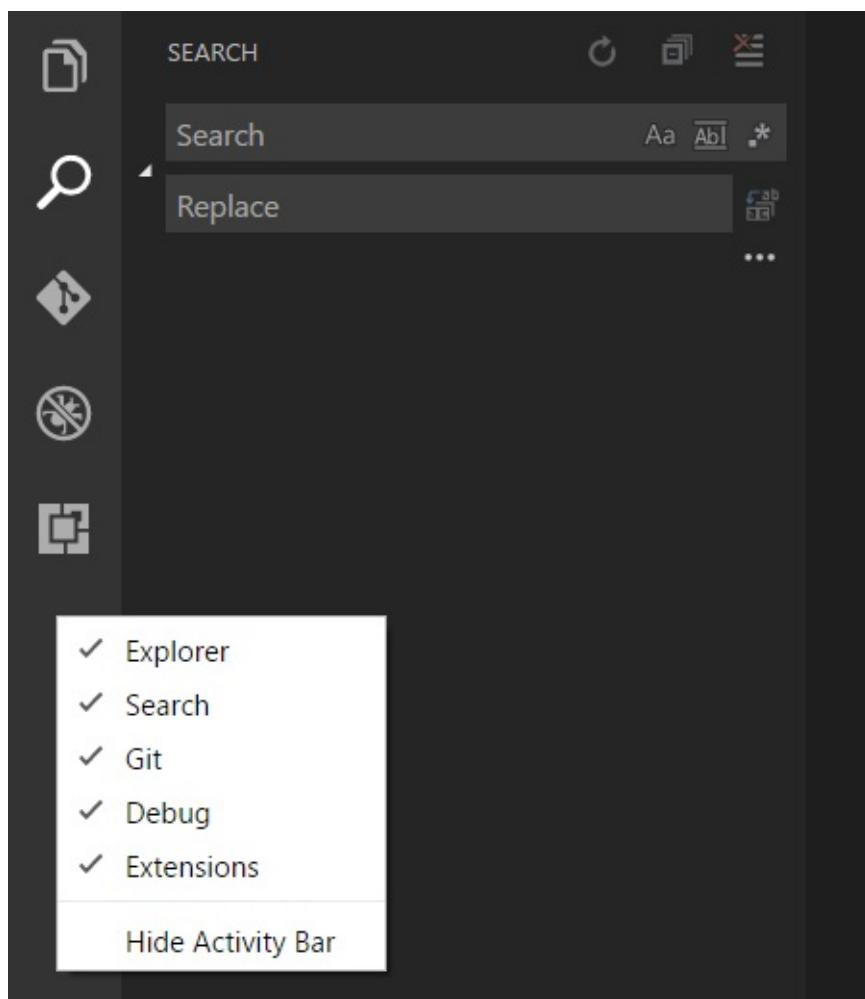
Once you are done with your task, you can remove files individually from the **OPEN EDITORS** section, or you can remove all files by using the **View: Close All Editors** or **View: Close All Editors in Group** actions.

Views and the Activity Bar

The File Explorer is just one of the Views available in VS Code. There are also Views for:

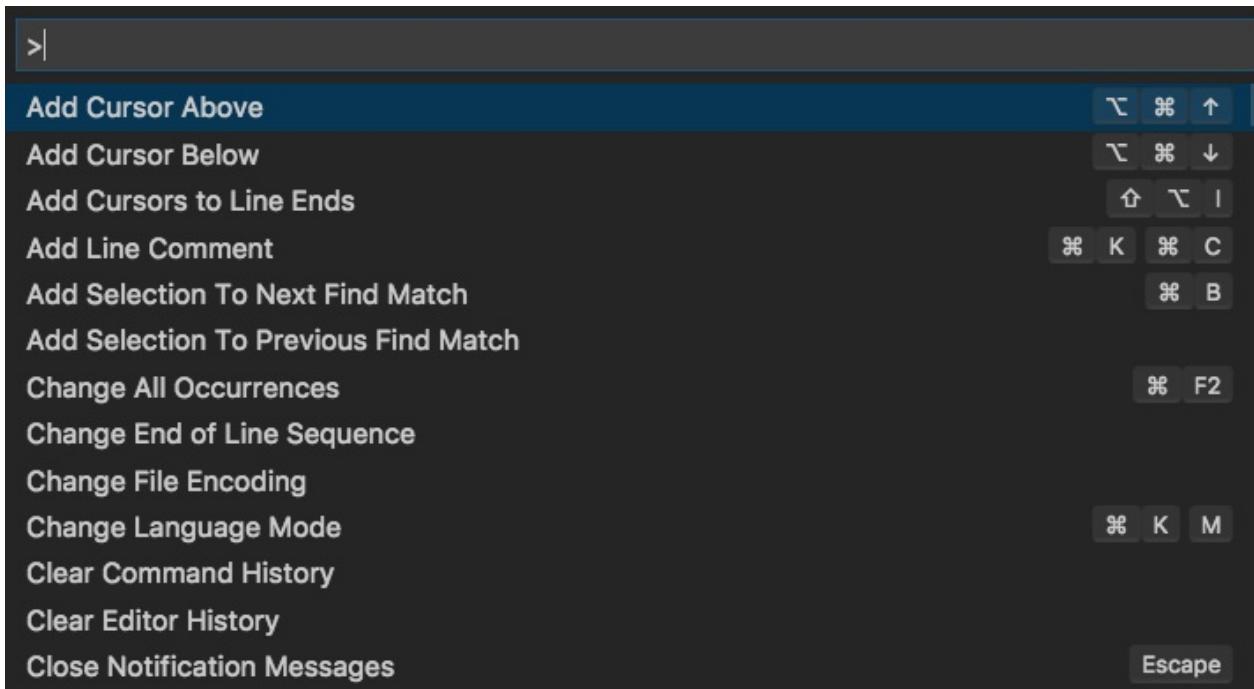
- **Search** - Provides global search and replace across your open folder.
- **Source Control** - VS Code includes Git source control by default.
- **Debug** - VS Code's Debug View displays variables, call stacks, and breakpoints.
- **Extensions** - Install and manage your extensions within VS Code.

The **Activity Bar** on the left lets you quickly switch between Views. You can also reorder Views by dragging and dropping them on the **Activity Bar** or remove a View entirely (right click **Hide from Activity Bar**).



Command Palette

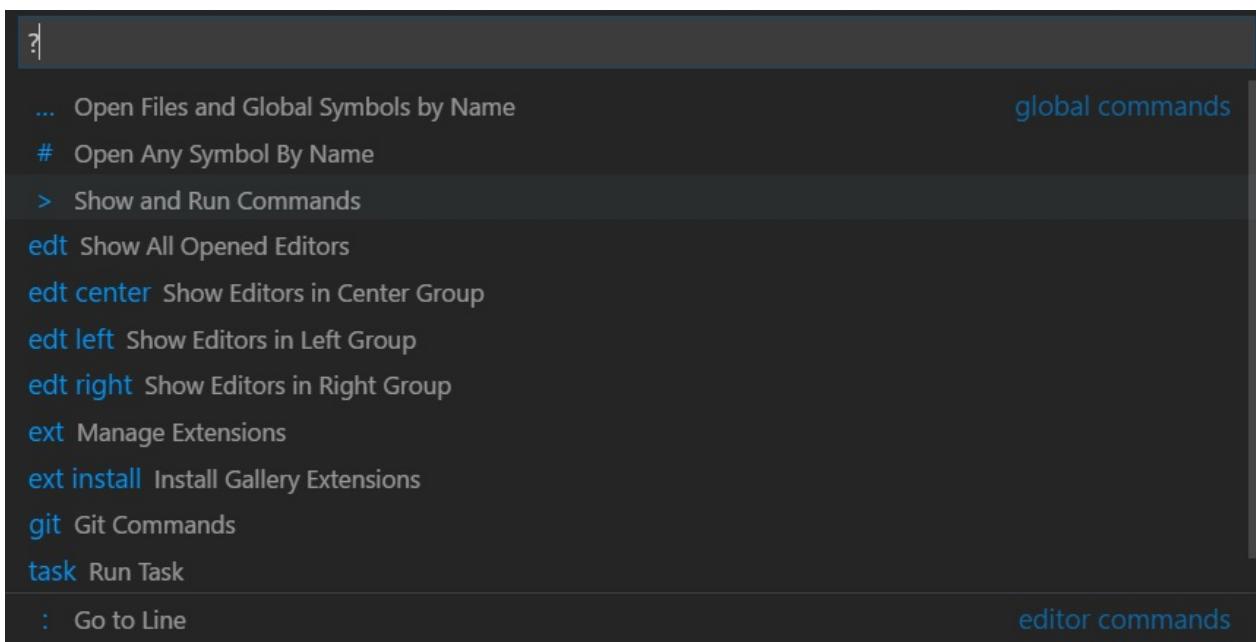
VS Code is equally accessible from the keyboard. The most important key combination to know is `kb(workbench.action.showCommands)`, which brings up the **Command Palette**. From here, you have access to all of the functionality of VS Code, including keyboard shortcuts for the most common operations.



The **Command Palette** provides access to many commands. You can execute editor commands, open files, search for symbols, and see a quick outline of a file, all using the same interactive window. Here are a few tips:

- `kb(workbench.action.quickOpen)` will let you navigate to any file or symbol by typing its name
- `kb(workbench.action.openPreviousRecentlyUsedEditorInGroup)` will cycle you through the last set of files opened
- `kb(workbench.action.showCommands)` will bring you directly to the editor commands
- `kb(workbench.action.gotoSymbol)` will let you navigate to a specific symbol in a file
- `kb(workbench.action.gotoLine)` will let you navigate to a specific line in a file

Type `?` into the input field to get a list of available commands you can execute from here:



Configuring the Editor

VS Code gives you many options to configure the editor. From the **View** menu, you can hide or toggle various parts of the user interface, such as the **Side Bar**, **Status Bar**, and **Activity Bar**.

Hide the Menu Bar (Windows, Linux)

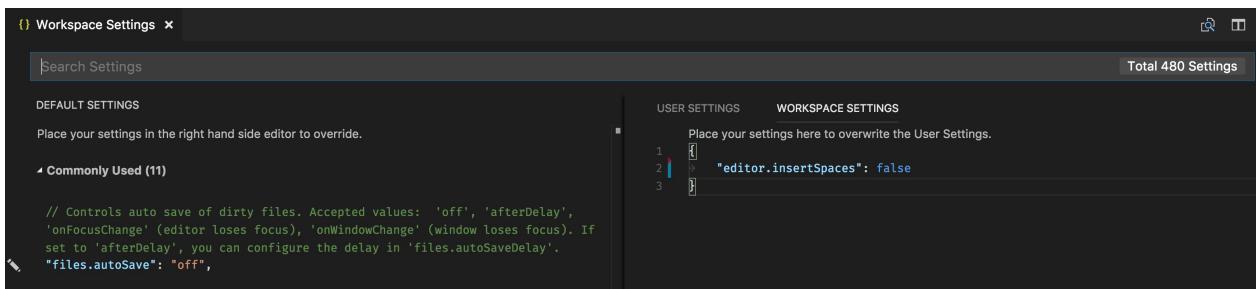
You can hide the Menu Bar on Windows and Linux with the **View > Toggle Menu Bar** command. You can still access the Menu Bar by pressing the `kbstyle(Alt)` key (`window.menuBarVisibility` setting).

Settings

Most editor configurations are kept in settings which can be modified directly. You can set options globally through user settings or per project/folder through workspace settings. Settings values are kept in a `settings.json` file.

- Select **File > Preferences > Settings** (or press `kb(workbench.action.showCommands)`), type `user` and press `kbstyle(Enter)` to edit the `user settings.json` file.
- To edit workspace settings, select **File > Preferences > Settings** and select the **WORKSPACE SETTINGS** Tab (or press `kb(workbench.action.showCommands)`), type `worksp` and press `kbstyle(Enter)` to edit the `workspace settings.json` file.

Note for Mac users: The **Preferences** menu is under **Code** not **File**. For example, **Code > Preferences > Settings**.



You will see the VS Code [Default Settings](#) in the left window and your editable `settings.json` on the right. You can easily filter settings in the `Default Settings` using the search box at the top. Copy a setting over to the editable `settings.json` on the right by clicking on the edit icon to the left of the setting. Settings with a fixed set values allow you to pick a value as part of their edit icon menu.

After editing your settings, type `kb(workbench.action.files.save)` to save your changes. The changes will take effect immediately.

Note: Workspace settings will override User settings and are useful for sharing project specific settings across a team.

Zen Mode

Zen Mode lets you focus on your code by hiding all UI except the editor (no Activity Bar, Status Bar, Side Bar and Panel) and going to full screen. Zen mode can be toggled using **View** menu, **Command Palette** or by the shortcut `kb(workbench.action.toggleZenMode)`. Double `kbstyle(Esc)` exits Zen Mode. The transition to full screen can be disabled via `zenMode.fullScreen`. Zen Mode can be further tuned by the following settings: `zenMode.hideStatusBar`, `zenMode.hideTabs`, `zenMode.fullScreen` and `zenMode.restore`.

Tabs

Visual Studio Code shows open items with Tabs (tabbed headings) in the title area above the editor.

When you open a file, a new Tab is added for that file.

The screenshot shows the Visual Studio Code interface. The left sidebar is the Explorer view, displaying a tree structure of files under 'VS CODE'. The 'pointerHandler.ts' file is selected. The main editor area contains the following TypeScript code:

```
5  'use strict';
6
7  import {IDisposable} from 'vs/base/common/lifecycle';
8  import * as dom from 'vs/base/browser/dom';
9  import {StandardMouseEvent} from 'vs/base/browser/mouseEvent';
10 import {EventType, Gesture, GestureEvent} from 'vs/base/browser/touch';
11 import {IScrollEvent} from 'vs/editor/common/editorCommon';
12 import {MouseHandler, IPointerHandlerHelper} from 'vs/editor/browser/control';
13 import {IViewController} from 'vs/editor/browser/editorBrowser';
14 import {ViewContext} from 'vs/editor/common/view/viewContext';
15
16 interface IThrottledGestureEvent {
17     translationX: number;
18     translationY: number;
19 }
20
21 var gestureChangeEventMerger = (lastEvent:IThrottledGestureEvent, currentEvent:IThrottledGestureEvent) => {
22     var r = {
23         translationY: currentEvent.translationY,
24         translationX: currentEvent.translationX
25     };
26     if (lastEvent) {
27         r.translationY += lastEvent.translationY;
28         r.translationX += lastEvent.translationX;
29     }
30 }
```

Tabs let you quickly navigate between items and you can Drag and Drop Tabs to reorder them.

When you have more open items than can fit in the title area, you can use the **Show Opened Editors** command (available through the  More button) to display a dropdown of tabbed items.

If you don't want to use Tabs, you can disable the feature by setting the `workbench.editor.showTabs` [setting](#) to false:

```
"workbench.editor.showTabs": false
```

See the section below to optimize VS Code when [working without Tabs](#).

Tab ordering

By default, new Tabs are added to the right of the existing Tabs but you can control where you'd like new Tabs to appear with the `workbench.editor.openPositioning` setting.

For example, you might like new tabbed items to appear on the left:

```
"workbench.editor.openPositioning": "left"
```

Preview mode

When you single-click or select a file in the Explorer, it is shown in a preview mode and reuses an existing Tab. This is useful if you are quickly browsing files and don't want every visited file to have its own Tab. When you start editing the file or use double-click to open the file from the Explorer, a new Tab is dedicated to that file.

Preview mode is indicated by **italics** in the Tab heading:

```

keyboardHandler.ts    pointerHandler.ts ×    mouseTargets.ts    mouseHandler.ts
5  'use strict';
6
7  import {IDisposable} from 'vs/base/common/lifecycle';

```

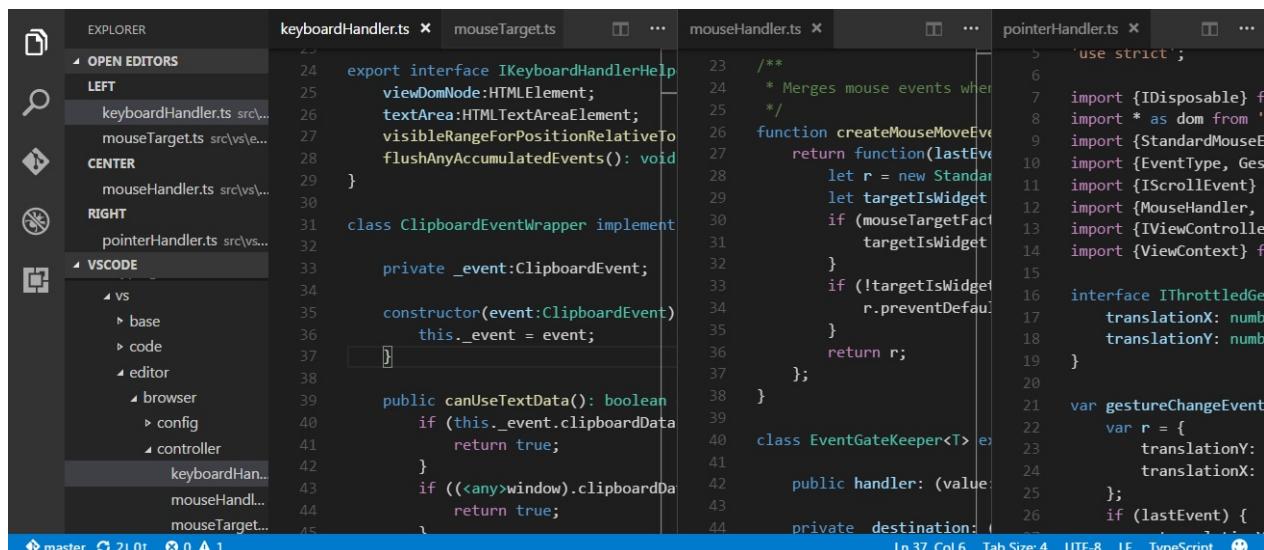
If you'd prefer to not use preview mode and always create a new Tab, you can control the behavior with these settings:

- `workbench.editor.enablePreview` to globally enable or disable preview editors
- `workbench.editor.enablePreviewFromQuickopen` to enable or disable preview editors when opened from **Quick Open**

Editor Groups

When you split an editor (using the **Split Editor** or **Open to the Side** commands), a new editor region is created which can hold a group of items. VS Code allows up to three editor groups which are designated **LEFT**, **CENTER**, and **RIGHT**.

You can see these clearly in the **OPEN EDITORS** section at the top of the Explorer view:



You can Drag and Drop editor groups on the workbench, move individual Tabs between groups and quickly close entire groups (**Close All**).

Note: VS Code uses editor groups whether or not you have enabled Tabs. Without Tabs, editor groups are a stack of your open items with the most recently selected item visible in the editor pane.

Horizontal layout

By default, editor groups are laid out in three vertical columns. If you prefer, you can change the layout to be three horizontal rows with editor groups designated **TOP**, **CENTER**, and **BOTTOM**.

You can toggle the editor group layout between vertical and horizontal with:

- **View > Toggle Editor Group Layout** menu
- **View: Toggle Editor Group Vertical/ Layout** command in the **Command Palette**
(`kb(workbench.action.showCommands)`)
- Toggle button in the **OPEN EDITORS** tool bar
- `kb(workbench.action.toggleEditorGroupLayout)` keyboard shortcut

Keyboard Shortcuts

Here are some handy keyboard shortcuts to quickly navigate between editors and editor groups.

If you'd like to modify the default keyboard shortcuts, see [Key Bindings](#) for details.

- `kb(workbench.action.nextEditor)` go to the right editor.
- `kb(workbench.action.previousEditor)` go to the left editor.
- `kb(workbench.action.openNextRecentlyUsedEditorInGroup)` open the next editor in the editor group MRU list.
- `kb(workbench.action.openPreviousRecentlyUsedEditorInGroup)` open the previous editor in the editor group MRU list.
- `kb(workbench.action.focusFirstEditorGroup)` go to the leftmost editor group.
- `kb(workbench.action.focusSecondEditorGroup)` go to the center editor group.
- `kb(workbench.action.focusThirdEditorGroup)` go to the rightmost editor group.
- `kb(workbench.action.focusPreviousGroup)` go to the previous editor group.
- `kb(workbench.action.focusNextGroup)` go to the next editor group.
- `kb(workbench.action.closeActiveEditor)` close the active editor.
- `kb(workbench.action.closeEditorsInGroup)` close all editors in the editor group.
- `kb(workbench.action.closeAllEditors)` close all editors.

Working without Tabs

If you prefer not to use Tabs (tabbed headings), you can disable Tabs (tabbed headings) entirely by setting `workbench.editor.showTabs` to false.

Disable Preview mode

Without Tabs, the **OPEN EDITORS** section of the File Explorer is a quick way to do file navigation. With [preview editor mode](#), files are not added to the **OPEN EDITOR** list nor editor group on single-click open. You can disable this feature through the `workbench.editor.enablePreview` and `workbench.editor.enablePreviewFromQuickOpen` settings.

Ctrl+Tab to navigate in entire editor history

You can change keybindings for `kbstyle(Ctrl+Tab)` to show you a list of all opened editors from the history independent from the active editor group.

Edit your [keybindings](#) and add the following:

```
{ "key": "ctrl+tab", "command": "workbench.action.openPreviousEditorFromHistory" },
{ "key": "ctrl+tab", "command": "workbench.action.quickOpenNavigateNext", "when": "inQuickOpen" },
```

Close an entire group instead of a single editor

If you liked the behavior of VS Code closing an entire group when closing one editor, you can bind the following in your [keybindings](#).

Mac:

```
{ "key": "cmd+w", "command": "workbench.action.closeEditorsInGroup" }
```

Windows/Linux:

```
{ "key": "ctrl+w", "command": "workbench.action.closeEditorsInGroup" }
```

Window Management

VS Code has some options to control how windows (instances) should be opened or restored between sessions.

The settings `window.openFoldersInNewWindow` and `window.openFilesInNewWindow` are provided to configure opening new windows or reusing the last active window for files or folders and possible values are `default`, `on` and `off`.

If configured to be `default`, we will make the best guess about reusing a window or not based on the context from where the open request was made. Flip this to `on` or `off` to always behave the same. For example, if you feel that picking a file or folder from the **File** menu should always open into a new window, set this to `on`.

Note: There can still be cases where this setting is ignored (for example, when using the `-new-window` or `-reuse-window` command line option).

The `window.restoreWindows` setting tells VS Code how to restore the opened windows of your previous session. By default, VS Code will reopen the last opened window you worked on (setting: `one`). Change this setting to `none` to never reopen any windows and always start with an empty VS Code instance. Change it to `all` to restore all windows you worked on during your previous session or `folders` to only restore windows that had folders opened.

Next Steps

Now that you know the overall layout of VS Code, start to customize the editor to how you like to work by looking at the following topics:

- [Changing the Theme](#) - Set a Color and/or File Icon theme to your preference.

Common Questions

Q: How can I change the color of the indent guides?

A: The indent guide colors are customizable as are most VS Code UI elements. To [customize](#) the indent guides color for your active color theme, use the `workbench.colorCustomizations` [setting](#) and modify the `editorIndentGuide.background` value.

For example, to make the indent guides bright blue, add the following to your

`settings.json` :

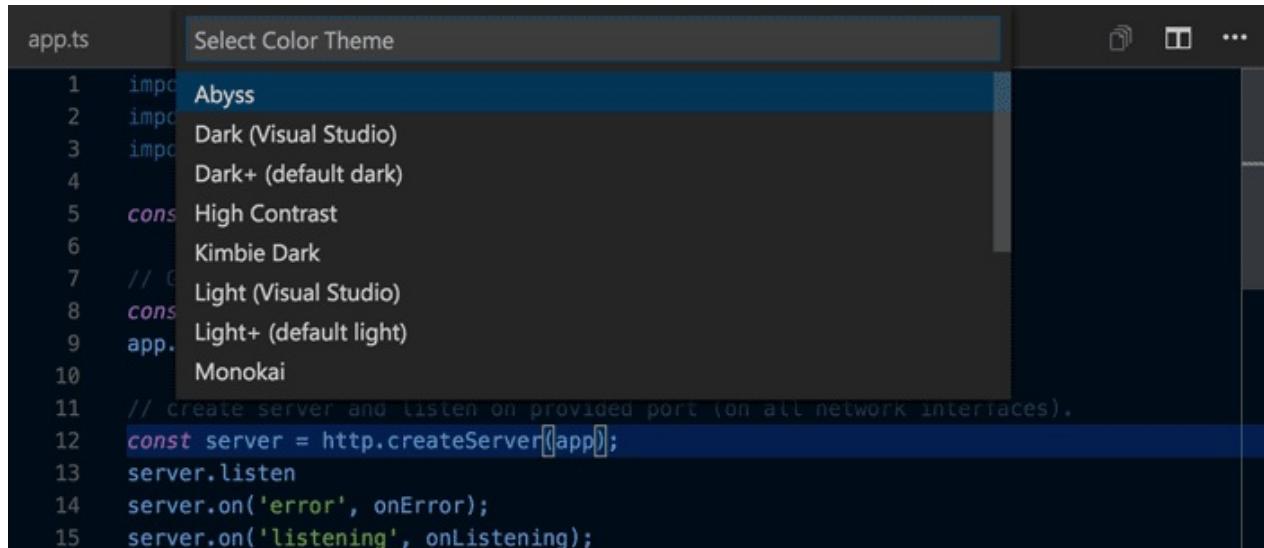
```
"workbench.colorCustomizations": {  
    "editorIndentGuide.background": "#0000ff"  
}
```

Q: Can I hide the OPEN EDITORS section in the Explorer?

A: Yes, you can hide the **OPEN EDITORS** list with the `explorer.openEditors.visible setting`, which declares how many items to display before a scroll bar appears. Setting `"explorer.openEditors.visible": 0` will hide **OPEN EDITORS** when you have an open folder. The list will still be displayed if you are using VS Code to view loose files.

Color Themes

Color themes let you modify VS Code's background, text, and language syntax colorization to suit your preferences and work environment. VS Code supports light, dark and high contrast themes.



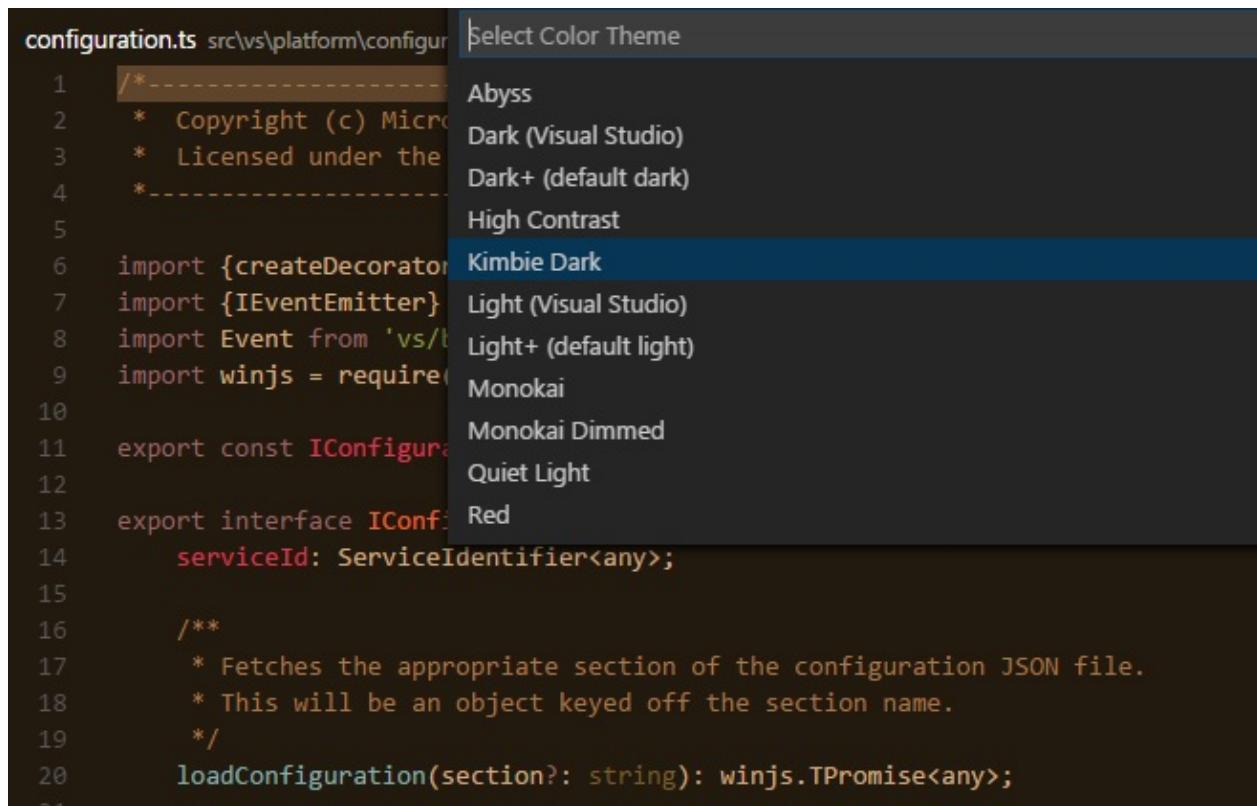
Selecting the Color Theme

The current color theme is configured in the [settings](#).

```
// Specifies the color theme used in the workbench.
"workbench.colorTheme": "Default Dark+"
}
```

However, there is no need to edit the settings directly. It's easier to use the Color Theme Picker to preview and select a theme.

1. Open the Color Theme picker with **File > Preferences > Color Theme**. (**Code > Preferences > Color Theme** on Mac)
2. Use the cursor keys to preview the colors of the theme.
3. Select the theme you want and hit `kbstyle(Enter)`.



Tip: By default, the theme is configured in the user settings and applies to all workspaces. But you can also configure a workspace specific theme. To do so, set a theme in the workspace settings.

Color Themes from the Marketplace

There are several out-of-the-box color themes in VS Code for you to try.

Many more themes have been uploaded to the VS Code [Extension Marketplace](#) by the community. If you find one you want to use, install it and restart VS Code and the new theme will be available.

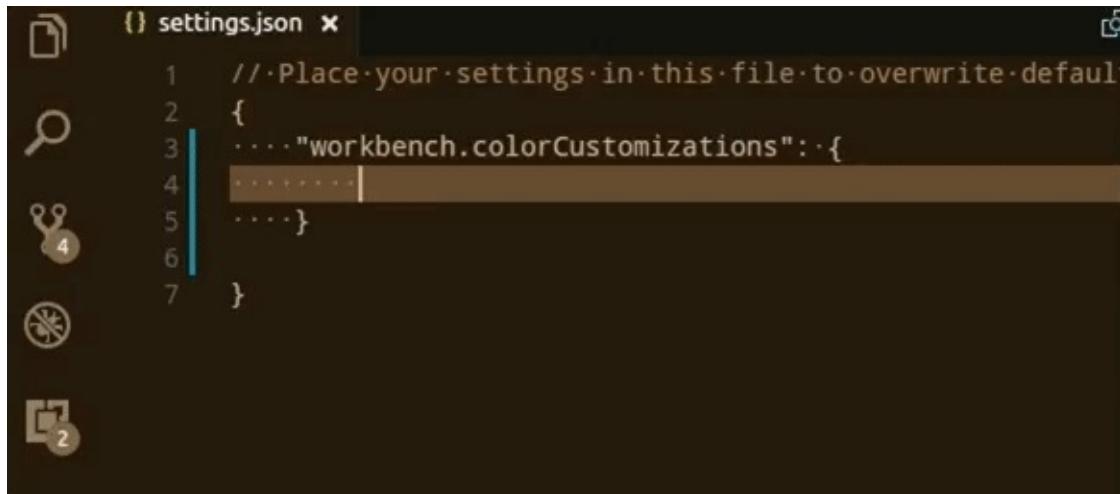
Tip: To search for themes, type 'theme' in the Extensions view
(`kb(workbench.view.extensions)`) search box.

You can also browse the [VS Code Marketplace](#) site directly to find available themes.

Customize a Color Theme

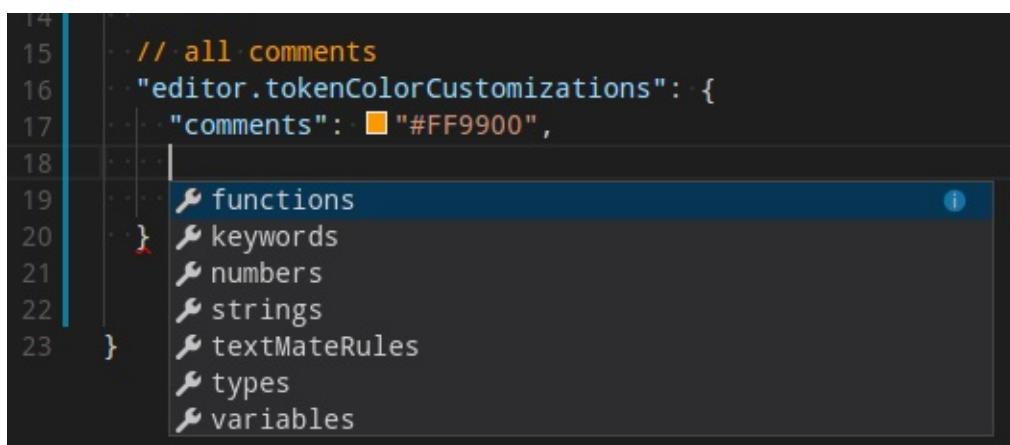
Note: Supported on VS Code version 1.12 or higher.

You can also customize your active color theme with the `workbench.colorCustomizations` user setting. You can set the colors of VS Code UI elements such as list & trees (File Explorer, suggestions widget), diff editor, Activity Bar, notifications, scroll bar, split view, buttons and more.



You can use IntelliSense while setting `workbench.colorCustomizations` values or, for a list of all customizable colors, see the [Theme Color Reference](#).

In VS Code version 1.15 or higher, you can also tune the syntax highlighting colors using the `editor.tokenColorCustomizations` setting:



A pre-configured set of syntax tokens ('comments', 'strings', ...) is available for the most common constructs. If you want more, you can do so by directly specifying TextMate theme color rules:

```

15  "editor.tokenColorCustomizations": {
16    "textMateRules": [
17      {
18        "scope": "support.type.property-name.json",
19        "settings": {
20          "foreground": "#7FB785"
21        }
22      }
23    ]
24  }

```

Note: Directly configuring TextMate rules is an advanced skill as you need to understand on how TextMate grammars work. Go [here](#) for more information

Using existing TextMate Themes

You can add existing TextMate color themes (.tmTheme) to VS Code. For example, the [ColorSublime](#) site has hundreds of TextMate themes available. See the [Adding a new Theme](#) topic in our Extension Authoring section to learn more.

Icon Themes

File icon themes can be contributed by extensions and selected by users as their favorite set of file icons. File icons are shown in the File Explorer and tabbed headings.

Selecting the File Icon Theme

The current File Icon theme is persisted in your user [settings](#).

```

// Specifies the icon theme used in the workbench.
"workbench.iconTheme": null
}

```

There is no need to edit the `settings.json` file directly. It is better to use the File Icon Theme picker to preview and select a theme.

1. Open the Icon Theme picker with **File > Preferences > File Icon Theme**. (**Code > Preferences > File Icon Theme** on Mac)
2. Use the cursor keys to preview the icons of the theme.
3. Select the theme you want and hit `kbstyle(Enter)`.

By default, no file icon set is configured, therefore the File Explorer shows no icons. Once an icon theme is selected, the selected theme will be remembered and set again when VS Code is started the next time .

VS code ships with two icon themes; **Minimal** and **Seti**. To install more icon themes, select the **Find more in the Marketplace...** item in the icon theme picker.

You can also browse the [VS Code Marketplace](#) site directly to find available themes.

Creating your own File Icon Theme

You can create your own File Icon Theme from icons (preferably SVG), see the [Adding a new Icon Theme](#) topic in our Extension Authoring section for details.

Next Steps

Themes are just one way to customize VS Code. If you'd like to learn more about VS Code customization and extensibility, try these topics:

- [Settings](#) - Learn how to configure VS Code to your preferences through user and workspace settings.
- [Snippets](#) - Add additional snippets to your favorite language.
- [Extending Visual Studio Code](#) - Learn about other ways to extend VS Code.
- [Themes, Snippets, and Colorizers](#) - You can package themes, snippets and language colorizers for use in VS Code.

User and Workspace Settings

It is easy to configure VS Code to your liking through settings. Nearly every part of VS Code's editor, user interface, and functional behavior has options you can modify.

VS Code provides two different scopes for settings:

- **User** These settings apply globally to any instance of VS Code you open
- **Workspace** These settings are stored inside your workspace in a `.vscode` folder and only apply when the workspace is opened. Settings defined on this scope override the user scope.

Creating User and Workspace Settings

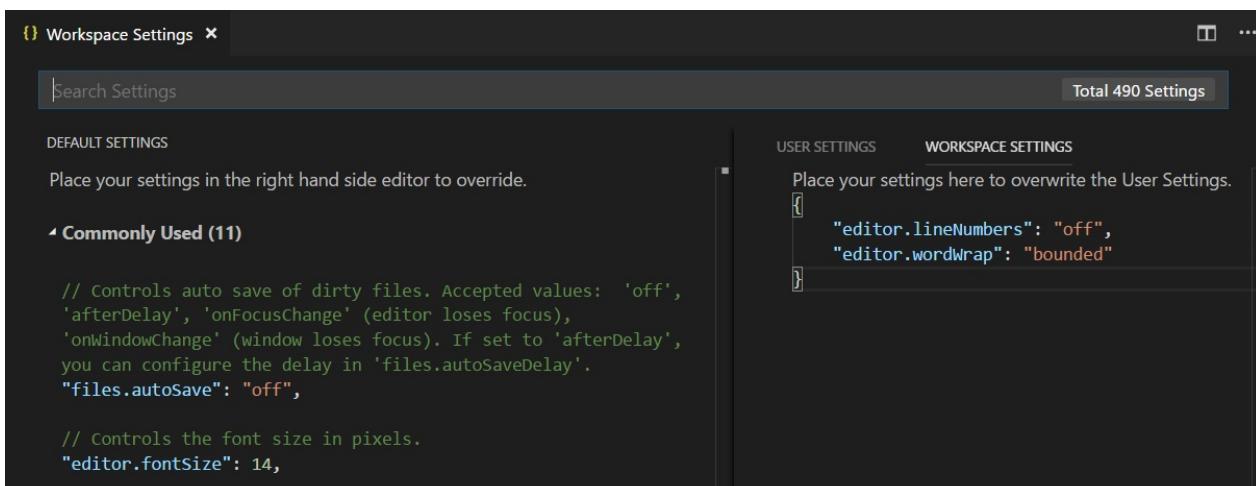
To get to the user and workspace settings:

- On a Windows computer, click **File > Preferences > Settings**
- On a Mac, click **Code > Preferences > Settings**

You are provided with a list of Default Settings. Copy any setting that you want to change to the appropriate `settings.json` file. The tabs on the right let you switch quickly between the user and workspace settings files.

You can also open the user and workspace settings from the **Command Palette** (`kb(workbench.action.showCommands)`) with **Preferences: Open User Settings** and **Preferences: Open Workspace Settings** or use the keyboard shortcut (`kb(workbench.action.openGlobalSettings)`).

In the example below, we disabled line numbers in the editor and configured line wrapping to wrap automatically based on the size of the editor.



Changes to settings are reloaded by VS Code after the modified `settings.json` file is saved.

Note: Workspace settings are useful for sharing project specific settings across a team.

Settings File Locations

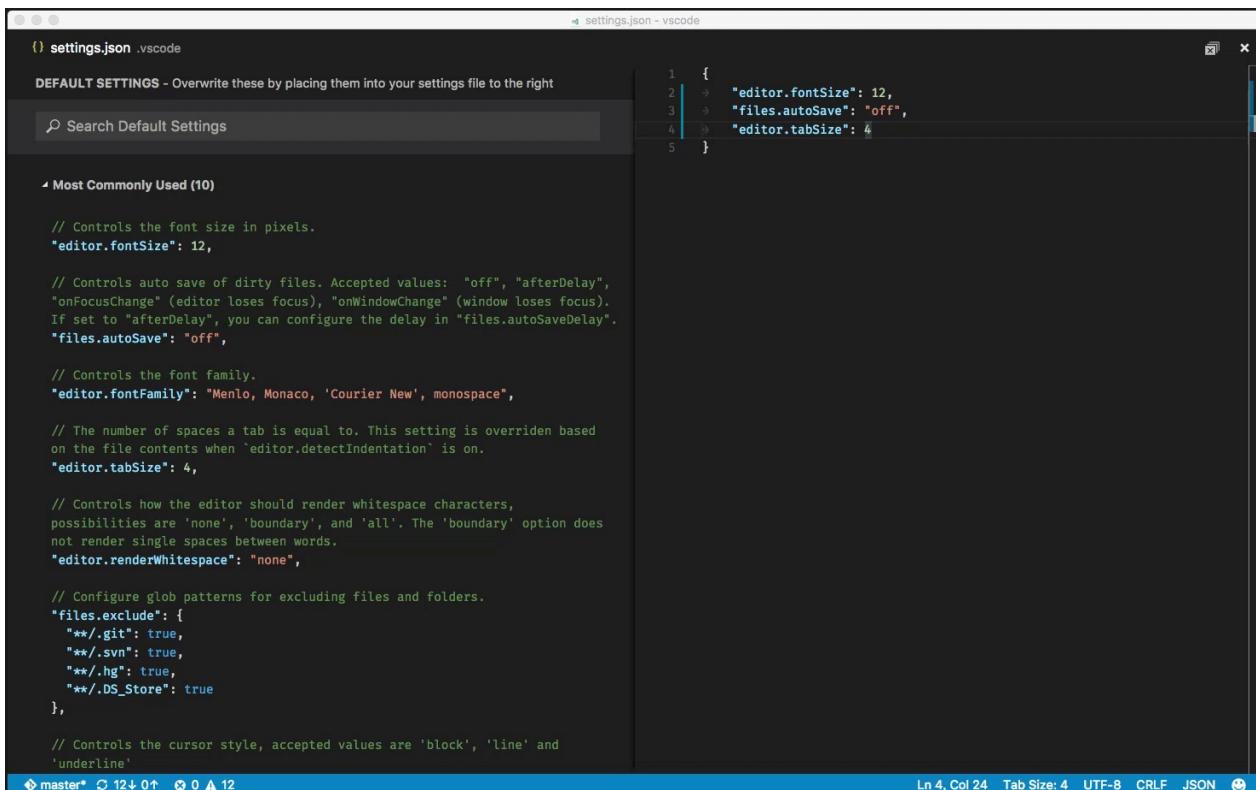
Depending on your platform, the user settings file is located here:

- **Windows** `%APPDATA%\Code\User\settings.json`
- **Mac** `$HOME/Library/Application Support/Code/User/settings.json`
- **Linux** `$HOME/.config/Code/User/settings.json`

The workspace setting file is located under the `.vscode` folder in your project.

Default Settings

When you open settings, we show **Default Settings** to search and discover settings you are looking for. When you search using the big Search bar, it will not only show and highlight the settings matching your criteria, but also filter out those which are not matching. This makes finding settings quick and easy. There are actions available inside **Default Settings** and `settings.json` editors which will help you quickly copy or update a setting.



The screenshot shows the VS Code interface with the title bar "settings.json - vscode". The main area displays the contents of the `settings.json` file. The code is as follows:

```

1  {
2   "editor.fontSize": 12,
3   "files.autoSave": "off",
4   "editor.tabSize": 4
5 }

```

Below the code editor, there is a sidebar with the following sections:

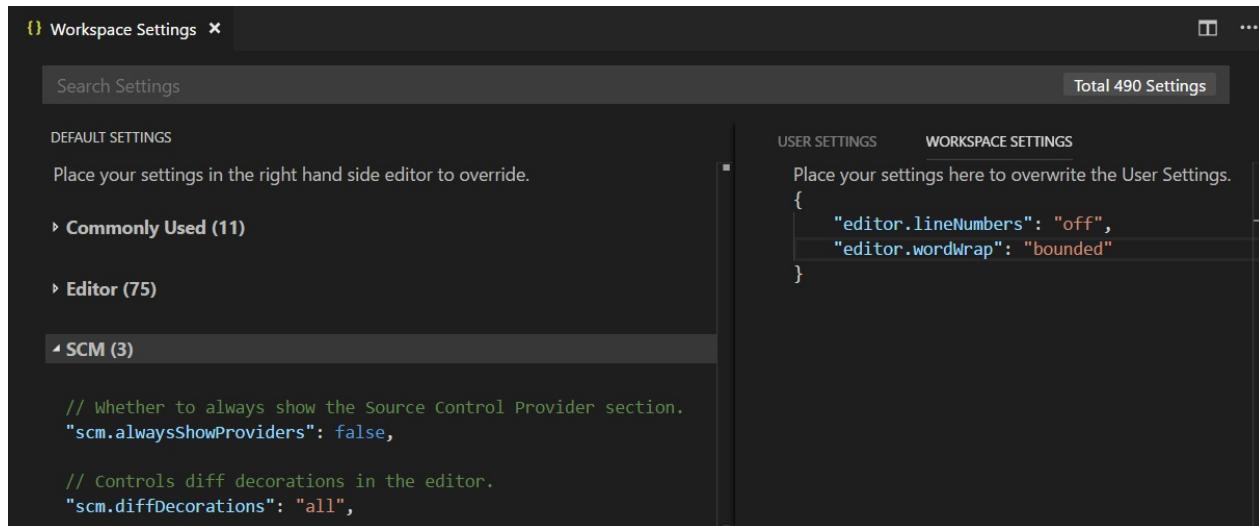
- DEFAULT SETTINGS - Overwrite these by placing them into your settings file to the right**: A note indicating that changes made here will be overwritten by the user's settings file.
- Search Default Settings**: A search input field.
- Most Commonly Used (10)**: A list of frequently used settings with their descriptions.

The bottom status bar shows the file path "master" and other details like "Ln 4, Col 24" and "Tab Size: 4".

Note: VS Code extensions can also add their own custom settings and they will be visible in the **Default Settings** list at runtime.

Settings groups

Default settings are represented in groups so that you can navigate them easily. It has **Most Commonly Used** group on the top to see the most common customizations done by VS Code users.

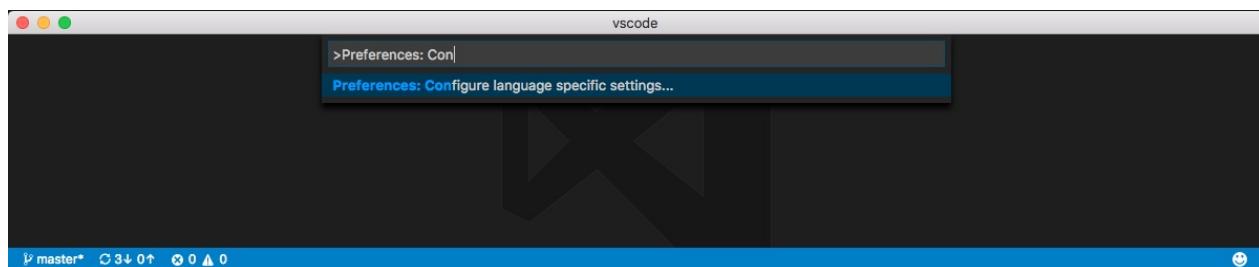


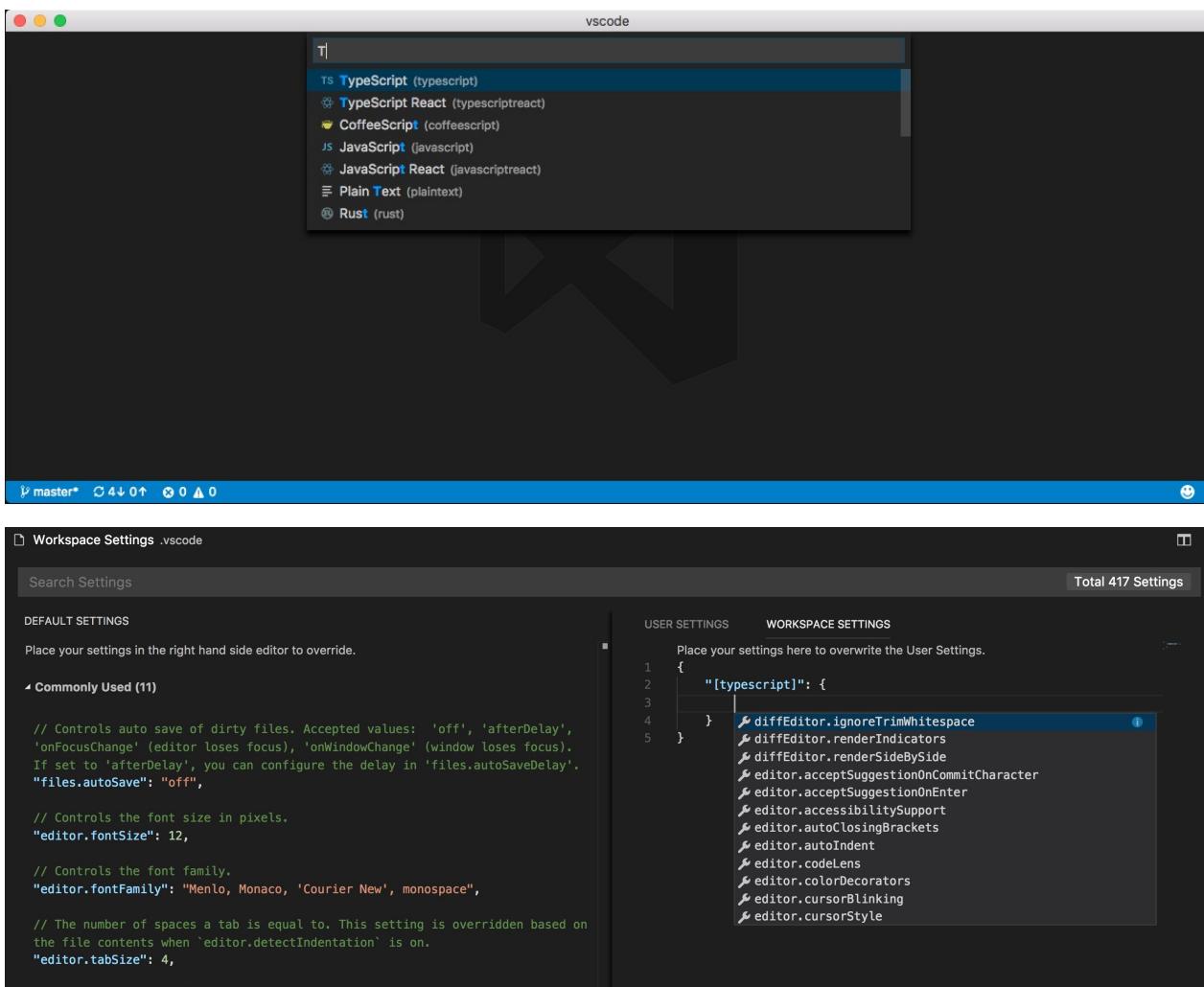
Here is the [copy of default settings](#) that comes with VS Code.

Language specific editor settings

To customize your editor by language, run the global command **Preferences: Configure language specific settings...** (command id:

```
workbench.action.configureLanguageBasedSettings ) from the Command Palette ( kb(workbench.action.showCommands) ) which opens the language picker. Selecting the language you want, opens the Settings editor with the language entry where you can add applicable settings.
```





If you have a file open and you want to customize the editor for this file type, click on the Language Mode in the Status Bar to the bottom-right of the VS Code window. This opens the Language Mode picker with an option **Configure 'language_name' language based settings....** Selecting this opens the Settings editor with the language entry where you can add applicable settings.

You can also configure language based settings by directly opening `settings.json`. You can scope them to the workspace by placing them in the Workspace settings just like other settings. If you have settings defined for a language in both user and workspace scopes, then they are merged by giving precedence to the ones defined in the workspace.

The following examples customize editor settings for language modes `typescript` and `markdown`.

```
{  
  "[typescript)": {  
    "editor.formatOnSave": true,  
    "editor.formatOnPaste": true  
  },  
  "[markdown)": {  
    "editor.formatOnSave": true,  
    "editor.wordWrap": "on",  
    "editor.renderWhitespace": "all",  
    "editor.acceptSuggestionOnEnter": "off"  
  }  
}
```

You can use IntelliSense in Settings editor to help you find allowed language based settings. All editor settings and some non-editor settings are supported.

Settings and security

In settings, we allow you to specify some of the executables that VS Code will run to do its work. For example, you can choose which shell the Integrated Terminal should use. For enhanced security, such settings can only be defined in user settings and not at workspace scope.

Here is the list of settings we don't support at the workspace scope:

- `git.path`
- `terminal.integrated.shell.linux`
- `terminal.integrated.shellArgs.linux`
- `terminal.integrated.shell.osx`
- `terminal.integrated.shellArgs.osx`
- `terminal.integrated.shell.windows`
- `terminal.integrated.shellArgs.windows`
- `terminal.external.windowsExec`
- `terminal.external.osxExec`
- `terminal.external.linuxExec`

The first time you open a workspace which defines any of these settings, VS Code will warn you and subsequently always ignore the values after that.

Copy of Default Settings

Below are the default settings and their values.

```
{
// Most Commonly Used

    // Controls auto save of dirty files. Accepted values: 'off', 'afterDelay', 'onFocusChange' (editor loses focus), 'onWindowChange' (window loses focus). If set to 'afterDelay', you can configure the delay in 'files.autoSaveDelay'.
    "files.autoSave": "off",

    // Controls the font size in pixels.
    "editor.fontSize": 14,

    // Controls the font family.
    "editor.fontFamily": "Consolas, 'Courier New', monospace",

    // The number of spaces a tab is equal to. This setting is overridden based on the file contents when `editor.detectIndentation` is on.
    "editor.tabSize": 4,

    // Controls how the editor should render whitespace characters, possibilities are 'none', 'boundary', and 'all'. The 'boundary' option does not render single spaces between words.
    "editor.renderWhitespace": "none",

    // Controls the cursor style, accepted values are 'block', 'block-outline', 'line', 'line-thin', 'underline' and 'underline-thin'
    "editor.cursorStyle": "line",

    // The modifier to be used to add multiple cursors with the mouse. `ctrlCmd` maps to `Control` on Windows and Linux and to `Command` on OSX. The Go To Definition and Open Link mouse gestures will adapt such that they do not conflict with the multicursor modifier.
    "editor.multiCursorModifier": "alt",

    // Insert spaces when pressing Tab. This setting is overridden based on the file contents when `editor.detectIndentation` is on.
    "editor.insertSpaces": true,

    // Controls how lines should wrap. Can be:
    // - 'off' (disable wrapping),
    // - 'on' (viewport wrapping),
    // - 'wordWrapColumn' (wrap at `editor.wordWrapColumn`) or
    // - 'bounded' (wrap at minimum of viewport and `editor.wordWrapColumn`).
    "editor.wordWrap": "off",

    // Configure glob patterns for excluding files and folders. For example, the files explorer decides which files and folders to show or hide based on this setting.
    "files.exclude": {
        "**/.git": true,
        "**/.svn": true,
        "**/.hg": true,
        "**/CVS": true,
        "**/.DS_Store": true
    }
}
```

```
},  
  
// Configure file associations to languages (e.g. "*.extension": "html"). These have  
precedence over the default associations of the languages installed.  
"files.associations": {},  
  
// Editor  
  
// Controls if the diff editor shows changes in leading or trailing whitespace as di-  
ffs  
"diffEditor.ignoreTrimWhitespace": true,  
  
// Controls if the diff editor shows +/- indicators for added/removed changes  
"diffEditor.renderIndicators": true,  
  
// Controls if the diff editor shows the diff side by side or inline  
"diffEditor.renderSideBySide": true,  
  
// Controls if suggestions should be accepted on commit characters. For instance in  
JavaScript the semi-colon (';') can be a commit character that accepts a suggestion an-  
d types that character.  
"editor.acceptSuggestionOnCommitCharacter": true,  
  
// Controls if suggestions should be accepted on 'Enter' - in addition to 'Tab'. Hel-  
ps 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 whether the editor should run in a mode where it is optimized for screen  
readers.  
"editor.accessibilitySupport": "auto",  
  
// Controls if the editor should automatically close brackets after opening them  
"editor.autoClosingBrackets": true,  
  
// Controls if the editor should automatically adjust the indentation when users typ-  
e, paste or move lines. Indentation rules of the language must be available.  
"editor.autoIndent": true,  
  
// Controls if the editor shows code lenses  
"editor.codeLens": true,  
  
// Controls whether the editor should render the inline color decorators and color p-  
icker.  
"editor.colorDecorators": true,  
  
// Control the cursor animation style, possible values are 'blink', 'smooth', 'phase'  
'', 'expand' and 'solid'  
"editor.cursorBlinking": "blink",  
  
// Controls the cursor style, accepted values are 'block', 'block-outline', 'line',  
'line-thin', 'underline' and 'underline-thin'  
"editor.cursorStyle": "line",
```

```
// When opening a file, `editor.tabSize` and `editor.insertSpaces` will be detected  
// based on the file contents.  
"editor.detectIndentation": true,  
  
// Controls if the editor should allow to move selections via drag and drop.  
"editor.dragAndDrop": true,  
  
// Controls whether copying without a selection copies the current line.  
"editor.emptySelectionClipboard": true,  
  
// Controls if Find in Selection flag is turned on when multiple characters or lines  
// of text are selected in the editor  
"editor.find.autoFindInSelection": false,  
  
// Controls if the Find Widget should read or modify the shared find clipboard on  
// macOS  
"editor.find.globalFindClipboard": true,  
  
// Controls if we seed the search string in Find Widget from editor selection  
"editor.find.seedSearchStringFromSelection": true,  
  
// Controls whether the editor has code folding enabled  
"editor.folding": true,  
  
// Controls the font family.  
"editor.fontFamily": "Consolas, 'Courier New', monospace",  
  
// Enables font ligatures  
"editor.fontLigatures": false,  
  
// Controls the font size in pixels.  
"editor.fontSize": 14,  
  
// Controls the font weight.  
"editor.fontWeight": "normal",  
  
// Controls if the editor should automatically format the pasted content. A formatter  
// must be available and the formatter should be able to format a range in a document.  
"editor.formatOnPaste": false,  
  
// Format a file on save. A formatter must be available, the file must not be auto-saved,  
// and editor must not be shutting down.  
"editor.formatOnSave": false,  
  
// Controls if the editor should automatically format the line after typing  
"editor.formatOnType": false,  
  
// Controls whether the editor should render the vertical glyph margin. Glyph margin  
// is mostly used for debugging.  
"editor.glyphMargin": true,  
  
// Controls if the cursor should be hidden in the overview ruler.
```

```
"editor.hideCursorInOverviewRuler": false,  
  
    // Insert spaces when pressing Tab. This setting is overridden based on the file contents when `editor.detectIndentation` is on.  
    "editor.insertSpaces": true,  
  
    // Controls the letter spacing in pixels.  
    "editor.letterSpacing": 0,  
  
    // Enables the code action lightbulb  
    "editor.lightbulb.enabled": true,  
  
    // Controls the line height. Use 0 to compute the lineHeight from the fontSize.  
    "editor.lineHeight": 0,  
  
    // Controls the display of line numbers. Possible values are 'on', 'off', and 'relative'.  
    "editor.lineNumbers": "on",  
  
    // Controls whether the editor should detect links and make them clickable  
    "editor.links": true,  
  
    // Highlight matching brackets when one of them is selected.  
    "editor.matchBrackets": true,  
  
    // Controls if the minimap is shown  
    "editor.minimap.enabled": true,  
  
    // Limit the width of the minimap to render at most a certain number of columns  
    "editor.minimap.maxColumn": 120,  
  
    // Render the actual characters on a line (as opposed to color blocks)  
    "editor.minimap.renderCharacters": true,  
  
    // Controls whether the minimap slider is automatically hidden. Possible values are 'always' and 'mouseover'  
    "editor.minimap.showSlider": "mouseover",  
  
    // A multiplier to be used on the `deltaX` and `deltaY` of mouse wheel scroll events  
    "editor.mousewheelScrollSensitivity": 1,  
  
    // Zoom the font of the editor when using mouse wheel and holding Ctrl  
    "editor.mouseWheelZoom": false,  
  
    // The modifier to be used to add multiple cursors with the mouse. `ctrlCmd` maps to `Control` on Windows and Linux and to `Command` on OSX. The Go To Definition and Open Link mouse gestures will adapt such that they do not conflict with the multicursor modifier.  
    "editor.multiCursorModifier": "alt",  
  
    // Controls whether the editor should highlight semantic symbol occurrences  
    "editor.occurrencesHighlight": true,
```

```
// Controls if a border should be drawn around the overview ruler.  
"editor.overviewRulerBorder": true,  
  
// Controls the number of decorations that can show up at the same position in the o  
verview ruler  
"editor.overviewRulerLanes": 3,  
  
// Enables pop-up that shows parameter documentation and type information as you type  
"editor.parameterHints": true,  
  
// Controls if suggestions should automatically show up while typing  
"editor.quickSuggestions": {  
    "other": true,  
    "comments": false,  
    "strings": false  
},  
  
// Controls the delay in ms after which quick suggestions will show up  
"editor.quickSuggestionsDelay": 10,  
  
// Controls whether the editor should render control characters  
"editor.renderControlCharacters": false,  
  
// Controls whether the editor should render indent guides  
"editor.renderIndentGuides": true,  
  
// Controls how the editor should render the current line highlight, possibilities a  
re 'none', 'gutter', 'line', and 'all'.  
"editor.renderLineHighlight": "line",  
  
// Controls how the editor should render whitespace characters, possibilities are 'n  
one', 'boundary', and 'all'. The 'boundary' option does not render single spaces betwe  
en words.  
"editor.renderWhitespace": "none",  
  
// Controls if selections have rounded corners  
"editor.roundedSelection": true,  
  
// Render vertical rulers after a certain number of monospace characters. Use multip  
le values for multiple rulers. No rulers are drawn if array is empty  
"editor.rulers": [],  
  
// Controls if the editor will scroll beyond the last line  
"editor.scrollBeyondLastLine": true,  
  
// Controls whether the editor should highlight similar matches to the selection  
"editor.selectionHighlight": true,  
  
// Controls whether the fold controls on the gutter are automatically hidden.  
"editor.showFoldingControls": "mouseover",  
  
// Controls if the editor will scroll using an animation
```

```
"editor.smoothScrolling": false,  
  
    // Controls whether snippets are shown with other suggestions and how they are sorted.  
    "editor.snippetSuggestions": "inline",  
  
    // Keep peek editors open even when double clicking their content or when hitting Escape.  
    "editor.stablePeek": false,  
  
    // Font size for the suggest widget  
    "editor.suggestFontSize": 0,  
  
    // Line height for the suggest widget  
    "editor.suggestLineHeight": 0,  
  
    // Controls if suggestions should automatically show up when typing trigger characters  
    "editor.suggestOnTriggerCharacters": true,  
  
    // Insert snippets when their prefix matches. Works best when 'quickSuggestions' are not enabled.  
    "editor.tabCompletion": false,  
  
    // The number of spaces a tab is equal to. This setting is overridden based on the file contents when `editor.detectIndentation` is on.  
    "editor.tabSize": 4,  
  
    // Overrides editor colors and font style from the currently selected color theme.  
    "editor.tokenColorCustomizations": {},  
  
    // Remove trailing auto inserted whitespace  
    "editor.trimAutoWhitespace": true,  
  
    // Inserting and deleting whitespace follows tab stops  
    "editor.useTabStops": true,  
  
    // Controls whether completions should be computed based on words in the document.  
    "editor.wordBasedSuggestions": true,  
  
    // Characters that will be used as word separators when doing word related navigations or operations  
    "editor.wordSeparators": "`~!@#$%^&*()=-+[{}]\\";:'\\",.;<>/?",  
  
    // Controls how lines should wrap. Can be:  
    // - 'off' (disable wrapping),  
    // - 'on' (viewport wrapping),  
    // - 'wordWrapColumn' (wrap at `editor.wordWrapColumn`) or  
    // - 'bounded' (wrap at minimum of viewport and `editor.wordWrapColumn`).  
    "editor.wordWrap": "off",  
  
    // Controls the wrapping column of the editor when `editor.wordWrap` is 'wordWrapColumn' or 'bounded'.  

```

```
"editor.wordWrapColumn": 80,  
  
    // Controls the indentation of wrapped lines. Can be one of 'none', 'same' or 'indent'.  
    "editor.wrappingIndent": "same",  
  
    // Workbench  
  
    // Controls the visibility of the activity bar in the workbench.  
    "workbench.activityBar.visible": true,  
  
    // Overrides colors from the currently selected color theme.  
    "workbench.colorCustomizations": {},  
  
    // Specifies the color theme used in the workbench.  
    "workbench.colorTheme": "Default Dark+",  
  
    // Controls the number of recently used commands to keep in history for the command palette. Set to 0 to disable command history.  
    "workbench.commandPalette.history": 50,  
  
    // Controls if the last typed input to the command palette should be restored when opening it the next time.  
    "workbench.commandPalette.preserveInput": false,  
  
    // Controls if editors showing a file should close automatically when the file is deleted or renamed by some other process. Disabling this will keep the editor open as dirty on such an event. Note that deleting from within the application will always close the editor and that dirty files will never close to preserve your data.  
    "workbench.editor.closeOnFileDelete": true,  
  
    // Controls if opened editors show as preview. Preview editors are reused until they are kept (e.g. via double click or editing) and show up with an italic font style.  
    "workbench.editor.enablePreview": true,  
  
    // Controls if opened editors from Quick Open show as preview. Preview editors are reused until they are kept (e.g. via double click or editing).  
    "workbench.editor.enablePreviewFromQuickOpen": true,  
  
    // Controls the format of the label for an editor. Changing this setting can for example make it easier to understand the location of a file:  
    // - short: 'parent'  
    // - medium: 'workspace/src/parent'  
    // - long: '/home/user/workspace/src/parent'  
    // - default: '.../parent', when another tab shares the same title, or the relative workspace path if tabs are disabled  
    "workbench.editor.labelFormat": "default",  
  
    // Controls where editors open. Select 'left' or 'right' to open editors to the left or right of the currently active one. Select 'first' or 'last' to open editors independently from the currently active one.  
    "workbench.editor.openPositioning": "right",
```

```
// Controls if an editor is revealed in any of the visible groups if opened. If disabled, an editor will prefer to open in the currently active editor group. If enabled, an already opened editor will be revealed instead of opened again in the currently active editor group. Note that there are some cases where this setting is ignored, e.g. when forcing an editor to open in a specific group or to the side of the currently active group.  
"workbench.editor.revealIfOpen": false,  
  
// Controls if opened editors should show with an icon or not. This requires an icon theme to be enabled as well.  
"workbench.editor.showIcons": true,  
  
// Controls if opened editors should show in tabs or not.  
"workbench.editor.showTabs": true,  
  
// Controls the position of the editor's tabs close buttons or disables them when set to 'off'.  
"workbench.editor.tabCloseButton": "right",  
  
// Controls the sizing of editor tabs. Set to 'fit' to keep tabs always large enough to show the full editor label. Set to 'shrink' to allow tabs to get smaller when the available space is not enough to show all tabs at once.  
"workbench.editor.tabSizing": "fit",  
  
// Specifies the icon theme used in the workbench or 'null' to not show any file icons.  
"workbench.iconTheme": "vs-seti",  
  
// Controls if Quick Open should close automatically once it loses focus.  
"workbench.quickOpen.closeOnFocusLost": true,  
  
// Controls whether to enable the natural language search mode for settings.  
"workbench.settings.enableNaturalLanguageSearch": true,  
  
// Controls if opening settings also opens an editor showing all default settings.  
"workbench.settings.openDefaultSettings": true,  
  
// Controls the location of the sidebar. It can either show on the left or right of the workbench.  
"workbench.sidebar.location": "left",  
  
// Controls which editor is shown at startup, if none is restored from the previous session. Select 'none' to start without an editor, 'welcomePage' to open the Welcome page (default), 'newUntitledFile' to open a new untitled file (only opening an empty workspace).  
"workbench.startupEditor": "welcomePage",  
  
// Controls the visibility of the status bar at the bottom of the workbench.  
"workbench.statusBar.visible": true,  
  
// When enabled, will show the watermark tips when no editor is open.  
"workbench.tips.enabled": true,
```

```
// Window

// Controls if closing the last editor should also close the window. This setting only applies for windows that do not show folders.
"window.closeWhenEmpty": false,

// If enabled, the main menus can be opened via Alt-key shortcuts. Disabling mnemonics allows to bind these Alt-key shortcuts to editor commands instead.
"window.enableMenuBarMnemonics": true,

// Control the visibility of the menu bar. A setting of 'toggle' means that the menu bar is hidden and a single press of the Alt key will show it. By default, the menu bar will be visible, unless the window is full screen.
"window.menuBarVisibility": "default",

// Controls the dimensions of opening a new window when at least one window is already opened. By default, a new window will open in the center of the screen with small dimensions. When set to 'inherit', the window will get the same dimensions as the last window that was active. When set to 'maximized', the window will open maximized and fullscreen if configured to 'fullscreen'. Note that this setting does not have an impact on the first window that is opened. The first window will always restore the size and location as you left it before closing.
"window.newWindowDimensions": "default",

// Controls if files should open in a new window.
// - default: files will open in the window with the files' folder open or the last active window unless opened via the dock or from finder (macOS only)
// - on: files will open in a new window
// - off: files will open in the window with the files' folder open or the last active window
// Note that there can still be cases where this setting is ignored (e.g. when using the -new-window or -reuse-window command line option).
"window.openFilesInNewWindow": "off",

// Controls if folders should open in a new window or replace the last active window.
// - default: folders will open in a new window unless a folder is picked from within the application (e.g. via the File menu)
// - on: folders will open in a new window
// - off: folders will replace the last active window
// Note that there can still be cases where this setting is ignored (e.g. when using the -new-window or -reuse-window command line option).
"window.openFoldersInNewWindow": "default",

// Controls if a window should restore to full screen mode if it was exited in full screen mode.
"window.restoreFullscreen": false,

// Controls how windows are being reopened after a restart. Select 'none' to always start with an empty workspace, 'one' to reopen the last window you worked on, 'folders' to reopen all windows that had folders opened or 'all' to reopen all windows of your last session.
"window.restoreWindows": "one",
```

```
// Controls the window title based on the active editor. Variables are substituted based on the context:  
// ${activeEditorShort}: the file name (e.g. myFile.txt)  
// ${activeEditorMedium}: the path of the file relative to the workspace folder (e.g. . myFolder/myFile.txt)  
// ${activeEditorLong}: the full path of the file (e.g. /Users/Development/myProject /myFolder/myFile.txt)  
// ${folderName}: name of the workspace folder the file is contained in (e.g. myFolder)  
// ${FolderPath}: file path of the workspace folder the file is contained in (e.g. / Users/Development/myFolder)  
// ${rootName}: name of the workspace (e.g. myFolder or myWorkspace)  
// ${rootPath}: file path of the workspace (e.g. /Users/Development/myWorkspace)  
// ${appName}: e.g. VS Code  
// ${dirty}: a dirty indicator if the active editor is dirty  
// ${separator}: a conditional separator (" - ") that only shows when surrounded by variables with values  
"window.title": "${dirty}${activeEditorShort}${separator}${rootName}${separator}${appName}",  
  
// Adjust the zoom level of the window. The original size is 0 and each increment above (e.g. 1) or below (e.g. -1) represents zooming 20% larger or smaller. You can also enter decimals to adjust the zoom level with a finer granularity.  
"window.zoomLevel": 0,  
  
// Files  
  
// Configure file associations to languages (e.g. "*.extension": "html"). These have precedence over the default associations of the languages installed.  
"files.associations": {},  
  
// When enabled, will attempt to guess the character set encoding when opening files. This setting can be configured per language too.  
"files.autoGuessEncoding": false,  
  
// Controls auto save of dirty files. Accepted values: 'off', 'afterDelay', 'onFocusChange' (editor loses focus), 'onWindowChange' (window loses focus). If set to 'afterDelay', you can configure the delay in 'files.autoSaveDelay'.  
"files.autoSave": "off",  
  
// Controls the delay in ms after which a dirty file is saved automatically. Only applies when 'files.autoSave' is set to 'afterDelay'  
"files.autoSaveDelay": 1000,  
  
// The default language mode that is assigned to new files.  
"files.defaultLanguage": "",  
  
// The default character set encoding to use when reading and writing files. This setting can be configured per language too.  
"files.encoding": "utf8",  
  
// The default end of line character. Use \n for LF and \r\n for CRLF.
```

```
"files.eol": "\r\n",  
  
    // Configure glob patterns for excluding files and folders. For example, the files e  
xplorer decides which files and folders to show or hide based on this setting.  
    "files.exclude": {  
        "**/.git": true,  
        "**/.svn": true,  
        "**/.hg": true,  
        "**/CVS": true,  
        "**/.DS_Store": true  
    },  
  
    // Controls whether unsaved files are remembered between sessions, allowing the save  
prompt when exiting the editor to be skipped.  
    "files.hotExit": "onExit",  
  
    // When enabled, insert a final new line at the end of the file when saving it.  
    "files.insertFinalNewline": false,  
  
    // When enabled, will trim all new lines after the final new line at the end of the  
file when saving it.  
    "files.trimFinalNewlines": false,  
  
    // When enabled, will trim trailing whitespace when saving a file.  
    "files.trimTrailingWhitespace": false,  
  
    // Use the new experimental file watcher.  
    "files.useExperimentalFilewatcher": false,  
  
    // Configure glob patterns of file paths to exclude from file watching. Patterns must  
match on absolute paths (i.e. prefix with ** or the full path to match properly). Ch  
anging this setting requires a restart. When you experience Code consuming lots of cpu  
time on startup, you can exclude large folders to reduce the initial load.  
    "files.watcherExclude": {  
        "**/.git/objects/**": true,  
        "**/.git/subtree-cache/**": true,  
        "**/node_modules/**": true  
    },  
  
// Zen Mode  
  
    // Controls if turning on Zen Mode also puts the workbench into full screen mode.  
    "zenMode.fullScreen": true,  
  
    // Controls if turning on Zen Mode also hides the activity bar at the left of the wo  
rkbench.  
    "zenMode.hideActivityBar": true,  
  
    // Controls if turning on Zen Mode also hides the status bar at the bottom of the wo  
rkbench.  
    "zenMode.hideStatusBar": true,  
  
    // Controls if turning on Zen Mode also hides workbench tabs.
```

```
"zenMode.hideTabs": true,  
  
    // Controls if a window should restore to zen mode if it was exited in zen mode.  
    "zenMode.restore": false,  
  
    // File Explorer  
  
        // Controls if the explorer should automatically reveal and select files when opening them.  
        "explorer.autoReveal": true,  
  
        // Controls if the explorer should ask for confirmation when deleting a file via the trash.  
        "explorer.confirmDelete": true,  
  
        // Controls if the explorer should ask for confirmation to move files and folders via drag and drop.  
        "explorer.confirmDragAndDrop": true,  
  
        // Controls if file decorations should use badges.  
        "explorer.decorations.badges": true,  
  
        // Controls if file decorations should use colors.  
        "explorer.decorations.colors": true,  
  
        // Controls if the explorer should allow to move files and folders via drag and drop.  
        "explorer.enableDragAndDrop": true,  
  
        // Controls if the height of the open editors section should adapt dynamically to the number of elements or not.  
        "explorer.openEditors.dynamicHeight": true,  
  
        // Number of editors shown in the Open Editors pane. Set it to 0 to hide the pane.  
        "explorer.openEditors.visible": 9,  
  
        // Controls sorting order of files and folders in the explorer. In addition to the default sorting, you can set the order to 'mixed' (files and folders sorted combined), 'type' (by file type), 'modified' (by last modified date) or 'filesFirst' (sort files before folders).  
        "explorer.sortOrder": "default",  
  
    // Search  
  
        // Configure glob patterns for excluding files and folders in searches. Inherits all glob patterns from the files.exclude setting.  
        "search.exclude": {  
            "**/node_modules": true,  
            "**/bower_components": true  
        },  
  
        // Controls whether to follow symlinks while searching.  
        "search.followSymlinks": true,
```

```
// Configure to include results from a global symbol search in the file results for
// Quick Open.
"search.quickOpen.includeSymbols": false,

// Controls whether to use .gitignore and .ignore files when searching for files.
"search.useIgnoreFiles": true,

// Controls whether to use ripgrep in text and file search
"search.useRipgrep": true,

// HTTP

// The proxy setting to use. If not set will be taken from the http_proxy and https_
proxy environment variables
"http.proxy": "",

// The value to send as the 'Proxy-Authorization' header for every network request.
"http.proxyAuthorization": null,

// Whether the proxy server certificate should be verified against the list of suppl
ied CAs.
"http.proxyStrictSSL": true,

// Update

// Configure whether you receive automatic updates from an update channel. Requires
a restart after change.
"update.channel": "default",

// Debug

// Allows setting breakpoint in any file
"debug.allowBreakpointsEverywhere": false,

// Controls if the floating debug action bar should be hidden
"debug.hideActionBar": false,

// Show variable values inline in editor while debugging
"debug.inlineValues": false,

// Controls behavior of the internal debug console.
"debug.internalConsoleOptions": "openOnFirstSessionStart",

// Controls whether debug viewlet should be open on debugging session start.
"debug.openDebug": "openOnFirstSessionStart",

// Automatically open explorer view on the end of a debug session
"debug.openExplorerOnEnd": false,

// Controls when the debug status bar should be visible
"debug.showInStatusBar": "onFirstSessionStart",
```

```
// Global debug launch configuration. Should be used as an alternative to 'launch.json'
on' that is shared across workspaces
"launch": {},


// HTML


// Enable/disable autoclosing of HTML tags.
"html.autoClosingTags": true,


// List of tags, comma separated, where the content shouldn't be reformatted. 'null'
defaults to the 'pre' tag.
"html.format.contentUnformatted": "pre,code,textarea",


// Enable/disable default HTML formatter
"html.format.enable": true,


// End with a newline.
"html.format.endWithNewline": false,


// List of tags, comma separated, that should have an extra newline before them. 'nu
ll' defaults to "head, body, /html".
"html.format.extraLiners": "head, body, /html",


// Format and indent {{#foo}} and {{/foo}}.
"html.format.indentHandlebars": false,


// Indent <head> and <body> sections.
"html.format.indentInnerHTML": false,


// Maximum number of line breaks to be preserved in one chunk. Use 'null' for unlimi
ted.
"html.format.maxPreserveNewLines": null,


// Whether existing line breaks before elements should be preserved. Only works befo
re elements, not inside tags or for text.
"html.format.preserveNewLines": true,


// List of tags, comma separated, that shouldn't be reformatted. 'null' defaults to
all tags listed at https://www.w3.org/TR/html5/dom.html#phrasing-content.
"html.format.unformatted": "wbr",


// Wrap attributes.
"html.format.wrapAttributes": "auto",


// Maximum amount of characters per line (0 = disable).
"html.format.wrapLineLength": 120,


// Configures if the built-in HTML language support suggests Angular V1 tags and pro
perties.
"html.suggest.angular1": true,


// Configures if the built-in HTML language support suggests HTML5 tags, properties
and values.
```

```
"html.suggest.html5": true,  
  
    // Configures if the built-in HTML language support suggests Ionic tags, properties  
    and values.  
    "html.suggest.ionic": true,  
  
    // Traces the communication between VS Code and the HTML language server.  
    "html.trace.server": "off",  
  
    // Configures if the built-in HTML language support validates embedded scripts.  
    "html.validate.scripts": true,  
  
    // Configures if the built-in HTML language support validates embedded styles.  
    "html.validate.styles": true,  
  
// JSON  
  
    // Enable/disable default JSON formatter (requires restart)  
    "json.format.enable": true,  
  
    // Associate schemas to JSON files in the current project  
    "json.schemas": [],  
  
    // Traces the communication between VS Code and the JSON language server.  
    "json.trace.server": "off",  
  
// Markdown  
  
    // Sets how line-breaks are rendered in the markdown preview. Setting it to 'true' c  
    reates a <br> for every newline.  
    "markdown.preview.breaks": false,  
  
    // Double click in the markdown preview to switch to the editor.  
    "markdown.preview.doubleClickToSwitchToEditor": true,  
  
    // Controls the font family used in the markdown preview.  
    "markdown.preview.fontFamily": "-apple-system, BlinkMacSystemFont, 'Segoe WPC', 'Seg  
    oe UI', 'HelveticaNeue-Light', 'Ubuntu', 'Droid Sans', sans-serif",  
  
    // Controls the font size in pixels used in the markdown preview.  
    "markdown.preview.fontSize": 14,  
  
    // Controls the line height used in the markdown preview. This number is relative to  
    the font size.  
    "markdown.preview.lineHeight": 1.6,  
  
    // Enable or disable conversion of URL-like text to links in the markdown preview.  
    "markdown.preview.linkify": true,  
  
    // Mark the current editor selection in the markdown preview.  
    "markdown.preview.markEditorSelection": true,  
  
    // When the markdown preview is scrolled, update the view of the editor.
```

```
"markdown.preview.scrollEditorWithPreview": true,  
  
    // Scrolls the markdown preview to reveal the currently selected line from the editor.  
    "markdown.preview.scrollPreviewWithEditorSelection": true,  
  
    // Sets how YAML front matter should be rendered in the markdown preview. 'hide' removes the front matter. Otherwise, the front matter is treated as markdown content.  
    "markdown.previewFrontMatter": "hide",  
  
    // A list of URLs or local paths to CSS style sheets to use from the markdown preview. Relative paths are interpreted relative to the folder open in the explorer. If there is no open folder, they are interpreted relative to the location of the markdown file. All '\' need to be written as '\\'.  
    "markdown.styles": [],  
  
    // Enable debug logging for the markdown extension.  
    "markdown.trace": "off",  
  
// PHP  
  
    // Configures if the built-in PHP language suggestions are enabled. The support suggests PHP globals and variables.  
    "php.suggest.basic": true,  
  
    // Enable/disable built-in PHP validation.  
    "php.validate.enable": true,  
  
    // Points to the PHP executable.  
    "php.validate.executablePath": null,  
  
    // Whether the linter is run on save or on type.  
    "php.validate.run": "onSave",  
  
// TypeScript  
  
    // Enable/disable default JavaScript formatter.  
    "javascript.format.enable": true,  
  
    // Defines space handling after a comma delimiter.  
    "javascript.format.insertSpaceAfterCommaDelimiter": true,  
  
    // Defines space handling after the constructor keyword. Requires TypeScript >= 2.3.0.  
    "javascript.format.insertSpaceAfterConstructor": false,  
  
    // Defines space handling after function keyword for anonymous functions.  
    "javascript.format.insertSpaceAfterFunctionKeywordForAnonymousFunctions": true,  
  
    // Defines space handling after keywords in a control flow statement.  
    "javascript.format.insertSpaceAfterKeywordsInControlFlowStatements": true,  
  
    // Defines space handling after opening and before closing JSX expression braces. Re
```

```
quires TypeScript >= 2.0.6.
  "javascript.format.insertSpaceAfterOpeningAndBeforeClosingJsxExpressionBraces": false
  ,

  // Defines space handling after opening and before closing non empty braces. Requires
  // TypeScript >= 2.3.0.
  "javascript.format.insertSpaceAfterOpeningAndBeforeClosingNonemptyBraces": true,

  // Defines space handling after opening and before closing non empty brackets.
  "javascript.format.insertSpaceAfterOpeningAndBeforeClosingNonemptyBrackets": false,

  // Defines space handling after opening and before closing non empty parenthesis.
  "javascript.format.insertSpaceAfterOpeningAndBeforeClosingNonemptyParenthesis": false
  ,

  // Defines space handling after opening and before closing template string braces. Requires
  // TypeScript >= 2.0.6.
  "javascript.format.insertSpaceAfterOpeningAndBeforeClosingTemplateStringBraces": false,
  ,

  // Defines space handling after a semicolon in a for statement.
  "javascript.format.insertSpaceAfterSemicolonInForStatements": true,

  // Defines space handling after a binary operator.
  "javascript.format.insertSpaceBeforeAndAfterBinaryOperators": true,
  ,

  // Defines space handling before function argument parentheses. Requires TypeScript
  // >= 2.1.5.
  "javascript.format.insertSpaceBeforeFunctionParenthesis": false,
  ,

  // Defines whether an open brace is put onto a new line for control blocks or not.
  "javascript.format.placeOpenBraceOnNewLineForControlBlocks": false,
  ,

  // Defines whether an open brace is put onto a new line for functions or not.
  "javascript.format.placeOpenBraceOnNewLineForFunctions": false,
  ,

  // Enable/disable semantic checking of JavaScript files. Existing jsconfig.json or
  // tsconfig.json files override this setting. Requires TypeScript >=2.3.1.
  "javascript.implicitProjectConfig.checkJs": false,
  ,

  // Enable/disable 'experimentalDecorators' for JavaScript files that are not part of
  // a project. Existing jsconfig.json or tsconfig.json files override this setting. Requires
  // TypeScript >=2.3.1.
  "javascript.implicitProjectConfig.experimentalDecorators": false,
  ,

  // Enable/disable including unique names from the file in JavaScript suggestion lists.
  "javascript.nameSuggestions": true,
  ,

  // Enable/disable references CodeLens in JavaScript files.
  "javascript.referencesCodeLens.enabled": false,
  ,

  // Enable/disable JavaScript validation.
```

```
"javascript.validate.enable": true,  
  
// Enable/disable auto JSDoc comments  
"jsDocCompletion.enabled": true,  
  
// Enable/disable auto import suggestions. Requires TypeScript >=2.6.1  
"typescript.autoImportSuggestions.enabled": true,  
  
// Check if NPM is installed for Automatic Type Acquisition.  
"typescript.check.npmIsInstalled": true,  
  
// Disables automatic type acquisition. Requires TypeScript >= 2.0.6.  
"typescript.disableAutomaticTypeAcquisition": false,  
  
// Enable/disable default TypeScript formatter.  
"typescript.format.enable": true,  
  
// Defines space handling after a comma delimiter.  
"typescript.format.insertSpaceAfterCommaDelimiter": true,  
  
// Defines space handling after the constructor keyword. Requires TypeScript >= 2.3.  
0.  
"typescript.format.insertSpaceAfterConstructor": false,  
  
// Defines space handling after function keyword for anonymous functions.  
"typescript.format.insertSpaceAfterFunctionKeywordForAnonymousFunctions": true,  
  
// Defines space handling after keywords in a control flow statement.  
"typescript.format.insertSpaceAfterKeywordsInControlFlowStatements": true,  
  
// Defines space handling after opening and before closing JSX expression braces. Re  
quires TypeScript >= 2.0.6.  
"typescript.format.insertSpaceAfterOpeningAndBeforeClosingJsxExpressionBraces": false  
,  
  
// Defines space handling after opening and before closing non empty braces. Require  
s TypeScript >= 2.3.0.  
"typescript.format.insertSpaceAfterOpeningAndBeforeClosingNonemptyBraces": true,  
  
// Defines space handling after opening and before closing non empty brackets.  
"typescript.format.insertSpaceAfterOpeningAndBeforeClosingNonemptyBrackets": false,  
  
// Defines space handling after opening and before closing non empty parenthesis.  
"typescript.format.insertSpaceAfterOpeningAndBeforeClosingNonemptyParenthesis": false  
,  
  
// Defines space handling after opening and before closing template string braces. R  
quires TypeScript >= 2.0.6.  
"typescript.format.insertSpaceAfterOpeningAndBeforeClosingTemplateStringBraces": fal  
se,  
  
// Defines space handling after a semicolon in a for statement.  
"typescript.format.insertSpaceAfterSemicolonInForStatements": true,
```

```
// Defines space handling after type assertions in TypeScript. Requires TypeScript >
= 2.4.
"typescript.format.insertSpaceAfterTypeAssertion": false,

// Defines space handling after a binary operator.
"typescript.format.insertSpaceBeforeAndAfterBinaryOperators": true,

// Defines space handling before function argument parentheses. Requires TypeScript
>= 2.1.5.
"typescript.format.insertSpaceBeforeFunctionParenthesis": false,

// Defines whether an open brace is put onto a new line for control blocks or not.
"typescript.format.placeOpenBraceOnNewLineForControlBlocks": false,

// Defines whether an open brace is put onto a new line for functions or not.
"typescript.format.placeOpenBraceOnNewLineForFunctions": false,

// Enable/disable implementations CodeLens. Requires TypeScript >= 2.2.0.
"typescript.implementationsCodeLens.enabled": false,

// Sets the locale used to report TypeScript errors. Requires TypeScript >= 2.6.0. Default of 'null' uses VS Code's locale for TypeScript errors.
"typescript.locale": null,

// Specifies the path to the NPM executable used for Automatic Type Acquisition. Requires TypeScript >= 2.3.4.
"typescript.npm": null,

// Enable/disable quick suggestions when typing out an import path.
"typescript.quickSuggestionsForPaths": true,

// Enable/disable references CodeLens in TypeScript files. Requires TypeScript >= 2.0.6.
"typescript.referencesCodeLens.enabled": false,

// Report style checks as warnings
"typescript.reportStyleChecksAsWarnings": true,

// Controls auto detection of tsc tasks. 'off' disables this feature. 'build' only creates single run compile tasks. 'watch' only creates compile and watch tasks. 'on' creates both build and watch tasks. Default is 'on'.
"typescript.tsc.autoDetect": "on",

// Specifies the folder path containing the tsserver and lib*.d.ts files to use.
"typescript.tsdk": null,

// Enables logging of the TS server to a file. This log can be used to diagnose TS server issues. The log may contain file paths, source code, and other potentially sensitive information from your project.
"typescript.tsserver.log": "off",

// Enables tracing of messages sent to the TS server. This trace can be used to diag
```

```
nose TS Server issues. The trace may contain file paths, source code, and other potentially sensitive information from your project.

"typescript.tsserver.trace": "off",

// Complete functions with their parameter signature.
"typescript.useCodeSnippetsOnMethodSuggest": false,

// Enable/disable TypeScript validation.
"typescript.validate.enable": true,

// CSS

// Invalid number of parameters
"css.lint.argumentsInColorFunction": "error",

// Do not use width or height when using padding or border
"css.lint.boxModel": "ignore",

// When using a vendor-specific prefix make sure to also include all other vendor-specific properties
"css.lint.compatibleVendorPrefixes": "ignore",

// Do not use duplicate style definitions
"css.lint.duplicateProperties": "ignore",

// Do not use empty rulesets
"css.lint.emptyRules": "warning",

// Avoid using 'float'. Floats lead to fragile CSS that is easy to break if one aspect of the layout changes.
"css.lint.float": "ignore",

// @font-face rule must define 'src' and 'font-family' properties
"css.lint.fontFaceProperties": "warning",

// Hex colors must consist of three or six hex numbers
"css.lint.hexColorLength": "error",

// Selectors should not contain IDs because these rules are too tightly coupled with the HTML.
"css.lint.idSelector": "ignore",

// IE hacks are only necessary when supporting IE7 and older
"css.lint.ieHack": "ignore",

// Avoid using !important. It is an indication that the specificity of the entire CSS has gotten out of control and needs to be refactored.
"css.lint.important": "ignore",

// Import statements do not load in parallel
"css.lint.importStatement": "ignore",

// Property is ignored due to the display. E.g. with 'display: inline', the width, h
```

```
eight, margin-top, margin-bottom, and float properties have no effect
"css.lint.propertyIgnoredDueToDisplay": "warning",

// The universal selector (*) is known to be slow
"css.lint.universalSelector": "ignore",

// Unknown property.
"css.lint.unknownProperties": "warning",

// Unknown vendor specific property.
"css.lint.unknownVendorSpecificProperties": "ignore",

// When using a vendor-specific prefix also include the standard property
"css.lint.vendorPrefix": "warning",

// No unit for zero needed
"css.lint.zeroUnits": "ignore",

// Traces the communication between VS Code and the CSS language server.
"css.trace.server": "off",

// Enables or disables all validations
"css.validate": true,

// LESS

// Invalid number of parameters
"less.lint.argumentsInColorFunction": "error",

// Do not use width or height when using padding or border
"less.lint.boxModel": "ignore",

// When using a vendor-specific prefix make sure to also include all other vendor-specific properties
"less.lint.compatibleVendorPrefixes": "ignore",

// Do not use duplicate style definitions
"less.lint.duplicateProperties": "ignore",

// Do not use empty rulesets
"less.lint.emptyRules": "warning",

// Avoid using 'float'. Floats lead to fragile CSS that is easy to break if one aspect of the layout changes.
"less.lint.float": "ignore",

// @font-face rule must define 'src' and 'font-family' properties
"less.lint.fontFaceProperties": "warning",

// Hex colors must consist of three or six hex numbers
"less.lint.hexColorLength": "error",

// Selectors should not contain IDs because these rules are too tightly coupled with
```

```
the HTML.  
"less.lint.idSelector": "ignore",  
  
// IE hacks are only necessary when supporting IE7 and older  
"less.lint.ieHack": "ignore",  
  
// Avoid using !important. It is an indication that the specificity of the entire CS  
S has gotten out of control and needs to be refactored.  
"less.lint.important": "ignore",  
  
// Import statements do not load in parallel  
"less.lint.importStatement": "ignore",  
  
// Property is ignored due to the display. E.g. with 'display: inline', the width, h  
eight, margin-top, margin-bottom, and float properties have no effect  
"less.lint.propertyIgnoredDueToDisplay": "warning",  
  
// The universal selector (*) is known to be slow  
"less.lint.universalSelector": "ignore",  
  
// Unknown property.  
"less.lint.unknownProperties": "warning",  
  
// Unknown vendor specific property.  
"less.lint.unknownVendorSpecificProperties": "ignore",  
  
// When using a vendor-specific prefix also include the standard property  
"less.lint.vendorPrefix": "warning",  
  
// No unit for zero needed  
"less.lint.zeroUnits": "ignore",  
  
// Enables or disables all validations  
"less.validate": true,  
  
// SCSS (Sass)  
  
// Invalid number of parameters  
"scss.lint.argumentsInColorFunction": "error",  
  
// Do not use width or height when using padding or border  
"scss.lint.boxModel": "ignore",  
  
// When using a vendor-specific prefix make sure to also include all other vendor-sp  
ecific properties  
"scss.lint.compatibleVendorPrefixes": "ignore",  
  
// Do not use duplicate style definitions  
"scss.lint.duplicateProperties": "ignore",  
  
// Do not use empty rulesets  
"scss.lint.emptyRules": "warning",
```

```
// Avoid using 'float'. Floats lead to fragile CSS that is easy to break if one aspect of the layout changes.  
"scss.lint.float": "ignore",  
  
// @font-face rule must define 'src' and 'font-family' properties  
"scss.lint.fontFaceProperties": "warning",  
  
// Hex colors must consist of three or six hex numbers  
"scss.lint.hexColorLength": "error",  
  
// Selectors should not contain IDs because these rules are too tightly coupled with the HTML.  
"scss.lint.idSelector": "ignore",  
  
// IE hacks are only necessary when supporting IE7 and older  
"scss.lint.ieHack": "ignore",  
  
// Avoid using !important. It is an indication that the specificity of the entire CSS has gotten out of control and needs to be refactored.  
"scss.lint.important": "ignore",  
  
// Import statements do not load in parallel  
"scss.lint.importStatement": "ignore",  
  
// Property is ignored due to the display. E.g. with 'display: inline', the width, height, margin-top, margin-bottom, and float properties have no effect  
"scss.lint.propertyIgnoredDueToDisplay": "warning",  
  
// The universal selector (*) is known to be slow  
"scss.lint.universalSelector": "ignore",  
  
// Unknown property.  
"scss.lint.unknownProperties": "warning",  
  
// Unknown vendor specific property.  
"scss.lint.unknownVendorSpecificProperties": "ignore",  
  
// When using a vendor-specific prefix also include the standard property  
"scss.lint.vendorPrefix": "warning",  
  
// No unit for zero needed  
"scss.lint.zeroUnits": "ignore",  
  
// Enables or disables all validations  
"scss.validate": true,  
  
// Extensions  
  
// Automatically update extensions  
"extensions.autoUpdate": true,  
  
// If set to true, the notifications for extension recommendations will stop showing up.
```

```
"extensions.ignoreRecommendations": false,  
  
// External Terminal  
  
// Customizes what kind of terminal to launch.  
"terminal.explorerKind": "integrated",  
  
// Customizes which terminal to run on Linux.  
"terminal.external.linuxExec": "xterm",  
  
// Customizes which terminal application to run on OS X.  
"terminal.external.osxExec": "Terminal.app",  
  
// Customizes which terminal to run on Windows.  
"terminal.external.windowsExec": "C:\\WINDOWS\\System32\\cmd.exe",  
  
// Integrated Terminal  
  
// A set of command IDs whose keybindings will not be sent to the shell and instead  
always be handled by Code. This allows the use of keybindings that would normally be c  
onsumed by the shell to act the same as when the terminal is not focused, for example  
ctrl+p to launch Quick Open.  
"terminal.integrated.commandsToSkipShell": [  
    "editor.action.toggleTabFocusMode",  
    "workbench.action.debug.continue",  
    "workbench.action.debug.pause",  
    "workbench.action.debug.restart",  
    "workbench.action.debug.run",  
    "workbench.action.debug.start",  
    "workbench.action.debug.stop",  
    "workbench.action.focusActiveEditorGroup",  
    "workbench.action.focusFirstEditorGroup",  
    "workbench.action.focusSecondEditorGroup",  
    "workbench.action.focusThirdEditorGroup",  
    "workbench.action.navigateDown",  
    "workbench.action.navigateLeft",  
    "workbench.action.navigateRight",  
    "workbench.action.navigateUp",  
    "workbench.action.openNextRecentlyUsedEditorInGroup",  
    "workbench.action.openPreviousRecentlyUsedEditorInGroup",  
    "workbench.action.quickOpen",  
    "workbench.action.quickOpenPreviousEditor",  
    "workbench.action.quickOpenView",  
    "workbench.action.showCommands",  
    "workbench.action.tasks.build",  
    "workbench.action.tasks.restartTask",  
    "workbench.action.tasks.runTask",  
    "workbench.action.tasks.showLog",  
    "workbench.action.tasks.showTasks",  
    "workbench.action.tasks.terminate",  
    "workbench.action.tasks.test",  
    "workbench.action.terminal.clear",  
    "workbench.action.terminal.copySelection",  
]
```

```
"workbench.action.terminal.deleteWordLeft",
"workbench.action.terminal.deleteWordRight",
"workbench.action.terminal.findWidget.history.showNext",
"workbench.action.terminal.findWidget.history.showPrevious",
"workbench.action.terminal.focus",
"workbench.action.terminal.focusAtIndex1",
"workbench.action.terminal.focusAtIndex2",
"workbench.action.terminal.focusAtIndex3",
"workbench.action.terminal.focusAtIndex4",
"workbench.action.terminal.focusAtIndex5",
"workbench.action.terminal.focusAtIndex6",
"workbench.action.terminal.focusAtIndex7",
"workbench.action.terminal.focusAtIndex8",
"workbench.action.terminal.focusAtIndex9",
"workbench.action.terminal.focusFindWidget",
"workbench.action.terminal.focusNext",
"workbench.action.terminal.focusPrevious",
"workbench.action.terminal.hideFindWidget",
"workbench.action.terminal.kill",
"workbench.action.terminal.new",
"workbench.action.terminal.paste",
"workbench.action.terminal.runActiveFile",
"workbench.action.terminal.runSelectedText",
"workbench.action.terminal.scrollDown",
"workbench.action.terminal.scrollDownPage",
"workbench.action.terminal.scrollToBottom",
"workbench.action.terminal.scrollToTop",
"workbench.action.terminal.scrollUp",
"workbench.action.terminal.scrollUpPage",
"workbench.action.terminal.selectAll",
"workbench.action.terminal.toggleTerminal",
"workbench.action.togglePanel"
],
// Whether to confirm on exit if there are active terminal sessions.
"terminal.integrated.confirmOnExit": false,
// Controls whether the terminal cursor blinks.
"terminal.integrated.cursorBlinking": false,
// Controls the style of terminal cursor.
"terminal.integrated.cursorStyle": "block",
// An explicit start path where the terminal will be launched, this is used as the current working directory (cwd) for the shell process. This may be particularly useful in workspace settings if the root directory is not a convenient cwd.
"terminal.integrated.cwd": "",
// Whether to enable bold text within the terminal, note that this requires support from the terminal shell.
"terminal.integrated.enableBold": true,
// Object with environment variables that will be added to the VS Code process to be
```

```
used by the terminal on Linux
"terminal.integrated.env.linux": {},  
  
// Object with environment variables that will be added to the VS Code process to be
used by the terminal on OS X
"terminal.integrated.env.osx": {},  
  
// Object with environment variables that will be added to the VS Code process to be
used by the terminal on Windows
"terminal.integrated.env.windows": {},  
  
// Controls the font family of the terminal, this defaults to editor.fontFamily's va-
lue.
"terminal.integrated.fontFamily": "",  
  
// Controls the font size in pixels of the terminal.
"terminal.integrated.fontSize": 14,  
  
// Controls the line height of the terminal, this number is multiplied by the termin-
al font size to get the actual line-height in pixels.
"terminal.integrated.lineHeight": 1,  
  
// When set, this will prevent the context menu from appearing when right clicking w-
ithin the terminal, instead it will copy when there is a selection and paste when ther-
e is no selection.
"terminal.integrated.rightClickCopyPaste": true,  
  
// Controls the maximum amount of lines the terminal keeps in its buffer.
"terminal.integrated.scrollback": 1000,  
  
// Controls whether locale variables are set at startup of the terminal, this defau-
lts to true on OS X, false on other platforms.
"terminal.integrated.setLocaleVariables": false,  
  
// The path of the shell that the terminal uses on Linux.
"terminal.integrated.shell.linux": "sh",  
  
// The path of the shell that the terminal uses on OS X.
"terminal.integrated.shell.osx": "sh",  
  
// The path of the shell that the terminal uses on Windows. When using shells shippe-
d with Windows (cmd, PowerShell or Bash on Ubuntu).
"terminal.integrated.shell.windows": "C:\\WINDOWS\\System32\\WindowsPowerShell\\v1.0
\\powershell.exe",  
  
// The command line arguments to use when on the Linux terminal.
"terminal.integrated.shellArgs.linux": [],  
  
// The command line arguments to use when on the OS X terminal.
"terminal.integrated.shellArgs.osx": [
  "-l"
],
```

```
// The command line arguments to use when on the Windows terminal.  
"terminal.integrated.shellArgs.windows": [],  
  
// Problems  
  
// Controls if Problems view should automatically reveal files when opening them  
"problems.autoReveal": true,  
  
// Show Errors & Warnings on files and folder.  
"problems.decorations.enabled": false,  
  
// Telemetry  
  
// Enable crash reports to be sent to Microsoft.  
// This option requires restart to take effect.  
"telemetry.enableCrashReporter": true,  
  
// Enable usage data and errors to be sent to Microsoft.  
"telemetry.enableTelemetry": true,  
  
// Default Configuration Overrides  
  
// Configure editor settings to be overridden for [git-commit] language.  
"[git-commit)": {  
    "editor.rulers": [  
        72  
    ]  
},  
  
// Configure editor settings to be overridden for [go] language.  
"[go)": {  
    "editor.insertSpaces": false  
},  
  
// Configure editor settings to be overridden for [json] language.  
"[json)": {  
    "editor.quickSuggestions": {  
        "strings": true  
    }  
},  
  
// Configure editor settings to be overridden for [makefile] language.  
"[makefile)": {  
    "editor.insertSpaces": false  
},  
  
// Configure editor settings to be overridden for [markdown] language.  
"[markdown)": {  
    "editor.wordWrap": "on",  
    "editor.quickSuggestions": false  
},  
  
// Configure editor settings to be overridden for [yaml] language.
```

```

"[yaml)": {
  "editor.insertSpaces": true,
  "editor.tabSize": 2
},
// Npm

// Controls whether auto detection of npm scripts is on or off. Default is on.
"npm.autoDetect": "on",

// The package manager used to run scripts.
"npm.packageManager": "npm",

// Run npm commands with the `--silent` option.
"npm.runSilent": false,

// Merge Conflict

// Enable/disable merge conflict block CodeLens within editor
"merge-conflict.codeLens.enabled": true,

// Enable/disable merge conflict decorators within editor
"merge-conflict.decorators.enabled": true,

// Emmet

// An array of languages where Emmet abbreviations should not be expanded.
"emmet.excludeLanguages": [
  "markdown"
],
// Path to a folder containing Emmet profiles and snippets.'
"emmet.extensionsPath": null,
// Enable Emmet abbreviations in languages that are not supported by default. Add a
mapping here between the language and emmet supported language.
// Eg: {"vue-html": "html", "javascript": "javascriptreact"}
"emmet.includeLanguages": {},

// Preferences used to modify behavior of some actions and resolvers of Emmet.
"emmet.preferences": {},

// Shows possible Emmet abbreviations as suggestions. Not applicable in stylesheets
or when emmet.showExpandedAbbreviation is set to "never".
"emmet.showAbbreviationSuggestions": true,
// Shows expanded Emmet abbreviations as suggestions.
// The option "inMarkupAndStylesheetFilesOnly" applies to html, haml, jade, slim, xm
l, xsl, css, scss, sass, less and stylus.
// The option "always" applies to all parts of the file regardless of markup/css.
"emmet.showExpandedAbbreviation": "always",
// If true, then Emmet suggestions will show up as snippets allowing you to order th

```

```
em as per editor.snippetSuggestions setting.  
"emmet.showSuggestionsAsSnippets": false,  
  
// Define profile for specified syntax or use your own profile with specific rules.  
"emmet.syntaxProfiles": {},  
  
// When enabled, Emmet abbreviations are expanded when pressing TAB.  
"emmet.triggerExpansionOnTab": false,  
  
// Variables to be used in Emmet snippets  
"emmet.variables": {},  
  
// Git  
  
// Whether auto fetching is enabled  
"git.autofetch": false,  
  
// Whether auto refreshing is enabled  
"git.autorefresh": true,  
  
// Controls what type of branches are listed when running `Checkout to...`. `all` shows all refs, `local` shows only the local branches, `tags` shows only tags and `remote` shows only remote branches.  
"git.checkoutType": "all",  
  
// Confirm before synchronizing git repositories  
"git.confirmSync": true,  
  
// Controls the git badge counter. `all` counts all changes. `tracked` counts only the tracked changes. `off` turns it off.  
"git.countBadge": "all",  
  
// Controls if Git contributes colors and badges to the explorer and the open editor's view.  
"git.decorations.enabled": true,  
  
// The default location where to clone a git repository  
"git.defaultCloneDirectory": null,  
  
// Enables commit signing with GPG.  
"git.enableCommitSigning": false,  
  
// Whether git is enabled  
"git.enabled": true,  
  
// Commit all changes when there are no staged changes.  
"git.enableSmartCommit": false,  
  
// Ignores the legacy Git warning  
"git.ignoreLegacyWarning": false,  
  
// Ignores the warning when there are too many changes in a repository  
"git.ignoreLimitWarning": false,
```

```
// Ignores the warning when Git is missing
"git.ignoreMissingGitWarning": false,

// Path to the git executable
"git.path": null,

// Jake

// Controls whether auto detection of Jake tasks is on or off. Default is on.
"jake.autoDetect": "on",

// Grunt

// Controls whether auto detection of Grunt tasks is on or off. Default is on.
"grunt.autoDetect": "on",

// Gulp

// Controls whether auto detection of Gulp tasks is on or off. Default is on.
"gulp.autoDetect": "on"
}
```

Common Questions

Q: VS Code says "Unable to write settings."

A: If you try to change a setting (for example turning on Auto Save or selecting a new Color Theme) and you see "Unable to write settings. Please open User Settings to correct errors/warnings in the file and try again.", it means your `settings.json` file is ill-formed or has errors. The errors can be as simple as a missing comma or setting value. Open the Settings editor **File > Preferences > Settings (Code > Preferences > Settings on Mac)** (`kb(workbench.action.openGlobalSettings)`) and you should see the error highlighted with red squiggles.

Q: When does it make sense to use workspace settings?

A: If you're using a workspace that needs custom settings but you don't want to apply them to your other VS Code projects. A good example is language-specific linting rules.

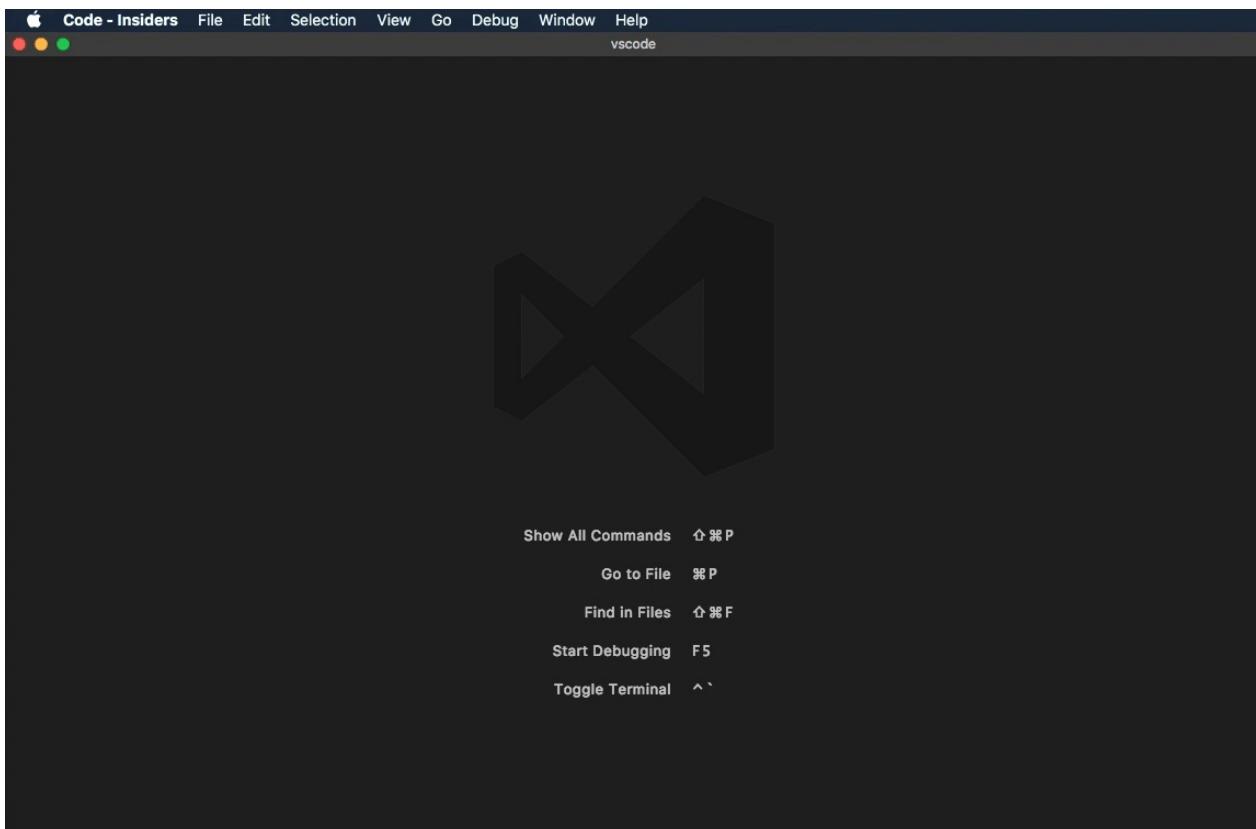
Key Bindings for Visual Studio Code

Visual Studio Code lets you perform most tasks directly from the keyboard. This page lists out the default bindings (keyboard shortcuts) and describes how you can update them.

Note: If you visit this page on a Mac, you will see the key bindings for the Mac. If you visit using Windows or Linux, you will see the keys for that platform. If you need the key binding for another platform, hover your mouse over the key you are interested in.

Keyboard Shortcuts Editor

Visual Studio Code provides a rich and easy keyboard shortcuts editing experience using **Keyboard Shortcuts** editor. It lists all available commands with and without keybindings and you can easily change / remove / reset their keybindings using the available actions. It also has a search box on the top that helps you in finding commands or keybindings. You can open this editor by going to the menu under **File > Preferences > Keyboard Shortcuts**. (**Code > Preferences > Keyboard Shortcuts** on Mac)



Most importantly, you can see keybindings according to your keyboard layout. For example, key binding `cmd+\`` in US keyboard layout will be shown as `ctrl+shift+alt+cmd+7` when layout is changed to German. The dialog to enter key binding will assign the correct and

desired key binding as per your keyboard layout.

For doing more advanced keyboard shortcut customization, read [Advanced Customization](#).

Keymap Extensions

Keyboard shortcuts are vital to productivity and changing keyboarding habits can be tough. To help with this, **File > Preferences > Keymap Extensions** shows you a list of popular keymap extensions. These extensions modify the VS Code shortcuts to match those of other editors so you don't need to learn new keyboard shortcuts. There is also a [Keymaps category](#) of extensions in the Marketplace.

Tip: Click on an extension tile above to read the description and reviews to decide which extension is best for you. See more in the [Marketplace](#).

Keyboard Shortcuts Reference

We also have a printable version of these keyboard shortcuts. **Help > Keyboard Shortcut Reference** displays a condensed PDF version suitable for printing as an easy reference.

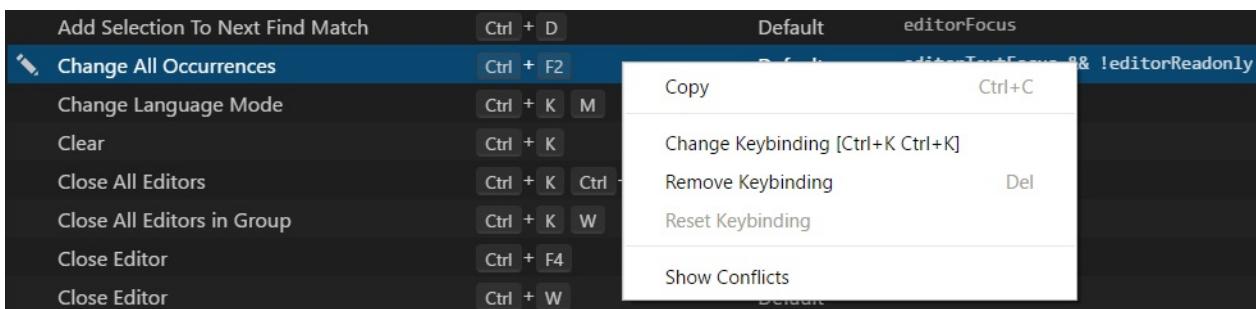
Below are links to the three platform-specific versions:

- [Windows](#)
- [macOS](#)
- [Linux](#)

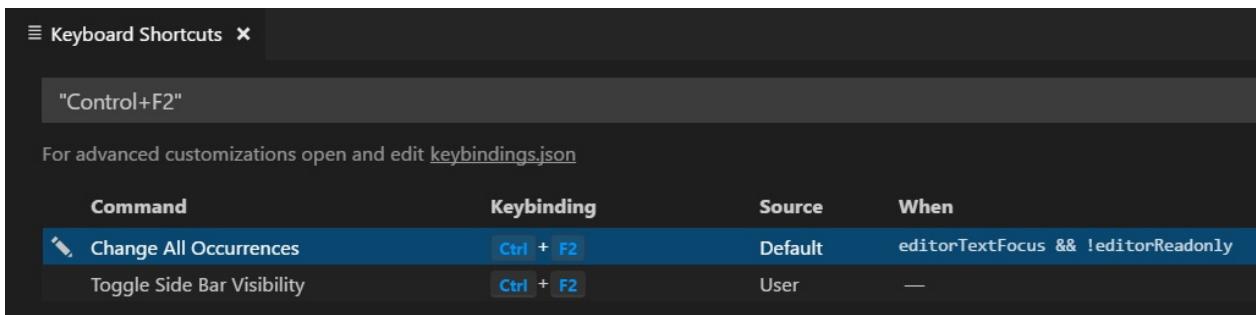
Detecting keybinding conflicts

If you have many extensions installed or you have [customized](#) your keyboard shortcuts, you can sometimes have keybinding conflicts where the same keyboard shortcut is mapped to several commands. This can result in confusing behavior, especially if different keybindings are going in and out of scope as you move around the editor.

The **Keyboard Shortcuts** editor has a context menu command **Show Conflicts**, which will filter the keybindings based on a keyboard shortcut to display conflicts.



Pick a command with the keybinding you think is overloaded and you can see if multiple commands are defined, the source of the keybindings and when they are active.



Default Keyboard Shortcuts

Note: The following keys are rendered assuming a standard US keyboard layout. If you use a different keyboard layout, please [read below](#). You can view the currently active keyboard shortcuts in VS Code in the **Command Palette** (**View -> Command Palette**) or in the **Keyboard Shortcuts** editor (**File > Preferences > Keyboard Shortcuts**).

Basic Editing

Key	Command	
kb(editor.action.clipboardCutAction)	Cut line (empty selection)	editor.action
kb(editor.action.clipboardCopyAction)	Copy line (empty selection)	editor.action
kb(editor.action.deleteLines)	Delete Line	editor.action
kb(editor.action.insertLineAfter)	Insert Line Below	editor.action
kb(editor.action.insertLineBefore)	Insert Line Above	editor.action
kb(editor.action.moveLinesDownAction)	Move Line Down	editor.action

kb(editor.action.moveLinesUpAction)	Move Line Up	editor.action
kb(editor.action.copyLinesDownAction)	Copy Line Down	editor.action
kb(editor.action.copyLinesUpAction)	Copy Line Up	editor.action
kb(editor.action.addSelectionToNextFindMatch)	Add Selection To Next Find Match	editor.action
kb(editor.action.moveSelectionToNextFindMatch)	Move Last Selection To Next Find Match	editor.action
kb(cursorUndo)	Undo last cursor operation	cursorUndo
kb(editor.action.insertCursorAtEndOfEachLineSelected)	Insert cursor at end of each line selected	editor.action
kb(editor.action.selectHighlights)	Select all occurrences of current selection	editor.action
kb(editor.action.changeAll)	Select all occurrences of current word	editor.action
kb(expandLineSelection)	Select current line	expandLineSel
kb(editor.action.insertCursorBelow)	Insert Cursor Below	editor.action
kb(editor.action.insertCursorAbove)	Insert Cursor Above	editor.action
kb(editor.action.jumpToBracket)	Jump to matching bracket	editor.action
kb(editor.action.indentLines)	Indent Line	editor.action
kb(editor.action.outdentLines)	Outdent Line	editor.action

kb(cursorHome)	Go to Beginning of Line	cursorHome
kb(cursorEnd)	Go to End of Line	cursorEnd
kb(cursorBottom)	Go to End of File	cursorBottom
kb(cursorTop)	Go to Beginning of File	cursorTop
kb(scrollLineDown)	Scroll Line Down	scrollLineDown
kb(scrollLineUp)	Scroll Line Up	scrollLineUp
kb(scrollPageDown)	Scroll Page Down	scrollPageDown
kb(scrollPageUp)	Scroll Page Up	scrollPageUp
kb(editor.fold)	Fold (collapse) region	editor.fold
kb(editor.unfold)	Unfold (uncollapse) region	editor.unfold
kb(editor.foldRecursively)	Fold (collapse) all subregions	editor.foldRe
kb(editor.unfoldRecursively)	Unfold (uncollapse) all subregions	editor.unfold
kb(editor.foldAll)	Fold (collapse) all regions	editor.foldAl
kb(editor.unfoldAll)	Unfold (uncollapse) all regions	editor.unfold
kb(editor.action.addCommentLine)	Add Line Comment	editor.action
kb(editor.action.removeCommentLine)	Remove Line Comment	editor.action

kb(editor.action.commentLine)	Toggle Line Comment	editor.action
kb(editor.action.blockComment)	Toggle Block Comment	editor.action
kb(actions.find)	Find	actions.find
kb(editor.action.startFindReplaceAction)	Replace	editor.action
kb(editor.action.nextMatchFindAction)	Find Next	editor.action
kb(editor.action.previousMatchFindAction)	Find Previous	editor.action
kb(editor.action.selectAllMatches)	Select All Occurrences of Find Match	editor.action
kb(toggleFindCaseSensitive)	Toggle Find Case Sensitive	toggleFindCas
kb(toggleFindRegex)	Toggle Find Regex	toggleFindReg
kb(toggleFindWholeWord)	Toggle Find Whole Word	toggleFindWho
kb(editor.action.toggleTabFocusMode)	Toggle Use of Tab Key for Setting Focus	editor.action
kb(toggleRenderWhitespace)	Toggle Render Whitespace	toggleRenderW
kb(editor.action.toggleWordwrap)	Toggle Word Wrap	editor.action

Rich Languages Editing

Key	Command	Comm
kb(editor.action.triggerSuggest)	Trigger Suggest	editor.action.trigger
kb(editor.action.triggerParameterHints)	Trigger Parameter Hints	editor.action.trigger
kb(editor.action.formatDocument)	Format Document	editor.action.formatD
kb(editor.action.formatSelection)	Format Selection	editor.action.formats
kb(editor.action.gotoDeclaration)	Go to Definition	editor.action.goToDec
kb(editor.action.showHover)	Show Hover	editor.action.showHov
kb(editor.action.previewDeclaration)	Peek Definition	editor.action.preview
kb(editor.action.openDeclarationToTheSide)	Open Definition to the Side	editor.action.openDec
kb(editor.action.quickFix)	Quick Fix	editor.action.quickFi
kb(editor.action.referenceSearch.trigger)	Show References	editor.action.referen
kb(editor.action.rename)	Rename Symbol	editor.action.rename
kb(editor.action.inPlaceReplace.down)	Replace with Next Value	editor.action.inPlace
kb(editor.action.inPlaceReplace.up)	Replace with Previous Value	editor.action.inPlace
kb(editor.action.smartSelect.grow)	Expand AST Select	editor.action.smartSe
kb(editor.action.smartSelect.shrink)	Shrink AST Select	editor.action.smartSe
kb(editor.action.trimTrailingWhitespace)	Trim Trailing Whitespace	editor.action.trimTra
kb(workbench.action.editor.changeLanguageMode)	Change Language Mode	workbench.action.edit

Navigation

Key	Command	
kb(workbench.action.showAllSymbols)	Show All Symbols	workbench
kb(workbench.action.gotoLine)	Go to Line...	workbench
kb(workbench.action.quickOpen)	Go to File..., Quick Open	workbench
kb(workbench.action.gotoSymbol)	Go to Symbol...	workbench
kb(workbench.actions.view.problems)	Show Problems	workbench
kb(editor.action.marker.next)	Go to Next Error or Warning	editor.action.marker.next
kb(editor.action.marker.prev)	Go to Previous Error or Warning	editor.action.marker.prev
kb(workbench.action.showCommands)	Show All Commands	workbench
kb(workbench.action.openPreviousRecentlyUsedEditorInGroup)	Navigate Editor Group History	workbench
kb(workbench.action.navigateBack)	Go Back	workbench
kb(workbench.action.navigateForward)	Go Forward	workbench

Editor/Window Management

Key	Command	Command
kb(workbench.action.newWindow)	New Window	workbench.action.newWindow
kb(workbench.action.closeWindow)	Close Window	workbench.action.closeWindow
kb(workbench.action.closeActiveEditor)	Close Editor	workbench.action.closeEditor

kb(workbench.action.closeFolder)	Close Folder	workbench.action.clos
kb(workbench.action.navigateEditorGroups)	Cycle Between Editor Groups	workbench.action.navi
kb(workbench.action.splitEditor)	Split Editor	workbench.action.split
kb(workbench.action.focusFirstEditorGroup)	Focus into First Editor Group	workbench.action.focus
kb(workbench.action.focusSecondEditorGroup)	Focus into Second Editor Group	workbench.action.focus
kb(workbench.action.focusThirdEditorGroup)	Focus into Third Editor Group	workbench.action.focus
kb(workbench.action.focusPreviousGroup)	Focus into Editor Group on the Left	workbench.action.focus
kb(workbench.action.focusNextGroup)	Focus into Editor Group on the Right	workbench.action.focus
kb(workbench.action.moveEditorLeftInGroup)	Move Editor Left	workbench.action.move
kb(workbench.action.moveEditorRightInGroup)	Move Editor Right	workbench.action.move
kb(workbench.action.moveActiveEditorGroupLeft)	Move Active Editor Group Left	workbench.action.move
kb(workbench.action.moveActiveEditorGroupRight)	Move Active Editor Group Right	workbench.action.move
kb(workbench.action.moveEditorToNextGroup)	Move Editor into Next Group	workbench.action.move

<code>kb(workbench.action.moveEditorToPreviousGroup)</code>	Move Editor into Previous Group	<code>workbench.action.move</code>
---	---------------------------------	------------------------------------

File Management

Key	Command	
<code>kb(workbench.action.files.newUntitledFile)</code>	New File	<code>workbench.</code>
<code>kb(workbench.action.files.openFile)</code>	Open File...	<code>workbench.</code>
<code>kb(workbench.action.files.save)</code>	Save	<code>workbench.</code>
<code>kb(workbench.action.files.saveAll)</code>	Save All	<code>workbench.</code>
<code>kb(workbench.action.files.saveAs)</code>	Save As...	<code>workbench.</code>
<code>kb(workbench.action.closeActiveEditor)</code>	Close	<code>workbench.</code>
<code>kb(workbench.action.closeOtherEditors)</code>	Close Others	<code>workbench.</code>
<code>kb(workbench.action.closeEditorsInGroup)</code>	Close Group	<code>workbench.</code>
<code>kb(workbench.action.closeEditorsInOtherGroups)</code>	Close Other Groups	<code>workbench.</code>
<code>kb(workbench.action.closeEditorsToLeft)</code>	Close Group to Left	<code>workbench.</code>
<code>kb(workbench.action.closeEditorsToTheRight)</code>	Close Group to Right	<code>workbench.</code>
<code>kb(workbench.action.closeAllEditors)</code>	Close All	<code>workbench.</code>
<code>kb(workbench.action.reopenClosedEditor)</code>	Reopen Closed Editor	<code>workbench.</code>
<code>kb(workbench.action.keepEditor)</code>	Keep Open	<code>workbench.</code>
<code>kb(workbench.action.openNextRecentlyUsedEditorInGroup)</code>	Open Next	<code>workbench.</code>
<code>kb(workbench.action.openPreviousRecentlyUsedEditorInGroup)</code>	Open Previous	<code>workbench.</code>
<code>kb(workbench.action.files.copyPathOfFile)</code>	Copy Path of Active File	<code>workbench.</code>

kb(workbench.action.files.revealActiveFileInWindows)	Reveal Active File in Windows	workbench.
kb(workbench.action.files.showOpenedFileInNewWindow)	Show Opened File in New Window	workbench.
kb(workbench.files.action.compareFileWith)	Compare Opened File With	workbench.

Display

Key	Command	Command
kb(workbench.action.toggleFullScreen)	Toggle Full Screen	workbench.action.toggleFullScreen
kb(workbench.action.toggleZenMode)	Toggle Zen Mode	workbench.action.toggleZenMode
kb(workbench.action.exitZenMode)	Leave Zen Mode	workbench.action.exitZenMode
kb(workbench.action.zoomIn)	Zoom in	workbench.action.zoomIn
kb(workbench.action.zoomOut)	Zoom out	workbench.action.zoomOut
kb(workbench.action.zoomReset)	Reset Zoom	workbench.action.zoomReset
kb(workbench.action.toggleSidebarVisibility)	Toggle Sidebar Visibility	workbench.action.toggleSidebarVisibility
kb(workbench.view.explorer)	Show Explorer / Toggle Focus	workbench.view.explorer
kb(workbench.view.search)	Show Search	workbench.view.search
kb(workbench.view.scm)	Show Source Control	workbench.view.scm
kb(workbench.view.debug)	Show Debug	workbench.view.debug
kb(workbench.view.extensions)	Show	workbench.view.extensions

kb(workbench.view.extensions)	Extensions	workbench.view.extensions
kb(workbench.action.output.toggleOutput)	Show Output	workbench.action.output.toggleOutput
kb(workbench.action.quickOpenView)	Quick Open View	workbench.action.quickOpenView
kb(workbench.action.terminal.openNativeConsole)	Open New Command Prompt	workbench.action.terminal.openNativeConsole
kb(markdown.showPreview)	Toggle Markdown Preview	markdown.showPreview
kb(markdown.showPreviewToSide)	Open Preview to the Side	markdown.showPreviewToSide
kb(workbench.action.terminal.toggleTerminal)	Toggle Integrated Terminal	workbench.action.terminal.toggleTerminal

Search

Key	Command	Command
kb(workbench.view.search)	Show Search	workbench.view.search
kb(workbench.action.replaceInFiles)	Replace in Files	workbench.action.replaceInFiles
kb(toggleSearchCaseSensitive)	Toggle Match Case	toggleSearchCaseSensitive
kb(toggleSearchWholeWord)	Toggle Match Whole Word	toggleSearchWholeWord
kb(toggleSearchRegex)	Toggle Use Regular Expression	toggleSearchRegex
kb(workbench.action.search.toggleQueryDetails)	Toggle Search Details	workbench.action.search.toggleQueryDetails
kb(search.action.focusNextSearchResult)	Focus Next Search Result	search.action.focusNextSearchResult
kb(search.action.focusPreviousSearchResult)	Focus Previous Search Result	search.action.focusPreviousSearchResult
kb(search.history.showNext)	Show Next Search Term	search.history.showNextSearchTerm
kb(search.history.showPrevious)	Show Previous Search Term	search.history.showPreviousSearchTerm

Preferences

Key	Command	Command id
kb(workbench.action.openGlobalSettings)	Open User Settings	workbench.action.openGlobalSettings
kb(workbench.action.openWorkspaceSettings)	Open Workspace Settings	workbench.action.openWorkspaceSettings
kb(workbench.action.openGlobalKeybindings)	Open Keyboard Shortcuts	workbench.action.openGlobalKeybindings
kb(workbench.action.openSnippets)	Open User Snippets	workbench.action.openSnippets
kb(workbench.action.selectTheme)	Select Color Theme	workbench.action.selectTheme
kb(workbench.action.configureLocale)	Configure Display Language	workbench.action.configureLocale

Debug

Key	Command	Command id
kb(editor.debug.action.toggleBreakpoint)	Toggle Breakpoint	editor.debug.action.toggleBreakpoint
kb(workbench.action.debug.start)	Start	workbench.action.debug.start
kb(workbench.action.debug.continue)	Continue	workbench.action.debug.continue
kb(workbench.action.debug.run)	Start (without debugging)	workbench.action.debug.run
kb(workbench.action.debug.pause)	Pause	workbench.action.debug.pause
kb(workbench.action.debug.stepInto)	Step Into	workbench.action.debug.stepInto
kb(workbench.action.debug.stepOut)	Step Out	workbench.action.debug.stepOut
kb(workbench.action.debug.stepOver)	Step Over	workbench.action.debug.stepOver
kb(workbench.action.debug.stop)	Stop	workbench.action.debug.stop
kb(editor.debug.action.showDebugHover)	Show Hover	editor.debug.action.showDebugHover

Tasks

Key	Command	Command id
kb(workbench.action.tasks.build)	Run Build Task	workbench.action.tasks.build
kb(workbench.action.tasks.test)	Run Test Task	workbench.action.tasks.test

Extensions

Key	Command	
kb(workbench.extensions.action.installExtension)	Install Extension	workber
kb(workbench.extensions.action.showInstalledExtensions)	Show Installed Extensions	workber
kb(workbench.extensions.action.listOutdatedExtensions)	Show Outdated Extensions	workber
kb(workbench.extensions.action.showRecommendedExtensions)	Show Recommended Extensions	workber
kb(workbench.extensions.action.showPopularExtensions)	Show Popular Extensions	workber
kb(workbench.extensions.action.updateAllExtensions)	Update All Extensions	workber

Advanced customization

All keyboard shortcuts in VS Code can be customized via the `keybindings.json` file.

- To configure keyboard shortcuts the way you want, open **Keyboard Shortcuts** editor and click on the link `keybindings.json`.
- This will open the **Default Keyboard Shortcuts** on the left and your `keybindings.json` file where you can overwrite the default bindings on the right.
- The list above isn't exhaustive. More commands may be listed under "Here are other available commands" in **Default Keyboard Shortcuts**.

Keyboard Rules

The keyboard shortcuts dispatching is done by analyzing a list of rules that are expressed in JSON. Here are some examples:

```

// Keybindings that are active when the focus is in the editor
{ "key": "home",           "command": "cursorHome",           "when": "editorTextFocus" },
{ "key": "shift+home",     "command": "cursorHomeSelect",   "when": "editorTextFocus" },

// Keybindings that are complementary
{ "key": "f5",             "command": "workbench.action.debug.continue", "when": "inDebugMode" },
{ "key": "f5",             "command": "workbench.action.debug.start",    "when": "!inDebugMode" },

// Global keybindings
{ "key": "ctrl+f",         "command": "actions.find" },
{ "key": "alt+left",        "command": "workbench.action.navigateBack" },
{ "key": "alt+right",       "command": "workbench.action.navigateForward" },

// Global keybindings using chords (two separate keypress actions)
{ "key": "ctrl+k enter",   "command": "workbench.action.keepEditor" },
{ "key": "ctrl+k ctrl+w",   "command": "workbench.action.closeAllEditors" },

```

Each rule consists of:

- a `key` that describes the pressed keys.
- a `command` containing the identifier of the command to execute.
- an **optional** `when` clause containing a boolean expression that will be evaluated depending on the current `context`.

Chords (two separate keypress actions) are described by separating the two keypresses with a space. E.g.: `kbstyle(ctrl+k ctrl+c)`.

When a key is pressed:

- the rules are evaluated from **bottom to top**.
- the first rule that matches, both the `key` and in terms of `when`, is accepted.
- no more rules are processed.
- if a rule is found and has a `command` set, the `command` is executed.

The additional `keybindings.json` rules are appended at runtime to the bottom of the default rules, thus allowing them to overwrite the default rules. The `keybindings.json` file is watched by VS Code so editing it while VS Code is running will update the rules at runtime.

Accepted keys

The `key` is made up of modifiers and the key itself.

The following modifiers are accepted:

Platform	Modifiers
Mac	kbstyle(ctrl+), kbstyle(shift+), kbstyle(alt+), kbstyle(cmd+)
Windows	kbstyle(ctrl+), kbstyle(shift+), kbstyle(alt+), kbstyle(win+)
Linux	kbstyle(ctrl+), kbstyle(shift+), kbstyle(alt+), kbstyle(meta+)

The following keys are accepted:

- kbstyle(f1-f19), kbstyle(a-z), kbstyle(0-9)
- kbstyle(`), kbstyle(-), kbstyle(=), kbstyle([]), kbstyle([]), kbstyle(\), kbstyle();), kbstyle('), kbstyle(,), kbstyle(.), kbstyle(/)
- kbstyle(left), kbstyle(up), kbstyle(right), kbstyle(down), kbstyle(pageup), kbstyle(pagedown), kbstyle(end), kbstyle(home)
- kbstyle(tab), kbstyle(enter), kbstyle(escape), kbstyle(space), kbstyle(backspace), kbstyle(delete)
- kbstyle(pausebreak), kbstyle(capslock), kbstyle(insert)
- kbstyle(numpad0-numpad9), kbstyle(numpad_multiply), kbstyle(numpad_add), kbstyle(numpad_separator)
- kbstyle(numpad_subtract), kbstyle(numpad_decimal), kbstyle(numpad_divide)

Command arguments

You can invoke a command with arguments. This is useful if you often perform the same operation on a specific file or folder. You can add a custom keyboard shortcut to do exactly what you want.

The following is an example overriding the `kbstyle(Enter)` key to print some text:

```
{ "key": "enter", "command": "type",
    "args": { "text": "Hello World" },
    "when": "editorTextFocus" }
```

The type command will receive `{"text": "Hello World"}` as its first argument and add "Hello World" to the file instead of producing the default command.

'when' clause contexts

VS Code gives you fine control over when your key bindings are enabled through the optional `when` clause. If your key binding doesn't have a `when` clause, the key binding is globally available at all times.

Below are some of the possible `when` clause contexts which evaluate to Boolean true/false:

Context name	True when
Editor contexts	
editorFocus	An editor has focus, either the text or a widget.
editorTextFocus	The text in an editor has focus (cursor is blinking).
editorHasSelection	Text is selected in the editor.
editorHasMultipleSelections	Multiple regions of text are selected (multiple cursors).
editorReadOnly	The editor is read only.
editorLangId	True when the editor's associated language Id matches. Example: <code>"editorLangId == typescript"</code> .
textCompareEditorVisible	Diff (compare) view is visible.
Mode contexts	
inDebugMode	A debug session is running.
inSnippetMode	The editor is in snippet mode.
inQuickOpen	The Quick Open drop-down has focus.
Explorer contexts	
explorerViewletVisible	True if Explorer view is visible.
explorerViewletFocus	True if Explorer view has keyboard focus.
filesExplorerFocus	True if File Explorer section has keyboard focus.
openEditorsFocus	True if OPEN EDITORS section has keyboard focus.
explorerResourcesFolder	True if a folder is selected in the Explorer.
Editor widget contexts	
findWidgetVisible	Editor Find widget is visible.
suggestWidgetVisible	Suggestion widget (IntelliSense) is visible.
suggestWidgetMultipleSuggestions	Multiple suggestions are displayed.
renameInputVisible	Rename input text box is visible.

referenceSearchVisible	Find All References peek window is open.
inReferenceSearchEditor	The Find All References peek window editor has focus.
config.editor.stablePeek	Keep peek editors open (controlled by <code>editor.stablePeek</code> setting).
quickFixWidgetVisible	Quick Fix widget is visible.
parameterHintsVisible	Parameter hints are visible (controlled by <code>editor.parameterHints</code> setting).
parameterHintsMultipleSignatures	Multiple parameter hints are displayed.
Integrated terminal contexts	
terminalFocus	An integrated terminal has focus.
Global UI contexts	
resourceLangId	True when the Explorer or editor title <code>language Id</code> matches. Example: <code>"resourceLangId == markdown"</code>
resourceFilename	True when the Explorer or editor filename matches. Example: <code>"resourceFilename == gulpfile.js"</code>
globalMessageVisible	Message box is visible at the top of VS Code.
searchViewletVisible	Search view is open.
sidebarVisible	Side Bar is displayed.
replaceActive	Search view Replace text box is open.
Configuration settings contexts	
config.editor.minimap.enabled	True when the setting <code>editor.minimap.enabled</code> is <code>true</code> .

Note: You can use any user or workspace setting that evaluates to a boolean here with the prefix `"config."`.

The list above isn't exhaustive and you may see some `when` contexts for specific VS Code UI in the **Default Keyboard Shortcuts**.

Removing a specific key binding rule

You can write a key binding rule that targets the removal of a specific default key binding. With the `keybindings.json`, it was always possible to redefine all the key bindings of VS Code, but it can be very difficult to make a small tweak, especially around overloaded keys,

such as `kbstyle(Tab)` or `kbstyle(Escape)`. To remove a specific key binding, add a `-` to the `command` and the rule will be a removal rule.

Here is an example:

```
// In Default Keyboard Shortcuts
...
{
  "key": "tab", "command": "tab", "when": ... },
{
  "key": "tab", "command": "jumpToNextSnippetPlaceholder", "when": ... },
{
  "key": "tab", "command": "acceptSelectedSuggestion", "when": ... },
...

// To remove the second rule, for example, add in keybindings.json:
{ "key": "tab", "command": "-jumpToNextSnippetPlaceholder" }
```

Keyboard layouts

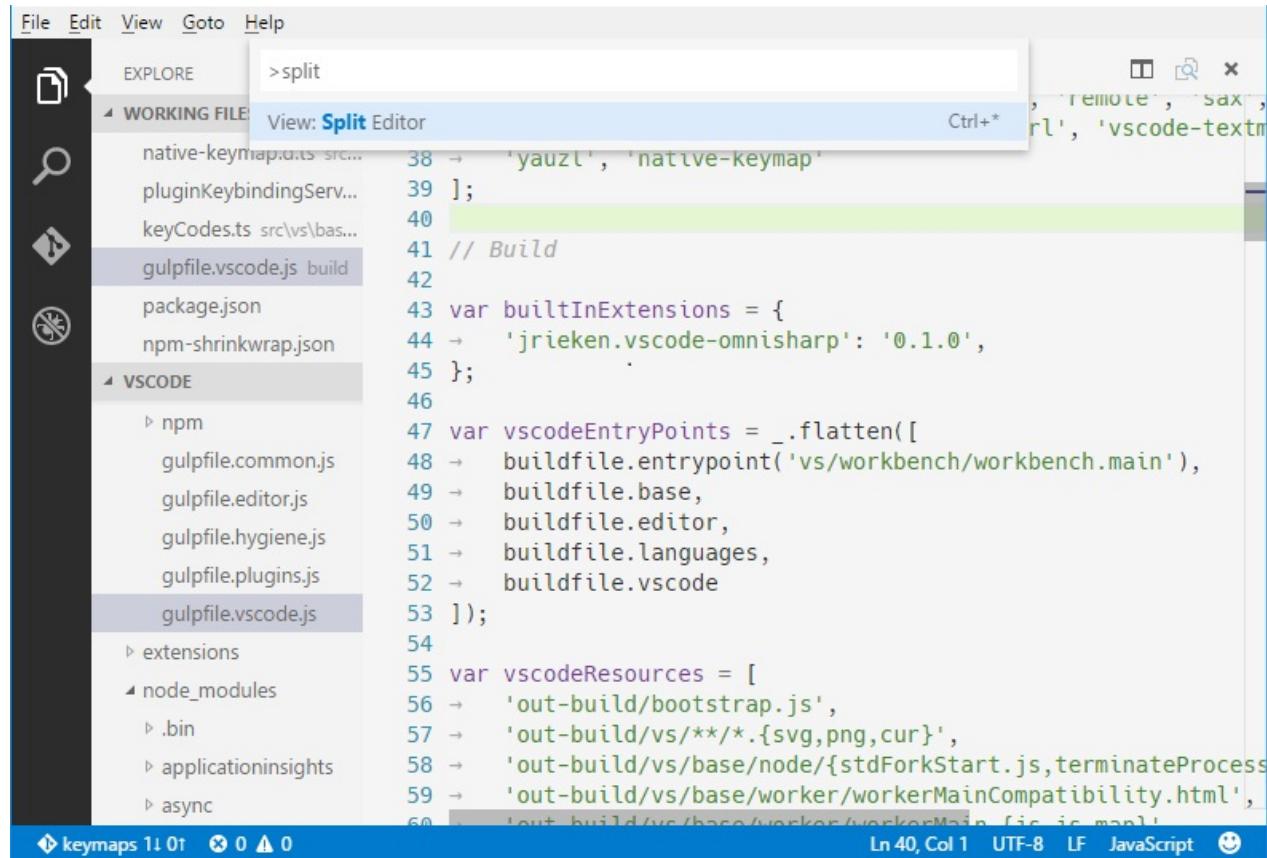
Note: This section relates only to key bindings, not to typing in the editor.

The keys above are string representations for virtual keys and do not necessarily relate to the produced character when they are pressed. More precisely:

- Reference: [Virtual-Key Codes \(Windows\)](#)
- `kbstyle(tab)` for `VK_TAB (0x09)`
- `kbstyle(;)` for `VK_OEM_1 (0xBA)`
- `kbstyle(=)` for `VK_OEM_PLUS (0xBB)`
- `kbstyle(,)` for `VK_OEM_COMMA (0xBC)`
- `kbstyle(-)` for `VK_OEM_MINUS (0xBD)`
- `kbstyle(.)` for `VK_OEM_PERIOD (0xBE)`
- `kbstyle(/)` for `VK_OEM_2 (0xBF)`
- `kbstyle(`)` for `VK_OEM_3 (0xC0)`
- `kbstyle([)` for `VK_OEM_4 (0xDB)`
- `kbstyle(\`)` for `VK_OEM_5 (0xDC)`
- `kbstyle([])` for `VK_OEM_6 (0xDD)`
- `kbstyle(')` for `VK_OEM_7 (0xDE)`
- etc.

Different keyboard layouts usually reposition the above virtual keys or change the characters produced when they are pressed. When using a different keyboard layout than the standard US, Visual Studio Code does the following:

All the key bindings are rendered in the UI using the current system's keyboard layout. For example, `Split Editor` when using a French (France) keyboard layout is now rendered as `kbstyle(Ctrl+*)`:



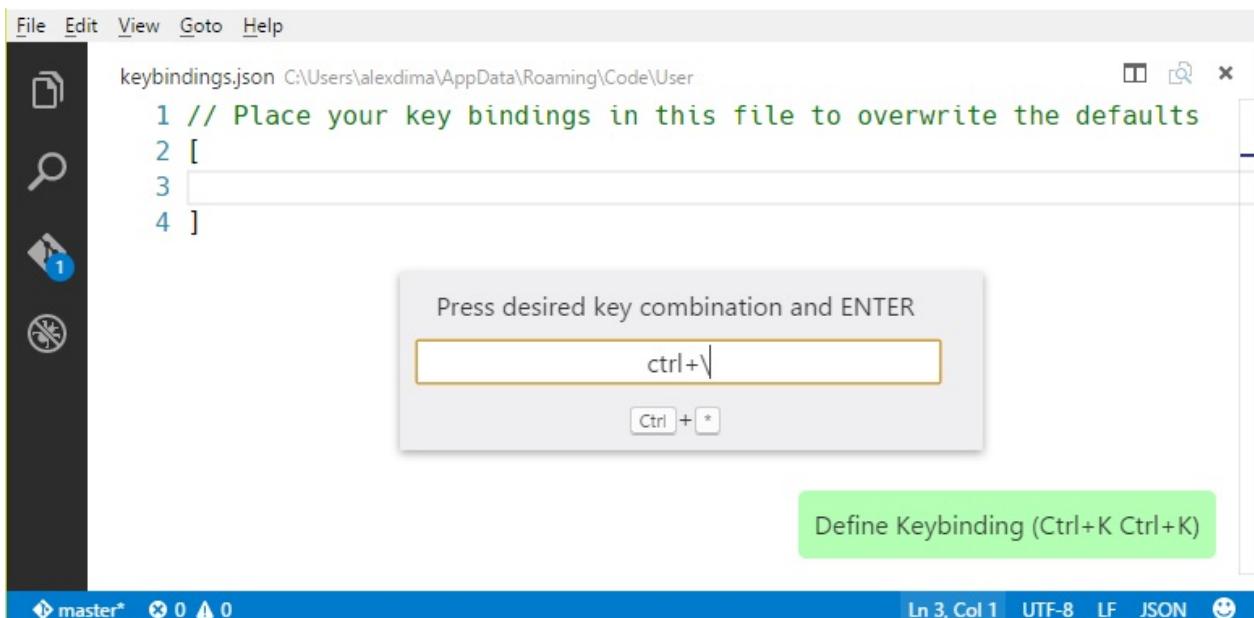
When editing `keybindings.json`, VS Code highlights misleading key bindings - those that are represented in the file with the character produced under the standard US keyboard layout, but which need pressing keys with different labels under the current system's keyboard layout. For example, here is how the **Default Keyboard Shortcuts** rules look like when using a French (France) keyboard layout:

```

300 { "key": "ctrl+shift+j",           "command": "workbench.action.search.toggleQueryDetails",
301 ..... "when": "searchViewletVisible" },
302 { "key": "ctrl+t",                 "command": "workbench.action.showAllSymbols" },
303 { "key": "f1",                     "command": "workbench.action.showCommands" },
304 { "key": "c",                      "command": "workbench.action.showCommands" },
305 { "key": "C",                      "command": "workbench.action.showErrorsWarnings" },
306 { "key": "ctrl+\\"},               "command": "workbench.action.splitEditor" },
307 { "key": "ctrl+shift+b",           "command": "workbench.action.tasks.build" },
308 { "key": "ctrl+shift+t",           "command": "workbench.action.tasks.test" },
309 { "key": "ctrl+shift+c",           "command": "workbench.action.terminal.openNativeConsole" },
310 { "key": "f11",                   "command": "workbench.action.toggleFullScreen" },
311 { "key": "ctrl+b",                "command": "workbench.action.toggleSidebarVisibility" },
312 { "key": "ctrl+=",                "command": "workbench.action.zoomIn" },
313 { "key": "ctrl+-",                "command": "workbench.action.zoomOut" },
314 { "key": "ctrl+k enter",          "command": "workbench.files.action.addToWorkingFiles" },
315 { "key": "ctrl+k ctrl+w",          "command": "workbench.files.action.closeAllFiles" },

```

There is also a widget that helps input the key binding rule when editing `keybindings.json`. To launch the **Define Keybinding** widget, press `kb(editor.action.defineKeybinding)`. The widget listens for key presses and renders the serialized JSON representation in the text box and below it, the keys that VS Code has detected under your current keyboard layout. Once you've typed the key combination you want, you can press `kbstyle(Enter)` and a rule snippet will be inserted.



Note: Visual Studio Code detects your current keyboard layout on start-up and then caches this information. For a good experience, we recommend restarting VS Code if you change your keyboard layout.

Next Steps

Now that you know about our Key binding support, what's next...

- [Language Support](#) - Our Good, Better, Best language grid to see what you can expect
- [Debugging](#) - This is where VS Code really shines
- [Node.js](#) - End to end Node.js scenario with a sample app

Common Questions

Q: How to find out what command is bound to a specific key?

A: In the Default Keyboard Shortcuts, open `Quick Outline` by pressing `kb(workbench.action.gotoSymbol)`

Shortcuts	@ctrl+shift+
"ctrl+end",	↳ ctrl+shift+c
"ctrl+shift+end",	↳ ctrl+shift+d
"down",	↳ ctrl+shift+down
"ctrl+shift+down",	↳ ctrl+shift+e
"shift+down",	↳ ctrl+shift+e
"end",	↳ ctrl+shift+end
"shift+end",	↳ ctrl+shift+enter
	↳ ctrl+shift+f
	↳ ctrl+shift+f11
	↳ ctrl+shift+f12
	when : editorTextFocus
	"command": "cursorLineDown"
	"command": "cursorLineUp"
	"command": "cursorPageDown"
	"command": "cursorPageUp"

Q: How to add a key binding to an action? For example add Ctrl+D to Delete Lines

A: Find a rule that triggers the action in the **Default Keyboard Shortcuts** and write a modified version of it in your `keybindings.json` file:

```
// Original, in Default Keyboard Shortcuts
{ "key": "ctrl+shift+k", "command": "editor.action.deleteLines",
  "when": "editorTextFocus" },
// Modified, in User/keybindings.json, Ctrl+D now will also trigger this action
{ "key": "ctrl+d", "command": "editor.action.deleteLines",
  "when": "editorTextFocus" },
```

Q: How can I add a key binding for only certain file types?

A: Use the `editorLangId` context key in your `when` clause:

```
{ "key": "shift+alt+a", "command": "editor.action.blockComment",
  "when": "editorTextFocus && editorLangId == cshtml" },
```

Q: I have modified my key bindings in `keybindings.json`, why don't they work?

A: The most common problem is a syntax error in the file. Otherwise, try removing the `when` clause or picking a different `key`. Unfortunately, at this point, it is a trial and error process.

Display Language

Visual Studio Code ships with 10 available display languages (locales): English (US), Simplified Chinese, Traditional Chinese, French, German, Italian, Japanese, Korean, Russian and Spanish. Localized display text for all 10 languages is included in the main VS Code download and as such, doesn't require a secondary install.

By default, VS Code picks up the operating system's display language, falling back to English (US) if the locale is not supported.

Available Locales

Display Language	Locale
English (US)	en
Simplified Chinese	zh-CN
Traditional Chinese	zh-TW
French	fr
German	de
Italian	it
Japanese	ja
Korean	ko
Russian	ru
Spanish	es

Setting the Language

If you want to configure a specific language, you can either use the command line switch `--locale` to specify a locale when you launch a VS Code session or use the [Configure Language](#) command to persist the display language to use when VS Code is started.

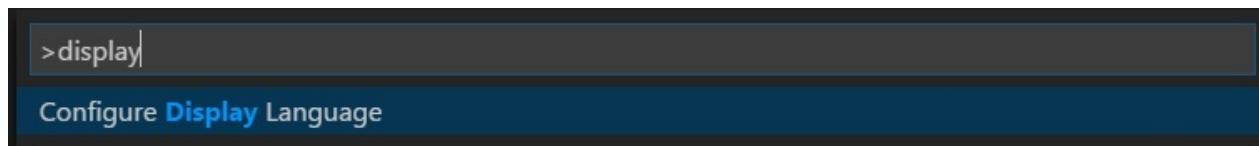
Below is an example of using the `--locale` command line switch to set the VS Code display language to French:

```
code . --locale=fr
```

Configure Language command

The **Configure Language** command creates a `locale.json` file in your user VS Code folder. Set the `locale` attribute to your preferred locale.

Press `kb(workbench.action.showCommands)` to bring up the **Command Palette** then start typing "config" to filter and display the **Configure Language** command.



Press `kbstyle(Enter)` and a `locale.json` file is created with the default value set to your operating system language. You can use IntelliSense (`kb(editor.action.triggerSuggest)`) to select a different supported language locale.

```
locale.json C:\Users\gregvanl\AppData\Roaming\Code - Alpha\User
1  {
2      // Defines VS Code's display language.
3      // See https://go.microsoft.com/fwlink/?LinkId=761051 for a list of supported languages.
4      // Changing the value requires to restart VS Code.
5      "locale": "en"
6 }
```

The code editor shows the `locale.json` file with the following content:

```
{
  // Defines VS Code's display language.
  // See https://go.microsoft.com/fwlink/?LinkId=761051 for a list of supported languages.
  // Changing the value requires to restart VS Code.
  "locale": "en"
}
```

An IntelliSense dropdown menu is open over the `"locale": "en"` line, listing various language codes: "de", "en", "en-US", "es", "fr", "it", "ja", "ko", "ru", "zh-CN", and "zh-tw". The item "en" is currently selected.

Save `locale.json` and restart VS Code to use the new display language.

The example below sets VS Code to display Simplified Chinese `zh-CN`:

```
{
  // Defines VS Code's display language.
  "locale": "zh-CN"
}
```

You can rerun the **Configure Language** command to review and change your `locale.json` file.

Note: Changing the `locale` value requires a restart of VS Code.

编辑器

- 基础
- 安装
- 扩展市场
- 任务
- 调试
- 为什么选用VSCode
- 版本控制
- 易用性
- 与时俱进的编辑体验

Visual Studio Code 基础

从骨子里来说，Visual Studio Code 是一款代码编辑器。跟其它的代码编辑器一样，VS Code 吸取了通用的用户接口和布局：左侧是用于展示所要编辑的所有文件和文件夹的文件管理器，右侧是打开文件的编辑器域。

另外，在VS Code当中还有一些与众不同的特性。在这个主题内将重点描述这些特性。

文件（File），文件夹（Folders） & 工程（Projects）

VS Code 是基于文件和文件夹的 - 可以在 VS Code 中通过打开文件和文件夹立即开始使用。

在这上面，VS Code 可以读取并自用一系列的不同框架和平台定义的工程文件。例如，如果你在VS Code中打开的文件夹中包含一个或多个'package.json'，'project.json'，'tsconfig.json' 亦或者 ASP.net core Visual Studio 解决方案和工程文件，VS Code 将读取这些文件并且利用它们来提供额外的功能，如在编辑器中的丰富的智能感知（IntelliSense）功能。

基本布局

VS Code 给我们带来一种简单直观的外观布局，这种布局可以给编辑器提供最大的空间同时又给浏览访问目录或工程上下文提供了充足的空间。整体UI被分成了四个区域：

- **Editor** 用来编辑文件的主体区域。可以并排打开三个编辑器。
- **Side Bar** 包含不同的像浏览器一样的视图来协助来完成工程。
- **Status Bar** 展示当前打开的工程和正在编辑的文件的信息。
- **View Bar** 在最左手边，帮助切换视图以及提供额外的上下文相关的提示，比如激活了Git的情况下，需要提交的变化的数目。

每次打开VS Code的时候，都会回到上一次关闭时的状态。文件夹，布局以及打开的文件都会保存。

```

logger.ts - HotTowel-Angular-TypeScript - Visual Studio Code
File Edit View Goto Help
EXPLORE WORKING FILES 2 UNSAVED
config.ts \app
shell.ts \app\layout
dashboard.ts \app\dashboard
common.ts \app\common
logger.ts \app\common
History.md C:\GitDevelopment\express
HOTTOWEL-ANGULAR-TYPESCRIPT
common.js
common.ts
logger.js
logger.ts
spinner.js
spinner.ts
dashboard
dashboard.js
dashboard.ts
layout
shell.js
shell.ts
sidebar.js
sidebar.ts
services
app.js
app.ts
master* 38 ▲ 2

```

```

logger.ts \app\common
1 interface loggerFunction {
2   (message: string, data: Object, source: string,
3 }
4
5 interface logger {
6   [fnName: string]: any;
7   getLogFn(moduleId: string, fnName?: string): (m
8   log: loggerFunction;
9   logError: loggerFunction;
10  logSuccess: loggerFunction;
11  logWarning: loggerFunction;
12 }
13
14
15
16 (function () {
17   'use strict';
18
19   angular.module('common').factory('logger', ['$l
20
21   function logger($log: ng.ILogService) {
22     var service: logger = {
23       getLogFn: getLogFn,
24       log: log,
25       logError: logError,
26       logSuccess: logSuccess,
27       logWarning: logWarning
28     };
29
30     return service;
31   };

```

```

dashboard.ts
1 interface dashboardVm {
2   messageCount: number;
3   news: {
4     title: string;
5     description: string;
6   }
7   people: person[];
8   title: string;
9 }
10
11 (function () {
12   'use strict';
13   var controllerId = 'dashboard';
14   angular.module('app').controller(controllerId,
15
16   function dashboard(common: common, datacontext
17   var getLogFn = common.logger.getLogFn;
18   var log = getLogFn(controllerId);
19
20   var vm: dashboardVm = this;
21   vm.news = {
22     title: 'Hot Towel Angular',
23     description: 'Hot Towel Angular is a S
24   };
25   vm.messageCount = 0;
26   vm.people = [];
27   vm.title = 'Dashboard';
28
29   activate();
30
31   function activate() {

```

In 15, Col 1 CRLF TypeScript

除了将文件放在独立的标签页中，VS Code允许一次性打开三个编辑器，可以将最多三个文件并排打开。

这减少了你切换标签页的时间开销，并且不会限制你同时操作的文件数。文件管理器视图含有一个你可以快速打开的文件列表。

小技巧：你可以将侧边栏移动到右手边(**View > Move Sidebar**) 或者将其隐藏
(`kb(workbench.action.toggleSidebarVisibility)`)。

并列编辑

可以最多并排打开三个编辑器。

如果你已经打开一个编辑器，有多种方式可以在旁边打开另一个编辑器：

- 按住 `kbstyle(Ctrl)` (`Mac: kbstyle('Cmd')`) 单击文件管理器中的文件
- 使用 `kb(workbench.action.splitEditor)` 命令来将编辑器拆分为两块
- 在文件管理器中文件右键菜单中单击**Open to the Side**

The screenshot shows the Visual Studio Code interface with three open files:

- logger.ts** (Left): A TypeScript file containing interface definitions for logger and loggerFunction.
- dashboard.ts** (Middle): A TypeScript file for a dashboard component, showing code for logging and a controller.
- History.md** (Right): A Markdown file listing commit history for a project, starting from version 2.5.11.

任何时候你打开另一个文件，将会显示在当前激活的编辑器中，所以如果你有两个并列编辑器，而你想在右边的编辑器中打开文件"foo.cs"，则打开文件之前必须确保右边的编辑器激活（通过单击编辑器）。

当你打开多个编辑器时，你可以通过按住 `kbstyle(Ctrl)` (`Mac: kbstyle('Cmd')`) 键并按下 `kbstyle(1)`，`kbstyle(2)` 或 `kbstyle(3)` 键来切换编辑器。

小技巧： 你可以通过在编辑器标题区拖拽来对编辑器进行排序和调整大小。

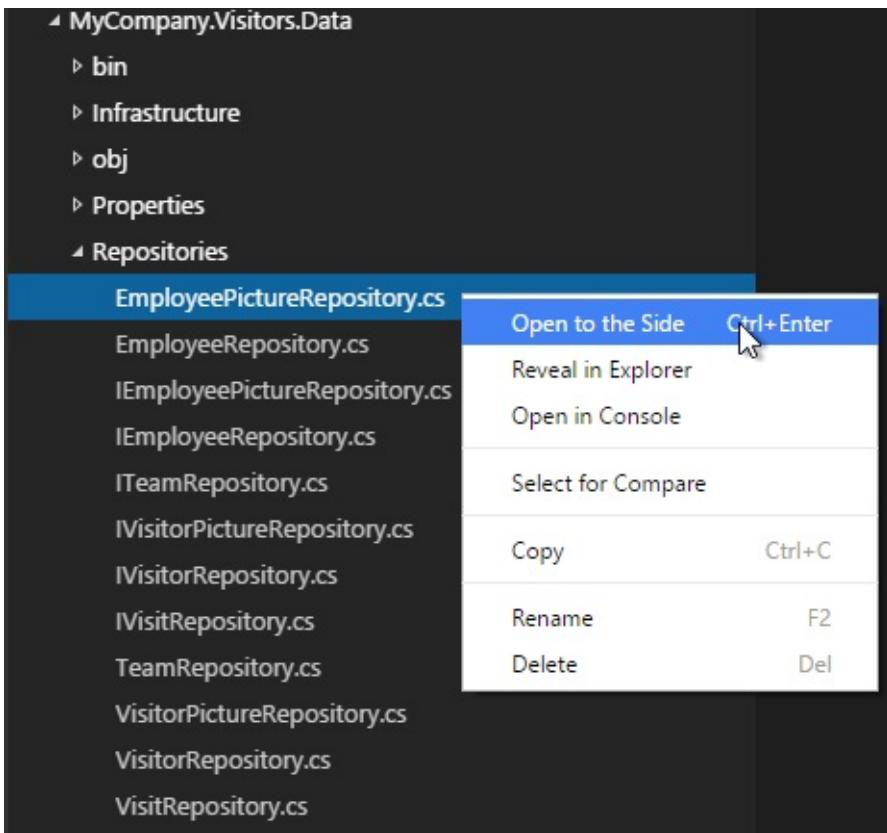
文件管理器 (Explorer)

文件管理器用来浏览、打开和管理项目内的所有文件和文件夹。

在VS Code中打开文件夹后，文件夹内的内容会显示在文件管理器中，可在此做如下操作：

- 创建、删除、重命名文件和文件夹
- 通过拖拽移动文件和文件夹
- 使用右键菜单操作文件

小技巧： 你可以从VS Code之外拖拽文件到文件管理器来拷贝文件到当前文件夹。



VS Code可以与你是用的其他工具协同工作，特别是命令行工具。如果你想要在当前VS Code中打开文件夹的右键菜单中打开命令行工具，在文件夹上右键选择**Open in Command Prompt** (在OS X或者Linux上选择 **Open in Terminal**)。

也可以在本地资源管理器中找到文件或文件夹，右键并选择**Reveal in Explorer** (在Mac上选择 **Reveal in Finder**或者在Linux上**Open Containing Folder**)。

小技巧：输入命令 `kb(workbench.action.quickOpen)` (**Quick Open**) 来根据名称快速搜索并打开文件。

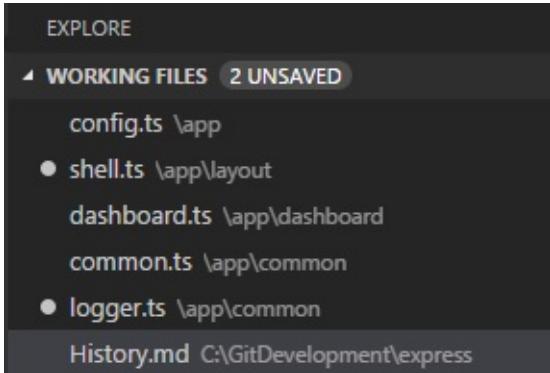
默认情况下，VS Code文件管理器中包括所有文件夹(如 `.git`)，我们可以通过 `files.exclude` `setting`文件配置不显示文件和文件夹的规则。

小技巧：这用于隐藏生成的文件非常有用，如Unity的 `*.meta` 文件、TypeScript项目的 `*.js` 文件。对于Unity，不显示 `*.cs.meta` 文件的选择模式应该是：`"**/*.cs.meta": true`，对于TypeScript，不显示TypeScript文件生成的JavaScript文件的选择模式是：`"**/*.js": {"when": "$(basename).ts"}`。

打开的编辑器

在文件管理器顶部有一个标签是**WORKING FILES**的部分，这是当前激活文件的列表。这些文件是你之前操作时在VS Code中打开的文件，当做了如下操作时文件会被加入打开编辑器列表：

- 对文件做出更改
- 在文件管理器中双击文件
- 打开一个不在当前文件夹中的文件



可以把**WORKING FILES**列表当做其他编辑器或IDE中的标签页，只要在打开的文件列表中单击文件就可以在VS Code中激活文件，显示在编辑器中。

一旦你完成了任务，你可以单独从打开文件列表中移除文件，或者你可以使用**Close All Files**操作从打开文件列表中移除所有文件。

小技巧：你可以输入命令 `kb(workbench.files.action.workingFilesPicker)` 来从文件选择器中选择当前打开文件列表中的文件，而不用去打开文件管理资源器。

配置编辑器

VS Code有很多选项可以来配置编辑器，你可以通过用户设置来设置全局选项，或在工作空间设置中针对每个文件夹或项目设置，选项设置保存在 `settings.json` 文件中。

- 选择 **Files > Preferences > User Settings** (或者按下 `kb(workbench.action.showCommands)`，输入 `user` 然后按下 `Enter`) 来编辑 `settings.json` 文件.
- 选择 **Files > Preferences > Workspace Settings** (或者按下 `kb(workbench.action.showCommands)`，输入 `worksp` 然后按下 `Enter`) 来编辑工作空间的 `settings.json` 文件.

你会看到VS Code 默认配置在左侧窗口并且你编辑的 `settings.json` 在右侧。你可以只是从默认设置查看和拷贝设置。

在完成设置编辑之后，输入 `kb(workbench.action.files.save)` 来保存更改，更改会立即生效。

保存/自动保存

VS Code默认需要一个显示的操作来保存更改到硬盘上：

```
kb(workbench.action.files.save)。
```

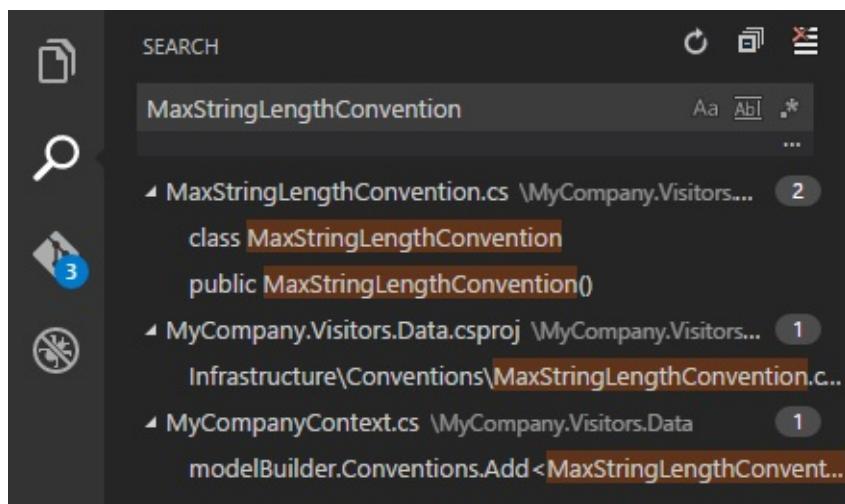
不过，`自动保存`功能会在配置的延迟时间后或焦点离开编辑器时，保存更改，打开了这个选项之后就没有必要显示保存文件了。

要想配置`自动保存`功能，打开**User Settings**或者**Workspace Settings**并找到相关设置：

- `files.autoSave`：自动保存值设置为`off`来禁用此功能，设置为`afterDelay`在配置的延迟时间后保存，设置为`onFocusChange`在鼠标离开已修改文件的编辑器时保存。
- `files.autoSaveDelay`：设置延迟时间，毫秒为单位，`files.autoSave`设置为`afterDelay`时有效。

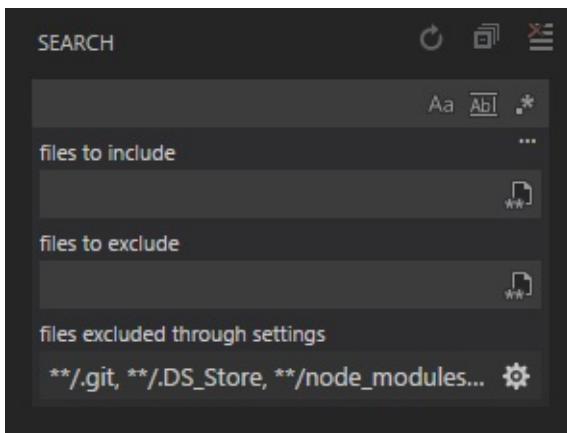
跨文件查找

VS Code允许你在当前打开文件夹中所有文件里面快速搜索，只要输入`kb(workbench.view.search)`然后输入要搜索的内容，搜索结果将会被按照文件分组显示，文件下显示搜索结果部分和在文件中显示的位置，展开一个文件可以预览该文件中所有的搜索结果，然后单击搜索结果便可以在编辑器中显示。



小技巧：在搜索框中搜索也支持正则表达式。

你可以通过输入`kb(workbench.action.search.toggleQueryDetails)`来配置搜索的高级配置，这样会在查询框显示额外的查询配置项。



在查询框下面的两个输入框，你可以选择包含和排除的文件。单击右侧图标来启用glob匹配模式语法：

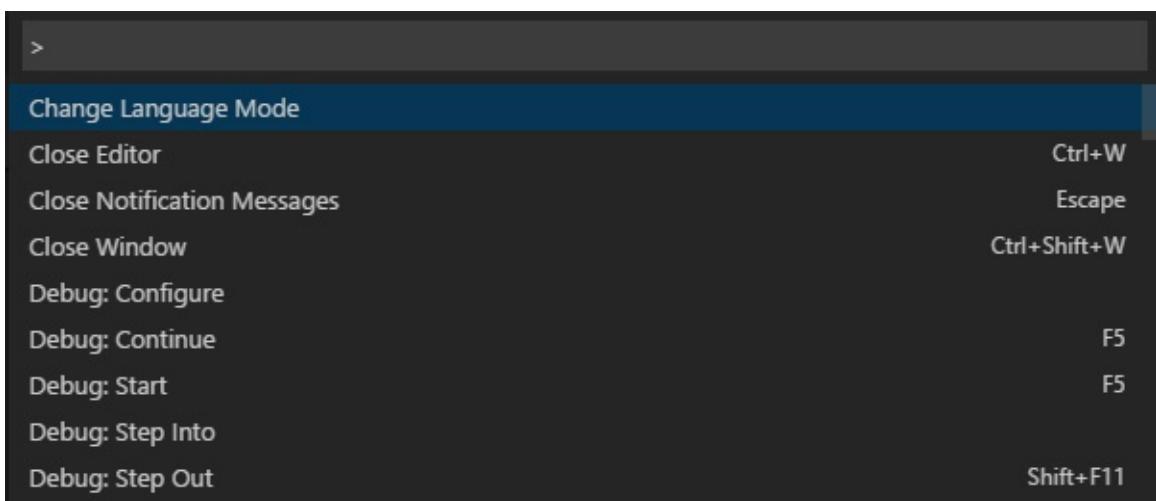
- `*` 匹配路径段中的一个或多个字符
- `?` 匹配路径段中的一个字符
- `**` 匹配任意数量的路径段，包括空
- `{}` 匹配分组条件（如：`{**/*.html, **/*.txt}` 匹配所有HTML和txt文件）
- `[]` 声明匹配一个范围内的所有字符（如：`example.[0-9]` 匹配 `example.0`, `example.1`, ...）

VS Code 默认会排除一些你不会感兴趣的文件夹（如：`node_modules`）来减少查询结果的数量，在设置文件中的 `files.exclude` 和 `search.exclude` 小节下可以修改这个规则。

小技巧：可以在文件管理器中右键一个文件夹并选择 **Find in Folder** 来仅在选择文件夹下搜索。

命令面板（Command Palette）

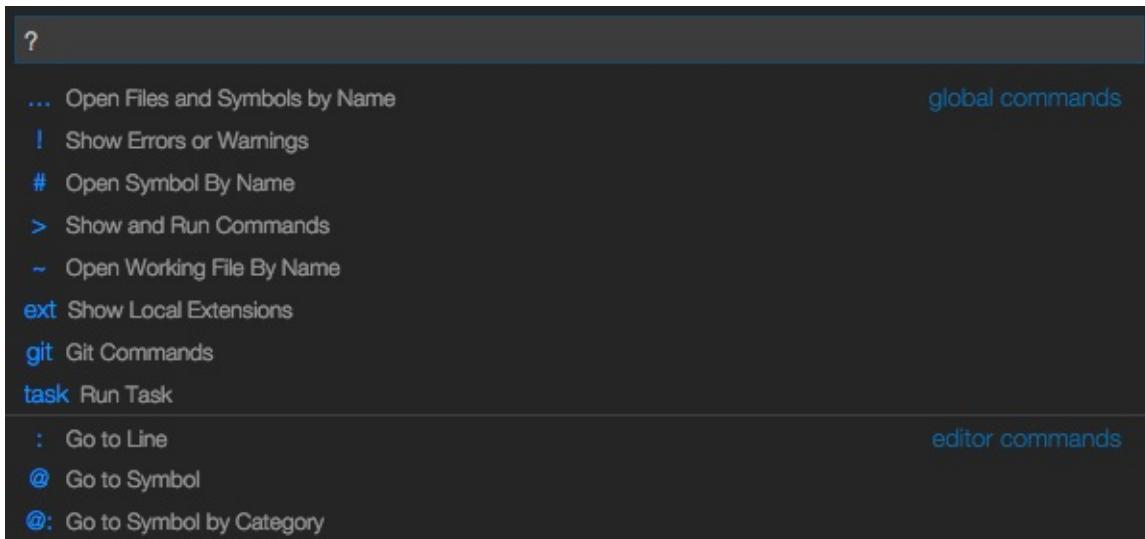
VS Code 可以同样简单的使用键盘操作，需要了解的最重要的组合键便是 `kb(workbench.action.showCommands)`，它可以显示出命令面板，在这里可以获取到 VS Code 的所有功能，包括最常用的键盘快捷键。



命令面板提供了许多的命令，你可以使用同一交互窗口执行编辑器命令、打开文件、查询字符或者查看文件大纲，如下一些常用操作：

- `kb(workbench.action.quickOpen)` 通过键入文件或符号来导航至相应的位置
- `kb(workbench.action.openPreviousEditor)` 在最近打开文件列表之间切换
- `kb(workbench.action.showCommands)` 直接跳转到编辑器命令面板
- `kb(workbench.action.gotoSymbol)` 导航至文件中的某一指定符号处
- `kb(workbench.action.gotoLine)` 导航至文件中的某一指定行

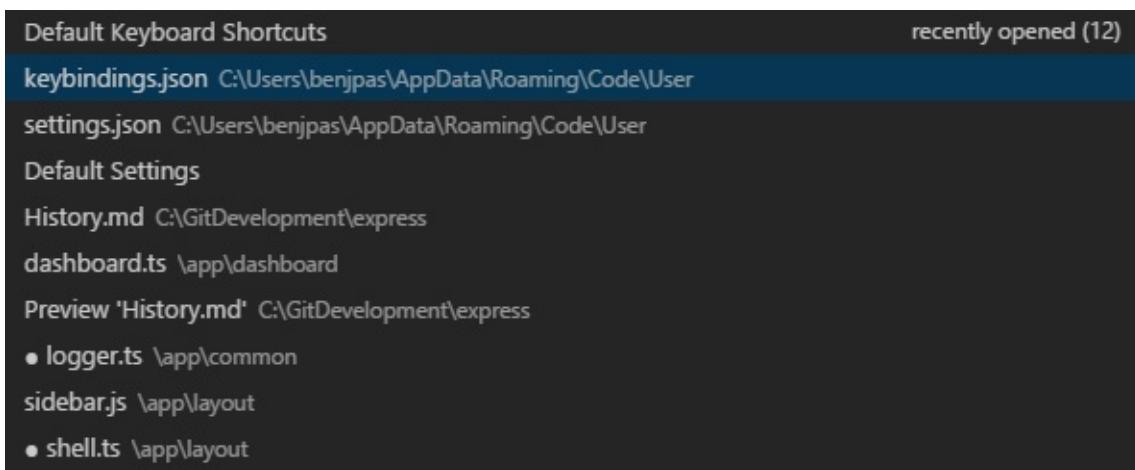
在输入框中输入 `?` 获取一个可以执行命令的列表：



快速文件导航

文件管理器很适合在项目的各个文件之间来回导航，在你进行工作的时候，可以在同一文件集下文件之间快速切换。VS Code有两个命令用来文件键导航，并绑定了相应快捷键。

按下 `kbstyle(Ctrl)` 并按 `kbstyle(Tab)` 来查看一个VS Code启动之后打开的文件列表，要打开其中一个文件，继续按 `kbstyle(Tab)` 来选择想要查看的文件，最后松开 `kbstyle(Ctrl)` 来打开文件。



另外，可以使用 `kb(workbench.action.navigateBack)` 和 `kb(workbench.action.navigateForward)` 在文件和编辑位置之间导航，如果你在同一文件中的不同行之间进行跳转，这两个快捷键可以让你在这些行之间导航。

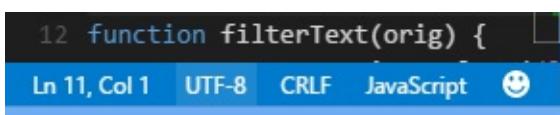
小技巧: 在使用 `kb(workbench.action.quickOpen)` (**Quick Open**) 时你可以通过名称打开任何文件。

文件编码支持

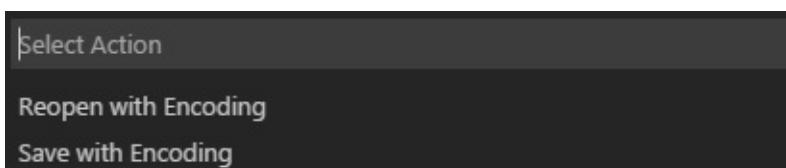
通过 **User Settings or Workspace Settings** 中的 `files.encoding` 设置项来配置全局文件编码或每个工作空间的文件编码。

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

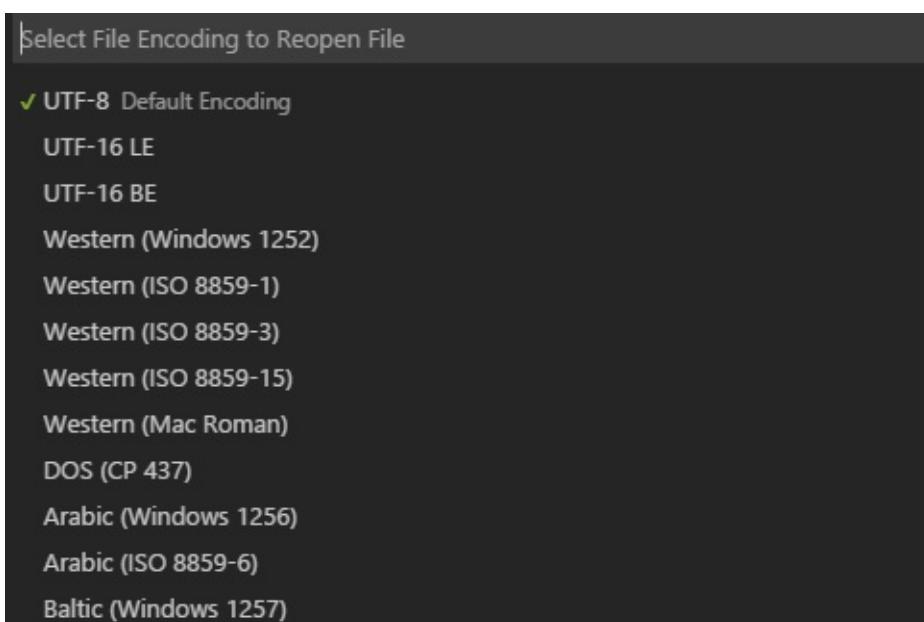
你可以在状态栏中查看文件的编码格式。



单击状态栏中的文件编码可以选择使用另一种编码重新打开或保存当前文件。



然后选择一种编码格式。



从命令行中启动

你可以从命令行中使用VS Code快速打开一个文件、文件夹或工程，通常，你想要一个文件夹上下文中打开VS Code的最简单方法就是在想要打开的文件夹下输入：

```
code .
```

小技巧：对于Mac用户我们在 [Setup](#) 小节中有相关的主题指导你在终端中启动VS Code。在Windows和Linux下VS Code可执行文件会自动添加到 PATH 环境变量中。

有时你想打开或创建文件，如果文件不存在的话，VS Code会为你创建：

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

小技巧：可以通过空格来区分多个文件名。

额外的命令行参数

在命令行中通过'code'来启动VS Code的时候有一些额外的命令行参数：

参数	描述
<code>-h</code> or <code>--help</code>	打印说明
<code>-v</code> or <code>--version</code>	打印 VS Code 版本 (例如 0.10.10)
<code>-n</code> or <code>--new-window</code>	打开一个新的VS Code会话窗口而不是使用之前的会话窗口
<code>-r</code> or <code>--reuse-window</code>	强制在上一个激活窗口中打开文件或文件夹
<code>-g</code> or <code>--goto</code>	当使用 <code>file:line:column?</code> 打开文件到指定行指定列时，这个参数用于允许文件名中含有 <code>:</code> 符号的操作系统来指定行列数
<code>file</code>	想要打开文件的名称，如果文件不存在，则会被创建并标记为编辑，你可以指定多个文件，文件名之间用空格隔开
<code>file:line:column?</code>	打开文件并跳转到指定行，也可选指定列，你可以通过这种方式打开多个文件，不过这时就必须要在 <code>file:line:column?</code> 参数前加 <code>-g</code> 参数
<code>folder</code>	想要打开的文件夹名，可以指定多个文件夹
<code>-d</code> or <code>--diff</code>	打开对比编辑器，需要两个文件路径作为参数
<code>--locale</code>	设置VS Code会话窗口显示的本地化语言，支持的本地化语言包括： <code>en-US</code> , <code>zh-TW</code> , <code>zh-CN</code> , <code>fr</code> , <code>de</code> , <code>it</code> , <code>ja</code> , <code>ko</code> , <code>ru</code> , <code>es</code>
<code>--disable-extensions</code>	禁用所有安装的扩展，扩展仍然可以在 <code>Extensions: Show Installed Extensions</code> 下拉列表中显示，但是不能被启用
<code>-w</code> or <code>--wait</code>	恢复前等待窗口被关闭

不管是文件还是文件夹，都可以是用绝对或相对路径，当运行 `code` 时，相对路径时相对于命令提示符的当前目录。

如果你在命令行中指定多个文件或文件夹，VS Code只会打开一个会话窗口。

打开工程

VS Code并不区分打开文件夹和打开工程，如果VS Code检测到你打开的文件是一个工程（如：`C#工程`），工程上下文会显示在状态栏中，如果找到了多个工程，你也可以在这里切换工程。

要想打开 `c:\src\WebApp` 文件夹下包含的工程，应该以如下方式启动VS Code：

```
code C:\src\webapp
```

在VS Code打开之后，打开源文件并使用状态栏切换激活工程即可。



管口管理

VS Code有些设置是用来控制窗口如何在会话之间打开和恢复的。

`window.openFilesInNewWindow` 设置控制文件是在新窗口中打开还是使用已经打开的VS Code窗口。默认情况下，在你在VS Code之外双击打开文件或在命令行中打开文件时，VS Code会打开一个新的窗口，设置这个值为 `false` 来在最近使用的VS Code窗口中打开文件。

`window.reopenFolders` 设置VS Code如何恢复之前会话打开的窗口，默认情况下，VS Code会重新打开上一个会话的最后一个工作目录（设置值：`one`），若设置为 `none` 则不重新打开任何文件夹并始终打开一个空的VS Code窗口，若设置为 `all` 则会打开上一次会话中使用的所有工作目录。

下一步

你已经了解了基本的用户界面，当然VS Code还有更多需要了解的，参见如下内容：

- [User/Workspace Settings](#) - 学习如何在用户设置和工作空间设置中配置VS Code首选项
- [Editing Evolved](#) - 代码分析，智能提醒，，Lightbulbs， 定义预览和转到定义等
- [Debugging](#) - 这就是VS Code的亮点所在
- [Customization](#) - 主题、设置和快捷键绑定

常见问题

Q: 是否可以全局搜索和替换？

A: 这个功能还未实现，但是未来一定会实现！

Q: 如何打开自动换行？

A: 可以通过 `editor.wrappingColumn` 设置 来控制自动换行。默认情况下，`editor.wrappingColumn` 设置为300个字符。你可以调整行宽度或设置这个值为0来根据编辑器视图宽度换行：

```
"editor.wrappingColumn": 0
```

你可以使用 `kb(editor.action.toggleWordWrap)` 来切换VS Code会话和自动换行功能，再次打开VS Code时会自动设置成上一次的 `editor.wrappingColumn` 值。

你还可以使用 `editor.rulers` 设置来设置编辑器的垂直列标尺，其值为一个数组，数组中每个值表示你想要显示垂直标尺的位置。

Q: 如何在“打开的编辑器”区域中显示更多的文件？

A: 你可以在你的 [设置](#) 中配置**WORKING FILES**的样式。比如，你可以通过 `explorer.workingFiles.maxVisible` 设置在滚动条出现前可以显示的最大文件数，或通过**WORKING FILES**是否动态调整高度。

安装 Visual Studio Code

安装与使用VS Code非常简单和快捷。各平台请按照如下指导安装即可。

VS Code轻量且可以运行在大多数可用硬件和平台版本。您可以重新查看[系统要求](#)以检查您的机器配置是否支持VS Code。

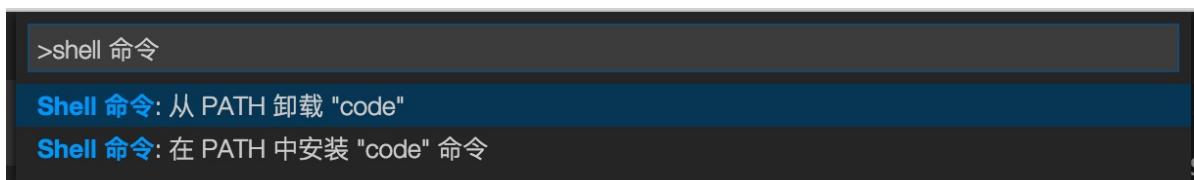
Note : VS Code是一款占用空间不大的编辑器。不同于传统IDEs几乎包含了所有组件，您可以为了您所关心的开发技术而调整您的安装。确保在平台指导后阅读[附加组件](#)部分学习如何客制化您的VS Code安装。

Mac OS X

1. 下载适用于Mac OS X的[Visual Studio Code](#)。
2. 双击下载的存档展开内容。
3. 拖动 `visual Studio Code.app` 至 `Applications` 文件夹，在 `Launchpad` 中点击打开。
4. 右键点击Dock栏中的VS Code图标，依次选择 `选项` 、`在Dock中保留`。

Tip : 如果您希望能够在终端中输入‘code’就能运行VS Code，VS Code有一条命令，**Shell Command: Install 'code' command in PATH**可以将‘code’添加至您的 `$PATH` 中。

安装完成之后，运行VS Code。现在，打开命令面板（按F1键）输入 `Shell 命令` 找到 `Shell 命令: 在PATH中安装“code”命令`。



命令执行完成之后，重启终端工具使新的 `$PATH` 可用。现在，您可以简单地在终端中任意文件夹下输入‘code’来编辑该文件夹下的文件了。

Linux

1. 下载适用于您发行版的Visual Studio Code，基于Debian的发行版如Ubuntu请点击[.deb](#)，基于Red Hat(`rpm`包管理格式，包括OpenSuSE)的发行版如Fedora或CentOS请点击[.rpm](#)。注意，32-bit二进制安装文件在[下载页面](#)同样可用。
2. 通过任一GUI软件包管理系统安装只需双击包文件即可，或者也可以使用命令行安装：

```

# For .deb
sudo dpkg -i <file>.deb

# For .rpm (Fedora 21 and below)
sudo yum install <file>.rpm

# For .rpm (Fedora 22 and above)
sudo dnf install <file>.rpm

```

3. 现在VS Code应该可以通过launcher或在命令行中输入 code 运行了。

Tip：在任意文件路径下输入‘code .’来编辑该文件夹下的文件。

Windows

1. 下载[Visual Studio Code](#)。
2. 双击 `vsCodeSetup.exe` 开始安装，这将会花费几分钟。
3. 64-bit构架下，VS Code默认将会安装在 `C:\Program Files (x86)\Microsoft VS Code` 下。

Note：VS Code要求.NET Framework 4.5已安装，如果您使用的是Windows 7系统，请确保.NET Framework 4.5已安装。

Tip：安装程序会选择添加Visual Studio Code至您的 `%PATH%` 中，于是您在控制台中任意文件路径下输入‘code .’来编辑该文件夹下的文件。

Tip：您可能需要在安装完成后注销让变化的 `%PATH%` 环境变量生效。

附加组件

VS Code的下载不大且只包含各种开发工作中共享的最小数量的组件，包括例如编辑器、文件管理、窗口管理和用户设置等基本功能。JavaScript/TypeScript语言服务和Node.js调试器也属于基本组件范畴。

如果您习惯于用更大、更整体的开发工具（IDEs），那么您可能会惊讶地发现您的情况不能完全支持开箱即用。例如，VS Code没有“文件 > 新建工程”这一选项及预先安装的工程模板。大多数VS Code使用者需要依据他们的特殊需求安装附加组件。

以下是几个常用的已安装组件：

- [Git](#) - VS Code内部支持了源代码版本控制系统Git，但是需要您电脑中预先安装好了Git
- [Node.js\(includes NPM\)](#) - 一个快速搭建运行JavaScript应用的平台和运行时
- [TypeScript](#) - TypeScript编译器 `tsc`
- [Typings](#) - 一个TypeScript类型定义管理程序，让VS Code可以为流行的JavaScript框架提

供智能感知

您将会发现上述组件经常在我们的文档和穿行测试中提及。

VS Code扩展

VS Code扩展让第三方提供额外的支持：

- 语言 - [C++, C#, Go, Python](#)
- 工具 - [ESLint, JSHint , PowerShell, Visual Studio Team Services](#)
- 调试器 - [Chrome, PHP XDebug.](#)

扩展融入了VS Code的用户界面、命令行和任务运行系统，所以您会发现通过VS Code的共享接口很容易使用不同的技术工作。点此查看[VS Code扩展市场](#)发现新东西。

附加工具

Visual Studio Code整合了现有工具链。我们认为下述工具可以提升您的开发体验。

- [Yeoman](#) - 一个应用搭建工具，想象成命令行版本的文件 > 新建工程
- [generator-aspnet](#) - 一个搭建**ASP.NET Core**应用的Yeoman生成器
- [hottowel](#) - 一个快速创建**AngularJS**应用的Yeoman生成器
- [Express](#) - 使用 **Jade** 模板引擎的Node.js应用框架
- [gulp](#) - 基于流的自动化构建工具，可以很容易地整合进VS Code任务
- [mocha](#) - 运行于Node.js之上的JavaScript测试框架
- [bower](#) - 客户端软件包管理器

下一步

现在您已经安装调试您的VS Code了。

更多文档，请查阅：

- [The Basics](#) - Basic orientation around VS Code
- [Editing Evolved](#) - Lint, IntelliSense, Lightbulbs, Peek and Goto Definition and more
- [Debugging](#) - This is where VS Code really shines

如果您想要快速开始，试试[Node.js runtime](#)，可以让您使用VS Code几分钟内调试一个Node.js web应用。

常见问题

Q: VS Code有什么系统要求？

A: 相关要求请查看[系统要求](#)。

Q: 如何创建并运行一个新项目？

A: VS Code并没有传统的“文件 > 新建工程”选项或预装工程模板，您需要依据您的开发需求添加附加组件和搭建工具。在项目搭建工具如[Yeoman](#)的帮助和[NPM](#)包管理器上众多的模块中，您肯定可以找到合适的模板和工具用来创建您的项目。

VS Code 扩展市场 VS Code Extension Marketplace

通过扩展，增强VS Code的能力 Increase the power of VS Code through Extensions

The features that VS Code includes out-of-the-box are just the start. VS Code extensions let you add new languages, features 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 topic explains how to find, install, and manage VS Code extensions.

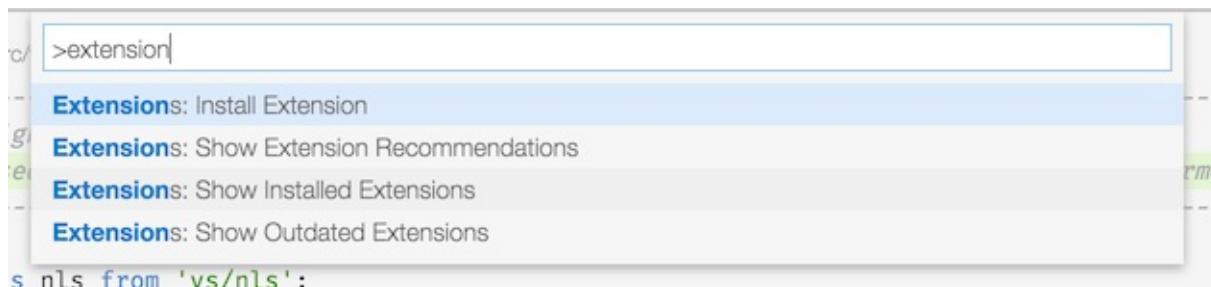
对于VS Code所包括的开箱即用的功能而言，仅仅是开始。通过安装VS Code扩展，让你添加新的语言，功能和工具，以支持你的开发工作流程。VS Code的丰富的扩展模型允许扩展的开发者可以直接插手到VS Code的UI，并通过使用相同的API，向VS Code贡献功能。本主题说明如何查找，安装和管理VS Code扩展。

在VS Code中浏览和安装扩展 Browse and Install Extensions in VS Code

You can browse and install extensions from within VS Code. Press

```
kb(workbench.action.showCommands) and narrow down the list commands by typing  
extension :
```

您可以在VS Code内浏览安装扩展程序。按下 `kb(workbench.action.showCommands)`，并通过键入 `extension` 来缩小命令列表的范围：

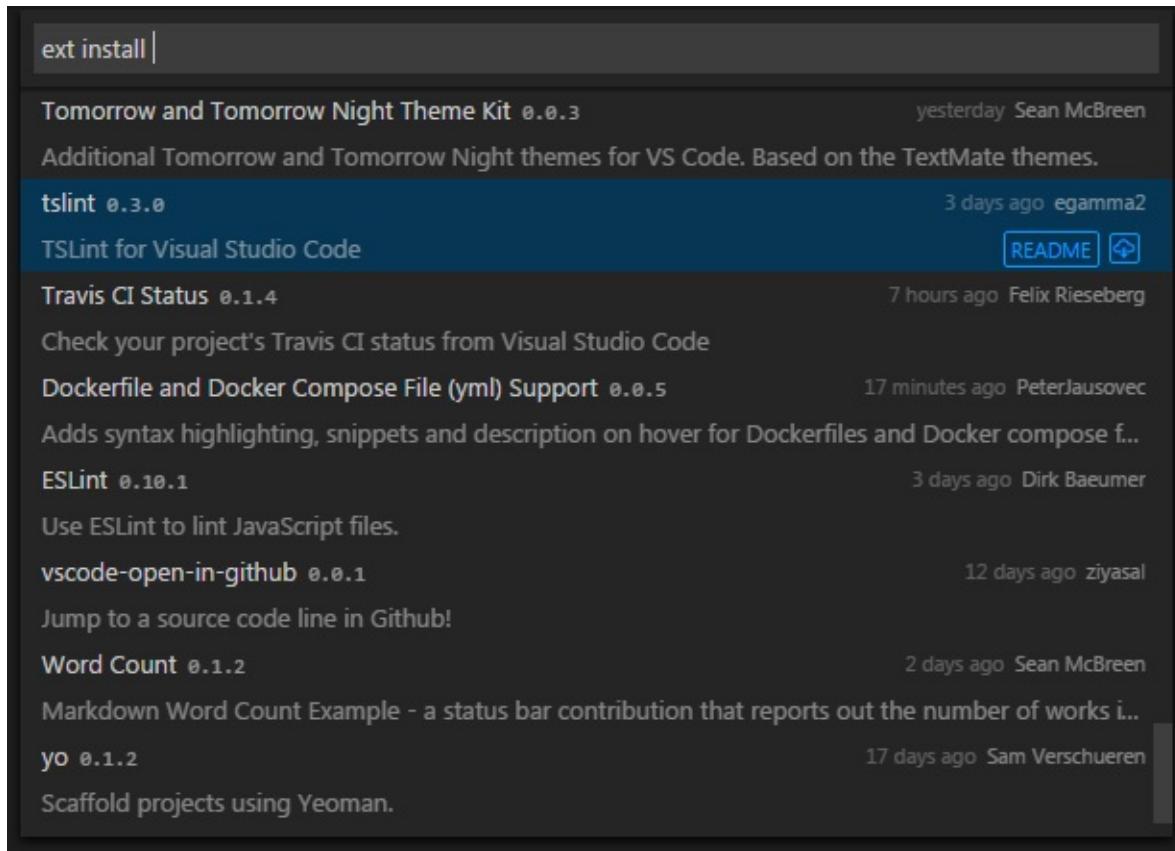


Pick `Extensions: Install Extension`.

选择 `Extensions: Install Extension`。

Tip: As an alternative, press `kb(workbench.action.quickOpen)` (**Quick Open**) and type `ext install` with a trailing space. Not sure what to install? Visit [VS Code Marketplace](#).

小技巧：作为一种替代，按下 `kb(workbench.action.quickOpen)` (**Quick Open**) and 输入后面有一个空格的 `ext install`。不知道要安装什么？访问 [VS Code 市场](#) 吧。



You'll see a list of extensions on the Marketplace along with the publisher, published date and a brief description. You can click the `README` button to go to the extension's [VS Code Marketplace](#) page where you can learn more.

在市场你会看到扩展的列表与出版商，发布日期和简要描述。您可以点击“README”按钮转到 [VS Code 市场](#) 页面，在那里可以学习更多扩展。

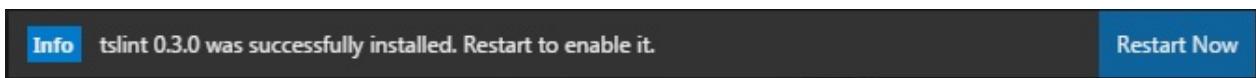
Note: If your computer's Internet access goes through a proxy server, you will need to configure the proxy server. See [Proxy Server Support](#) for details.

注意：如果你的计算机通过代理服务器上网，则需要配置代理服务器。详细信息，参看 [Proxy Server Support](#)。

安装扩展 Install an Extension

Simply pick the extension from the list. After a successful install, you'll get the following notification:

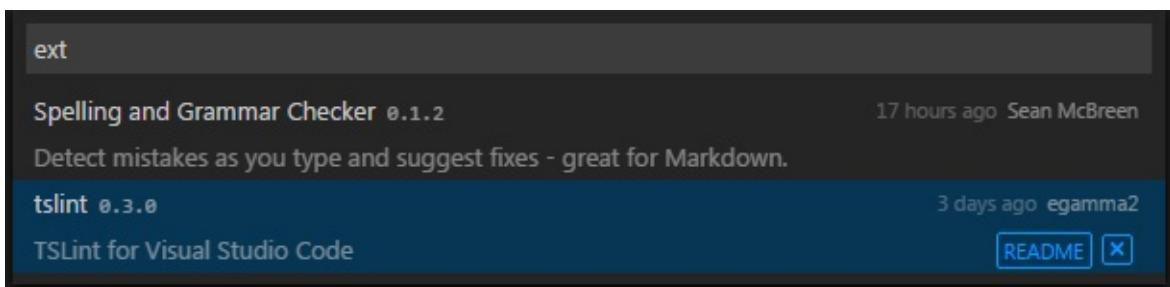
只需从扩展列表中选择。然后安装，成功后，你会得到以下通知：



列出已安装的扩展 List Installed Extensions

You can also browse installed extensions with the `Extensions: Show Installed Extensions` command or by typing `kb(workbench.action.quickopen)` (**Quick Open**) and `ext` with a trailing space.

您也可以浏览已安装的扩展，通过 `Extensions: Show Installed Extensions` 命令，或输入 `kb(workbench.action.quickopen)` (**Quick Open**) 和后面有一个空格的 `ext`



卸载扩展 Uninstall an Extension

To uninstall an extension, bring up the `Extensions: Show Installed Extensions` dropdown and click the `x` button in the lower right of the extension entry. This will uninstall the extension and prompt you to restart VS Code.

要卸载的延伸，把 `Extensions: Show Installed Extensions` 菜单下拉，然后单击扩展入口的右下角的 `x` 按钮。这将卸载扩展并提示您重新启动VSCode。

更新扩展 Update an Extension

You can quickly look for extension updates by using the `Extensions: Show Outdated Extensions` dropdown. This will display any available updates for your currently installed extensions. Simply click the `Update Extension` button in the lower right for the outdated extension and the update will be installed and you'll be prompted to restart VS Code.

通过 `Extensions: Show Outdated Extensions` 下拉菜单，您可以快速查找扩展更新。它将显示您当前已安装扩展的所有可用更新。只需点击位于右下角的过时扩展更新按钮，更新就会被安装，你会收到重新启动VSCode的提示。

Tip: Code will also notify you of available updates in the extension icon at the bottom left corner of its window.

小技巧：VS Code也将在窗口的左下角的扩展图标通知您，有可用更新。

浏览扩展 Browse Extensions

Additionally, you can browse and search for VS Code extensions through the [VS Code Marketplace site](#).

此外，你可以在[VS Code 市场](#)网站，搜索浏览和检索VSCode扩展。

The screenshot shows the Visual Studio Marketplace homepage. At the top, there's a navigation bar with links for Visual Studio, Visual Studio Team Services, Visual Studio Code (which is highlighted in pink), Subscriptions, and a 'Build your own' button. To the right is a sign-in link and a search icon. Below the navigation is a header that reads 'Extensions for the Visual Studio family of products'. A search bar with a magnifying glass icon is positioned below the header. The main content area is titled 'Featured' and displays six extension cards. Each card includes the extension name, developer, download count, a brief description, a star rating, and a 'PREVIEW' or 'FREE' badge. The extensions shown are: C/C++ (Microsoft, 38.5K), React Native Tools (Visual Studio Mobile, 11.1K), Ruby (Peng Lv, 9K), Visual Studio Code Settings (Shan Khan, 6.9K), SVG (csho, 3K), and NativeScript (Telerik, 2.3K).

You can review our handy **Featured**, **Most Popular**, and **Recently Added** extension lists and filter by **Category** (Debuggers, Languages, Linters, etc.).

您可以按类别（调试器，语言，棉短绒等）预览我们便捷扩展列表和过滤器，包括特色，最受欢迎，最新添加。

This screenshot shows the 'Filter by category / collection' section of the Visual Studio Marketplace. It features a horizontal menu with several categories: Debuggers, Languages, Linters, Snippets, Themes, and Other. Each category has a corresponding button with a small icon above it. The 'Languages' category is currently selected, indicated by a darker background.

下一步 Next Steps

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

在这里，你可能会发现有趣的几个主题...

- [Publishing to the Marketplace](#) - Publish your own customization or extension to the VS

Code Marketplace

- [Publishing to the Marketplace](#) - 发布自己的定制或扩展到VSCode市场
- [Customization](#) - Learn how to integrate TextMate themes, colorizers and snippets into Visual Studio Code.
- [Customization](#) - 了解如何把TextMate的主题，着色器和片段整合到VS Code。
- [Yo Code](#) - Learn how the Yo Code extension generator can scaffold out new extensions and package existing TextMate files.
- [Extending Visual Studio Code](#) - Start learning about VS Code extensibility
- [Extending Visual Studio Code](#) - 开始学习VSCode的可扩展性
- [Your First Extension](#) - Try creating a simple Hello World extension
- [你的第一个扩展](#) - 试着创建一个简单的Hello World扩展。

常见问题 Common Questions

Q: The `Extensions: Install Extension` command just hangs and never shows a dropdown listing available extensions?

Q: `Extensions: Install Extension` 命令只是挂起，而没有显示一个下可用扩展的拉列表？

A: This could be due to an incomplete uninstall of an extension which left some extension files under [your `.vscode/extensions` folder](#). Navigate to `.vscode/extensions` and see if there is an extension folder (named for the publisher and extension as `publisher.extension`) for a recently deleted extension. Delete that folder and restart VS Code.

A: 这可能是由于扩展的不完全卸载，而在[你的 `.vscode/extensions` 目录](#)内留下一些扩展文件。到 `.vscode/extensions` 内，查看。如果有最近删除的扩展的目录（被命名为出版商和扩展名`publisher.extension`），删除这些目录，并重启VSCode。

Q: Can VS Code read TextMate bundles directly?

Q: VS Code能够直接读取TextMate bundles吗？

A: No, VS Code can read some TextMate files such as `.tmTheme` and `.tmLanguage` but can not install full TextMate bundles. Also in order to use TextMate theme and syntax files, VS Code needs extra metadata for integration. The [Yo Code](#) extension generator makes it easy to package these files for use in VS Code.

A: 不能, VS Code能够读取一些TextMate文件,诸如 `.tmTheme` 和 `.tmLanguage`，但不能安装完整的TextMate bundles。另外为了使用TextMate的主题和语法文件，VSCode需要集成额外的元数据。[Yo Code](#)扩展生成器可以很容易地打包这些文件，在VSCode中使用。

Q: Can I install Visual Studio Community extensions (shipped in `.vsix`) in Visual Studio Code?

Q: 我可以在VSCode中安装Visual Studio Community 扩展（以.vsix打包）吗？

A: No, Visual Studio Code's extensibility points are different from Visual Studio Community.

A: 不行, VSCode的扩展点和 visual Studio Community是不同的。

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

Q: 每当我尝试安装任何扩展，我得到一个连接错误**ETIMEDOUT**.

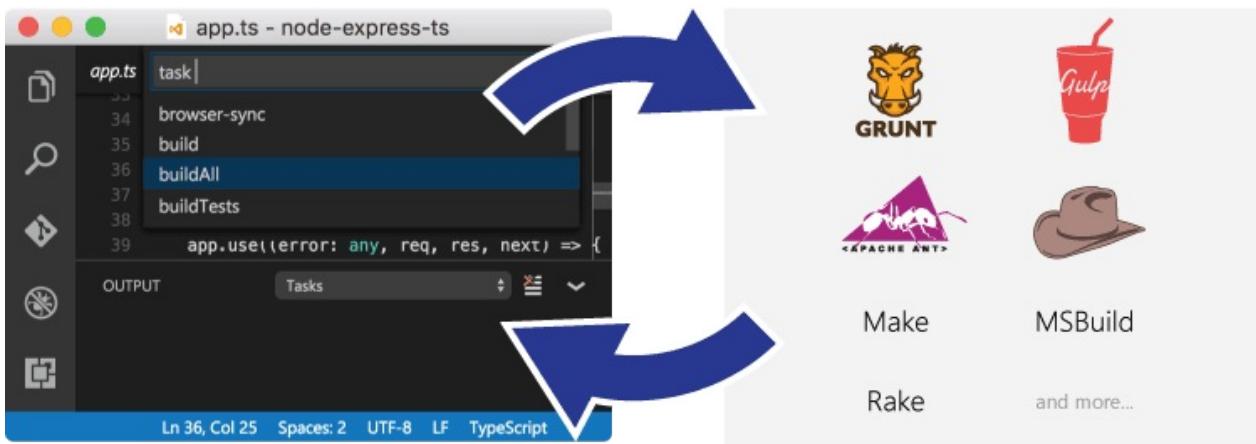
A: You may see this error if your machine is going through a proxy server to access the Internet. See the [Proxy Server Support](#) section in our FAQ for details.

A: 如果你的机器是通过代理服务器访问Internet，您可能会看到这个错误。请参阅我们的FAQ [代理服务器支持](#)部分。

通过任务集成外部工具 Integrate with External Tools via Tasks

存在很多工具来使构建，打包，测试或部署软件系统一类的任务自动运行。这些例子中有[Make](#), [Ant](#), [Gulp](#), [Jake](#), [Rake](#) and [MSBuild](#)。

Lots of tools exist to automate tasks like building, packaging, testing or deploying software systems. Examples include [Make](#), [Ant](#), [Gulp](#), [Jake](#), [Rake](#) and [MSBuild](#).



这些工具大多数是从命令行启动，并且自动运行的工作是在软件开发循环(编辑，编译，测试和调试)之外。考虑到他们在开发生命周期中的重要性，如果能在VS Code中运行他们并且分析结果将会是非常有帮助的。

These tools are mostly run from the command line and automate jobs outside the inner software development loop (edit, compile, test and debug). Given their importance in the development life-cycle, it is very helpful to be able run them and analyze their results from within VS Code.

请注意任务支持只在打开一个工作空间文件夹时可用。它在编辑单独的文件时不可用。

Please note that task support is only available when working on a workspace folder. It is not available when editing single files.

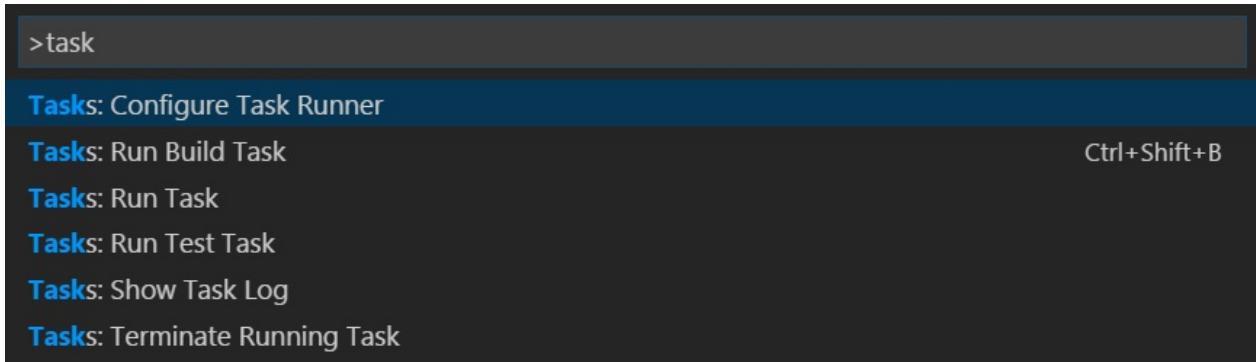
你好，世界 Hello World

让我们从一个简单的"Hello World"任务开始，它运行时会在输出面板上显示文字。

Let's start with a simple "Hello World" task which will display text to the **OUTPUT** panel when run.

在工作空间的 `task.json` 文件中定义任务，VS Code有一些一般的任务运行器的模版。在命令面板 (`kb(workbench.action.showCommands)`) 中，你可以使用'任务'过滤然后看到不同的关于任务的命令。

Tasks are defined in a workspace `tasks.json` file and VS Code has templates for common task runners. In the **Command Palette** (`kb(workbench.action.showCommands)`), you can filter on 'task' and can see the various Task related commands.



选择任务: 配置任务运行程序 命令，你将看到一个任务运行器的列表。选择运行任意外部命令的示例来创建一个运行外部命令的任务。

Select the **Tasks: Configure Task Runner** command and you will see a list of task runner templates. Select **Others** to create a task which runs an external command.

你现在应该在你的工作空间的 `.vscode` 文件夹下看到一个 `tasks.json` 的文件，里面的内容是如下：

You should now see a `tasks.json` file in your workspace `.vscode` folder with the following content:

```
{
  "version": "0.1.0",
  "command": "echo",
  "isShellCommand": true,
  "args": ["Hello World"],
  "showOutput": "always"
}
```

在这个例子中，我们只是运行了 `echo`，参数为"Hello World"的shell命令。

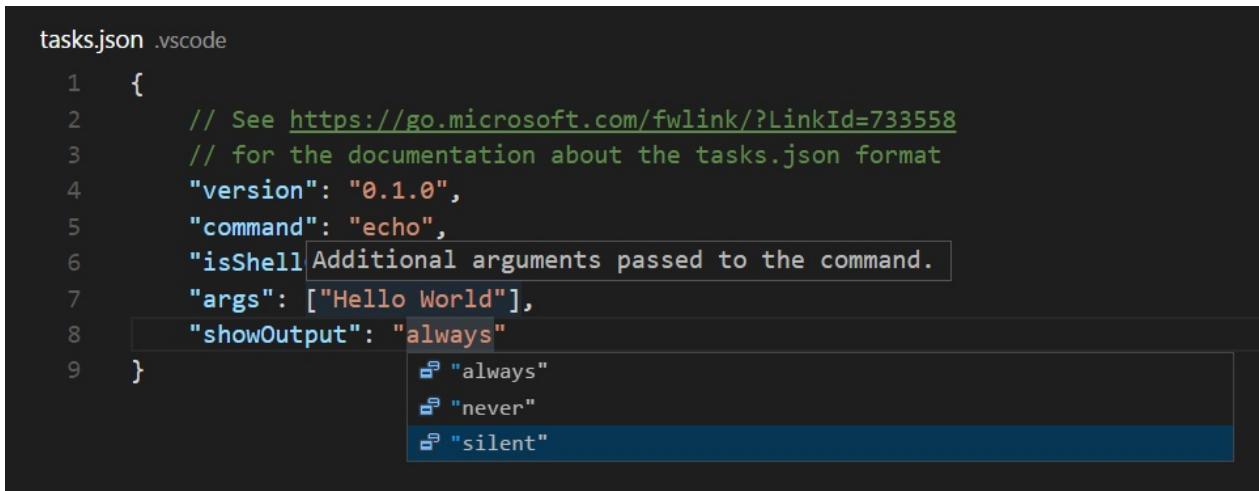
In this example, we are just running the `echo` shell command with "Hello World" as an argument.

运行任务: 运行任务测试这个 `echo` 任务，然后选择从下拉框中选择 `echo`。输出将打开，你将看到"Hello World"的文本。

Test the `echo` task by running **Tasks: Run Tasks** and selecting `echo` from the dropdown. The **OUTPUT** panel will open and you'll see the text "Hello World".

你在编辑 `tasks.json` 时鼠标移动到上面时可以得到关于变量键值的智能感知和使用 `kb(editor.action.triggerSuggest)` 的自动补全。

You can get IntelliSense on `tasks.json` variables and their values with hover and trigger smart completions with `kb(editor.action.triggerSuggest)`.



```
tasks.json .vscode
1  {
2      // See https://go.microsoft.com/fwlink/?LinkId=733558
3      // for the documentation about the tasks.json format
4      "version": "0.1.0",
5      "command": "echo",
6      "isShell": Additional arguments passed to the command.
7      "args": ["Hello World"],
8      "showOutput": "always"
9  }                                ↗ "always"
                                     ↗ "never"
                                     ↗ "silent"
```

提示:你可以键入'任务', `kbstyle(Space)` 和任务名称通过快速打开(`kb(workbench.action.quickOpen)`)运行你的任务。在这个例子中是'task echo'。

Tip: You can run your task through **Quick Open** (`kb(workbench.action.quickopen)`) by typing 'task', `kbstyle(Space)` and the command name. In this case, 'task echo'.

输出窗口的行为 Output Window Behavior

有时你希望在运行任务时控制输出窗口的行为。比如你可能希望最大化编辑器空间并且只在你认为有问题时查看任务输出。这由**showOutput**属性控制，合理的值有:

Sometimes you will want to control how the output window behaves when running tasks. For instance, you may want to maximize editor space and only look at task output if you think there is a problem. The property **showOutput** controls this and the valid values are:

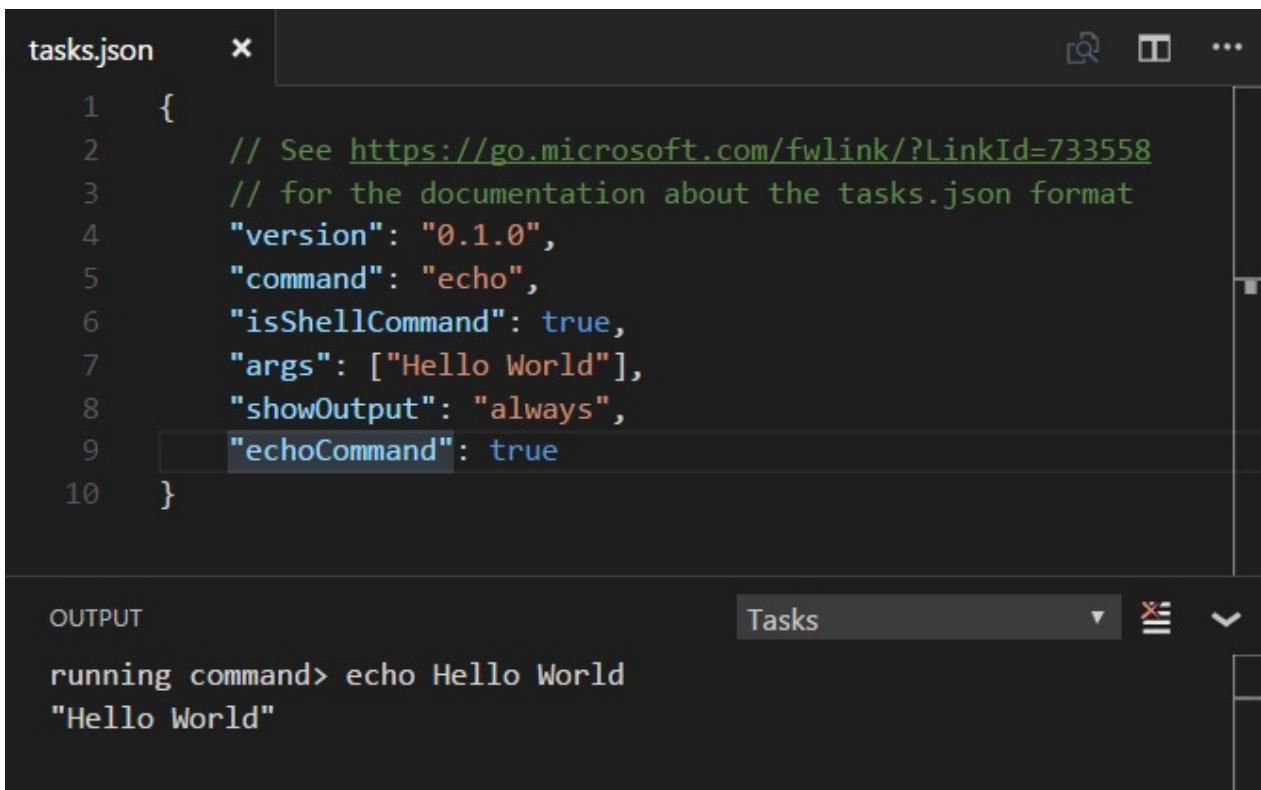
- **always** - The output window is always brought to front. This is the default.
- **always** - 输出窗口永远被放到前台。这是默认值。
- **never** - The user must explicitly bring the output window to the front using the **View > Toggle Output** command (`kb(workbench.action.output.toggleOutput)`).
- **never** - 用户必须明确使用查看 > 输出(`kb(workbench.action.output.toggleOutput)`)命令将输出窗口放到前台。

- **silent** - The output window is brought to front only if no [problem matchers](#) are set for the task.
- **silent** - 只有当没有设置[问题匹配器](#)时输出窗口才会被放到前台。

输出命令 echoCommand

为了看到VS Code正在运行的命令,如果可以启用 `tasks.json` 里的 `echoCommand` 选项 :

To see the exact command VS Code is running, you can enable the `echoCommand` setting in `tasks.json` :



The screenshot shows the VS Code interface with the tasks.json file open in the editor. The file contains the following JSON code:

```

1  {
2      // See https://go.microsoft.com/fwlink/?LinkId=733558
3      // for the documentation about the tasks.json format
4      "version": "0.1.0",
5      "command": "echo",
6      "isShellCommand": true,
7      "args": ["Hello World"],
8      "showOutput": "always",
9      "echoCommand": true
10 }

```

Below the editor, the Output panel shows the command being run and its output:

```

running command> echo Hello World
"Hello World"

```

注意: VS Code 运行 npm, MSBuild, maven 和其他命令行工具。一个学习任务的很好的方法是查看这些模版查看哪些工具或者任务运行器与你可能用到的其他工具相似。

Note: VS Code ships with predefined `tasks.json` templates to run npm, MSBuild, maven and other command line tools. A great way to learn about tasks is to review these templates and see which tools or task runners are similar to other tools you may be using.

命令和任务数组 command and tasks[]

`tasks.json` 接受单一的 `command` 值, 它可能是一个像 gulp, grunt 或者一切像编译器或者检查器的命令行工具的任务运行器。默认的 `command` 将会显示在任务: 运行任务的下拉框中。

`tasks.json` takes a single `command` value which can be a task runner like gulp or grunt or any command line tool like a compiler or linter. By default the `command` will show up in the **Tasks: Run Task** dropdown.

你也可以在 `tasks` 数组中定义多个任务来在运行 `command` 时传递不同的参数或使用不同的设置。

You can also define multiple tasks in a `tasks` array in order to pass different arguments or use different settings when the `command` is run.

这里是一个简单的传递不同参数给 `echo` 命令的例子：

Here's a simple example passing different arguments to the `echo` command:

```
{  
  "version": "0.1.0",  
  "command": "echo",  
  "isShellCommand": true,  
  "args": [],  
  "showOutput": "always",  
  "echoCommand": true,  
  "suppressTaskName": true,  
  "tasks": [  
    { "taskName": "hello",  
      "args": ["Hello World"],  
    },  
    { "taskName": "bye",  
      "args": ["Good Bye"],  
    }  
  ]  
}
```

现在当你运行任务：运行任务时，你将看到在下拉框中看到 `hello` 和 `bye` 两个任务。我们设置 `suppressTaskName` 为真，因为默认情况下任务名称也传递给命令，这样的结果是"echo hello Hello World"。

Now when you run **Tasks: Run Task**, you will now see two tasks in the dropdown `hello` and `bye`. We set `suppressTaskName` to true as by default the task name is also passed to the command which would result in "echo hello Hello World".

The screenshot shows the VS Code interface with the 'tasks.json' file open in the editor. The file contains configuration for two tasks: 'hello' and 'bye'. The 'hello' task runs 'echo Hello World' with the output 'Hello World'. The 'bye' task runs 'Good Bye'.

```

tasks.json  task
1   bye
2   hello
3
4       "isShellCommand": true,
5       "args": [],
6       "showOutput": "always",
7       "echoCommand": true,
8       "tasks": [
9           { "taskName": "hello",
10              "args": ["Hello World"],
11              "suppressTaskName": true
12          },
13          { "taskName": "bye",
14              "args": ["Good Bye"],
15              "suppressTaskName": true
16          }
17      ]
18  }

OUTPUT
running command> echo Hello World
"Hello World"
  
```

一些在 `task.json` 中的属性，像 `showOutput` 和 `suppressTaskName`，可以被设置为全局然后被特定任务重写。这些任务数组的 `args` 属性值被添加到全局参数。

Some `tasks.json` properties such as `showOutput` and `suppressTaskName` can be set both globally and then overridden in specific tasks. The `tasks args` property values are appended to the global arguments.

任务数组也有一些的特定属性。一个有用的属性是 `isBuildCommand`，如果它被设置为真，会从任务：运行生成任务 `kb(workbench.action.tasks.build)` 中运行。

There are also `tasks` specific properties. One useful property is `isBuildCommand`, which if set to true, will run the task with the **Tasks: Run Build Task** (`kb(workbench.action.tasks.build)`) command.

运行多种命令 Running multiple commands

要是你想在你的工作空间中运行不同的命令行工具怎么办？在 `tasks.json` 中定义不同的任务并没有被VS Code完全支持(参见 [#981](#))。你可以通过一个Shell命令(Linux 和 OS X上的 `sh`，Windows上的 `cmd`)运行你的任务命令来绕过这个限制。

What if you want to run different command line tools in your workspace? Defining multiple tasks in `tasks.json` is not yet fully supported by VS Code (see [#981](#)). You can work around this limitation by running your task commands through a shell command (`sh` on Linux and OS X, `cmd` on Windows).

这里是一个添加 `make` 和 `ls` 两个命令的示例：

Here is an example to add two tasks for `make` and `ls` :

```
{
  "version": "0.1.0",
  "command": "sh",
  "args": ["-c"],
  "isShellCommand": true,
  "showOutput": "always",
  "suppressTaskName": true,
  "tasks": [
    {
      "taskName": "make",
      "args": ["make"]
    },
    {
      "taskName": "ls",
      "args": ["ls"]
    }
  ]
}
```

`make` 和 `ls` 两个命令将在任务：运行任务下拉框中可见。

Both tasks `make` and `ls` will be visible in the **Tasks: Run Task** dropdown.

在Windows上, 你需要传递 '/C' 参数给 `cmd` 来运行任务.

For Windows, you will need to pass the '/C' argument to `cmd` so that the tasks arguments are run.

```
"command": "cmd",
"args": ["/C"]
```

变量替换 Variable substitution

当创作任务配置时，一写预先定义的一般变量经常很有用。VS Code支持 `tasks.json` 中字符串里的变量替换，并且提供以下预先定义的变量：

When authoring tasks configurations, it is often useful to have a set of predefined common variables. VS Code supports variable substitution inside strings in the `tasks.json` file and has the following predefined variables:

- `${workspaceRoot}` VS Code中打开的文件夹的路径
- `${workspaceRoot}` the path of the folder opened in VS Code
- `${file}` 当前打开的文件路径
- `${file}` the current opened file
- `${relativeFile}` 当前打开的文件相对于 `workspaceRoot` 的路径
- `${relativeFile}` the current opened file relative to `workspaceRoot`
- `${fileBasename}` 当前打开的文件的basename
- `${fileBasename}` the current opened file's basename
- `${fileDirname}` 当前打开的文件的文件夹名
- `${fileDirname}` the current opened file's dirname
- `${fileExtname}` 当前打开的文件扩展名
- `${fileExtname}` the current opened file's extension
- `${cwd}` 任务运行器启动时的当前路径
- `${cwd}` the task runner's current working directory on startup

你也可以使用 `${env.Name}` (比如 `${env.PATH}`)来引用环境变量。请确认它匹配环境变量的大小写，比如Windows上的 `env.Path` 。

You can also reference environment variables through `${env.Name}` (e.g. `${env.PATH}`). Be sure to match the environment variable name's casing, for example `env.Path` on Windows.

下面是一个传递当前打开的文件给TypeScript编译器的配置项的例子。

Below is an example of a configuration that passes the current opened file to the TypeScript compiler.

```
{  
  "command": "tsc",  
  "args": ["${file}"]  
}
```

操作系统特有的属性 Operating System Specific Properties

任务系统支持定义不同的变量(比如要运行的命令)给特定的操作系统。要这样做，只需要在 `tasks.json` 中添加一个操作系统特定的字面量并且在字面量中指定相符合的属性。

The task system supports defining values (for example, the command to be executed) specific to an operating system. To do so, simply put an operating system specific literal into the `tasks.json` file and specify the corresponding properties inside that literal.

下面是一个使用Node.js可执行文件作为命令的例子，并且它对Windows和Linux做了不同的对待：

Below is an example that uses the Node.js executable as a command and is treated differently on Windows and Linux:

```
{
  "version": "0.1.0",
  "windows": {
    "command": "C:\\Program Files\\nodejs\\node.exe"
  },
  "linux": {
    "command": "/usr/bin/node"
  }
}
```

合法的操作系统属性包括"windows", "linux"和"osx"。特定操作系统中定义的属性将重写全局范围定义的属性。

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

在下面的例子中：

In the example below:

```
{
  "version": "0.1.0",
  "showOutput": "never",
  "windows": {
    "showOutput": "always"
  }
}
```

运行的任务的输出在其他系统中永远不会放在前台，但是在Windows中它永远显示。

Output from the executed task is never brought to front except for Windows where it is always shown.

任务的例子 Examples of Tasks in Action

最好的突出任务作用的方法是提供几个VS Code使用任务来集成检查器和编译器的例子。

To highlight the power of Tasks, here are a few examples of how VS Code can use Tasks to integrate external tools like linters and compilers.

编译TypeScript到JavaScript Transpiling TypeScript to JavaScript

[TypeScript主题](#) 包含了一个创建一个任务来编译TypeScript到JavaScript并且在VS Code中观察任何相关的错误的例子。

The [TypeScript topic](#) includes an example that creates a task to transpile TypeScript to JavaScript and observe any related errors from within VS Code.

编译Markdown到HTML Compiling Markdown to HTML

Markdown主题提供了两个编译Markdown到HTML的例子。

The Markdown topic provides two examples for compiling Markdown to HTML:

1. [通过构建项目手动编译](#)
2. [通过文件观察者自动编译](#)
3. [Manually compiling with a Build task](#)
4. [Automation of the compile step with a file watcher](#)

编译Sass和Less到CSS Transpiling Less and Sass into CSS

CSS主题提供了怎样使用任务生成CSS文件的例子。

The CSS topic provides examples of how to use Tasks to generate CSS files.

1. [通过构建项目手动编译](#)
2. [通过文件观察者自动编译](#)
3. [Manually transpiling with a Build task](#)

4. Automation of the compile step with a file watcher

自动发现Gulp，Grunt 和 Jake 任务 Autodetecting Gulp, Grunt and Jake Tasks

VS Code可以自动发现包含Gulp, Grunt 和 Jake文件。不需要额外的配置来添加它们的任务到任务列表中（除非你需要使用一个问题匹配器，稍后再作更多的介绍）。

VS Code can autodetect tasks from within Gulp, Grunt and Jake files. This adds their tasks to the task list without requiring additional configuration (unless you need to use a problem matcher, more on that in a moment).

为了使这个例子更加具体，我们使用一个简单的Gulp文件。这里定义了两个任务：构建和调试。第一个使用[Mono](#)的编译器编译了C#代码，第二个在Mono调试器下启动MyApp。

To help make this example more concrete, let's use this simple Gulp file. This defines two tasks: build and debug. The first compiles C# code using [Mono](#)'s compiler. The second starts the MyApp under the Mono debugger.

```
var gulp = require("gulp");

var program = "MyApp";
var port = 55555;

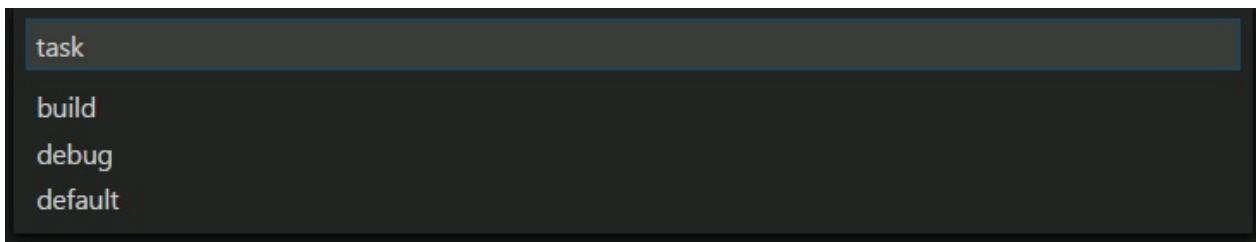
gulp.task('default', ['debug']);

gulp.task('build', function() {
    return gulp
        .src('./**/*.cs')
        .pipe(msc(['-fullpaths', '-debug', '-target:exe', '-out:' + program]));
});

gulp.task('debug', ['build'], function(done) {
    return mono.debug({ port: port, program: program}, done);
});
```

处理 `kb(workbench.action.showCommands)` 然后键入 `Run Task` 加上 `kbstyle(Enter)` 将显示所有可得的任务。选择一个然后按 `kbstyle(Enter)` 将执行这个任务。

Pressing `kb(workbench.action.showCommands)` and then typing `Run Task` followed by `kbstyle(Enter)` will list all available tasks. Selecting one and pressing `kbstyle(Enter)` will execute the task.



注意: Gulp, Grunt 和 Jake 只有在相符合的文件(比如 `gulpfile.js`)在打开的文件夹的根部录下存在时才会自动发现。

Note: Gulp, Grunt and Jake are autodetected only if the corresponding files (for example `gulpfile.js`) are present in the root of the opened folder.

使用问题匹配器处理任务输出 Processing Task Output with Problem Matchers

VS Code 使用问题匹配器处理任务的输出，然后我们在'箱子里'使用它们：

VS Code can process the output from a task with a problem matcher and we ship with a number of them 'in the box':

- **TypeScript:** `$tsc` assumes that file names in the output are relative to the opened folder.
- **TypeScript Watch:** `$tsc-watch` matches problems reported from the `tsc` compiler when executed in watch mode.
- **JSHint:** `$jshint` assumes that file names are reported as an absolute path.
- **JSHint Stylish:** `$jshint-stylish` assumes that file names are reported as an absolute path.
- **ESLint Compact:** `$eslint-compact` assumes that file names in the output are relative to the opened folder.
- **ESLint Stylish:** `$eslint-stylish` assumes that file names in the output are relative to the opened folder.
- **CSharp and VB Compiler:** `$mscompile` assumes that file names are reported as an absolute path.
- **Less:** `$lessCompile` assumes that file names are reported as absolute path.
- **TypeScript:** `$tsc` 假定输出中的文件名与打开的文件夹有关。
- **TypeScript Watch:** 当执行观察模式时 `$tsc-watch` 匹配 `tsc` 编译器报告的问题。
- **JSHint:** `$jshint` 假定报告的文件名是一个绝对路径。
- **JSHint Stylish:** `$jshint-stylish` 假定报告的文件名是一个绝对路径。
- **ESLint Compact:** `$eslint-compact` 假定输出中的文件名与打开的文件夹有关。
- **ESLint Stylish:** `$eslint-stylish` 假定输出中的文件名与打开的文件夹有关。

- **CSharp and VB Compiler:** `$mscompile` 假定报告的文件名是一个绝对路径。
- **Less:** `$lessCompile` 假定输出中的文件名与打开的文件夹有关。

问题匹配器扫描任务输出的文本来寻找已知的警告或错误字符串，然后报告在编辑器的问题面板中内联的报告它们。可以全局设置或在特定的任务入口中设置问题匹配器。

Problem matchers scan the task output text for known warning or error strings and report these inline in the editor and in the Problems panel. Problem matchers can be set globally or in a specific task entry.

你也可以创建你自己的问题匹配器，我们将很快讨论它。

You can also create your own problem matcher which we'll talk about soon.

将 Gulp, Grunt 和 Jake 的输出映射到问题匹配器 Mapping Gulp, Grunt and Jake Output to Problem Matchers

你需要设置 `tasks.json` (在你的工作空间的 `.vscode` 文件夹中)文件中的任务如果你选哟做更多的事而不仅仅简单的运行这个任务。比如，你也许想在VS Code中匹配并高亮报告的错误，或者想使用运行生成任务(`kb(workbench.action.tasks.build)`)命令来触发生成任务。

You need to configure the tasks in a `tasks.json` file (located under your workspace `.vscode` folder) if you want to do more than simply run the task. For example, you might want to match reported problems and highlight them within VS Code, or to trigger a build task using the **Tasks: Run Build Task** command (`kb(workbench.action.tasks.build)`).

如果你没有一个在工作空间里的 `.vscode` 目录下的 `tasks.json` 文件，运行命令面板(`kb(workbench.action.showCommands)`)的任务：配置任务运行程序操作将给你提供一些模版来挑选。

If you don't already have a `tasks.json` under your workspace `.vscode` folder, running the **Tasks: Configure Task Runner** action from the **Command Palette** (`kb(workbench.action.showCommands)`) will offer you a set of templates to pick from.

比如，从列表中选择 `Gulp`。这将生成一个类似一下内容的 `tasks.json`：

For this example, select `Gulp` from the list. Given a `gulpfile.js` like the example above, this will generate a `tasks.json` file like this:

```
{
    // See http://go.microsoft.com/fwlink/?LinkId=733558
    // for the documentation about the tasks.json format
    "version": "0.1.0",
    "command": "gulp",
    "isShellCommand": true,
    "args": [
        "--no-color"
    ],
    "tasks": [
        {
            "taskName": "build",
            "args": [],
            "isBuildCommand": true,
            "isWatching": false,
            "problemMatcher": [
                "$lessCompile",
                "$tsc",
                "$jshint"
            ]
        }
    ]
}
```

既然我们运行了 Mono 编译器来编译 C# 文件，我们使用 `$msCompile` 问题匹配器来发现编译器报告的任何问题。

Since we execute the Mono compiler to compile C# files, we should use the `$msCompile` problem matcher to detect any problems reported by the compiler.

这个 `problemMatcher` 属性将会是:

The `problemMatcher` property will then be:

```
"problemMatcher": [
    "$msCompile"
]
```

关于这个 `tasks.json`，我们需要注意:

Several things to note about this `tasks.json`:

1. 我们想在 shell 中运行 Gulp 命令 (VS Code 直接运行它)，所以我们使用 `isShellCommand`.
2. 我们增加了一个明确的 `tasks` 属性，它使我们能够 *optionally augment a task that was in the gulpfile*.
3. 我们定义了一个问题匹配器 `$msCompile` 来处理输出-既然我们正在使用 Mono 编译器编译 C#，内建的一个任务 `msc` 依赖微软编译模式。

4. `iswatching` 属性被设为假所以我们不能在源文件改动时自动运行声称命令。
5. We want to run the gulp command in a shell (VS Code directly executing it) so we used `isShellCommand`.
6. We added an explicit `tasks` property which allowed us to *optionally* augment a task that was in the `gulpfile.js`.
7. We defined a problem matcher `$msCompile` to process the output - since we are compiling C# using the Mono compiler, the built-in one works as `msc` adheres to the Microsoft compiler pattern.
8. The `iswatching` property is set to false so we won't automatically run the build on source code file changes.

定义一个问题匹配器 Defining a Problem Matcher

VS Code ships some of the most common problem matchers out of the box.但是，有很多编译器或者检查工具不在这里，它们产生自己独特的错误和警告风格，所以我们来讨论怎样创建你自己的问题匹配器。

VS Code ships some of the most common problem matchers out of the box. However, there are lots of compilers and linting tools out there, all of which produce their own style of errors and warnings. So let's talk about how to make your own problem matcher.

我们现在有一个 `helloworld.c` 程序，其中开发者将 `printf` 错误拼写为 `prinft`。使用 `gcc` 编译它会产生如下警告：

We have a `helloworld.c` program in which the developer mistyped `printf` as `prinft`. Compiling it with `gcc` will produce the following warning:

```
helloworld.c:5:3: warning: implicit declaration of function ‘prinft’
```

我们希望产生一个能够捕获输出窗口信息的问题匹配器，然后在 VS Code 中显示相符合的问题匹配器。问题匹配器严重依赖 [正则表达式](#)。下面的章节假定你很熟悉正则表达式。

We want to produce a problem matcher that can capture the message in the output and show a corresponding problem in VS Code. Problem matchers heavily rely on [regular expressions](#). The section below assumes you are familiar with regular expressions.

提示：我们发现 [RegEx101 playground](#) 是一个开发和测试正则表达式的好东西。

Tip: We have found the [RegEx101 playground](#) a really good way to develop and test regular expressions.

一个捕捉以上警告(和错误)的匹配器看起来是这样：

A matcher that captures the above warning (and errors) looks like:

```
{  
    // The problem is owned by the cpp language service.  
    "owner": "cpp",  
    // The file name for reported problems is relative to the opened folder.  
    "fileLocation": ["relative", "${workspaceRoot}"],  
    // The actual pattern to match problems in the output.  
    "pattern": {  
        // The regular expression. Example to match: helloWorld.c:5:3: warning: implicit declaration of function 'printf' [-Wimplicit-function-declaration]  
        "regexp": "^(.+):(\d+):(\d+):\s+(warning|error):\s+(.+)$",  
        // The first match group matches the file name which is relative.  
        "file": 1,  
        // The second match group matches the line on which the problem occurred.  
        "line": 2,  
        // The third match group matches the column at which the problem occurred.  
        "column": 3,  
        // The fourth match group matches the problem's severity. Can be ignored. Then all problems are captured as errors.  
        "severity": 4,  
        // The fifth match group matches the message.  
        "message": 5  
    }  
}
```

请注意文件，行数和信息属性是强制的。

Please note that the file, line and message properties are mandatory.

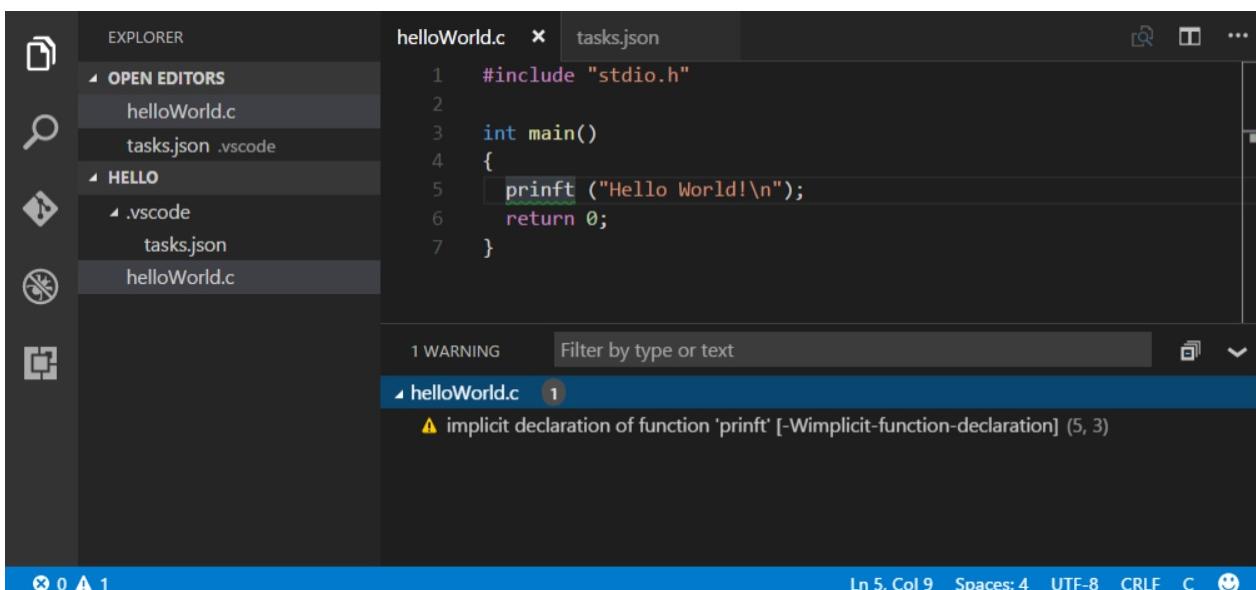
这是一个以上代码完整(去除注释)的具有实际任务细节的 tasks.json 文件：

Here is a finished `tasks.json` file with the code above (comments removed) wrapped with the actual task details:

```
{
  "version": "0.1.0",
  "command": "gcc",
  "args": ["-Wall", "helloWorld.c", "-o", "helloWorld"],
  "problemMatcher": {
    "owner": "cpp",
    "fileLocation": ["relative", "${workspaceRoot}"],
    "pattern": {
      "regexp": "^(.*):(\\d+):(\\d+):\\s+(warning|error):\\s+(.*)$",
      "file": 1,
      "line": 2,
      "column": 3,
      "severity": 4,
      "message": 5
    }
  }
}
```

在VS Code中运行它，按 `kb(workbench.actions.view.problems)` 来得到问题的列表，得到以下输出：

Running it inside VS Code and pressing `kb(workbench.actions.view.problems)` to get the list of problems gives you the following output:



这里有一些模式中可以用到的属性：

There are a couple more properties that can be used inside a pattern. These are:

- **location** 如果问题位置是行 或者 行,列 或者 开始行, 开始列, 结束行, 结束列 , 可以使用我们的一般位置匹配组。
- **endLine** 匹配组中问题结束行的索引。当编译器不产生结束行时可以省略。
- **endColumn** 匹配组中问题结束列的索引。当编译器不产生结束列时可以省略。
- **code** 匹配组中出现问题的代码。当编译器不产生代码值时可以省略。

- **location** if the problem location is line or line,column or startLine,startColumn,endLine,endColumn then our generic location match group can be used.
- **endLine** the match group index for the problem's end line. Can be omitted if no end line value is provided by the compiler.
- **endColumn** the match group index for the problem's end column. Can be omitted if no end column value is provided by the compiler.
- **code** the match group index for the problem's code. Can be omitted if no code value is provided by the compiler.

注意:一个可用的模式必须至少提供一个文件，信息和行或者位置的匹配组。

Note: A functional pattern must at least provide a match group for file, message and line or location.

定义多行问题匹配器 **Defining a Multi-Line Problem Matcher**

一些工具将在源文件中找到的问题分散在多行，特别是风格报告器这么用。一个例子是[ESLint](#)，在风格模式中它产生的输出像这样：

Some tools spread problems found in a source file over several lines, especially if stylish reporters are used. An example is [ESLint](#); in stylish mode it produces output like this:

```
test.js
  1:0  error  Missing "use strict" statement          strict
* 1 problems (1 errors, 0 warnings)
```

我们的问题匹配器是基于行的，所以我们需要使用与实际问题位置和信息(1:0 error Missing "use strict" statement)不同的正则表达式捕捉文件名(test.js)。

Our problem matcher is line-based so we need to capture the file name (test.js) with a different regular expression than the actual problem location and message (1:0 error Missing "use strict" statement).

要这么做，我们使用一个问题模式的数组来定义**pattern**属性。这样你可以定义每一行的匹配器来匹配。

To do this we use an array of problem patterns for the **pattern** property. This way you define a pattern per each line you want to match.

以下的问题匹配器匹配ESLint风格模式的输出 - 但是仍然有一个很小问题，我们将在以后解决。以下的代码有一个第一行的正则表达式来捕捉文件名，第二行捕捉行，列，重要性，信息和错误码：

The following problem pattern matches the output from ESLint in stylish mode - but still has one small issue which we will resolve next. The code below has a first regular expression to capture the file name and the second to capture the line, column, severity, message and error code:

```
{
  "owner": "javascript",
  "fileLocation": ["relative", "${workspaceRoot}"],
  "pattern": [
    {
      "regexp": "^([^\s].*)$",
      "file": 1
    },
    {
      "regexp": "\n\s+(\d+):(\d+)\s+(error|warning|info)\s+(.*\n\s+(.*))$"
    }
  ]
}
```

当然事情不是一直这么简单，如果问题多于一个，这个模式不会工作。比如。想象ESLint的以下输出：

Of course it's never quite that simple, and this pattern will not work if there is more than one problem on a resource. For instance, imagine the following output from ESLint:

```
test.js
  1:0  error  Missing "use strict" statement          strict
  1:9  error  foo is defined but never used        no-unused-vars
  2:5  error  x is defined but never used        no-unused-vars
  2:11 error  Missing semicolon                  semi
  3:1  error  "bar" is not defined            no-undef
  4:1  error  Newline required at end of file but not found eol-last
* 6 problems (6 errors, 0 warnings)
```

这个模式的第一行正则表达式将匹配"test.js"，第二行匹配"1:0 error ..."。下一行的"1:9 error ..."被处理但是不会被第一行的正则表达式捕获所以没捕捉到问题。

The pattern's first regular expression will match "test.js", the second "1:0 error ...". The next line "1:9 error ..." is processed but not matched by the first regular expression and so no problem is captured.

为了使它能够工作，最后一个正则表达式可以指定为**loop**属性。如果它被设置为**true**，它指导任务系统在可以匹配时多次使用多行匹配器的最后一个模式匹配输出的行。

To make this work, the last regular expression of a multi-line pattern can specify the **loop** property. If set to **true**, it instructs the task system to apply the last pattern of a multi-line matcher to the lines in the output as long as the regular expression matches.

所有之前的模式捕捉到的信息都被组合到最后一个模式匹配到的信息，然后成为VS Code的一个问题。

The information captured by all previous patterns is combined with the information captured by the last pattern and turned into a problem inside VS Code.

这里是一个完全捕捉ESLint风格模式的问题匹配器：

Here is a problem matcher to fully capture ESLint stylish problems:

```
{
  "owner": "javascript",
  "fileLocation": ["relative", "${workspaceRoot}"],
  "pattern": [
    {
      "regexp": "^([^\s].*)$",
      "file": 1
    },
    {
      "regexp": "\n+(\\d+):(\\d+)(error|warning|info)\n+(.*)\n+(.)$"
    }
  ]
}
```

下一步

任务就到这里 - 让我们继续 That was tasks - let's keep going...

- [tasks.json Schema](#) - 你可以检查完整的 tasks.json 概要和描述.

- [Editing Evolved](#) - 检查，智能感知，灯泡，获取和转到定义，和更多。
- [Language Support](#) - 通过VS Code和社区插件学习我们怎样支持编程语言。
- [Debugging](#) - 直接在VS Code编辑器中调试你的源代码。
- [tasks.json Schema](#) - You can review the full `tasks.json` schema and descriptions.
- [Editing Evolved](#) - Lint, IntelliSense, Lightbulbs, Peek and Go to Definition and more.
- [Language Support](#) - Learn about our supported programming languages, both shipped with VS Code and through community extensions.
- [Debugging](#) - Debug your source code directly in the VS Code editor.

一般问题

问：怎样定义运行多个任务来执行不同的命令？

Q: How can I define multiple tasks to run different commands?

答：在 `tasks.json` 中定义不同的任务并没有被VS Code完全支持(参见 [#981](#))。你可以通过一个Shell命令(Linux 和 OS X上的 `sh`，Windows上的 `cmd`)运行你的任务命令来绕过这个限制。

A: Defining multiple tasks in `tasks.json` is not yet fully supported by VS Code (see [#981](#)).
You can work around this limitation by running your task commands through a shell command (`sh` on Linux and OS X, `cmd` on Windows).

这里是一个添加 `make` 和 `ls` 两个命令的示例：

Here is an example to add two tasks for `make` and `ls` :

```
{
  "version": "0.1.0",
  "command": "sh",
  "args": ["-c"],
  "isShellCommand": true,
  "showOutput": "always",
  "suppressTaskName": true,
  "tasks": [
    {
      "taskName": "make",
      "args": ["make"]
    },
    {
      "taskName": "ls",
      "args": ["ls"]
    }
  ]
}
```

`make` 和 `ls` 两个命令将在任务: 运行任务下拉框中可见。

Both tasks `make` and `ls` will be visible in the **Tasks: Run Task** dropdown.

在Windows上, 你需要传递 '/C' 参数给 `cmd` 来运行任务.

For Windows, you will need to pass the '/C' argument to `cmd` so that the tasks arguments are run.

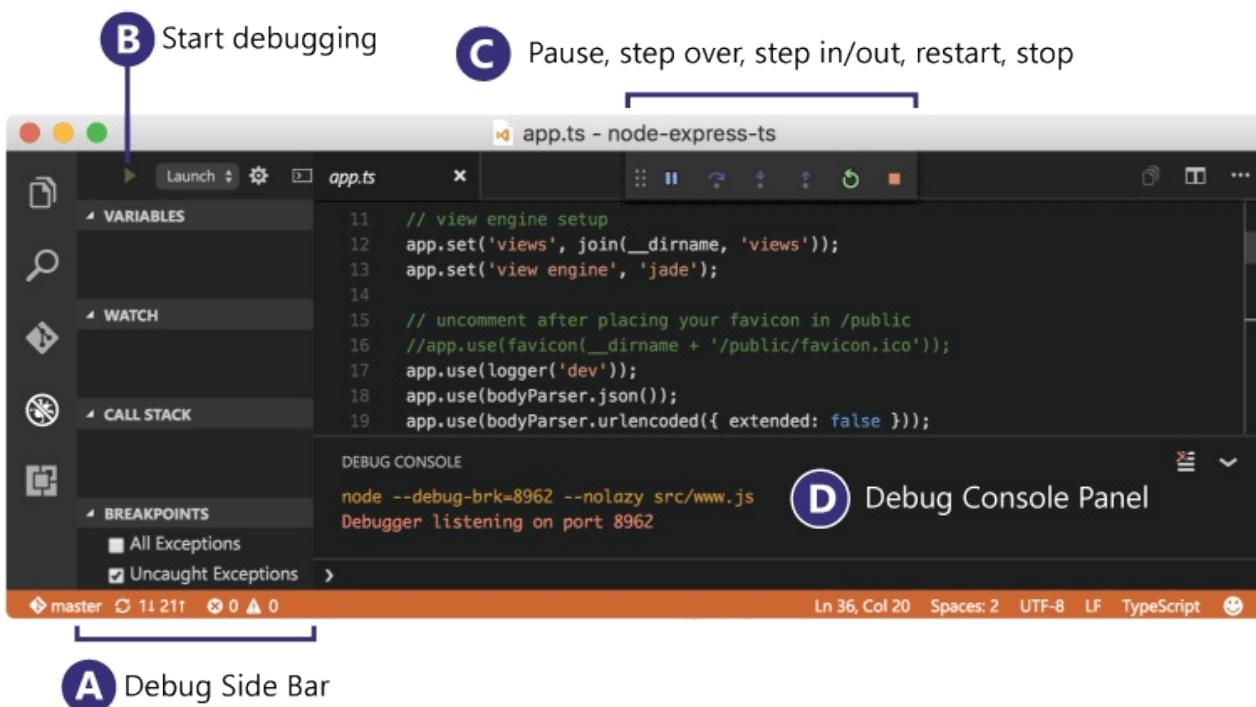
```
"command": "cmd",
"args": ["/C"]
```

调试

Debugging

Visual Studio Code的一个很重要的特性是他很棒的调试支持。VS Code的内置调试器帮助加速你的编辑，编译和调试循环。

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.



调试扩展

Debugger Extensions

VS Code内置一个支持[Node.js](#)运行时的调试，可以调试JavaScript，TypeScript和其他可以编译到JavaScript的语言。

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

为了调试其他语言和运行时（包括[PHP](#), [Ruby](#), [Go](#), [C#](#), [Python](#)和其他），请到我们的VS Code [市场](#)寻找 [调试器](#) 插件。

For debugging other languages and runtimes (including [PHP](#), [Ruby](#), [Go](#), [C#](#), [Python](#) and many others), please look for [Debuggers](#) [extensions](#) in our [VS Code Marketplace](#).

下面是一些流行的包含调试支持的插件：

Below are several popular extension which include debugging support:

提示：上边展示的插件是动态查询的。单击插件的标题来阅读他们的描述，决定哪个插件最适合你。

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.

开始调试

Start Debugging

下面的文档基于内置的[Node.js](#)调试器，但是很多的概念和特性也能应用到其他调试器中。

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.

在阅读关于调试的部分前创建一个简单的[Node.js](#)应用是很有帮助的。你可以跟随[Node.js walkthrough](#)来安装[Node.js](#)和创建一个简单的"Hello World"JavaScript应用(`app.js`)。一旦你搭建完成一个简单的一个应用，这一节将给你介绍VS Code调试的方方面面。

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 all set up, this page will take you through VS Code debugging features.

调试视图

Debug View

为了把调试视图放在最前，单击VS Code侧边的视图栏中的调试图标。

To bring up the Debug view, click on the Debugging icon in the View Bar on the side of VS Code.



调试视图将会显示所有和调试器有关的信息，并且在最上方显示一个带有调试命令和配置的面板。

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

运行配置

Launch Configurations

为了在VS Code中调试你的应用，你首先需要创建你的运行配置文件 - `launch.json`。单击最上面板调试视图中的齿轮图标，选择你的调试环境，然后VS Code将在你的工作空间的 `.vscode` 目录下生成一个 `launch.json` 文件。

To debug your app in VS Code, you'll first need to set up your launch configuration file - `launch.json`. Click on the Configure gear icon on the Debug view top bar, choose your debug environment and VS Code will generate a `launch.json` file under your workspace's `.vscode` folder.

这里是一个生成的Node.js调试器：

Here is the one generated for Node.js debugging:

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Launch",
      "type": "node",
      "request": "launch",
      "program": "${workspaceRoot}/app.js",
      "stopOnEntry": false,
      "args": [],
      "cwd": "${workspaceRoot}",
      "preLaunchTask": null,
      "runtimeExecutable": null,
      "runtimeArgs": [
        "--nolazy"
      ],
      "env": {
        "NODE_ENV": "development"
      },
      "externalConsole": false,
      "sourceMaps": false,
      "outDir": null
    },
    {
      "name": "Attach",
      "type": "node",
      "request": "attach",
      "port": 5858,
      "address": "localhost",
      "restart": false,
      "sourceMaps": false,
      "outDir": null,
      "localRoot": "${workspaceRoot}",
      "remoteRoot": null
    }
  ]
}
```

请注意这些运行设置中的属性对于不同的调试器不同。你可以使用智能感知来找出对于一个调试器哪些属性是存在的。另外，鼠标悬停帮助对所有属性都有效。如果在你的运行配置文件中看到绿色波浪线，鼠标悬停在上面来问题是什么和尝试在运行一个调试会话前修复。

Please note that the attributes available in these launch configurations vary from debugger to debugger. You can use IntelliSense to find out which attributes exist for a specific debugger. In addition, hover help is available for all attributes. 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.

在VS Code中我们支持用调试模式启动你的应用，或者连接到一个已经运行的应用。取决于不同的要求(`attach` 或者 `launch`)，需要不同的属性。我们的 `launch.json` 确认和建议应该对它有帮助。

In VS Code, we support launching your app in debug mode or attaching to an already running app. Depending on the request (`attach` or `launch`) different attributes are required and our `launch.json` validation and suggestions should help with that.

查看生成的值，并且确认它们对你的项目和调试环境有效。你可以添加额外的配置到 `launch.json` 中(使用悬停和智能感知来帮助你)。

Review the generated values and make sure that they make sense for your project and debugging environment. You can add additional configurations to the `launch.json` (use hover and IntelliSense to help).

在调试视图中选择名为 `Launch` 的配置使用 **Configuration dropdown**。一旦你完成运行配置的创建，使用 `kb(workbench.action.debug.start)` 开始你的调试会话。

Select the configuration named `Launch` using the **Configuration dropdown** in the Debug view. Once you have your launch configuration set, start your debug session with `kb(workbench.action.debug.start)` .

为了在开始一个调试会话前运行一个任务，设置 `preLaunchTask` 为 [tasks.json](#)()中指定的任务。

To launch a task before the start of each debug session, set the `preLaunchTask` to the name of one of the tasks specified in [tasks.json](#) (located under the workspace's `.vscode` folder).

VS Code 支持和配置 [tasks.json](#) 相同字符串中的的变量替换。

VS Code supports variable substitution inside strings in `launch.json` the same way as for [tasks.json](#).

运行模式

Run mode

除了调试一个程序，VS Code 支持运行一个程序。运行动作被 `'kb(workbench.action.debug.run)'` 触发，并且使用当前选中的运行配置。'运行'模式支持很多运行配置属性。当程序正在运行时VS Code 维护一个调试会话，按下停止按钮来终止这个程序。

In addition to debugging a program, VS Code supports running the program. The **Run** action is triggered with `kb(workbench.action.debug.run)` 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.

请注意：运行动作一直可用，但是调试扩展需要“选择进入”来支持“运行”。如果一个调试扩展没有被更新，“运行”将会退化到“调试”（内置的Node.js调试器和[Mono 调试器](#)已经支持“运行”了）

Please note: The **Run** action is always available, but a debugger extension has to 'opt-in' in order to support 'Run'. If a debugger extension has not been updated, 'Run' will fall back to 'Debug' (the built-in Node.js Debug and [Mono Debug](#) already support 'Run').

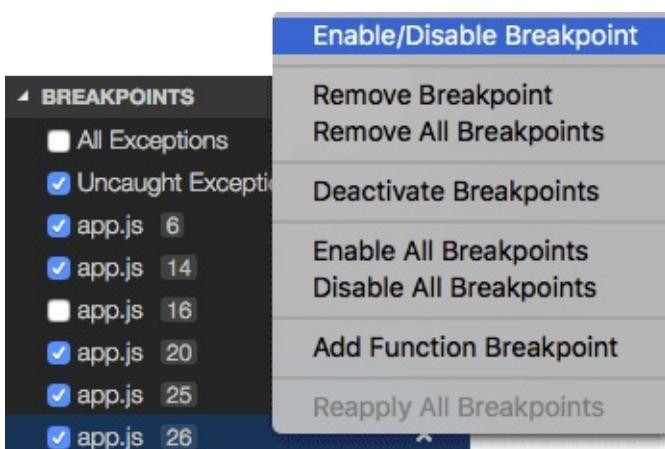
断点

Breakpoints

点击编辑器边距可以设置或取消断点。更好的断点控制(启用/禁用/重新启用)可以在调试视图中的断点区域设置。

Breakpoints can be toggled by clicking on the **editor margin**. Finer breakpoint control (enable/disable/reapply) can be done in the Debug view's **BREAKPOINTS** section.

- 编辑器边距中的断点一般用红色的实心圆表示。
- 禁用的断点有一个涂成灰色的圆。
- 当一个调试会话开始后，不能注册到调试器的断点变为灰色突起的圆。
- Breakpoints in the editor margin are normally shown as red filled circles.
- Disabled breakpoints have a filled gray circle.
- When a debugging sessions starts, breakpoints that cannot be registered with the debugger change to a gray hollow circle.



命令再次设置所以断点到它们最初的位置。如果你的调试环境是"lazy"这很有用，并且源码中"错位"的断点还没有被执行的。（更多的细节参考下面的[Node 调试](#)：断点）

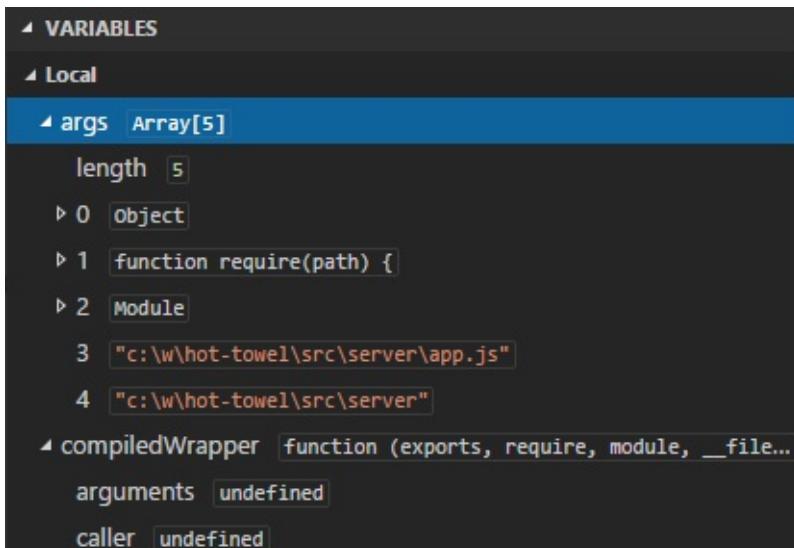
The `Reapply All Breakpoints` command sets all breakpoints again to their original location. This is helpful if your debug environment is "lazy" and "misplaces" breakpoints in source code that has not yet been executed. (For details see below under **Node Debugging: Breakpoint Validation**)

数据注入

Data inspection

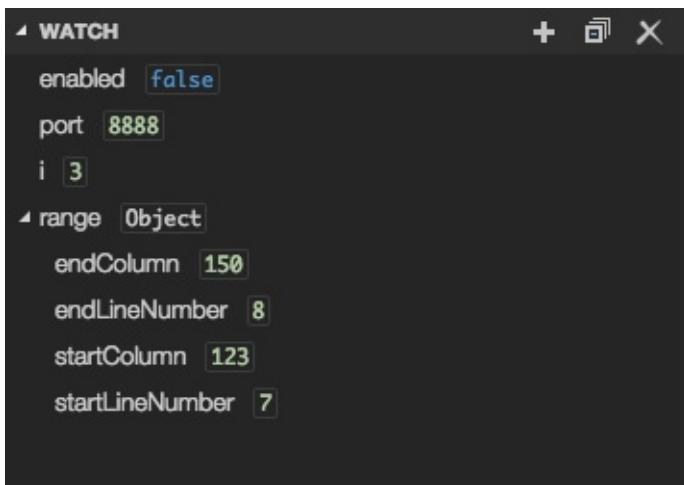
可以注入变量到调试视图的变量区域，或者通过悬停在编辑器中的源码上。变量和表达式计算是相对于调用栈中选中的栈帧的。

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



变量和表达式可以在调试视图的观察区域中被计算和监控。

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



调试控制台

Debug Console

可以在调试控制台中计算表达式。要打开调试控制台，使用调试面板最上方的打开控制台动作，或者使用命令面板（`kb(workbench.action.showCommands)`）

Expressions can be evaluated in the **Debug Console**. To open the Debug Console, use the **Open Console** action at the top of the Debug pane or using the **Command Palette** (`kb(workbench.action.showCommands)`).

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

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

> |
```

调试动作

Debug actions

一旦开始一个调试会话，调试动作面板将出现在编辑器的最上方

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



- 继续 / 暂停 `kb(workbench.action.debug.continue)`
 - 单步跳过 `kb(workbench.action.debug.stepOver)`
 - 单步跳入 `kb(workbench.action.debug.stepInto)`
 - 单步跳出 `kb(workbench.action.debug.stepOut)`
 - 重新启动 `kb(workbench.action.debug.restart)`
 - 停止 `kb(workbench.action.debug.stop)`
-
- Continue / Pause `kb(workbench.action.debug.continue)`
 - Step Over `kb(workbench.action.debug.stepOver)`
 - Step Into `kb(workbench.action.debug.stepInto)`
 - Step Out `kb(workbench.action.debug.stepOut)`
 - Restart `kb(workbench.action.debug.restart)`
 - Stop `kb(workbench.action.debug.stop)`

Node调试

Node Debugging

下面的章节是特化给Node.js调试器的。

The following sections are specific to the Node.js debugger.

Node控制台

Node Console

默认情况下，Node.js调试会话运行内置的VS Code调试控制台。既然调试控制台不支持需要从控制台接收输入的程序，你可以在运行配置的属性中设置 `externalConsole` 为 `true` 来调用一个外部的，原生的控制台。

By default, Node.js debug sessions launch the target in the internal VS Code Debug Console. Since the Debug Console does not support programs that need to read input from the console, you can enable an external, native console by setting the attribute `externalConsole` to `true` in your launch configuration.

断点检查

Breakpoint Validation

为了表现的原因，Node.js懒惰的在第一次进入时解析JavaScript文件中的函数。结果是，断点不会在Node.js没有看到（解析）的源代码附近的区域生效。

For performance reasons, Node.js parses the functions inside JavaScript files lazily on first access. As a consequence, breakpoints don't work in source code areas that haven't been seen (parsed) by Node.js.

既然这种行为对调试不理想，VS Code自动给Node.js传递了`--nolazy`选项。这将防止延迟解析并且确保断点可以在运行前被确定（所以他们不会“跳跃”）

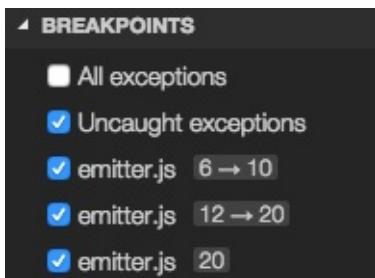
Since this behavior is not ideal for debugging, VS Code passes the `--nolazy` option to Node.js automatically. This prevents the delayed parsing and ensures that breakpoints can be validated before running the code (so they no longer "jump").

既然`--nolazy`选项可能显著增加调试目标的启动时间，你可以很容易的使用传递`--lazy`参数作为`runtimeArgs`来不使用。

Since the `--nolazy` option might increase the start-up time of the debug target significantly, you can easily opt out by passing a `--lazy` as a `runtimeArgs` attribute.

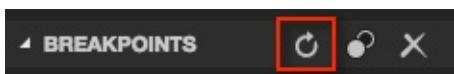
当这么做时你将发现一些你设置的断点并不“停止”在需要的行，而是“跳”到了下一个可能已经解析的代码中的行。为了避免这个疑惑，VS Code永远在Node.js认为断点在的地方展示断点。在断点区域中，这些断点在请求的行号与实际的行号之间显示为一个箭头。

When doing so you will find that some of your breakpoints don't "stick" to the line requested but instead "jump" for the next possible line in already-parsed code. To avoid confusion, VS Code always shows breakpoints at the location where Node.js thinks the breakpoint is. In the **BREAKPOINTS** section, these breakpoints are shown with an arrow between requested and actual line number:



当一个会话开始并且断点注册到Node.js时，或者一个回话已经在运行但是设置一个新的断点时发生断点检测。在这个例子中，主要的断点可能"跳"到一个不同的位置。在Node.js完成解析所有的代码(e.g. 通过运行它)后，断点可以很轻松的使用断点区域头部的重新应用被重新应用到请求的位置。这可能使断点"向后跳"到请求的位置。

This breakpoint validation occurs when a session starts and the breakpoints are registered with Node.js, or when a session is already running and a new breakpoint is set. In this case, the breakpoint may "jump" to a different location. After Node.js has parsed all the code (e.g. by running through it), breakpoints can be easily re-applied to the requested locations with the **Reapply** button in the **BREAKPOINTS** section header. This should make the breakpoints "jump back" to the requested location.



函数断点

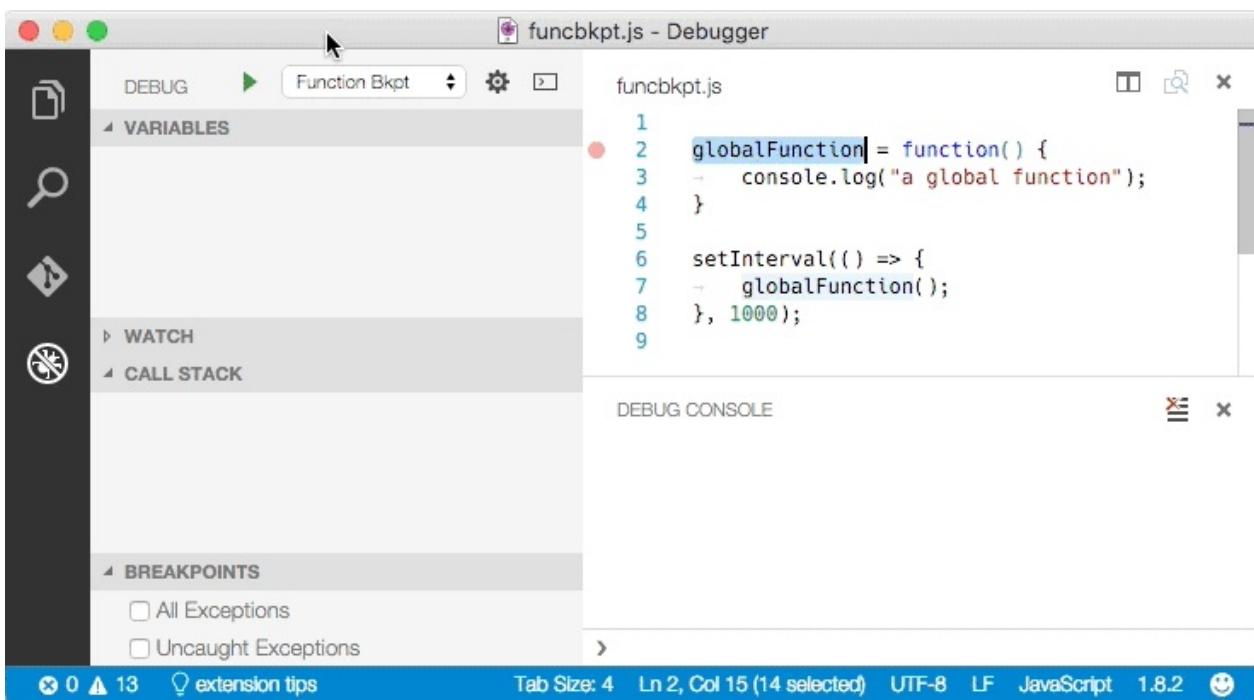
Function Breakpoints

不同于直接在源代码中设置断点，Node.js调试器现在支持使用一个特定的函数名称创建断点。这在源代码不可得但是函数名称已经知晓的情况下很有用。

Instead of placing breakpoints directly in source code, the Node.js debugger now supports creating breakpoints by specifying a function name. This is useful in situations where source is not available but a function name is known.

一个"函数断点"可以通过按断点区域头部的+来创建。

A 'function breakpoint' is created by pressing the + button in the **BREAKPOINTS** section header:



请注意：Node.js 的函数断点支持是受限的，因为：

Please note: Node.js support for function breakpoints is limited because:

- 函数断点只在全局的，非原生的函数中生效，而且
- 函数断点只能当函数被创建后才能创建(参见Node.js)。
- function breakpoints only work for global, non-native functions and
- function breakpoints can only be created if the function has been defined (seen by Node.js).

JavaScript源码映射

JavaScript Source Maps

VS Code 中的 Node.js 调试器支持用来调试编译型语言的 JavaScript 源码映射，比如 TypeScript 或者最小化/丑化的 JavaScript。有了源码映射，可以在源码中单步调试或者设置断点。如果没有源代码的源码映射，或者源码映射已经损坏并且不能成功映射源代码到生成的 JavaScript，这些断点将展现为灰色突起的圆。

The Node.js debugger of VS Code supports JavaScript Source Maps which help debugging of transpiled languages, e.g. TypeScript or minified/uglified JavaScript. With source maps, it is possible to single step through or set breakpoints in the original source. If no source map exists for the original source or if the source map is broken and cannot successfully map between the source and the generated JavaScript, the breakpoints are shown as gray hollow circles.

设置运行配置中的 `sourceMaps` 属性为 `true` 来启用源码映射的特性。有了它你可以使用 `program` 属性来指定一个源码文件(e.g. `app.ts`)。如果生成的JavaScript文件不在源文件的同一目录下而是在一个单独的目录，你可以设置 `outDir` 属性来帮助VS Code调试器确定它们的位置。当你在源代码中设置一个断点时，VS Code尝试在 `outDir` 寻找目录中生成的代码，然后联系到源码映射。

The source map feature is enabled by setting the `sourceMaps` attribute to `true` in the launch configuration. With that you can now specify a source file (e.g. `app.ts`) with the `program` attribute. If the generated (transpiled) JavaScript files do not live next to their source but in a separate directory, you can help the VS Code debugger locate them by setting the `outDir` attribute. Whenever you set a breakpoint in the original source, VS Code tries to find the generated source, and the associated source map, in the `outDir` directory.

既然源码映射不是自动创建的，你可以设置TypeScript编译器来创建它们：

Since source maps are not automatically created, you must configure the TypeScript compiler to create them:

```
tsc --sourceMap --outDir bin app.ts
```

这和TypeScript程序的运行配置一致：

This is the corresponding launch configuration for a TypeScript program:

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Launch TypeScript",
      "type": "node",
      "request": "launch",
      "program": "app.ts",
      "sourceMaps": true,
      "outDir": "bin"
    }
  ]
}
```

源码映射可以通过如下两种方法内联：

Source maps can be generated with two kinds of inlining:

- 内联源码映射：生成的JavaScript文件在最后包含作为一个URI的源码映射(而不是通过文件URI引用源码映射)
- 内联源码：生成的源码映射包含最初的源码(而不是通过路径引用源码)

- **Inlined source maps:** the generated JavaScript file contains the source map as a data URI at the end (instead of referencing the source map through a file URI).
- **Inlined source:** the source map contains the original source (instead of referencing the source through a path).

VS Code 支持内联源码映射和内联源码。

VS Code supports both the **inlined source maps** and the **inlined source**.

链接VS Code到Node.js中

Attaching VS Code to Node.js

如果你想把VS Code 调试器连接到一个Node.js程序，像下面这样运行Node.js程序：

If you want to attach the VS Code debugger to a Node.js program, launch Node.js as follows:

```
node --debug program.js
node --debug-brk program.js
```

通过 `--debug-brk` 选项, Node.js 停止在程序的第一行。

With the `--debug-brk` option Node.js stops on the first line of the program.

一致的配置看起来像下面这样：

The corresponding launch configuration looks like this:

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Attach to Node",
      "type": "node",
      "request": "attach",
      "address": "localhost",
      "port": 5858,
      "restart": false
    }
  ]
}
```

`restart` 属性控制Node.js调试器是否在调试会话结束后自动重新启动。这项特性在你使用 `nodemon` 来在文件发生改变时自动重启 Node.js 时很有用。设置运行配置属性 `restart` 为 `true` 来使 `node-debug` 在 Node.js 结束后自动尝试重新链接到 Node.js。

The `restart` attribute controls whether the Node.js debugger automatically restarts after the debug session has ended. This feature is useful if you use `nodemon` to restart Node.js on file changes. Setting the launch configuration attribute `restart` to `true` makes `node-debug` automatically try to re-attach to Node.js after Node.js has terminated.

或者在命令行中，使用 `nodemon` 来开始你的 Node.js 程序 `server.js`。

On the command line, start your Node.js program `server.js` with `nodemon`:

```
nodemon --debug server.js
```

在 VS Code 中，设置'链接'运行配置的 `restart` 属性为 `true`。

In VS Code, set the `restart` attribute to `true` in the 'attach' launch configuration.

提示：按停止按钮来停止调试会话并且断开与 Node.js 连接，但是 `nodemon`(和 Node.js) 将继续运行。要停止 `nodemon`，你将需要从命令行杀死它。

Tip: Pressing the **Stop** button stops the debug session and disconnects from Node.js, but `nodemon` (and Node.js) will continue to run. To stop `nodemon`, you will have to kill it from the command line.

提示：当有拼写错误时，`nodemon` 将不会成功启动 Node.js，直到错误被修复。在这种情况下，VS Code 将持续尝试连接到 Node.js 但是最后会放弃(在 10 秒后)。为了避免这种情况，你可以把 `time` 属性增加到更大的值(单位为毫秒)来增加超时时间。

Tip: In case of syntax errors, `nodemon` will not be able to start Node.js successfully until the error has been fixed. In this case, VS Code will continue trying to attach to Node.js but eventually give up (after 10 seconds). To avoid this, you can increase the timeout by adding a `timeout` attribute with a larger value (in milliseconds).

远程调试Node.js

Remote Debugging Node.js

Node.js 调试器支持最近版本($\geq 4.x$)的远程调试。通过 `address` 属性指定一个远程主机。

The Node.js debugger supports remote debugging for recent versions of Node.js ($\geq 4.x$). Specify a remote host via the `address` attribute.

默认情况下，VS Code将调试源码从远程Node.js文件夹流到本地的VS Code，然后在一个只读编辑器中显示它。你可以单步跳过这些代码，但是不能修改它。如果你希望VS Code从你的工作目录中打开一个可以修改的源码，你可以安装一个远程和内部位置的映射。`attach` 运行配置支持一个 `localRoot` 和一个 `remoteRoot` 属性，可以被用来映射本地VS Code工程和(远程)Node.js文件夹。这在本地基于同一个操作系统和跨越不同的操作系统时都有效。当一个文件路径需要被转换从本地VS Code路径到远程Node.js文件夹时，`remoteRoot` 路径将被去掉，并且使用 `localRoot` 替换。在相反的情况下，`localRoot` 路径被 `remoteRoot` 替换。

By default, VS Code will stream the debugged source from the remote Node.js folder to the local VS Code and show it in a read-only editor. You can step through this code, but cannot modify it. If you want VS Code to open the editable source from your workspace instead, you can setup a mapping between the remote and local locations. The `attach` launch configuration supports a `localRoot` and a `remoteRoot` attribute that can be used to map paths between a local VS Code project and a (remote) Node.js folder. This works even locally on the same system or across different operating systems. Whenever a code path needs to be converted from the remote Node.js folder to a local VS Code path, the `remoteRoot` path is stripped off the path and replaced by `localRoot`. For the reverse conversion, the `localRoot` path is replaced by the `remoteRoot`.

Mono调试

Mono Debugging

在Linux和OS X中，VS Code中的Mono调试器支持需要[Mono](#)版本3.12或更新。如果你想要使用VS Code构建.NET Core应用，我们推荐你先跟随[.NET Core and Visual Studio](#)中的步骤。

On Linux or OS X, the Mono debugging support of VS Code requires [Mono](#) version 3.12 or later. If you intend to build .NET Core applications with Visual Studio Code, we recommend you first follow the steps in [.NET Core and Visual Studio](#).

如果你只想尝试VS Code的Mono调试，你可以从[Mono project](#)下载最新的Linux和OS X的Mono，或者你可以使用你自己的包管理器。

If you just want to try VS Code Mono debugging, you can either download the latest Mono version for Linux or OS X at [Mono project](#) or you can use your package manager.

- 在OS X中: `brew install mono`
- 在Linux中: `sudo apt-get install mono-complete`
- On OS X: `brew install mono`
- On Linux: `sudo apt-get install mono-complete`

安装Mono调试器插件

Installing the Mono Debug Extension

VS Code的Mono调试集成从VS Code市场的'Mono Debug'中来

VS Code Mono debugging integration comes from the '[Mono Debug](#)' extension on the Visual Studio Marketplace.

如果你已经有了一个基于mono你可以使用VS Code的扩展：安装扩展命令

You can either install the **Mono Debug** extension with the VS Code **Extensions: Install Extension** command or if you already have a Mono based project with a `mono launch` configuration, simply by starting a debug session. VS Code will then prompt you to download and install **Mono Debug**.

启用Mono调试器

Enable Mono debugging

为了启用基于Mono的C#(和F#)程序，你需要传递 `-debug` 选项给编译器：

To enable debugging of Mono based C# (and F#) programs, you have to pass the `-debug` option to the compiler:

```
mcs -debug Program.cs
```

如果你想链接VS Code调试器到一个Mono程序，传递这些额外的参数给Mono运行时：

If you want to attach the VS Code debugger to a Mono program, pass these additional arguments to the Mono runtime:

```
mono --debug --debugger-agent=transport=dt_socket,server=y,address=127.0.0.1:55555 Program.exe
```

一致的运行配置看起来像这样：

The corresponding launch configuration looks like this:

```
{  
  "version": "0.2.0",  
  "configurations": [  
    {  
      "name": "Attach to Mono",  
      "request": "attach",  
      "type": "mono",  
      "address": "localhost",  
      "port": 55555  
    }  
  ]  
}
```

调试其他语言

Debugging Other Languages

在[VS Code extensions](#)有很多其他语言的调试器，包括Go, Powershell, Python, PHP，还有很多。

Debugging many other languages is supported by [VS Code extensions](#). These include Go, PowerShell, Python, PHP, ...

下一步

Next Steps

如果你没有读过Node.js章节，在这里阅读它：

In case you didn't already read the Node.js section, take a look at:

- [Node.js - End to end Node scenario with a sample application](#)
- [Node.js - End to end Node scenario with a sample application](#)

为了阅读更多基于Node.js调试器的教材，签出：

To see a tutorial on the basics of Node.js debugging, check out:

- [Intro Video - Debugging](#) - 介绍调试器的基础的介绍视频。
- [Intro Video - Debugging](#) - Introductory video showcasing the basics of debugging.

为了学习VS Code的任务支持，参阅：

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

- [任务](#) - 运行使用Gulp, Grunt 和 Jake的任务，显示错误和警告。
- [Tasks](#) - Running tasks with Gulp, Grunt and Jake. Showing Errors and Warnings

要编写你自己的插件，请转到：

To write your own debugger extension, visit:

- [Debuggers](#) - 从一个mock样例创建一个VS Code调试器的步骤。
- [Debuggers](#) - Steps to create a VS Code debug extension starting from a mock sample

一般问题

Common Questions

问：有哪些支持的语言脚本？

Q: What are the supported debugging scenarios?

答：Linux，OS X和Windows都支持调试基于Node.js的应用。Linux和OS X支持调试运行在Mono上的C#。.Net Core的程序是使用[Roslyn](#)编译器编译的，而不是Mono编译器。.Net Core调试器可以通过一个VS Code扩展得到。很多其他的脚本语言被[VS Code extensions](#)支持。

A: Debugging of Node.js based applications is supported on Linux, OS X, and Windows.

Debugging of C# applications running on Mono is supported on Linux and OS X. .NET Core applications are compiled using the [Roslyn](#) compiler, not the Mono compiler. .NET Core debugging will be available through a VS Code extension. Many other scenarios are supported by [VS Code extensions](#).

问：在调试视图中我没有看到任何运行配置，出错了吗？

Q: I do not see any launch configurations in the debug view drop down, what is wrong?

答：最普遍的问题是你没有设置 `launch.json` 或者 `launch.json` 中有语法错误。

A: The most common problem is that you did not set up `launch.json` yet or there is a syntax error in the `launch.json` file.

问：Node.js调试器支持哪些版本的Node.js？

Q: What Node.js version is required for Node.js debugging?

答：我们推荐最新的LTS版本的[Node.js](#)。

A: The latest LTS version of [Node.js](#) is recommended.

问：Mono调试器支持Windows吗？

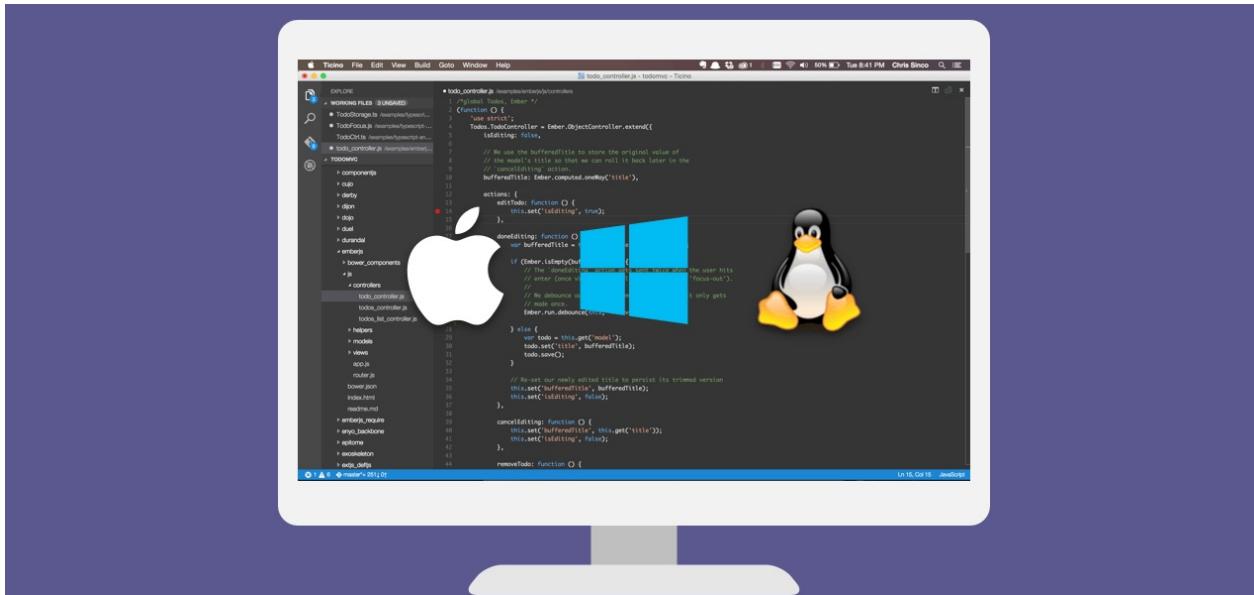
Q: Is Mono debugging supported on Windows?

答：不。现在Mono调试器只支持Mac和Linux。

A: No. Currently Mono debugging is only supported on Mac and Linux.

Why Visual Studio Code? - 为什么选用 VSCode

VSCode 提供给开发者一个新的简约开发工具的选择，它可以简化开发人员需要的 编译-构建-调试 流程。VSCode 是VS家族中第一个代码编辑器以及第一个支持OS X，Linux 和 Windows 的跨平台开发工具。



VSCode 的初衷是提供一个强大的，迅捷的源代码编辑器，并且可以每日使用。VSCode 有很多开发者在编码及编辑中需要的特性，包括导航，自定义键盘快捷键绑定，语法高亮，括弧匹配，自动缩进以及支持10多种语言的代码片段。

在大型的编程项目中，开发者经常需要投入更多精力在编码而不只是打字。VSCode 内建了不间断的智能代码补全，丰富的语义分析以及代码导航系统，同时也提供了重构功能。

VSCode 尤其对于使用 TypeScript 和 JavaScript 的 Node.js 开发提供了强大的支持，这些支持由 VSCode 的底层驱动提供。VSCode 还能为 HTML，CSS，Less，Sass，JSON 等 web 语言提供加工。VSCode 同时集成了包管理器，代码仓库和构建工具来执行一般的任务以加速每日的工作。而且 VSCode 提供了良好的 git 工作流支持，源代码 diff 功能也集成在了编辑器中。

但是开发者不只是写代码，他们还要不断调试。调试功能是 VSCode 最受欢迎的特性，而且这个特性来源于 IDE 中，它能让开发者感觉更方便。VSCode 提供了一个简约，智能的调试系统，并以对 Node.js 的调试作为例子。

在架构上，VSCode 包括了web技术，本地技术以及特定于语言的技术，并把它们最好的部分所结合。VSCode 使用 [GitHub Electron Shell](#)，既提供了快速的 web 开发技术，又提供了灵活的本地应用开发支持。Monaco，Internet Explorer 的 F12 工具 使用的是基于HTML的编辑器。VSCode 使用了一个更新、更快的版本。而且VSCode使用了工具服务架构，这允许了它

使用与Visual Studio相同的技术，包括C#的Roslyn，TypeScript，以及VS的调试引擎等。VSCode有一个公共的，可扩展的模型。所以开发者可以构建、使用扩展，并且丰富自己的开发体验。

如果你更喜欢用源代码编辑器开发或者正在构建跨平台的web或者云应用。我们诚邀您使用VSCode，并且让我们知道你的想法！

下一步

继续阅读以继续发现：

- [基础](#) - 关于VSCode的快速预览
- [更进一步](#) - 代码着色，多光标以及智能补全
- [调试](#) - 是时候用些真正好玩的了 - break, step, watch

版本控制(Version Control)

Visual Studio Code has integrated [Git](#) support for the most common commands. This makes it an excellent choice to manage your code commits while you develop.

VS Code 已经集成了 [Git](#) 并支持 [Git](#) 的大部分常用命令，这使得它是你在开发过程中，管理你的代码并提交的良好选择。

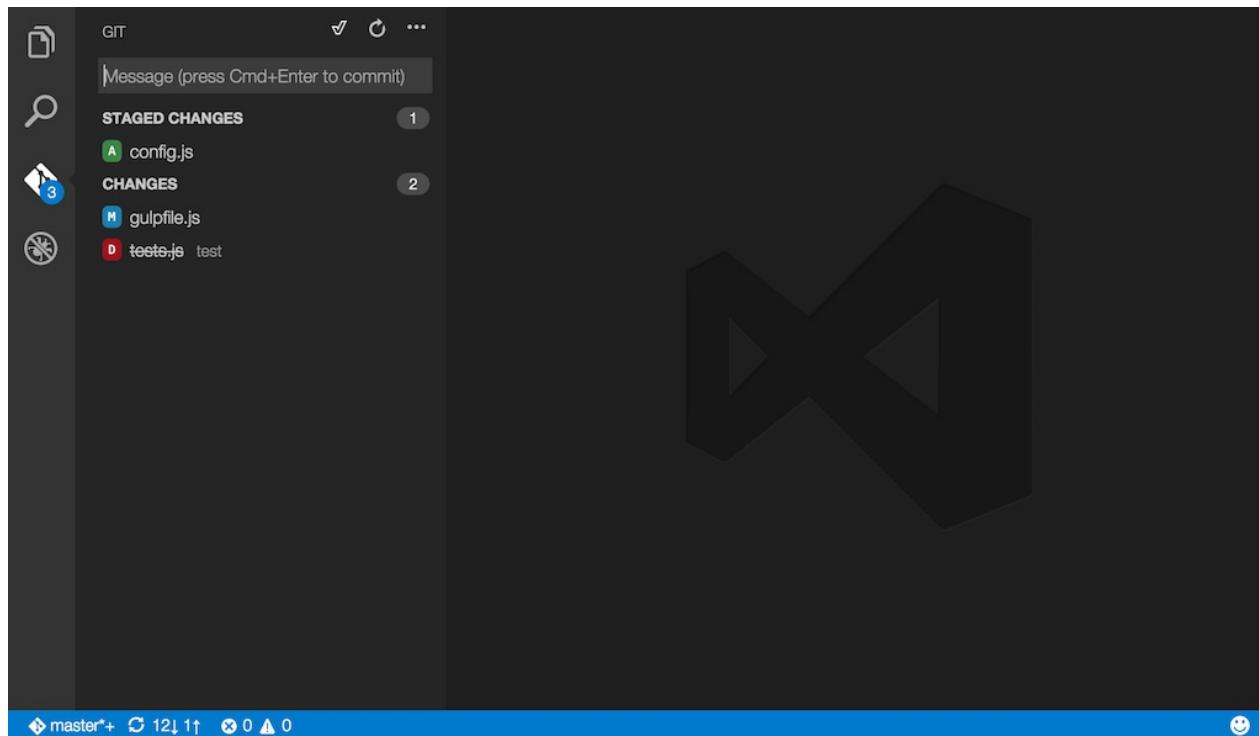
Note: VS Code will leverage your machine's Git installation, so you need to [install Git](#) first before you get these features. Make sure you install at least version [2.0.0](#).

注意： VS Code 依赖您的机器上的 Git，因此你需要先[安装 Git](#)，然后才能获得这些功能。确保你安装了 Git 2.0.0以上的版本。

Tip: VS Code will work with any Git repository. If you don't already have a private hosted Git provider, [Visual Studio Team Services](#) is a great free option. [Click here to sign-up.](#)

提示： VS Code 可适用于任何 Git 仓库。如果您还没有私人托管的 Git 仓库，[Visual Studio Team Services](#)是一个免费、不错的选择。[点击这里注册](#)。

概述(Overview)



The Git icon on the left will always indicate an **overview of how many changes** you currently have in your repository. Clicking it will show you the details of your current repository changes: **unstaged**, **staged** and **unresolved conflicting merge** changes.

左侧的 Git 图标将始终指示您当前在仓库中有多少的更改。单击它将显示当前仓库更改的详细信息：未暂存，暂存和未解决的冲突合并更改。

Clicking each item will show you in detail **the textual changes within each file**. Note that for unstaged changes, the editor on the right still lets you edit the file: feel free to use it!

单击每个文件将详细显示每个文件中的文本更改。注意，对于未暂存的更改，右侧的编辑器仍然允许您编辑文件：使用时放轻松！

You can also find indicators of the **status of your repository** in the bottom left corner of VS Code: the **current branch**, **dirty indicators** and the number of **incoming and outgoing commits** of the current branch. You can **checkout** any branch in your repository by clicking that status indicator and selecting the Git reference from the list.

您还可以在 VS Code 的左下角找到仓库的状态栏：当前分支，**dirty indicators**以及当前分支的**Git Pull**和**Git Push**提交数。您可以通过单击该状态栏并从列表中选择 Git 引用来切换到仓库中的任何分支。

Tip: You can open VS Code in a sub-directory of a Git repository. VS Code's Git services will still work as usual, showing all changes within the repository, but file changes outside of the scoped directory are shaded with a tool tip indicating they are located outside the current workspace.

提示：您可以在 Git 仓库的子目录中打开 VS Code。VS Code 的 Git 服务仍将照常工作，显示仓库中的所有更改，但在范围目录以外的文件更改用一个工具提示，来表示它们位于当前工作区之外。

Git 状态栏操作(Git Status Bar Actions)

There is a **Synchronize** action in the Status Bar, next to the branch indicator, when the current checked out branch has an upstream branch configured.

如果当前切换到的分支配置了上游分支时，在 Git 状态栏中，在分支指示符旁边有一个同步操作。



If there is no upstream branch configured and the Git repository has remotes set up, the **Publish** action is enabled. This will let you publish the current branch to a remote.

如果没有配置上游分支并且 Git 仓库已配置远程仓库，那么将启用 PUSH 操作。这将允许您将当前分支 PUSH 到远程。



提交(Commit)

Staging and **unstaging** can be done via contextual actions in the files or by drag-and-drop.

分段和拆分可以通过文件中的上下文操作或通过拖放来完成。

You can type a commit message above the changes and press `kbstyle(Ctrl+Enter)` (Mac: `kbstyle(⌘+Enter)`) to commit them. If there are any staged changes, only those will be committed, otherwise all changes will be committed.

您可以在更改上方输入 Commit Message，然后按 `kbstyle (Ctrl + Enter)` (Mac : `kbstyle (⌘ + Enter)`) 提交它们。如果有任何分阶段的更改，只有那些添加提交的将被提交，否则所有更改都将提交。

We've found this to be a great workflow. For example, in the previous screenshot, only the `config.js` file will be included in the commit. A consecutive commit action would commit both `vinyl-zip.js` and `tests.js` in a separate commit.

我们发现这是一个伟大的工作流。例如，在上一个屏幕截图中，只有 `config.js` 文件被包含在 commit 中。连续的提交操作将提交 `vinyl-zip.js` 和 `tests.js`。

More specific **commit actions** can be found in the `...` menu on the top of the Git view.

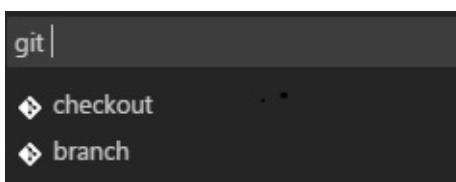
更具体的 **commit** 操作可以在 Git 视图顶部的 `...` 菜单中找到。

分支和标签(Branches and Tags)

You can create and checkout branches directly within VS code through **Quick Open**. Press `kb(workbench.action.quickopen)` , type `git` and then press `space` . You should see the following:

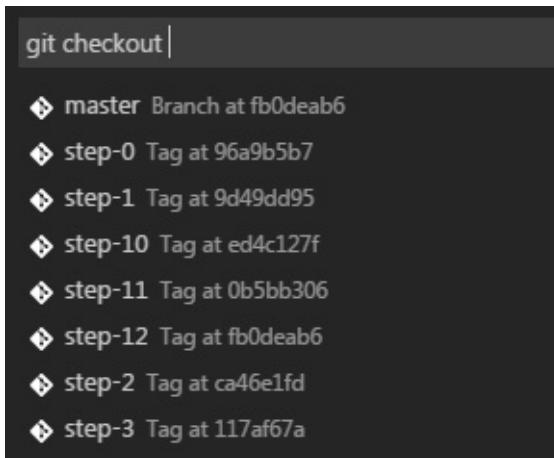
你可以通过快速打开，在 VS Code 内直接创建和切换分支。

按 `kb (workbench.action.quickopen)` ，键入 `git` ，然后按 `空格键` 。您应该看到以下内容：



If you type `checkout` and press `kbstyle(Space)` again, you will see a dropdown containing all of the branches or tags in the current repository.

如果您键入 `checkout` 并再次按 `kbstyle(Space)`，您将看到一个下拉列表，其中包含当前仓库中的所有分支或标记，如下图：



The `git branch` command lets you quickly create a new branch. Just provide the name of your new branch and VS Code will create the branch and switch to it.

使用 `git branch` 命令可以快速创建一个新的分支。你只需提供新分支的名称，VS Code 将创建分支并切换到它。

Remotes

Given that your repository is connected to some remote and that your checked out branch has an [upstream link](#) to a branch in that remote, VS Code offers you useful actions to **push**, **pull** and **sync** that branch (the latter will run a **pull** command followed by a **push** command). You can find these actions in the `...` menu.

鉴于您的仓库可能连接到远程，并且您的当前分支有一个[上游分支](#)，链接到该远程分支，VS Code 提供了方便的集成操作，**push**, **pull** 和 同步该分支（后者将运行一个PULL命令，然后 PUSH）。您可以在 `...` 菜单中找到这些操作。

Tip: You should [set up a credential helper](#) to avoid getting asked for credentials every time VS Code talks to your Git remotes. If you don't do this, you may want to consider disabling automatic fetching via the `git.autofetch` [setting](#) to reduce the number of prompts you get.

提示： 您应该设置一个[credential helper](#)，以避免每次VS Code 与您的 Git 远程仓库连接时要求输入凭据。如果不这样做，您可能需要考虑通过 `git.autofetch` 设置禁用自动提取，以减少您收到的提示数。

合并冲突(Merge Conflicts)

```

GIT      ⌂ ⌂ ...
Commit message
MERGE CHANGES  1
vinyl-zip.js \lib
STAGED CHANGES  2
config.js \
tests.js \test
CHANGES  0
vinyl-zip.js \lib
1 'use strict';
2
3 var File = require('vinyl');
4 |var utility = require('util');
5
6 function ZipFile2(file) {
7   File.call(this, file);
8 |<<<<< HEAD
9 }
10
11 util.inherits(ZipFile2, File);
12 module.exports = ZipFile2;
13 =====
14 // don't forget the symlink
15 this.symlink = file.symlink || null;
16 }
17
18 utility.inherits(ZipFile, File);
19
20 module.exports = ZipFile;
21 >>>> master
22

```

Ln 22, Col 1 CRLF JavaScript

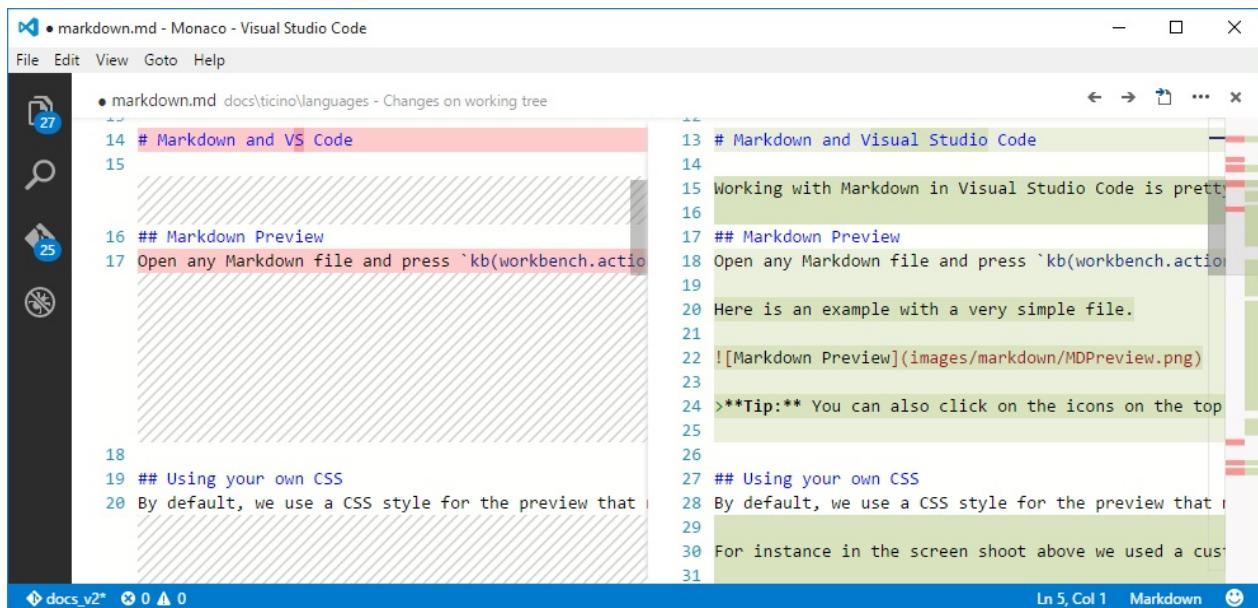
Merge conflicts are recognized by VS Code; we try to provide useful coloring markers to help you with resolving them. Once resolved, stage the conflicting file so you can commit those changes.

冲突的合并由 VS Code 识别;我们尝试提供有用的着色标记，以帮助您解决它们。一旦解决，突出冲突的文件，以便您可以提交这些更改。

查看差异(Viewing Diffs)

Our Git tooling supports viewing of diffs within VS Code.

我们的 Git 工具支持在VS Code中查看差异。



Tip: You can diff any two files by first right clicking on a file in the **WORKING FILES** list and selecting **Select for Compare** and then right-click on the second file to compare with and select **Compare with 'file_name_you_chose'**. Alternatively from the keyboard hit `kb(workbench.action.showCommands)` and select **File: Compare Active File With...** and you will be presented with a list of recent files.

提示：首先右键单击**WORKING FILES**列表中的文件并选择选择进行比较，然后右键单击要比较的第二个文件，然后选择与 '`file_name_you_chose`' 比较，即可对任何两个文件进行diff。或者从键盘按下 `kb (workbench.action.showCommands)` 并选择文件：`比较活动文件与其它`，您将看到一个最近的文件的列表。

Git输出(Git Output Window)

You can always peek under the hood to see the Git commands we are using. This is helpful if something strange is happening or if you simply get curious. :)

你可以随时从 Git 输出窗口中看到我们使用的 Git 命令。如果有错误发生时它是非常有用的，或者如果你只是好奇。 :)

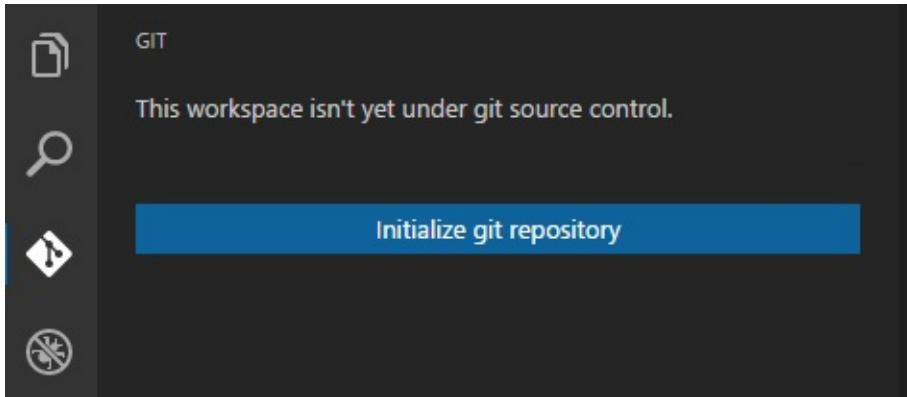
To open the Git output window, run **View > Toggle Output** and select `Git` from the dropdown.

要打开Git输出窗口，请运行**View> Toggle Output**，然后从下拉列表中选择 `Git`。

初始化仓库(Initialize a Repository)

If your workspace isn't under Git source control, you can easily create a Git repository with the **Initialise git repository** command. When VS Code doesn't detect an existing Git repository, you will see a **This workspace isn't yet under git source control.** message in the Git View and the **Initialize git repository** command will be available.

如果您的工作空间未初始化 Git 仓库，您可以使用 `Initialise git repository` 命令轻松创建 Git 仓库。当 VS Code 没有检测到当前目录存在 Git 仓库时，您将看到一个 `此工作空间尚未初始化 Git 仓库`。消息在 `Git` 视图 和 `初始化git仓库` 命令将可用。



Running **Initialize git repository** will create the necessary Git repository metadata files and show your workspace files as unstaged changes.

运行初始化`git`仓库将创建必要的Git仓库元数据文件，并将您的工作空间文件显示为未暂存的更改。

Git patch/diff 模式(Git patch/diff mode)

When you run VS Code from the command line, you can pass the `--wait` argument to make the command wait until you have closed the current VS Code instance. This can be used to configure VS Code as your Git external editor.

当从命令行运行VS Code时，您可以传递 `--wait` 参数，使命令等待，直到您关闭当前 VS Code 窗口。这可以用来配置 VS Code 作为你的 Git 外部编辑器。

Here are the steps to do so:

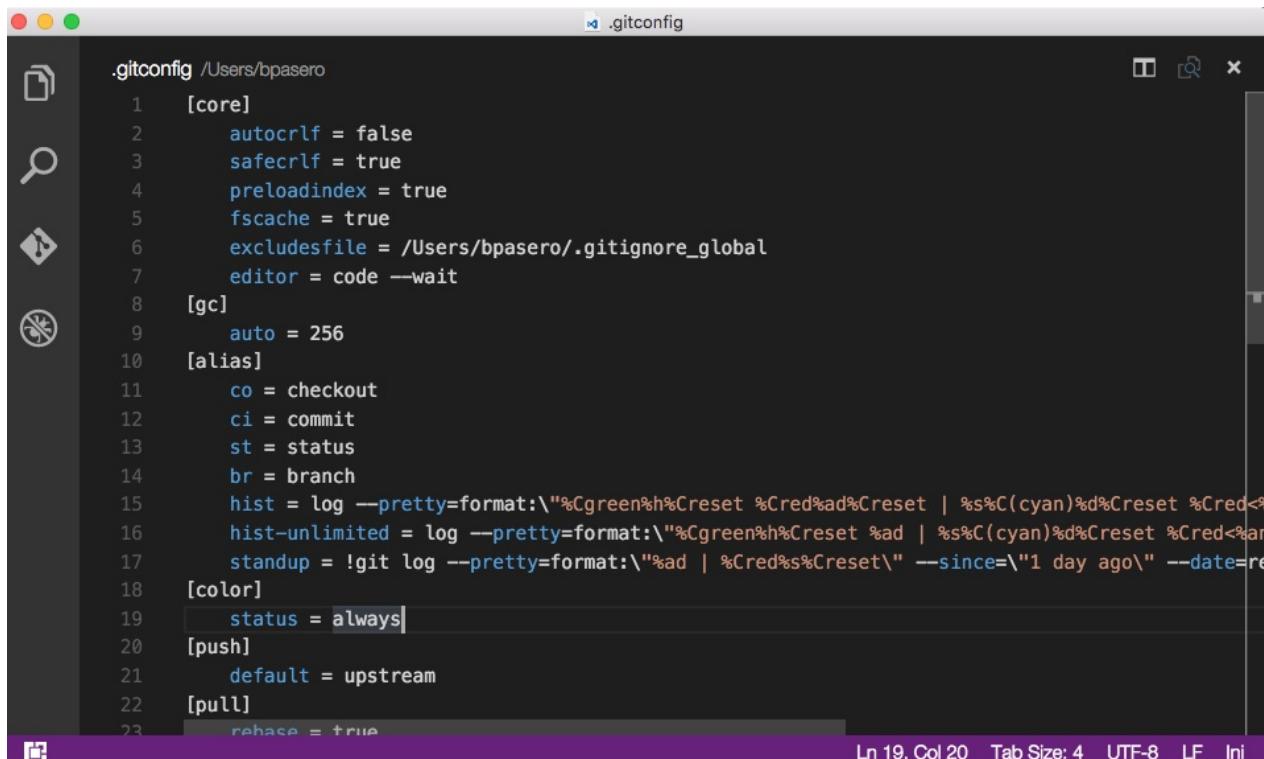
1. Make sure you can run `code --help` from the command line and you get help.
 - if you do not see help, please follow these steps:
 - Mac: Select **Shell Command: Install 'Code' command in path** from the **Command Palette**.
 - Windows: Make sure you selected **Add to PATH** during the installation.
 - Linux: Make sure you installed Code via our new .deb or .rpm packages.
2. From the command line, run `git config --global core.editor "code --wait"`

下面是这样做的一些建议：

1. 确保你可以从命令行运行 `code --help` 命令，并能得到帮助。
 - 如果你没有看见帮助，尝试以下操作：
 - Mac：选择 **Shell** 命令：在命令面板的路径中安装 '**code**' 命令。
 - Windows：确保你安装时添加了环境变量到 `PATH`。
 - Linux：确保您通过我们的新的.deb或.rpm软件包安装了VS Code。
2. 从命令行，运行 `git config --global core.editor"code --wait"`

Now you can run `git config --global -e` and use VS Code as editor for configuring Git.

现在，您可以运行 `git config --global -e` 并使用 VS Code 作为编辑器来配置 Git。



```
.gitconfig /Users/bpasero
1 [core]
2   autocrlf = false
3   safecrlf = true
4   preloadindex = true
5   fscache = true
6   excludesfile = /Users/bpasero/.gitignore_global
7   editor = code --wait
8 [gc]
9   auto = 256
10 [alias]
11   co = checkout
12   ci = commit
13   st = status
14   br = branch
15   hist = log --pretty=format:\"%Cgreen%h%Creset %Cred%ad%Creset | %s%C(cyan)%d%Creset %Cred<%an %ar%Creset%>\" --date=relative
16   hist-unlimited = log --pretty=format:\"%Cgreen%h%Creset %ad | %s%C(cyan)%d%Creset %Cred<%an %ar%Creset%>\" --date=relative
17   standup = !git log --pretty=format:\"%ad | %Cred%s%Creset\" --since=\"1 day ago\" --date=relative
18 [color]
19   status = always
20 [push]
21   default = upstream
22 [pull]
23   rebase = true
```

Add the following to your Git configurations to use VS Code as the diff tool:

将以下内容添加到您的 Git 配置中，以使用 VS Code 作为比较差异的工具：

```
[diff]
tool = default-difftool
[difftool "default-difftool"]
cmd = code --wait --diff $LOCAL $REMOTE
```

This leverages the `--diff` option you can pass to VS Code to compare 2 files side by side.

这利用 `--diff` 选项，可以传递到 VS Code，以并排比较 2 个文件。

To summarize, here are some examples of where you can use VS Code as the editor:

总而言之，下面是一些可以使用VS Code作为编辑器的示例：

- `git rebase HEAD~3 -i` do interactive rebase using VS Code
- `git commit` use VS Code for the commit message
- `git add -p` followed by `kbstyle(e)` for interactive add
- `git difftool <commit>^ <commit>` use VS Code as the diff editor for changes
- `git rebase HEAD~3 -i` do interactive rebase using VS Code
- `git commit` use VS Code for the commit message
- `git add -p` followed by `kbstyle(e)` for interactive add
- `git difftool <commit>^ <commit>` use VS Code as the diff editor for changes

下一步(Next Steps)

- [Editing Evolved](#) - Lint, IntelliSense, Lightbulbs, Peek and Goto Definition and more
- [Debugging](#) - This is where VS Code really shines
- [Tasks](#) - Running tasks with Gulp, Grunt and Jake. Showing Errors and Warnings
- [Customization](#) - Themes, settings and keyboard bindings

常见问题(Common Questions)

Q: Hey, I initialized my repo but the actions in the ... menu are all grayed out. What gives?

A: To **push, pull and sync** you need to have a Git origin set up. You can get the required URL from the repo host. Once you have that URL, you simply need to add it to the Git settings by running a couple of command line actions. For example, for Visual Studio Team Services:

```
> git remote add origin https://<AccountName>.visualstudio.com/DefaultCollection/_git/<RepoName>
> git push -u origin master
```

Q: My team is using Team Foundation version control (TFVC) instead of Git. What should I do?

A: Use the Team Foundation command line tools.

- For cross-platform use: [Cross-Platform Command-Line Client Beginner's Guide](#)
- For Windows: [Use Team Foundation version control commands](#)

Q: Why do the Pull, Push and Sync actions never finish?

This usually means there is no credential management configured in Git and you're not getting credential prompts for some reason.

You can always set up a [credential helper](#) in order to pull and push from a remote server without having VS Code prompt for your credentials each time.

Q: How can I sign into Git with my Team Services account which requires multi-factor authentication?

A: There are now [Git credential helpers](#) that assist with multi-factor authentication. You can download these from [Git Credential Manager for Mac and Linux](#) and [Git Credential Manager for Windows](#).

Q: Using Visual Studio Code, I accidentally initialized a Git repo on a folder with a massive number of files, like my entire hard drive. Now VS Code is too slow to use or hangs. What do I do?

A: First, to get VS Code running again, exit VS Code, then open a command prompt and run

```
code -n
```

which opens VS Code in a new window.

Next, assuming you want to remove the unintended repo initialization, look for the `.git` sub-folder in the large folder where you unintentionally initialized the repo, and then delete it. Note that `.git` is a hidden folder, so you might need to show hidden folders to see it. For example, at a command prompt in Windows you can run `dir .git /ah` to see hidden folders named `.git` in a specific folder. If you are not sure where you created the initial folder, run `dir .git /ah /s` at the root folder to see hidden `.git` folders, including sub-folders.

Q: I have GitHub Desktop installed on my computer but VS Code ignores it.

A: VS Code expects `git.exe` to be on the operating system's `PATH` (`$PATH` on OS X or Linux). **GitHub Desktop** installs isolated git binaries and does not automatically add `git.exe` to `PATH`.

You can either:

- Add the location of `git.exe` to `PATH` and restart VS Code.
- Set the `git.path` [setting](#) to the location of `git.exe`.

On a **GitHub Desktop** Windows installation, `git.exe` is usually under `C:\Users\USERNAME\AppData\Local\GitHub\PortableGit_COMMITID\ming32\bin`. Searching for `git.exe` under `AppData\Local\GitHub` should find the binary.

You can also install Git from [git-scm](#) and this will not interfere with **GitHub Desktop**.

Accessibility 易用性

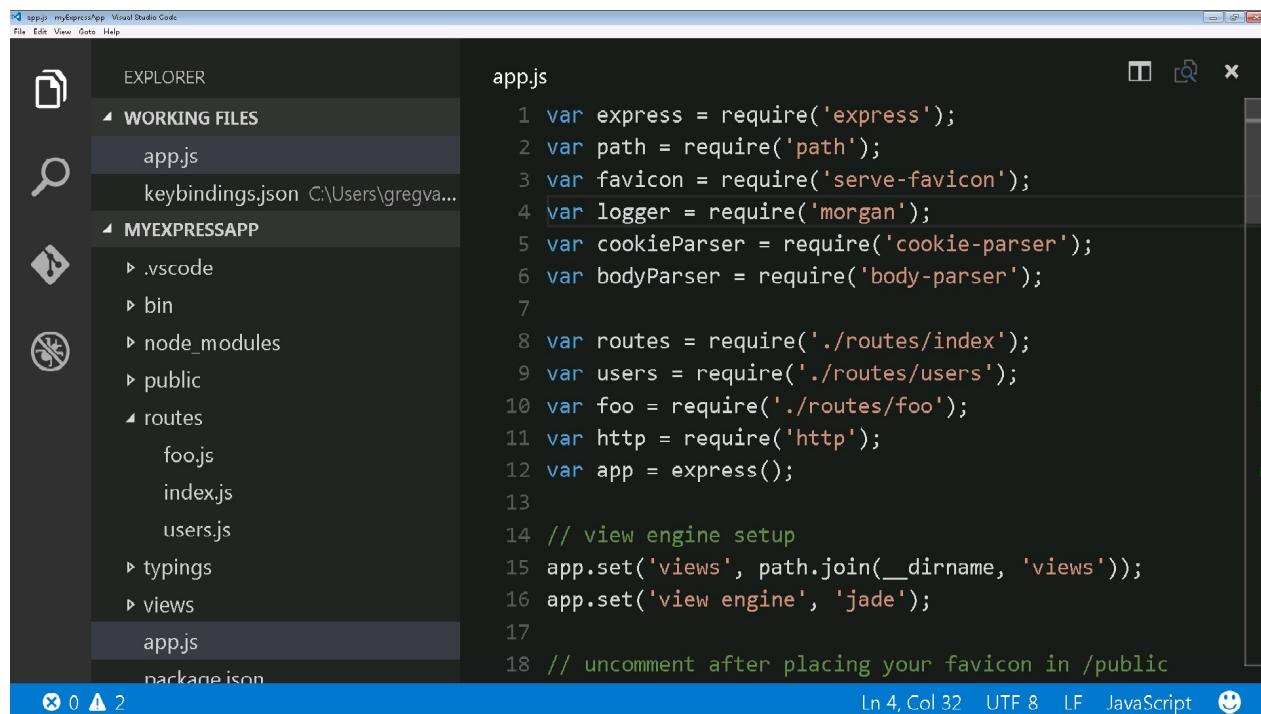
Visual Studio Code has many features to help make the editor accessible to all users. Zoom and High Contrast colors improve editor visibility, keyboard-only navigation allows use without a mouse and the editor has been optimized for screen readers.

Visual Studio Code有很多特性能够帮助用户更容易地实现编辑。缩放和高对比度色彩提升了编辑器的可视性，纯键盘导航能够让用户无需使用鼠标，同时编辑器也已经优化了屏幕阅读器。

Zoom 缩放

You can increase the Zoom level in VS Code with the **View > Zoom In** command (`kb(workbench.action.zoomIn)`). The zoom level increases by 20% each time the command is executed. The **View > Zoom Out** (`kb(workbench.action.zoomOut)`) command lets you decrease the Zoom level.

你可以在VS Code中的**View > Zoom In**命令(`kb(workbench.action.zoomIn)`)增大缩放比例。每次执行该命令，缩放比例都会增大20%。而命令**View > Zoom Out** (`kb(workbench.action.zoomOut)`)则减小缩放比例。



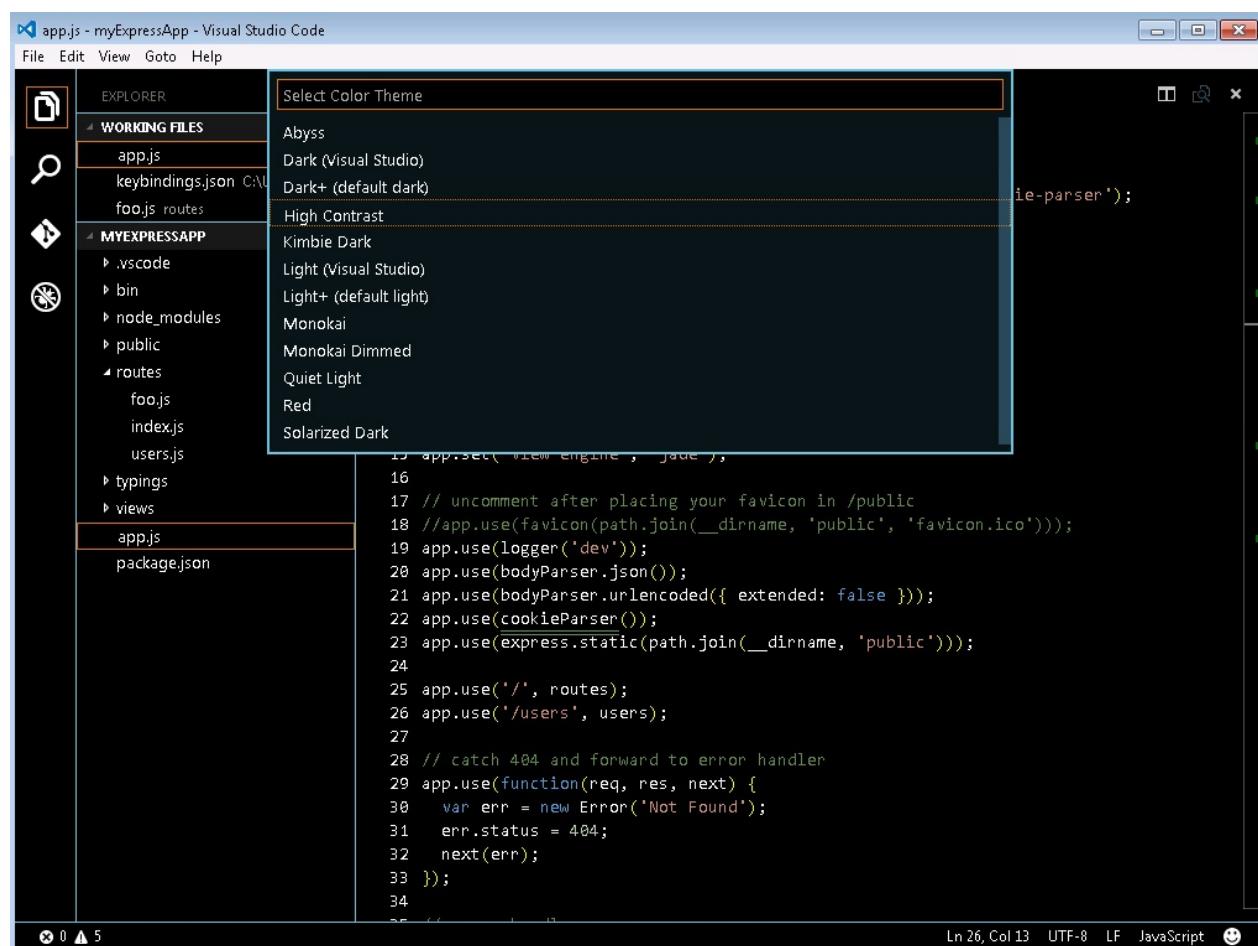
Persisted Zoom Level 保持缩放比例

With the `window.zoomLevel` setting, you can change and persist the zoom level of the window. The default value is 0 and each increment increases the zoom level by 20% similar to the effect of the **View > Zoom** command.

通过 `window.zoomLevel` setting, 你可以改变和保持窗口的缩放比例。默认值为0，每一次增值都增大20%的缩放比例，类似于**View > Zoom**命令的作用。

High Contrast Theme 高对比度主题

We support a High Contrast color theme on all platforms. Use **File > Preferences > Color Theme** to display the **Select Color Theme** dropdown and select the **High Contrast** theme.



我们支持在所有平台中的高对比度色彩主题。使用**File > Preferences > Color Theme**来显示**Select Color Theme**下拉框并选择**High Contrast**主题。

Keyboard Navigation 键盘导航

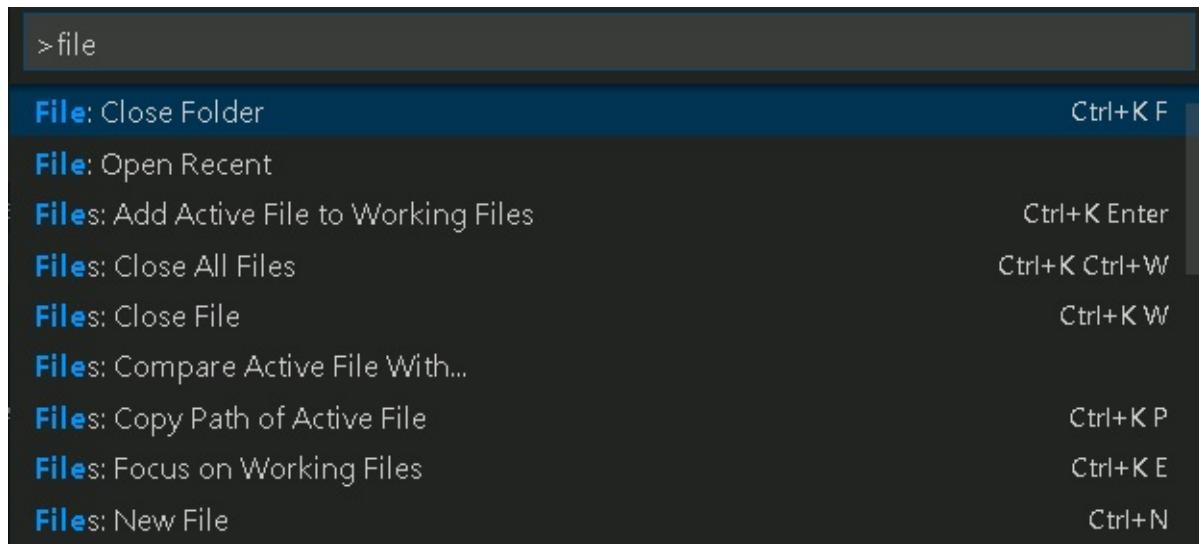
You will find that VS Code provides an exhaustive list of commands in the **Command Palette** (`kb(workbench.action.showCommands)`) so that you can run VS Code without using the mouse. Press `kb(workbench.action.showCommands)` then type a command name (e.g. 'git') to

filter the list of commands.

VS Code在**Command Palette** (`kb(workbench.action.showCommands)`)中提供了一个全面的命令列表，这样你在使用VS Code的过程中就无需使用鼠标。按下 `kb(workbench.action.showCommands)`，然后输入一个名为(e.g. 'git')的命令来筛选命令列表。

VS Code also has many preset keyboard shortcuts for commands. These are displayed to the right of the command in the **Command Palette**.

VS Code同时也可以通过许多预先设置的键盘快捷键来执行命令。这些快捷键展示的是**Command Palette**中已被授权的命令。



You can also set your own keyboard shortcuts. **File > Preferences > Keyboard Shortcuts** brings up the **Default Keyboard Shortcuts** in the left pane and your customizable `keybindings.json` on the right. See [Key Bindings](#) for more details on customizing or adding your own keyboard shortcuts.

你也可以设置自己的键盘快捷键。通过**File > Preferences > Keyboard Shortcuts**会在左边窗口打开**Default Keyboard Shortcuts**，右边窗口 `keybindings.json` 则是你的个性化设置。从[Key Bindings](#)中可以看到更多关于个性化和个人键盘快捷键设置的细节。

Tab Navigation 标签导航

You can use the `kbstyle(Tab)` key to jump between VS Code UI controls. Use `kbstyle(Shift+Tab)` to tab in reverse order. As you tab through the UI controls, you can see an indicator around the UI element once the element gains focus.

你可以使用 `kbstyle(Tab)` 键在VS Code的界面控件中切换。使用 `kbstyle(Shift+Tab)` 可以在相反的顺序切换。当你在界面控件中切换时，你所选中的某个界面元素会出现指示。

Some areas that support Tab navigation are:

- The View switcher (Files, Search, Git, Debug)
- The header of collapsible sections in a view to expand/collapse
- Actions in views and sections
- Actions for items in the tree

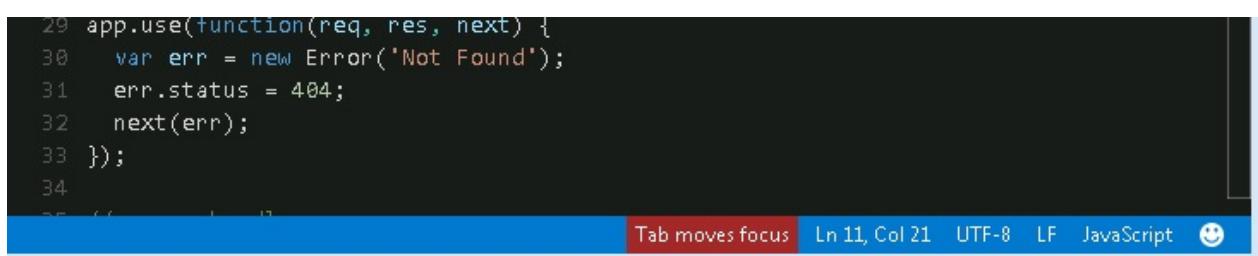
支持标签导航的领域有：

- 视图选择（文件，搜索，Git，调试）
- 可折叠部分的标题的视图扩展／折叠
- 对视图和章节的动作
- 对树中项目的动作

Tab trapping 标签锁定

By default, pressing the `kbstyle(Tab)` within a source code file inserts the Tab character (or spaces depending on your Indentation setting) and does not leave the open file. You can toggle the trapping of `kbstyle(Tab)` with `kbstyle(Ctrl+M)` and subsequent `kbstyle(Tab)` keys will move focus out of the file. When default `kbstyle(Tab)` trapping is off, you will see an indicator in the Status Bar.

默认设置中，在插入了标签特性的源代码文件中（或者是取决于你的缩进设置的空间中）按下 `kbstyle(Tab)` 键不会离开正在打开的文件。你可以切换锁定的 `kbstyle(Tab)` 和 `kbstyle(Ctrl+M)`，接着用 `kbstyle(Tab)` 键可以取消选中当前的文件。当默认的 `kbstyle(Tab)` 锁定解除，你可以在状态栏中看到相应指示。



```

29 app.use(function(req, res, next) {
30   var err = new Error('Not Found');
31   err.status = 404;
32   next(err);
33 });
34

```

The status bar at the bottom shows: Tab moves focus Ln 11, Col 21 UTF-8 LF JavaScript 😊

Read-only files never trap the `kbstyle(Tab)` key.

只读文件不能锁定 `kbstyle(Tab)` 键。

Screen Readers 屏幕阅读器

VS Code supports screen readers in the editor using a strategy based on paging the text. We have tested using the [NVDA screen reader](#), but we expect all screen readers to benefit from this support.

VS Code 支持屏幕阅读器在编辑器中使用基于文档页码的策略。我们已经使用 [NVDA screen reader](#) 进行过测试，但是我们希望所有的屏幕阅读器都能从这种支持中受益。

The **Go to Next/Previous Error or Warning** actions (`kb(editor.action.marker.next)` and `kb(editor.action.marker.prev)`) allow screen readers to announce the error or warning messages.

Go to Next/Previous Error or Warning 动作 (`kb(editor.action.marker.next)` and `kb(editor.action.marker.prev)`) 允许屏幕阅读器通报错误或者警告信息。

When the suggestions pop up, they will get announced to screen readers. It is possible to navigate the suggestions using `kbstyle(Alt+Up)` and `kbstyle(Alt+Down)`, you can dismiss the suggestions with `kbstyle(Shift+Escape)` and if suggestions get in your way, you can disable the auto-popup of suggestions with the `editor.quickSuggestions` setting.

当建议突然弹出来时，屏幕阅读器会通报给他们。你可以使用 `kbstyle(Alt+Up)` 和 `kbstyle(Shift+Escape)` 来导航该建议，以及使用 `kbstyle(Shift+Escape)` 解除建议。如果弹出的建议让你觉得困扰，你可以用 `editor.quickSuggestions` 来禁用自动弹出建议的功能。

Debugger Accessibility 调试器的可访问性

The VS Code debugger UI is user accessible and has the following feature:

- Changes in debug state are read out (e.g. 'started', 'breakpoint hit', 'terminated', ...).
- All debug actions are keyboard accessible.
- Both the Debug View and Debug Console support Tab navigation.
- Debug hover is keyboard accessible (`kb(editor.action.showHover)`).

用户可以访问 VS Code 的调试器界面，且具有以下特性：

- 读出调试模式的改变 (e.g. 'started', 'breakpoint hit', 'terminated', ...).
- 所有的调试动作都可以用键盘访问
- 调试视图和调试控制都支持标签导航；
- 调试循环支持键盘访问 (`kb(editor.action.showHover)`).

Current Known Issues 现行问题

VS Code has some known accessibility issues depending on the platform.

VS Code 在不同平台上有一些已知的问题。

Windows

You can not use the key board (right, left arrow keys) to move between top-level menu items (**File**, **Edit**, **View**, etc). This is due to Electron shell issue #2504.

不可以使用键盘（右，左方向键）来移动顶层的菜单项(**File**, **Edit**, **View**, etc). 这主要是因为选择Shell脚本的问题#2504.

OS X

There is no screen reader support for the editor.

该平台上没有支持本编辑器的屏幕阅读器。

Next Steps 下一步

Read on to find out about:

- [Visual Studio Code Basics](#) - a quick orientation of VS Code
- [Editing Evolved](#) - from code colorization & multi-cursor to IntelliSense

阅读更多资料：

- [Visual Studio Code Basics](#)- VS Code的快速定向
- [Editing Evolved](#)- 从代码色彩化&多程光标到智能感知

Common Questions 常见问题

Editing Evolved - 与时俱进的编辑体验

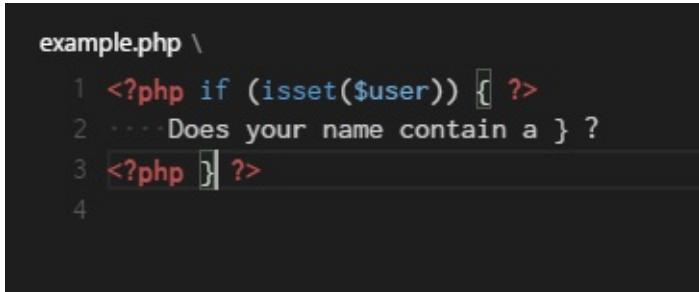
Visual Studio Code features a battle-tested code editor that has most of the industry standard features, but also has some delights. We've been using it to build VS Code and we hope you'll love it too. This topic will walk you through some of the notable features of the code editor.

Visual Studio Code 是一款经过考验的编辑器，拥有大多数已成产业标准的特点，但同时也有惊喜。我们用这些构建出的 VS Code，希望你也能爱上它。本主题将向你介绍 VS Code 的那些值得注意的特点。

Bracket matching - 括号匹配

Matching brackets will be highlighted as soon as the cursor is near one of them. The right bracket will always be found, regardless of embedded languages.

当光标接近一个括号时，与之配对的括号就会高亮显示。右括号总是会被发现，不管其嵌入的是何种语言。



```
example.php \
1 <?php if (isset($user)) { ?>
2 ...Does your name contain a } ?
3 <?php } ?>
4
```

Tip: You can jump to the matching bracket with `↑⌘\``

小技巧：你可以使用快捷键 `↑⌘\`` 直接跳转到匹配的括号处。（Windows or Linux : `ctrl+Shift+\``）

Selection & Multi-cursor - 挑选 & 动态多光标

VS Code has support for multiple cursors. You can add secondary cursors (rendered thinner) with `Alt+click`. Each cursor operates independently based on the context it sits in. The most common way to add more cursors is with `kb(editor.action.insertCursorBelow)` or `kb(editor.action.insertCursorAbove)` that insert cursors below or above.

VS Code 支持动态多光标。你可以使用 `Alt+Click` 的方式添加第二光标（被渲染为较细的光标）。每个光标根据其所在位置的上下文独立操作。最常用的多光标添加方式是使用快捷键 `⌘↓` 或 `⌘↑` 向上或下插入光标。（Window： `Ctrl+Alt+Down` 或 `Ctrl+Alt+Up`
Linux： `Shift+Alt+Down` 或 `Shift+Alt+Up`）

Note: Your graphics card provider might overwrite these default shortcuts.

注意：你的显卡提供商可能会重写这些快捷按键。

```
31 .global-message-list.transition {
32 →   -webkit-transition: top 200ms linear;
33 →   -ms-transition:      top 200ms linear;
34 →   -moz-transition:    top 200ms linear;
35 →   -khtml-transition:  top 200ms linear;
36 →   -o-transition:      top 200ms linear;
37 →   transition:         top 200ms linear;
38 }
```

`kb(editor.action.addSelectionToNextFindMatch)` selects the word at the cursor, or the next occurrence of the current selection. `kb(editor.action.moveSelectionToNextFindMatch)` moves the last added cursor to next occurrence of the current selection.

`⌘D` 选择光标处的词，或下一个选中词出现的位置。`⌘K ⌘D` 将添加一个光标到下一个选中词所出现的位置。

```
6
7 The quick brown fox jumps over the lazy dog.
8 → The quick brown fox jumps over the lazy dog.
9 →   → The quick brown fox jumps over the lazy dog.
10 →   →   → The quick brown fox jumps over the lazy dog.
11 →   →   →   → The quick brown fox jumps over the lazy dog.
12 →   →   →   →   → The quick brown fox jumps over the lazy dog.
13
14
```

Tip: You can add more cursors also with `kb(editor.action.selectHighlights)` , which will add a selection at each occurrence of the current selected text or with `kb(editor.action.changeAll)` , which will add a selection at each occurrence of the current word.

小贴士：你也可以使用 `⇧⌘L` 在每个出现选中文本处添加一个光标，或使用 `⌘F2` 在每个出现当前词的位置添加光标。（Window： `Ctrl+Shift+L` 、 `Ctrl+F2`
Linux： `Ctrl+Shift+L` 、 `Ctrl+F2`）

Column text selection - 选择一列文本

Hold `kbstyle(Shift)` and `kbstyle(Alt)` while dragging to do column text selection:

按住 Shift 或 Alt 时，用鼠标拖动可以按列选择文本：

	Character	Virtual Key Code
209 // Key	' ;	186
210 // Semicolon	' :'	186
211 // Colon	' ='	187
212 // Equals sign	' +'	187
213 // Plus	' ,'	188
214 // Comma	' <'	188
215 // Less than sign	' -'	189
216 // Minus	' _'	189
217 // Underscore	' .'	190
218 // Period	' >'	190
219 // Greater than sign	' /'	191
220 // Forward slash		
221		

There are also default key bindings for column selection on OS X and Windows, but not on Linux. You can [edit](#) your `keybindings.json` to bind them to something more familiar if you wish.

在OS X和Windows上同样有一些默认的列文本选择快捷键可以设置，很遗憾Linux没有这一功能。你可以[编辑](#)你的 `keybindings.json` 文件来绑定快捷键到你希望的组合。

For example:

例如：

```
{
  "key": "shift+alt+down",      "command": "cursorColumnSelectDown",
                                  "when": "editorTextFocus" },
  { "key": "shift+alt+left",     "command": "cursorColumnSelectLeft",
                                  "when": "editorTextFocus" },
  { "key": "shift+alt+pagedown", "command": "cursorColumnSelectPageDown",
                                  "when": "editorTextFocus" },
  { "key": "shift+alt+pageup",   "command": "cursorColumnSelectPageUp",
                                  "when": "editorTextFocus" },
  { "key": "shift+alt+right",    "command": "cursorColumnSelectRight",
                                  "when": "editorTextFocus" },
  { "key": "shift+alt+up",       "command": "cursorColumnSelectUp",
                                  "when": "editorTextFocus" }
}
```

Shrink/expand selection - 收缩、展开选择文本段

Quickly shrink or expand the current selection (applies to all languages). Trigger it with

`kb(editor.action.smartSelect.shrink)` and `kb(editor.action.smartSelect.grow)`

快速收缩或展开选择当前选中文本段（适用于所有语言），只需要按 $\text{Shift}+\text{Alt}+\text{Left}$ 或 $\text{Shift}+\text{Alt}+\text{Right}$ 即可。

（在Windows或Linux上请按 $\text{Shift}+\text{Alt}+\text{Left}$ 或 $\text{Shift}+\text{Alt}+\text{Right}$ ）

Here's an example of expanding the selection with `kb(editor.action.smartSelect.grow)` :

以下是一个展开选择的例子，使用按键 `^⌘→` (`Shift+Alt+Right`) :

```

7 namespace Hello1
8 {
9     0 references
10    class Program
11    {
12        0 references
13        static void Main(string[] args)
14        {
15            System.Console.WriteLine("Hello World!");
16        }
17    }
18 }
```

IntelliSense - 智能感知

We'll always offer word completion, but for the rich [languages](#), such as JavaScript, JSON, HTML, CSS, Less, Sass, C# and TypeScript, we offer a true IntelliSense experience.

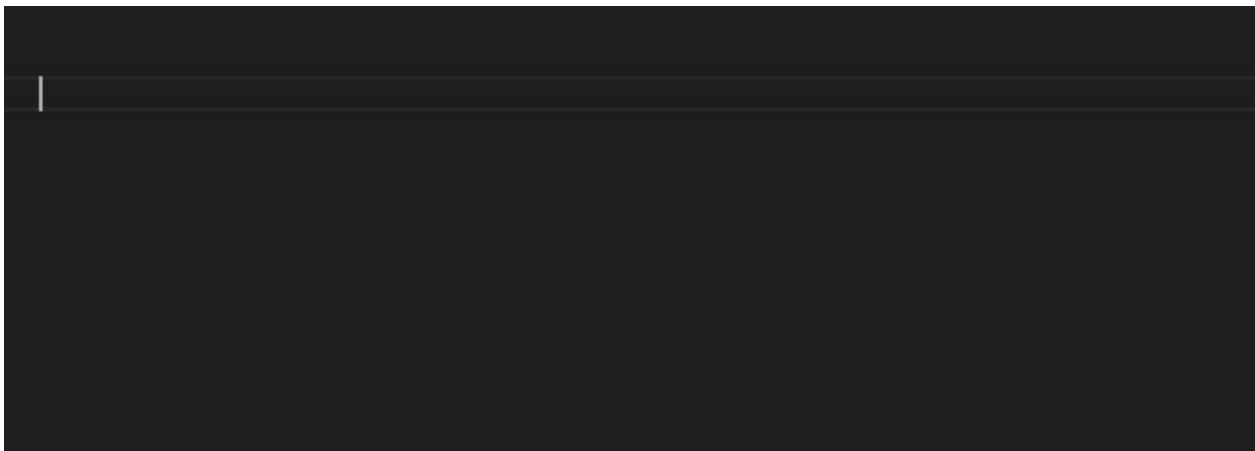
我们将会提供代码补全，但对于丰富的[语言](#)，例如JavaScript, JSON, HTML, CSS, Less, Sass, C# 和 TypeScript，我们提供一个真正的智能感知的体验。

If a language service knows possible completions, the IntelliSense suggestions will pop up as you type (we call it affectionately 24x7 IntelliSense). You can always manually trigger it with `kb(editor.action.triggerSuggest)` .

如果一个语言服务知道代码可能如何完成，智能感知代码建议将会弹出随着您的键入（我们亲切地称它为24x7智能感知）。你也可以通过用 `kb(editor.action.triggerSuggest)` 触发它。

Out of the box, `kbstyle(.)` , `kbstyle(Tab)` or `kbstyle(Enter)` are accept triggers but you can also [customize these key bindings](#).

开箱即用，`kbstyle(.)` , `kbstyle(Tab)` or `kbstyle(Enter)` 是接收快捷键的，但您也可以[自定义绑定快捷键](#)。

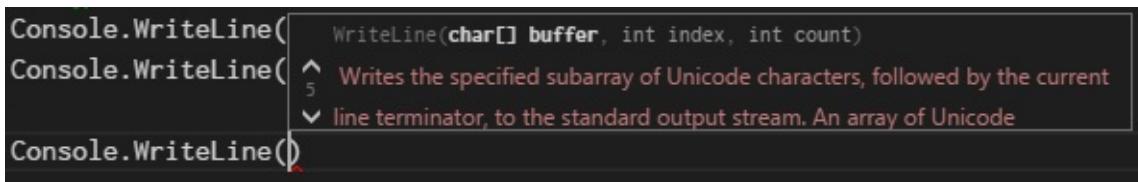


Tip: The suggestions filtering supports CamelCase so you can type the upper case letters of a method name to limit the suggestions. For example, "wl" will quickly bring up `WriteLine`.

Tip: The 24x7 IntelliSense can be configured via the `editor.quickSuggestions` and `editor.suggestOnTriggerCharacters` [settings](#).

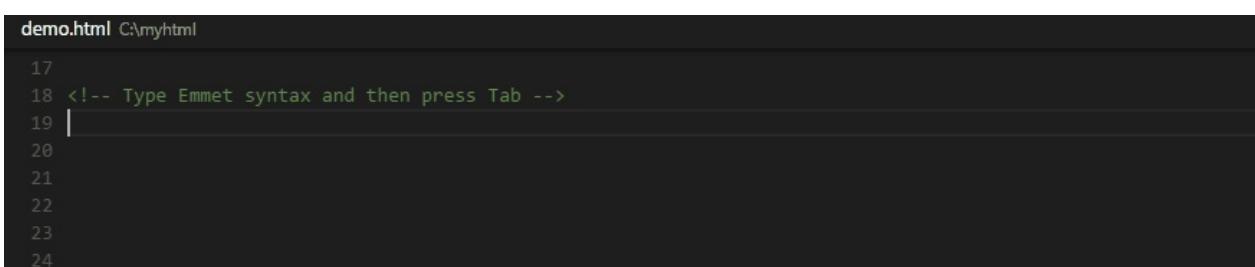
Parameter Hints

In JavaScript, TypeScript or C#, parameter hints will pop up as you're typing a method invocation. You can navigate between different overloads with `kbstyle(up)` and `kbstyle(Down)` and the best overload will be presented based on the arguments you pass in.



Snippets and Emmet Abbreviations

We offer built-in snippets across languages as well as support for [Emmet abbreviations](#). You can expand Emmet abbreviations in HTML, Razor, CSS, Less, Sass, XML or Jade with `kbstyle(Tab)` .



(See the [Emmet cheat sheet](#) for syntax examples.)

You can also define your own snippets: Open **User Snippets** under **File > Preferences** and select the language for which the snippets should appear. Find out more about this in the [customization section](#) of our docs.

Go to Definition

If a [language](#) supports it, you can go to the definition of a symbol by pressing

```
kb(editor.action.gotoDeclaration) .
```

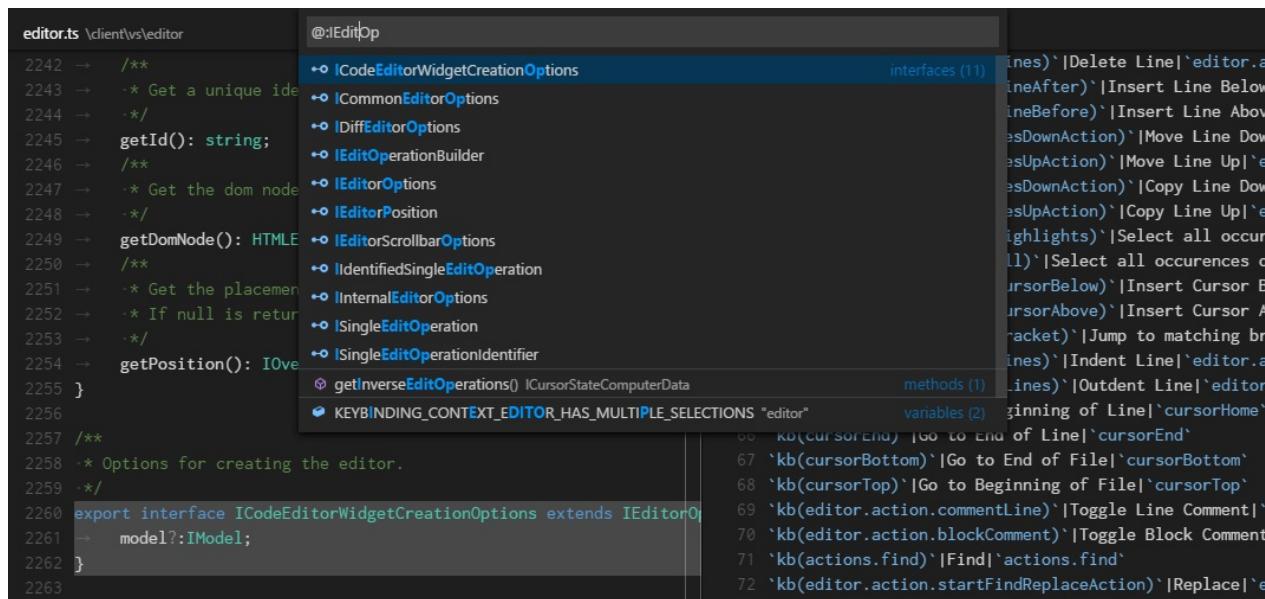
If you press `kbstyle(Ctrl)` and hover over a symbol, a preview of the declaration will appear:



Tip: You can jump to the definition with `kbstyle(Ctrl+Click)` or open the definition to the side with `kbstyle(Ctrl+Alt+Click)`. If you opened a new editor window, you can go back to the previous editor with `kb(workbench.action.focusLeftEditor)`.

Goto Symbol

You can navigate symbols inside a file with `kb(workbench.action.gotoSymbol)`. By typing `kbstyle(:)` the symbols will be grouped by category. Just press `kbstyle(Up)` or `kbstyle(Down)` and navigate to the place you want.



Open symbol by name

In C# and in TypeScript, you can jump to a symbol across files with

`kb(workbench.action.showAllSymbols)`. Just type the first letter of a type you want to navigate to, regardless of which file contains it, and press `kbstyle(Enter)`.

The screenshot shows the Visual Studio code editor with the search bar containing '#Ad'. A tooltip labeled 'symbol results' is displayed above the list of results. The results list includes various symbols such as AccountController, AccountKey, AdminConstants, AdminController, AnnouncementComponent, AzureADAccountAuthenticationHandler, AzureADAuthenticatedContext, AzureADAuthProviderCredentials, AzureApiAppConfig, AzureBlobStorageConfig, AzureMLAuthenticatedHttpClient, and AzureMLFrequentlyBoughtTogetherConfig. Below the results list, there are sections for '3 references' and '0 references'.

```

1  #Ad
2  ↗ AccountController \Controllers\AccountController.cs
3  ↗ AccountKey \WebsiteConfiguration\AzureMLFrequentlyBoughtTogetherConfig.cs
4  ↗ AccountKey \WebsiteConfiguration\AzureMLFrequentlyBoughtTogetherConfig.cs
5  ↗ AddToCart(int id) \Controllers\ShoppingCartController.cs
6  ↗ AddToCart(Product product) \Models\ShoppingCart.cs
7  ↗ AdminConstants \Areas\Admin\AdminConstants.cs
8  ↗ AdminController \Areas\Admin\Controllers\AdminController.cs
9  ↗ AnnouncementComponent \Components\AnnouncementComponent.cs
10 ↗ AzureADAccountAuthenticationHandler C:\Alex\_build\fabrikam\src\code\Fabrikam\src\Authentication...
11 ↗ AzureADAuthenticatedContext C:\Alex\_build\fabrikam\src\code\Fabrikam\src\Authentication.AzureAD\...
12 ↗ AzureADAuthProviderCredentials \Security\AzureADAuthProviderCredentials.cs
13 ↗ AzureApiAppConfig \WebsiteConfiguration\AzureApiAppConfig.cs
14 ↗ AzureBlobStorageConfig \WebsiteConfiguration\AzureBlobStorageConfig.cs
15 ↗ AzureMLAuthenticatedHttpClient \Recommendations\AzureMLAuthenticatedHttpClient.cs
16 ↗ AzureMLFrequentlyBoughtTogetherConfig \WebsiteConfiguration\AzureMLFrequentlyBoughtTogether...
17
18
19
  
```

3 references

```

15     public virtual Product Product { get; set; }
16
17     0 references
18     public virtual Order Order { get; set; }
19   
```

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 to fold and unfold regions. The folding regions are 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.

You can also use the following actions:

- Fold (`kb(editor.fold)`) folds the innermost uncollapsed region at the cursor
- Unfold (`kb(editor.unfold)`) unfolds the collapsed region at the cursor
- Fold All (`kb(editor.foldAll)`) folds all region in the editor
- Unfold All (`kb(editor.unfoldAll)`) unfolds all regions in the editor
- Fold Level X (`kb(editor.foldLevel2)` for level 2) folds all regions of level X, except the region at the current cursor position

```

364     private onCursorSelectionChanged(): void {
365         if (this.state === State.Hidden) {
366             return;
367         }
368         this.editor.layoutContentWidget(this);
369     }
370
371     private onEditorBlur(): void {
372     }
373
374     private onListSelection(e: ISelectionChangeEvent<CompletionItem>): void {
375         if (!e.elements.length) {
376             return;
377         }
378     }

```

Gutter indicators

If you open a folder that is a Git repository and begin making changes, VS Code will add useful annotations to the gutter and to the overview ruler.

- A red triangle indicates where lines have been deleted
- A green bar indicates new added lines
- A blue bar indicates modified lines

```

Program.cs \
1 using System;
2
3 // This is a new line
4 class Program
5 {
6     public static void Main()
7     {
8         var x = 123;
9         Console.WriteLine();
10        Console.WriteLine("hello world!");
11    }
12 }
13

```

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 Reference Search (via `kb(editor.action.referenceSearch.trigger)`), or a Peek Definition (via

`kb(editor.action.previewDeclaration)), we embed the result inline:`

```
1 using Microsoft.AspNet.Mvc.Rendering;
2 using System.Collections.Generic;
3 using System.ComponentModel.DataAnnotations;
4
5 namespace PartsUnlimited.Models
6 {
7     public class ExternalLoginConfirmationViewModel
```

370
377
378
379 ant, then prompt the user to create an account
380
381 provider;
382 ot in claims?
383 indFirstValue(ClaimTypes.Email);
384 on", new ExternalLoginConfirmationViewModel { Email = email }
385
386
387
388
389
390
391
392

8 {
9 [Required]
10 [Display(Name = "Email")]
11 public string Email { get; set; }

Tip: You can navigate between different references in the peeked editor and, if you need to, you can even make quick edits right there!

Tip: Clicking on the peeked editor filename or double-clicking in the result list will open the reference in the outer editor.

Hover

For languages that support it, the hover will show useful information, such as types of symbols, or, in the case of CSS below, the shape of the HTML that would match a certain CSS rule:

```
.FishMenuItem
{
    font-family: Opificio;
    color: #00a3ef;
<element class="header">
    ...
<element class="LoadingCover">
    ...
    <p>
.header .LoadingCover p
{
    margin-top: 300px;
    text-align: center;
    color: #4312ff;
    font-size: 16pt;
}
```

Reference information

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

```

OrderDetail.cs \Models
1 namespace PartsUnlimited.Models
2 {
3     11 references
4     public class OrderDetail
5     {
6         1 reference
7         public int OrderDetailId { get; set; }
8
9         3 references
10        public int OrderId { get; set; }
11
12        4 references
13        public int ProductId { get; set; }
14
15        7 references
16        public int Quantity { get; set; }
17
18        4 references
19        public decimal UnitPrice { get; set; }
20
21        3 references
22        public virtual Product Product { get; set; }
23
24        0 references
25        public virtual Order Order { get; set; }
26    }
27}

```

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

Tip: Reference information can be turned on or off through the `editor.referenceInfos` setting.

Rename symbol

TypeScript and C# support rename symbol across files. Simply press

`kb(editor.action.rename)` and then type the new desired name and press `kbstyle(Enter)`.

All usages of the symbol will be renamed, across files.

```

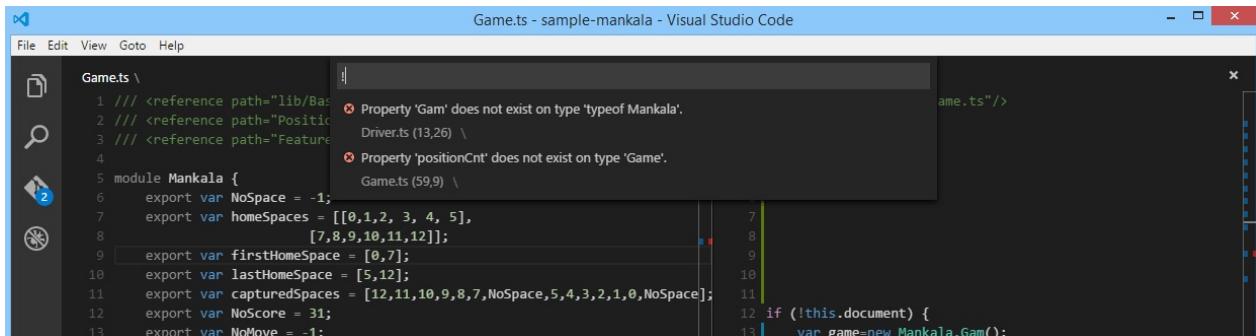
3 references
public int OrderId { get; set; }
    OrderIdentifier
4 references
public int ProductId { get; set; }

```

Errors & Warnings

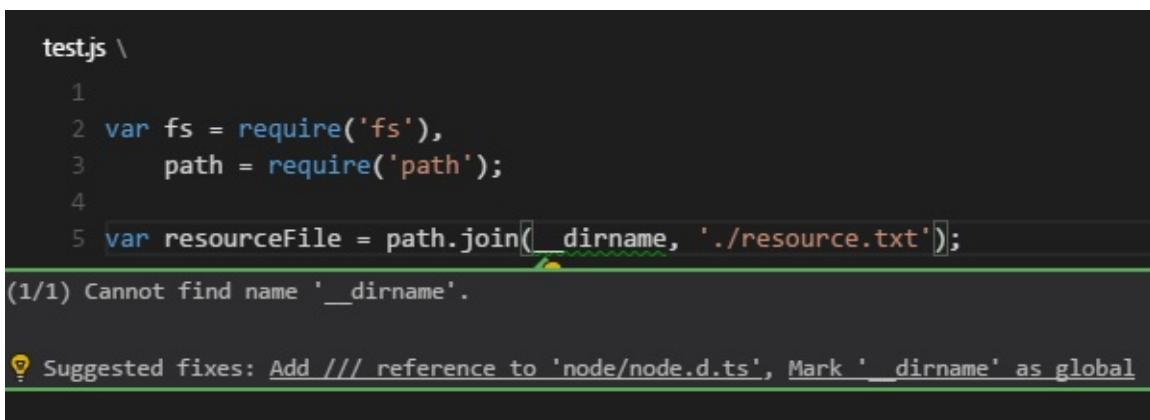
Warnings or Errors can be generated either via [configured tasks](#) or by the rich language services, that constantly analyze your code in the background. Since we love bug-free code, warnings and errors show up in multiple places:

- In the status line there is a summary of all errors and warnings counts.
- You can click on the summary or press `kb(workbench.action.showErrorsWarnings)` to see 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.



Tip: To loop through errors or warnings in the current file, you can press

`kb(editor.action.marker.next)` or `kb(editor.action.marker.prev)` which will show an inline zone detailing the problem and possible code actions (if available):



Next Steps

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

- [Why VS Code](#) - Why we exist and where we think we can help
- [The Basics](#) - Basic orientation around VS Code
- [Debugging](#) - This is where VS Code really shines
- [Customization](#) - Configure VS Code the way you want - Themes, Settings

定制化

- 概述
- 用户和工作空间
- 快捷键绑定
- 用户定义代码段
- 调色板
- 主题
- 语言区域

定制化Visual Studio Code(Customize Visual Studio Code)

You can customize VS Code to work the way you like to work. Here is a quick primer on some of the most common ways to configure VS Code. Navigate to the relevant sections to learn more.

您可以定制化 VS Code 来用您喜欢的方式工作。这里是一些最常见的配置VS Code方法的快速入门。跳转到到相关部分了解详情。

Tip: Several categories of customizations (Themes, Snippets, Language Support) can be shared in the VS Code [Extension Marketplace](#). It's always a good idea to look there first.

提示：可以在[VS Code扩展市场](#)中发布自己的定制主题（主题，代码段，语言支持等）。这是个好习惯，先看看那里。

定制主题(Customization Topics)

Category	Scenario	Marketplace
User and Workspace settings	Configure settings for an individual workspace or all workspaces. Word-wrapping, linting options and much more.	No
Key Bindings	Review all key bindings and change them to suit your needs.	No
Tasks	Tasks are a great way to connect VS Code with your broader development workflow.	No
Themes	Add additional color themes to VS Code.	Yes
Basic Language Support	Add additional basic language support (colorization and bracket matching) to VS Code via a TextMate bundle. You can also associate more file extensions with an existing language.	Yes
Snippets	Add additional snippets to your favorite language	Yes
Language	Configure the display language	No

类别	场景	扩展市场
用户和工作区设置	配置单个工作区或所有工作区的设置。 Word-wrapping, linting options and much more.	不支持
快捷键绑定	查看所有快捷键绑定，并根据您的需要更改它们。	不支持
主题	向VS Code添加其他颜色主题。	支持
基本语言支持	通过TextMate包添加额外的基本语言支持（代码高亮和括号匹配）到VS Code。您还可以将更多文件扩展名与现有语言相关联。	支持
用户定义代码段	在您喜欢的语言中添加其他代码段	支持
语言	配置显示语言	不支持

下一步(Next Steps)

Here are a few pointers to help you on your way...

下面几个要点可能在某些方面帮助你...

- [Extension Marketplace](#) - browse the extensions others have shared
- [Yo Code](#) - to generate a customization, then install it locally
- [Publishing Tool](#) - use the vsce publishing tool to share your customization with others
- 扩展程序市场 - 浏览其他人共享的扩展程序
- [Yo Code](#) - 生成自定义，然后在本地安装
- [发布工具](#) - 使用vsce发布工具与他人共享您的自定义

Common Questions

Q: How can I make my customization get loaded into VS Code on start-up?

A: If you move a copy of your customization into your `.vscode/extensions` folder it will be loaded up as VS Code is started.

Q: What are the valid fields in the `project.json` (extension Manifest) file?

A: We have extended the `package.json` to include the required fields for customisation and extension loading/distribution. We have an overview of the optional and mandatory sections of the [extension manifest](#) available.

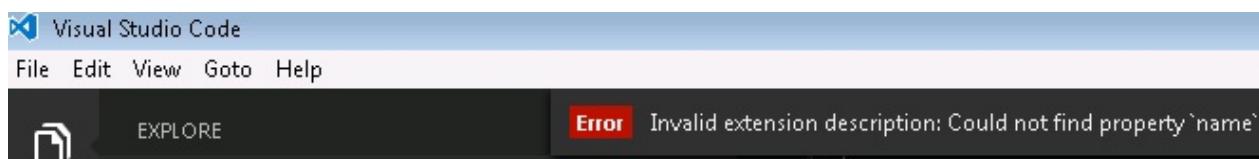
Q: Can a `package.json` contribute more than one customization?

A: Yes, the `contributes` attribute can take a comma delimited list of contribution types and, as you'll notice from the json above, each contribution type takes an array (e.g. `"themes": []`).

Q: I created a new customization but I don't see it displayed in VS Code?

A: Make sure you have copied all of the generator's output files to a new folder under [your `.vscode/extensions` folder](#) (e.g. `".vscode/extensions/cooltheme"`) and that you have restarted VS Code after installing the customization.

On startup, if VS Code detects an issue with a customization, you will see an error message which can aid in debugging your customization.



用户和工作区设置(User and Workspace Settings)

It's easy to configure VS Code the way you want by editing the various setting files where you will find a great number of settings to play with.

很容易通过编辑各种设置文件来配置VS Code，你会发现许多有意思的设计。

VS Code provides two different scopes for settings:

VS Code 的设置分为两种作用域：

- **User** these settings apply globally to any instance of VS Code you open
 - **Workspace** these settings are stored inside your workspace in a `.vscode` folder and only apply when the workspace is opened. Settings defined on this scope overwrite the user scope.
- 用户设置 这些设置全局应用于您打开的任何VS Code 项目
- 工作区设置 这些设置存储在工作区内的 `.vscode` 文件夹中，并且仅在打开的工作区适用。在此范围内定义的设置将覆盖用户范围的设置。

创建用户和工作区设置(Creating User and Workspace Settings)

The menu under **File > Preferences** provides entries to configure user and workspace settings. You are provided with a list of Default Settings. Copy any setting that you want to change to the related `settings.json` file.

文件>首选项 下的 菜单 提供用于配置用户和工作空间设置的选项。您将看到一个默认设置列表。将要更改的任何设置复制到相关的（右侧） `settings.json` 文件。

In the example below, we disabled line numbers in the editor and configured line wrapping to wrap automatically based on the size of the editor.

如下面的示例，我们在编辑器中禁用了 行号，并且配置换行为根据编辑器的大小自动换行。

```

Default Settings
// Overwrite settings by placing them into your settings
file.
{
    //----- Editor configuration -----
    // Controls the font family.
    "editor.fontFamily": "",

    // Controls the font size.
    "editor.fontSize": 0,
}

settings.json
// Place your settings in this file to overwrite the
default settings
{
    "editor.lineNumbers": false,
    "editor.wrappingColumn": 0
}

```

Changes to settings are reloaded by VS Code after the modified `settings.json` file is saved.

保存修改的 `settings.json` 文件后，VS Code 将重新加载对设置的更改。

设置文件的位置(Settings File Locations)

Depending on your platform, the user settings file is located here:

根据您的操作系统，用户设置文件位于下面几处：

- **Windows** `%APPDATA%\Code\User\settings.json`
- **Mac** `$HOME/Library/Application Support/Code/User/settings.json`
- **Linux** `$HOME/.config/Code/User/settings.json`

The workspace setting file is located under the `.vscode` folder in your project.

工作区设置文件位于项目中的 `.vscode` 文件夹下。

设置文件内容(Settings File Sections)

The `settings.json` file is divided into these sections:

`settings.json` 文件分为以下几个部分：

- **Editor Configuration** - font, word wrapping, tab size, line numbers, indentation, ...
- **Window Configuration** - restore folders, zoom level, ...
- **Files Configuration** - excluded file filters, default encoding, trim trailing whitespace, ...
- **File Explorer Configuration** - encoding, **WORKING FILES** behavior, ...
- **HTTP Configuration** - proxy settings
- **Search Configuration** - file exclude filters
- **Git Configuration** - disable Git integration, auto fetch behavior
- **Telemetry Configuration** - disable telemetry reporting, crash reporting
- **HTML Configuration** - HTML format configuration
- **CSS Configuration** - CSS linting configuration

- **JavaScript Configuration** - Language specific settings
- **JSON Configuration** - Schemas associated with certain JSON files
- **Markdown Preview Configuration** - Add a custom CSS to the Markdown preview
- **Less Configuration** - Control linting for Less
- **Sass Configuration** - Control linting for Sass
- **TypeScript Configuration** - Language specific settings
- **PHP Configuration** - PHP linter configuration

默认设置(Default Settings)

Below is a copy of the default `settings.json` file.

下面是默认的 `settings.json` 文件的实例。

Tip: While in the `settings.json` file, press `kb(workbench.action.gotoSymbol)` to see an outline of all available settings and navigate through the file.

提示：在 `settings.json` 文件中，按 `kb (workbench.action.gotoSymbol)` 可查看所有可用设置的大纲，并浏览该文件。

```
// Overwrite settings by placing them into your settings file.
{
  //----- Editor configuration -----
  // Controls the font family.
  "editor.fontFamily": "",

  // Controls the font size.
  "editor.fontSize": 0,

  // Controls the line height.
  "editor.lineHeight": 0,

  // Controls visibility of line numbers
  "editor.lineNumbers": true,

  // Controls visibility of the glyph margin
  "editor.glyphMargin": false,

  // Columns at which to show vertical rulers
  "editor.rulers": [],

  // Characters that will be used as word separators when doing word related navigations or operations
  "editor.wordSeparators": "`~!@#$%^&*()=-+[{}]\\";:'\\",.<>/?",

  // The number of spaces a tab is equal to.
}
```

```
"editor.tabSize": 4,  
  
    // Insert spaces when pressing Tab.  
    "editor.insertSpaces": true,  
  
    // When opening a file, `editor.tabSize` and `editor.insertSpaces` will be detected  
    // based on the file contents.  
    "editor.detectIndentation": true,  
  
    // Controls if selections have rounded corners  
    "editor.roundedSelection": true,  
  
    // Controls if the editor will scroll beyond the last line  
    "editor.scrollBeyondLastLine": true,  
  
    // Controls after how many characters the editor will wrap to the next line. Setting  
    // this to 0 turns on viewport width wrapping  
    "editor.wrappingColumn": 300,  
  
    // Controls the indentation of wrapped lines. Can be one of 'none', 'same' or 'indent'.  
    "editor.wrappingIndent": "same",  
  
    // A multiplier to be used on the `deltaX` and `deltaY` of mouse wheel scroll events  
    "editor.mouseWheelScrollSensitivity": 1,  
  
    // Controls if quick suggestions should show up or not while typing  
    "editor.quickSuggestions": true,  
  
    // Controls the delay in ms after which quick suggestions will show up  
    "editor.quickSuggestionsDelay": 10,  
  
    // Controls if the editor should automatically close brackets after opening them  
    "editor.autoClosingBrackets": true,  
  
    // Controls if the editor should automatically format the line after typing  
    "editor.formatOnType": false,  
  
    // Controls if suggestions should automatically show up when typing trigger characters  
    "editor.suggestOnTriggerCharacters": true,  
  
    // Controls if suggestions should be accepted 'Enter' - in addition to 'Tab'. Helps  
    // to avoid ambiguity between inserting new lines or accepting suggestions.  
    "editor.acceptSuggestionOnEnter": true,  
  
    // Controls whether the editor should highlight similar matches to the selection  
    "editor.selectionHighlight": true,  
  
    // Controls the number of decorations that can show up at the same position in the  
    // overview ruler  
    "editor.overviewRulerLanes": 3,
```

```
// Controls the cursor blinking animation, accepted values are 'blink', 'visible',  
and 'hidden'  
"editor.cursorBlinking": "blink",  
  
// Controls the cursor style, accepted values are 'block' and 'line'  
"editor.cursorStyle": "line",  
  
// Enables font ligatures  
"editor.fontLigatures": false,  
  
// Controls if the cursor should be hidden in the overview ruler.  
"editor.hideCursorInOverviewRuler": false,  
  
// Controls whether the editor should render whitespace characters  
"editor.renderWhitespace": false,  
  
// Controls if the editor shows reference information for the modes that support it  
"editor.referenceInfos": true,  
  
// Controls whether the editor has code folding enabled  
"editor.folding": true,  
  
// Controls if the diff editor shows the diff side by side or inline  
"diffEditor.renderSideBySide": true,  
  
// Controls if the diff editor shows changes in leading or trailing whitespace as  
diffs  
"diffEditor.ignoreTrimWhitespace": true,  
  
//----- Window configuration -----  
  
// When enabled, will open files in a new window instead of reusing an existing instance.  
"window.openFilesInNewWindow": true,  
  
// Controls how folders are being reopened after a restart. Select 'none' to never  
reopen a folder, 'one' to reopen the last folder you worked on or 'all' to reopen all  
folders of your last session.  
"window.reopenFolders": "one",  
  
// Adjust the zoom level of the window. The original size is 0 and each increment  
above (e.g. 1) or below (e.g. -1) represents zooming 20% larger or smaller. You can al  
so enter decimals to adjust the zoom level with a finer granularity.  
"window.zoomLevel": 0,  
  
//----- Files configuration -----  
  
// Configure glob patterns for excluding files and folders.  
"files.exclude": {
```

```

    "**/.git": true,
    "**/.DS_Store": true
  },

  // Configure file associations to languages (e.g. "*.extension": "html"). These have precedence over the default associations of the languages installed.
  "files.associations": {},

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

  // The default end of line character.
  "files.eol": "\n",

  // When enabled, will trim trailing whitespace when you save a file.
  "files.trimTrailingWhitespace": false,

  // Controls auto save of dirty files. Accepted values: "off", "afterDelay", "onFocusChange". If set to "afterDelay" you can configure the delay in "files.autoSaveDelay".
  "files.autoSave": "off",

  // Controls the delay in ms after which a dirty file is saved automatically. Only applies when "files.autoSave" is set to "afterDelay"
  "files.autoSaveDelay": 1000,

  // Configure glob patterns of file paths to exclude from file watching. Changing this setting requires a restart. When you experience Code consuming lots of cpu time on startup, you can exclude large folders to reduce the initial load.
  "files.watcherExclude": {
    "**/.git/objects/**": true,
    "**/node_modules/**": true
  },

  //----- File Explorer configuration -----

  // Maximum number of working files to show before scrollbars appear.
  "explorer.workingFiles.maxVisible": 9,

  // Controls if the height of the working files section should adapt dynamically to the number of elements or not.
  "explorer.workingFiles.dynamicHeight": true,

  //----- HTTP configuration -----

  // The proxy setting to use. If not set will be taken from the http_proxy and https_proxy environment variables
  "http.proxy": "",

  // Whether the proxy server certificate should be verified against the list of supplied CAs.

```

```
"http.proxyStrictSSL": true,  
  
//----- Update configuration -----  
  
// Configure the update channel to receive updates from. Requires a restart after  
change.  
"update.channel": "default",  
  
//----- Search configuration -----  
  
// Configure glob patterns for excluding files and folders in searches. Inherits a  
ll glob patterns from the file.exclude setting.  
"search.exclude": {  
    "**/node_modules": true,  
    "**/bower_components": true  
},  
  
//----- Git configuration -----  
  
// Is git enabled  
"git.enabled": true,  
  
// Path to the git executable  
"git.path": null,  
  
// Whether auto fetching is enabled.  
"git.autofetch": true,  
  
//----- Telemetry configuration -----  
  
// Enable usage data and errors to be sent to Microsoft.  
"telemetry.enableTelemetry": true,  
  
//----- Markdown preview configuration -----  
  
// A list of URLs or local paths to CSS style sheets to use from the markdown prev  
iew.  
"markdown.styles": [],  
  
//----- JSON configuration -----  
  
// Associate schemas to JSON files in the current project  
"json.schemas": [],  
  
//----- HTML configuration -----
```

```
// Maximum amount of characters per line (0 = disable).
"html.format.wrapLineLength": 120,

// List of tags, comma separated, that shouldn't be reformatted. 'null' defaults to all inline tags.
"html.format.unformatted": null,

// Indent <head> and <body> sections.
"html.format.indentInnerHTML": false,

// Whether existing line breaks before elements should be preserved. Only works before elements, not inside tags or for text.
"html.format.preserveNewLines": true,

// Maximum number of line breaks to be preserved in one chunk. Use 'null' for unlimited.
"html.format.maxPreserveNewLines": null,

// Format and indent {{#foo}} and {{/foo}}.
"html.format.indentHandlebars": false,

// End with a newline.
"html.format.endWithNewline": false,

// List of tags, comma separated, that should have an extra newline before them. 'null' defaults to "head, body, /html".
"html.format.extraLiners": null,

//----- Telemetry configuration -----
// Enable crash reports to be sent to Microsoft.
// This option requires restart to take effect.
"telemetry.enableCrashReporter": true,

//----- CSS configuration -----
// Controls CSS validation and problem severities.

// Enables or disables all validations
"css.validate": true,

// When using a vendor-specific prefix make sure to also include all other vendor-specific properties
"css.lint.compatibleVendorPrefixes": "ignore",

// When using a vendor-specific prefix also include the standard property
"css.lint.vendorPrefix": "warning",

// Do not use duplicate style definitions
"css.lint.duplicateProperties": "ignore",
```

```
// Do not use empty rulesets
"css.lint.emptyRules": "warning",

// Import statements do not load in parallel
"css.lint.importStatement": "ignore",

// Do not use width or height when using padding or border
"css.lint.boxModel": "ignore",

// The universal selector (*) is known to be slow
"css.lint.universalSelector": "ignore",

// No unit for zero needed
"css.lint.zeroUnits": "ignore",

// @font-face rule must define 'src' and 'font-family' properties
"css.lint.fontFaceProperties": "warning",

// Hex colors must consist of three or six hex numbers
"css.lint.hexColorLength": "error",

// Invalid number of parameters
"css.lint.argumentsInColorFunction": "error",

// Unknown property.
"css.lint.unknownProperties": "warning",

// IE hacks are only necessary when supporting IE7 and older
"css.lint.ieHack": "ignore",

// Unknown vendor specific property.
"css.lint.unknownVendorSpecificProperties": "ignore",

// Property is ignored due to the display. E.g. with 'display: inline', the width,
height, margin-top, margin-bottom, and float properties have no effect
"css.lint.propertyIgnoredDueToDisplay": "warning",

// Avoid using !important. It is an indication that the specificity of the entire
CSS has gotten out of control and needs to be refactored.
"css.lint.important": "ignore",

// Avoid using 'float'. Floats lead to fragile CSS that is easy to break if one as-
pect of the layout changes.
"css.lint.float": "ignore",

// Selectors should not contain IDs because these rules are too tightly coupled wi-
th the HTML.
"css.lint.idSelector": "ignore",

//----- LESS configuration -----

// Controls LESS validation and problem severities.
```

```
// Enables or disables all validations
"less.validate": true,

// When using a vendor-specific prefix make sure to also include all other vendor-
specific properties
"less.lint.compatibleVendorPrefixes": "ignore",

// When using a vendor-specific prefix also include the standard property
"less.lint.vendorPrefix": "warning",

// Do not use duplicate style definitions
"less.lint.duplicateProperties": "ignore",

// Do not use empty rulesets
"less.lint.emptyRules": "warning",

// Import statements do not load in parallel
"less.lint.importStatement": "ignore",

// Do not use width or height when using padding or border
"less.lint.boxModel": "ignore",

// The universal selector (*) is known to be slow
"less.lint.universalSelector": "ignore",

// No unit for zero needed
"less.lint.zeroUnits": "ignore",

// @font-face rule must define 'src' and 'font-family' properties
"less.lint.fontFaceProperties": "warning",

// Hex colors must consist of three or six hex numbers
"less.lint.hexColorLength": "error",

// Invalid number of parameters
"less.lint.argumentsInColorFunction": "error",

// Unknown property.
"less.lint.unknownProperties": "warning",

// IE hacks are only necessary when supporting IE7 and older
"less.lint.ieHack": "ignore",

// Unknown vendor specific property.
"less.lint.unknownVendorSpecificProperties": "ignore",

// Property is ignored due to the display. E.g. with 'display: inline', the width,
height, margin-top, margin-bottom, and float properties have no effect
"less.lint.propertyIgnoredDueToDisplay": "warning",

// Avoid using !important. It is an indication that the specificity of the entire
CSS has gotten out of control and needs to be refactored.
```

```
"less.lint.important": "ignore",

// Avoid using 'float'. Floats lead to fragile CSS that is easy to break if one aspect of the layout changes.
"less.lint.float": "ignore",

// Selectors should not contain IDs because these rules are too tightly coupled with the HTML.
"less.lint.idSelector": "ignore",

//----- Sass configuration -----

// Controls Sass validation and problem severities.

// Enables or disables all validations
"sass.validate": true,

// When using a vendor-specific prefix make sure to also include all other vendor-specific properties
"sass.lint.compatibleVendorPrefixes": "ignore",

// When using a vendor-specific prefix also include the standard property
"sass.lint.vendorPrefix": "warning",

// Do not use duplicate style definitions
"sass.lint.duplicateProperties": "ignore",

// Do not use empty rulesets
"sass.lint.emptyRules": "warning",

// Import statements do not load in parallel
"sass.lint.importStatement": "ignore",

// Do not use width or height when using padding or border
"sass.lint.boxModel": "ignore",

// The universal selector (*) is known to be slow
"sass.lint.universalSelector": "ignore",

// No unit for zero needed
"sass.lint.zeroUnits": "ignore",

// @font-face rule must define 'src' and 'font-family' properties
"sass.lint.fontFaceProperties": "warning",

// Hex colors must consist of three or six hex numbers
"sass.lint.hexColorLength": "error",

// Invalid number of parameters
"sass.lint.argumentsInColorFunction": "error",

// Unknown property.
```

```
"sass.lint.unknownProperties": "warning",

// IE hacks are only necessary when supporting IE7 and older
"sass.lint.ieHack": "ignore",

// Unknown vendor specific property.
"sass.lint.unknownVendorSpecificProperties": "ignore",

// Property is ignored due to the display. E.g. with 'display: inline', the width,
height, margin-top, margin-bottom, and float properties have no effect
"sass.lint.propertyIgnoredDueToDisplay": "warning",

// Avoid using !important. It is an indication that the specificity of the entire
CSS has gotten out of control and needs to be refactored.
"sass.lint.important": "ignore",

// Avoid using 'float'. Floats lead to fragile CSS that is easy to break if one aspect
of the layout changes.
"sass.lint.float": "ignore",

// Selectors should not contain IDs because these rules are too tightly coupled with
the HTML.
"sass.lint.idSelector": "ignore",

//----- TypeScript configuration -----//

// Specifies the folder path containing the tsserver and lib*.d.ts files to use.
"typescript.tsdk": null,

// Complete functions with their parameter signature.
"typescript.useCodeSnippetsOnMethodSuggest": false,

// Enable / disable TypeScript validation
"typescript.validate.enable": true,

// Defines space handling after a comma delimiter
"typescript.format.insertSpaceAfterCommaDelimiter": true,

// Defines space handling after a semicolon in a for statement
"typescript.format.insertSpaceAfterSemicolonInForStatements": true,

// Defines space handling after a binary operator
"typescript.format.insertSpaceBeforeAndAfterBinaryOperators": true,

// Defines space handling after keywords in control flow statement
"typescript.format.insertSpaceAfterKeywordsInControlFlowStatements": true,

// Defines space handling after function keyword for anonymous functions
"typescript.format.insertSpaceAfterFunctionKeywordForAnonymousFunctions": true,

// Defines space handling after opening and before closing non empty parenthesis
"typescript.format.insertSpaceAfterOpeningAndBeforeClosingNonemptyParenthesis": fa
```

```
lse,  
  
    // Defines space handling after opening and before closing non empty brackets  
    "typescript.format.insertSpaceAfterOpeningAndBeforeClosingNonemptyBrackets": false  
,  
  
    // Defines whether an open brace is put onto a new line for functions or not  
    "typescript.format.placeOpenBraceOnNewLineForFunctions": false,  
  
    // Defines whether an open brace is put onto a new line for control blocks or not  
    "typescript.format.placeOpenBraceOnNewLineForControlBlocks": false,  
  
    // Enable / disable JavaScript validation  
    "javascript.validate.enable": true,  
  
    // Defines space handling after a comma delimiter  
    "javascript.format.insertSpaceAfterCommaDelimiter": true,  
  
    // Defines space handling after a semicolon in a for statement  
    "javascript.format.insertSpaceAfterSemicolonInForStatements": true,  
  
    // Defines space handling after a binary operator  
    "javascript.format.insertSpaceBeforeAndAfterBinaryOperators": true,  
  
    // Defines space handling after keywords in control flow statement  
    "javascript.format.insertSpaceAfterKeywordsInControlFlowStatements": true,  
  
    // Defines space handling after function keyword for anonymous functions  
    "javascript.format.insertSpaceAfterFunctionKeywordForAnonymousFunctions": true,  
  
    // Defines space handling after opening and before closing non empty parenthesis  
    "javascript.format.insertSpaceAfterOpeningAndBeforeClosingNonemptyParens": false,  
,  
  
    // Defines space handling after opening and before closing non empty brackets  
    "javascript.format.insertSpaceAfterOpeningAndBeforeClosingNonemptyBrackets": false  
,  
  
    // Defines whether an open brace is put onto a new line for functions or not  
    "javascript.format.placeOpenBraceOnNewLineForFunctions": false,  
  
    // Defines whether an open brace is put onto a new line for control blocks or not  
    "javascript.format.placeOpenBraceOnNewLineForControlBlocks": false,  
  
    //----- PHP Configuration options -----  
  
    // Whether php validation is enabled or not.  
    "php.validate.enable": true,  
  
    // Points to the php executable.  
    "php.validate.executablePath": null,
```

```
// Whether the linter is run on save or on type.  
"php.validate.run": "onSave"  
}
```

Common Questions

Q: When does it make sense to use workspace settings?

A: If you're using a workspace that needs custom settings but you don't want to apply them to your other VS Code projects. A good example is language-specific linting rules.

VS Code 的快捷按键(Key Bindings for Visual Studio Code)

Visual Studio Code lets you perform most tasks directly from the keyboard. This page lists out the default bindings and describes how you can update them.

Visual Studio Code允许您直接从键盘执行大多数任务。此页面列出了默认绑定，并介绍了如何更新它们。

Note: If you visit this page on a Mac, you will see the key bindings for the Mac. If you visit using Windows or Linux, you will see the keys for that OS. If you need the key binding for another OS, hover your mouse over the key you are interested in.

注意：如果您在Mac上访问此页面，您将看到Mac的键绑定。如果使用Windows或Linux访问，您将看到该操作系统的密钥。如果您需要另一个操作系统的键绑定，将鼠标悬停在您感兴趣的键上。

Note: The following keys are rendered assuming a standard US keyboard layout. If you use a different keyboard layout, please [read below](#).

注意：以下键是使用标准美式键盘布局呈现的。如果您使用不同的键盘布局，请阅读[这一部分](#)。

基本编辑(Basic Editing)

Key	Command	Comm
kb(editor.action.clipboardCutAction)	Cut line (empty selection)	editor.action.clipbo
kb(editor.action.clipboardCopyAction)	Copy line (empty selection)	editor.action.clipbo
kb(editor.action.deleteLines)	Delete Line	editor.action.deleteL
kb(editor.action.insertLineAfter)	Insert Line Below	editor.action.insertL
kb(editor.action.insertLineBefore)	Insert Line Above	editor.action.insertL
kb(editor.action.moveLinesDownAction)	Move Line Down	editor.action.moveLin

kb(editor.action.moveLinesUpAction)	Move Line Up	editor.action.moveLineUp
kb(editor.action.copyLinesDownAction)	Copy Line Down	editor.action.copyLineDown
kb(editor.action.copyLinesUpAction)	Copy Line Up	editor.action.copyLineUp
kb(editor.action.addSelectionToNextFindMatch)	Add Selection To Next Find Match	editor.action.addSelectionToNextFindMatch
kb(editor.action.moveSelectionToNextFindMatch)	Move Last Selection To Next Find Match	editor.action.moveSelectionToNextFindMatch
kb(cursorUndo)	Undo last cursor operation	cursorUndo
kb(editor.action.selectHighlights)	Select all occurrences of current selection	editor.action.selectAllHighlights
kb(editor.action.changeAll)	Select all occurrences of current word	editor.action.changeAll
kb(expandLineSelection)	Select current line	expandLineSelection
kb(editor.action.insertCursorBelow)	Insert Cursor Below	editor.action.insertCursorBelow
kb(editor.action.insertCursorAbove)	Insert Cursor Above	editor.action.insertCursorAbove
kb(editor.action.jumpToBracket)	Jump to matching bracket	editor.action.jumpToBracket
kb(editor.action.indentLines)	Indent Line	editor.action.indentLines
kb(editor.action.outdentLines)	Outdent Line	editor.action.outdentLines
kb(cursorHome)	Go to Beginning of Line	cursorHome
kb(cursorEnd)	Go to End	cursorEnd

kb(cursorEnd)	of Line	cursorEnd
kb(cursorBottom)	Go to End of File	cursorBottom
kb(cursorTop)	Go to Beginning of File	cursorTop
kb(scrollLineDown)	Scroll Line Down	scrollLineDown
kb(scrollLineUp)	Scroll Line Up	scrollLineUp
kb(scrollPageDown)	Scroll Page Down	scrollPageDown
kb(scrollPageUp)	Scroll Page Up	scrollPageUp
kb(editor.fold)	Fold (collapse) region	editor.fold
kb(editor.unfold)	Unfold (uncollapse) region	editor.unfold
kb(editor.foldAll)	Fold (collapse) all regions	editor.foldAll
kb(editor.unfoldAll)	Unfold (uncollapse) all regions	editor.unfoldAll
kb(editor.action.addCommentLine)	Add Line Comment	editor.action.addComm
kb(editor.action.removeCommentLine)	Remove Line Comment	editor.action.removeC
kb(editor.action.commentLine)	Toggle Line Comment	editor.action.comment
kb(editor.action.blockComment)	Toggle Block Comment	editor.action.blockCo
kb(actions.find)	Find	actions.find
kb(editor.action.startFindReplaceAction)	Replace	editor.action.startFi
kb(editor.action.nextMatchFindAction)	Find Next	editor.action.nextMat
kb(editor.action.previousMatchFindAction)	Find	editor.action.previous

	Previous	
kb(toggleFindCaseSensitive)	Toggle Find Case Sensitive	toggleFindCaseSensitive
kb(toggleFindRegex)	Toggle Find Regex	toggleFindRegex
kb(toggleFindWholeWord)	Toggle Find Whole Word	toggleFindWholeWord
kb(editor.action.toggleTabFocusMode)	Toggle Use of Tab Key for Setting Focus	editor.action.toggleTabFocusMode
kb(toggleRenderWhitespace)	Toggle Render Whitespace	toggleRenderWhitespace

更加丰富的语言编辑(Rich Languages Editing)

Key	Command	Comm
kb(editor.action.triggerSuggest)	Trigger Suggest	editor.action.trigger
kb(editor.action.triggerParameterHints)	Trigger Parameter Hints	editor.action.trigger
kb(editor.action.format)	Format Code	editor.action.format
kb(editor.action.goToDeclaration)	Go to Definition	editor.action.goToDec
kb(editor.action.previewDeclaration)	Peek Definition	editor.action.preview
kb(editor.action.openDeclarationToTheSide)	Open Definition to the Side	editor.action.openDec
kb(editor.action.quickFix)	Quick Fix	editor.action.quickFi
kb(editor.action.referenceSearch.trigger)	Show References	editor.action.referen
kb(editor.action.rename)	Rename Symbol	editor.action.rename
kb(editor.action.inPlaceReplace.down)	Replace with Next Value	editor.action.inPlace
kb(editor.action.inPlaceReplace.up)	Replace with Previous Value	editor.action.inPlace
kb(editor.action.smartSelect.grow)	Expand AST Select	editor.action.smartSe
kb(editor.action.smartSelect.shrink)	Shrink AST Select	editor.action.smartSe
kb(editor.action.trimTrailingWhitespace)	Trim Trailing Whitespace	editor.action.trimTra
kb(workbench.action.editor.changeLanguageMode)	Change Language Mode	workbench.action.edit

导航(Navigation)

Key	Command	Command id
kb(workbench.action.showAllSymbols)	Show All Symbols	workbench.action.showAllSymbols
kb(workbench.action.gotoLine)	Go to Line...	workbench.action.gotoLine
kb(workbench.action.quickOpen)	Go to File..., Quick Open	workbench.action.quickOpen
kb(workbench.action.gotoSymbol)	Go to Symbol...	workbench.action.gotoSymbol
kb(workbench.action.showErrorsWarnings)	Show Errors and Warnings	workbench.action.showErrorsWarnings
kb(editor.action.marker.next)	Go to Next Error or Warning	editor.action.marker.next
kb(editor.action.marker.prev)	Go to Previous Error or Warning	editor.action.marker.prev
kb(workbench.action.showCommands)	Show All Commands	workbench.action.showCommands
kb(workbench.action.openPreviousEditor)	Navigate History	workbench.action.openPreviousEditor
kb(workbench.action.navigateBack)	Go Back	workbench.action.navigateBack
kb(workbench.action.navigateForward)	Go Forward	workbench.action.navigateForward

编辑器/窗口管理(Editor/Window Management)

Key	Command	Command id
kb(workbench.action.newWindow)	New Window	workbench.action.newWindow
kb(workbench.action.closeWindow)	Close Window	workbench.action.closeWindow
kb(workbench.action.closeActiveEditor)	Close Editor	workbench.action.closeActiveEditor
kb(workbench.action.closeFolder)	Close Folder	workbench.action.closeFolder
kb(workbench.action.cycleEditor)	Cycle Between Opened Editors	workbench.action.cycleEditor
kb(workbench.action.splitEditor)	Split Editor	workbench.action.splitEditor
kb(workbench.action.focusFirstEditor)	Focus into Left Hand Editor	workbench.action.focusFirstEditor
kb(workbench.action.focusSecondEditor)	Focus into Side Editor	workbench.action.focusSecondEditor
kb(workbench.action.focusThirdEditor)	Focus into Right Hand Editor	workbench.action.focusThirdEditor
kb(workbench.action.focusLeftEditor)	Focus into Next Editor on the Left	workbench.action.focusLeftEditor
kb(workbench.action.focusRightEditor)	Focus into Next Editor on the Right	workbench.action.focusRightEditor
kb(workbench.action.moveActiveEditorLeft)	Move Active Editor Left	workbench.action.moveActiveEditorLeft
kb(workbench.action.moveActiveEditorRight)	Move Active Editor Right	workbench.action.moveActiveEditorRight

文件管理(File Management)

Key	Command	
kb(workbench.action.files.newUntitledFile)	New File	workbench.action.files.newUntitledFile
kb(workbench.action.files.openFile)	Open File...	workbench.action.files.openFile
kb(workbench.action.files.save)	Save	workbench.action.files.save
kb(workbench.action.files.saveAll)	Save All	workbench.action.files.saveAll
kb(workbench.action.files.saveAs)	Save As...	workbench.action.files.saveAs
kb(workbench.files.action.closeFile)	Close File	workbench.files.action.closeFile
kb(workbench.files.action.closeAllFiles)	Close All Files	workbench.files.action.closeAllFiles
kb(workbench.files.action.closeOtherFiles)	Close Other Files	workbench.files.action.closeOtherFiles
kb(workbench.files.action.addToWorkingFiles)	Add to Working Files	workbench.files.action.addToWorkingFiles
kb(workbench.files.action.openNextWorkingFile)	Open Next Working File	workbench.files.action.openNextWorkingFile
kb(workbench.files.action.openPreviousWorkingFile)	Open Previous Working File	workbench.files.action.openPreviousWorkingFile
kb(workbench.action.files.copyPathOfFile)	Copy Path of Active File	workbench.action.files.copyPathOfFile
kb(workbench.action.files.revealActiveFileInWindows)	Reveal Active File in Windows	workbench.action.files.revealActiveFileInWindows
kb(workbench.action.files.showOpenedFileInNewWindow)	Show Opened File in New Window	workbench.action.files.showOpenedFileInNewWindow
kb(workbench.files.action.compareFileWith)	Compare Opened File With	workbench.files.action.compareFileWith

Display

Key	Command	
kb(workbench.action.toggleFullScreen)	Toggle Full Screen	workbench.action.
kb(workbench.action.zoomIn)	Zoom in	workbench.action.
kb(workbench.action.zoomOut)	Zoom out	workbench.action.
kb(workbench.action.toggleSidebarVisibility)	Toggle Sidebar Visibility	workbench.action.
kb(workbench.view.debug)	Show Debug	workbench.view.de
kb(workbench.view.explorer)	Show Explorer	workbench.view.ex
kb(workbench.view.git)	Show Git	workbench.view.gi
kb(workbench.view.search)	Show Search	workbench.view.se
kb(workbench.action.search.toggleQueryDetails)	Toggle Search Details	workbench.action.
kb(workbench.action.terminal.openNativeConsole)	Open New Command Prompt	workbench.action.
kb(workbench.action.output.toggleOutput)	Show Output	workbench.action.
kb(workbench.action.markdown.togglePreview)	Toggle Markdown Preview	workbench.action.
kb(workbench.action.markdown.openPreviewSideBySide)	Open Preview to the Side	workbench.action.

界面(Preferences)

Key	Command	Command id
kb(workbench.action.openGlobalSettings)	Open User Settings	workbench.action.openGlobalSettings
kb(workbench.action.openWorkspaceSettings)	Open Workspace Settings	workbench.action.openWorkspaceSettings
kb(workbench.action.openGlobalKeybindings)	Open Keyboard Shortcuts	workbench.action.openGlobalKeybindings
kb(workbench.action.openSnippets)	Open User Snippets	workbench.action.openSnippets
kb(workbench.action.selectTheme)	Select Color Theme	workbench.action.selectTheme
kb(workbench.action.configureLocale)	Configure Display Language	workbench.action.configureLocale

调试(Debug)

Key	Command	Command id
kb(editor.debug.action.toggleBreakpoint)	Toggle Breakpoint	editor.debug.action.toggleBreakpoint
kb(workbench.action.debug.continue)	Continue	workbench.action.debug.continue
kb(workbench.action.debug.start)	Pause	workbench.action.debug.start
kb(workbench.action.debug.stepInto)	Step Into	workbench.action.debug.stepInto
kb(workbench.action.debug.stepOut)	Step Out	workbench.action.debug.stepOut
kb(workbench.action.debug.stepOver)	Step Over	workbench.action.debug.stepOver
kb(workbench.action.debug.stop)	Stop	workbench.action.debug.stop
kb(editor.action.showHover)	Show Hover	editor.action.showHover

任务(Tasks)

Key	Command	Command id
kb(workbench.action.tasks.build)	Run Build Task	workbench.action.tasks.build
kb(workbench.action.tasks.test)	Run Test Task	workbench.action.tasks.test

扩展(Extensions)

Key	Command	
kb(workbench.extensions.action.installExtension)	Install Extension	workb
kb(workbench.extensions.action.listExtensions)	Show Installed Extensions	workb
kb(workbench.extensions.action.listOutdatedExtensions)	Show Outdated Extensions	workb
kb(workbench.extensions.action.listSuggestedExtensions)	Show Extension Recommendations	workb

自定义快捷键(Customizing Shortcuts)

All keyboard shortcuts in VS Code can be customized via the `User/keybindings.json` file.

VS Code 中的所有键盘快捷键都可以通过 `User/keybindings.json` 文件进行自定义。

- To configure keyboard shortcuts the way you want, go to the menu under **File > Preferences > Keyboard Shortcuts**.
- This will open the Default Keyboard Shortcuts on the left and your `User/keybindings.json` file where you can overwrite the default bindings on the right.
- 要根据需要配置键盘快捷键，请转到 `文件>首选项>键盘快捷键` 下的菜单。
- 这将打开左侧的默认键盘快捷键和您的 `User/keybindings.json` 文件，您可以在右侧文件中，覆盖左侧的默认绑定。

快捷键规则(Keyboard Rules)

The keyboard shortcuts dispatching is done by analyzing a list of rules that are expressed in JSON. Here are some examples:

键盘快捷键分配通过以 `JSON` 表示的规则列表来完成。下面是一些例子：

```

// Keybindings that are active when the focus is in the editor
{ "key": "home",           "when": "editorTextFocus", "command": "cursorHome" },
{ "key": "shift+home",     "when": "editorTextFocus", "command": "cursorHomeSelect" },

// Keybindings that are complementary
{ "key": "f5",             "when": "inDebugMode",      "command": "workbench.action.debug.c
ontinue" },
{ "key": "f5",             "when": "!inDebugMode",    "command": "workbench.action.debug.s
tart" }

// Global keybindings
{ "key": "ctrl+f",          "command": "actions.find" },
{ "key": "alt+left",        "command": "workbench.action.navigateBack" },
{ "key": "alt+right",       "command": "workbench.action.navigateForward" },

// Global keybindings using chords
{ "key": "ctrl+k enter",    "command": "workbench.files.action.addActionToWorkingFiles" },
{ "key": "ctrl+k ctrl+w",   "command": "workbench.files.action.closeAllFiles" },

```

Each rule consists of:

每个规则包括：

- a **required** `key` that describes the pressed keys.
- an **optional** `when` containing a boolean expression that will be evaluated depending on the current **context**.
- an **optional** `command` containing the identifier of the command to execute.
- 描述按下的键的 必需键 `key`。
- `when` 包含将根据当前上下文进行求值的布尔表达式,非必须。
- 一个包含要执行的 命令的标识符 的 可选命令。

When a key is pressed:

当按下一个键：

- the rules are evaluated from **bottom to top**.
- the first rule that matches, both the `key` and in terms of `when`, is accepted.
- no more rules are processed.
- if a rule is found and has a `command set`, the `command` is executed.
- 快捷键规则从底层到顶层寻找，顶层（用户设置）会覆盖默认。
- 匹配的第一个规则，无论是 `key` 还是 `when`，都被接受。

- 不再处理任何规则。
- 如果发现规则并且具有 命令集，则按照命令集合依次执行命令。

The additional `user/keybindings.json` rules are appended at runtime to the bottom of the default rules, thus allowing them to overwrite the default rules. The `user/keybindings.json` file is watched by VS Code so editing it while VS Code is running will update the rules at runtime.

附加的 `user/keybindings.json` 规则在运行时附加到默认规则的底部，从而允许它们覆盖默认规则。VS Code 监视 `user/keybindings.json` 文件，因此在VS Code运行时进行编辑将立即更新规则。

接受的键(Accepted keys)

The `key` is made up of modifiers and the key itself.

`key` 由修改符和 `key` 本身组成。

The following modifiers are accepted:

接受以下修饰符：

OS	Modifiers
OS X	<code>kbstyle(ctrl+)</code> , <code>kbstyle(shift+)</code> , <code>kbstyle(alt+)</code> , <code>kbstyle(cmd+)</code>
Windows	<code>kbstyle(ctrl+)</code> , <code>kbstyle(shift+)</code> , <code>kbstyle(alt+)</code> , <code>kbstyle(win+)</code>
Linux	<code>kbstyle(ctrl+)</code> , <code>kbstyle(shift+)</code> , <code>kbstyle(alt+)</code> , <code>kbstyle(meta+)</code>

The following keys are accepted:

接受以下键：

- `kbstyle(f1-f19)` , `kbstyle(a-z)` , `kbstyle(0-9)`
- `kbstyle(`)` , `kbstyle(-)` , `kbstyle(=)` , `kbstyle([])` , `kbstyle([])` , `kbstyle(\)` , `kbstyle(;)` , `kbstyle(')` , `kbstyle(,)` , `kbstyle(.)` , `kbstyle(/)`
- `kbstyle(left)` , `kbstyle(up)` , `kbstyle(right)` , `kbstyle(down)` , `kbstyle(pageup)` , `kbstyle(pagedown)` , `kbstyle(end)` , `kbstyle(home)`
- `kbstyle(tab)` , `kbstyle(enter)` , `kbstyle(escape)` , `kbstyle(space)` , `kbstyle(backspace)` , `kbstyle(delete)`
- `kbstyle(pausebreak)` , `kbstyle(capslock)` , `kbstyle(insert)`
- `kbstyle(numpad0-numpad9)` , `kbstyle(numpad_multiply)` , `kbstyle(numpad_add)` , `kbstyle(numpad_separator)`
- `kbstyle(numpad_subtract)` , `kbstyle(numpad_decimal)` , `kbstyle(numpad_divide)`

Chords are described by separating the two keypresses with a space. E.g.: `kbstyle(ctrl+k ctrl+c)` .

通过用空格分隔两个按键来描述和弦。例如：`kbstyle (ctrl + k ctrl + c)`。

键盘布局(Keyboard layouts)

Note: This section relates only to key bindings, not to typing in the editor.

注意：本节仅涉及键绑定，而不是在编辑器中键入。

The keys above are string representations for virtual keys and do not necessarily relate to the produced character when they are pressed. More precisely:

上面的键是虚拟键的字符串形式表示，并且当它们被按下时不与所产生的字符相关。也就是说：

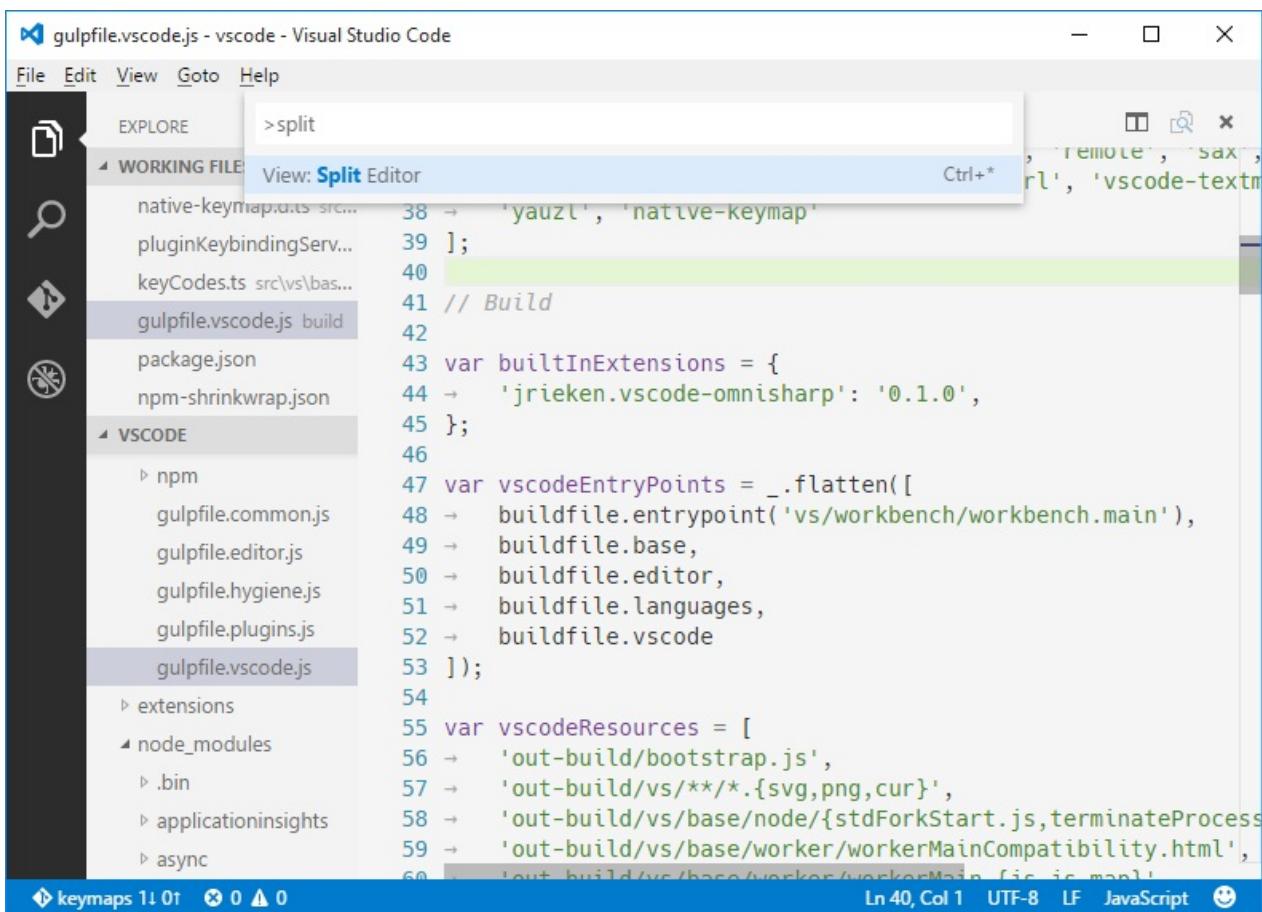
- Reference: [https://msdn.microsoft.com/en-us/library/windows/desktop/dd375731\(v=vs.85\)](https://msdn.microsoft.com/en-us/library/windows/desktop/dd375731(v=vs.85))
- `kbstyle(tab) for VK_TAB (0x09)`
- `kbstyle(;) for VK_OEM_1 (0xBA)`
- `kbstyle(=) for VK_OEM_PLUS (0xBB)`
- `kbstyle(,) for VK_OEM_COMMA (0xBC)`
- `kbstyle(-) for VK_OEM_MINUS (0xBD)`
- `kbstyle(.) for VK_OEM_PERIOD (0xBE)`
- `kbstyle(/) for VK_OEM_2 (0xBF)`
- `kbstyle(`) for VK_OEM_3 (0xC0)`
- `kbstyle([) for VK_OEM_4 (0xDB)`
- `kbstyle(\) for VK_OEM_5 (0xDC)`
- `kbstyle([]) for VK_OEM_6 (0xDD)`
- `kbstyle(') for VK_OEM_7 (0xDE)`
- etc.

Different keyboard layouts usually reposition the above virtual keys or change the characters produced when they are pressed. When using a different keyboard layout than the standard US, Visual Studio Code does the following:

不同的键盘布局通常重新定位上述虚拟键或改变按下时产生的字符。当使用与标准美国不同的键盘布局时，Visual Studio Code执行以下操作：

All the key bindings are rendered in the UI using the current system's keyboard layout. For example, `split Editor` when using a French (France) keyboard layout is now rendered as `kbstyle(Ctrl+*)` :

所有的键绑定都使用当前系统的键盘布局在UI中呈现。例如，使用法语（法国）键盘布局时的 `Split Editor` 现在呈现为 `kbstyle (Ctrl + *)`：



When editing `keybindings.json`, VS Code highlights misleading key bindings - those that are represented in the file with the character produced under the standard US keyboard layout, but which need pressing keys with different labels under the current system's keyboard layout. For example, here is how the `Default keybindings` rules look like when using a French (France) keyboard layout:

当编辑 `keybindings.json` 时，VS Code 突出显示误导键绑定 - 在文件中用在美国标准键盘布局下产生的字符表示的那些，但是需要在当前系统的键盘布局下按下具有不同标签的键。例如，以下是使用法语（法国）键盘布局时 `默认键盘` 绑定规则的外观：

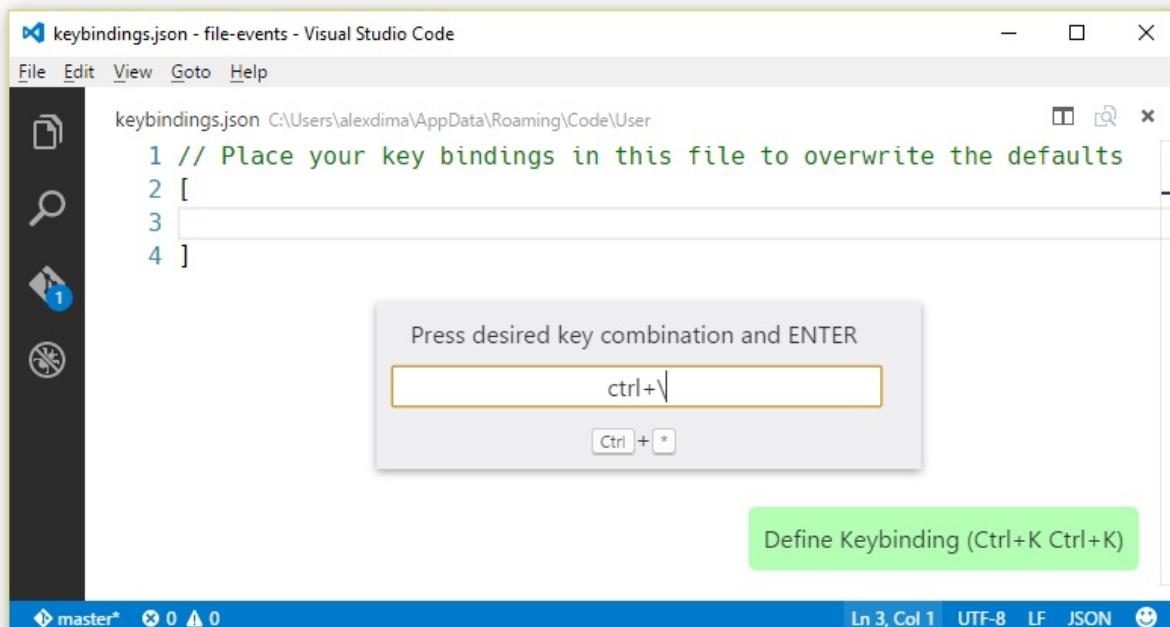
```

300 { "key": "ctrl+shift+j", "command": "workbench.action.search.toggleQueryDetails",
301 ..... "when": "searchViewletVisible" },
302 { "key": "ctrl+t", "command": "workbench.action.showAllSymbols" },
303 { "key": "f1", "command": "workbench.action.showCommands" },
304 { "key": "c" For your current keyboard layout press Ctrl + * tion.showCommands" },
305 { "key": "C Key or key sequence (separated by space)" "command": "workbench.action.showErrorsWarnings" },
306 { "key": "I" "ctrl+\\" "command": "workbench.action.splitEditor" },
307 { "key": "ctrl+shift+b", "command": "workbench.action.tasks.build" },
308 { "key": "ctrl+shift+t", "command": "workbench.action.tasks.test" },
309 { "key": "ctrl+shift+c", "command": "workbench.action.terminal.openNativeConsole" },
310 { "key": "f11", "command": "workbench.action.toggleFullScreen" },
311 { "key": "ctrl+b", "command": "workbench.action.toggleSidebarVisibility" },
312 { "key": "ctrl+=" "command": "workbench.action.zoomIn" },
313 { "key": "O" "ctrl+-" "command": "workbench.action.zoomOut" },
314 { "key": "ctrl+k enter", "command": "workbench.files.action.addToWorkingFiles" },
315 { "key": "ctrl+k ctrl+w", "command": "workbench.files.action.closeAllFiles" },

```

There is also a widget that helps input the key binding rule when editing `keybindings.json`. To launch the **Define Keybinding** widget, press `kb(editor.action.defineKeybinding)`. The widget listens for key presses and renders the serialized JSON representation in the text box and below it, the keys that VS Code has detected under your current keyboard layout. Once you've typed the key combination you want, you can press `kbstyle(Enter)` and a rule snippet will be inserted.

还有一个小部件，用于在编辑 `keybindings.json` 时帮助输入键绑定规则。要启动定义键绑定小部件，请按 `kb(editor.action.defineKeybinding)`。窗口小部件监听按键并在文本框及其下面呈现序列化的 JSON 表示，VS Code 在您当前的键盘布局下检测到的键。键入所需的组合键后，您可以按 `kbstyle(Enter)`，然后插入规则片段。



Note: Visual Studio Code detects your current keyboard layout on start-up and then caches this information. For a good experience, we recommend restarting VS Code if you change your keyboard layout.

注意：Visual Studio Code 在启动时检测您当前的键盘布局，然后缓存此信息。为了获得良好的体验，我们建议您在更改键盘布局时重新启动VS Code。

下一步(Next Steps)

Now that you know about our Key binding support, what's next...

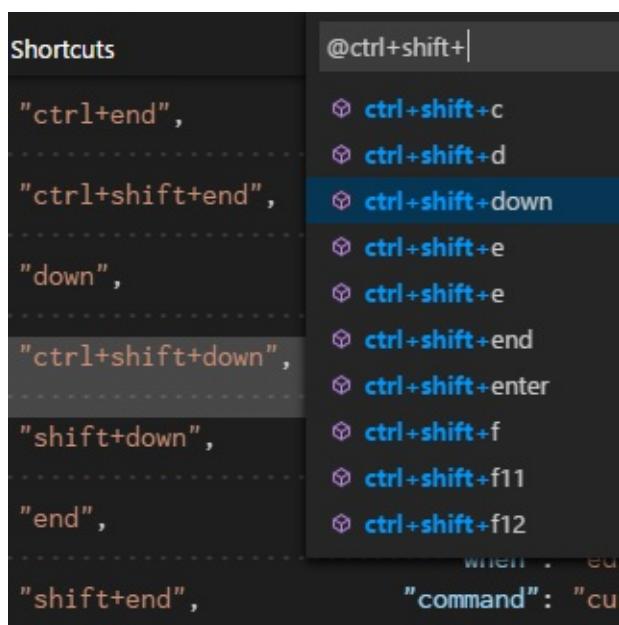
现在你知道了我们支持快捷键自定义，接下来是什么...

- [Customization](#) - Configure Code the way you want - Themes, Settings and more
- [Language Support](#) - Our Good, Better, Best language grid to see what you can expect
- [Debugging](#) - This is where VS Code really shines
- [Node.js](#) - End to end Node.js scenario with a sample app

Common Questions

Q: How to find out what command is bound to a specific key?

A: In the Default Keyboard Shortcuts, open `Quick outline` by pressing `kb(workbench.action.gotoSymbol)`



Q: How to add a key binding to an action? E.g. Add Ctrl+D to Delete Lines

A: Find a rule that triggers the action in the Default Keyboard Shortcuts and write a modified version of it in your `User/keybindings.json` file:

```
// Original, in Default Keyboard Shortcuts
{ "key": "ctrl+shift+k",           "command": "editor.action.deleteLines",
  "when": "editorTextFocus" },
// Modified, in User/keybindings.json, Ctrl+D now will also trigger this action
{ "key": "ctrl+d",                "command": "editor.action.deleteLines",
  "when": "editorTextFocus" },
```

Q: How to remove a key binding from an action? E.g. Remove Ctrl+Shift+K from Delete Lines

A: Find a rule that triggers the action in the Default Keyboard Shortcuts and write a modified version of it in your `User/keybindings.json` file:

```
// Original, in Default Keyboard Shortcuts
{ "key": "ctrl+shift+k",           "command": "editor.action.deleteLines",
  "when": "editorTextFocus" },
// Modified, in User/keybindings.json, Ctrl+Shift+K won't do anything anymore since command is empty
{ "key": "ctrl+shift+k",           "when": "editorTextFocus" },
```

Q: How can I add a key binding for only certain file types?

A: Use the `editorLangId` context key in your `when` clause:

```
{ "key": "shift+alt+a",           "command": "editor.action.blockComment",
  "when": "editorTextFocus && editorLangId == 'csharp' },
```

Q: I have modified my key bindings in `User/keybindings.json`, why don't they work?

A: The most common problem is a syntax error in the file. Otherwise, try removing the `when` clause or picking a different `key`. Unfortunately, at this point, it is a trial and error process.

向 VSC 添加代码段(Adding Snippets to Visual Studio Code)

Code snippets are ready-made snippets of code you can quickly insert into your source code. For example, a `for` code snippet creates an empty `for` loop.

代码段是一种可以快速插入到源代码中的代码片段。例如，对于代码片段创建一个空的`for`循环。

Each snippet defines a prefix under which it will appear in IntelliSense via `(kb(editor.action.triggerSuggest))` as well as a body inserted when the snippet is selected. The snippet syntax follows the [TextMate snippet syntax](#) with the exception of 'regular expression replacements', 'interpolated shell code' and 'transformations', which are not supported.

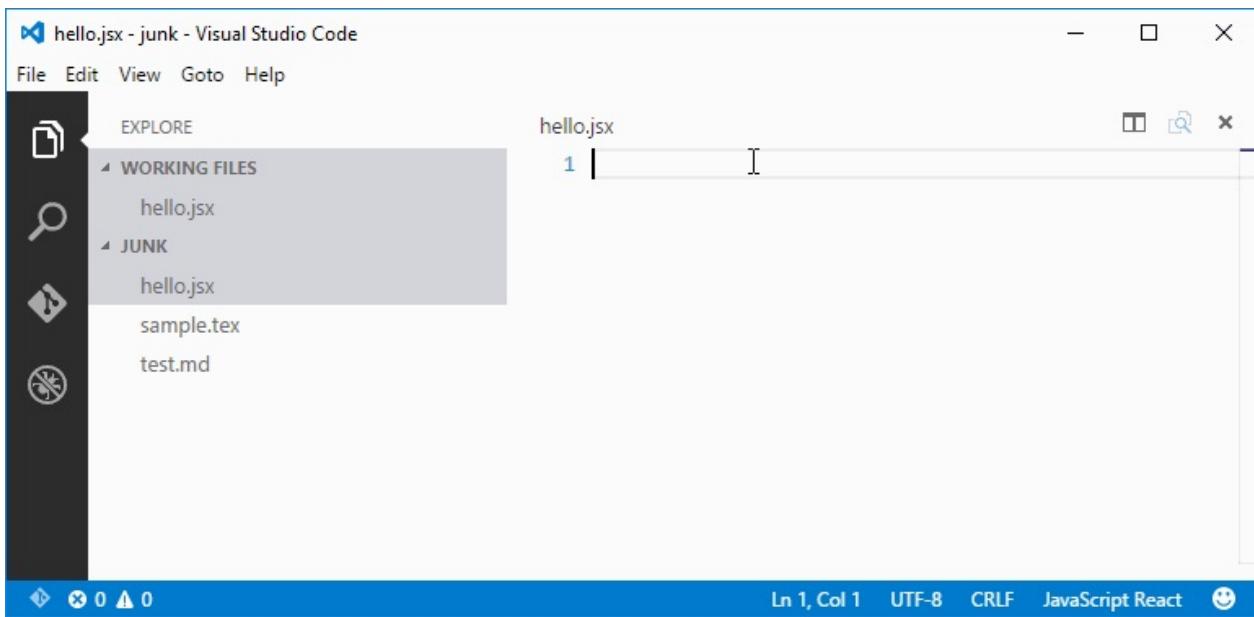
每个代码段定义时有一个前缀，只要该前缀出现，它就会出现在 `IntelliSense` `via (kb (editor.action.triggerSuggest))` 中，以及在选择代码段时插入的主体。代码段语法遵循 `TextMate` 代码段 语法，但不支持 “正则表达式替换” ， “插入的shell代码” 和 “转换” 。

```
typescript.json C:\Users\martinae\AppData\Roaming\Code\User\snippets
1 {
2 /*
3 Place your snippets for TypeScript here. Each snippet is defined under a snippet name and has a prefix, body and
4 description. The prefix is what is used to trigger the snippet and the body will be expanded and inserted. Possible
5 variables are: $1, $2 for tab stops, ${id} and ${id:label} and ${1:label} for variables. Variables with the same id
6 are connected.
7 */
8   "class": {
9     "prefix": "class",
10    "body": [
11      "class $1 ${2:extends ${3:SuperClass}} ${4:implements ${5:SuperInterface}} {",
12      "\t$0",
13      "}"
14    ],
15    "description": "New class with optional supertype and superinterface"
16  }
17 }
```

从扩展市场添加代码段(Add Snippets from the Marketplace)

Many snippets have been uploaded to the VS Code [Extension Marketplace](#) by the community. If you find one you want to use, simply install it and restart VS Code and the new snippet will be available.

许多片段已被社区上传到VS Code扩展市场。如果你发现了一个代码段并且想要使用，你只需安装它，并重新启动VS Code，这个代码段就可以用了。



You can also browse the [VS Code Marketplace](#) site directly to find available snippets.

您还可以直接浏览[VS Code 扩展市场的网站](#)，来查找可用的代码段。

创建自己的代码段(**Creating your Own Snippets**)

You can define your own snippets for specific languages. Snippets are defined in a JSON format.

您可以为特定语言定义自己的代码段。代码段以JSON格式定义。

The example below is a `For Loop` snippet for `JavaScript`.

下面的示例是一个 `JavaScript` 的 `For` 循环代码段。

```
"For Loop": {  
    "prefix": "for",  
    "body": [  
        "for (var ${index} = 0; ${index} < ${array}.length; ${index}++) {}",  
        "\tvar ${element} = ${array}[${index}];",  
        "\t${element}",  
        "}"],  
    "description": "For Loop"  
},
```

In the example above:

在上面的实例中：

- `For Loop` is the snippet name

- `prefix` defines a prefix used in the IntelliSense drop down. In this case `for`.
- `body` is the snippet content.
- `For Loop` 是这个代码段的名称。
- `prefix` 在这种情况下定义了在 `IntelliSense` 下拉列表 中使用的前缀。
- `body` 是这个代码段的额内容。

Possible variables are:

可能的变量有：

- `$1, $2` for tab stops
- `${id}` and `${id:label}` and `${1:label}` for variables
- Variables with the same id are connected.
- `description` is the description used in the IntelliSense drop down
- `$1, $2` 用于制表符。
- `${id}` , `${id:label}` 和 `${1:label}` 是变量。
- 有相同 `id` 的变量相互关联。
- `description` 是在 `IntelliSense` 下拉列表 中的描述。

To open up a snippet file for editing, open **User Snippets** under **File > Preferences** and select the language for which the snippets should appear.

打开要编辑的代码段文件，请在 `文件>首选项` 下打开用户代码段，然后选择 `代码段应显示的语言`。

In case your snippet should contain `{` or `}`, it is possible to escape them, in JSON as `\{\` and `\}`

如果您的代码片段应包含 `{` 或 `}`，可以转义它们，在 `JSON` 中为 `\{\` 和 `\}`。

Once you have added a new snippet, you can try it out right away, no restart needed.

添加新代码段后，您可以立即尝试，无需重新启动。

使用 TextMate 代码段(Using TextMate Snippets)

You can also add TextMate snippets (`.tmSnippets`) to your VS Code installation using the [yo code](#) extension generator. The generator has an option `New Code Snippets` which lets you point to a folder containing multiple `.tmSnippets` files and they will be packaged into a VS Code snippet extension. The generator also supports Sublime snippets (`.sublime-snippets`).

您还可以使用 `yo` 代码扩展生成器 将 `TextMate` 代码段 (`.tmSnippets`) 添加到 `VS Code` 安装。生成器有一个 `选择新代码片段`，它允许您指向包含多个 `.tmSnippets` 文件的文件夹，它们将被打包到 `VS Code` 代码段代码扩展中。生成器还支持 `Sublime` 代码段 (`.sublime-snippets`)。

The final generator output has two files: an extension manifest `package.json` which has metadata to integrate the snippets into `VS Code` and a `snippets.json` file which includes the snippets converted to the `VS Code` snippet format.

最终的生成器输出有两个文件：`扩展清单 package.json`，其将会把代码集成到 `VS Code` 的元默认数据以及 `snippets.json` 文件中，其包括被转换为 `VS Code` 代码片段格式的片段。

```
.
├── snippets           // VS Code integration
│   └── snippets.json // The JSON file w/ the snippets
└── package.json       // extension's manifest
```

Copy the generated snippets folder to a new folder under your `.vscode/extensions` folder and restart `VS Code`.

将生成的 `snippets` 文件夹复制到 `.vscode/extensions` 文件夹下的新文件夹中，然后重新启动 `VS Code`。

在扩展市场中共享您的代码段(**Sharing Your Snippets in the Marketplace**)

Once you have created your snippets and tested them out, you can share them with the community.

一旦您创建了代码段并对其进行测试，就可以与社区共享。

To do this, you need to create a snippet extension. If you've used the `yo code` extension generator, your snippet extension is ready to be published.

为此，您需要创建一个代码段扩展。如果您已使用 `yo code` 扩展程序生成器，则您的代码段扩展程序已经可以发布。

If you want to share user snippets, you'll need to package your snippet json file along with an extension manifest which has the necessary metadata to integrate the snippets into `VS Code`.

如果您想要共享你的代码段，则需要打包您的代码段 `json` 文件以及具有必要默认数据的扩展清单，以将代码段集成到 `VS Code` 中。

Depending on your platform, your user snippets file is located here:

根据您的操作系统，您的用户代码段文件位于以下几处：

- **Windows** %APPDATA%\Code\User\snippets\language.json
- **Mac** \$HOME/Library/Application Support/Code/User/snippets/language.json
- **Linux** \$HOME/.config/Code/User/snippets/language.json

where language.json depends on the targeted language of the snippets (e.g. markdown.json for Markdown snippets). Create a new folder for your extension and copy your snippet file to a snippets subdirectory.

(language).json 的位置取决于片段的目标语言（例如 Markdown 片段的 markdown.json）。为扩展程序创建一个新文件夹，并将您的代码段文件复制到 snippets 子目录。

Now add an extension manifest package.json file to the extension folder. The snippet extension manifest follows the structure defined in the [Extension Manifest](#) reference and provides a snippets contribution.

现在，将扩展清单 package.json 文件添加到扩展文件夹。代码段扩展清单遵循 “扩展清单” 参考中定义的结构，并提供了代码段贡献。

Below is an example manifest for Markdown snippets:

以下是 Markdown 片段的示例清单：

```
{
  "name": "DM-Markdown",
  "publisher": "mscott",
  "description": "Dunder Mifflin Markdown snippets",
  "version": "0.1.0",
  "engines": { "vscode": "0.10.x" },
  "categories": ["Snippets"],
  "contributes": {
    "snippets": [
      {
        "language": "markdown",
        "path": "./snippets/markdown.json"
      }
    ]
  }
}
```

Note that snippets need to be associated with a language identifier. This can be a language supported directly by VS Code or a language provided by an extension. Make sure the language identifier is correct.

请注意，代码段需要与 语言标识符 相关联。这是被 VS Code 默认支持的语言，或由扩展提供的语言。请确保语言标识符正确。

You then use the [vsce publishing tool](#) to publish the snippet extension to the [VS Code Extension Marketplace](#).

然后使用[vsce](#)发布工具将代码段扩展名发布到VS Code扩展市场。

Tip: To make it easy for users to find your snippet, include the word "snippet" in the extension description and set the `Category` to `Snippets` in your `package.json`.

提示：为了方便用户查找您的代码段，请在扩展说明中包含“snippet”一词，并将`package`中的`category`设置为`Snippets`。

We also have recommendations on how to make your extension look great on the VS Code Marketplace, see [Marketplace Presentation Tips](#).

我们还有一些建议如何使您的扩展在VS代码市场发展壮大，请参阅市场演示提示。

下一步(Next Steps)

Snippets are just one way to extend VS Code. If you'd like to learn more about VS Code extensibility, try these topics:

代码片段只是扩展VS Code的一种方法。如果您想了解有关VS代码可扩展性的更多信息，请尝试以下信息：

- [Colorizers and Bracket Matchers](#) - Learn how to import TextMate colorizers
- [Custom themes](#) - Learn how to import existing TextMate themes.
- [Extending Visual Studio Code](#) - Learn about other ways to extend VS Code
- [Editing Evolved](#) - Learn more about the VS Code editor's capabilities

Common Questions

Q: I created a snippets extension but they aren't showing up in the VS Code editor?

A: Be sure you have correctly specified the `language` identifier for your snippet (e.g. `markdown` for Markdown .md files, `plaintext` for Plain Text .txt files). Also verify that the relative path to the snippets json file is correct.

Themes, Snippets and Colorizers - 主题，片段与调色板

Custom color and icons themes, snippets and language syntax colorizers bring an editor to life. There are lots of existing TextMate customization files available and VS Code lets you easily package and reuse these. You can directly use `.tmTheme`, `.tmSnippets`, and `.tmLanguage` files in your extensions and share them in the extension [Marketplace](#). This topic describes how to reuse TextMate files as well as create and share your own themes, snippets and colorizers.

颜色、图标主题、片段和语法高亮的自定义调色板可以给编辑器以生命力。已经存在大量可用的TextMate自定义文件，并且VS Code让你可以轻易的打包和重用它们。你可以直接使用扩展名为 `.tmTheme`、`.tmSnippets` 和 `.tmLanguage` 的文件，也可以在扩展[市场](#)中分享你的主题。这个标题描述了如何重用TextMate的文件，以及如何创建或分享你自己的主题、片段和调色板。

Adding a new Color Theme - 添加一个新的颜色主题

Colors visible in the VS Code user interface fall in two categories:

VS Code用户界面中的可见颜色分为两类：

- Workbench colors used in views and editors, from the Activity Bar to the Status Bar. A complete list of all these colors can be found in the [color reference](#).
- Syntax highlighting colors used for source code in the editor. The theming of these colors is different as syntax colorization is based on TextMate grammars and TextMate themes.

Workbench colors

The easiest way to create a new workbench color theme is to start with an existing color theme and customize it:

- Switch to the color theme that you want to modify.
- Open the [settings](#) and make changes to view and editor colors using the `workbench.colorCustomizations`. Changes are applied live to your VS Code instance and no refreshing or reloading is necessary.

- A complete list of all themable colors can be found in the [color reference](#).

Syntax highlighting colors

For syntax highlighting colors, there are two approaches. You just simply reference an existing TextMate theme (`.tmTheme` file) from the community, or you can come up with your own theming rules. The easiest way is to start with an existing theme and customize it:

- Switch to the color theme to customize and use the `editor.tokenColorCustomizations settings`. Changes are applied live to your VS Code instance and no refreshing or reloading is necessary.
- The setting supports a simple model with a set of common token types such as 'comments', 'strings' and 'numbers' available. If you want to color more than that, you need to use textMate theme rules directly.

TextMate theme rules

To write TextMate theme rules, you need to know about TextMate grammars and scopes.

`TextMate grammars` consist of a set of regular expression that are used to create a syntax tree out of the source code. Each tree node spans a source range and represents a `scope`. Scopes have a name and stand for either code sections (such as functions, blocks, comments) or symbols (for example keywords, numbers, operators).

Here's an example of the scope hierarchy generated for a JavaScript code sample:

```
function f1() {
```

```

      function          f1           ()           {
        source.js
        meta.function.js
      storage.type.function.js meta.definition.function.js   meta.parameters.js   meta.block.js
        entity.name.function.js punctuation.definition.parameters.js punctuation.definition.block.js
    
```

Each scope name consists of segments separated by dots. The last segment is the name of the language the symbol belongs to: `entity.name.function.js`.

A good overview of scope names that TextMate grammars typically generate can be found [here](#).

The list of scope names active at a given offset are the input for syntax highlighting.

Text Mate themes describe the theming rules used for syntax highlighting. Each rule consists of one or more scope selectors and a set of styles: colors (foreground & background) and font styles (bold, italics and underline).

To evaluate the style of a symbol at a given offset, the scopes at that offset are computed. The theming rules are then processed first to last. The rule's scope selectors are matched against that set of scopes. The rule with the most specific match wins.

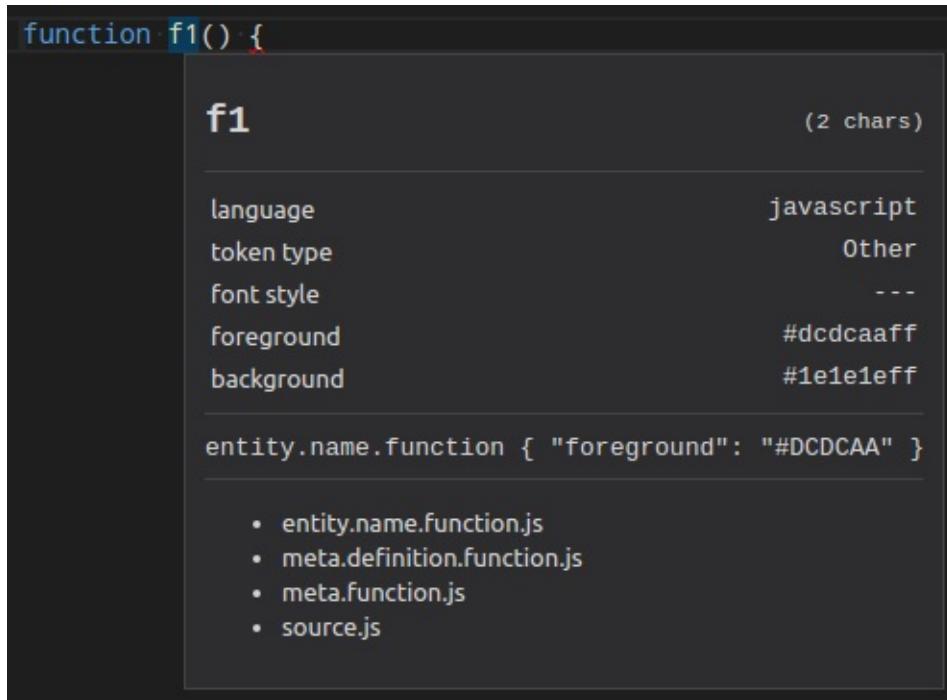
Here are some example theming rules. The `scope` property lists the rules scope selectors. The `setting` property describes the styles to apply when the rule wins. The `name` is just used for documentation.

- TextMate themes assign a set of styles to one or more scopes. The styles are the foreground color, the background color and bold, italics and underline. A theme consist of a set of rules. To evaluate the style of a symbol, the rules are processed first to last and each scope selector is matched against to symbols scope and parent scopes. The most specific rule is used for styling the symbol.
- Scope selector support prefix matching and matching against parent scopes

```
{
  "name": "Variables",
  "scope": "variable",
  "settings": {
    "foreground": "#dc3958",
    "fontStyle": "bold underline"
  }
},
{
  "name": "Functions",
  "scope": [
    "entity.name.function",
    "meta.selector.css entity.name.tag",
    "entity.name.method - source.java"
  ],
  "settings": {
    "foreground": "#8ab1b0"
  }
}
```

- `variable` matches all scopes that start with `variable : variable.js , variable.parameter.java ...`
- `meta.selector.css entity.name.tag` matches all scopes that start with `entity.name.tag` and have a parent scope that matches `meta.selector.css`
- `entity.name.method - source.java` matches all scopes that start with `entity.name.method` but are not inside a parent scope that matches `source.java`
- Learn more about scope selectors [here](#).

You can use the **Developer Tools: Inspect TM Scopes** command from the **Command Palette** (`kb(workbench.action.showCommands)`) to inspect the scopes of a token at the cursor and to see which theming rule has been applied.

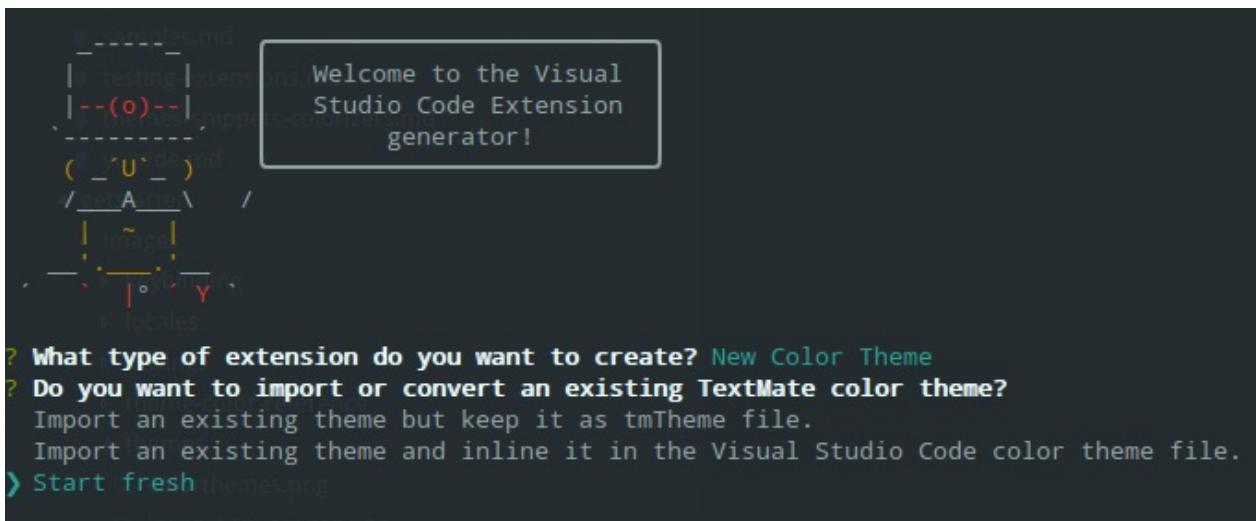


Create a new color theme

- Generate a theme file using the **Generate Color Theme from Current Settings** command from the **Command Palette**
- Use VS Code's [Yeoman](#) extension generator, `yo code`, to generate a new theme extension:

```
npm install -g yo generator-code
yo code
```

- If you customized a theme as described above, select 'Start fresh'.



- Copy the theme file generated from your settings to the new extension.
- To use a existing TextMate theme, you can tell the extension generator to import a TextMate theme file and package it for use in VS Code. Alternatively, if you have already downloaded the theme, replace the `tokenColors` section with a link to the `.tmTheme` file to use.

```
{
  "type": "dark",
  "colors": {
    "editor.background": "#1e1e1e",
    "editor.foreground": "#d4d4d4",
    "editorIndentGuide.background": "#404040",
    "editorRuler.foreground": "#333333",
    "activityBarBadge.background": "#007acc",
    "sideBarTitle.foreground": "#bbbbbb"
  },
  "tokenColors": "./Diner.tmTheme"
}
```

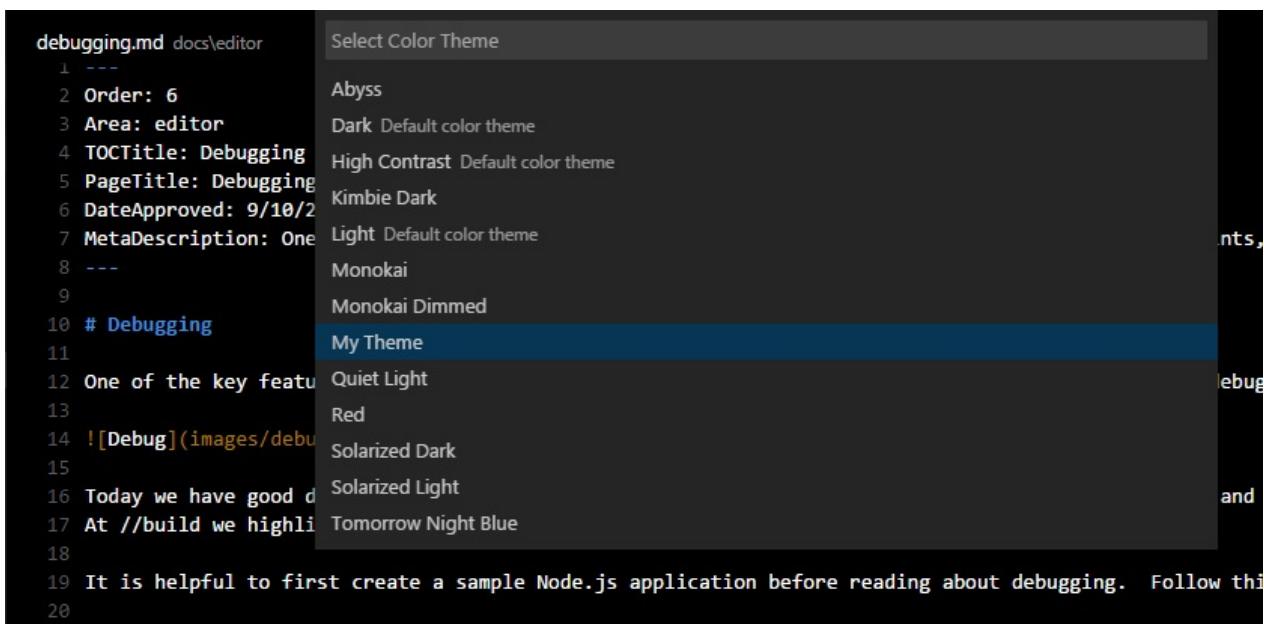
Tip: [ColorSublime](#) has hundreds of existing TextMate themes to choose from. Pick a theme you like and copy the Download link to use in the Yeoman generator or into your extension. It will be in a format like

```
"https://raw.githubusercontent.com/ColorSublime/ColorSublime-Themes/master/themes/(name).tmTheme"
```

Test a new color theme

To try out the new theme, copy the generated theme folder to a new folder under [your .vscode/extensions folder](#) and restart VS Code.

Open the Color Theme picker theme with **File > Preferences > Color Theme** and you can see your theme in the dropdown. Arrow up and down to see a live preview of your theme.



After making changes to any theme file, it is necessary to reload VS Code with `Reload Window`.

Publishing a Theme to the Extension Marketplace

If you'd like to share your new theme with the community, you can publish it to the [Extension Marketplace](#). Use the [vsce publishing tool](#) to package your theme and publish it to the VS Code Marketplace.

Tip: To make it easy for users to find your theme, include the word "theme" in the extension description and set the `Category` to `Theme` in your `package.json`.

We also have recommendations on how to make your extension look great on the VS Code Marketplace, see [Marketplace Presentation Tips](#).

Adding a new Icon Theme

You can create your own icon theme from icons (preferably SVG) and from icon fonts. As example, check out the two built-in themes: [Minimal](#) and [Seti](#).

To begin, create a VS Code extension and add the `iconTheme` contribution point.

```

"contributes": {
  "iconThemes": [
    {
      "id": "turtles",
      "label": "Turtles",
      "path": "./fileicons/turtles-icon-theme.json"
    }
  ]
}

```

The `id` is the identifier for the icon theme. It is currently only used internally. In the future, it might be used in the settings, so make it unique but also readable. `label` is shown in the icon theme picker drop-down. The `path` points to a file in the extension that defines the icon set. If your icon set name follows the `*icon-theme.json` name scheme, you will get completion support and hovers in VS Code.

Icon Set File

The icon set file is a JSON file consisting file icon associations and icon definitions.

An icon association maps a file type ('file', 'folder', 'json-file'...) to an icon definition. Icon definitions define where the icon is located: That can be an image file or also glyph in a font.

Icon definitions

The `iconDefinitions` section contains all definitions. Each definition has an id, which will be used to reference the definition. A definition can be referenced also by more than one file association.

```

"iconDefinitions": {
  "_folder_dark": {
    "iconPath": "./https://code.visualstudio.com/assets/docs/extensions/Folder
_16x_inverse.svg"
  }
}

```

This icon definition above contains a definition with the identifier `_folder_dark`.

The following properties are supported:

- `iconPath` : When using a svg/png: the path to the image.
- `fontCharacter` : When using a glyph font: The character in the font to use.
- `fontColor` : When using a glyph font: The color to use for the glyph.
- `fontSize` : When using a font: The font size. By default, the size specified in the font

specification is used. Should be a relative size (e.g. 150%) to the parent font size.

- `fontId` : When using a font: The id of the font. If not specified, the first font specified in font specification section will be picked.

File association

Icons can be associated to folders, folder names, files, file extensions, file names and [language ids](#).

Additionally each of these associations can be refined for 'light' and 'highContrast' color themes.

Each file association points to an icon definition.

```
"file": "_file_dark",
"folder": "_folder_dark",
"folderExpanded": "_folder_open_dark",
"folderNames": {
    ".vscode": "_vscode_folder",
},
"fileExtensions": {
    "ini": "_ini_file",
},
"fileName": {
    "win.ini": "_win_ini_file",
},
"languageIds": {
    "ini": "_ini_file"
},
"light": {
    "folderExpanded": "_folder_open_light",
    "folder": "_folder_light",
    "file": "_file_light",
    "fileExtensions": {
        "ini": "_ini_file_light",
    }
},
"highContrast": {
}
```

- `file` is the default file icon, shown for all files that don't match any extension, filename or language id. Currently all properties defined by the definition of the file icon will be inherited (only relevant for font glyphs, useful for the `fontSize`).
- `folder` is the folder icon for collapsed folders, and if `folderExpanded` is not set, also for expanded folders. Icons for specific folder names can be associated using the `folderNames` property. The folder icon is optional. If not set, no icon will be shown for folder.

- `folderExpanded` is the folder icon for expanded folders. The expanded folder icon is optional. If not set, the icon defined for `folder` will be shown.
- `folderNames` associates folder names to icons. The key of the set is the folder name, not including any path segments. Patterns or wildcards are not supported. Folder name matching is case insensitive.
- `folderNamesExpanded` associates folder names to icons for expanded folder. The key of the set is the folder name, not including any path segments. Patterns or wildcards are not supported. Folder name matching is case insensitive.
- `languageIds` associates languages to icons. The key in the set is the language id as defined in the [language contribution point](#). The language of a file is evaluated based on the file extensions and file names as defined in the language contribution. Note that the 'first line match' of the language contribution is not considered.
- `fileExtensions` associates file extensions to icons. The key in the set is the file extension name. The extension name is a file name segment after a dot (not including the dot). File names with multiple dots such as `lib.d.ts` can match multiple extensions; 'd.ts' and 'ts'. Extensions are compared case insensitive.
- `fileNames` associates file names to icons. The key in the set is the full file name, not including any path segments. Patterns or wildcards are not supported. File name matching is case insensitive. A 'fileName' match is the strongest match, and the icon associated to the file name will be preferred over an icon of a matching `fileExtension` and also of a matching language id.

A file extension match is preferred over a language match, but is weaker than a file name match.

The `light` and the `highContrast` section have the same file association properties as just listed. They allow to override icons for the corresponding themes.

Font definitions

The 'fonts' section lets you declare any number of glyph fonts that you want to use. You can later reference these font in the icon definitions. The font declared first will be used as the default if an icon definition does not specify a font id.

Copy the font file into your extension and set the path accordingly. It is recommended to use [WOFF](#) fonts.

- Set 'woff' as the format.
- the weight property values are defined [here](#).
- the style property values are defined [here](#).
- the size should be relative to the font size where the icon is used. Therefore always use percentage.

```

"fonts": [
  {
    "id": "turtles-font",
    "src": [
      {
        "path": "./turtles.woff",
        "format": "woff"
      }
    ],
    "weight": "normal",
    "style": "normal",
    "size": "150%"
  }
],
"iconDefinitions": {
  "_file": {
    "fontCharacter": "\u263a",
    "fontColor": "#5f8b3b",
    "fontId": "turtles-font"
  }
}

```

Folder icons in File Icon Themes

File Icon themes can instruct the File Explorer not to show the default folder icon (the rotating triangles or "twisties") when the folder icons are good enough to indicate the expansion state of a folder. This mode is enabled by setting `"hidesExplorerArrows":true` in the File Icon theme definition file.

Using TextMate Snippets

You can also add TextMate snippets (.tmSnippets) to your VS Code installation using the [yo code](#) extension generator. The generator has an option `New Code Snippets` which lets you point to a folder containing multiple .tmSnippets files and they will be packaged into a VS Code snippet extension. The generator also supports Sublime snippets (.sublime-snippets).

The final generator output has two files: an extension manifest `package.json` which has metadata to integrate the snippets into VS Code and a `snippets.json` file which includes the snippets converted to the VS Code snippet format.

```
.  
└── snippets           // VS Code integration  
    └── snippets.json // The JSON file w/ the snippets  
└── package.json      // extension's manifest
```

Copy the generated snippets folder to a new folder under `your .vscode/extensions folder` and restart VS Code.

Sharing Your Snippets in the Marketplace

Once you have created your snippets and tested them out, you can share them with the community.

To do this, you need to create a snippet extension. If you've used the `yo code extension` generator, your snippet extension is ready to be published.

If you want to share user snippets, you'll need to package your snippet json file along with an extension manifest which has the necessary metadata to integrate the snippets into VS Code.

Depending on your platform, your user snippets file is located here:

- **Windows** `%APPDATA%\Code\User\snippets\language.json`
- **Mac** `$HOME/Library/Application Support/Code/User/snippets/(language).json`
- **Linux** `$HOME/.config/Code/User/snippets/(language).json`

where `(language).json` depends on the targeted language of the snippets (e.g. `markdown.json` for Markdown snippets). Create a new folder for your extension and copy your snippet file to a `snippets` subdirectory.

Now add an extension manifest `package.json` file to the extension folder. The snippet extension manifest follows the structure defined in the [Extension Manifest](#) reference and provides a `snippets contribution`.

Below is an example manifest for Markdown snippets:

```
{
  "name": "DM-Markdown",
  "publisher": "mscott",
  "description": "Dunder Mifflin Markdown snippets",
  "version": "0.1.0",
  "engines": { "vscode": "0.10.x" },
  "categories": ["Snippets"],
  "contributes": {
    "snippets": [
      {
        "language": "markdown",
        "path": "./snippets/markdown.json"
      }
    ]
  }
}
```

Note that snippets need to be associated with a `language` identifier. This can be a [language supported](#) directly by VS Code or a language provided by an extension. Make sure the `language` identifier is correct.

You then use the [vsce publishing tool](#) to publish the snippet extension to the [VS Code Extension Marketplace](#).

Tip: To make it easy for users to find your snippet, include the word "snippet" in the extension description and set the `Category` to `Snippets` in your `package.json`.

We also have recommendations on how to make your extension look great on the VS Code Marketplace, see [Marketplace Presentation Tips](#).

Adding a New Language (Colorizer)

Using the ['code' Yeoman generator](#), you can create an extension that adds syntax highlighting and bracket matching for a language to your VS Code installation.

Central to language support is a TextMate [language specification](#) file (`.tmLanguage`) that describes the colorizer rules. The yeoman generator either takes an existing TextMate language specification file or lets you start with a fresh one.

A good place to look for existing TextMate `.tmLanguage` files is on GitHub. Search for a TextMate bundle for the language you are interested in and then navigate to the `Syntaxes` folder. The ['code' Yeoman generator](#) can import either `.tmLanguage` or `.pList` files. When prompted for the URL or file location, pass the raw path to the `.tmLanguage` file, for example

<https://raw.githubusercontent.com/textmate/ant.tmbundle/master/Syntaxes/Ant.tmLanguage>.

Make sure that the path points to the content of the file, not the HTML file showing the content.



The generator will prompt you for other information such a unique name (this should be unique to avoid clashing with other extensions) and the language name, aliases and file extensions. You also have to provide the top level scope name of the grammar. That scope name must match the scope name in the tmLanguage file.

When the generator is finished, open the created folder in Visual Studio Code. Have a look at the generated `language-configuration.json` file: It contains more language settings such as the tokens used for comments and brackets. Make sure the configurations are accurate.

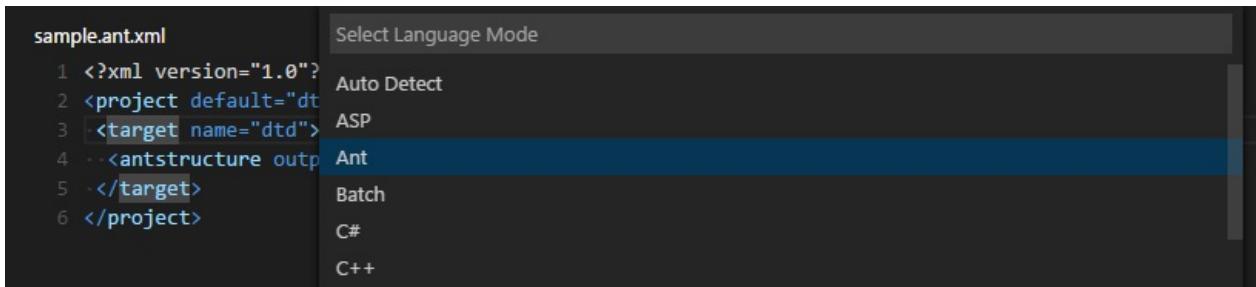
Here is an example for a language with XML-like brackets:

```
{
  "comments": {
    "lineComment": "",
    "blockComment": ["<!--", "-->"]
  },
  "brackets": [
    ["<", ">"]
  ],
  "autoClosingPairs": [
    ["<", ">"],
    ["'", "'"],
    ["\"", "\""]
  ],
  "surroundingPairs": [
    ["<", ">"],
    ["'", "'"],
    ["\"", "\""]
  ]
}
```

For more details check out the [languages contribution point documentation](#).

The generated `vsc-extension-quickstart.md` file also contains more information on how to run and debug your extension.

To use your extension in your stable VS Code installation, copy the complete output folder to a new folder under [your `.vscode/extensions` folder](#) and restart VS Code. When you restart VS Code, your new language will be visible in the language specifier drop-down and you'll get full colorization and bracket/tag matching for files matching the language's file extension.



Publishing Language Support to the Extension Marketplace

If you'd like to share your new language with the community, you can publish it to the [Extension Marketplace](#). Use the [vsce publishing tool](#) to package your extension and publish it to the VS Code Marketplace.

Tip: To make it easy for users to find your language support, include the language name and words "language" or "language support" in the extension description and set the `Category` to `Languages` in your `package.json`.

We also have recommendations on how to make your extension look great on the VS Code Marketplace, see [Marketplace Presentation Tips](#).

Add to your Language Support Extension

When you're adding a new language to VS Code, it is also great to add language [snippets](#) to support common editing actions. It is easy to [combine multiple extensions](#) like snippets and colorizers into the same extension. You can modify the colorizer extension manifest `package.json` to include a `snippets` contribution and the `snippets.json`.

```
{
  "name": "language-latex",
  "description": "LaTeX Language Support",
  "version": "0.0.1",
  "publisher": "someone",
  "engines": {
    "vscode": "0.10.x"
  },
  "categories": [
    "Languages",
    "Snippets"
  ],
  "contributes": {
    "languages": [
      {
        "id": "latex",
        "aliases": ["LaTeX", "latex"],
        "extensions": [".tex"]
      }
    ],
    "grammars": [
      {
        "language": "latex",
        "scopeName": "text.tex.latex",
        "path": "./syntaxes/latex.tmLanguage"
      }
    ],
    "snippets": [
      {
        "language": "latex",
        "path": "./snippets/snippets.json"
      }
    ]
  }
}
```

Language Identifiers

In VS Code, each language mode has a unique specific language identifier. That identifier is rarely seen by the user except in the settings, e.g. when associating file extensions to a language:

```
"files.associations": {
  "*.myphp": "php"
}
```

Note that casing matters for exact identifier matching ('Markdown' != 'markdown')

The language identifier becomes essential for VS Code extension developers when adding new language capabilities or when replacing a language support.

Every language defines its *id* through the `languages` configuration point:

```
"languages": [{  
    "id": "java",  
    "extensions": [ ".java", ".jav" ],  
    "aliases": [ "Java", "java" ]  
}]
```

Language supports are added using the language identifier:

```
"grammars": [{  
    "language": "groovy",  
    "scopeName": "source.groovy",  
    "path": "./syntaxes/Groovy.tmLanguage"  
}],  
"snippets": [{  
    "language": "groovy",  
    "path": "./snippets/groovy.json"  
}]
```

New identifier guidelines

When defining a new language identifier, use the following guidelines:

- Use the lowercased programming language name.
- Search for other extensions in the Marketplace to find out if a language identifier has already been used.

You can find a list of known language identifiers in the [language identifier reference](#).

Next Steps

If you'd like to learn more about VS Code extensibility, try these topics:

- [Extending Visual Studio Code](#) - Learn about other ways to extend VS Code
- [Additional Extension Examples](#) - Take a look at our list of example extension projects.

Common Questions

Q: What parts of VS code can I theme with a custom color theme?

The VS Code color themes affect the editor input area (text foreground, background, selection, lineHighlight, caret, and the syntax tokens) as well as some of the custom UI (see the list in [Creating a Theme](#)). When contributing a theme, you also specify a base theme: light (`vs`), dark (`vs-dark`) and high contrast (`hc-black`). The base theme is used for all other areas in the workbench such as the File Explorer. Base themes are not customizable or contributable by extensions.

Q: Is there a list of scopes that I can use in my custom color theme?

VS Code themes are standard TextMate themes and the tokenizers used in VS code are well established TextMate tokenizers, mostly maintained by the community and in use in other products.

To learn about what scopes are used where, check out the [TextMate documentation](#) and this useful [blog post](#). A great place to examine themes is [here](#).

Q: I created a snippets extension but they aren't showing up in the VS Code editor?

A: Be sure you have correctly specified the `language` identifier for your snippet (e.g. `markdown` for Markdown .md files, `plaintext` for Plain Text .txt files). Also verify that the relative path to the snippets json file is correct.

Q: Can I add more file extensions to my colorizer?

A: Yes, the `yo code` generator provides the default file extensions from the `.tmLanguage` file but you can easily add more file extensions to a `languages` contribution `extensions` array. In the example below, the `.asp` file extension has been added to the default `.asa` file extension.

```
{
  "name": "asp",
  "version": "0.0.1",
  "engines": {
    "vscode": "0.10.x"
  },
  "publisher": "none",
  "contributes": {
    "languages": [
      {
        "id": "asp",
        "aliases": ["ASP", "asp"],
        "extensions": [".asa", ".asp"]
      }
    ],
    "grammars": [
      {
        "language": "asp",
        "scopeName": "source.asp",
        "path": "./syntaxes/asp.tmLanguage"
      }
    ]
  }
}
```

Q: Can I add more file extensions to an existing colorizer?

A: Yes. To extend an existing colorizer, you can associate a file extension to an existing language identifier with the `files.associations` setting. IntelliSense will show you the list of currently available language ids.

For example, the setting below adds the `.mmd` file extension to the `markdown` colorizer:

```
"files.associations": {
  "*.mmd": "markdown"
}
```

Q: What if I want to completely override an existing colorizer?

A: Yes. You override the colorizer by providing a new `grammars` element for an existing language id. Also, add a `extensionDependencies` attribute that contains the name of the extension that defines the grammar that you want to replace.

```
{  
  "name": "override-xml",  
  "version": "0.0.1",  
  "engines": {  
    "vscode": "0.10.x"  
  },  
  "publisher": "none",  
  "extensionDependencies": [  
    "xml"  
  ],  
  "contributes": {  
    "grammars": [{  
      "language": "xml",  
      "scopeName": "text.xml",  
      "path": "./syntaxes/BetterXML.tmLanguage"  
    }]  
  }  
}
```

Color Themes - 颜色主题

Color themes let you modify VS Code's background, text and language syntax colorization to suit your preferences and work environment. VS Code supports light, dark and high contrast themes.

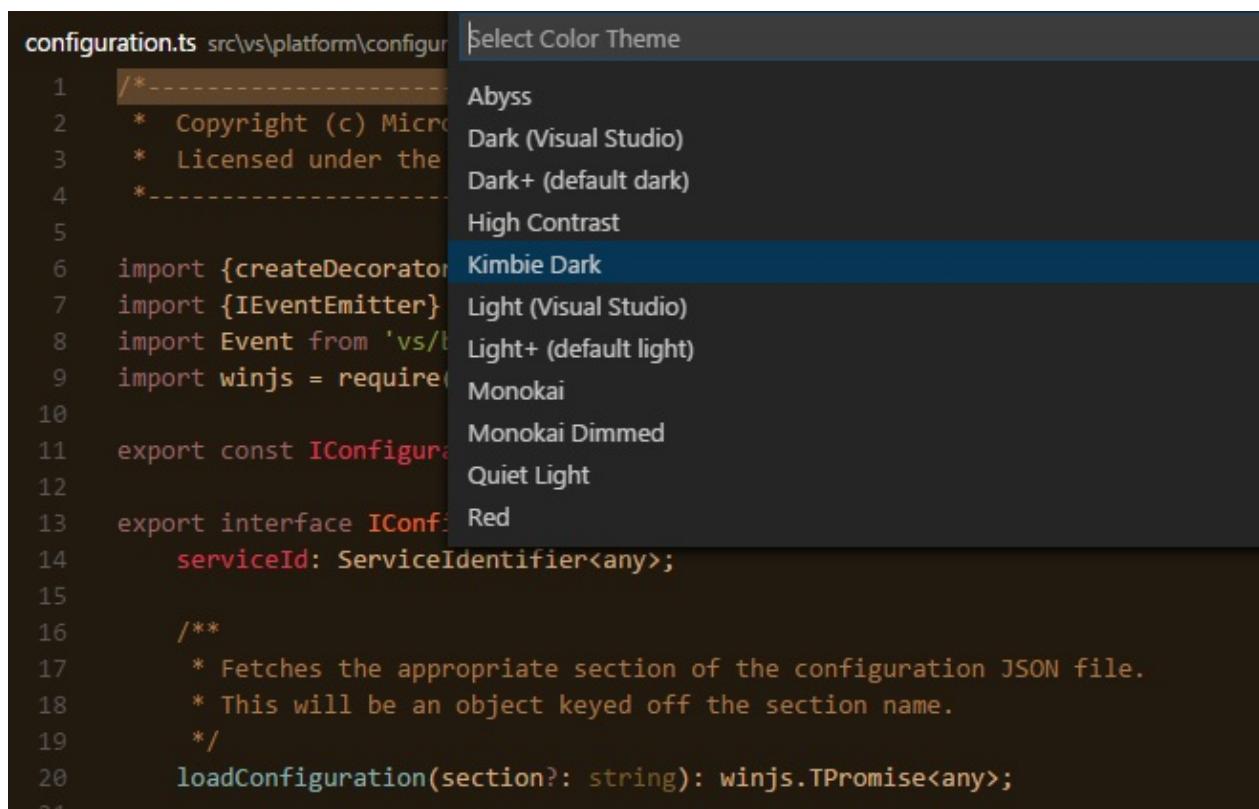
颜色主题允许你修改 VS Code 的背景颜色、文本颜色以及语言的语法着色，以此满足你的个人偏好和工作环境。VS Code 支持浅色、深色以及高对比度主题。

Selecting the Color Theme - 选择颜色主题

There are several out-of-the-box color themes in VS Code for you to try.

在 VS Code 中有大量的内置颜色主题供你选择。

1. Open the Color Theme picker with **File > Preferences > Color Theme**.
2. Use the cursor keys to preview the colors of the theme.
3. Select the theme you want and hit `kbstyle(Enter)`.
4. 通过文件 > 首选项 > 颜色主题 打开颜色主题选择器。
5. 使用光标键来预览各种主题的颜色。
6. 选中你想要的主题后敲击 `kbstyle(回车键)`.



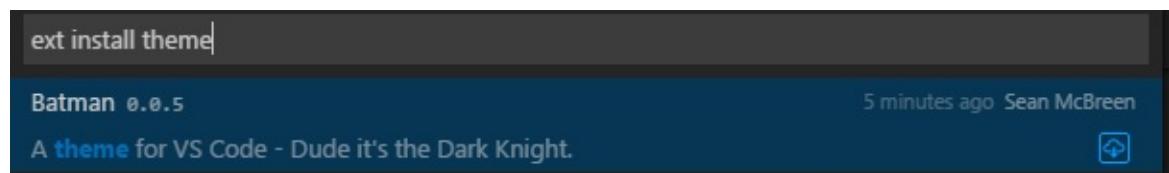
Adding Themes from the Extension Marketplace - 从扩展市场中添加主题

Many themes have been uploaded to the VS Code [Extension Marketplace](#) by the community. If you find one you want to use, simply install it and restart VS Code and the new theme will be available.

通过社区，我们向 VS Code 的 [扩展市场](#) 上传了大量的主题。如果在扩展市场中找到了一个心仪的主题，只需要安装该主题并重启 VS Code，之后就可以使用该主题。

Tip: To search for themes, type 'theme' in the `Extension: Install Extension` dropdown to filter on extensions with 'theme' in their name.

提示：为了寻找主题，在命令 `Extension: Install Extension` 中键入 'theme'，其下拉列表中可以筛选出名字带有 theme 的扩展插件。



You can also browse the [VS Code Marketplace](#) site directly to find available themes.

你也可以直接浏览 [VS Code 市场](#) 网址寻找可用的主题。

Adding a new Theme - 添加新主题

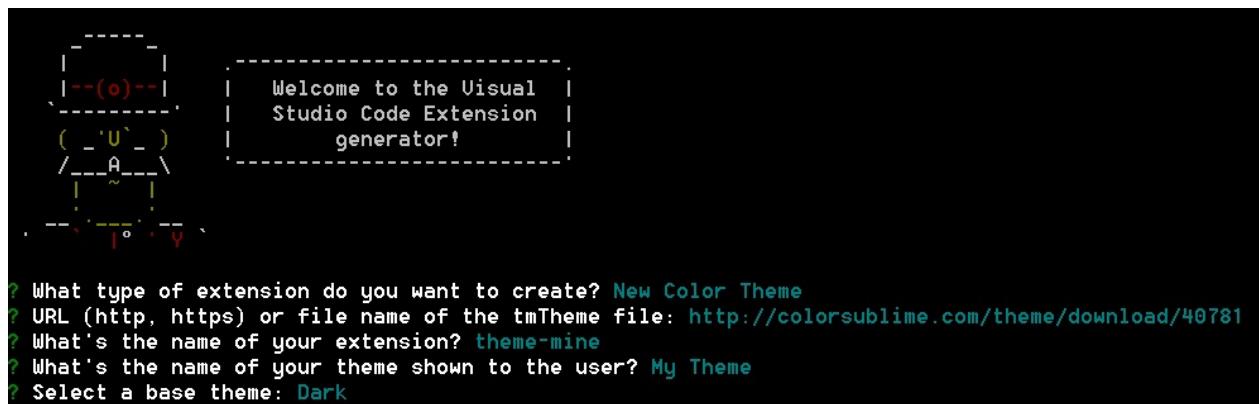
You can also add new TextMate theme files (.tmTheme) to your VS Code installation using the [yo code](#) extension generator.

通过使用 [yo code](#) 扩展生成器，你也可以向你的 VS Code 添加新的 TextMate 的主题文件 (.tmTheme)

[ColorSublime](#) has hundreds of existing TextMate themes to choose from. Pick a theme you like and copy the Download link to use in the Yeoman generator. It will be in a format like `"http://colorsublime.com/theme/download/(number)"`. The 'code' generator will prompt you for the URL or file location of the .tmTheme file, the theme name as well as other information for the theme.

[ColorSublime](#) 有数百个可供选择的 TextMate 主题。选择一个你喜欢的主题，然后把它的下载链接复制到 Yeoman 生成器中并使用。下载链接应类似于

`"http://colorsublime.com/theme/download/(number)"` 这样的形式。之后，“代码”生成器将提示关于该主题文件的 URL 或者文件路径，以及其他关于该主题的信息，比如主题的名称。

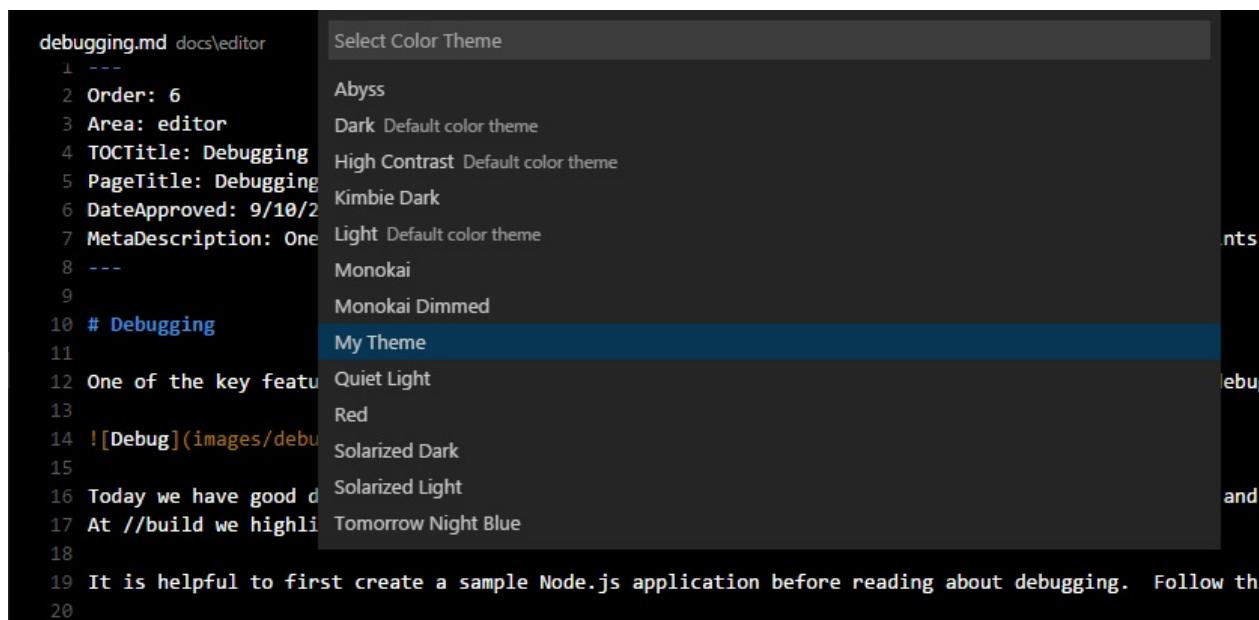


Copy the generated theme folder to a new folder under your `.vscode/extensions` folder and restart VS Code.

将产生的主题文件夹复制到 your `.vscode/extensions` folder 文件夹下并重启 VS Code。

Open the Color Theme picker theme with **File > Preferences > Color Theme** and you can see your theme in the dropdown. Arrow up and down to see a live preview of your theme.

使用文件 > 首选项 > 颜色主题 打开颜色主题选择器，你会发现在下拉列表中看到你新添加的主题。通过上下方向键可以实时预览你的主题。



Publishing a Theme to the Extension Marketplace - 将主题发布到扩展市场中

If you'd like to share your new theme with the community, you can publish it to the [Extension Marketplace](#). Use the [vsce publishing tool](#) to package your theme and publish it to the VS Code Marketplace.

如果你希望将你的新主题分享到整个社区，你可以将它发布到 [扩展市场](#) 中。利用 [vsce](#) 发布工具 打包你的主题并将它发布到 VS Code 市场中。

Tip: To make it easy for users to find your theme, include the word "theme" in the extension description and set the `Category` to `Theme` in your `package.json`.

提示：为了让用户更加简便的寻找主题，我们在扩展描述中添加 'theme' 的关键字，以及在你的 `package.json` 中将 `Category` 设置为 `Theme`。

We also have recommendations on how to make your extension look great on the VS Code Marketplace, see [Marketplace Presentation Tips](#).

关于在VS Code 市场中如何让你的扩展更加出色，我们也有一些建议，详细请看[市场介绍要点](#)。

Next Steps - 下一步

Themes are just one way to customize VS Code. If you'd like to learn more about VS Code extensibility, try these topics:

主题仅仅只是定制化 VS Code 的一步，如果你想了解更多的 VS Code 的可扩展性，请了解以下主题：

- [Colorizers and Bracket Matchers](#) - Learn how to import TextMate colorizers
- [Snippets](#) - Add additional snippets to your favorite language
- [Extending Visual Studio Code](#) - Learn about other ways to extend VS Code
- 调色板 - 了解如何导入 TextMate 调色板
- 用户定义代码段 - 向你喜欢的语言添加更多的用户定义代码段
- 扩展 VS Code - 了解扩展 VS Code 的其他方法

Common Questions - 常见问题

Nothing yet

暂无

Display Language - 语言区域

Visual Studio Code 附带 10 种可供显示的语言（区域设置）：英语（美国），简体中文，繁体中文，法语，德语，意大利语，日语，韩语，俄语和西班牙语。10 种语言均包含于 Visual Studio Code 本体当中，无需再次下载。

默认情况下，VS Code 将自动使用系统默认语言，如不支持当前系统语言将使用 英语（美国）作为默认语言。

Available Locales - 可选语言

语言	区域
英语（美国）	en-US
简体中文	zh-CN
繁体中文	zh-TW
法语	fr
德语	de
意大利语	it
日语	ja
韩语	ko
俄语	ru
西班牙语	es

Setting the Language - 语言设置

如果您想要切换语言，可以于启动 VS Code 时在命令行参数中添加 `--locale`，或者使用命令面板（F1）中的 配置语言（Configure Language）选项来切换为其他语言。

例：使用 `--locale` 命令行参数的方法将语言切换为法语

```
code . --locale=fr
```

配置语言（Configure Language）命令将会在您的 VS Code 用户目录中创建一个 `locale.json` 文件，并设置 `locale` 属性为您系统的首选语言。

例：使用命令面板方法将语言切换为繁体中文

```
{  
    // Defines VS Code's display language.  
    "locale": "zh-TW"  
}
```

注：修改 `locale` 值后需要重新启动 VS Code 方能生效。

工具

- [vse命令行工具](#)
- [yocode扩展生成器](#)
- [范例](#)

vsce - Publishing Tool Reference

vsce - 发布工具推荐

vsce is the command line tool you'll use to publish extensions to the [Extension Marketplace](#). You can also load extensions locally and share them via email or a UNC drive.

vsce是你用于在[Extension Marketplace](#)发布扩展的命令行工具。你也可以在本地加载扩展并通过邮件或UNC drive分享它们。

Installation

安装

Make sure you have [node.js](#) installed. Then simply run:

确保你已经安装了[node.js](#)。然后便可以简单的运行下方的命令：

```
npm install -g vsce
```

Usage

用法

You'll use the `vsce` command directly from the command line. For example, here's how you can quickly publish an extension:

你可以直接在命令行使用 `vsce` 命令。例如，你可以像这样快速发布一个扩展：

```
$ vsce publish
Publishing uuid@0.0.1...
Successfully published uuid@0.0.1!
```

For a reference on all the available commands, run `vsce --help`.

想了解所有可用命令，运行 `vsce --help` 即可。

Publishing Extensions

发布扩展

Visual Studio Code leverages [Visual Studio Team Services](#) for its Marketplace services. This means that authentication, hosting and management of extensions is provided through that service.

Visual Studio Code 使用 [Visual Studio Team Services](#) 为它的商店提供服务。这意味着可以通过这个服务验证、托管和管理扩展。

`vsce` can only publish extensions using [Personal Access Tokens](#). You need to create at least one in order to publish an extension.

`vsce` 只可以使用[Personal Access Tokens](#)进行发布。为了发布扩展，你至少需要创建一个 token。

Get a Personal Access Token

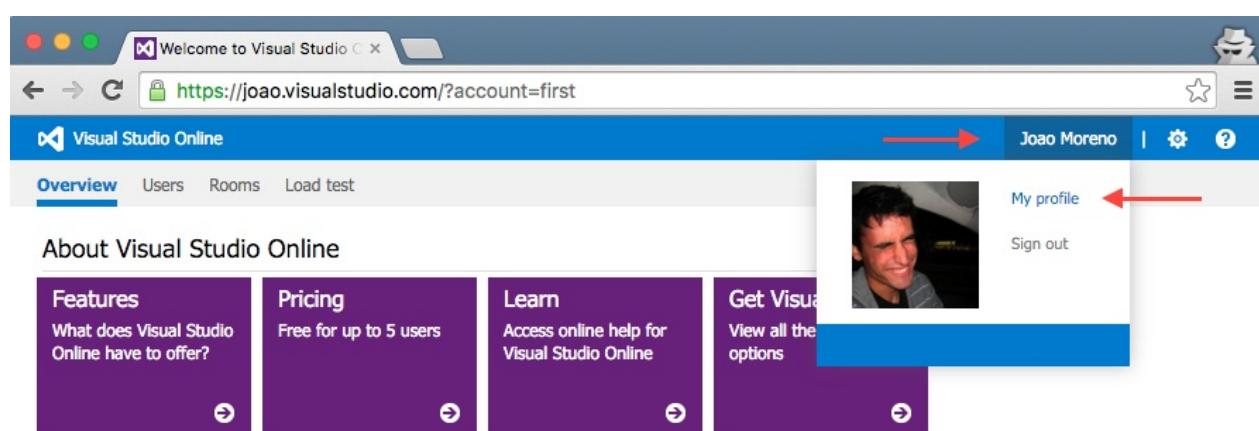
获取一个个人访问标识

First, login to or sign up for [Visual Studio Team Services](#).

首先，使用[Visual Studio Team Services](#)进行登录或注册。

Then, from your account's home page <https://ACCOUNT.visualstudio.com> , go to the **My Profile** page:

然后，从你的账户首页 <https://ACCOUNT.visualstudio.com>，进入**My Profile**页面：



Switch to the **Security** tab and **Add** a new Personal Access Token:

切换到**Security**标签并且**Add**一个新的Personal Access Token:

Personal access tokens

Personal access tokens can be used instead of a password to allow applications outside the browser access to the resources stored in your account.

Your tokens are like passwords. Keep them secret.

Description	Expiration	Status	Actions

Give the Personal Access Token a nice description, optionally extend its expiration date to 1 year and make it access every account:

给Personal Access Token添加一个nice的描述, 可以选择扩展它的过期时间最多到一年并且用它访问所有账号:

Create a personal access token

Applications that work outside the browser may require access to your projects. Generate personal access tokens for applications that require a username and password.

Description	vsce
Expires In	1 year
Accounts	All accessible accounts

Authorized Scopes

All scopes Selected scopes

<input type="checkbox"/> Build (read and execute)	<input type="checkbox"/> Build (read)
<input type="checkbox"/> Code (read and write)	<input type="checkbox"/> Code (read)
<input type="checkbox"/> Code (read, write, and manage)	<input type="checkbox"/> Team rooms (read and write)
<input type="checkbox"/> Team rooms (read, write, and manage)	<input type="checkbox"/> Test management (read and write)
<input type="checkbox"/> Test management (read)	<input type="checkbox"/> User profile (read)
<input type="checkbox"/> Work items (read and write)	<input type="checkbox"/> Work items (read)

The next screen will display your newly created Personal Access Token. **Copy** it, you'll need it to create a publisher.

下一个屏幕将展示一个新创建的Personal Access Token. **Copy** 它, 你需要创建一个publisher.

Create a Publisher

创建一个Publisher

A **publisher** is an identity who can publish extensions to the Visual Studio Code Marketplace. Every extension needs to include a `publisher` name in its `package.json` file. **publisher**用于验证是否能在Visual Studio Code Marketplace上发布扩展。每一个扩展需要在 `package.json` 文件中包含 `publisher` 名称。

Once you have a [Personal Access Token](#), you can create a new publisher using `vsce` :
一旦你有了一个[Personal Access Token](#), 你就可以使用 `vsce` 创建一个新的publisher:

```
vsce create-publisher (publisher name)
```

`vsce` will remember the provided Personal Access Token for future references to this publisher.

`vsce` 会保存这个publisher的Personal Access Token。

Login to a Publisher

登录Publisher

If you already created a publisher before and simply want to use it with `vsce` :
如果你在使用 `vsce` 之前就已经创建了一个publisher :

```
vsce login (publisher name)
```

Similarly to the `create-publisher` command, `vsce` will ask you for the Personal Access Token and remember it for future commands.

与 `create-publisher` 命令很相似, `vsce` 将会要求你提供Personal Access Token并为接下来的命令保存它。

You can also enter your Personal Access Token as you publish with an optional parameter `-p <token>` .

你也可以使用一个可选参数 `-p <token>` 输入你的Personal Access Token进行发布。

```
vsce publish -p <token>
```

Auto-incrementing the Extension Version

扩展版本号自动增长

You can auto-increment an extension's version number when you publish by specifying the [SemVer](#) compatible number to increment: `major`, `minor`, or `patch`.

当你通过指定[SemVer](#)兼容号码增加 `major`, `minor`, or `patch` 的时候，你的扩展版本号将自动递增。

For example, if you want to update an extension's version from 1.0.0 to 1.1.0, you would specify `minor`:

例如, 如果你想将扩展的版本从1.0.0更新到1.1.0, 你需要指定 `minor`:

```
vsce publish minor
```

This will modify the extension's `package.json` `version` attribute before publishing the extension.

在发布扩展之前, 这么做将会修改扩展的 `package.json` `version`的属性。

You can also specify a complete SemVer compatible version on the command line:

你也可以在命令行中指定一个完整的deSemVer兼容版本。

```
vsce publish 2.0.1
```

Packaging Extensions

包扩展

You may want to simply package extensions without publishing them to the store.

Extensions will always be packaged into a `.vsix` file. Here's how:

你可能想要简单的包扩展而不把他们发布到商店。扩展总是会被打包到 `.vsix` 文件。方法如下：

```
vsce package
```

This will package your extension into a `.vsix` file and place it in the current directory. It's possible to install `.vsix` files into Visual Studio Code. See [Installing Extensions](#) for more details.

这条命令会将你的扩展打包到 `.vsix` 文件中并放到当前目录下。它可能会安装 `.vsix` 文件到 Visual Studio Code. 查看[安装扩展](#)获取更多细节。

Advanced Usage

高级用法

Marketplace Integration

商店整合

You can customize how your extension looks in the Visual Studio Marketplace. See the [Go extension](#) for an example.

你可以自定义你的扩展在Visual Studio Marketplace中的样式。请查看[Go extension](#)这个例子。

Here are some tips for making your extension look great on the Marketplace:

这里有一些让你的扩展在商店中看起来更棒的小提示：

- Any `README.md` file at the root of your extension will be used to populate the extension's Marketplace page's contents. `vsce` can fix this for you in two different ways:
- 在你的扩展根目录下的 `README.md` 文件中的内容会显示在扩展商店页面上。`vsce` 可以使用两种不同的方式为你修复这个问题：
- Likewise, any `LICENSE` file at the root of your extension will be used as the contents for the extension's license.
- 同样的，你的扩展根目录下的 `LICENSE` 文件也被作为这个扩展的证书。
- If you add a `repository` field to your `package.json` and if it is a public GitHub repository, `vsce` will automatically detect it and adjust the links accordingly.
- 如果你添加一个 `repository` 字段到 `package.json` 文件中并且他是一个公开的GitHub仓库，`vsce` 将会自动检测到并且根据它来调整链接。
- You can override that behavior and/or set it by using the `--baseContentUrl` and `--baseImagesUrl` flags when running `vsce package`. Then publish the extension by passing the path to the packaged `.vsix` file as an argument to `vsce publish`.
- 在运行 `vsce package` 命令的时候，你可以通过使用 `--baseContentUrl` 和 `--baseImagesUrl` 参数覆盖或者设置这种行为。
- You can set the banner background color by setting `galleryBanner.color` to the intended hex value in `package.json`.
- 你可以在 `package.json` 中设置 `galleryBanner.color` 的16进制值来改变banner的背景色。
- You can set an icon by setting `icon` to a relative path to a squared `128px` PNG file included in your extension, in `package.json`.
- 你可以在 `package.json` 设置 `icon` 的值为一个在你的扩展目录中的边长 `128px` 的正方形PNG文件的相对路径来配置你的扩展图标。

Also see [Marketplace Presentation Tips](#).

也可以看这个商店演示提示.

.vscodeignore

You can create a `.vscodeignore` file to exclude some files from being included in your extension's package. This file is a collection of [glob](#) patterns, one per line.

你可以创建 `.vscodeignore` 文件来忽略掉你不想打包到扩展包的文件。这个文件是[glob](#)样式的集合，每行一个。

For example:

例子：

```
**/*.ts
**/tsconfig.json
!file.ts
```

You should ignore all files not needed at runtime. For example, if your extension is written in TypeScript, you should ignore all `**/*.ts` files, like in the previous example.

你应该忽略所有运行时不需要的文件。例如，如果你的扩展是用TypeScript写的，你应该忽略所有的 `**/*.ts` 文件，就就像上面的例子那样。

Note: Development dependencies listed in `devDependencies` will be automatically ignored, you don't need to add them to the `.vscodeignore` file.

注意：罗列在 `devDependencies` 中的开发依赖会被自动忽略，你不需要把他们添加到 `.vscodeignore` 文件中。

Pre-publish step

预发布步骤

It's possible to add a pre-publish step to your manifest file. The command will be called every time the extension is packaged.

添加预发布步骤到你的manifest文件中是可行的。扩展的每次打包都会调用到这个命令。

```
{
  "name": "uuid",
  "version": "0.0.1",
  "publisher": "joaomoreno",
  "engines": {
    "vscode": "0.10.x"
  },
  "scripts": {
    "vscode:prepublish": "tsc"
  }
}
```

This will always invoke the [TypeScript](#) compiler whenever the extension is packaged.
只要扩展被打包[TypeScript](#)编译器总是会被调用。

Next Steps

接下来的步骤

- [Extension Marketplace](#) - Learn more about VS Code's public extension Marketplace.
- 扩展商店 - 可以学习更多VSCode公共扩展相关内容的商店。
- [Installing Extensions](#) - Learn about other options for installing and sharing extensions.
- 安装扩展 - 学习安装和分享扩展等其他操作。

Common Questions

常见问题

Q: I get 403 Forbidden (or 401 Unauthorized) error when I try to publish my extension?

问题：当我尝试发布我的扩展的时候收到了**403**禁止的错误。

A: One easy mistake to make when creating the PAT (Personal Access Token) is to not select `all accessible accounts` in the Accounts field dropdown (instead selecting a specific account). You should also set the Authorized Scopes to `All scopes` for the publish to work.
回答：创建PAT (Personal Access Token)的时候在账户下拉框没有选择 `all accessible accounts` (而是选择了指定账户) 很容易造成这个错误。为了让发布能正常工作，你还应该将Authorized Scopes设置为 `All scopes`。

Q: I can't unpublish my extension through the `vsce` tool?

问题：我不能通过 `vsce` 工具取消发布我的扩展吗？

A: You may have changed your extension ID or publisher name. You can also manage your extensions directly on the Marketplace by going to the [manage page](#). You can update or unpublish your extension from your publisher manage page.

回答：你可能修改了你的扩展ID或者publisher名称。你也可以通过[manage page](#)来直接管理你的扩展。你可以从发布者管理页面更新或者取消发布你的扩展。

Yo Code - Extension Generator

Yo Code - 扩展生成器

We have written a Yeoman generator to help get you started.

我们写一个Yeoman生成器来作为开始。

Install the Generator

安装生成器

Install Yeoman and the VS Code Extension generator from the command prompt:

从命令行安装Yeoman和VSCode扩展生成器：

```
npm install -g yo generator-code
```

Run Yo Code

运行Yo Code

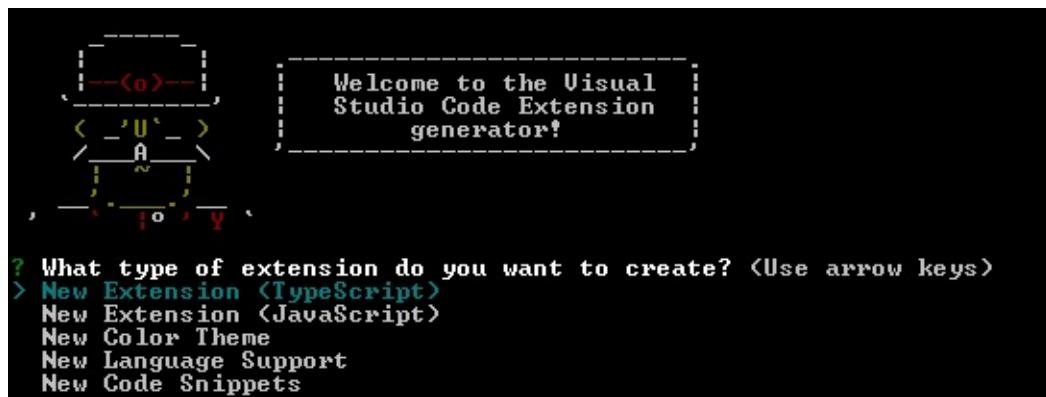
The Yeoman generator will walk you through the steps required to create your customization or extension prompting for the required information.

Yeoman生成器会要求你一步步的填写相应信息来创建自定义或扩展的提示。

To launch the generator simply type the following in a command prompt:

在命令行中输入如下命令来启动生成器：

```
yo code
```



Generator Options

生成器选项

The generator can either create an extension skeleton for a new extension or create a ready-to-use extension for languages, themes or snippets based on existing TextMate definition files.

生成器可以为新的扩展创建一个扩展框架或者基于已经存在的TextMate定义文件为语言、主题和片段创建一个准备使用的扩展，

New Extension in TypeScript

使用 TypeScript 写新扩展

Creates an extension skeleton implementing a 'hello world' command. Use this as a starting point for your own extension.

创建一个扩展框架实现'hello world'命令。用这个作为开发你自己的扩展的起点。

- Prompts for the extension identifier and will create a folder of that name in the current directory
提示在当前目录下创建一个以扩展标识为名称的文件夹
- Creates a base folder structure with a source, test and output folder
创建包含source、test和output文件夹的基础文件夹结构
- Templates out a `package.json` file and an extension main file
生成一份 `package.json` 模板和扩展主文件
- Sets-up `launch.json` and `tasks.json` so that F5 will compile and run your extension and attach the debugger
配置 `launch.json` 和 `tasks.json` 使得按F5就能编译和运行你的扩展，以及挂载调试器
- Optionally sets up a Git repository
可以选择配置一个Git仓库

Once created, open VS Code on the created folder. The folder contains a file `vsc-extension-quickstart.md` as a quick guide with the next steps. The extension is setup so that you get IntelliSense for the extension API.

创建成功后，打开这个被创建的文件夹。文件夹包含一个 `vsc-extension-quickstart.md` 文件作为后续步骤的快速指南。扩展会为你配置扩展API的智能提示。

New Extension in JavaScript

使用JavaScript写新扩展

Does the same as `New Extension (TypeScript)` , but for JavaScript. The extension is setup so that you get IntelliSense for the extension API.

和 `New Extension (TypeScript)` 一样，但是是为JavaScript写的。扩展会为你配置扩展API的智能提示。

New Color Theme

新的颜色主题

Creates an extension that contributes a new color theme based on an existing TextMate color theme.

基于已存在的TextMate颜色主题为扩展贡献一个新的颜色主题。

- Prompts for the location (URL or file path) of the existing TextMate color theme (`.tmTheme`). This file will be imported into the new extension.
- 提示输入现有的TextMate颜色主题(`.tmTheme`)的位置（URL或者文件路径）。
- Prompts for the color theme name as well as the color base theme (light or dark)
- 提示为这个基本的颜色主题(light or dark)输入名称。
- Prompts for the extension identifier and will create a folder of that name in the current directory
- 提示在当前目录下创建一个以扩展标识为名称的文件夹

Once created, open VS Code on the created folder and run the extension to test the new theme.

创建成功后，用VSCode打开这个文件夹并运行扩展来测试这个新主题。Check out `vsc-extension-quickstart.md` . It's a quick guide with the next steps.

查看 `vsc-extension-quickstart.md` 。它是后续步骤的快速指南。

New Language Support

新语言支持

Creates an extension that contributes a language with colorizer.

为带着色器的语言创建一个扩展。

- Prompts for the location (URL or file path) of an existing TextMate language file (.tmLanguage, .plist or .json). This file will be imported to the new extension
- 提示输入现有的TextMate语言文件(.tmLanguage, .plist or .json)的位置（URL或者文件路径）。这个文件将被导入到新的扩展
- Prompts for the extension identifier and will create a folder of that name in the current directory
- 提示在当前目录下创建一个以扩展标识为名称的文件夹

Once created, open VS Code on the created folder and run the extension to test the colorization. Check out `vsc-extension-quickstart.md` for the next steps. Have a look at the language configuration file that has been created and defines configuration options such what style of comments and brackets the language uses.

创建成功后，用VSCode打开这个文件夹并运行扩展来测试着色。查看 `vsc-extension-quickstart.md` 文件了解后续步骤。看看已经创建的语言配置文件，并定义语言使用什么样的注释和括号。

New Code Snippets

新代码段

Creates an extension that contributes new code snippets.

为新代码段创建一个扩展。

- Prompts for the folder location that contains TextMate snippets (.tmSnippet) or Sublime snippets (.sublime-snippet). These file are converted to a VS Code snippet file.
- 提示输入已有的TextMate片段(.tmSnippet)或者Sublime片段(.sublime-snippet)的位置。这些文件会被转换成VSCode代码段文件。
- Prompts for the language for which these snippets will be active
- 提示输入被激活的代码段的语言
- Prompts for the extension identifier and will create a folder of that name in the current directory
- 提示在当前目录下创建一个以扩展标识为名称的文件夹

Once created, open VS Code on the created folder and run the extension to test the snippets. Check out `vsc-extension-quickstart.md` for the next steps.

创建成功后，用VSCode打开这个文件夹并运行扩展来测试这个片段。查看 `vsc-extension-quickstart.md` 文件了解后续步骤

Loading an Extension

加载扩展

To load an extension, you need to copy the files to your VS Code extensions folder. We cover this in detail here: [Installing Extensions](#).

要加载扩展，你需要复制一些文件到你的VSCode扩展文件夹。 我们在这里进行详细介绍：[安装扩展](#)。

Next Steps

后续步骤

- [Publishing Tool](#) - Learn how to publish your extensions to the VS Code Marketplace
- [发布工具](#) - 学习如何将你的扩展发布到商店中
- [Hello World](#) - Try the 'Hello World' walkthrough to build your first extension * [Hello World](#)
- 尝试'Hello World'演练作为你的第一个扩展

Common Questions

常见问题

Q: The `yo code` generator doesn't respond to arrow keys on Windows 10.

问题：`yo code` 在Win10中不返回许可Keys。

A: Try starting the Yeoman generator with just `yo` and then select the `code` generator.

回答：尝试仅适用 `yo` 启动Yeoman生成器然后再选择 `code` 生成器。

```
C:\Users\gregvanl>yo
? 'Allo Greg! What would you like to do? (Use arrow keys)
Run a generator
> Code
_____
Update your generators
Install a generator
Find some help
Clear global config
Get me out of here!
```

VS Code Extension Samples

VS Code 扩展范例

This is a list to make it easy to see where all the samples are...

这里是一个便于查看所有范例的列表。

We have two walkthroughs that cover many of the core concepts - start with these:

我们从覆盖了许多核心概念的两个演练开始：

- [Your First Extension](#) - explains the core extensibility concepts with a walkthrough
• 你的第一个扩展 - 通过一个演练讲解核心扩展概念
- [Word Count Extension](#) - another walkthrough building on the last
• 单词计数扩展 - 建立在上个例子基础之上的另一个演练

Sample Extensions

扩展范例

Sample	Description	Type	In Marketplace
Word Count	Adds a word count to the status bar for Markdown files that updates on editing events. We have a walkthrough on how this was created .	Extension	Y
MDTools	Work with selections and update based on common text processing e.g. ToUpper, HTMLEncode, ...	Extension	Y
Decorator	Shows how to decorate editor text with a border, colors, and a custom cursor as well as add an overview ruler highlight.	Extension	N
TextDocumentProvider	Shows how to create virtual documents and preview them.	Extension	N
TSLint	Lint your TypeScript files based on TSLint	Language Server	Y
Spelling and Grammar Checker	Configurable Markdown spelling and grammar checker. Calls an external web service for checking and supports activation, add to dictionary, error mapping. Watches for config file changes in real time.	Extension	Y
Mock Debugger	Helps you build and test a debugger.	Debuggers	Y
Go Language Support	Rich language support for Go Lang - IntelliSense, Debug, Peek, Rename, Syntax, ...	Extension	Y

范例	描述	类型	在商店中
Word Count	在 editing 更新事件触发时，为 MarkDown 文件添加一个单词计数的状态栏。这里是 关于这个扩展是如何被创建的范例 。	Extension	Y
MDTools	普通文本处理选择和更新是如何工作的。ToUpper, HTMLEncode, ...	Extension	Y
Decorator	展示如何用边框、颜色和自定义光标或者高亮的规则总览装饰一个可编辑文本。	Extension	N
TextDocumentProvider	展示如何创建一个虚拟文档并预览它们。	Extension	N
TSLint	TSLint 实现 TypeScript 的语法提示。	Language Server	Y
Spelling and Grammar Checker	可配置的 MarkDown 拼写和语法检查器。调用额外的 Web 服务实现检测和支持激活，添加到字典，错误映射。实时检测配置文件变化。	Extension	Y
Mock Debugger	帮助你创建和测试一个调试器。	Debuggers	Y
Go Language Support	富语言支持 Go Lang - 智能提示，调试，选取，重命名，语法，...	Extension	Y

Tools to Help you build an Extension

帮助你创建扩展的工具

Tool	Purpose
Extension Generator	To help you getting started implementing an extension, we have a Yeoman generator. This creates all the initial settings you need for the development environment to work well and includes the API Typing files and any relevant modules. You can find the generator source here .
Debugging Extensions	We have worked hard to provide an easy way to develop, debug and locally test your extensions.
Publishing Tool	Once you have a working extension, it's time to share it in the extension Marketplace . We have a simple command line tool for this. You can find the source code here .

工具	目的
Extension Generator	帮助你开始实现一个扩展，我们有 Yeoman 生成器。创建你的开发环境所需的所有初始化配置，包括API类型文件和任意相关的模块。你可以从 这里 获取生成器源码。
Debugging Extensions	
Publishing Tool	一旦你有了一个可以正常工作的扩展，那么是时候在 扩展商店 分享它了。对此我们有一个简单的命令行工具。你可以从 这里 找到它的源码。

Runtime samples

运行时范例

- [Node.js](#)

Next Steps

接下来的步骤

- [Extension Marketplace](#) - Learn more about VS Code's public extension Marketplace.
- 扩展商店 - 可以学习更多VSCode公共扩展相关内容的商店。

Common Questions

常见问题

Nothing yet

现在还没有

技术支持

- 常见问题
- 错误代码
- 如何升级
- 系统要求

Visual Studio Code FAQ

Our docs contain a **Common Questions** section as needed for specific topics. We've captured items here that don't fit in the other topics.

If you don't see an answer to your question here, check our previously [reported issues](#) and our [Updates](#) notes.

What is the difference between VS Code and VS Community?

Visual Studio Code is a streamlined code editor with support for development operations like debugging, task running and version control. It aims to provide just the tools a developer needs for a quick code-build-debug cycle and leaves more complex workflows to fuller featured IDEs. For more details about the goals of VS Code, see [Why VS Code](#).

Which OS's are supported?

VS Code runs on Mac, Linux, and Windows. See [Requirements](#) for the supported versions.

Is VS Code free?

Yes, VS Code is a free, [open source](#) editor.

How big is VS Code?

VS Code is a small download (< 100 MB) and has a disk footprint of less than 200 MB, so you can quickly install VS Code and try it out.

How do I update to the latest version?

See [how to update](#). You'll find downloads for Linux (32-bit and 64-bit) and OS X, and both an installer and download for Windows.

How do I opt out of VS Code auto-updates?

By default, VS Code is set up to auto-update for OS X and Windows users when we release new updates. (Auto-update of VS Code is not supported for Linux.) If you do not want to get automatic updates, you can set the `update.channel` setting from the default `stable` to `none`.

To modify the update channel, go to **File > Preferences > User Settings** and add the `update.channel` setting with the value `"none"`.

```
"update.channel": "none"
```

You can install a previous release of VS Code by uninstalling your current version and then installing the download provided at the top of a specific release page under [Updates](#).

Licensing

Location

You can find the VS Code licenses, third party notices and [Chromium Open Source credit list](#) under your VS Code installation location `resources\app` folder. VS Code's `ThirdPartyNotices.txt`, Chromium's `Credits_*.html`, and VS Code's English language `LICENSE.txt` are available under `resources\app`. Localized versions of `LICENSE.txt` by Language ID are under `resources\app\licenses`.

Why does Visual Studio Code have a different license than the vscode GitHub repository?

To learn why Visual Studio Code, the product, has a different license than vscode, the open source [GitHub repository](#), see [issue #60](#) for a detailed explanation.

How can I test prerelease versions of VS Code?

Want get an early peek at new VS Code features? You can try prerelease versions of VS Code by installing the "Insiders" build. The Insiders build installs side by side to your stable VS Code install and has isolated settings, configurations and extensions. The Insiders build

will automatically update when we release new builds, towards the end of each month or whenever there is new functionality we'd like to get into the hands of developers early.

To install the Insiders build, go to the Insiders [download page](#).

Windows FAQ

Trouble with the installer

Try using the [zip file](#) instead of the installer. To use this, unzip VS Code in your **Program Files** folder.

Note: When VS Code is installed via a Zip you will need to manually update it for each release.

Icons are missing

I installed Visual Studio Code on my Windows 7 or 8 machine. Why are some icons not appearing in the workbench and editor?

VS Code uses [SVG](#) icons and we have found instances where the .SVG file extension is associated with something other than `image/svg+xml`. We're considering options to fix it, but for now here's a workaround:

Using the Command Prompt:

1. Open an Administrator Command Prompt.
2. Type `REG ADD HKCR\.svg /f /v "Content Type" /t REG_SZ /d image/svg+xml`.

Using the Registry Editor:

1. Start regedit.
2. Open the `HKEY_CLASSES_ROOT` key.
3. Find the `.svg` key.
4. Set its `Content Type` Data value to `image/svg+xml`.
5. Exit regedit.

Windows 7 - Error when deleting files from Explorer

19 Dec 2015 | version 0.10.5

When deleting a file from the VS Code Explorer on **Windows 7** using the 0.10.5 release, you may receive an error "Failed to move '*filename*' to the trash" which prompts you to "Delete Permanently", "Retry", or "Cancel".

By default, VS Code attempts to move the file to the Trash (Windows Recycle Bin). In the 0.10.5 release, there is an issue [3656](#) with the Electron Shell preventing this from working correctly.

You can choose to delete the file permanently or delete the file using the Windows Explorer or Command Prompt, which will properly move the file to the Windows Recycle Bin.

OS X FAQ

VS Code fails to start OmniSharp

On OS X, VS Code can no longer start OmniSharp after updating VS Code.

To fix this issue, run these commands to update mono:

```
brew update  
brew reinstall mono
```

Mono and El Capitan

Mono stopped working in Visual Studio Code after I installed OS X 10.11 El Capitan Public Beta. What do I do?

Run these commands:

```
brew update  
brew reinstall mono
```

Linux FAQ

Azure VM Issues

I'm getting a "Running without the SUID sandbox" error?

Unfortunately, this is a known issue that we're still investigating.

Debian and Moving Files to Trash

If you see an error when deleting files from the VS Code Explorer on the Debian operating system, it might be because the trash implementation that VS Code is using is not there.

Run these commands to solve this issue:

```
sudo apt-get install gvfs-bin
```

error ENOSPC

When you see this error, it indicates that the VS Code file watcher is running out of handles. The current limit can be viewed by running:

```
cat /proc/sys/fs/inotify/max_user_watches
```

The limit can be increased to its maximum by editing `/etc/sysctl.conf` and adding this line to the end of the file:

```
fs.inotify.max_user_watches=524288
```

The new value can then be loaded in by running `sudo sysctl -p`. Note that Arch Linux works a little differently, [view this page for advice](#).

While 524288 is the maximum number of files that can be watched, if you're in an environment that is particularly memory constrained, you may wish to lower the number. Each file watch [takes up 540 bytes \(32-bit\) or ~1kB \(64-bit\)](#), so assuming that all 524288 watches are consumed that results in an upperbound of around 256MB (32-bit) or 512MB (64-bit).

I can't see Chinese characters in Ubuntu

We're working on a fix. In the meantime, open the application menu, then choose **File > Preferences > User Settings**. Then set `editor.fontFamily` as shown:

```
"editor.fontFamily": "Droid Sans Mono, Droid Sans Fallback"
```

Proxy Server Support

If you work on a machine where Internet traffic needs to go through a proxy server, then configure the proxy server in one of the following ways:

- Set the operating system environment variables ‘`http.proxy`’ and ‘`https.proxy`’

```
SET http_proxy=http://10.203.0.1:5187/
```

- Configure the ‘http.proxy’ setting in your user settings (**File > Preferences > User Settings**)

```
"http.proxy": "http://10.203.0.1:5187/"
```

Additionally, use `"http.proxyStrictSSL": false` if your proxy server uses a self-signed certificate.

Note: VS Code supports http and https proxies, but not SOCKS proxies.

VS Code gets unresponsive right after opening a folder

When you open a folder, VS Code will search for typical project files to offer you additional tooling (e.g. the solution picker in the status bar to open a solution). If you open a folder with lots of files, the search can take a large amount of time and CPU resources during which VS Code might be slow to respond. We plan to improve this in the future but for now you can exclude folders from the explorer via settings and they will not be searched for project files:

```
"files.exclude": {
  "**/largeFolder": true
}
```

Missing csharp-o extension?

If you get an error at startup about a missing `csharp-o` extension, you can fix it by completely deleting its directory from the installation directory:

```
C:\Program Files (x86)\Microsoft VS Code\resources\app\extensions\csharp-o
```

How to disable crash reporting

From **File > Preferences > User Settings**, add the following option to disable crash reporting:

```
"telemetry.enableCrashReporter": false
```

Important Notice: This option requires a restart of VS Code to take effect.

How to disable telemetry reporting

VS Code collects usage data and sends it to Microsoft to help improve our products and services. Read our [privacy statement](#) to learn more.

If you don't wish to send usage data to Microsoft, you can set the `telemetry.enableTelemetry` setting to `false`.

From **File > Preferences > User Settings**, add the following option to disable telemetry reporting:

```
"telemetry.enableTelemetry": false
```

Note: VS Code gives you the option to install Microsoft and third party extensions. These extensions may be collecting their own usage data and are not controlled by the `telemetry.enableTelemetry` setting. Consult the specific extension's documentation to learn about its telemetry reporting.

Technical Support

You can ask questions and search for answers on [Stack Overflow](#), provide suggestions on [UserVoice](#), and enter issues directly in our [GitHub repository](#).

If you'd like to contact a professional support engineer, you can open a ticket with the [Microsoft assisted support team](#).

Common Error Cases

Some errors that happen in Visual Studio Code can be worked around or resolved by you. This topic describes several of the most common error conditions, and what you can do to resolve them.

If these steps don't help you, you probably hit a bug. You can check our [reported issues](#) list to see if others have had the same issue.

20001

Error: Cannot start OmniSharp because Mono version >=3.10.0 is required.

Currently, debugging support of VS Code on Linux or OS X requires Mono version 3.12 or later. If you intend to build ASP.NET Core applications with VS Code, we recommend to first follow the steps **Installing ASP.NET Core and DNX** in [ASP.NET Core Applications](#), since this will install a version of Mono that also supports debugging.

If you just want to try debugging support of VS Code, you can either download the latest Mono version for Linux or OS X at the [Mono Project Site](#), or you can use your package manager:

- On OS X: `brew install mono`
- On Linux: `sudo apt-get install mono-complete`

20002

Error: Cannot find '/usr/bin/gnome-terminal' for launching your Node.js program

On Linux the VS Code Node.js debugger requires the **gnome-terminal** for launching the Node.js program. If gnome-terminal is not installed, the VS Code debugger cannot launch your program for debugging.

There are two options for solving this problem:

- Install the gnome-terminal by running the command `sudo apt-get install gnome-terminal` (or the equivalent of your Linux distribution)
- Manually launch your program in debug mode by passing a `--debug` or `--debug-brk` option to Node.js and then attach the VS Code debugger to port 5858 on 'localhost'.

20003

Error: Attribute 'program' is not absolute; consider adding '\${workspaceRoot}' as a prefix to make it absolute.

Before VS Code release 0.10.11, it was possible to use relative paths in launch configurations. VS Code would silently convert them to absolute paths.

There were two problems with this:

- VS Code would only fix paths for some well-known attributes like `program`, `cwd`, or `outDir`. Relative paths passed as an argument or set as an environment variable would not be fixed and this behavior was not transparent.
- VS Code would only fix paths in the `launch.json` configuration file. It would not touch paths in `tasks.json` and this inconsistency was difficult to understand.

Starting with release 0.10.11, VS Code no longer modifies launch configuration paths. If you are using relative paths in your launch configurations, you'll need to fix them by prefixing the relative path with `${workspaceRoot}/`.

如何升级到最新版 How to update to the latest release

The following shows you how to update to the latest release of Visual Studio Code.

下面将向您介绍如何升级VS Code到最新版本。

Note: For Mac and Windows users, we have enabled the auto-update channel. If you're prompted by VS Code, accept the newest update and it will get installed (you won't need to do anything else to get the latest bits). If you'd rather control VS Code updates manually, see [How do I opt out of auto-updates](#).

注意：对于Mac和Windows的用户，我们已经启用了自动更新通道。如果你收到了来自VS Code的提示，接受最新版的更新，它就会被安装（你不需要做任何其他的事情）。

如果您愿意手动更新 VS Code，请参看 [How do I opt out of auto-updates](#)。

Auto-updates are not supported for Linux.

自动更新不支持Linux版本。

在Linux上更新 Updating on Linux

- 下载 VS Code的zip文件：[64-bit](#)或[32-bit](#).
- 打开zip文件，运行**Code**
- Download the VS Code zip file: [64-bit](#) or [32-bit](#).
- Open the zip and run **Code**

在 OS X 上更新 Updating on OS X

You need to do this only if auto-update did not complete.

除非自动更新没有完成，你才需要做的这些。

- 下载 VS Code的 [zip 文件](#).
- 打开zip文件，拖拽**Code** 到 **Applications**.
- 启动**Code**.
- Download the VS Code zip file from [here](#).
- Open the zip file and drag **Code** over to **Applications**.
- Launch **Code**.

在 Windows 上更新 Updating on Windows

You need to do this only if auto-update did not complete.

除非自动更新没有完成，你才需要做的这些。

Important: Close any running instances of VS Code before attempting to update (to avoid VS Code not being able to start after you update).

重要：更新之前，关闭所有 VS Code 的运行实例

- 从[此处](#)开始安装。
- 如果安装时遇到麻烦，请从[此处](#)下载VS Code的zip文件。
- Run the installer from [here](#).
- If you have trouble with the Windows installer, download the VS Code zip file from [here](#).

常见问题 Common Questions

Q: How do I know which version I'm running?

Q：我怎么知道我运行哪个版本？

A: In Linux and Windows, choose **Help, About**. In OS X, use **Code, About Visual Studio Code**.

A: 对于Linux和Windows，选择 帮助，关于。对于 OS X，使用 **Code, About Visual Studio Code**。

Q: What are the supported operating system versions needed to run Visual Studio Code?

Q: VS Code 支持 哪些操作系统版本？

A: See [Requirements](#) for the supported OS versions.

A: 对于所支持的操作系统，参考 [Requirements](#)

Requirements for Visual Studio Code

硬件 Hardware

Visual Studio Code is a small download (< 100 MB) and has a disk footprint of 200 MB. VS Code is lightweight and should easily run on today's hardware.

VS Code 的下载文件小于100MB。磁盘空间约200MB。VS Code是轻量级的，很容易在现今的硬件上运行的软件。

We recommend:

推荐：

- 1.6 GHz or faster processor
- 1 GB of RAM

平台 Platforms

VS Code has been tested on the following platforms:

VS Code在下面的平台上通过了测试：

- OS X Yosemite
- Windows 7 (with .NET Framework 4.5), 8.0, 8.1 and 10 (32-bit and 64-bit)
- Linux (Debian): Ubuntu Desktop 14.04, Debian 7
- Linux (Red Hat): Red Hat Enterprise Linux 7, CentOS 7, Fedora 23

对Windows的额外要求 Additional Windows requirements

.NET Framework 4.5 is required for VS Code. If you are using Windows 7, please make sure [.NET Framework 4.5](#) is installed.

VS Code需要.NET Framework 4.5。如果使用Windows 7,请确认已安装了[.NET Framework 4.5](#)

对Linux的额外要求 Additional Linux requirements

- GLIBCXX version 3.4.15 or later
- GLIBC version 2.15 or later

For a list of currently known issues, see our [FAQ](#).

当前已知问题列表：[FAQ](#).

扩展

- 概述
- 范例-hello-world
- 范例-word-count
- 范例-language-server
- 范例-调试器
- 调试-扩展
- 安装-扩展
- 范式-原则
- 测试-扩展
- 用我们的方法创造扩展

Visual Studio Code 扩展 Extending Visual Studio Code

如果你对扩展VS Code有兴趣，你就应该看看这篇文档。在这篇文档里将对VS Code的插件做一个概括性的描述，并且将教你快速的制作你的第一个vs Code插件。如果你对更深入的插件制作内容有兴趣，那么你可以阅读这篇 [文章](#).

If you are interested in extending VS Code, you are in the right place. Here we present an outline of the VS Code extensibility documentation and how to quickly build your first VS Code extension. If you're curious about our design approach to extensibility for VS Code, you can read about it [here](#).

如果你仅仅是想使用已经存在的插件，可以阅读这篇 [文档](#)，这篇文档将展示如何从VS Code插件商店里寻找并安装插件。

If you just want to use existing extensions, see the [Extension Marketplace](#) topic where we show you how to find and install extensions from the VS Code [Marketplace](#).

小贴士: 别忘了除了插件以外，我们还有很多种方法去 [定制](#) VS Code而不用写一行代码。包括添加[主题](#), [基础语言支持](#), 和 [用户定义代码段](#)。

Tip: Don't forget there are several ways to [customize](#) VS Code without writing an extension. This includes adding [Themes](#), [basic Language Support](#), and [Snippets](#) without writing a single line of code.

所有的VS Code插件在发布（注册），激活（加载）时共享使用公共的模块去访问VS Code插件API。值得一提的是两种特殊的VS Code插件，语言服务和调试器，他们有自己独有的额外协议，这些协议在他们单独的文档中说明。

1. [插件](#) - 基础构建模块
2. [语言服务](#) - 针对于CPU占用率较高和IO占用率较高的加强任务
3. [调试器](#) - 连接一个外部的调试器

All VS Code extensions share a common model of contribution (registration), activation (loading) and access to the VS Code extensibility API. There are however two special flavors of VS Code extensions, language servers and debuggers, which have their own additional protocols and are covered in their own sections of the documentation.

1. [Extensions](#) - the base building block
2. [Language Servers](#) - for high cost IO or CPU intensive tasks
3. [Debuggers](#) - wire up an external debugger

插件 Extensions

所有的插件在激活时都运行在我们的一个共享的插件宿主进程。这种单独的进程模型能确保 VS Code一直保持灵敏的响应

All extensions when activated run in our shared extension host process. This separate process for extensions ensures that VS Code remains responsive through-out.

插件包含了以下组件的支持：

- 激活 - 当检测到指定的文件类型，或者指定的文件存在，或者通过命令面板或者键盘快捷键选中一条命令时加载插件
- 编辑器 - 用来处理编辑器的内容 - 读和控制文本, 使用选择区域
- 工作空间 - 访问打开的文件, 状态栏, 信息提示等
- 事件 - 连接编辑器的生命周期，类似：打开，关闭，修改等等
- 高级编辑器 - 为高级语言提供包括智能感知，预览，悬停，诊断以及更多的支持

Extensions include support for:

- **Activation** - load an extension when a specific file type is detected, when a specific file exists, or when a command is selected via the Command Palette or a key combination
- **Editor** - work with the editor's content - read and manipulate text, leverage selection(s)
- **Workspace** - access working files, the status bar, information messages and more
- **Eventing** - connect to the editor life-cycle events such as: open, close, change, and more
- **Evolved editing** - create providers for rich language support including IntelliSense, Peek, Hover, Diagnostics and much, much more

我们有两个手把手的例子演示来让你学习插件的基础开发

1. **Hello World** - 生成一个基础的插件, 理解插件的目录结构, 插件清单, 学习插件怎样激活，运行，调试和在本地安装。
2. **Word Count** - 学习基于指定的文件类型激活插件, 更新状态栏, 响应文本编辑器的修改, 当文件关闭的时候怎样去释放插件。

We have two end-to-end walkthroughs to get you going on extension basics:

1. **Hello World** - generate a basic extension, understand an extension's folder structure, the extension manifest, learn how activation works, run and debug your extension and install it locally.
2. **Word Count** - activate based on a specific file type, update the status bar, respond to changes in the text editor, and dispose your extension when moving off the file.

语言服务 Language Servers

语言服务将为你的插件创建一个专用的进程。当你的插件占用了很高的CPU或者IO以至于拖慢了其他插件时，这是一个很有用的设计。那些在所有的文件上做处理的任务通常都有这些特点，例如代码检查或者静态分析。

在这里可以找到更多关于[语言服务](#)的资料。

Language servers let you create a dedicated process for your extension. This is a useful design choice for your extension when your extension runs high cost CPU or IO intensive tasks which could slow other extensions. This is common for tasks that work across all files in a workspace e.g. linters or static analysis suites.

Find out more about [language servers](#).

调试器 Debuggers

创建一个调试服务用来连接一个任一语言的调试器到VS Code

在这里可以找到更多关于[集成调试器](#)的信息。

最简单的方法是通过插件[商店](#)去看已经发布的VS Code插件，你可以找到一些很有用的插件，安装他们并且了解他们来想办法扩展你自己开发环境的VS Code。

Connecting a debugger written for any language to VS Code is possible through the creation of a debug service.

Find out more about [integrating debuggers](#).

The easiest way to see VS Code extensions in action is via the [Extension Marketplace](#). You can browse for useful extensions, install them to try them out and get an idea how you might extend VS Code for your own development scenarios.

编写一个插件 Writing an Extension

你可以用TypeScript或者JavaScript来编写一个插件。VS Code提供了一个一流的插件开发体验，你可以用VS Code自己完成[开发, 构建, 运行, 测试 和 调试](#)的所有环节。

Extensions can be written in either TypeScript or JavaScript. VS Code offers a first class extension development experience where you can [develop, build, run, test and debug](#) all from within VS Code itself.

安装和分享 Install and Share

一旦你拥有了一个可以正常工作的插件，你可以[安装或者分享给其他人](#)。我们支持本地安装，私有分享，或者发布到公共的[插件商店](#)。

Once you have a working extension, you can [install it or share it with others](#). We support local installation, private sharing, or publishing to the public [Extension Marketplace](#).

测试插件 Testing Extensions

我们也对插件的[编写、运行和测试](#)有完美的支持。你可以很容易创建一个针对调用的所有VS Code API的集成测试，并且在一个运行中的VS Code实例中测试你的代码。

We also have great support for [writing and running tests](#) for your extension. You can easily create integration tests which call the VS Code APIs and test your code in a running VS Code instance.

下一步 Next Steps

- [你的第一个插件](#) - 试着去创建一个简单的Hello World插件
- [插件API](#) - 学习VS Code插件API
- [示例](#) - 你可以复习和构建的插件示例列表
- [Your First Extension](#) - Try creating a simple Hello World extension
- [Extension API](#) - Learn about the VS Code extensibility APIs
- [Samples](#) - A list of extension samples you can review and build

常见问题 Common Questions

无

Nothing yet

示例 - Hello World Example - Hello World

你的第一个插件 Your First Extension

这篇文档将指引你创建你的第一个插件("Hello World")，并解释基础的VS Code扩展的概念。

在这篇文档的指引下，首先你将会向VS Code中添加一个显示一条"Hello World"消息的插件。稍后你会通过和VS Code编辑器交互来查询用户现在选中的文本。

This document will take you through creating your first VS Code extension ("Hello World") and will explain the basic VS Code extensibility concepts.

In this walkthrough, you'll add a new command to VS Code which will display a simple "Hello World" message. Later in the walkthrough, you'll interact with the VS Code editor and query for the user's currently selected text.

准备 Prerequisites

你需要安装[node.js](#)并且确保在 `$PATH` 中可用。

You need [node.js](#) installed and available in your `$PATH`.

生成一个新的插件 Generate a New Extension

要想给你的VS Code添加功能，最简单的方法是添加一条命令。每条命令都会注册一个可以通过命令面板或者键盘快捷键调用的回调函数。

我们编写了一个Yeoman生成器来帮助你开始第一步。安装Yeoman和[Yeoman VS Code插件生成器](#)来生成一个新的插件开发框架。

```
npm install -g yo generator-code  
yo code
```

对于hello world插件，你可以选择创建一个[TypeScript](#)插件或者[JavaScript](#)插件，在这个例子中，我们选择[TypeScript](#)。

```
? What type of extension do you want to create? (Use arrow keys)
❯ New Extension (TypeScript)
New Extension (JavaScript)
New Color Theme
New Language Support
New Code Snippets
```

The simplest way to add your own functionality to VS Code is through adding a command. A command registers a callback function which can be invoked from the Command Palette or with a key binding.

We have written a Yeoman generator to help get you started. Install Yeoman and the [Yeoman VS Code Extension generator](#) and scaffold a new extension:

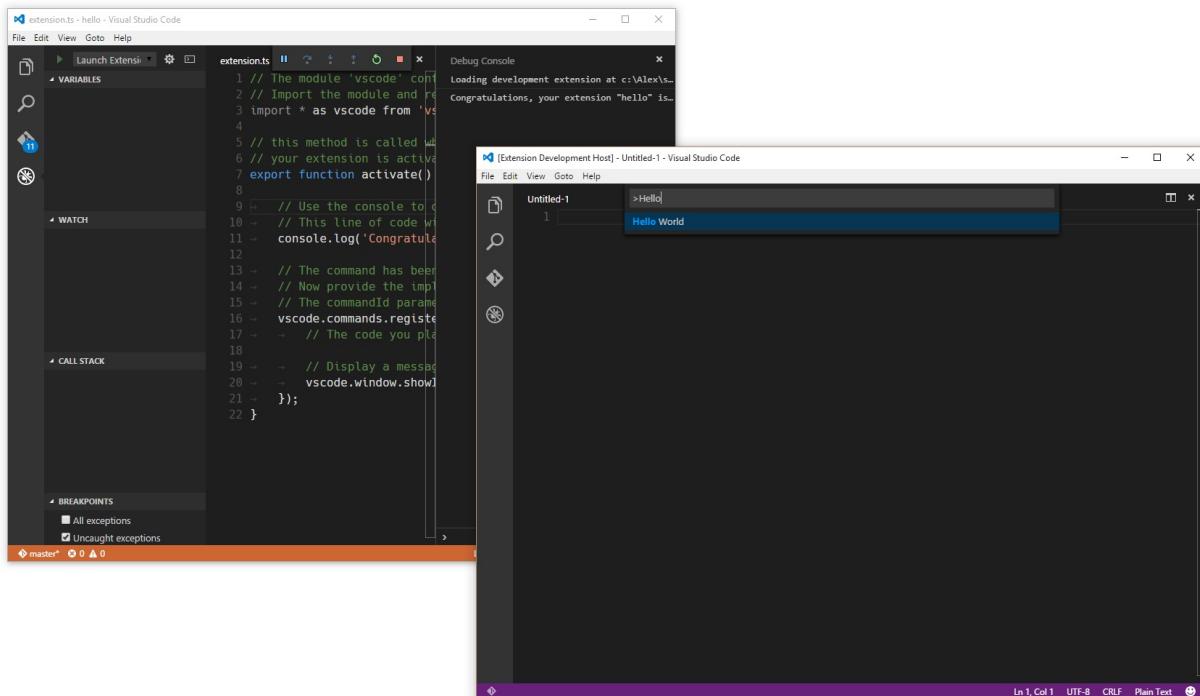
```
npm install -g yo generator-code  
yo code
```

For the hello world extension, you can either create a **TypeScript** extension or a **JavaScript** one. For this example, we pick a **TypeScript** extension.

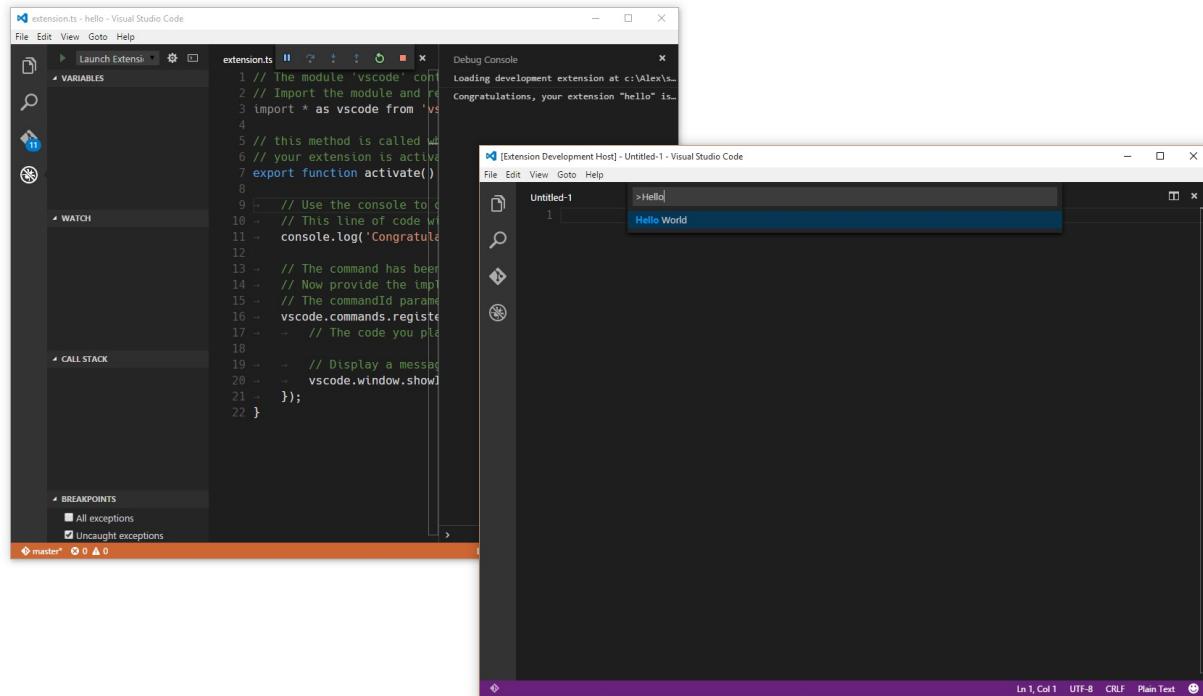
```
-----  
| --(o)-- |     Welcome to the Visual  
`-----|     Studio Code Extension  
( _ 'U' _ )     generator!  
/___A___\-----  
|   ~   |  
--' .-' .-' --  
. | ° | ° Y .  
  
? What type of extension do you want to create? (Use arrow keys)  
➤ New Extension (TypeScript)  
New Extension (JavaScript)  
New Color Theme  
New Language Support  
New Code Snippets
```

运行你的插件 Running your Extension

- 运行VS Code，选择 `文件 > 打开...` 然后选择你刚才生成的文件夹。
- 按下 `kb(workbench.action.debug.start)` 或者点击 `调试` 图标然后点击 `开始`。
- 一个新的VS Code实例将以一种特殊的模式(`Extension Development Host`)启动，这个新的实例是可以使用你的插件的。
- 按下 `kb(workbench.action.showCommands)` 然后运行名为 `Hello World` 的命令。
- 恭喜！至此你已经创建并且执行了你的第一个VS Code命令！



- Launch VS Code, choose `File > Open Folder` and pick the folder that you generated.
- Press `kb(workbench.action.debug.start)` or click on the `Debug` icon and click `Start`.
- A new instance of VS Code will start in a special mode (`Extension Development Host`) and **this new instance is now aware of your extension**.
- Press `kb(workbench.action.showCommands)` and run the command named `Hello World`.
- Congratulations! You've just created and executed your first VS Code command!



插件结构 The Structure of an Extension

在运行过后，生成的插件应该有如下的结构：

```
.  
├── .gitignore  
├── .vscode           // VS Code 集成配置  
│   ├── launch.json  
│   ├── settings.json  
│   └── tasks.json  
├── .vscodeignore  
└── README.md  
├── src                // 源码  
│   └── extension.ts    // 如果是JavaScript插件，那么此处就是extension.js  
├── test               // 测试文件夹  
│   ├── extension.test.ts // 如果是JavaScript插件，那么此处就是extension.test.js  
│   └── index.ts        // 如果是JavaScript插件，那么此处就是index.js  
├── node_modules  
│   ├── vscode          // 语言服务  
│   └── typescript       // typescript编译器(仅TypeScript插件才有)  
└── out                // 编译结果(仅TypeScript插件才有)  
    ├── src  
    │   ├── extension.js  
    │   └── extension.js.map  
    └── test  
        ├── extension.test.js  
        ├── extension.test.js.map  
        ├── index.js  
        └── index.js.map  
├── package.json        // 插件的清单  
├── tsconfig.json       // 如果是JavaScript插件，那么此处就是jsconfig.json  
└── typings             // 类型定义文件  
    ├── node.d.ts         // 链接到Node.js的API  
    └── vscode-typings.d.ts // 链接到VS Code的API  
└── vsc-extension-quickstart.md // 插件开发快速入门文档
```

接下来让我们深入了解所有这些文件的作用和用途：

After running, the generated extension should have the following structure:

```

.
├── .gitignore
├── .vscode
│   ├── launch.json          // VS Code integration
│   ├── settings.json
│   └── tasks.json
└── .vscodeignore
├── README.md
├── src
│   └── extension.ts         // extension.js, in case of JavaScript extension
├── test
│   ├── extension.test.ts    // extension.test.js, in case of JavaScript extension
│   └── index.ts              // index.js, in case of JavaScript extension
├── node_modules
│   ├── vscode               // language services
│   └── typescript            // compiler for typescript (TypeScript only)
└── out
    ├── src
    │   ├── extension.js
    │   └── extension.js.map
    └── test
        ├── extension.test.js
        ├── extension.test.js.map
        ├── index.js
        └── index.js.map
├── package.json             // extension's manifest
├── tsconfig.json            // jsconfig.json, in case of JavaScript extension
└── typings
    ├── node.d.ts              // link to Node.js APIs
    └── vscode-typings.d.ts     // link to VS Code APIs
└── vsc-extension-quickstart.md // extension development quick start

```

Let's go through the purpose of all these files and explain what they do:

插件清单 : `package.json` The extension manifest:

`package.json`

- 请阅读 `package.json` 插件清单参考
- 更多的信息请看这篇文档 `package.json` 扩展项
- 任何一个VS Code插件都必须有一个用来描述插件和插件功能的 `package.json` 文件。
- VS Code在启动的时候读取这个文件，此时所有定义在 `contributes` 节的内容将会生效。
- Please read the `package.json` extension manifest reference
- More information on `package.json` contribution points
- Each VS Code extension must have a `package.json` file that describes it and its capabilities.

- VS Code reads this file during start-up and reacts to each `contributes` section immediately.

TypeScript插件清单示例 Example TypeScript extension manifest

```
{
  "name": "myFirstExtension",
  "description": "",
  "version": "0.0.1",
  "publisher": "",
  "engines": {
    "vscode": "^0.10.1"
  },
  "categories": [
    "Other"
  ],
  "activationEvents": [
    "onCommand:extension.sayHello"
  ],
  "main": "./out/src/extension",
  "contributes": {
    "commands": [
      {
        "command": "extension.sayHello",
        "title": "Hello World"
      }
    ],
    "scripts": {
      "vscode:prepublish": "node ./node_modules/vscode/bin/compile",
      "compile": "node ./node_modules/vscode/bin/compile -watch -p ."
    },
    "devDependencies": {
      "typescript": "^1.7.5",
      "vscode": "^0.11.x"
    }
  }
}
```

注意：JavaScript插件不需要 `scripts` 成员，因为JavaScript插件不需要编译。

- 这个`package.json`文件描述了一个如下属性的插件：
 - 扩展了一条显示标签为 `"Hello world"`，实际调用 `"extension.sayHello"` 命令的命令面板条目。
 - 请求VS Code在 `"extension.sayHello"` 命令被调用时加载(*activationEvents*)。
 - 主要的JavaScript代码在 `"/./out/src/extension.js"` 文件中。

注意：VS Code不会在启动的时候就加载插件的代码。一个插件必须在 `activationEvents` 属性中描述什么条件下应该被激活（加载）。

```
{
  "name": "myFirstExtension",
  "description": "",
  "version": "0.0.1",
  "publisher": "",
  "engines": {
    "vscode": "^0.10.1"
  },
  "categories": [
    "Other"
  ],
  "activationEvents": [
    "onCommand:extension.sayHello"
  ],
  "main": "./out/src/extension",
  "contributes": {
    "commands": [
      {
        "command": "extension.sayHello",
        "title": "Hello World"
      }
    ],
    "scripts": {
      "vscode:prepublish": "node ./node_modules/vscode/bin/compile",
      "compile": "node ./node_modules/vscode/bin/compile -watch -p ."
    },
    "devDependencies": {
      "typescript": "^1.7.5",
      "vscode": "^0.11.x"
    }
  }
}
```

Note: A JavaScript extension doesn't require the `scripts` field as no compilation is needed.

- This specific package.json describes an extension that:
 - contributes an entry to the Command Palette (`kb(workbench.action.showCommands)`) with the label `"Hello world"` that will invoke a command `"extension.sayHello"`.
 - requests to get loaded (*activationEvents*) when the command `"extension.sayHello"` is invoked.
 - has its *main* JavaScript code in a file called `"/./out/src/extension.js"`.

Note: VS Code **does not** load the code of an extension eagerly at start-up. An extension must describe, through the `activationEvents` property under what conditions it should get activated (loaded).

生成代码 Generated Code

生成的插件代码在 `extension.ts` (如果是JavaScript插件就是 `extension.js`)文件中:

```

// 'vscode'模块包含了VS Code插件API
// 导入模块并且在下面你的代码中用vscode的别名引用这个模块
import * as vscode from 'vscode';

// 这个函数将在你的插件被激活时被调用
// 你的插件在第一次被执行命令的时候被激活
export function activate(context: vscode.ExtensionContext) {

    // 使用控制台去输出诊断信息(console.log)和错误信息(console.error)
    // 只有当你的插件被激活时才会执行下面这行代码
    console.log('Congratulations, your extension "my-first-extension" is now active!')

;

    // 这条命令被定义在package.json文件里
    // 现在使用registerCommand来提供这条命令的实现
    // commandId参数必须和package.json文件中的command成员匹配
    var disposable = vscode.commands.registerCommand('extension.sayHello', () => {
        // 每次命令被执行的时候都将执行你这里的代码

        // 向用户显示一个消息提示框
        vscode.window.showInformationMessage('Hello World!');
    });

    context.subscriptions.push(disposable);
}

```

- 每个插件都应该从他们的主文件中导出一个名为 `activate()` 函数，当在 `package.json` 文件中描述的 `activationEvents` 中的任何事件发生时，VS Code仅调用一次这个函数。
- 如果一个插件使用了操作系统的资源（例如分裂出一个进程），那么插件可以在主文件中导出一个名为 `deactivate()` 的函数，VS Code在关闭的时候将会调用这个函数来清理工作资源。
- 这个插件导入了 `vscode API`，注册了一个可以在VS Code里显示"Hello world"的命令，并且将函数的调用和 `"extension.sayHello"` 命令的调用联系起来。

注意：`package.json` 文件中的 `contributes` 节向命令面板里添加了一个条目。

`extension.ts/.js`里的代码定义了 `"extension.sayHello"` 命令的具体实现。

注意：对于TypeScript插件而言，生成的 `out/src/extension.js` 文件将被VS Code在运行时加载并执行。

The generated extension's code is in `extension.ts` (or `extension.js` in case of a JavaScript extension):

```
// The module 'vscode' contains the VS Code extensibility API
// Import the module and reference it with the alias vscode in your code below
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
    vscode.log('Congratulations, your extension "my-first-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
    var disposable = vscode.commands.registerCommand('extension.sayHello', () => {
        // The code you place here will be executed every time your command is executed

        // Display a message box to the user
        vscode.window.showInformationMessage('Hello World!');
    });

    context.subscriptions.push(disposable);
}

```

- Each extension should export from its main file a function named `activate()`, which VS Code will invoke **only once** when any of the `activationEvents` described in the `package.json` file occur.
- If an extension makes use of OS resources (e.g. spawns processes), the extension can export from its main file a function named `deactivate()` where it can do clean-up work and VS Code will invoke that function on shutdown.
- This specific extension imports the `vscode` API and then registers a command, associating a function to be called when the command `"extension.sayHello"` gets invoked. The command's implementation displays a "Hello world" message in VS Code.

Note: The `contributes` section of the `package.json` adds an entry to the Command Palette. The code in `extension.ts/.js` defines the implementation of `"extension.sayHello"`.

Note: For TypeScript extensions, the generated file `out/src/extension.js` will be loaded at runtime and executed by VS Code.

其他文件 Miscellaneous files

- `.vscode/launch.json` 定义了插件开发模式下去启动VS Code的行为。他也通过 `preLaunchTask` 来指定定义在 `.vscode/tasks.json` 文件中的用来运行TypeScript编译器的任务。
- `.vscode/settings.json` 插件默认设置不包含 `out` 文件夹。你可以在其中设置你想隐藏的文件类型。
- `.gitignore` - 告诉Git版本控制系统哪些类型文件可以忽略。
- `.vscodeignore` - 告诉打包工具档发布插件的时候哪些文件可以忽略。
- `README.md` - README文件向VS Code的用户描述了你的插件。
- `vsc-extension-quickstart.md` - 一份快速入门文档。
- `test/extension.test.ts` - 你可以将插件的单元测试写在这个文件里，并且运行测试用例(具体信息可以阅读这里[Testing Your Extension](#))
- `.vscode/launch.json` defines launching VS Code in the Extension Development mode. It also points with `preLaunchTask` to a task defined in `.vscode/tasks.json` that runs the TypeScript compiler.
- `.vscode/settings.json` by default excludes the `out` folder. You can modify which file types you want to hide.
- `.gitignore` - Tells Git version control which patterns to ignore.
- `.vscodeignore` - Tells the packaging tool which files to ignore when publishing the extension.
- `README.md` - README file describing your extension for VS Code users.
- `vsc-extension-quickstart.md` - A Quick Start guide for you.
- `test/extension.test.ts` - you can put your extension unit tests in here and run your tests against the VS Code API (see [Testing Your Extension](#))

插件激活 Extension Activation

现在已经将插件中各个文件的作用讲清楚了，接下来讲一下插件激活的过程：

- VS Code插件开发实例发现插件然后读取插件的 `package.json` 文件。
- 然后当你按下 `kb(workbench.action.showCommands)` 时：
 - 注册的命令被现实在命令面板里。
 - 在出现的命令列表中有一个我们在 `package.json` 文件中定义的 "Hello world" 条目。
- 当我们选择了 "Hello world" 条目时：
 - "`extension.sayHello`" 命令被调用：
 - 一个 "`onCommand:extension.sayHello`" 激活事件被创建了出来。
 - 所有在 `activationEvents` 成员中监听这个激活事件的插件将被激活。
 - `./out/src/extension.js` 文件被JavaScript虚拟机加载。
 - VS Code寻找其中的导出函数 `activate` 并且调用它。

- "extension.sayHello" 命令被注册并且定义了这条命令的具体实现。
 - "extension.sayHello" 命令的实现函数被调用。
 - 这条命令的具体实现显示一条"Hello World"消息。

Now that the roles of the files included in the extension are clarified, here is how your extension gets activated:

- The extension development instance discovers the extension and reads its `package.json` file.
- Later when you press `kb(workbench.action.showCommands)` :
 - The registered commands are displayed in the Command Palette.
 - In this list there is now an entry "Hello world" that is defined in the `package.json`.
- When selecting the "Hello world" command:
 - The command "extension.sayHello" is invoked:
 - An activation event "onCommand:extension.sayHello" is created.
 - All extensions listing this activation event in their `activationEvents` are activated.
 - The file at `./out/src/extension.js` gets loaded in the JavaScript VM.
 - VS Code looks for an exported function `activate` and calls it.
 - The command "extension.sayHello" is registered and its implementation is now defined.
 - The command "extension.sayHello" implementation function is invoked.
 - The command implementation displays the "Hello World" message.

调试你的插件 Debugging your Extension

Simply set a breakpoint, for example inside the registered command and run the "Hello world" command in the Extension Development VS Code instance.

```

extension.ts
1 // The module 'vscode' contains the VS Code extensibility API
2 // Import the module and reference it with the alias vscode in your
3 import * as vscode from 'vscode';
4
5 // this method is called when your extension is activated
6 // your extension is activated the very first time the command is e
7 export function activate() {
8
9    // Use the console to output diagnostic information (console.log)
10   // This line of code will only be executed once when your exten
11   console.log('Congratulations, your extension "hello" is now acti
12
13   // The command has been defined in the package.json file
14   // Now provide the implementation of the command with registerC
15   // The commandId parameter must match the command field in packa
16   vscode.commands.registerCommand('extension.sayHello', () => {
17       // The code you place here will be executed every time your
18       // Display a message box to the user
19       vscode.window.showInformationMessage('Hello World!');
20   });
21 }
22

```

注意：对于TypeScript插件来说，VS Code加载和运行的依然是 `out/src/extension.js` 文件，如果你真的需要调试原生的TypeScript代码，你应该生成一个源代码映射文件 `out/src/extension.js.map`，VS Code的调试器支持这种源代码映射。

小贴士 调试控制台将显示所有你向控制台输出的信息。

想了解更详细的关于调试的信息请阅读这篇文档[开发环境](#).

Simply set a breakpoint, for example inside the registered command and run the "Hello world" command in the Extension Development VS Code instance.

```

extension.ts
1 // The module 'vscode' contains the VS Code extensibility API
2 // Import the module and reference it with the alias vscode in your
3 import * as vscode from 'vscode';
4
5 // this method is called when your extension is activated
6 // your extension is activated the very first time the command is e
7 export function activate() {
8
9    // Use the console to output diagnostic information (console.log)
10   // This line of code will only be executed once when your exten
11   console.log('Congratulations, your extension "hello" is now acti
12
13   // The command has been defined in the package.json file
14   // Now provide the implementation of the command with registerC
15   // The commandId parameter must match the command field in packa
16   vscode.commands.registerCommand('extension.sayHello', () => {
17       // The code you place here will be executed every time your
18       // Display a message box to the user
19       vscode.window.showInformationMessage('Hello World!');
20   });
21 }
22

```

Note: For TypeScript extensions, even though VS Code loads and executes `out/src/extension.js`, you are actually able to debug the original TypeScript code due to the generated source map `out/src/extension.js.map` and VS Code's debugger support for source maps.

Tip: The Debug Console will show all the messages you log to the console.

To learn more about the extension [development environment](#).

简单的修改 A Simple Change

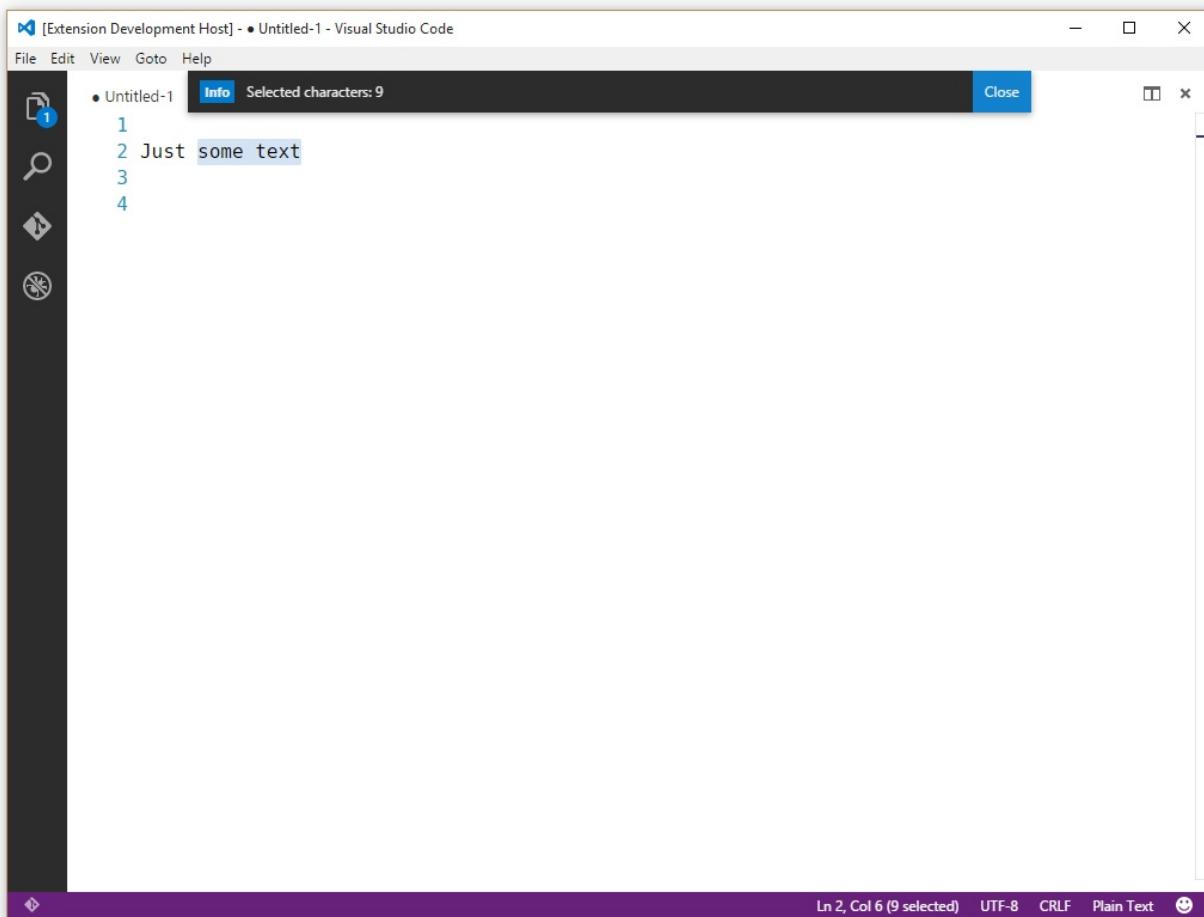
在 `extension.ts` (如果是JavaScript插件，那么此处就是`extension.js`)文件里，试着把 `extension.sayHello` 命令的实现修改为显示编辑器中选中的字符数：

```
var editor = vscode.window.activeTextEditor;
if (!editor) {
    return; // 没有打开的文件
}

var selection = editor.selection;
var text = editor.document.getText(selection);

// 向用户显示一个消息框
vscode.window.showInformationMessage('Selected characters: ' + text.length);
```

小贴士：一旦你修改了插件的源代码，你需要重新启动VS Code插件开发实例。你可以在VS Code开发实例里按下 `kbstyle(Ctrl+R)` (Mac: `kbstyle(Cmd+R)`)或者点击主VS Code实例上方的重新启动按钮来重新启动VS Code插件开发实例。



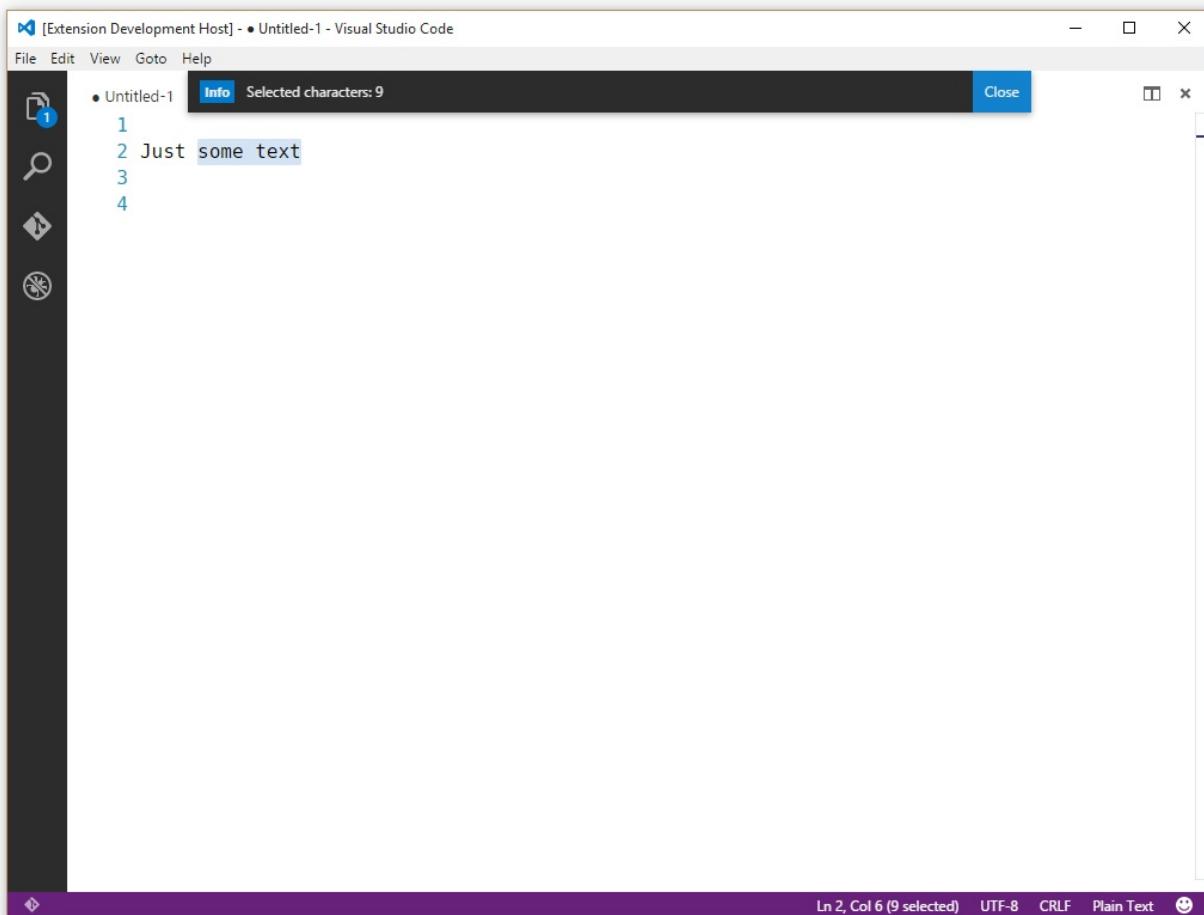
In `extension.ts` (or `extension.js`, in a JavaScript extension), try replacing the `extension.sayHello` command implementation to show the number of characters selected in the editor:

```
var editor = vscode.window.activeTextEditor;
if (!editor) {
    return; // No open text editor
}

var selection = editor.selection;
var text = editor.document.getText(selection);

// Display a message box to the user
vscode.window.showInformationMessage('Selected characters: ' + text.length);
```

Tip: Once you make changes to the extension source code, you need to restart the Extension Development instance of VS Code. You can do that by using `kbstyle(Ctrl+R)` (Mac: `kbstyle(Cmd+R)`) in the second instance or by clicking the Restart button at the top of your primary VS Code instance.



在本地安装你的插件 **Installing your Extension Locally**

到此为止，你写的插件还是仅仅运行在VS Code插件开发实例这个特殊的实例里，如果想在所有的VS Code实例里运行你的插件你需要将你的插件复制到你的本地插件目录下的一个新的文件夹：

- Windows: `%USERPROFILE%\.vscode\extensions`
- Mac/Linux: `$HOME/.vscode/extensions`

So far, the extension you have written only runs in a special instance of VS Code, the Extension Development instance. To get your extension running in all instances of VS Code, you need to copy it to a new folder under your local extensions folder:

- Windows: `%USERPROFILE%\.vscode\extensions`
- Mac/Linux: `$HOME/.vscode/extensions`

发布你的插件 **Publishing your Extension**

阅读怎样[共享插件](#)。

Read about how to [Share an Extension](#).

下一步 Next Steps

在这篇教程中，我们学习了一个非常简单的插件。这里有个更复杂的示例[单词数统计示例](#)展示了如何针对某一种特定的语言（Markdown）以及监听编辑器的内容修改事件。

如果你想了解更多的插件API的信息，可以试试阅读以下文档：

- [插件API概述](#) - 学习全面的VS Code插件扩展设计。
- [API模式和原理](#) - VS Code的扩展性是基于一些规范和原理的。
- [扩展项](#) - 详细的描述VS Code的各种扩展项。
- [激活事件](#) - VS Code的激活事件参考说明。

In this walkthrough, we've seen a very simple extension. For a more detailed example, see the [Word Count Example](#) which shows how to target a specific language (Markdown) and listen to the editor's document changed events.

If you'd like to read more generally about the extension APIs, try these topics:

- [Extension API Overview](#) - Learn about the full VS Code extensibility model.
- [API Patterns and Principles](#) - VS Code extensibility is based on several guiding patterns and principles.
- [Contribution Points](#) - Details about the various VS Code contribution points.
- [Activation Events](#) - VS Code activation events reference

常见问题 Common Questions

无

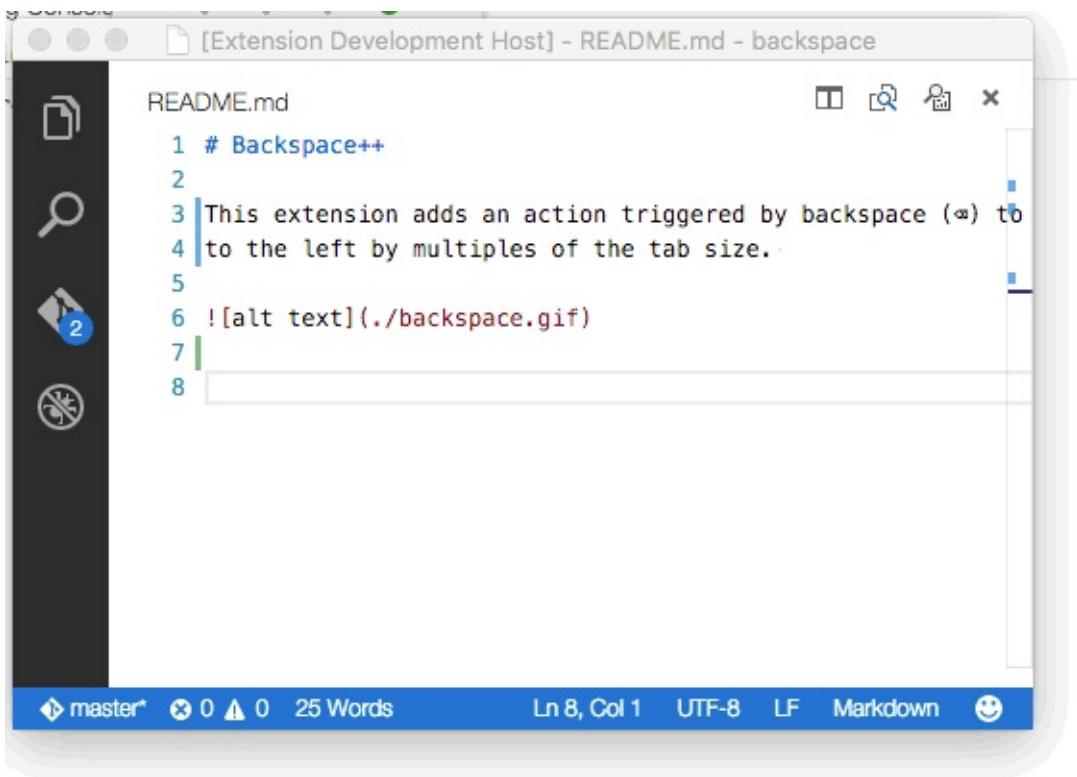
Nothing yet

示例 - 单词数统计 Example - Word Count

这篇文章假定你已经读了之前讲述VS Code插件基本知识的文章[你的第一个插件](#)。

单词数统计是一篇手把手教你怎样创建一个用来辅助编写Markdown的插件的教程，在你了解这个插件所有的运行细节前，让我们先看一下这个插件的核心功能的效果演示。

只要当 Markdown 文件正在被编辑，状态栏上就会添加文档单词数的信息。这个信息将随着你的键入或者打开其他文件而随之变化：

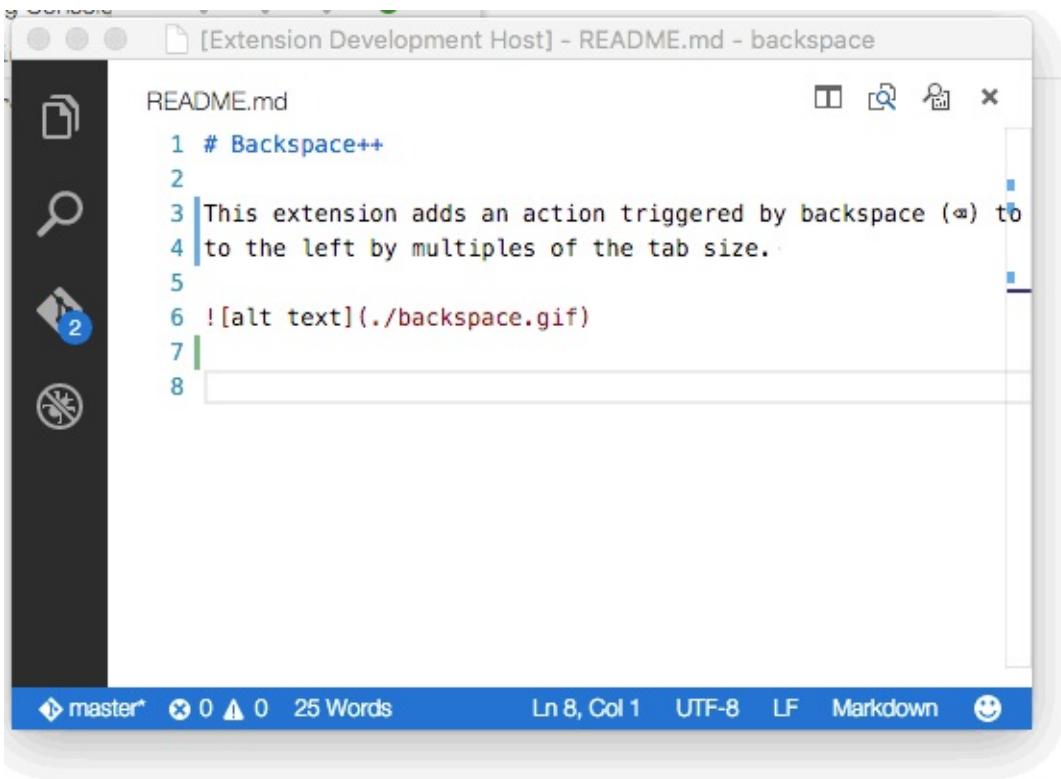


小贴士：这个完整的插件实例在[这个GitHub仓库](#)，如果你有任何的问题可以在这里提出来。

This document assumes you have read [Your First Extension](#) which covers the basics of VS Code extensibility.

Word Count is an end to end tutorial to show you how to create an extension to aid in Markdown authoring. Before we get into how all of this works, let's have a quick demo of the core features you will be building so you know what to expect.

Whenever a `Markdown` file is edited, a status bar message is added. The message includes the current word count and updates as you type and move from file to file:



Tip: The finished sample is available from [this GitHub repository](#) should you have any issues.

概述 Overview

这个例子将分三节来告诉你一系列相关联的概念：

1. [更新状态栏](#) - 在VS Code的 状态栏 上显示自定义的文字
2. [订阅事件通知](#) - 基于编辑器的事件更新 状态栏
3. [销毁插件资源](#) - 释放时事件订阅以及界面元素句柄之类的资源

首先确保你已经安装了最新的VS Code插件生成器并运行它：

```
npm install -g yo generator-code
yo code
```

这将会打开插件生成器 - 我们选择TypeScript New Extension 选项。现在同样简单地填写下方图片上完成填写的那些成员（使用'WordCount'作为插件的名字并且用你的名字作为发布者）。

```
? What type of extension do you want to create? New Extension <TypeScript>
? What's the name of your extension? WordCount
? What's the description of your extension? Markdown Word Count
? What's your publisher name? gregvanl
? Initialize a git repository? Yes
```

现在你可以在生成器描述的输出目录里打开VS Code：

```
cd WordCount
code .
```

This example has three sections which will take you through a set of related concepts:

1. [Update the Status Bar](#) - display custom text in the VS Code `Status Bar`
2. [Subscribing to Events](#) - updating the `Status Bar` based on editor events
3. [Disposing Extension Resources](#) - release resources like event subscriptions or UI handles

First make sure you have the latest VS Code extension generator installed then run it:

```
npm install -g yo generator-code
yo code
```

This will open up the extension generator - we will base this example on the TypeScript `New Extension` option. For now, simply fill in the fields the same way you see them completed in the image below (using 'WordCount' as the extension name and your own name as the publisher).

```
? What type of extension do you want to create? New Extension <TypeScript>
? What's the name of your extension? WordCount
? What's the description of your extension? Markdown Word Count
? What's your publisher name? gregvanl
? Initialize a git repository? Yes
```

You can now open VS Code as described in the generator output:

```
cd WordCount
code .
```

运行插件 Run the Extension

在我们继续下一步之前，我们可以先按下 `kb(workbench.action.debug.start)` 运行一下插件以确保到目前为止一切正常。如同你在之前"Hello World"教程中看到的那样，VS Code加载插件后打开另一个窗口（【扩展开发主机】窗口）。你应该在命令面板找到"Hello World"命令（按下 `kb(workbench.action.showCommands)`）并选择它，你将会在看到窗口顶部看到一个显示"Hello World"的消息提示框。

现在你已经确定了插件可以正常的运行，如果你喜欢你可以让插件开发窗口保持打开。接下来你可以在开发窗口中再次按下 `kb(workbench.action.debug.continue)` 或者按下 `kbstyle(Ctrl+R)` (Mac: `kbstyle(Cmd+R)`) 来重新加载插件开发窗口以确定你对插件的修改生效了。

Before we go on, we can run the extension to make sure everything works as expected by pressing `kb(workbench.action.debug.start)`. As you saw in the previous "Hello World" walkthrough, VS Code opens another window (the **[Extension Development Host]** window) in which your extension will be loaded. You should find the "Hello World" command in the Command Palette (press `kb(workbench.action.showCommands)`) and when you select it, you will see an information box at the top of the window saying "Hello World".

Now that you have confirmed that the extension is running properly, you can keep the extension development window open if you like. To test out any changes that you make to your extension, you can either press `kb(workbench.action.debug.continue)` again in the development window or reload the extension development window by pressing `kbstyle(Ctrl+R)` (Mac: `kbstyle(Cmd+R)`).

更新状态栏 Update the Status Bar

将生成的 `extension.ts` 文件内容替换成下面展示的代码。代码声明并实例化了一个可以统计单词数并且显示在VS Code状态栏上的 `WordCounter` 类。当"Hello Word"命令被调用的时候将调用 `updateWordCount` 函数。

```
// 'vscode'模块包括VS Code插件API
// 导入下面代码使用的必备的插件类型
import {window, commands, Disposable, ExtensionContext, StatusBarAlignment, StatusBarItem, TextDocument} from 'vscode';

// 这个函数将在你的插件被激活时被调用，激活是由定义在
```

```
// package.json文件中的activation events定义的
export function activate(context: ExtensionContext) {

    // 使用控制台去输出诊断信息(console.log)和错误信息(console.error)
    // 只有当你的插件被激活时才会执行下面这行代码
    console.log('Congratulations, your extension "WordCount" is now active!');

    // 创建一个新的单词数统计对象
    let wordCounter = new WordCounter();

    var disposable = commands.registerCommand('extension.sayHello', () => {
        wordCounter.updateWordCount();
    });

    // 添加到当插件关闭时被清理的可清理列表
    context.subscriptions.push(wordCounter);
    context.subscriptions.push(disposable);
}

class WordCounter {

    private _statusBarItem: StatusBarItem;

    public updateWordCount() {

        // 创建所需的状态栏元素
        if (!this._statusBarItem) {
            this._statusBarItem = window.createStatusBarItem(StatusBarAlignment.Left);
        }

        // 得到当前的文本编辑器
        let editor = window.activeTextEditor;
        if (!editor) {
            this._statusBarItem.hide();
            return;
        }

        let doc = editor.document;

        // 只有当是Markdown文件的时候才更新状态
        if (doc.languageId === "markdown") {
            let wordCount = this._getWordCount(doc);

            // 更新状态栏
            this._statusBarItem.text = wordCount !== 1 ? `${wordCount} Words` : '1 Word';
            this._statusBarItem.show();
        } else {
            this._statusBarItem.hide();
        }
    }

    public _getWordCount(doc: TextDocument): number {

```

```

let docContent = doc.getText();

// 去除多余的空格以方便与分割
docContent = docContent.replace(/(< ([^>]+)<)/g, '').replace(/\s+/g, ' ');
docContent = docContent.replace(/^\s\s*/, '').replace(/\s\s*$/, '');
let wordCount = 0;
if (docContent != "") {
    wordCount = docContent.split(" ").length;
}

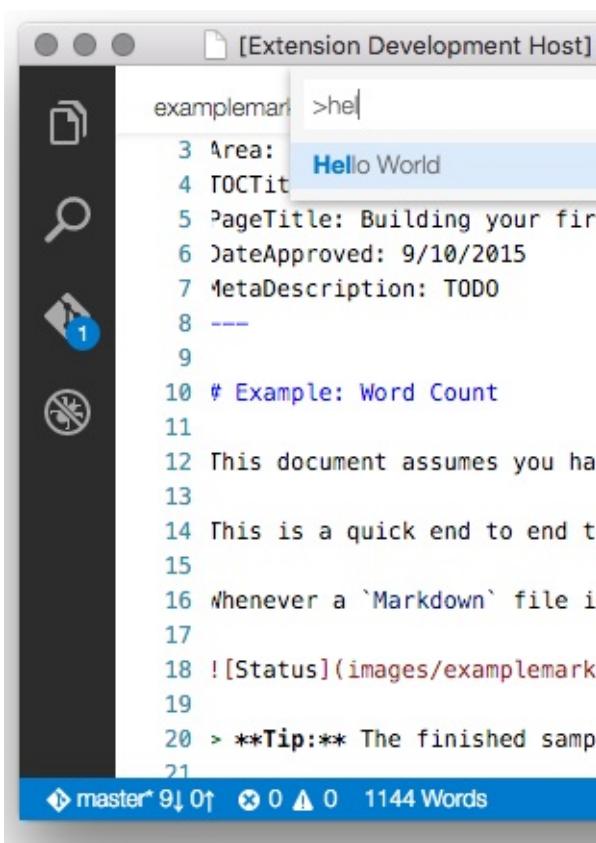
return wordCount;
}

dispose() {
    this._statusBarItem.dispose();
}
}

```

现在让我们试一下我们的修改有什么效果

我们定义了监视TypeScript文件修改并编译的任务(在插件的.vscode\tasks.json文件里)，所以我们无需手动重新构建。只需要在运行的代码的【扩展开发主机】窗口里简单的按下 kbstyle(Ctrl+R) 就会重新加载插件(你也可以在主开发窗口按 kb(workbench.action.debug.start))。我们依然需要像之前"Hello World"命令那样激活修改后的代码。如果你现在打开的是Markdown文件，你的状态栏上将会显示单词数统计。



现在我们已经有了一个好的开始，但是统计还不能随着文件内容的修改而变化。

Replace the contents of the generated `extension.ts` file with the code shown below. It declares and instantiates a `WordCounter` class which can count words and shows them in the VS Code Status Bar. The "Hello Word" command will call `updateWordCount` when invoked.

```
// The module 'vscode' contains the VS Code extensibility API
// Import the necessary extensibility types to use in your code below
import {window, commands, Disposable, ExtensionContext, StatusBarAlignment, StatusBarItem, TextDocument} from 'vscode';

// This method is called when your extension is activated. Activation is
// controlled by the activation events defined in package.json.
export function activate(context: 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 "WordCount" is now active!');

    // Create a new word counter
    let wordCounter = new WordCounter();

    var disposable = commands.registerCommand('extension.sayHello', () => {
        wordCounter.updateWordCount();
    });

    // Add to a list of disposables which are disposed when this extension is deactivated.
    context.subscriptions.push(wordCounter);
    context.subscriptions.push(disposable);
}

class WordCounter {

    private _statusBarItem: StatusBarItem;

    public updateWordCount() {

        // Create as needed
        if (!this._statusBarItem) {
            this._statusBarItem = window.createStatusBarItem(StatusBarAlignment.Left);
        }

        // Get the current text editor
        let editor = window.activeTextEditor;
        if (!editor) {
            this._statusBarItem.hide();
            return;
        }
    }
}
```

```

        let doc = editor.document;

        // Only update status if an MarkDown file
        if (doc.languageId === "markdown") {
            let wordCount = this._getWordCount(doc);

            // Update the status bar
            this._statusBarItem.text = wordCount !== 1 ? `${wordCount} Words` : '1 Wor
d';
            this._statusBarItem.show();
        } else {
            this._statusBarItem.hide();
        }
    }

    public _getWordCount(doc: TextDocument): number {

        let docContent = doc.getText();

        // Parse out unwanted whitespace so the split is accurate
        docContent = docContent.replace(/(< ([^>]+)<)/g, '').replace(/\s+/g, ' ');
        docContent = docContent.replace(/^\s\s*/, '').replace(/\s\s*$/, '');
        let wordCount = 0;
        if (docContent != "") {
            wordCount = docContent.split(" ").length;
        }

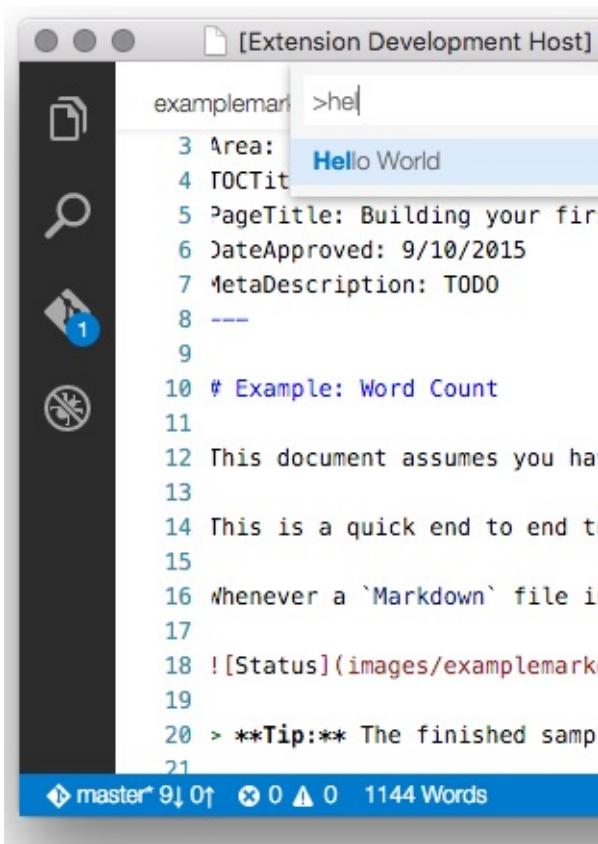
        return wordCount;
    }

    dispose() {
        this._statusBarItem.dispose();
    }
}

```

Now let's try our updates to the extension.

We have the compilation of the TypeScript file set on a watch (in the extension's `.vscode\tasks.json` file) so there is no need to re-build. Simply hit `kbstyle(Ctrl+R)` in the **[Extension Development Host]** window where your code is running and the extension will reload (you can also just `kb(workbench.action.debug.start)` from your primary development window). We still need to activate the code in the same way as before with the "Hello World" command. Assuming you are in a Markdown file, your Status Bar will display the word count.



This is a good start but it would be cooler if the count updated as your file changed.

订阅事件 **Subscribing to Events**

让我们用一个辅助类来监听这一系列的事件。

- `onDidChangeTextEditorSelection` - 当光标位置改变时触发这个事件
- `onDidChangeActiveTextEditor` - 当被激活的编辑器发生改变时触发这个事件

要完成这一切，我们需要向 `extension.ts` 文件里添加一个新类。这个类将订阅这些事件并且让 `WordCounter` 对象去更新单词统计数，同时注意这个类在自己被销毁时怎样去停止监听和怎样去管理这些订阅的销毁。

在 `extension.ts` 文件的底部加入下方显示的代码来添加 `WordCounterController` 类。

```

class WordCounterController {

    private _wordCounter: WordCounter;
    private _disposable: Disposable;

    constructor(wordCounter: WordCounter) {
        this._wordCounter = wordCounter;
        this._wordCounter.updateWordCount();

        // 订阅选择区域变化和编辑器激活事件
        let subscriptions: Disposable[] = [];
        window.onDidChangeTextEditorSelection(this._onEvent, this, subscriptions);
        window.onDidChangeActiveTextEditor(this._onEvent, this, subscriptions);

        // 更新当前文件的的单词数
        this._wordCounter.updateWordCount();

        // 从这些事件订阅中创建一个Disposable类型的集合
        this._disposable = Disposable.from(...subscriptions);
    }

    dispose() {
        this._disposable.dispose();
    }

    private _onEvent() {
        this._wordCounter.updateWordCount();
    }
}

```

现在我们不再想让单词数统计插件在调用命令时才被加载，而是在打开任何一个Markdown文件的时候就可以被加载。

首先，用下面的代码替换 `activate` 函数

```

// 使用控制台去输出诊断信息(console.log)和错误信息(console.error)
// 只有当你的插件被激活时才会执行下面这行代码
console.log('Congratulations, your extension "WordCount" is now active!');

// 创建一个新的单词数统计对象
let wordCounter = new WordCounter();
let controller = new WordCounterController(wordCounter);

// 添加到当插件关闭时被清理的可清理列表
context.subscriptions.push(controller);
context.subscriptions.push(wordCounter);

```

接下来我们需要确保当 `Markdown` 文件被打开时加载插件。为了实现这一点，我们需要修改 `package.json` 文件。之前我们的插件激活基于 `extension.sayHello` 命令，我们现在不需要了，所以我们可以从 `package.json` 文件里删除 `contributes` 属性的条目：

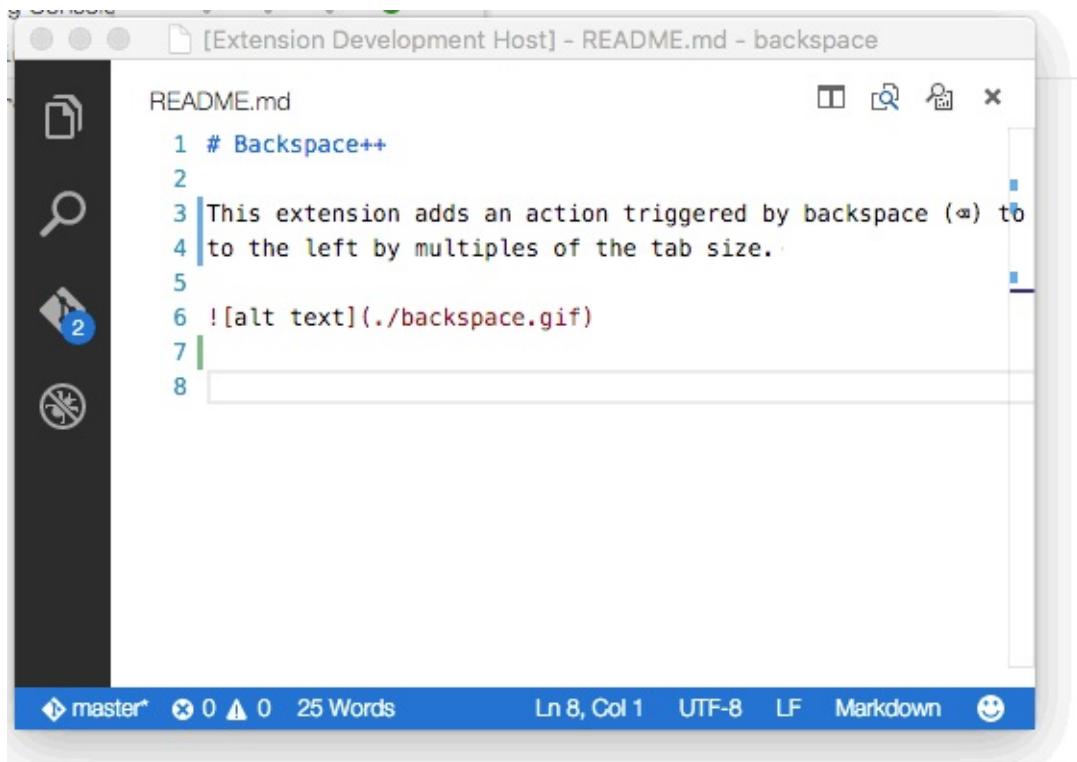
```
"contributes": {
  "commands": [
    {
      "command": "extension.sayHello",
      "title": "Hello World"
    }
  ]
},
```

现在来修改你的插件，通过更新 `activationEvents` 属性来实现当打开一个 `Markdown` 文件时激活插件：

```
"activationEvents": [
  "onLanguage:markdown"
]
```

这种 `onLanguage:${language}` 事件接受一个语言 id，此处是“`markdown`”，这个事件将在这种语言的文件被打开时产生、、

通过 `kbstyle(Ctrl+R)` 或者 `kb(workbench.action.debug.start)` 重新加载窗口来运行插件然后开始编辑一个 `Markdown` 文件。你现在应该已经拥有了一个实时的单词数统计。



如果你在 `activate` 函数里设置了断点，只有当第一个Markdown文件被打开时你会收到通知。那时 `WordCountController` 类的构造函数将会运行并且订阅编辑器的事件以便于在 Markdown文件被打开或者内容被修改时调用 `updateWordCount` 函数。

Let's hook your helper class up to a set of events.

- `onDidChangeTextEditorSelection` - Event is raised as the cursor position changes
- `onDidChangeActiveTextEditor` - Event is raised as the active editor changes.

To do this, we'll add a new class into the `extension.ts` file. It will set up subscriptions to those events and ask the `WordCounter` to update the word count. Also note how this class manages the subscription as `Disposable`s and how it stops listing when being disposed itself.

Add the `WordCounterController` as shown below to the bottom of the `extension.ts` file.

```
class WordCounterController {

    private _wordCounter: WordCounter;
    private _disposable: Disposable;

    constructor(wordCounter: WordCounter) {
        this._wordCounter = wordCounter;
        this._wordCounter.updateWordCount();

        // subscribe to selection change and editor activation events
        let subscriptions: Disposable[] = [];
        window.onDidChangeTextEditorSelection(this._onEvent, this, subscriptions);
        window.onDidChangeActiveTextEditor(this._onEvent, this, subscriptions);

        // update the counter for the current file
        this._wordCounter.updateWordCount();

        // create a combined disposable from both event subscriptions
        this._disposable = Disposable.from(...subscriptions);
    }

    dispose() {
        this._disposable.dispose();
    }

    private _onEvent() {
        this._wordCounter.updateWordCount();
    }
}
```

We no longer want the Word Count extension to be loaded when a command is invoked but instead be available for each *Markdown* file.

First, replace the body of the `activate` function with this:

```
// 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 "WordCount" is now active!');

// Create a new word counter
let wordCounter = new WordCounter();
let controller = new WordCounterController(wordCounter);

// Add to a list of disposables which are disposed when this extension is deactivated.
context.subscriptions.push(controller);
context.subscriptions.push(wordCounter);
```

Second, we must make sure the extension is activated upon the opening of a `Markdown` file. To do this, we'll need to modify the `package.json` file. Previously the extension was activated via the `extension.sayHello` command which we no longer need and so we can delete the entire `contributes` attribute from `package.json`:

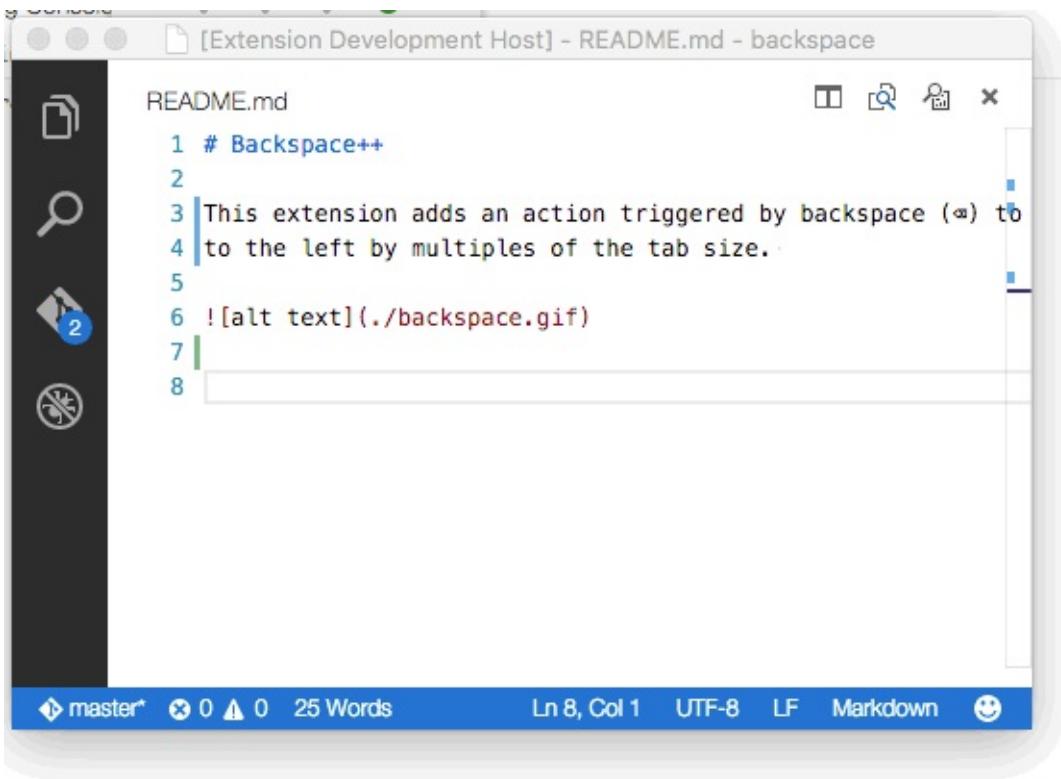
```
"contributes": {
  "commands":
    [<{
      "command": "extension.sayHello",
      "title": "Hello World"
    }>
  ]
},
```

Now change your extension so that it is activated upon the opening of a `Markdown` file by updating the `activationEvents` attribute to this:

```
"activationEvents": [
  "onLanguage:markdown"
]
```

The `onLanguage:${language}` event takes the language id, in this case "markdown", and will be raised whenever a file of that language is opened.

Run the extension by either doing a window reload `kbstyle(Ctrl+R)` or with `kb(workbench.action.debug.start)` and then start editing a Markdown file. You should now have a live updating Word Count.



If you set a breakpoint on the `activate` function, you'll notice that it is only called once when the first Markdown file is opened. The `WordCountController` constructor runs and subscribes to the editor events so that the `updateWordCount` function is called as Markdown files are opened and their text changes.

自定义的状态栏 Customizing the Status Bar

我们已经看到了怎样在状态栏上显示格式化的文字。VS Code允许你自定义状态栏，甚至进一步增加颜色、图标、工具提示等等。使用智能提示，你就可以看到很多种 `StartBarItem` 成员。另一个学习VS Code插件API的好方法是包含在你生成的插件项目里的 `vscode.d.ts` 类型文件，在编辑器里打开 `node_modules\vscode\vscode.d.ts` 文件，你就可以看到完整的VS Code插件API和注释。

```

vscode.d.ts node_modules\vscode
1343  /**
1344   * A status bar item is a status bar contribution that can
1345   * show text and icons and run a command on click.
1346   */
1347  export interface StatusBarItem {
1348
1349    /**
1350     * The alignment of this item, either left or right      //
1351     * @readonly
1352     */
1353    alignment: StatusBarAlignment;
1354
1355    /**
1356     * The text to show for the entry. You can embed icons in the text by leveraging the syntax:
1357     *
1358     * `My text ${icon name} contains icons like ${icon name} this one.`
1359     *
1360     * Where the icon name is taken from the octicon icon set (https://octicons.github.com/), e.g.
1361     * light-bulb, thumbsup or zap.
1362     */
1363    text: string;
1364

```

使用下面的代码来替换StatusBarItem：

```

// 更新状态栏
this._statusBarItem.text = wordCount !== 1 ? `$(pencil) ${wordCount} Words` : '$(p
encil) 1 Word';
this._statusBarItem.show();

```

在计算出来的单词数的左边显示一个GitHub Octicon上的铅笔图标。



We've seen how you can display formatted text on the Status Bar. VS Code allows you to customize your Status Bar additions even further with color, icons, tooltips and more. Using IntelliSense, you can see the various `StartBarItem` fields. Another great resource for learning about the VS Code extensibility APIs is the `vscode.d.ts` typings file included in your generated Extension project. Open `node_modules\vscode\vscode.d.ts` in the editor, you'll see the complete VS Code extensibility API with comments.

```
vscode.d.ts node_modules\vscode
1343  /**
1344   * A status bar item is a status bar contribution that can
1345   * show text and icons and run a command on click.
1346   */
1347  export interface StatusBarItem {
1348
1349    /**
1350     * The alignment of this item, either left or right      //
1351     * @readonly
1352     */
1353    alignment: StatusBarAlignment;
1354
1355    /**
1356     * The text to show for the entry. You can embed icons in the text by leveraging the syntax:
1357     *
1358     * `My text ${icon name} contains icons like ${icon name} this one.`
1359     *
1360     * Where the icon name is taken from the octicon icon set (https://octicons.github.com/), e.g.
1361     * light-bulb, thumbsup or zap.
1362     */
1363    text: string;
1364}
```

Replace the StatusBarItem update code with:

```
// Update the status bar
this._statusBarItem.text = wordCount !== 1 ? `$(pencil) ${wordCount} Words` : '$(pencil) 1 Word';
this._statusBarItem.show();
```

to display a GitHub Octicon pencil icon to the left of the calculated word count.



销毁插件资源 Disposing Extension Resources

我们将在[销毁](#)里更深入的而学习如何销毁插件。

一个插件在激活的时候传入一个有Disposable集合类型的subscriptions的ExtensionContext对象。你的插件可以向这个集合中添加自己的可销毁对象，VS Code将在插件关闭时销毁这些对象。

许多创建工作空间或者UI对象的VS Code的API(类似registerCommand)都会返回一个Disposable对象，插件可以调用这些销毁接口来从VS Code里移除这些元素。

Events are another example where `onDid*` event subscriber methods return a Disposable. Extensions unsubscribe to an event by disposing the event's Disposable. In our example, `wordCountController` handles the event subscription Disposables directly by keeping its own Disposable collection which it cleans up on deactivation.

```
// 订阅选择区域变化和编辑器激活事件
let subscriptions: Disposable[] = [];
window.onDidChangeTextEditorSelection(this._onEvent, this, subscriptions);
window.onDidChangeActiveTextEditor(this._onEvent, this, subscriptions);

// 从这些事件订阅中创建一个Disposable类型的集合
this._disposable = Disposable.from(...subscriptions);
```

Now we'll take a deeper look at how extensions should handle VS Code resources through [Disposables](#).

When an extension is activated, it is passed an `ExtensionContext` object which has a `subscriptions` collection of Disposables. Extensions can add their Disposable objects to this collection and VS Code will dispose of those objects when the extension is deactivated.

Many VS Code APIs which create workspace or UI objects (e.g. `registerCommand`) return a Disposable and extensions can remove these elements from VS Code by calling their `dispose` method directly.

Events are another example where `onDid*` event subscriber methods return a Disposable. Extensions unsubscribe to an event by disposing the event's Disposable. In our example, `WordCountController` handles the event subscription Disposables directly by keeping its own Disposable collection which it cleans up on deactivation.

```
// subscribe to selection change and editor activation events
let subscriptions: Disposable[] = [];
window.onDidChangeTextEditorSelection(this._onEvent, this, subscriptions);
window.onDidChangeActiveTextEditor(this._onEvent, this, subscriptions);

// create a combined disposable from both event subscriptions
this._disposable = Disposable.from(...subscriptions);
```

本地安装你的插件 **Installing your Extension Locally**

到目前为止，你写的插件运行在一个特殊的VS Code实例-插件开发主机实例。为了让你的插件对于所有的VS Code实例都能使用，复制你的插件目录内容到一个[你的 .vscode/extensions 目录](#)下的新目录。

So far, the extension you have written only runs in a special instance of VS Code, the Extension Development Host instance. To make your extension available to all VS Code instances, copy the extension folder contents to a new folder under [your .vscode/extensions folder](#).

发布你的插件 Publishing your Extension

阅读怎样[共享一个插件](#)。

Read about how to [Share an Extension](#).

下一步 Next Steps

阅读下面这些文档：

- [生成代码](#) - 学习 Yo Code 里的其他选项
- [插件API](#) - 查看插件 API 概述
- [定制化](#) - 这题，设置和键盘绑定
- [发布工具](#) - 学习怎样发布一个插件到公共市场里
- [编辑器API](#) - 学习更多的关于文本文档，文本编辑器，和编辑文本的知识

Read on to find out about:

- [Yo Code](#) - learn about other options in Yo Code
- [Extension API](#) - Get an overview of the Extension API
- [Customization](#) - Themes, settings and keyboard bindings
- [Publishing Tool](#) - Learn how to publish an extension to the public Marketplace
- [Editor API](#) - Learn more about Text Documents, Text Editors and editing text

常见问题 Common Questions

无

Nothing yet

示例 - 语言服务 Example - Language Server

语言服务允许你针对VS Code打开的文件添加一个的验证逻辑。最典型的一个应用是检查编程语言的语法。However validating other file types is useful as well. A language server could, for example, check files for inappropriate language.

通常校检查程语言的代价非常高，尤其是当检查动作需要解析多个文件并且建立抽象的语法树。为了避免性能问题，语言服务运行在一个单独的进程。这个特性也让使用除 TypeScript/JavaScript之外的语言编写语言服务成为可能，这可以支持一些很耗费性能的操作，例如代码自动完成或者 `Find All References`。

这篇文章假定你熟悉正常的VS Code插件开发。

Language servers allow you to add your own validation logic to files open in VS Code. Typically you just validate programming languages. However validating other file types is useful as well. A language server could, for example, check files for inappropriate language.

In general, validating programming language can be expensive. Especially when validation requires parsing multiple files and building up abstract syntax trees. To avoid that performance cost, language servers in VS Code are executed in a separate process. This architecture also makes it possible that language servers can be written in other languages besides TypeScript/JavaScript and that they can support expensive additional language features like code completion or `Find All References`.

The remaining document assumes that you are familiar with normal extension development for VS Code.

实现你自己的语言服务 Implement your own Language Server

理论上语言服务可以使用任何语言实现，然而现在VS Code只提供Node.js的库。未来将提供其他语言的库。一个使用Node.js语言实现的服务例子是这个示例仓库[Node语言服务示例](#)。

克隆这个仓库然后执行如下命令：

```
> cd client  
> npm install  
> code .  
> cd ../server  
> npm install  
> code .
```

Language servers can be implemented in any language. However, right now VS Code only provides libraries for Node.js. Additional libraries will follow in the future. A good starting point for a language server implementation in Node.js is the example repository [Language Server Node Example](#).

Clone the repository and then do:

```
> cd client
> npm install
> code .
> cd ../server
> npm install
> code .
```

‘客户端’解析 Explaining the ‘Client’

这个客户端实际上就是一个正常的VS Code插件。它的工作空间根目录包含 `package.json` 文件。这个文件有3个值得注意的节。

首先来看 `activationEvents` :

```
"activationEvents": [
    "onLanguage:plaintext"
]
```

这个节告诉VS Code当纯文本文件被打开时激活这个插件(例如后缀名为 `.txt` 的文件).

接下来看 `configuration` 节:

```
"configuration": {
    "type": "object",
    "title": "Example configuration",
    "properties": {
        "languageServerExample.maxNumberOfProblems": {
            "type": "number",
            "default": 100,
            "description": "Controls the maximum number of problems produced by the server."
        }
    }
}
```

这个扩展节是 `configuration` VS Code的配置。这个例子将解析这些设置是怎样在每次配置改变和语言服务启动时发送给语言服务的。

最后一部分添加了一个 `vscode-languageclient` 库的依赖：

```
"dependencies": {  
    "vscode-languageclient": "^1.4.2"  
}
```

如同上面提到的，这个客户端就是一个普通的VS Code插件。

下面就是相应的 `extension.ts` 文件内容：

```
/*
-----
* Copyright (c) Microsoft Corporation. All rights reserved.
* Licensed under the MIT License. See License.txt in the project root for license information.
-----
*/
'use strict';

import * as path from 'path';

import { workspace, Disposable, ExtensionContext } from 'vscode';
import { LanguageClient, LanguageClientOptions, SettingMonitor, ServerOptions } from 'vscode-languageclient';

export function activate(context: ExtensionContext) {

    // 服务使用nodejs实现
    let serverModule = context.asAbsolutePath(path.join('server', 'server.js'));
    // 服务的调试选项
    let debugOptions = { execArgv: ["--nolazy", "--debug=6004"] };

    // 服务的调试选项将在插件以调试模式启动的情况下生效
    // 运行选项将在其他情况下生效
    let serverOptions: ServerOptions = {
        run : { module: serverModule, transport: TransportKind.ipc },
        debug: { module: serverModule, transport: TransportKind.ipc, options: debugOptions }
    }

    // 控制语言客户端的选项
    let clientOptions: LanguageClientOptions = {
        // 为纯文本文档注册服务
        documentSelector: ['plaintext'],
        synchronize: {
            // 将'languageServerExample'选项同步到服务
            configurationSection: 'languageServerExample',
            // 当工作空间中的'.clientrc'文件改变时通知服务
            fileEvents: workspace.createFileSystemWatcher('**/.clientrc')
        }
    }

    // 创建一个语言客户端并启动这个客户端。
    let disposable = new LanguageClient('Language Server Example', serverOptions, clientOptions).start();

    // 向context的订阅里插入一个disposable对象使的客户端可以在插件
    // 关闭的时候被销毁
    context.subscriptions.push(disposable);
}
```

The client is actually a normal VS Code extension. It contains a `package.json` file in the root of the workspace folder. There are three interesting sections of that file.

First look the `activationEvents`:

```
"activationEvents": [
    "onLanguage:plaintext"
]
```

This section tells VS Code to activate the extension as soon as a plain text file is opened (e.g. a file with the extension `.txt`).

Next look at the `configuration` section:

```
"configuration": {
    "type": "object",
    "title": "Example configuration",
    "properties": {
        "languageServerExample.maxNumberOfProblems": {
            "type": "number",
            "default": 100,
            "description": "Controls the maximum number of problems produced by the se
rver."
        }
    }
}
```

This section contributes `configuration` settings to VS Code. The example will explain how these settings are sent over to the language server on startup and on every change of the settings.

The last part adds a dependency to the `vscode-languageclient` library:

```
"dependencies": {
    "vscode-languageclient": "^1.4.2"
}
```

As mentioned, the client is implemented as a normal VS Code extension.

Below is the content of the corresponding `extension.ts` file:

```

/*
-----
* Copyright (c) Microsoft Corporation. All rights reserved.
* Licensed under the MIT License. See License.txt in the project root for license information.
-----
*/
'use strict';

import * as path from 'path';

import { workspace, Disposable, ExtensionContext } from 'vscode';
import { LanguageClient, LanguageClientOptions, SettingMonitor, ServerOptions } from 'vscode-languageclient';

export function activate(context: ExtensionContext) {

    // The server is implemented in node
    let serverModule = context.asAbsolutePath(path.join('server', 'server.js'));
    // The debug options for the server
    let debugOptions = { execArgv: ["--nolazy", "--debug=6004"] };

    // If the extension is launch in debug mode the debug server options are used
    // Otherwise the run options are used
    let serverOptions: ServerOptions = {
        run : { module: serverModule, transport: TransportKind.ipc },
        debug: { module: serverModule, transport: TransportKind.ipc, options: debugOptions }
    }

    // Options to control the language client
    let clientOptions: LanguageClientOptions = {
        // Register the server for plain text documents
        documentSelector: ['plaintext'],
        synchronize: {
            // Synchronize the setting section 'languageServerExample' to the server
            configurationSection: 'languageServerExample',
            // Notify the server about file changes to '.clientrc files contain in the
            workspace
            fileEvents: workspace.createFilesystemWatcher('**/.clientrc')
        }
    }

    // Create the language client and start the client.
    let disposable = new LanguageClient('Language Server Example', serverOptions, clientOptions).start();

    // Push the disposable to the context's subscriptions so that the
    // client can be deactivated on extension deactivation
    context.subscriptions.push(disposable);
}

```

'服务'解析 Explaining the 'Server'

在这个例子里，服务也使用TypeScript语言实现并使用Node.js执行。因为VS Code已经提供了Node.js运行时，不需要你自己再提供，除非你对运行时有特殊的要求。

在服务的package.json文件中我们感兴趣的节是这个：

```
"dependencies": {  
    "vscode-languageserver": "^1.4.1"  
}
```

这将拉取vscode-languageserver库。

下面的代码是一个服务实现，这个服务实现了一个通过VS Code发送过全部文本内容来同步文档的简单的文本文档管理器。

```

/*
-----
* Copyright (c) Microsoft Corporation. All rights reserved.
* Licensed under the MIT License. See License.txt in the project root for license information.
-----
*/
'use strict';

import {
    createConnection, IConnection,
    TextDocuments, ITextDocument, Diagnostic,
    InitializeParams, InitializeResult
} from 'vscode-languageserver';

// 创建一个服务的连接，连接使用Node的IPC作为传输
let connection: IConnection = createConnection(new IPCMessageReader(process), new IPCMessageWriter(process));

// 创建一个简单的文本文档管理器，这个管理器仅仅支持同步所有文档
let documents: TextDocuments = new TextDocuments();
// 让文本文档管理器在连接上监听文本文档的打开、改变、关闭事件
documents.listen(connection);

// 在服务启动后，客户端发送初始化请求。服务收到的参数中包含工作空间的根目录和客户端的能力。
let workspaceRoot: string;
connection.onInitialize((params): InitializeResult => {
    workspaceRoot = params.rootPath;
    return {
        capabilities: {
            // 告诉客户端服务工作在全文本文档同步模式
            textDocumentSync: documents.syncKind
        }
    }
});

// 监听连接
connection.listen();

```

In the example, the server is also implemented in TypeScript and executed using Node.js. Since VS Code already ships with a Node.js runtime, there is no need to provide your own, unless you have very specific requirements for the runtime.

The interesting section in the server's package.json file is:

```

"dependencies": {
    "vscode-languageserver": "^1.4.1"
}

```

This pulls in the vscode-languageserver library.

Below is a server implementation that uses the provided simple text document manager which synchronizes text documents by always sending the file's full content from VS Code to the server.

```
/*
-----
 * Copyright (c) Microsoft Corporation. All rights reserved.
 * Licensed under the MIT License. See License.txt in the project root for license information.
 *
-----
'use strict';

import {
    createConnection, IConnection,
    TextDocuments, ITextDocument, Diagnostic,
    InitializeParams, InitializeResult
} from 'vscode-languageserver';

// Create a connection for the server. The connection uses Node's IPC as a transport
let connection: IConnection = createConnection(new IPCMessageReader(process), new IPCMessageWriter(process));

// Create a simple text document manager. The text document manager
// supports full document sync only
let documents: TextDocuments = new TextDocuments();
// Make the text document manager listen on the connection
// for open, change and close text document events
documents.listen(connection);

// After the server has started the client sends an initialize request. The server receives
// in the passed params the rootPath of the workspace plus the client capabilities.
let workspaceRoot: string;
connection.onInitialize((params): InitializeResult => {
    workspaceRoot = params.rootPath;
    return {
        capabilities: {
            // Tell the client that the server works in FULL text document sync mode
            textDocumentSync: documents.syncKind
        }
    }
});

// Listen on the connection
connection.listen();
```

完成一个简单的校验器 Adding a Simple Validation

想完成一个文档校验器，我们只需要简单的添加文本文档管理器的监听器去监听文本文档的改变。它会在最适合校验文档的时机通知服务。在这个实例实现里，服务校验纯文本文档并且通过消息拼写为 `TypeScript` 来标记所有的 `typescript` 文字。代码片段如下：

```
// The content of a text document has changed. This event is emitted
// when the text document first opened or when its content has changed.
documents.onDidChangeContent((change) => {
    let diagnostics: Diagnostic[] = [];
    let lines = change.document.getText().split(/\r?\n/g);
    lines.forEach((line, i) => {
        let index = line.indexOf('typescript');
        if (index >= 0) {
            diagnostics.push({
                severity: DiagnosticSeverity.Warning,
                range: {
                    start: { line: i, character: index},
                    end: { line: i, character: index + 10 }
                },
                message: `${line.substr(index, 10)} should be spelled TypeScript`,
                source: 'ex'
            });
        }
    })
    // Send the computed diagnostics to VS Code.
    connection.sendDiagnostics({ uri: change.document.uri, diagnostics });
});
```

To add document validation to the server, we simply add a listener to the text document manager that gets called whenever the content of a text document changes. It is then up to the server to decide when the best time is to validate a document. In the example implementation, the server validates the plain text document and flags all occurrences of `typescript` with a message to spell it `TypeScript`. The corresponding code snippet looks like this:

```
// The content of a text document has changed. This event is emitted
// when the text document first opened or when its content has changed.
documents.onDidChangeContent((change) => {
    let diagnostics: Diagnostic[] = [];
    let lines = change.document.getText().split(/\r?\n/g);
    lines.forEach((line, i) => {
        let index = line.indexOf('typescript');
        if (index >= 0) {
            diagnostics.push({
                severity: DiagnosticSeverity.Warning,
                range: {
                    start: { line: i, character: index},
                    end: { line: i, character: index + 10 }
                },
                message: `${line.substr(index, 10)} should be spelled TypeScript`,
                source: 'ex'
            });
        }
    })
    // Send the computed diagnostics to VS Code.
    connection.sendDiagnostics({ uri: change.document.uri, diagnostics });
});
```

诊断提示和窍门 **Diagnostics Tips and Tricks!**

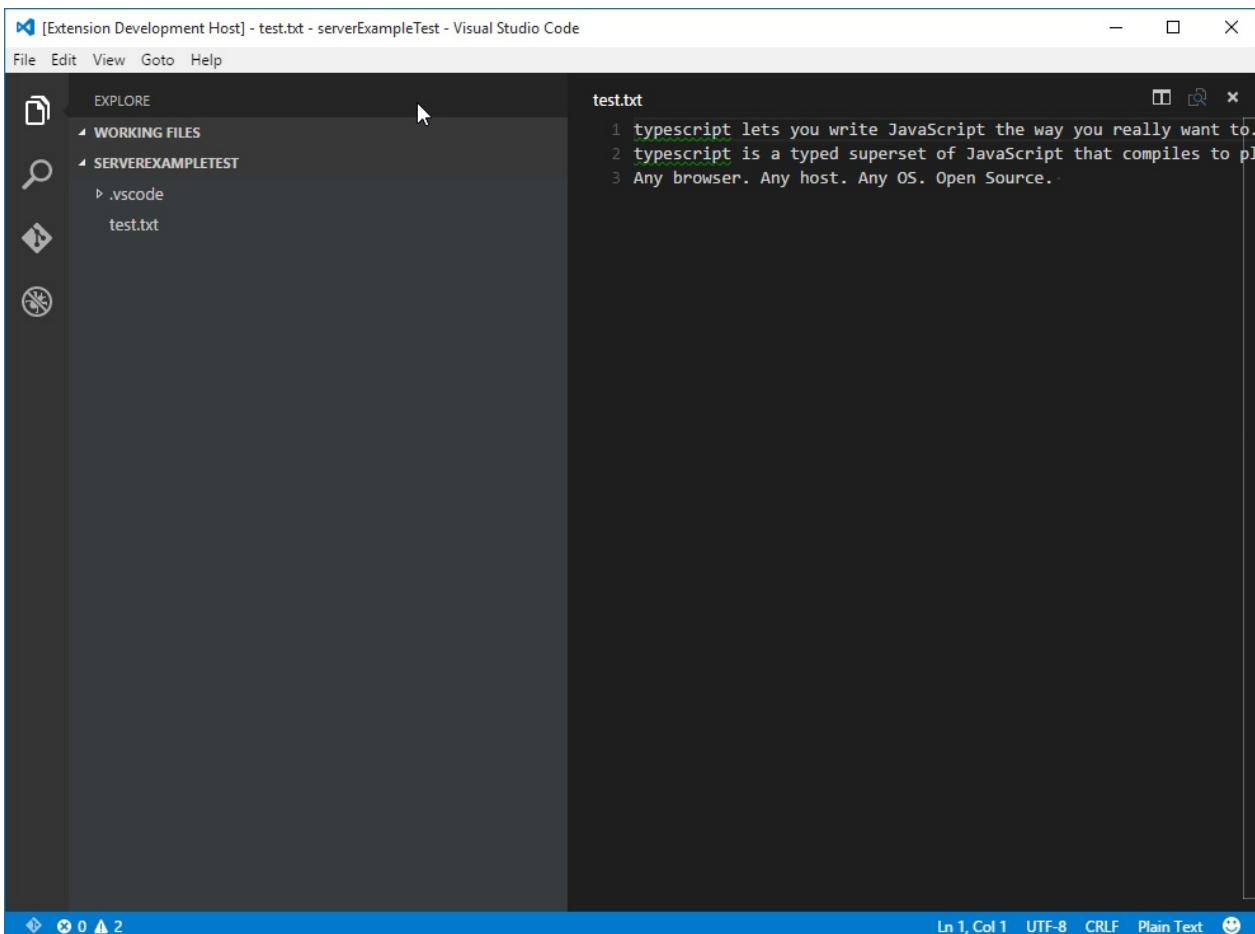
- 如果开始和结束位置相同，VS Code将在这个单词de的位置画曲线
- 如果你想画曲线一直到行尾，设置结束位置的字符为Number.MAX_VALUE。

向下面这样测试语言服务：

- 到包含服务代码（见上文）的VS Code的实例中并且按下 kb(workbench.action.tasks.build) 启动构建任务。任务将编译服务代码并且安装服务到插件目录。
- 现在会带有客户端插件的VS Code实例中并且按下 kb(workbench.action.debug.start) 启动 扩展开发主机 VS Code实例执行插件代码。
- 在根目录创建一个test.txt文件并且粘贴如下内容：

```
typescript lets you write JavaScript the way you really want to.
typescript is a typed superset of JavaScript that compiles to plain JavaScript.
Any browser. Any host. Any OS. Open Source.
```

扩展开发主机 实例将看起来如下图这样



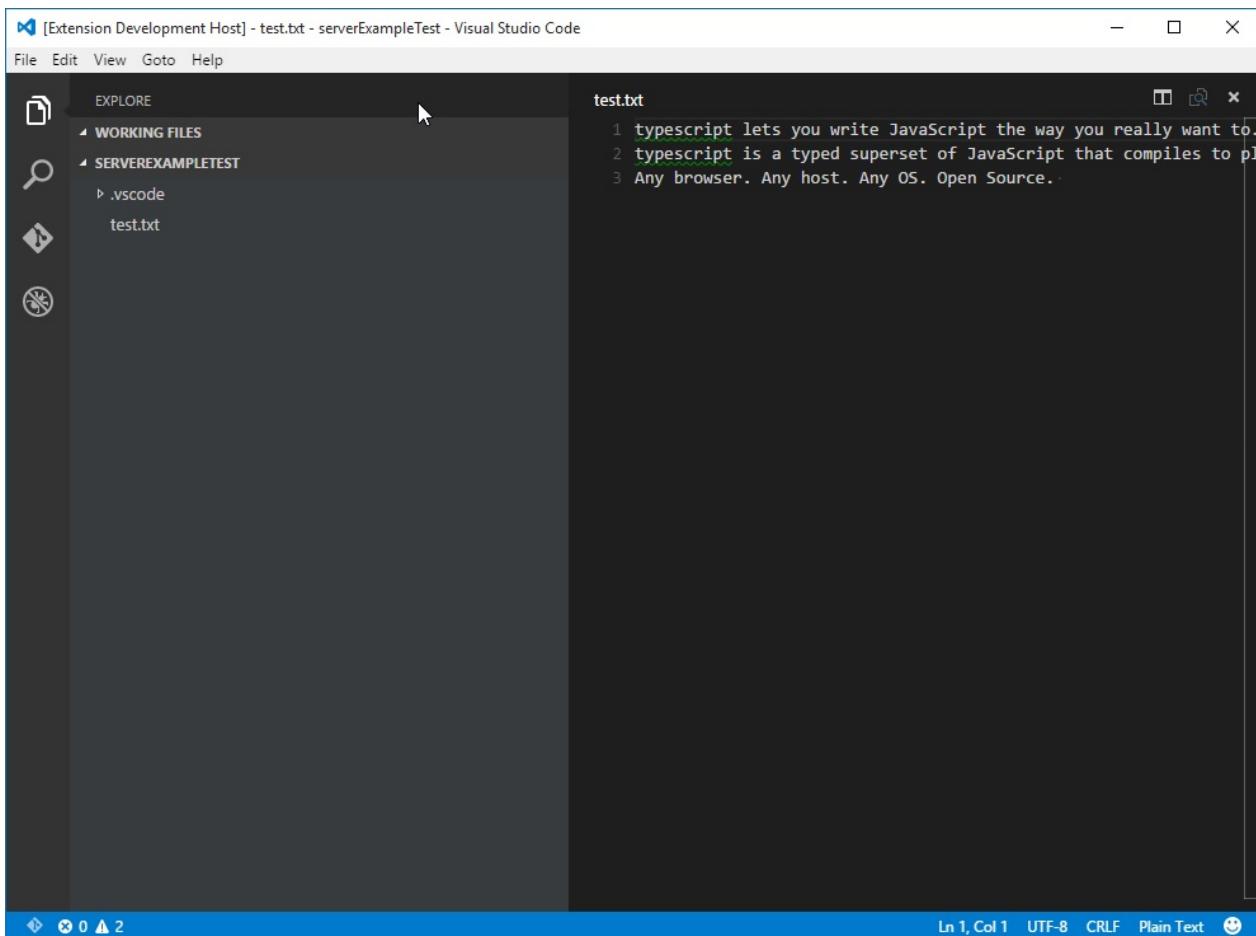
- If the start and end positions are the same, VS Code will squiggle the word at that position.
- If you want to squiggle until the end of the line, then set the character of the end position to Number.MAX_VALUE.

To test the language server do the following:

- Go to the VS Code instance containing the server code (see above) and press `kb(workbench.action.tasks.build)` to start the build task. The task compiles the server code and installs (e.g. copies) it into the extension folder.
- Now go back to the VS Code instance with the extension (client) and press `kb(workbench.action.debug.start)` to launch an additional `Extension Development Host` instance of VS Code that executes the extension code.
- Create a test.txt file in the root folder and paste the following content:

```
typescript lets you write JavaScript the way you really want to.  
typescript is a typed superset of JavaScript that compiles to plain JavaScript.  
Any browser. Any host. Any OS. Open Source.
```

The `Extension Development Host` instance will then look like this:



一起调试客户端和服务 Debugging both Client and Server

调试客户端代码就像调试普通插件一样简单。在包含客户端代码的VS Code实例中简单的设置断点然后按下 `kb(workbench.action.debug.start)` 调试插件。关于启动和调试插件的详细描述请看[运行和调试你的插件](#)。

The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** extension.ts - client - Visual Studio Code
- File Menu:** File Edit View Goto Help
- Toolbar:** DEBUG Launch Extension
- Left Sidebar:**
 - VARIABLES: Local, clientOptions, context, debugOptions, disposable, serverModule, serverOptions
 - WATCH
 - CALL STACK: Paused on breakpoint. Shows a stack trace from activate down to _dirname.defined.Lenter.
 - BREAKPOINTS: All exceptions, Uncaught exceptions, extension.ts
- Editor:** extension.ts src (Line 39, Col 1)
- Debug Console:** 'C:\Program Files (x86)\Visual Studio Code\code.exe' ... Loading development extension at p:/mseng/VSCode/vsco...
- Bottom Status Bar:** In 39, Col 1 UTF-8 CRLF TypeScript

因为服务是被在客户端中运行 `LanguageClient` 启动的。我们需要附加到运行中的服务去调试。要完成这一切，切换到服务代码的VS Code实例中并且按下 `kb(workbench.action.debug.start)`。这将附加一个调试器到服务中。然后使用正常的调试视图调试即可。

```

30 →   capabilities: {
31 →     →   // Tell the client that the server works in FULL text document sync mode
32 →     →   textDocumentSync: documents.syncKind
33 →     → }
34 →   }
35 });
36
37 // The content of a text document has changed. This event is emitted
38 // when the text document first opened or when its content has changed.
39 documents.onDidChangeContent((change) => {
40   let diagnostics: Diagnostic[] = [];
41   let lines = change.document.getText().split(/\r?\n/g);
42   lines.forEach((line, i) => {
43     let index = line.indexOf('typescript');
44     if (index >= 0) {
45       diagnostics.push({
46         severity: DiagnosticSeverity.Warning,
47         range: {
48           start: { line: i, character: index },
49           end: { line: i, character: index + 10 }
50         },
51         message: `${line.substr(index, 10)} should be spelled TypeScript`
52       });
53     }
54   })
55   // Send the computed diagnostics to VSCode.
56   connection.sendDiagnostics({ uri: change.document.uri, diagnostics });
57 });
58
59
60 interface Settings {
61   languageClientExample: ExampleSettings;
62 }
63
64 interface ExampleSettings {
65   maxNumberOfProblems: number;
66 }

```

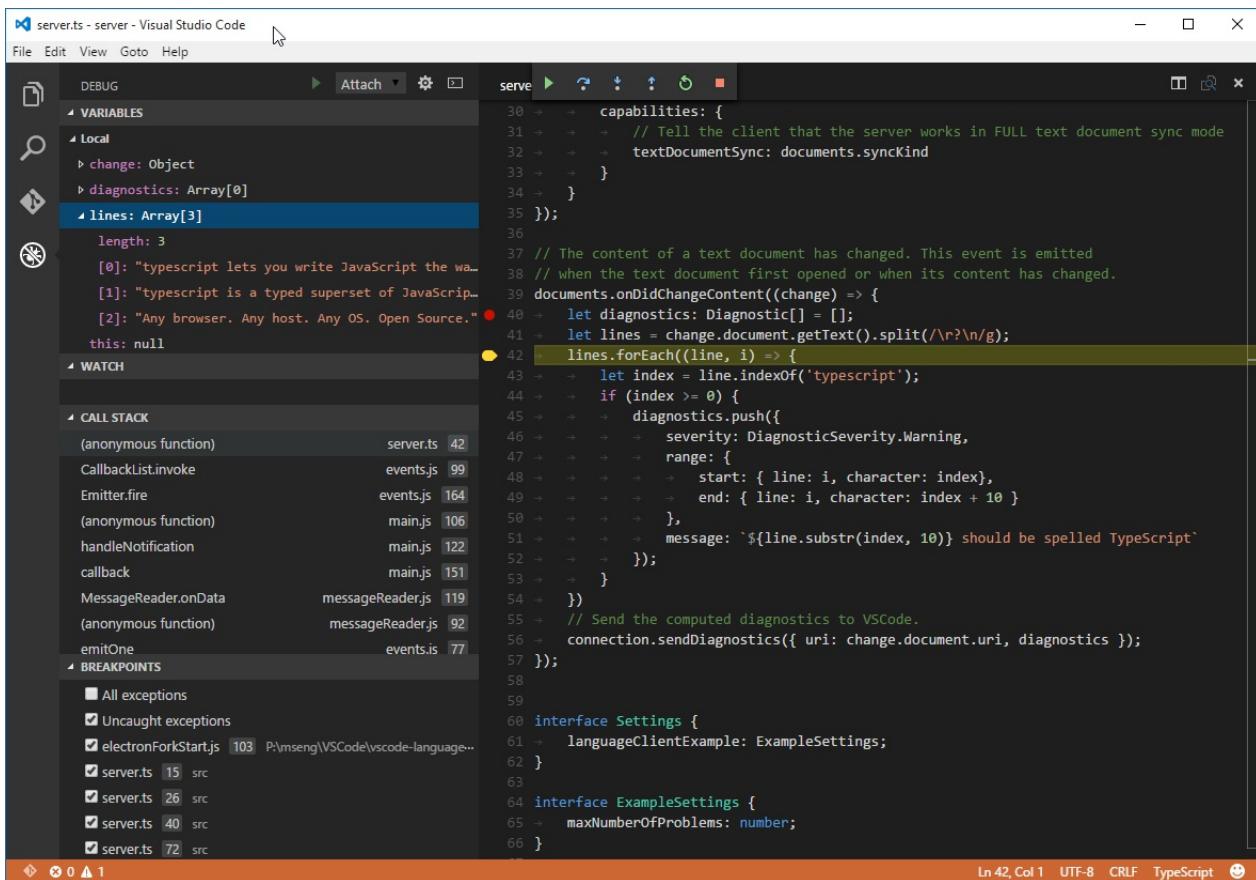
Ln 42, Col 1 UTF-8 CRLF TypeScript

Debugging the client code is as easy as debugging a normal extension. Simply set a breakpoint in the VS Code instance that contains the client code and debug the extension by pressing `kb(workbench.action.debug.start)`. For a detailed description about launching and debugging an extension see [Running and Debugging Your Extension](#).

The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** extension.ts - client - Visual Studio Code
- File Menu:** File Edit View Goto Help
- Toolbar:** DEBUG Launch Extension
- Left Sidebar:**
 - VARIABLES: Local, clientOptions, context, debugOptions, disposable, serverModule, serverOptions
 - WATCH: (empty)
 - CALL STACK: Paused on breakpoint. Shows a stack trace from activate down to _dirname.defined.Lenter.
 - BREAKPOINTS: All exceptions, Uncaught exceptions, extension.ts (checked)
- Code Editor:** extension.ts src (Line 39, Col 1) showing code related to language client creation.
- Debug Console:** Loading development extension at p:/mseng/VSCode/vsco...
- Bottom Status Bar:** In 39, Col 1 UTF-8 CRLF TypeScript

Since the server is started by the `LanguageClient` running in the extension (client), we need to attach a debugger to the running server. To do so, switch to the VS Code instance containing the server code and press `kb(workbench.action.debug.start)`. This will attach the debugger to the server. Use the normal Debug View to interact with the running server.



在服务中使用设置选项 Using Configuration Settings in the Server

当我们编写已经定义了最大问题数设置的插件的客户端部分时，我们可以用同步选项让 `LanguageClient` 对象去同步这些设置到服务端。

```
synchronize: {
  // 将'languageServerExample'选项同步到服务
  configurationSection: 'languageServerExample',
  // 当工作空间中的'.clientrc'文件改变时通知服务
  fileEvents: workspace.createFileSystemWatcher('**/.clientrc')
}
```

现在我们唯一需要做的就是在服务端监听设置改变并且如果有设置改变就对打开的文本文档重新生效。为了重用文档改变事件处理的逻辑，我们提取这部分代码到 `validateTextDocument` 函数中并且添加添加 `maxNumberOfProblems` 变量：

```
function validateTextDocument(textDocument: ITextDocument): void {
    let diagnostics: Diagnostic[] = [];
    let lines = textDocument.getText().split(/\r?\n/g);
    let problems = 0;
    for (var i = 0; i < lines.length && problems < maxNumberOfProblems; i++) {
        let line = lines[i];
        let index = line.indexOf('typescript');
        if (index >= 0) {
            problems++;
            diagnostics.push({
                severity: DiagnosticSeverity.Warning,
                range: {
                    start: { line: i, character: index},
                    end: { line: i, character: index + 10 }
                },
                message: `${line.substr(index, 10)} should be spelled TypeScript`
            });
        }
    }
    // 发送诊断信息到VS Code.
    connection.sendDiagnostics({ uri: textDocument.uri, diagnostics });
}
```

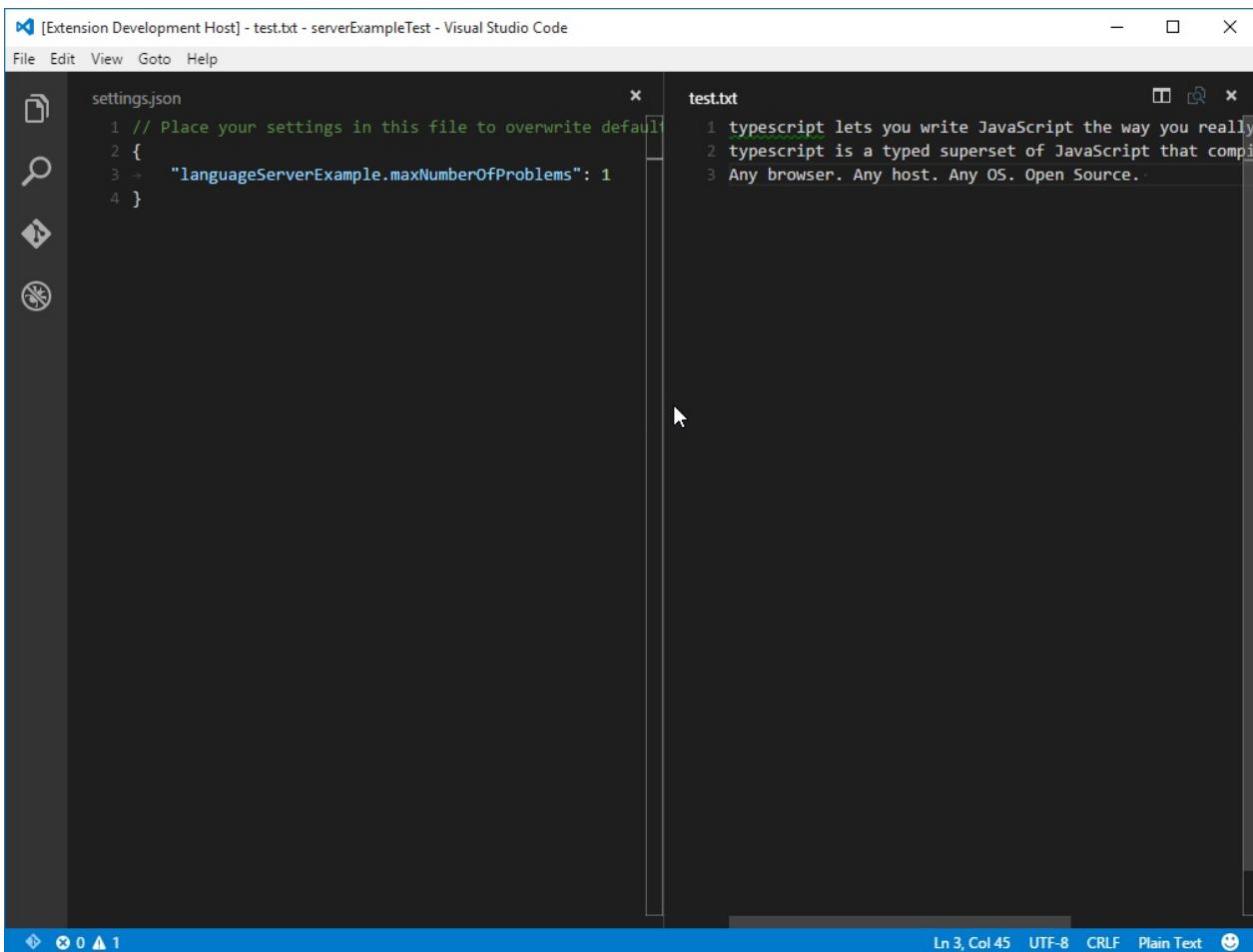
通过向连接添加配置改变通知处理器来处理配置变化的处理。相应的代码如下：

```
// 设置接口描述了服务相关的设置部分
interface Settings {
    languageServerExample: ExampleSettings;
}

// 这些是我们在客户端的package.json文件中定义的设置示例
interface ExampleSettings {
    maxNumberOfProblems: number;
}

// 控制最大问题数设置
let maxNumberOfProblems: number;
// 设置改变，也发送服务器激活
connection.onDidChangeConfiguration((change) => {
    let settings = <Settings>change.settings;
    maxNumberOfProblems = settings.languageServerExample.maxNumberOfProblems || 100;
    // 验证所有打开的文档
    documents.all().forEach(validateTextDocument);
});
```

修改最大报告问题数为1个并再次启动客户端的结果：



When writing the client part of the extension we already defined a setting to control the maximum numbers of problems reported. We also instructed the `LanguageClient` to sync these settings to the server using the synchronization configuration

```
synchronize: {
    // Synchronize the setting section 'languageClientExample' to the server
    configurationSection: 'languageServerExample',
    // Notify the server about file changes to '.clientrc' files contain in the workspace
    fileEvents: workspace.createFileSystemWatcher('**/.clientrc')
}
```

The only thing we need to do now is to listen to configuration changes on the server side and if a settings changes revalidate the open text documents. To be able to reuse the validate logic of the document change event handling we extract the code into a `validateTextDocument` function and modify the code to honor a `maxNumberOfProblems` variable:

```

function validateTextDocument(textDocument: ITextDocument): void {
    let diagnostics: Diagnostic[] = [];
    let lines = textDocument.getText().split(/\r?\n/g);
    let problems = 0;
    for (var i = 0; i < lines.length && problems < maxNumberOfProblems; i++) {
        let line = lines[i];
        let index = line.indexOf('typescript');
        if (index >= 0) {
            problems++;
            diagnostics.push({
                severity: DiagnosticSeverity.Warning,
                range: {
                    start: { line: i, character: index},
                    end: { line: i, character: index + 10 }
                },
                message: `${line.substr(index, 10)} should be spelled TypeScript`
            });
        }
    }
    // Send the computed diagnostics to VS Code.
    connection.sendDiagnostics({ uri: textDocument.uri, diagnostics });
}

```

The handling of the configuration change is done by adding a notification handler for configuration changes to the connection. The corresponding code looks like this:

```

// The settings interface describe the server relevant settings part
interface Settings {
    languageServerExample: ExampleSettings;
}

// These are the example settings we defined in the client's package.json
// file
interface ExampleSettings {
    maxNumberOfProblems: number;
}

// hold the maxNumberOfProblems setting
let maxNumberOfProblems: number;
// The settings have changed. Is send on server activation
// as well.
connection.onDidChangeConfiguration((change) => {
    let settings = <Settings>change.settings;
    maxNumberOfProblems = settings.languageServerExample.maxNumberOfProblems || 100;
    // Revalidate any open text documents
    documents.all().forEach(validateTextDocument);
});

```

Starting the client again and changing the setting to maximum report 1 problem results in the following validation:

The screenshot shows a Visual Studio Code interface with two tabs open. The left tab is 'settings.json' containing the following JSON code:

```

1 // Place your settings in this file to overwrite default
2 {
3   "languageServerExample.maxNumberOfProblems": 1
4 }

```

The right tab is 'test.txt' containing the following text:

```

1typescript lets you write JavaScript the way you really
2typescript is a typed superset of JavaScript that compi
3Any browser. Any host. Any OS. Open Source.

```

The status bar at the bottom of the screen displays 'Ln 3, Col 45'.

添加额外的语言特性 Adding additional Language Features

第一个有趣的特性是，语言服务通常实现的功能都是校验文档。甚至感觉语法检查器就是语言服务，因为在VS Code中语法检查器通常都是作为语言服务来实现的(参考 [eslint](#) 和 [jshint](#) 例子)。但是实际上有更多的语言服务，提供代码自动完成，查找所有引用或者跳转到定义的功能。下面的代码给服务添加了代码自动完成。它简单的提名两个单词'TypeScript'和'JavaScript'。

```
// 这个处理器提供完成项的初始化列表。
connection.onCompletion((textDocumentPosition: TextDocumentIdentifier): CompletionItem[] => {
    // 传递的参数中包含了在文本文档中请求代码自动完成的位置。
    // 这个例子中忽略了这些信息，总是提供相同的完成选项。
    return [
        {
            label: 'TypeScript',
            kind: CompletionItemKind.Text,
            data: 1
        },
        {
            label: 'JavaScript',
            kind: CompletionItemKind.Text,
            data: 2
        }
    ]
});

// 这个处理器解析了在自动完成列表里选择的选项的额外信息。
connection.onCompletionResolve((item: CompletionItem): CompletionItem => {
    if (item.data === 1) {
        item.detail = 'TypeScript details',
        item.documentation = 'TypeScript documentation'
    } else if (item.data === 2) {
        item.detail = 'JavaScript details',
        item.documentation = 'JavaScript documentation'
    }
    return item;
});
```

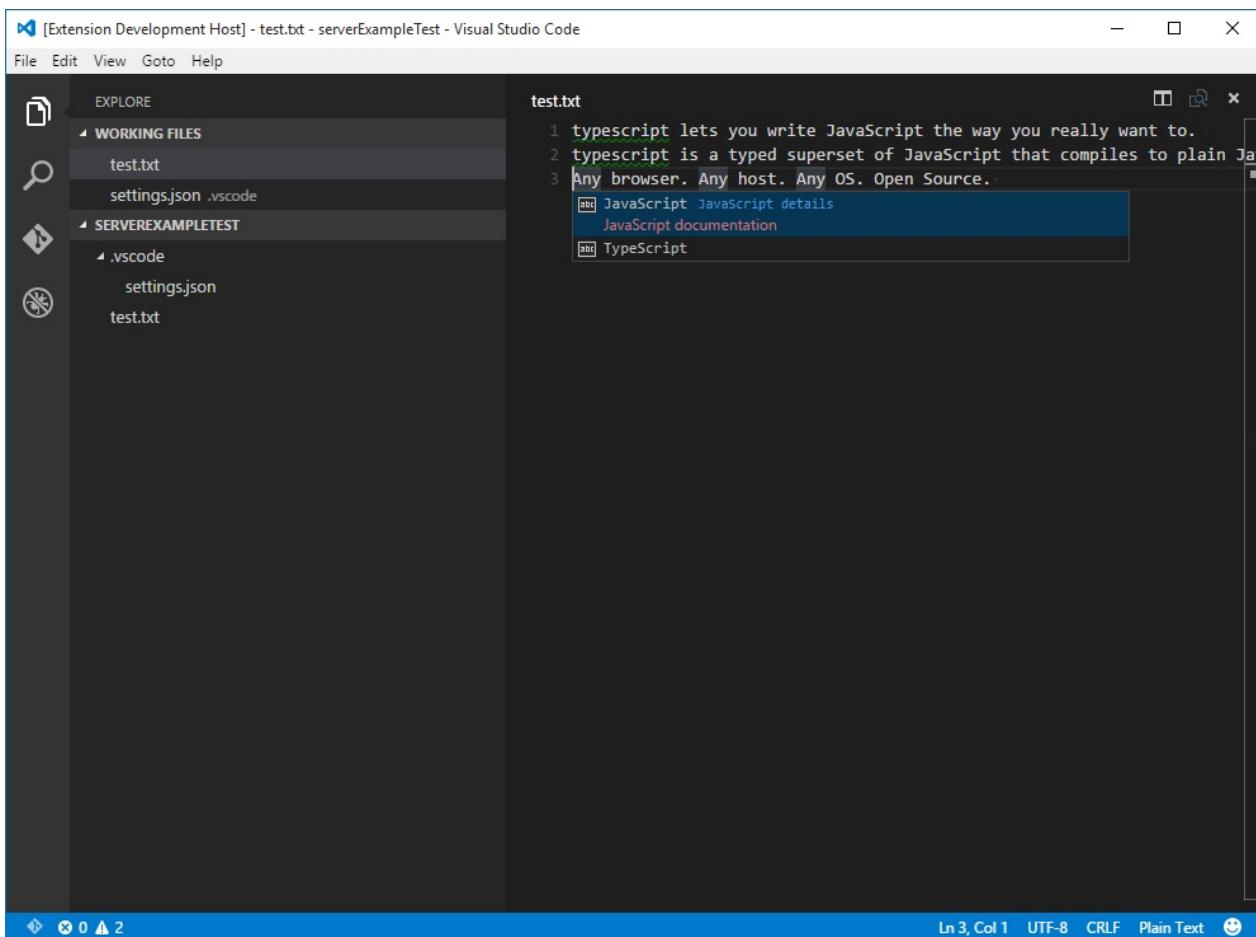
在解析处理器中 `data` 成员通常用来作为自动完成选项的唯一标识符。这个数据属性通过协议来传输。因为底层的消息传输协议是基于数据成员的**JSON**，所以应该保证数据可以使用**JSON**序列化和反序列化。

剩下的就是告诉**VS Code**服务支持代码自动完成。要完成这个功能，需要在初始化处理器中声明相应的能力：

```

connection.onInitialize((params): InitializeResult => {
    ...
    return {
        capabilities: {
            ...
            // 告诉客户端服务支持代码自动完成
            completionProvider: {
                resolveProvider: true
            }
        }
    }
});
```

下面的屏幕截图显示纯文本文件中运行的代码完成：



The first interesting feature a language server usually implements is validation of documents. In that sense even a linter counts as a language server and in VS Code linters are usually implemented as language servers (see [eslint](#) and [jshint](#) for examples). But there is more to language servers. They can provide code complete, find all references or goto definition. The example code below adds code completion to the server. It simply proposes the two words 'TypeScript' and 'JavaScript'.

```
// This handler provides the initial list of the completion items.
connection.onCompletion((textDocumentPosition: TextDocumentIdentifier): CompletionItem[] => {
    // The pass parameter contains the position of the text document in
    // which code complete got requested. For the example we ignore this
    // info and always provide the same completion items.
    return [
        {
            label: 'TypeScript',
            kind: CompletionItemKind.Text,
            data: 1
        },
        {
            label: 'JavaScript',
            kind: CompletionItemKind.Text,
            data: 2
        }
    ]
});

// This handler resolve additional information for the item selected in
// the completion list.
connection.onCompletionResolve((item: CompletionItem): CompletionItem => {
    if (item.data === 1) {
        item.detail = 'TypeScript details',
        item.documentation = 'TypeScript documentation'
    } else if (item.data === 2) {
        item.detail = 'JavaScript details',
        item.documentation = 'JavaScript documentation'
    }
    return item;
});
```

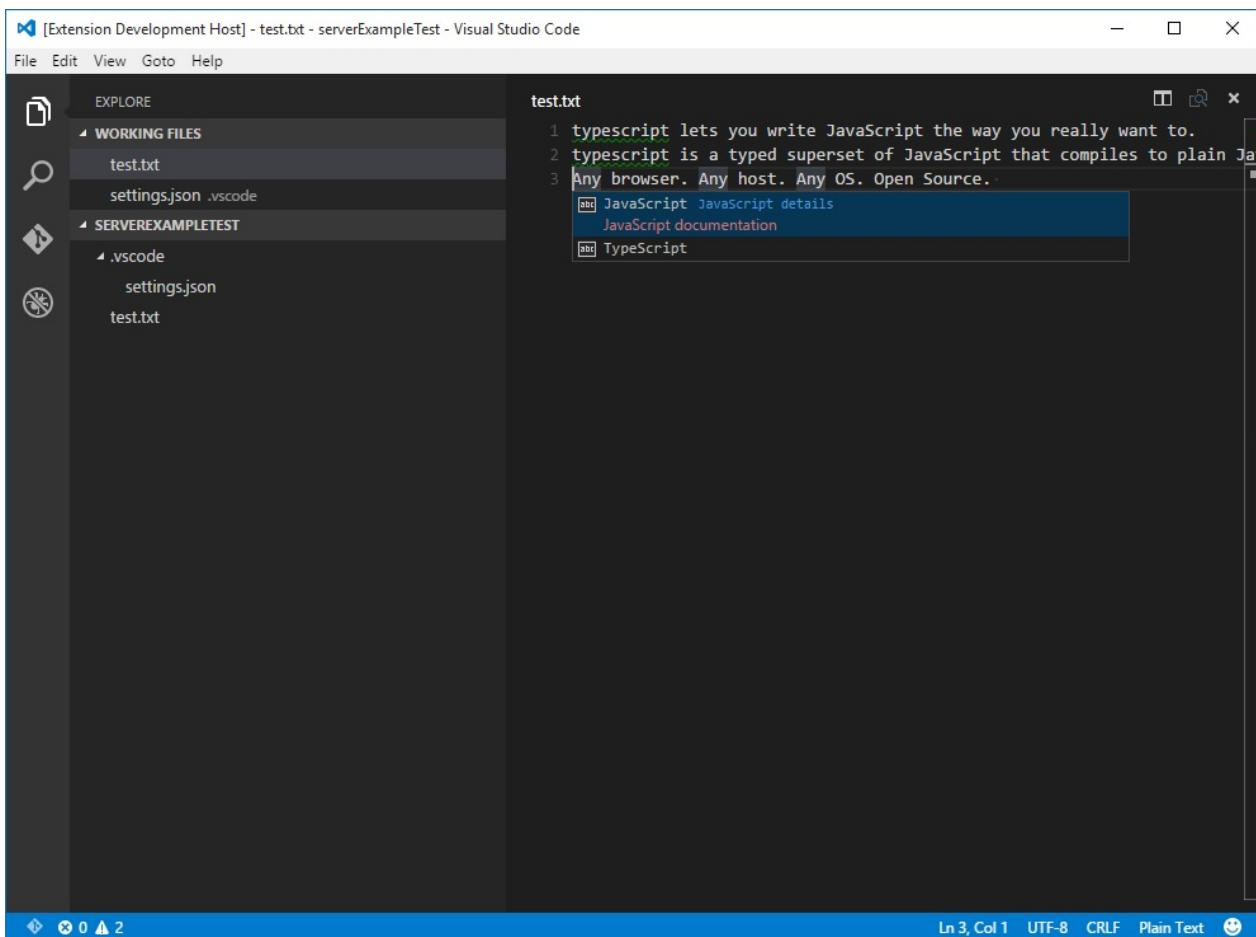
The `data` fields is used to uniquely identify a completion item in the resolve handler. The `data` property is transparent for the protocol. Since the underlying message passing protocol is JSON based the `data` field should only hold data that is serializable to and from JSON.

All that is missing is to tell VS Code that the server support code completion requests. To do so, flag the corresponding capability in the intialize handler:

```

connection.onInitialize((params): InitializeResult => {
    ...
    return {
        capabilities: {
            ...
            // Tell the client that the server support code complete
            completionProvider: {
                resolveProvider: true
            }
        }
    }
});
```

The screen shot below shows the completed code running on a plain text file:



Additional Language Server features

下面的来的语言特性是语言服务现在支持的除代码完成以外的特性：

- **Document Highlights:** 高亮显示所有在文本文档中相同的符号。
- **Hover:** 提供在文本文档中选择的符号的悬停信息。
- **Signature Help:** 提供在文本文档中选择的符号的签名信息。

- *Goto Definition*: 提供跳转到文本文档中选择的符号的定义处支持。
- *Find References*: 提供查找文本文档中选择的符号在所有项目中的引用。
- *List Document Symbols*: 列出文本文档中定义的所有符号。
- *List Workspace Symbols*: 列出所有项目中的符号。
- *Code Actions*: 对给出的文本文档和范围执行命令。
- *CodeLens*: 对给出的文本文档计算代码元信息统计。
- *Document Formatting*: 包括格式化整个文档、文档范围、和类型格式化。
- *Rename*: 在项目内重命名符号。

The following language features are currently support in a language server besides code complete:

- *Document Highlights*: highlights all 'equal' symbols in a text document.
- *Hover*: provides hover information for a symbol selected in a text document.
- *Signature Help*: provides signature help for a symbol selected in a text document.
- *Goto Definition*: provides goto definition support for a symbol selected in a text document.
- *Find References*: finds all project-wide references for a symbol selected in a text document.
- *List Document Symbols*: lists all symbols defined in a text document.
- *List Workspace Symbols*: lists all project-wide symbols.
- *Code Actions*: compute commands for a given text document and range.
- *CodeLens*: compute CodeLens statistics for a given text document.
- *Document Formatting*: this includes formatting of whole documents, document ranges and formatting on type.
- *Rename*: project-wide rename of a symbol.

增量同步文本文档 Incremental Text Document Synchronization

示例中通过 `vscode-languageserver` 模块提供的简单的文本文档管理器在VS Code和语言服务间同步文档。有两个缺点：

- 因为整个文本文档内容被重复发送给服务造成大量数据传输。
- 如果使用了语言库，很多库支持增量文档更新以避免非必须的解析和虚拟语法树创建。

因为协议也支持增量文档同步。使用增量文档同步需要安装三个通知处理器：

- *onDidOpenTextDocument*: 当文档在VS Code中被打开时调用。
- *onDidChangeTextDocument*: 当文档内容在VS Code中被修改时调用。
- *onDidCloseTextDocument*: 当文档在VS Code中被关闭时调用。

下面的代码片段将说明怎样在连接上安装这些通知处理器并且在初始化里返回正确的功能声明：

```

connection.onInitialize((params): InitializeResult => {
    ...
    return {
        capabilities: {
            // 启用增量文档同步
            textDocumentSync: TextDocumentSyncKind.Incremental,
            ...
        }
    }
});

connection.onDidOpenTextDocument((params) => {
    // 一个文档在VS Code中被打开。
    // params.uri 唯一标识一个文档。对于存储在磁盘上的文档这是文件URI。
    // params.text 文档的全部初始化内容。
});

connection.onDidChangeTextDocument((params) => {
    // 文档内容在VS Code中被修改。
    // params.uri 唯一标识一个文档。
    // params.contentChanges 描述文档的内容变化。
});

connection.onDidCloseTextDocument((params) => {
    // 一个文档在VS Code中被关闭。
    // params.uri 唯一标识一个文档。
});

```

The example uses the simple text document manager provided by the `vscode-languageserver` module to synchronize documents between VS Code and the language server. This has two drawbacks:

- lots of data transfer since the whole content of a text document is sent to the server repeatedly.
- if an existing language library is used, such libraries usually support incremental document updates to avoid unnecessary parsing and abstract syntax tree creation.

The protocol therefore supports incremental document synchronization as well. To make use of incremental document synchronization a server needs to install three notification handlers:

- `onDidOpenTextDocument`: is called when a text document got opened in VS Code.
- `onDidChangeTextDocument`: is called when the content of a text document changes in VS Code.
- `onDidCloseTextDocument`: is called when a text document got closed in VS Code.

Below a code snippet that illustrates how to hook these notification handlers on a connection and how to return the right capability on initialize:

```
connection.onInitialize((params): InitializeResult => {
    ...
    return {
        capabilities: [
            // Enable incremental document sync
            TextDocumentSyncKind.Incremental,
            ...
        ]
    }
});

connection.onDidOpenTextDocument((params) => {
    // A text document got opened in VS Code.
    // params.uri uniquely identifies the document. For documents store on disk this is a file URI.
    // params.text the initial full content of the document.
});

connection.onDidChangeTextDocument((params) => {
    // The content of a text document did change in VS Code.
    // params.uri uniquely identifies the document.
    // params.contentChanges describe the content changes to the document.
});

connection.onDidCloseTextDocument((params) => {
    // A text document got closed in VS Code.
    // params.uri uniquely identifies the document.
});
```

下一步 Next Steps

向学习更多的VS Code插件模型，试试阅读这些文档：

- [vscode API Reference](#) - 学习VS Code语言服务和语言的深度整合。

To learn more about VS Code extensibility model, try these topic:

- [vscode API Reference](#) - Learn about deep language integration with VS Code language services.

常见问题 Common Questions

问：当我尝试去附加到服务时，我遇到"**cannot connect to runtime process (timeout after 5000ms)**"？

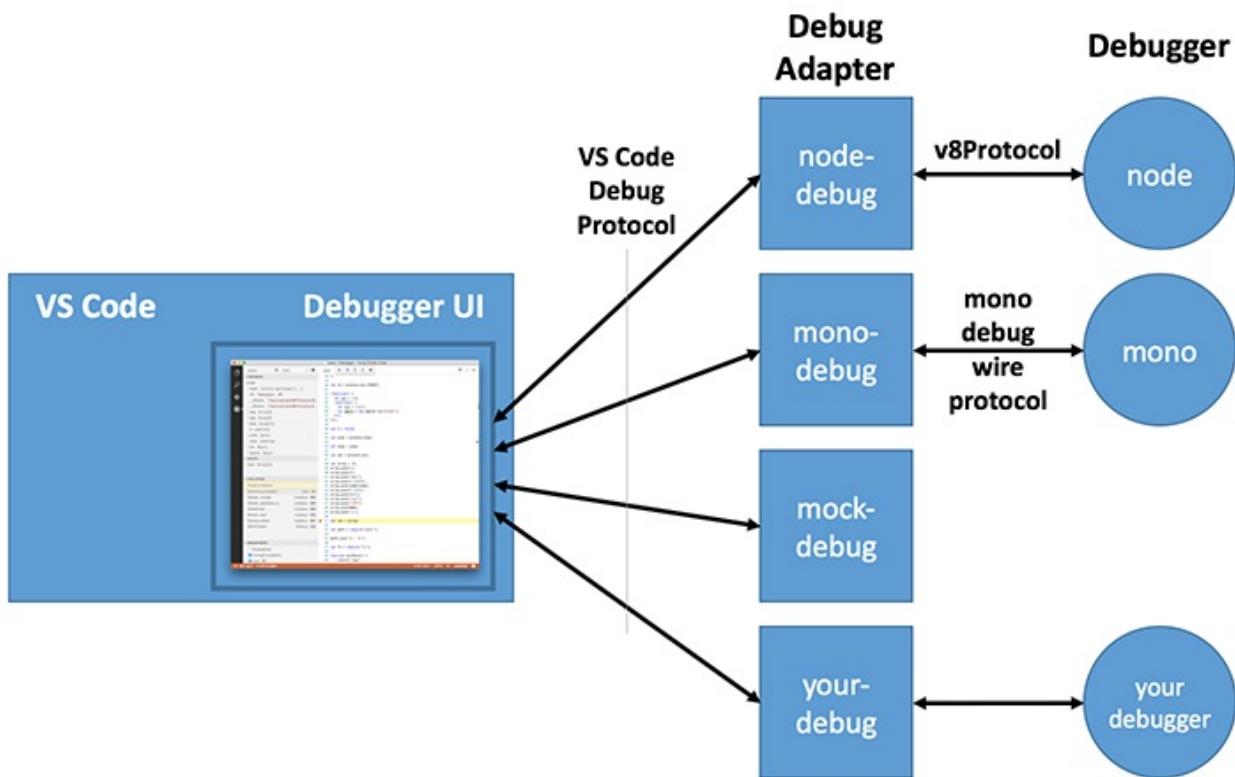
答：如果你的服务没有正在运行那么当你试图去附加调试器时就会看到超时错误。客户端启动夫妇，所以为了拥有一个已经运行的服务，确保你已经启动客户端。

Q: When I try to attach to the server, I get "cannot connect to runtime process (timeout after 5000ms)"?

A: You will see this timeout error if the server isn't running when you try to attach the debugger. The client starts the language server so make sure you have started the client in order to have a running server.

调试器示例

因为VS Code只是实现了一个通用的(未知语言)调试器界面，所以他不能和真正的调试器直接通信，而是通过一个所谓的调试适配器使用一个抽象的写入协议来通信。我们称这个协议为*VS Code Debug Protocol* (简写为CDP)。一个调试适配器是一个单独的可执行程序，它和真正的调试器通信并且将抽象的CDP装换为针对调试器的具体的调试协议。



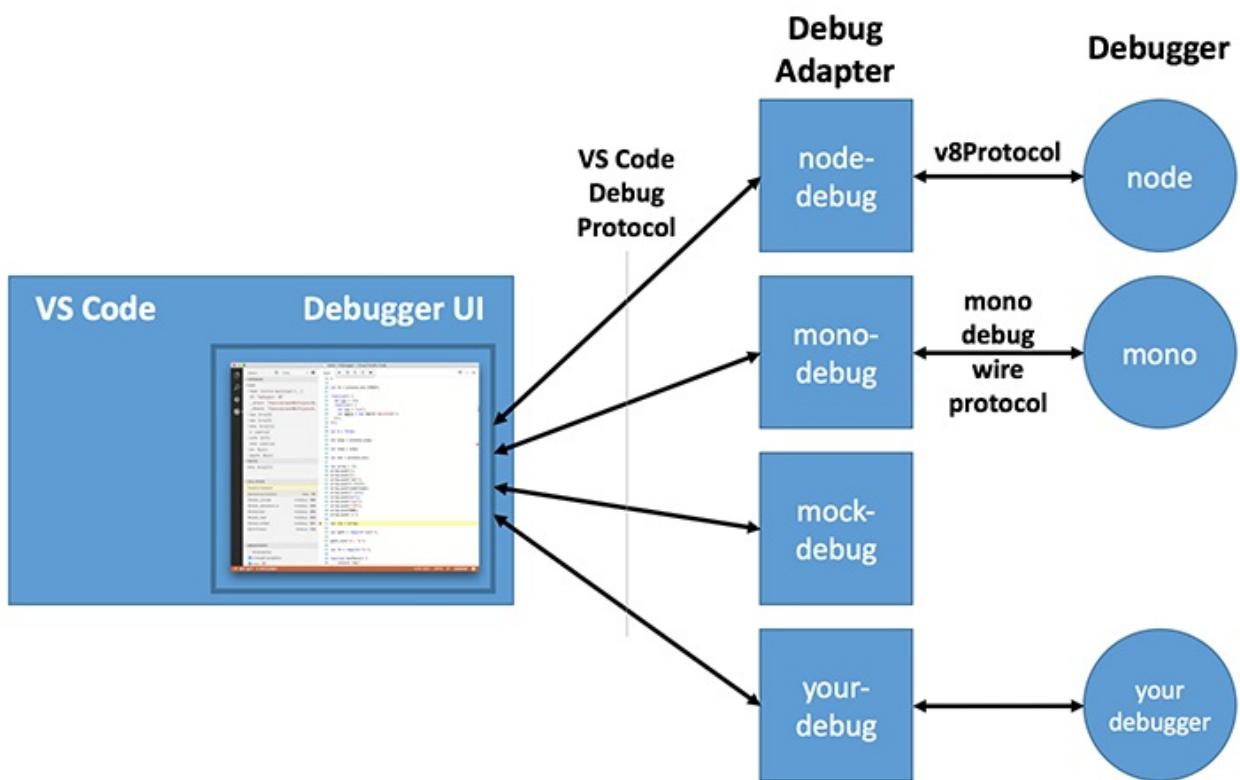
为了避免本地防火墙的问题，VS Code通过标准输入/输出来替代其他复杂的方式和适配器通信。

每个调试适配器定义了一个从VS Code启动配置中引用的调试类型。当调试会话启动后，VS Code查找一个基于调试类型的调试适配器然后在单独的进程中启动可执行文件。当调试会话结束后，调试适配器停止运行。

调试适配器是VS Code的扩展性架构的一部分：他们被作为插件发布。之所以将调试适配器和其他插件分开是因为调试适配器的代码不运行在插件实例上，而是一个单独的程序。有两个原因：第一，使得可以使用对于调试器或者运行时来说最适合的语言来实现适配器。第二，如果有需要，一个单独的程序可以更容易地运行在高级权限下。

VS Code只内嵌了Node.js的调试器插件更多的调试器插件可以从[这里找到VS Code 市场](#)或者你可以自己创建一个调试器插件。

这篇文档将向你展示如何创建一个新的调试器插件。



为了避免本地防火墙问题，VS Code通过 `stdin/stdout` 来和网络适配器对话，而不是使用一个更加复杂的对话机制。

每个调试适配器声明了一个会被 VS Code 启动配置里面读取的调试类型。当一段调试开始时，VS Code 寻找基于调试类型的调试适配器，然后启动一个可独立线程运行的可执行文件。当一段调试结束时，适配器停止工作。

调试适配器是VS Code可扩展架构的一部分：他们贡献于插件。和其他插件不同的是调试适配器代码不是运行在插件主机上，而是一个独立程序上。有两方面原因：首先，可以以一种，对于给定调试器或者运行时最合适语言，来实现适配器；第二，如果需要的话，一个独立程序可以在一个高级模式下更容易运行。

VS Code为 Node.js实现了一个调试插件。更多调试器插件可见[VS Code 商店](#)，你也可以自己创造一个调试插件.

这篇文档将会告诉你如何创建一个新的调试插件。

安装一个调试器插件示例 **Installing a Sample Debug Extension**

因为对于入门教程来说，从零开始创建一个调试适配器有些太复杂了。我们将从一个我们为了教学而创建的'starter kit'简单的调试适配器开始。它名叫'mock-debug'因为它并没有和真正的调试器通讯，而是虚拟了一个。所以mock-debug模拟了一个调试适配器并且支持单步，继续，断点，异常和变量查看但是它并没有连接到真正的调试器。

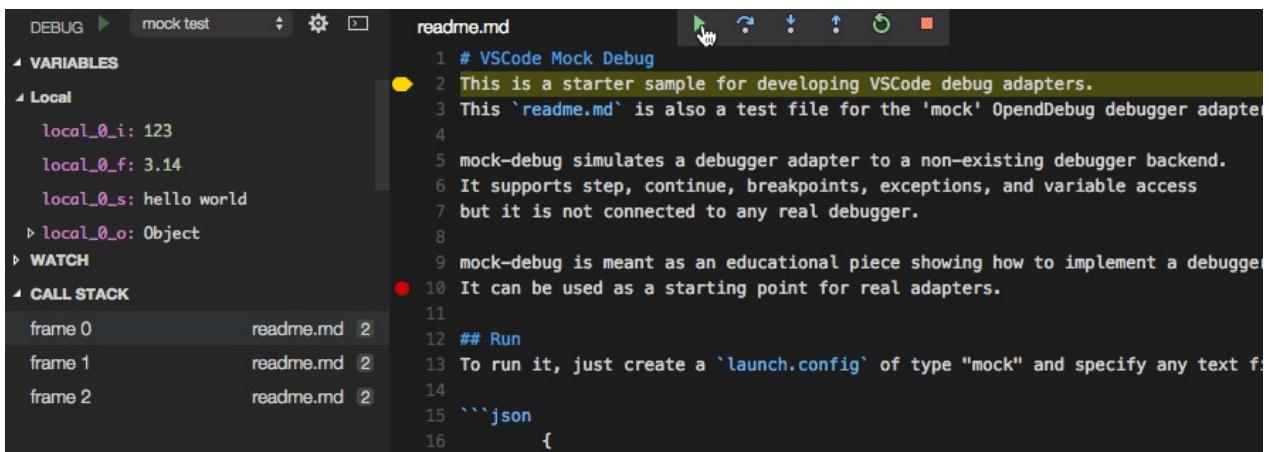
在探索mock-debug的开发环境设置之前，让我们先从VS Code市场上安装一个生成前版本，做如下操作：

- 使用命令面板 扩展：安装扩展 找到并安装mock-debug插件，
- 重新启动VS Code。

尝试如下步骤：

- 在VS Code中，创建一个测试项目和一个新的程序文件 `readme.md` 然后随意的输入几行文本。
- 切换到调试试图并且按下齿轮图标。
- VS Code将让你选择调试环境(选择"Mock Debugger")然后会创建一个默认的启动配置。

如果你开始了启动配置，你可以单步调试目标文件，设置断点，运行进异常(如果单词 `exception` 出现在某一行里)。



mock-debug开发环境建立

现在让我们获取mock-debug的源码并开始在 VS Code上开发吧：

```
git clone https://github.com/Microsoft/vscode-mock-debug.git
cd vscode-mock-debug
npm install
```

在VS Code中打开 `vscode-mock-debug` 项目目录。

包中有什么？

- mock-debug实现在 `src/mockDebug.ts`。这里你可以找到各种CDP请求的请求处理。
- `package.json`，mock-debug插件的清单文件：
 - 列举mock-debug插件的配置项
 - `compile` 和 `watch` 用来转换TypeScript源码到 `out` 目录并且监控随后的代码修改。
 - 两个依赖项 `vscode-debugprotocol` 和 `vscode-debugadapter` 是基于node的调试适配器

的简单开发环境NPM模块。

现在通过选择 `Launch Extension` 配置并且敲击 F5 来构建并启动调试适配器。最开始将会完全的将TypeScript源码转换到 `out` 目录。在完全构建之后，'观察任务'将被启动并增量转换接下来的改动。

在构建完成之后，一个在调试模式运行mock-debug插件的新的VS Code窗口将显示出来。现在你可以打开你的测试项目中的 `readme.md` 文件并'调试'它。

就像像其他插件一样运行来启动调试插件相对于调试来说工作的很好。问题是调试适配器运行在插件实例外的一个单独的进程中。解决方案是以服务模式运行调试适配器：

- 运行 `mock-debug server` 启动配置来以服务模式启动调试适配器(它监听4711端口)
- 在 `mockDebug.ts` 文件里的 `launchRequest(...)` 方法的开始设置断点
- 在额外的VS Code窗口打开测试项目的 `readme.md`
- 在项目里添加顶级 `debugServer` 属性，如同下面代码：

```
{
  "version": "0.2.0",
  "debugServer": 4711,
  "configurations": [
    {
      "name": "mock test",
      "request": "launch",
      "type": "mock",
      "program": "${workspaceRoot}/readme.md",
      "stopOnEntry": true
    }
}
```

- 如果你现在启动调试配置，VS Code将不会在单独的进程启动调试适配器而是连接本地端口4711。
- 此时你应该已经出发了 `launchRequest` 中的断点。

通过上面的步骤，你可以容易的修改，编译，调试mock debug并且将它改造成你想要的调试适配器。

实现VS Code调试协议

一个调试适配器必须实现VS Code调试协议。你可以[在这里](#)找到它的详细信息。

剖析调试适配器的package.json

让我们进一步看一下VS Code插件的调试适配器属性。和每一个VS Code插件一样，调试适配器插件也拥有 `package.json` 文件来声明一些基本的属性如插件的名字，发布者，和版本。使用类别 字段去让插件更容易在VS Code插件市场被找到。

```
{  
    "name": "mock-debug",  
    "version": "0.10.18",  
    "publisher": "vscode",  
    "description": "Starter extension for developing debug adapters for VS Code.",  
    "engines": { "vscode": "0.10.x" },  
    "categories": ["Debuggers"],  
  
    "contributes": {  
        "debuggers": [{  
            "type": "mock",  
            "label": "Mock Debugger",  
  
            "enableBreakpointsFor": { "languageIds": ["markdown"] },  
  
            "program": "./out/mockDebug.js",  
            "runtime": "node",  
  
            "configurationAttributes": {  
                "launch": {  
                    "required": ["program"],  
                    "properties": {  
                        "program": {  
                            "type": "string",  
                            "description": "Workspace relative path to a text file.",  
                            "default": "${workspaceRoot}/readme.md"  
                        },  
                        "stopOnEntry": {  
                            "type": "boolean",  
                            "description": "Automatically stop after launch.",  
                            "default": true  
                        }  
                    }  
                }  
            }  
        }  
    },  
  
    "initialConfigurations": [  
        {  
            "name": "Mock-Debug",  
            "type": "mock",  
            "request": "launch",  
            "program": "readme.md",  
            "stopOnEntry": true  
        }  
    ]  
}
```

更值得关注的是**contributes**下的**debuggers**节。

调试的**type**下介绍了一个调试适配器。用户可以在启动配置中引用这个类型。可选属性**label**可以在UI界面里给调试类型里显示一个更好看的名字。

在**enableBreakpointsFor**属性里你可以列出可以设置断点的语言文件类型。

因为调试适配器是一个单独的程序，**program**属性指定程序的路径。为了让插件自己包含程序，必须把程序放进插件的目录。按照惯例我们应该将程序放在名为 `out` 或者 `bin` 的目录咯，但是你也可以使用其他名字。

因为VS Code运行在不同的平台下，我们不得不确保调试适配器程序能够很好的支持不同的平台。基于这一点有以下的选项：

1. 如果程序实现是平台无关的，例如程序运行在可以支持全平台的运行时上，你可以通过**runtime**属性指定有效的运行时。目前，VS Code支持'node'和'mono'运行时。我们的mock-debug就使用了这个特性。
2. 如果调试适配器需要在不同的平台上有不同的可执行程序，**program**属性可以被用来指定平台，像这样：

```
"debuggers": [{"  
    "type": "gdb",  
    "windows": {  
        "program": "./bin/gdbDebug.exe",  
    },  
    "osx": {  
        "program": "./bin/gdbDebug.sh",  
    },  
    "linux": {  
        "program": "./bin/gdbDebug.sh",  
    }  
}]
```

1. 而这两种方法的组合也是可能的。接下来的例子就是需要运行在OS X和Linux运行时上的mono程序的mono-debug适配器：

```
"debuggers": [{"  
    "type": "mono",  
    "program": "./bin/monoDebug.exe",  
    "osx": {  
        "runtime": "mono"  
    },  
    "linux": {  
        "runtime": "mono"  
    }  
}]
```

configurationAttributes属性指定了调试器在'launch.json'文件中建议和有效的值。

initialConfigurations在VS Code生成'launch.json'的时候被使用。这个默认启动配置将覆盖适配器启动配置。

发布你的调试适配器 **Publishing your Debug Adapter**

当你完成了你的调试适配器时，你可以将它发布到市场上：

- 更新 `package.json` 中的属性来体现你的调试适配器的名字和用途。
- 如这篇文章中描述的方法上传到市场[共享插件](#)。

常见问题 **Common Questions**

无

扩展（以下简称：插件）运行和调试

你可以使用VS Code来开发一个插件，而且VS Code也提供了几个简化开发的工具：

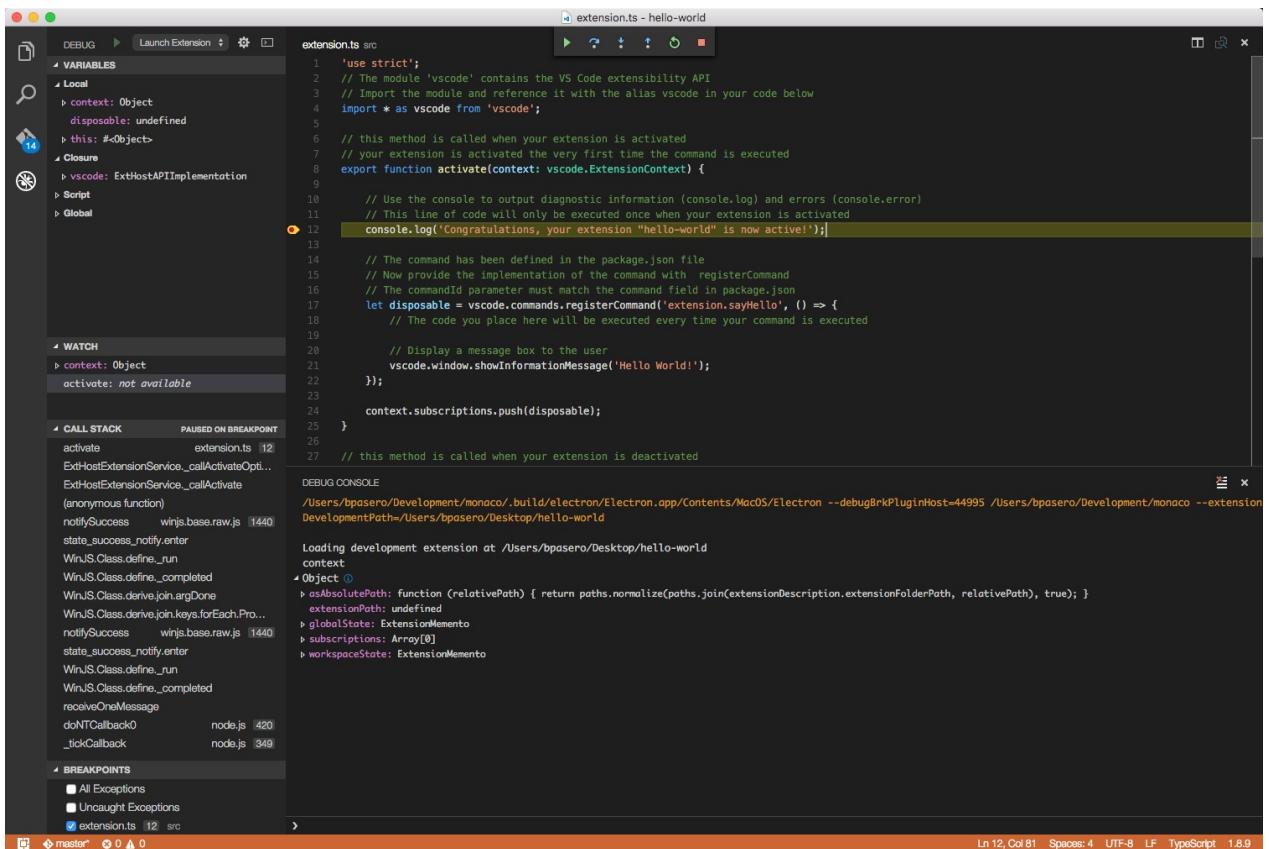
- Yeoman 生成器 to 给插件提供脚手架
 - 智能化，还有针对插件 API 的代码导航
 - 编译 TypeScript（当用 TypeScript 实现的时候）
 - 插件运行和调试
 - 插件发布

生成一个插件

我们建议你通过基本文件脚手架化来开始。你可以使用 `yo code Yeoman` 生成器来做到，而且我们在[Yo Code 文档](#)中描述了细节。生成器会保证一切就绪，开发体验良好。

插件运行和调试

你可以按下 F5，在调试器下轻易地运行。这会打开一个加载好插件的 VS Code 窗口。插件输出在调试控制台。支持在调试视图或者调试控制台设置断点，单步调试，查看变量。



我们来看看幕后将要发生什么。如果你用 TypeScript 来写插件那么你的代码必须先编译成 JavaScript。

编译 TypeScript

生成好的插件内的TypeScript 编译像下面这样设置好

- `tsconfig.json` 声明了 TypeScript 编译器的编译选项。更多请看[TypeScript wiki](#) 或者在我们的[TypeScript章节](#)。
- 合适版本的TypeScript编译器包含在 `node_modules` 文件夹内。
- `typings/vscode-typings.d.ts`：告诉TypeScript 编译器要包含 `vscode` API 声明。
- API 声明 包含在 `node_modules/vscode` 。

在运行插件之前TypeScript 编译会被触发。这是通过 `.vscode/launch.json` 文件内的 `preLaunchTask` 属性（声明了一个在启动调试之前执行的任务）。这个任务在 `.vscode/tasks.json` 内部定义。

说明: TypeScript 编译器在监视模式触发，就能够文件变动时重新编译。

插件启动

插件在一个带有 `Extension Development Host` 标题的窗口内启动。该窗口运行着 VS Code，更准确的说，是带有开发模式插件的 `Extension Host`。

可以在命令行通过 `extensionDevelopmentPath` 选项实现同样的效果。该选项告诉 VS Code 搜索插件的位置，比如，

```
code --extensionDevelopmentPath=_my_extension_folder .
```

一旦 `Extension Host` 启动了，VS Code 就把调试器附加于其上，启动调试。

这是按下 F5 后发生的：

1. `.vscode/launch.json` 告知要首次运行 `npm` 命令。
2. `.vscode/tasks.json` 通过脚本命令 `npm run compile`，声明了 `npm` 任务。
3. `package.json` 声明 `compile` 脚本，做为 `node ./node_modules/vscode/bin/compile -watch -p ./`。
4. 这会最终调用包含在 `node_modules` 的TypeScript 编译器，生成了 `out/src/extension.js` 和 `out/src/extension.js.map`。
5. 一旦TypeScript 编译任务结束，`code --extensionDevelopmentPath=${workspaceRoot}` 进程启动。
6. VS Code 的第二个实例在一个特殊窗口中启动，然后寻找一个 `${workspaceRoot}` 目录下的插件。

插件热加载

因为 TypeScript 编译器在监视模式下运行，所以 TypeScript 文件会在你改动时自动编译。可以在 VS Code 左侧状态栏里观察编译进程。当编译没有错误地结束时，你必须重新加载 Extension Development Host 来确保它发现了你的改变。你可以有以下选项：

- 点击调试重启，重新启动插件开发主机窗口。
- 在插件开发主机窗口内按下 `kbstyle(Ctrl+R)` (Mac: `kbstyle(Cmd+R)`)。

下一步

- [插件测试](#) - 学习如何写插件的单元和集成测试。
- [发布工具](#) - `vsce`命令行工具来发布插件
- [插件清单文件](#) - VS Code 插件清单文件指南
- [插件API](#) - 了解VS Code 可扩展的 API

常见问题

问：我如何才能在我的插件使用**VS Code** 新版本中引入的 **API**？

答：如果你的插件使用VS Code 新版本中引入的 API，你就不得不在插件 `package.json` 文件的 `engines` 字段中去声明依赖。

步骤如下：

- 设置 `package.json` 文件内具备 `engine` 字段的VS Code 最小版本号。
- 保证 `vscode` 模块的开发依赖 `>= 0.11.0` 。
- 像下面这样把 `postinstall` 脚本加到 `package.json` 文件里去：

```
"scripts": {
  "postinstall": "node ./node_modules/vscode/bin/install"
}
```

- 在插件根目录下键入 `npm install` 。
- `vscode` 模块会下载你声明的 `engine` 字段中 `vscode.d.ts` 的合适版本。
- 返回 VS Code，瞧一瞧你选择的特定版本 API是如何智能感知，并进行交互验证的。

安装扩展

个人扩展文件夹

VS Code 会在个人扩展文件夹中 `.vscode/extensions` 来寻找扩展组件。不同的平台其文件夹所在的位置也不同：

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

如果你想在VS Code 每次启动都能够加载你自己的扩展或者定制化信息，那么就需要在 `.vscode/extensions` 文件夹下新建一个文件夹，并把项目文件放进去。例如：`~/.vscode/extensions/myextension`。

单独与他人共享(旁加载)

如果你想单独和其他人分享你的扩展或者自定义配置，只需要简单地把生成器的结果文件发送给他们并让他们把文件放在他们自己的 `.vscode/extensions` 文件夹下。或者，你可以执行命令 `vsce package` 来打包你的扩展，该命令使用[vsce publishing tool](#)工具将扩展打包成 `.vsix` 文件，之后将该文件发送给他们即可。

安装已打包的扩展(`.vsix`)

你可以手动地安装已打包成 `.vsix` 文件的VS Code扩展。只需要使用VS Code的命令行并提供相关 `.vsix` 文件的路径。

```
code myextension.vsix
```

扩展将会被安装到个人扩展文件夹 `.vscode/extensions` 中，你可以一次安装多个扩展，只要在命令行中提供多个 `.vsix` 文件的路径。

你也可以通过在VS Code中打开 `.vsix` 文件来安装扩展。点击文件 > 打开文件... 或者 `kb(workbench.action.files.openFile)` 并选择 `.vsix` 扩展文件。

发布到商店中

如果你想将你的扩展与他人分享到VS Code 的[应用商店](#)中，可以使用[vsce publishing tool](#)工具来打包你的扩展并提交它。

下一步

- [应用商店](#) - 了解更多关于VS Code 的公开扩展
- [发布扩展](#) - 了解如何打包并发布你的扩展

扩展（以下简称插件）编写的基本方法和原则

VS Code 的插件 API 遵循一些会贯穿到整个 API 的基本方法和原则。

Promises

VS Code API 采用 `promises` 来进行异步操作。在插件当中，可以返回任何类型的 promise，比如 ES6, WinJS, A+ 等等。

API 当中，`Thenable` 类型用以依赖某个 promise 库。`Thenable` 是 `then` 属性的“公分母”。

大部分的 promise 使用都不是必要的，当 VS Code 运行一个插件时，可以像处理 `result type` 的一个 `Thenable` 来处理 `result type` 本身。当 promise 没必要用上时，API 会通过 `or - types` 给出提示。

```
provideNumber(): number | Thenable<number>
```

CancellationToken 取消标志

不稳定的状态经常触发操作异步操作，状态有可能在异步操作结束前改变。举个例子，智能感知开始计算，但是用户继续输入使得异步操作“过期”了。

给此类行为的 API 都会被传递 `CancellationToken` 参数，我们可以通过 `isCancellationRequested` 来检查取消状态，或者取消时通过 `onCancellationRequested` 得到通知。取消标志通常是函数调用的最后一个可选参数。

Disposable 释放模式

VS Code API 在其内部资源上使用 `释放模式`。这涉及到事件监听，命令行，UI 交互，还有各种语言贡献。

在实例内部，`setStatusbarMessage(value: string)` 返回一个可释放类型 `Disposable`，在调用 `dispose` 方法的时候它再一次移除了消息。

Events

VS Code API 内的事件是通过调用监听器函数来订阅的函数。订阅函数返回一个可释放类型 `Disposable`，他会把 `dispose` 方法上的监听器给移除掉。Events in the VS Code API are exposed as functions which you call with a listener-function to subscribe. Calls to subscribe return a `Disposable` which removes the event listener upon dispose.

```
var listener = function(event) {
    console.log("It happened", event);
};

// 开始监听
var subscription = fsWatcher.onDidDelete(listener);

// 更多
subscriptions.dispose(); // 停止监听
```

事件名采用 `on[Will|Did]VerNoun` 结构，意思是事件是否将要发生 (`onWill`) 或者已经发生 (`onDid`)，现在发生 (`verb`)，还有上下文 (`noun`)（除非上下文明确）。

一个 VS Code API 的例子是 `window.onDidChangeActiveTextEditor` 事件会在活动文本编辑器 (`noun`) 已经 (`onDid`) 改变了 (`verb`)

使用 Node.js 模块

你的插件允许在运行时依赖[Node.js](#)模块。和一个 `node` 模块本身相似，还可以增加一些依赖到 [package.json 插件清单](#) 的 `dependencies` 字段上去。

安装和打包

你安装VS Code插件的时候，VS Code 不会自动安装其依赖，所以你必须在发布前执行 `npm install`。插件的安装包会包含他内的所有依赖。你可以运行 `vsce ls` 去枚举所有 `vsce` 将会包含进安装包的文件。

下一步

- [插件清单文件](#) - VS Code `package.json` 插件清单文件指南
- [贡献关键点](#) - VS Code 贡献关键点 指南
- [激活事件](#) - VS Code 激活事件 指南

常见问题

问：我可以使用原生 **Node.js** 模块吗

答：一个 VS Code 插件安装包包含许多依赖。意味着如果你在 Windows 开发插件而且发布插件时依赖原生 Node.js 模块，那么你的插件会包含 Windows 上编译好的原生依赖。OS X、Linux 用户就不能使用该插件。

此刻奏效的唯一办法是把四个平台（Windows X86, X64, Linux, OSX）的二进制文件都放进插件，并且插件包含动态加载正确二进制文件的代码。

扩展（以下简称插件）测试

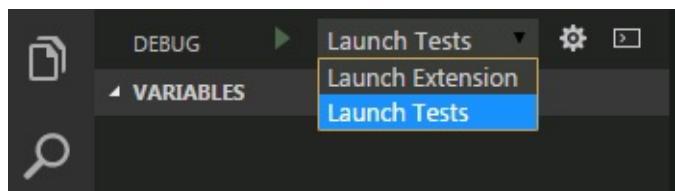
VS Code 支持符合 VS Code API 的插件测试的运行和调试。这些测试会在 VS Code 内部一个特殊实例内运行，Extension Development Host 有全套 API 的权限。我们把这些测试看做集成测试，因为他们有别于可以运行在一个 VS Code 单独窗口的单元测试。本篇文档有关 VS Code 集成测试。对于但与测试，你可以使用任何时髦的测试框架，比如 Mocha 或者 Jasmine。

Yo Code 测试脚手架

yo code 生成器 插件项目包含一个测试样例还有支持其运行的必要基础设施。

说明：接下来的文档建立在你已经创建了一个 TypeScript 或者 JavaScript 插件的基础上，但是有些文件命名可能比较特别。

在你创建一个插件，在 VS Code 上打开时，你可以从调试视图的顶部下拉菜单里进行 Launch Tests 配置



该配置被选中时，运行 `Debug: Start (kb(workbench.action.debug.start))`，VS Code 会在 Extension Development Host 实例上启动你的插件。你可以看到测试结果会在调试控制台上输出。

```

extension.test.ts test
1  //
2  // Note: This example test is leveraging the Mocha test framework.
3  // Please refer to their documentation on https://mochajs.org/ for help.
4  //
5
6  // The module 'assert' provides assertion methods from node
7  import * as assert from 'assert';
8
9  // You can import and use all API from the 'vscode' module
10 // as well as import your extension to test it
11 import * as vscode from 'vscode';
12 import * as myExtension from '../src/extension';
13
14 // Defines a Mocha test suite to group tests of similar kind together
15 suite("Extension Tests", () => {
16
17     // Defines a Mocha unit test
18     test("Something 1", () => {
19         assert.equal(-1, [1, 2, 3].indexOf(5));
20         assert.equal(-1, [1, 2, 3].indexOf(0));
21     });
22 });

DEBUG CONSOLE
/Users/bpasero/Development/monaco/.build/electron/Electron.app/Contents/MacOS/Electron --debugBrkPluginHost=22406 /Users/bpasero/Development/monaco --extensionDevelopmentPath=/Users/bpasero/Desktop/hello-world --extensionTestsPath=/Users/bpasero/Desktop/hello-world/out/test

Extension Tests
✓ Something 1
1 passing (1ms)
Loading development extension at /Users/bpasero/Desktop/hello-world

CALL STACK
BREAKPOINTS
    All Exceptions
    Uncaught Exceptions

```

该测试使用 [Mocha 测试框架](#) 来做为测试运行库。

该插件项目有一个包含 `index.js` 文件（配置 Mocha 测试库），`extension.test.js`（存在 `Something 1` 测试样例）的 `test` 文件夹。可以不编辑 `index.ts`，但是你也可以修改它来调整 Mocha 配置。

```

└── test
    ├── extension.test.ts
    └── index.ts

```

可以在 `test` 目录下创建更多的 `test.ts` 文件，之后他们会自动构建并运行在 `out/test` 目录下。测试库只会运行 `*.test.ts` 文件。

启动测试配置

`Launch Tests` 配置在项目的 `.vscode\launch.json` 文件中定义。和带有 `--extensionTestsPath` 参数的 `Launch Extension` 配置类似，指向编译好的测试文件（假定这是一个 TypeScript 项目）。

```
{
  "name": "Launch Tests",
  "type": "extensionHost",
  "request": "launch",
  "runtimeExecutable": "${execPath}",
  "args": ["--extensionDevelopmentPath=${workspaceRoot}", "--extensionTestsPath=${workspaceRoot}/out/test" ],
  "stopOnEntry": false,
  "sourceMaps": true,
  "outDir": "${workspaceRoot}/out/test",
  "preLaunchTask": "npm"
}
```

给 Extension Development Host 传递参数

可以在启动配置的参数列表开头，插入测试用例应该打开的文件或者目录

```
"args": ["file or folder name", "--extensionDevelopmentPath=${workspaceRoot}", "--extensionTestsPath=${workspaceRoot}/out/test" ],
```

这样，你就可以运行带有可预测内容和目录结构的测试。

在插件包内排除测试文件

你想要分享插件的话，插件不会包含测试文件吧。[.vscodeignore](#) 文件允许你在通过 [vsce](#) 发布工具打包，发布插件的时候排除测试文件。默认地，[yo code](#) 生成好的插件项目排除了 `test`，和 `out/test` 目录。

```
out/test/**
test/**
```

在 [Travis CI](#) 构建机器上自动运行测试

你可以在 [Travis CI](#) 这样的构建机器上自动运行插件测试。

[vscode npm](#) 模块提供了测试命令，用来开启插件测试自动化：

- 下载并解压 VS Code
- 启动 VS Code 内部的插件测试
- 在控制台输出结果，测试成功或者失败返回结束码。

为了开启这条测试命令，打开 `package.json`，增加下列入口到 `scripts` 字段上：

```
"test": "node ./node_modules/vscode/bin/test"
```

可以轻易的通过这样的顶级 `.travis.yml` 配置来开启 Travis Ci：

```
sudo: false

os:
  - osx
  - linux

before_install:
  - if [ $TRAVIS_OS_NAME == "linux" ]; then
      export CXX="g++-4.9" CC="gcc-4.9" DISPLAY=:99.0;
      sh -e /etc/init.d/xvfb start;
      sleep 3;
    fi

install:
  - npm install
  - npm run vscode:prepublish

script:
  - npm test --silent
```

以上脚本会在 Linux 和 Mac OS X 上跑测试用例。需要注意为了在 Linux 上运行测试用例，你需要有 `before_install` 配置使得 Linux 在构建时启动 VS Code。

说明：现在我们不支持在 Windows 上跑测试用例（比如使用 Appveyor）。

有一些用来配置测试库的可选环境变量：

名称	描述
<code>CODE_VERSION</code>	跑测试用例的 VS Code 版本（比如 <code>0.10.10</code> ）
<code>CODE_DOWNLOAD_URL</code>	可以跑测试用例的 VS Code 的完整下载地址
<code>CODE_TESTS_PATH</code>	要排除的测试文件夹
<code>CODE_TESTS_WORKSPACE</code>	打开测试实例的工作区间

下一步

- [插件调试](#) - 学习如何运行并调试插件
- [vsce](#) - 通过 VSCE 命令行工具发布插件.

- [插件清单文件](#) - VS Code 插件清单文件指南
- [插件 API](#) - 了解 VS Code 可扩展性 API

常见问题

暂无。

可拓展途径

VS Code 拥有一个极具拓展性，并且有许多方法去拓展的模块。但是，我们没有为插件作者们提供底层 UI DOM 的权限。基于 VS Code，我们继续努力优化顶层 Web 技术的使用去交付一个高可用，高响应的编辑器，而且当 DOM 技术还有 VS Code 发展时我们会继续调优。VS Code 也针对许多智能感知这样的场景内建了一套 UI 组件。这样的话，那些不同编程语言间的体验是流畅的，而且插件和插件作者们不需要另外开发。

我们意识到这个途径可能初始感觉限制到插件作者。同时我们也总是在寻找提高可拓展性，拓展插件可用功能。我们十分期待聆听您的反馈和创意！

核心概念

当我们开始着手提高 VS Code 可拓展性的时候，我们有许多考量。下面的章节告知了我们许多关键决定的上下文环境。在 API 内，我们也拥有一份描绘许多已采纳的核心模式轮廓的文档

稳定性 - 插件隔离

插件很美妙但是也会影响启动性能或者 VS Code 自身的总体性能。为了避开这些问题，VS Code 在一个单独的 `extension host process` 进程中加载并运行插件。一个行为异常的插件不能影响到 VS Code，特别是启动时间。

我们和终端用户用心构建了这个架构，可以看出此架构确保终端用户总是能够控制 VS Code：用户可以在任何时候打开，输入或者保存文件，不管插件干了什么，VS Code 都可以保证 UI 的响应式。

`extension host` 是一个 Node.js 进程，它把 VS Code API 保护给插件作者们。在 `extension host` 内部，VS Code 为插件运行提供了调试支持。

性能 - 插件激活

VS Code 尽可能晚地加载插件，当前时段没有使用的插件不会被加载，因此不消耗内存。为了支持插件懒加载，VS Code 定义了所谓的 `激活事件`。一个 `激活事件` 会被 VS Code 的特定活动触发，同时插件可以定义它需要激活的哪些事件。举个例子，一个编辑 markdown 的插件只会在用户打开一个 markdown 文件的时候激活

插件清单

为了懒激活插件，VS Code 需要一份插件描述 `extension manifest`，相当于在 `package.json` 文件增加了一些特定字段。这个包含触发插件加载的激活事件。VS Code 提供一套插件可以添加的 贡献点。举个例子，当添加一个 VS Code 命令的时候，你需要通过 `commands` 贡献点提供命令声明。在 `package.json` 内部声明了插件贡献。VS Code 在启动和 UI 初始化的时候读取并解析插件清单。

因为 `extensioin host` 是一个 Node.js 进程，所以你可以在插件内使用 Node API，甚至实现一个插件的时候可以复用 Node.js 模块。可以在 `package.json` 里声明模块依赖，然后用 `npm` 去安装一个Node.js 模块。

更多细节请参考 [package.json 贡献点指南](#)。

可拓展 API

在一个单独进程运行插件的办法，使得 VS Code 严格控制了暴露给扩展的 API。更多当前 API 细节请参考[可拓展 API概述](#)。

VS Code 使用 Web 技术（HTML，CSS）来实现，因为 Web 技术在修改UI，写样式的时候十分强大。你可以轻易添加节点到 DOM 中，使用 CSS 实现一个个性化外观。但是，当发展成一个像 VS Code 应用的时候就显得不够了。以至于结构改变，和 UI 高耦合的插件会卡住。出于这一点，VS Code 选择保守性办法，确保 DOM 不暴露给插件作者们。

插件协议

VS Code 一个常见的插件模式是用一个和 VS Code 以某种协议通信的单独进程里，去执行插件代码。比如说，语言服务和调试适配器。通常，这个协议通过`stdin/stdout`，使用 JSON 载荷，来和进程间通信。使用单独进程会提供一个帮助 VS Code 维持核心编辑器稳定性的边界隔离区。另外，使用插件作者们可以对于特殊插件实现最合适的语言。

下一步

- [第一个插件](#) - 尝试生成一个简单的 Hello world 插件
- [插件API](#) - 了解VS Code 可拓展 APIs
- [插件样例](#) - 你可以查看并构建的插件样例列表

常见问题

暂无

扩展API

- 概述
- 扩展manifest文件
- 扩展点
- 激活事件
- vscode-api
- vscode-api-命令
- api调试

可扩展性参考

本节将会详细介绍VS Code可扩展性的各种功能，并深入到其内部细节。在开始之前，建议回顾之前关于[扩展部分](#)以及范例['Hello World'](#)。

查看VS Code扩展运行的最简单方法是通过[扩展市场](#)。当你编写好了你的第一个插件后或者准备分享你的[定制化信息](#)后，那么你可以把它[发布出去](#)，供他人下载并安装。

可扩展性参考文档

在本节中将会讲述以下几个主题：

主题	描述
package.json 扩展清单	每个VS Code扩展必须在其根目录下面有一个 <code>package.json</code> 文件。该文件提供本文件的结构和相关必填字段。
贡献点	基于 <code>project.json</code> 文件，你可以在某些方面添加新的贡献点，比如命令、主题、调试器等。
激活事件	VS Code会延迟激活扩展。该文档概述了 <code>project.json</code> 中支持的激活选项，比如加载了某个指定类型的文件、或者命中了某个命令等。
API vscode	查看完整的VS Code的API文档。
API 命令	查看VS Code命令API文档.
API 调试	了解关于VS Code中集成调试器的细节。

常见问题

暂无

扩展清单文件 - package.json

每个VS Code扩展需要一个清单文件 `package.json`，该文件位于扩展的根目录中。

字段

名称	是否必要	类型	说明
<code>name</code>	是	<code>string</code>	扩展的名称，该名称必须为小写且不能有空格。
<code>version</code>	是	<code>string</code>	SemVer 兼容版本。
<code>publisher</code>	是	<code>string</code>	发布人名字
<code>engines</code>	是	<code>object</code>	一个至少包含 <code>vscode</code> 键值对的对象，该键表示的是本扩展可兼容的VS Code的版本，其值不能为 *。比如 <code>^0.10.5</code> 表示扩展兼容VS Code的最低版本是 <code>0.10.5</code> 。
<code>license</code>	否	<code>string</code>	参考 npm's 文档 。如果你确实需要在扩展根目录下有一个授权文档，那么应该把 <code>license</code> 值设为 <code>"SEE LICENSE IN <filename>"</code> 。
<code>displayName</code>	否	<code>string</code>	用于在扩展市场中本扩展显示的名字。
<code>description</code>	否	<code>string</code>	一份简短的说明，用来说明本插件是什么以及做什么
<code>categories</code>	否	<code>string[]</code>	你希望你的扩展属于哪一类，只允许使用这几种值：[Languages, Snippets, Linters, Themes, Debuggers, Other]
<code>keywords</code>	否	<code>array</code>	一组关键字或者标记，方便在市场中查找。
<code>galleryBanner</code>	否	<code>object</code>	帮助格式化市场标题以匹配你的图标，详情如下。
<code>preview</code>	否	<code>boolean</code>	在市场中把本扩展标记为预览版本。
<code>main</code>	否	<code>string</code>	扩展的入口点。
<code>contributes</code>	否	<code>object</code>	一个描述扩展 贡献点 的对象。
<code>activationEvents</code>	否	<code>array</code>	一组用于本扩展的 激活事件 。

<code>dependencies</code>	否	object	你的扩展所需的任何运行时的Node.js依赖项，和 npm's <code>dependencies</code> 一样。
<code>devDependencies</code>	否	object	你的扩展所需的任何开发的Node.js依赖项。和 npm's <code>devDependencies</code> 一样。
<code>extensionDependencies</code>	否	array	一组本扩展所需的其他扩展的ID值。扩展的ID值始终是 <code> \${publisher}.\${name} </code> 。比如： <code> vscode.csharp </code> 。
<code>scripts</code>	否	object	和 npm's <code>scripts</code> 一样，但还有一些额外 VS Code特定字段 。
<code>icon</code>	否	string	一个128x128像素图标的路径。

也可以查看[npm's `package.json` 参考文档](#)。

范例

这里有一个完整的 `package.json` ：

```
{
  "name": "Spell",
  "displayName": "Spelling and Grammar Checker",
  "description": "Detect mistakes as you type and suggest fixes - great for Markdown .",
  "icon": "images/spellIcon.svg",
  "version": "0.0.19",
  "publisher": "seanmcbreen",
  "galleryBanner": {
    "color": "#0000FF",
    "theme": "dark"
  },
  "license": "SEE LICENSE IN LICENSE.md",
  "bugs": {
    "url": "https://github.com/Microsoft/vscode-spell-check/issues",
    "email": "smcbreen@microsoft.com"
  },
  "homepage": "https://github.com/Microsoft/vscode-spell-check/blob/master/README.md"

  ,
  "repository": {
    "type": "git",
    "url": "https://github.com/Microsoft/vscode-spell-check.git"
  },
  "categories": [
    "Linters", "Languages", "Other"
  ],
  "engines": {
    "vscode": "0.10.x"
  },
  "main": "./out/extension",
}
```

```

"activationEvents": [
    "onLanguage:markdown"
],
"contributes": {
    "commands": [
        {
            "command": "Spell.suggestFix",
            "title": "Spell Checker Suggestions"
        }
    ],
    "keybindings": [
        {
            "command": "Spell.suggestFix",
            "key": "Alt+."
        }
    ]
},
"scripts": {
    "vscode:prepublish": "node ./node_modules/vscode/bin/compile",
    "compile": "node ./node_modules/vscode/bin/compile -watch -p ./"
},
"dependencies": {
    "teacher": "^0.0.1"
},
"devDependencies": {
    "vscode": "^0.11.x"
}
}
}

```

市场呈现要点

为了让你自己的扩展在[VS Code市场]中看起来更好，这里有几个要点和建议。

始终使用最新的 `vsce`，即用 `npm install -g vsce` 可保证最新。

在你的扩展根目录中编写一个 `README.md` 的MAERKDOWN文档，这样我们会在市场中的扩展信息中显示其中的内容。你可以在 `README.md` 中提供图片的相对路径链接。

这里有几个例子做说明：

1. [Spell-Checker](#)
2. [MD Tools](#)

请提供一个良好的显示名称和描述。这对于市场和产品显示都非常重要。在VS Code中这些字符串能用于文本搜索，并且有相关关键字将有很大帮助。

```

    "displayName": "Spelling and Grammar Checker",
    "description": "Detect mistakes as you type and suggest fixes - great for Markdown
  .",

```

在市场页面头中，有一个图标以及对比色横幅也会让扩展看起来非常棒，`theme` 属性指的是在横幅中使用的字体：`dark` 或者是 `light`。

```

  "icon": "images/spellIcon.svg",
  "galleryBanner": {
    "color": "#5c2d91",
    "theme": "dark"
  },

```

你也可以设置一些其他的链接(错误、主页、代码库)，他们会在市场中资源一栏所呈现出来。

```

  "license": "SEE LICENSE IN LICENSE.md",
  "bugs": {
    "url": "https://github.com/Microsoft/vscode-spell-check/issues"
  },
  "homepage": "https://github.com/Microsoft/vscode-spell-check/blob/master/README.md"

  "repository": {
    "type": "git",
    "url": "https://github.com/Microsoft/vscode-spell-check.git"
  }

```



市场资源链接	package.json 属性
支持	<code>bugs:url</code>
开始页	<code>repository:url</code>
主页	<code>homepage</code>
授权	<code>license</code>

为你的扩展设置 `category`。在市场中，同一个 `category` 将会放在一起，这样就方便过滤和查找。

注意为你的扩展使用正确的值，只能使用这些值 [Languages, Snippets, Linters, Themes, Debuggers, Other]

```

  "categories": [
    "Linters", "Languages", "Other"
  ],

```

结合扩展贡献

`yo code` 生成器能让你轻松打包文本主题、颜色设置和代码片段并创建新的扩展。当生成器运行时，它为每个选项创建一个完整的独立扩展包。然而很多时候我们更倾向于创建一个包含多个贡献点的扩展。比如说，如果你希望为一门新语言添加支持，则希望为用户提供带有代码着色的语言定义和代码段功能，甚至可以提供调试支持。

要使能够结合扩展贡献，仅仅只需要编辑已经存在的清单文件 `package.json` 并添加新的贡献点和关联文件即可。

以下是一个扩展的清单文件，它包含了Latex语言定义(语言标识符和文件扩展名)、着色器(`grammar`)和代码段。

```
{
  "name": "language-latex",
  "description": "LaTeX Language Support",
  "version": "0.0.1",
  "publisher": "someone",
  "engines": {
    "vscode": "0.10.x"
  },
  "categories": [
    "Languages",
    "Snippets"
  ],
  "contributes": {
    "languages": [
      {
        "id": "latex",
        "aliases": ["LaTeX", "latex"],
        "extensions": [".tex"]
      }
    ],
    "grammars": [
      {
        "language": "latex",
        "scopeName": "text.tex.latex",
        "path": "./syntaxes/latex.tmLanguage"
      }
    ],
    "snippets": [
      {
        "language": "latex",
        "path": "./snippets/snippets.json"
      }
    ]
  }
}
```

注意，扩展的清单文件中 `categories` 属性现在可以同时包含 `Languages` 和 `Snippets`，这样方便在市场中查找和过滤。

要点 确保你的多个贡献点使用的是相同的标识符。在上例中，三个贡献点都是使用“`latex`”作为语言标识符。这让VS Code知道语法着色器和代码段是用于LaTeX语言并当编辑LaTeX文件时激活它。

下一步

要想了解更多关于VS Code可扩展性模型，可以查看这些主题：

- [贡献点](#) - VS Code 贡献点参考文档
- [激活事件](#) - VS Code 激活事件文档
- [扩展市场](#) - 了解更多的VS Code扩展市场

常见问题

暂无

Contribution Points - package.json

This document covers the various contribution points that are defined in the `package.json extension manifest`.

这篇文档包含了 `package.json extension manifest` 中 `contribution` 选项的所有可用字段。

- `configuration`
- `commands`
- `keybindings`
- `languages`
- `debuggers`
- `grammars`
- `themes`
- `snippets`
- `jsonValidation`

`contributes.configuration`

Contribute configuration keys that will be exposed to the user. The user will be able to set these configuration options either from User Settings or from the Workspace Settings.

When contributing configuration keys, a JSON schema describing these keys is actually contributed. This ensures the user gets great tooling support when authoring VS Code settings files.

You can read these values from your extension using

```
vscode.workspace.getConfiguration('myExtension').
```

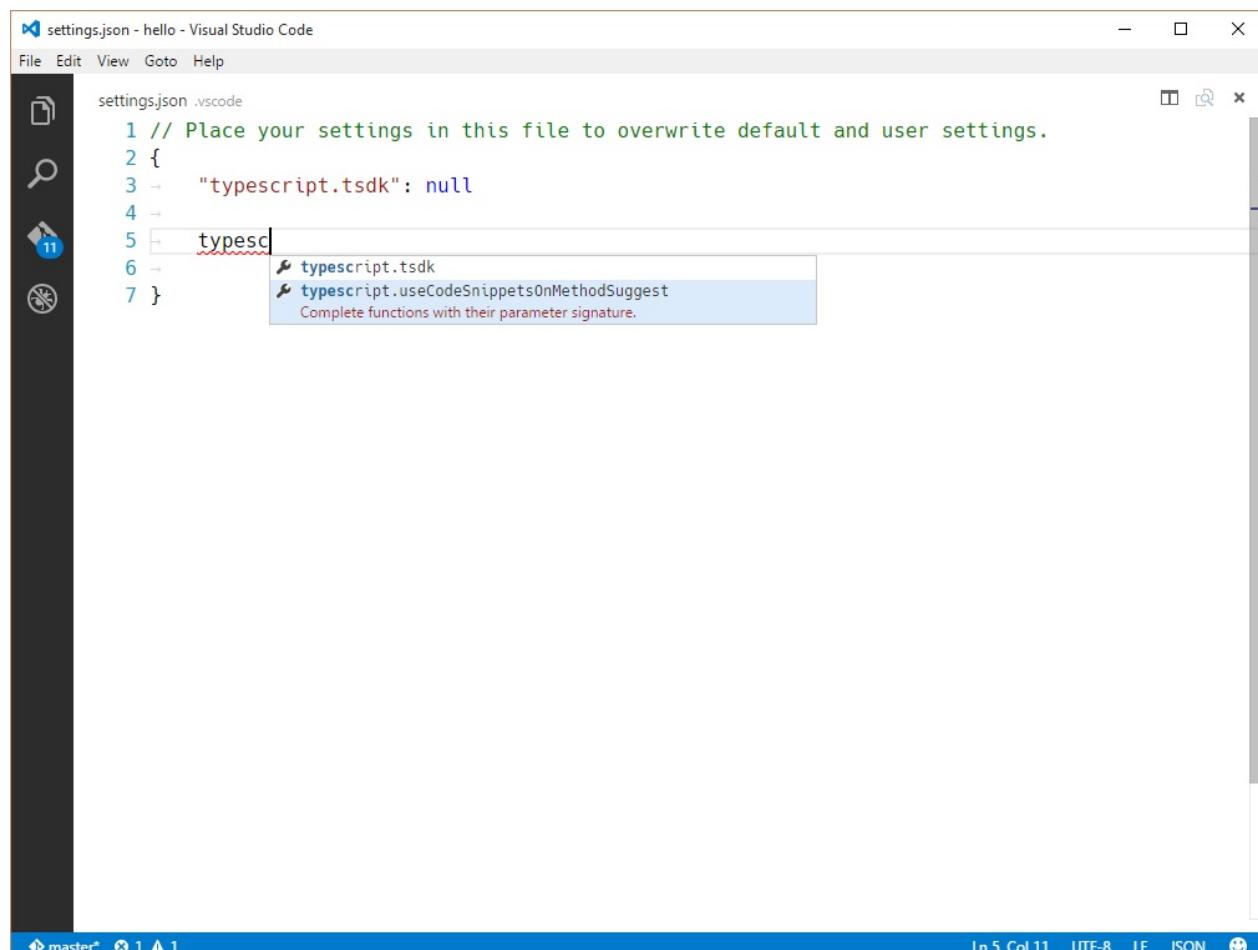
`contributes.configuration` 选项会被暴露给用户。用户能够在“用户设置”或“工作区设置”面板中设置这些配置选项。

在配置 `contributes.configuration`，同时也配置了这些选项的一种JSON模式的描述。这确保了用户在设置VS Code 选项文件时能够获得更好的工具支持。

译者注：在配置 `contributes.configuration` 时，实际上配置了某个选项的诸如类型/默认值／介绍等选项，这样用户在配置 setting 文件时VS Code能够根据这些配置来提示用户。

例子

```
// package.json
...
"contributes": {
  "configuration": {
    "type": "object",
    "title": "TypeScript configuration",
    "properties": {
      "typescript.useCodeSnippetsOnMethodSuggest": {
        "type": "boolean",
        "default": false,
        "description": "Complete functions with their parameter signature."
      },
      "typescript.tsdk": {
        "type": ["string", "null"],
        "default": null,
        "description": "Specifies the folder path containing the tsserver and lib*.d.ts files to use."
      }
    }
  }
}
```



contributes.configurationDefaults

Contribute default language specific editor configurations. This will override default editor configurations for the provided language.

The following example contributes default editor configurations for the `markdown` language:

设置默认语言特定的编辑器配置。这将覆盖所提供语言的默认编辑器配置。下面的例子配置了 `markdown` 语言的默认编辑器配置

例子

```
// package.json
"contributes": {
  "configurationDefaults": {
    "[markdown)": {
      "editor.wordWrap": "on",
      "editor.quickSuggestions": false
    }
  }
}
```

contributes.commands

Contribute an entry consisting of a title and a command to invoke to the Command Palette (⌘P) . You can also optionally define a `category` string which will prefix the command title and allow easy grouping within the Command Palette drop-down.

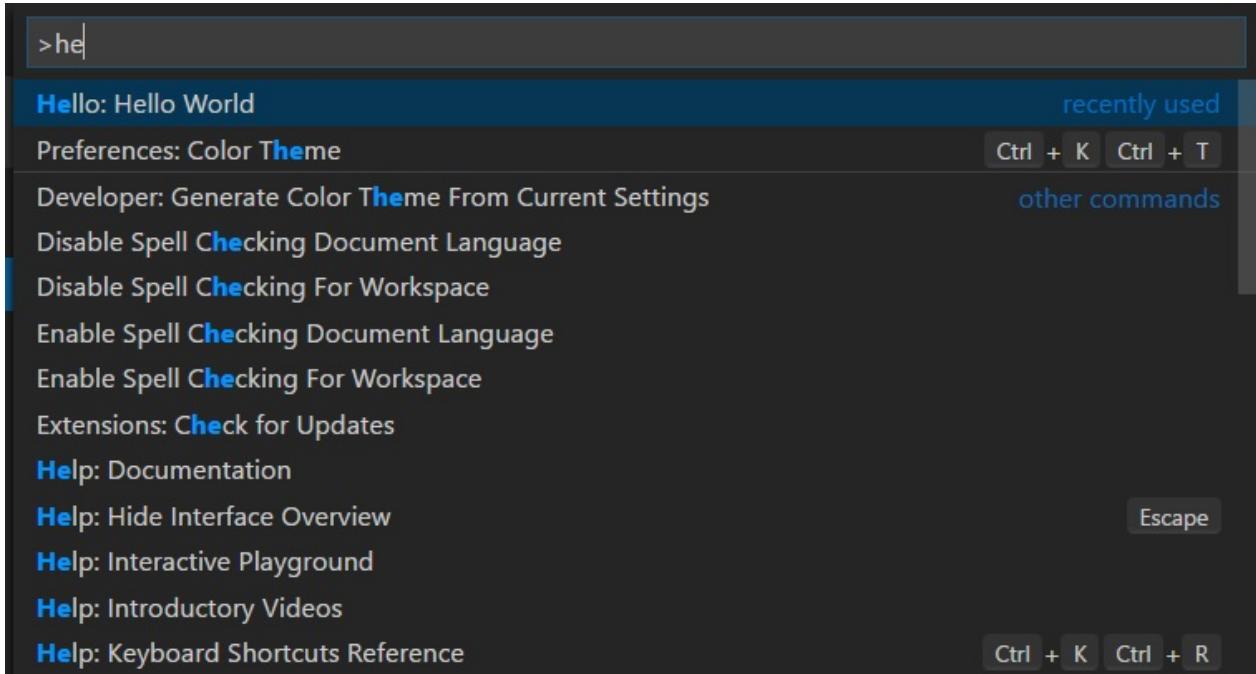
提供了一个由 `commands` 和 `title` 字段组成的条目，用于在 命令面板(⌘P) 中调用。你同时可以选择添加一个 `category` 字符串，此字符串会用作命令的前缀，同时在命令面板展开的时候方便进行命令的分组。

Note: When a command is invoked (from a key binding or from the **Command Palette**), VS Code will emit an activationEvent `onCommand:${command}` .

Note: 当一个命令被调用的时候（无论是通过按键绑定还是 命令面板），VS Code 将触发一个 activationEvent `onCommand:${command}` 。

Example

```
...
"contributes": {
  "commands": [
    {
      "command": "extension.sayHello",
      "title": "Hello World"
    }
  ]
}
...
```



contributes.menus

Contribute a menu item for a command to the editor or Explorer. The menu item definition contains the command that should be invoked when selected and the condition under which the item should show. The latter is defined with the `when` clause which uses the key bindings [when clause contexts](#). In addition to the mandatory `command` property, an alternative command can be defined using the `alt`-property. It will be shown and invoked when pressing Alt while hovering over a menu item. Last, a `group`-property defines sorting and grouping of menu items. The `navigation` group is special as it will always be sorted to the top/beginning of a menu.

定义了编辑器或者资源管理器中一个命令在的菜单项。菜单项定义了选择时应该调用的命令以及不同情况下命令的的显示方式。后者使用 `when` 子句，并结合 [子句上下文](#) 通过按键绑定来定义的。除了必要的 `command` 属性，我们还可以使用 `alt` 来替换。当鼠标悬浮在菜单项上时，按住 Alt 可以显示和调用。最后，`group` 属性定义了菜单项的分组与排序。

`navigation` 组比较特殊，因为它会一直排在最顶部或者最开始的位置。

Currently extension writers can contribute to:

目前拓展的开发者能够修改这些部分：

- 全局的命令面板 - `commandPalette`
- 资源管理器环境菜单 - `explorer/context`
- 编辑器环境菜单 - `editor/context`
- 编辑器的标题菜单栏 - `editor/title`
- 编辑器标题的上下文菜单 - `editor/title/context`
- 调试调用堆栈的上下文菜单 - `debug/callstack/context`
- **SCM 标题菜单** - `scm/title`
- **SCM 资源组菜单** - `scm/resourceGroup/context`
- **SCM 资源菜单** - `scm/resource/context`
- **SCM 修改标题时的菜单** - `scm/change/title`
- **View 面板标题菜单** - `view/title`
- **View 面板中项目的菜单** - `view/item/context`

Note: When a command is invoked from a (context) menu, VS Code tries to infer the currently selected resource and passes that as a parameter when invoking the command. For instance, a menu item inside the Explorer is passed the URI of the selected resource and a menu item inside an editor is passed the URI of the document.

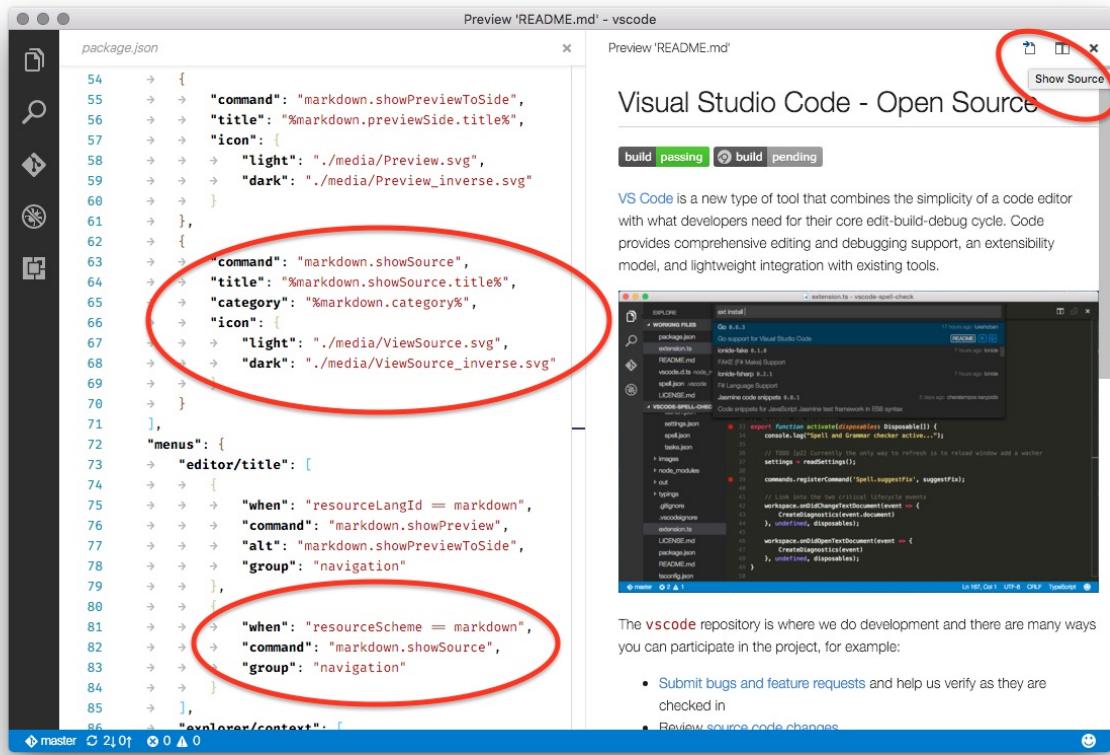
Note: 当一个命令通过（上下文）菜单调用时，VS Code 会尝试推断出当前选定的资源，并以参数的形式传递给被调用的命令

In addition to a title, commands can also define icons which VS Code will show in the editor title menu bar.

除了标题之外，命令还可以定义 VS Code 在编辑器标题菜单栏中使用的图标。

例子

```
"contributes": {
  "menus": {
    "editor/title": [
      {
        "when": "resourceLangId == markdown",
        "command": "markdown.showPreview",
        "alt": "markdown.showPreviewToSide",
        "group": "navigation"
      }
    ]
  }
}
```



Context specific visibility of Command Palette menu items - 命令面板中的命令可见性

When registering commands in package.json, they will automatically be shown in the Command Palette ($\text{Shift} + \text{Command} + \text{P}$). To allow more control over command visibility, there is the `commandPalette` menu item. It allows you to define a `when` condition to control if a command should be visible in the Command Palette or not.

当我们在 `package.json` 中定义命令的时候，它们就会自动显示在命令面板中($\text{Shift} + \text{Command} + \text{P}$)，为了更好地控制命令的可见性，这里有一个 `commandPalette` 的菜单选项。它允许你定义一个 `when` 条件来控制命令是否应该在命令面板中显示。

The snippet below makes the 'Hello World' command only visible in the **Command Palette** when something is selected in the editor:

下面的代码片段使得 `Hello World` 命令只有当用户在编辑器中选择了某些东西的时候才会在命令面板中显示出来。

```

"commands": [
    "command": "extension.sayHello",
    "title": "Hello World"
],
"menus": {
    "commandPalette": [
        "command": "extension.sayHello",
        "when": "editorHasSelection"
    ]
}
}

```

Sorting of groups - 组排序

Menu items can be sorted into groups. They are sorted in lexicographical order with the following defaults/rules.

菜单项可以被分成组。它们按照以下默认值/规则按字典顺序排序。

The context menu of the editor has these default:

编辑器的上下文菜单具有以下默认值：

- **navigation** - `navigation` 组无论何时都会被排在第一位。
- **1_modification** - 接下来的这个组包含来修改你代码的一些命令
- **9_cutcopypaste** - 最后这个组包含了最基本的编辑命令



You can add menu items to these groups or add new groups of menu items in between, below, or above. Only the editor context menu allows this grouping control.

您可以将菜单项添加到这些组中，或者在这些组中间、下面或者上面添加新的菜单项组。只有编辑器上下文菜单允许这个分组控制。

Sorting inside groups - 组内排序

组内的顺序取决于标题或顺序属性。菜单项的组内顺序通过将 `@<number>` 附加到组标识符指定，如下所示：

```
"editor/title": [{
  "when": "editorHasSelection",
  "command": "extension.Command",
  "group": "myGroup@1"
}]
```

contributes.keybindings

Contribute a key binding rule defining what command should be invoked when the user presses a key combination. See the [Key Bindings](#) topic where key bindings are explained in detail.

提供的按键绑定规则定义了按下组合键时应当调用的命令。请参阅 [按键绑定](#) 主题，其中详细介绍了按键绑定。

Contributing a key binding will cause the Default Keyboard Shortcuts to display your rule, and every UI representation of the command will now show the key binding you have added. And, of course, when the user presses the key combination the command will be invoked.

设置键绑定将导致默认键盘快捷键显示您的规则，并且该命令的每个UI表示现在将显示您添加的键绑定。当然，当用户按下组合键时，该命令将被调用。

Note: Because VS Code runs on Windows, Mac and Linux, where modifiers differ, you can use "key" to set the default key combination and overwrite it with a specific platform.

Note: 由于VS代码在Windows，Mac和Linux上运行，其中修饰符不同，因此你可以使用“key”来设置默认组合键并依照特定的平台来覆盖它。

Note: When a command is invoked (from a key binding or from the Command Palette), VS Code will emit an activationEvent `onCommand:${command}`.

Note: 当一个命令被调用的时候（无论是通过按键绑定还是命令面板），VS Code 将触发一个 activationEvent `onCommand:${command}`。

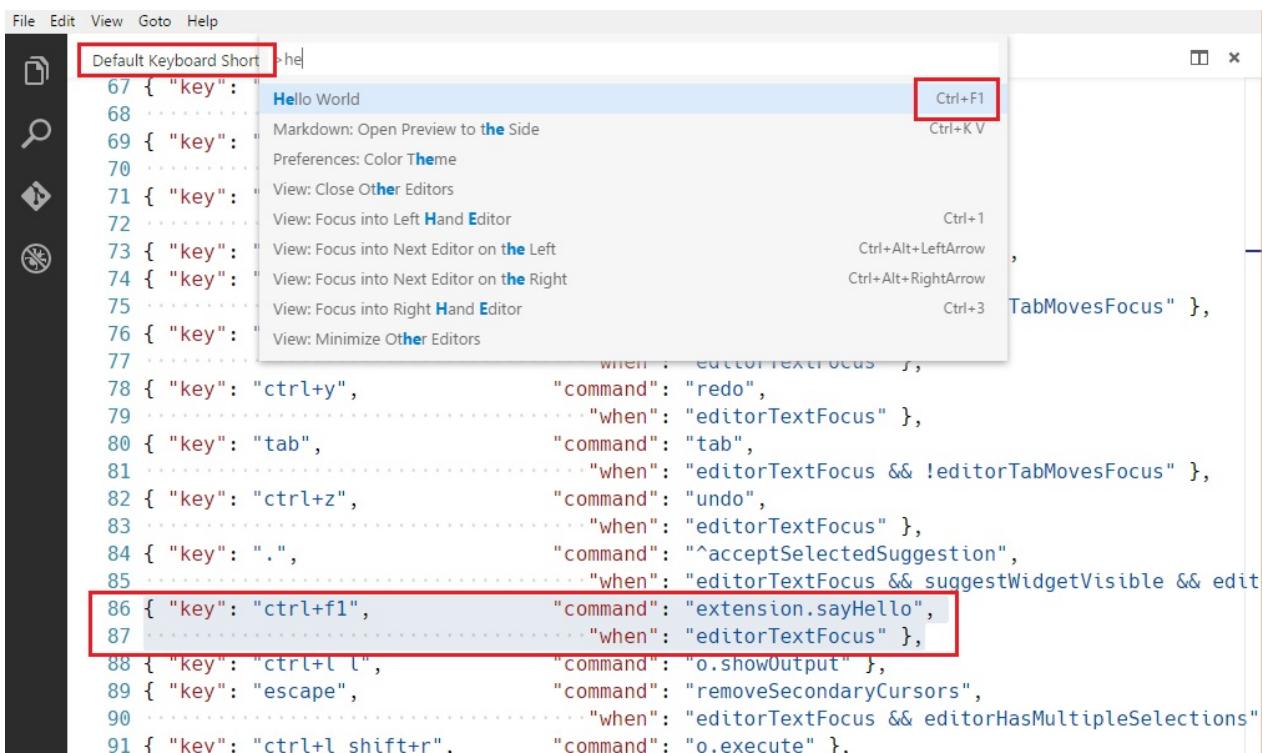
例子

Defining that `kbstyle(Ctrl+F1)` under Windows and Linux and `kbstyle(Cmd+F1)` under Mac trigger the `extension.sayHello` command:

定义了 Windows 和 Linux 下的 `kbstyle(Ctrl+F1)` 和 Mac 下的 `kbstyle(Cmd+F1)` 会触发 `extension.sayHello` 命令。

```
```json
"contributes": {
 "keybindings": [
 {
 "command": "extension.sayHello",
 "key": "ctrl+f1",
 "mac": "cmd+f1",
 "when": "editorTextFocus"
 }
]
}
```

```



contributes.languages

Contribute the definition of a language. This will introduce a new language or enrich the knowledge VS Code has about a language.

提供一种语言的定义。这会引入一门新的语言或者提升VS Code关于一门语言的认知。

In this context, a language is basically a string identifier that is associated to a file (See `TextDocument.getLanguageId()`).

在这种情况下，语言基本上是与文件关联的字符串标识符（参阅 `TextDocument.getLanguageId()`）。

VS Code uses three hints to determine the language a file will be associated with. Each "hint" can be enriched independently:

VS代码使用三个提示来确定文件将与之关联的语言。每个“提示”都可以独立显示：

1. the extension of the filename (`extensions` below)
2. the filename (`filenames` below)
3. the first line inside the file (`firstLine` below)
4. 文件名的扩展名（下面的提到的 `extensions`）
5. 文件名（下面的提到的 `filename`）
6. 文件中的第一行（下面的提到的 `firstLine`）

When a file is opened by the user, these three rules are applied and a language is determined. VS Code will then emit an activationEvent `onLanguage:${language}` (e.g. `onLanguage:python` for the example below)

当一个文件被用户打开的时候，这三种规则将被应用同时将确定出一种语言。此时VS Code 将会触发一个 activationEvent `onLanguage:${language}` (例如下面提到的例子 `onLanguage:python`)

The `aliases` property contains human readable names under which the language is known. The first item in this list will be picked as the language label (as rendered in the status bar on the right).

Example

The `aliases` property contains human readable names under which the language is known. The first item in this list will be picked as the language label (as rendered in the status bar on the right).

`alias` 属性包含了可识别语言的人类可读名称。该列表中的第一个项目将被选为语言标签（在右侧的状态栏中显示）。

The `configuration` property specifies a path to the language configuration file. The path is relative to the extension folder, and is typically `./language-configuration.json`. The file uses the JSON format and can contain the following properties:

`configuration` 属性指定了语言配置文件的路径。路径相对于扩展文件夹，通常是 `./language-configuration.json`。该文件使用JSON格式，可以包含以下属性：

- `comments` - 定义了表示注释的符号

- `blockComment` - 用于标记块注释的开始和结束标记。由“切换块注释”命令使用。
- `lineComment` - 用于标记行注释的开始标记。由“添加行注释”命令使用。
- `brackets` - 定义影响括号内代码缩进的括号符号。输入新行时，由编辑器用于确定或更新新的缩进级别。
- `autoClosingPairs` - 定义自动关闭功能的打开和关闭符号。当输入一个打开的符号时，编辑器会自动插入关闭符号。自动关闭对可选地使用 `notIn` 参数来禁用字符串或注释中的对。
- `surroundingPairs` - 定义了选定字符串的打开和关闭符号。
- `folding` - 定义何时以及如何在编辑器中折叠代码。
 - `offSide` - 尾随代码段的空行属于下一个折叠部分（用于基于缩进的语言，如 Python 或 F）
 - `markers` - 用于识别代码中自定义折叠区域的标记的正则表达式

If your language configuration file name is or ends with `language-configuration.json` , you will get validation and editing support in VS Code.

如果您的语言配置文件名是或以 `language-configuration.json` 结尾，您将在VS Code中得到验证和编辑支持。

例子

```
...
"contributes": {
  "languages": [
    {
      "id": "python",
      "extensions": [ ".py" ],
      "aliases": [ "Python", "py" ],
      "filenames": [ ... ],
      "firstLine": "^\#!.*\bpython[0-9.-]*\b",
      "configuration": "./language-configuration.json"
    }
  ]
}
```

`language-configuration.json`

```
{
  "comments": {
    "lineComment": "//",
    "blockComment": [ /*, */
      "/*", */
      "*/"
    ]
  },
  "brackets": [
    ["{", "}"],
    ["[", "]"],
    ["(", ")"]
  ],
  "autoClosingPairs": [
    ["{", "}"],
    ["[", "]"],
    ["(", ")"],
    { "open": "'", "close": "'", "notIn": ["string", "comment"] },
    { "open": "/*", "close": " */", "notIn": ["string"] }
  ],
  "surroundingPairs": [
    ["{", "}"],
    ["[", "]"],
    ["(", ")"],
    ["<", ">"],
    ["'", "'"]
  ],
  "folding": {
    "offSide": true,
    "markers": {
      "start": "\\\\s*//#region",
      "end": "\\\\s*//#endregion"
    }
  }
}
```

contributes.debuggers

Contribute a debugger to VS Code. A debugger contribution has the following properties:

为 VS Code 设置一个调试器。 调试器可以具有以下属性。

- `type` 是用于在启动配置中标识此调试器的唯一ID。
- `label` 标签是用户界面中此调试器的用户可见名称。
- `program` 指的是实现了VS Code debug协议的调试器文件。
- `runtime` 如果调试适配器的路径不是可执行文件，但需要运行时。
- `configurationAttributes` 是特定于此调试器的启动配置参数的模式。
- `initialConfigurations` 列出了用于填充初始 `launch.json` 的启动配置。
- `configurationSnippets` 列出了在编辑 `launch.json` 时通过IntelliSense提供的启动配置。

- `variables` 引入了替换变量并将它们绑定到由调试器扩展实现的命令。
- `languages` 那些调试扩展可能被认为是“默认调试器”的语言。
- `adapterExecutableCommand` 命令可以动态计算调试适配器可执行文件路径和参数的命令 ID。该命令返回具有以下格式的结构：

```
command: "<executable>",
args: [ "<argument1>", "<argument2>", ... ]
```

The attribute `command` must be either an absolute path to an executable or a name of executable looked up via the PATH environment variable. The special value `node` will be mapped to VS Code's built-in node runtime without being looked up on the PATH.

属性 `command` 必须设置为可执行文件的绝对路径或通过PATH环境变量查找的可执行文件的名称。特殊值 `node` 将被映射到VS Code的内置节点运行时间，而不是PATH上查找。

Example

```

"contributes": {
  "debuggers": [
    {
      "type": "node",
      "label": "Node Debug",

      "program": "./out/node/nodeDebug.js",
      "runtime": "node",

      "languages": ["javascript", "typescript", "javascriptreact", "typescriptreact"
    ],
    "configurationAttributes": {
      "launch": {
        "required": [ "program" ],
        "properties": {
          "program": {
            "type": "string",
            "description": "The program to debug."
          }
        }
      }
    },
    "initialConfigurations": [
      {
        "type": "node",
        "request": "launch",
        "name": "Launch Program",
        "program": "${workspaceFolder}/app.js"
      }
    ],
    "configurationSnippets": [
      {
        "label": "Node.js: Attach Configuration",
        "description": "A new configuration for attaching to a running node pr
ogram.",
        "body": {
          "type": "node",
          "request": "attach",
          "name": "${2:Attach to Port}",
          "port": 9229
        }
      }
    ],
    "variables": {
      "PickProcess": "extension.node-debug.pickNodeProcess"
    }
  ]
}

```

For a full walkthrough on how to integrate a `debugger`, go to [Debuggers](#).

有关如何集成 `debugger` 的完整演示，请转至 [调试器](#).

contributes.breakpoints

Usually a debugger extension will also have a `contributes.breakpoints` entry where the extension lists the language file types for which setting breakpoints will be enabled.

通常情况下，调试器扩展也会有一个 `contributions.breakpoints` 条目，其中扩展名列出了将启用设置断点的语言文件类型。

```
"contributes": {  
  "breakpoints": [  
    {  
      "language": "javascript"  
    },  
    {  
      "language": "javascriptreact"  
    }  
  ]  
}
```

contributes.grammars

Contribute a TextMate grammar to a language. You must provide the `language` this grammar applies to, the TextMate `scopeName` for the grammar and the file path.

设置一种语言的 TextMate 语法。您必须提供此语法适用的 `language` 字段，语法和文件路径的 TextMate `scopeName`。

Note: The file containing the grammar can be in JSON (filenames ending in `.json`) or in XML plist format (all other files).

注意：包含语法的文件可以是JSON（以`.json`结尾的文件名）或XML plist格式（所有其他文件）。

例子

```

"contributes": {
  "grammars": [{
    "language": "shellscript",
    "scopeName": "source.shell",
    "path": "./syntaxes/Shell-Unix-Bash.tmLanguage"
  }]
}

```

See [Adding Language Colorization](#) for instructions on using the [yo code extension generator](#) to quickly package TextMate .tmLanguage files as VS Code extensions.

有关使用 [yo code extension generator](#) 将 TextMate .tmLanguage 文件快速打包为VS代码扩展的说明，请参阅 [Adding Language Colorization](#)。

```

code.sh C:\Alex\src\Monaco\tools
1 #!/bin/bash
2
3 if [[ $OSTYPE == "darwin"* ]]; then
4   realpath() { [[ $1 = /* ]] && echo "$1" || echo "${PWD}/${1#.}"; }
5   ROOT=$(dirname $(dirname $(realpath "$0")))
6 else
7   ROOT=$(dirname $(dirname $(readlink -f $0)))
8 fi
9
10 # Configuration
11 export NODE_ENV=development
12 export VS CODE _DEV=1
13
14 # Prepare
15 cd $ROOT ; node node_modules/gulp/bin/gulp.js electron
16
17 if [[ $OSTYPE == "darwin"* ]]; then
18   cd $ROOT; ../Electron-Build/Electron.app/Contents/MacOS/Electron . $*
19 else
20   cd $ROOT; ../Electron-Build/electron . $*
21 fi
22

```

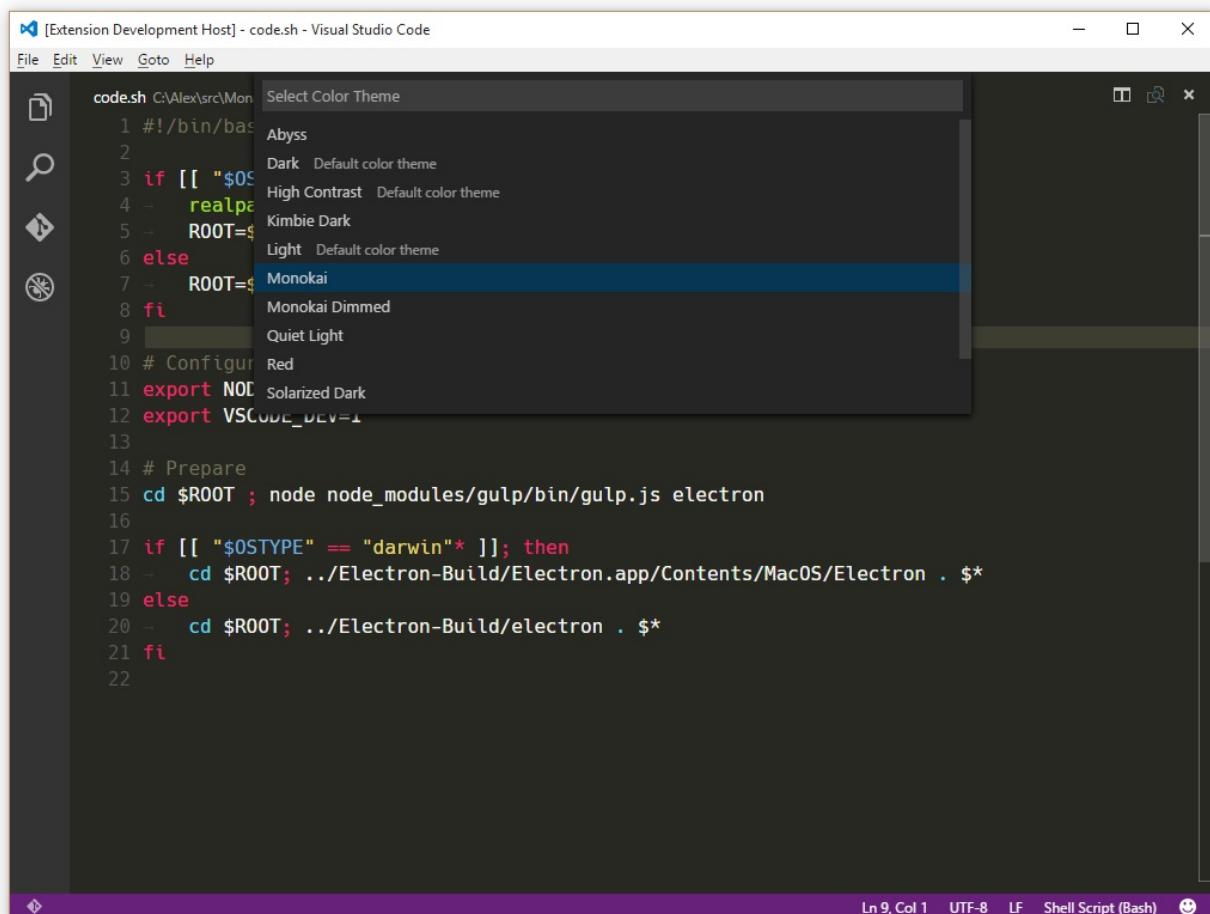
contributes.themes

Contribute a TextMate theme to VS Code. You must specify a label, whether the theme is a dark theme or a light theme (such that the rest of VS Code changes to match your theme) and the path to the file (XML plist format).

为 VS Code 提供一个 TextMate 主题。你必须指定一个标签，无论主题是黑暗的主题还是轻的主题（使VS码的其余部分更改为与主题相匹配）以及文件路径（XML plist格式）

例子

```
"contributes": {
  "themes": [
    {
      "label": "Monokai",
      "uiTheme": "vs-dark",
      "path": "./themes/Monokai.tmTheme"
    }
}
```



See [Changing the Color Theme](#) for instructions on using the [yo code extension generator](#) to quickly package TextMate .tmTheme files as VS Code extensions.

有关使用 [yo code extension generator](#) 将 TextMate .tmLanguage 文件快速打包为VS代码扩展的说明，请参阅 [Changing the Color Theme](#)。

contributes.snippets

Contribute snippets for a specific language. The `language` attribute is the language identifier and the `path` is the relative path to the snippet file, which defines snippets in the [VS Code snippet format](#).

为某一个具体的语言提供代码片段。其中，`language` 属性是 [语言标识符](#)，`path` 属性是代码片段文件的相对路径。代码文件会在 [VS Code snippet format](#) 中定义。

下面的例子显示为 Go 语言添加片段。

```
"contributes": {
  "snippets": [
    {
      "language": "go",
      "path": "./snippets/go.json"
    }
  ]
}
```

contributes.jsonValidation

Contribute a validation schema for a specific type of `json` file. The `url` value can be either a local path to a schema file included in the extension or a remote server URL such as a [json schema store](#).

为特定类型的 json 文件贡献一个验证模式。`url` 值可以是包含在扩展中的模式文件的本地路径，也可以是远程服务器 URL（如 json 模式存储）。

```
"contributes": {
  "jsonValidation": [
    {
      "fileMatch": ".jshintrc",
      "url": "http://json.schemastore.org/jshintrc"
    }
  ]
}
```

contributes.views

Contribute a view to VS Code. You must specify an identifier and name for the view. You can contribute to following locations:

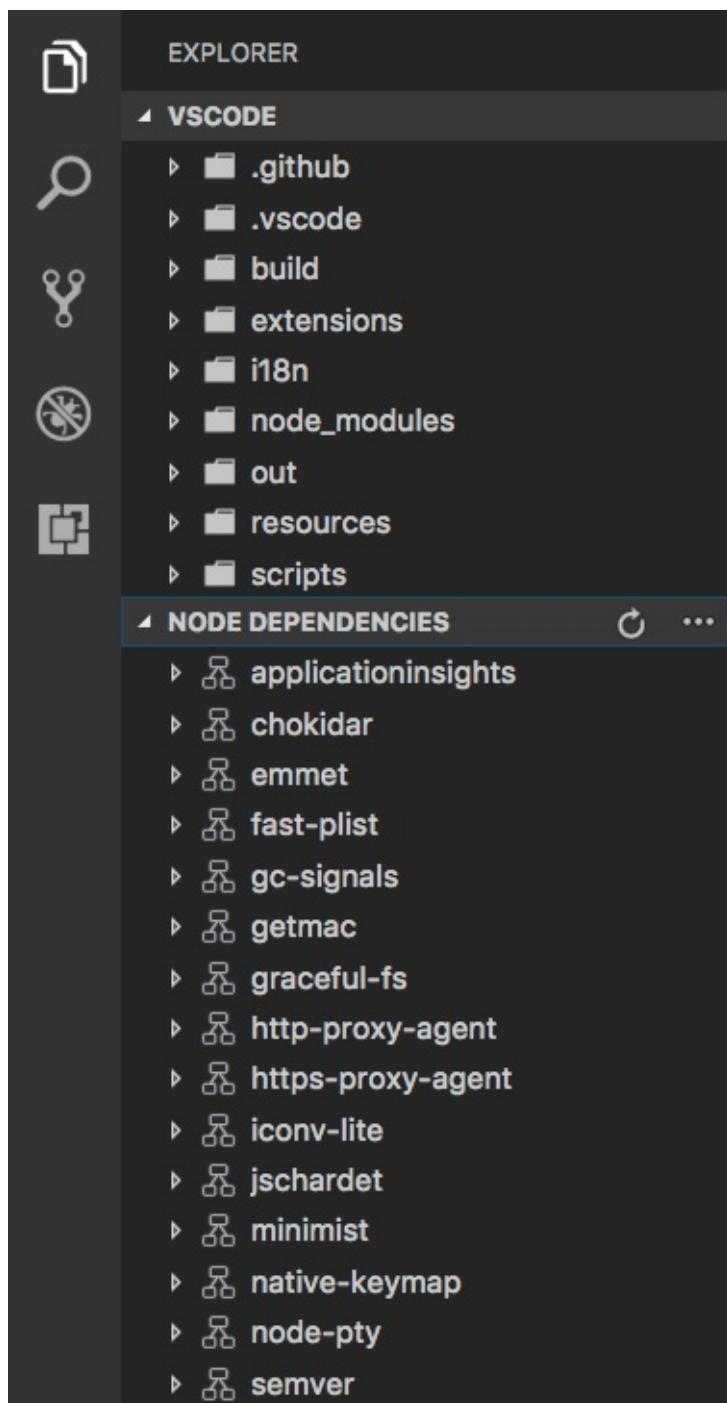
给 VS Code 配置一个 `view`。你必须为此 `view` 指定标示符与名称。你可以将 `view` 设置到以下位置

- `explorer`：侧边栏中的资源管理器
- `debug`：侧边栏中的 debug 菜单

When the user opens the view, VS Code will then emit an activationEvent `onView:${viewId}` (e.g. `onView:nodeDependencies` for the example below). You can also control the visibility of the view by providing the `when` context value.

当一个用户打开 view 的时候，VS Code 将触发一个 activationEvent `onView:${viewId}` (比如下面的例子触发了 `onView:nodeDependencies`)。您可以通过 `when` 字段根据环境来设置 view 是否显示。

```
"contributes": {  
  "views": {  
    "explorer": [  
      {  
        "id": "nodeDependencies",  
        "name": "Node Dependencies",  
        "when": "workspaceHasPackageJSON"  
      }  
    ]  
  }  
}
```



Extension writers should register a [provider](#) programmatically to populate data in the view.
Refer to examples [here](#).

拓展编写者需要注册一个 [provider](#) 程序来把数据填充到 view 中。参考此例 [here](#)。

contributes.problemMatchers

Contribute problem matcher patterns. These contributions work in both the output panel runner and in the terminal runner. Below is an example to contribute a problem matcher for the gcc compiler in an extension:

设置一个问题匹配器。这些设置在输出面板运行者和终端运行者中都可以工作。下面是一个在扩展中为gcc编译器提供一个问题匹配器的例子：

```
"contributes": {
  "problemMatchers": [
    {
      "name": "gcc",
      "owner": "cpp",
      "fileLocation": ["relative", "${workspaceFolder}"],
      "pattern": {
        "regexp": "^(.*):(\\d+):(\\d+):\\s+(warning|error):\\s+(.*$)",
        "file": 1,
        "line": 2,
        "column": 3,
        "severity": 4,
        "message": 5
      }
    }
  ]
}
```

This problem matcher can now be used in a `tasks.json` file via a name reference `$gcc`. An example looks like this:

这个问题匹配器现在可以通过类似 `$ gcc` 这样的名称引用的方式在 `tasks.json` 文件中使用。一个例子大概是：

```
{
  "version": "0.1.0",
  "command": "gcc",
  "args": ["-Wall", "helloWorld.c", "-o", "helloWorld"],
  "problemMatcher": "$gcc"
}
```

Also see: [Defining a Problem Matcher](#)

同时也可以参见: [Defining a Problem Matcher](#)

contributes.problemPatterns

Contributes named problem patterns that can be used in problem matchers (see above).

Contributes 命名的问题匹配器可以用于错误匹配参见(上文)。

Next Steps - 下一步

To learn more about VS Code extensibility model, try these topic:

要了解更多关于VS Code 的可扩展性模型，请尝试以下主题：

- [Extension Manifest File](#) - VS Code package.json extension manifest file reference
- [Activation Events](#) - VS Code activation events reference

Activation Events - package.json

Extensions are activated lazily in VS Code. As a result you need to provide VS Code with context as to when your extension should be activated. We support the following activation events:

- `onLanguage:${language}`
- `onCommand:${command}`
- `workspaceContains:${toplevelfilename}`
- *

We also provide an overview of the [package.json](#) extension manifest and the minimum required fields.

activationEvents.onLanguage

This activation event is emitted and interested extensions will be activated whenever a file that resolves to a certain language gets opened.

```
...
"activationEvents": [
    "onLanguage:python"
]
...
```

activationEvents.onCommand

This activation event is emitted and interested extensions will be activated whenever a command is being invoked:

```
...
"activationEvents": [
    "onCommand:extension.sayHello"
]
...
```

activationEvents.workspaceContains

This activation event is emitted and interested extensions will be activated whenever a folder is opened and the folder contains a top-level file.

```
...
"activationEvents": [
    "workspaceContains:.editorconfig"
]
...
```

activationEvents.*

This activation event is emitted and interested extensions will be activated whenever VS Code starts up. To ensure a great end user experience, please use this activation event in your extension only when no other activation events combination works in your use-case.

```
...
"activationEvents": [
    "*"
]
...
```

Note: An extension can listen to multiple activation events, and that is preferable to listening to `"*"`.

Note: An extension **must** export an `activate()` function from its main module and it will be invoked **only once** by VS Code when any of the specified activation events is emitted. Also, an extension **should** export a `deactivate()` function from its main module to perform cleanup tasks on VS Code shutdown.

Next Steps

To learn more about VS Code extensibility model, try these topics:

- [Extension Manifest File](#) - VS Code package.json extension manifest file reference
- [Contribution Points](#) - VS Code contribution points reference

Common Questions

Nothing yet

vscode namespace API

commands

Namespace for dealing with commands. In short, a command is a function with a unique identifier. The function is sometimes also called *command handler*.

Commands can be added to the editor using the `registerCommand` and `registerTextEditorCommand` functions. Commands can be executed `manually` or from a UI gesture. Those are:

- palette - Use the `commands`-section in `package.json` to make a command show in the [command palette](#).
- keybinding - Use the `keybindings`-section in `package.json` to enable [keybindings](#) for your extension.

Commands from other extensions and from the editor itself are accessible to an extension. However, when invoking an editor command not all argument types are supported.

This is a sample that registers a command handler and adds an entry for that command to the palette. First register a command handler with the identifier `extension.sayHello`.

```
commands.registerCommand('extension.sayHello', () => {
    window.showInformationMessage('Hello World!');
});
```

Second, bind the command identifier to a title under which it will show in the palette (`package.json`).

```
{
  "contributes": {
    "commands": [
      {
        "command": "extension.sayHello",
        "title": "Hello World"
      }
    ]
}
```

Functions

`executeCommand<T>(command: string, ...rest: any[]): Thenable<T>`

Executes the command denoted by the given command identifier.

When executing an editor command not all types are allowed to be passed as arguments. Allowed are the primitive types `string`, `boolean`, `number`, `undefined`, and `null`, as well as classes defined in this API. There are no restrictions when executing commands that have been contributed by extensions.

| Parameter | Description |
|--------------------------------|--|
| <code>command: string</code> | Identifier of the command to execute. |
| <code>...rest: any[]</code> | Parameters passed to the command function. |
| Returns | Description |
| <code>Thenable<T></code> | A thenable that resolves to the returned value of the given command. <code>undefined</code> when the command handler function doesn't return anything. |

[`getCommands\(filterInternal?: boolean\): Thenable<string\[\]>`](#)

Retrieve the list of all available commands. Commands starting an underscore are treated as internal commands.

| Parameter | Description |
|---------------------------------------|--|
| <code>filterInternal?: boolean</code> | Set <code>true</code> to not see internal commands (starting with an underscore) |
| Returns | Description |
| <code>Thenable<string[]></code> | Thenable that resolves to a list of command ids. |

[`registerCommand\(command: string, callback: \(args: any\[\]\) => any, thisArg?: any\): Disposable`](#)

Registers a command that can be invoked via a keyboard shortcut, a menu item, an action, or directly.

Registering a command with an existing command identifier twice will cause an error.

| Parameter | Description |
|---|--|
| command: string | A unique identifier for the command. |
| callback: (args: any []) => any | A command handler function. |
| thisArg?: any | The <code>this</code> context used when invoking the handler function. |
| Returns | Description |
| Disposable | Disposable which unregisters this command on disposal. |

`registerTextEditorCommand(command: string, callback: (textEditor: TextEditor, edit: TextEditorEdit) => void, thisArg?: any): Disposable`

Registers a text editor command that can be invoked via a keyboard shortcut, a menu item, an action, or directly.

Text editor commands are different from ordinary [commands](#) as they only execute when there is an active editor when the command is called. Also, the command handler of an editor command has access to the active editor and to an [edit-builder](#).

| Parameter | Description |
|--|---|
| command: string | A unique identifier for the command. |
| callback: (textEditor: TextEditor , edit: TextEditorEdit) => void | A command handler function with access to an editor and an edit . |
| thisArg?: any | The <code>this</code> context used when invoking the handler function. |
| Returns | Description |
| Disposable | Disposable which unregisters this command on disposal. |

env

Namespace describing the environment the editor runs in.

Variables

language: string

Represents the preferred user-language, like `de-CH`, `fr`, or `en-US`.

- *readonly*

machineId: string

A unique identifier for the computer.

- *readonly*

sessionId: string

A unique identifier for the current session. Changes each time the editor is started.

- *readonly*

extensions

Namespace for dealing with installed extensions. Extensions are represented by an [extension](#)-interface which allows to reflect on them.

Extension writers can provide APIs to other extensions by returning their API public surface from the `activate` -call.

```
export function activate(context: vscode.ExtensionContext) {
    let api = {
        sum(a, b) {
            return a + b;
        },
        mul(a, b) {
            return a * b;
        }
    };
    // 'export' public api-surface
    return api;
}
```

When depending on the API of another extension add an `extensionDependency` -entry to `package.json`, and use the [getExtension](#)-function and the `exports`-property, like below:

```
let mathExt = extensions.getExtension('genius.math');
let importedApi = mathExt.exports;

console.log(importedApi.mul(42, 1));
```

Variables

[all: Extension<any>\[\]](#)

All extensions currently known to the system.

Functions

[getExtension\(extensionId: string\): Extension<any>](#)

Get an extension by its full identifier in the form of: `publisher.name`.

| Parameter | Description |
|-----------------------------------|--|
| extensionId: <code>string</code> | An extension identifier. |
| Returns | Description |
| <code>Extension<any></code> | An extension or <code>undefined</code> . |

[getExtension<T>\(extensionId: string\): Extension<T>](#)

Get an extension its full identifier in the form of: `publisher.name`.

| Parameter | Description |
|----------------------------------|--|
| extensionId: <code>string</code> | An extension identifier. |
| Returns | Description |
| <code>Extension<T></code> | An extension or <code>undefined</code> . |

languages

Namespace for participating in language-specific editor [features](#), like IntelliSense, code actions, diagnostics etc.

Many programming languages exist and there is huge variety in syntaxes, semantics, and paradigms. Despite that, features like automatic word-completion, code navigation, or code checking have become popular across different tools for different programming languages.

The editor provides an API that makes it simple to provide such common features by having all UI and actions already in place and by allowing you to participate by providing data only. For instance, to contribute a hover all you have to do is provide a function that can be called

with a [TextDocument](#) and a [Position](#) returning hover info. The rest, like tracking the mouse, positioning the hover, keeping the hover stable etc. is taken care of by the editor.

```
languages.registerHoverProvider('javascript', {
    provideHover(document, position, token) {
        return new Hover('I am a hover!');
    }
});
```

Registration is done using a [document selector](#) which is either a language id, like `javascript` or a more complex [filter](#) like `{ language: 'typescript', scheme: 'file' }`. Matching a document against such a selector will result in a [score](#) that is used to determine if and how a provider shall be used. When scores are equal the provider that came last wins. For features that allow full arity, like [hover](#), the score is only checked to be `>0`, for other features, like [IntelliSense](#) the score is used for determining the order in which providers are asked to participate.

Functions

[`createDiagnosticCollection\(name?: string\): DiagnosticCollection`](#)

Create a diagnostics collection.

| Parameter | Description |
|---|---|
| <code>name?: string</code> | The name of the collection. |
| Returns | Description |
| <code>DiagnosticCollection</code> | A new diagnostic collection. |

[`getLanguages\(\): Thenable<string\[\]>`](#)

Return the identifiers of all known languages.

| Returns | Description |
|---|--|
| <code>Thenable<string[]></code> | Promise resolving to an array of identifier strings. |

[`match\(selector: DocumentSelector, document: TextDocument\): number`](#)

Compute the match between a document [selector](#) and a document. Values greater than zero mean the selector matches the document. The more individual matches a selector and a document have, the higher the score is. These are the abstract rules given a [selector](#):

```

(1) When selector is an array, return the maximum individual result.
(2) When selector is a string match that against the [languageId](#TextDocument.languageId).
    (2.1) When both are equal score is `10`,
    (2.2) When the selector is `*` score is `5`,
    (2.3) Else score is `0`.
(3) When selector is a [filter](#DocumentFilter) every property must score higher `0`.
    Iff the score is the sum of the following:
        (3.1) When [language](#DocumentFilter.language) is set apply rules from #2, when `0` the total score is `0`.
        (3.2) When [scheme](#Document.scheme) is set and equals the [uri](#TextDocument.uri)-scheme add `10` to the score, else the total score is `0`.
        (3.3) When [pattern](#Document.pattern) is set
            (3.3.1) pattern equals the [uri](#TextDocument.uri)-fsPath add `10` to the score,
            (3.3.1) if the pattern matches as glob-pattern add `5` to the score,
            (3.3.1) the total score is `0`

```

| Parameter | Description |
|---|--|
| selector:
DocumentSelector | A document selector. |
| document:
TextDocument | A text document. |
| Returns | Description |
| number | A number <code>>0</code> when the selector matches and <code>0</code> when the selector does not match. |

[`registerCodeActionsProvider\(selector: DocumentSelector, provider: CodeActionProvider\): Disposable`](#)

Register a code action provider.

Multiple providers can be registered for a language. In that case providers are asked in parallel and the results are merged. A failing provider (rejected promise or exception) will not cause a failure of the whole operation.

| Parameter | Description |
|---|--|
| selector:
DocumentSelector | A selector that defines the documents this provider is applicable to. |
| provider:
CodeActionProvider | A code action provider. |
| Returns | Description |
| Disposable | A disposable that unregisters this provider when being disposed. |

[registerCodeLensProvider\(selector: DocumentSelector, provider: CodeLensProvider\): Disposable](#)

Register a code lens provider.

Multiple providers can be registered for a language. In that case providers are asked in parallel and the results are merged. A failing provider (rejected promise or exception) will not cause a failure of the whole operation.

| Parameter | Description |
|---|--|
| selector:
DocumentSelector | A selector that defines the documents this provider is applicable to. |
| provider:
CodeLensProvider | A code lens provider. |
| Returns | Description |
| Disposable | A disposable that unregisters this provider when being disposed. |

[registerCompletionItemProvider\(selector: DocumentSelector, provider: CompletionItemProvider, ...triggerCharacters: string\[\]\): Disposable](#)

Register a completion provider.

Multiple providers can be registered for a language. In that case providers are sorted by their [score](#) and groups of equal score are sequentially asked for completion items. The process stops when one or many providers of a group return a result. A failing provider (rejected promise or exception) will not fail the whole operation.

| Parameter | Description |
|---|---|
| selector:
DocumentSelector | A selector that defines the documents this provider is applicable to. |
| provider:
CompletionItemProvider | A completion provider. |
| ...triggerCharacters:
string[] | Trigger completion when the user types one of the characters, like <code>.</code> or <code>:</code> . |
| Returns | Description |
| Disposable | A disposable that unregisters this provider when being disposed. |

[`registerDefinitionProvider\(selector: DocumentSelector, provider: DefinitionProvider\): Disposable`](#)

Register a definition provider.

Multiple providers can be registered for a language. In that case providers are asked in parallel and the results are merged. A failing provider (rejected promise or exception) will not cause a failure of the whole operation.

| Parameter | Description |
|---|--|
| selector:
DocumentSelector | A selector that defines the documents this provider is applicable to. |
| provider:
DefinitionProvider | A definition provider. |
| Returns | Description |
| Disposable | A disposable that unregisters this provider when being disposed. |

[`registerDocumentFormattingEditProvider\(selector: DocumentSelector, provider: DocumentFormattingEditProvider\): Disposable`](#)

Register a formatting provider for a document.

Multiple providers can be registered for a language. In that case providers are sorted by their [score](#) and the result of best-matching provider is used. Failure of the selected provider will cause a failure of the whole operation.

| Parameter | Description |
|--|--|
| selector: DocumentSelector | A selector that defines the documents this provider is applicable to. |
| provider: DocumentFormattingEditProvider | A document formatting edit provider. |
| Returns | Description |
| Disposable | A disposable that unregisters this provider when being disposed. |

[registerDocumentHighlightProvider\(selector: DocumentSelector, provider: DocumentHighlightProvider\): Disposable](#)

Register a document highlight provider.

Multiple providers can be registered for a language. In that case providers are sorted by their [score](#) and groups sequentially asked for document highlights. The process stops when a provider returns a [non-falsy](#) or [non-failure](#) result.

| Parameter | Description |
|---|--|
| selector: DocumentSelector | A selector that defines the documents this provider is applicable to. |
| provider: DocumentHighlightProvider | A document highlight provider. |
| Returns | Description |
| Disposable | A disposable that unregisters this provider when being disposed. |

[registerDocumentRangeFormattingEditProvider\(selector: DocumentSelector, provider: DocumentRangeFormattingEditProvider\): Disposable](#)

Register a formatting provider for a document range.

Multiple providers can be registered for a language. In that case providers are sorted by their [score](#) and the result of best-matching provider is used. Failure of the selected provider will cause a failure of the whole operation.

| Parameter | Description |
|---|--|
| selector: DocumentSelector | A selector that defines the documents this provider is applicable to. |
| provider: DocumentRangeFormattingEditProvider | A document range formatting edit provider. |
| Returns | Description |
| Disposable | A disposable that unregisters this provider when being disposed. |

[registerDocumentSymbolProvider\(selector: DocumentSelector, provider: DocumentSymbolProvider\): Disposable](#)

Register a document symbol provider.

Multiple providers can be registered for a language. In that case providers are asked in parallel and the results are merged. A failing provider (rejected promise or exception) will not cause a failure of the whole operation.

| Parameter | Description |
|--|--|
| selector: DocumentSelector | A selector that defines the documents this provider is applicable to. |
| provider: DocumentSymbolProvider | A document symbol provider. |
| Returns | Description |
| Disposable | A disposable that unregisters this provider when being disposed. |

[registerHoverProvider\(selector: DocumentSelector, provider: HoverProvider\): Disposable](#)

Register a hover provider.

Multiple providers can be registered for a language. In that case providers are asked in parallel and the results are merged. A failing provider (rejected promise or exception) will not cause a failure of the whole operation.

| Parameter | Description |
|--|--|
| selector: DocumentSelector | A selector that defines the documents this provider is applicable to. |
| provider: HoverProvider | A hover provider. |
| Returns | Description |
| Disposable | A disposable that unregisters this provider when being disposed. |

`registerOnTypeFormattingEditProvider(selector: DocumentSelector, provider: OnTypeFormattingEditProvider, firstTriggerCharacter: string, ...moreTriggerCharacter: string[]): Disposable`

Register a formatting provider that works on type. The provider is active when the user enables the setting `editor.formatOnType`.

Multiple providers can be registered for a language. In that case providers are sorted by their `score` and the result of best-matching provider is used. Failure of the selected provider will cause a failure of the whole operation.

| Parameter | Description |
|--|--|
| selector: DocumentSelector | A selector that defines the documents this provider is applicable to. |
| provider: OnTypeFormattingEditProvider | An on type formatting edit provider. |
| firstTriggerCharacter: string | A character on which formatting should be triggered, like <code>}</code> . |
| ...moreTriggerCharacter: string[] | More trigger characters. |
| Returns | Description |
| Disposable | A disposable that unregisters this provider when being disposed. |

`registerReferenceProvider(selector: DocumentSelector, provider: ReferenceProvider): Disposable`

Register a reference provider.

Multiple providers can be registered for a language. In that case providers are asked in parallel and the results are merged. A failing provider (rejected promise or exception) will not cause a failure of the whole operation.

| Parameter | Description |
|--|--|
| selector:
DocumentSelector | A selector that defines the documents this provider is applicable to. |
| provider:
ReferenceProvider | A reference provider. |
| Returns | Description |
| Disposable | A disposable that unregisters this provider when being disposed. |

[registerRenameProvider\(selector: DocumentSelector, provider: RenameProvider\): Disposable](#)

Register a reference provider.

Multiple providers can be registered for a language. In that case providers are sorted by their [score](#) and the result of best-matching provider is used. Failure of the selected provider will cause a failure of the whole operation.

| Parameter | Description |
|---|--|
| selector:
DocumentSelector | A selector that defines the documents this provider is applicable to. |
| provider:
RenameProvider | A rename provider. |
| Returns | Description |
| Disposable | A disposable that unregisters this provider when being disposed. |

[registerSignatureHelpProvider\(selector: DocumentSelector, provider: SignatureHelpProvider, ...triggerCharacters: string\[\]\): Disposable](#)

Register a signature help provider.

Multiple providers can be registered for a language. In that case providers are sorted by their [score](#) and the result of best-matching provider is used. Failure of the selected provider will cause a failure of the whole operation.

| Parameter | Description |
|---|--|
| selector: DocumentSelector | A selector that defines the documents this provider is applicable to. |
| provider: SignatureHelpProvider | A signature help provider. |
| ...triggerCharacters: string[] | Trigger signature help when the user types one of the characters, like , or (. |
| Returns | Description |
| Disposable | A disposable that unregisters this provider when being disposed. |

[registerWorkspaceSymbolProvider\(provider: WorkspaceSymbolProvider\)](#):

[Disposable](#)

Register a workspace symbol provider.

Multiple providers can be registered for a language. In that case providers are asked in parallel and the results are merged. A failing provider (rejected promise or exception) will not cause a failure of the whole operation.

| Parameter | Description |
|---|--|
| provider: WorkspaceSymbolProvider | A workspace symbol provider. |
| Returns | Description |
| Disposable | A disposable that unregisters this provider when being disposed. |

[setLanguageConfiguration\(language: string, configuration: LanguageConfiguration\)](#):

[Disposable](#)

Set a [language configuration](#) for a language.

| Parameter | Description |
|--|--|
| language: string | A language identifier like <code>typescript</code> . |
| configuration: LanguageConfiguration | Language configuration. |
| Returns | Description |
| Disposable | A disposable that unsets this configuration. |

window

Namespace for dealing with the current window of the editor. That is visible and active editors, as well as, UI elements to show messages, selections, and asking for user input.

Variables

[activeTextEditor: TextEditor](#)

The currently active editor or undefined. The active editor is the one that currently has focus or, when none has focus, the one that has changed input most recently.

[visibleTextEditors: TextEditor\[\]](#)

The currently visible editors or an empty array.

Events

[onDidChangeActiveTextEditor: Event<TextEditor>](#)

An [event](#) which fires when the [active editor](#) has changed.

[onDidChangeTextEditorOptions: Event<TextEditorOptionsChangeEvent>](#)

An [event](#) which fires when the options of an editor have changed.

[onDidChangeTextEditorSelection: Event<TextEditorSelectionChangeEvent>](#)

An [event](#) which fires when the selection in an editor has changed.

[onDidChangeTextEditorViewColumn: Event<TextEditorViewColumnChangeEvent>](#)

An [event](#) which fires when the view column of an editor das changed.

Functions

[createOutputChannel\(name: string\): OutputChannel](#)

Create a new [output channel](#) with the given name.

| Parameter | Description |
|-------------------------------|--|
| name: string | Human-readable string which will be used to represent the channel in the UI. |
| Returns | Description |
| OutputChannel | |

[createStatusBarItem\(alignment?: StatusBarAlignment, priority?: number\): StatusBarItem](#)

Creates a status bar item.

| Parameter | Description |
|--|---|
| alignment?: StatusBarAlignment | The alignment of the item. |
| priority?: number | The priority of the item. Higher values mean the item should be shown more to the left. |
| Returns | Description |
| StatusBarItem | A new status bar item. |

[createTextEditorDecorationType\(options: DecorationRenderOptions\): TextEditorDecorationType](#)

Create a TextEditorDecorationType that can be used to add decorations to text editors.

| Parameter | Description |
|--|--|
| options: DecorationRenderOptions | Rendering options for the decoration type. |
| Returns | Description |
| TextEditorDecorationType | A new decoration type instance. |

[setStatusBarMessage\(text: string\): Disposable](#)

Set a message to the status bar. This is a short hand for the more powerful status bar items.

| Parameter | Description |
|------------------------------|--|
| text: string | The message to show, support icon substitution as in status bar items. |
| Returns | Description |
| Disposable | A disposable which hides the status bar message. |

[`setStatusBarMessage\(text: string, hideAfterTimeout: number\): Disposable`](#)

Set a message to the status bar. This is a short hand for the more powerful status bar [items](#).

| Parameter | Description |
|---|---|
| <code>text: string</code> | The message to show, support icon substitution as in status bar items . |
| <code>hideAfterTimeout: number</code> | Timeout in milliseconds after which the message will be disposed. |
| Returns | Description |
| <code>Disposable</code> | A disposable which hides the status bar message. |

[`setStatusBarMessage\(text: string, hideWhenDone: Thenable<any>\): Disposable`](#)

Set a message to the status bar. This is a short hand for the more powerful status bar [items](#).

| Parameter | Description |
|--|---|
| <code>text: string</code> | The message to show, support icon substitution as in status bar items . |
| <code>hideWhenDone: Thenable<any></code> | Thenable on which completion (resolve or reject) the message will be disposed. |
| Returns | Description |
| <code>Disposable</code> | A disposable which hides the status bar message. |

[`showErrorMessage\(message: string, ...items: string\[\]\): Thenable<string>`](#)

Show an error message.

- see - [showInformationMessage](#)

| Parameter | Description |
|---|---|
| <code>message: string</code> | The message to show. |
| <code>...items: string[]</code> | A set of items that will be rendered as actions in the message. |
| Returns | Description |
| <code>Thenable<string></code> | A thenable that resolves to the selected item or <code>undefined</code> when being dismissed. |

showErrorMessage<T extends MessageItem>(message: string, ...items: T[]): Thenable<T>

Show an error message.

- see - [showInformationMessage](#)

| Parameter | Description |
|-----------------|---|
| message: string | The message to show. |
| ...items: T[] | A set of items that will be rendered as actions in the message. |
| Returns | Description |
| Thenable<T> | A thenable that resolves to the selected item or <code>undefined</code> when being dismissed. |

showInformationMessage(message: string, ...items: string[]): Thenable<string>

Show an information message to users. Optionally provide an array of items which will be presented as clickable buttons.

| Parameter | Description |
|--------------------|---|
| message: string | The message to show. |
| ...items: string[] | A set of items that will be rendered as actions in the message. |
| Returns | Description |
| Thenable<string> | A thenable that resolves to the selected item or <code>undefined</code> when being dismissed. |

showInformationMessage<T extends MessageItem>(message: string, ...items: T[]): Thenable<T>

Show an information message.

- see - [showInformationMessage](#)

| Parameter | Description |
|-----------------|---|
| message: string | The message to show. |
| ...items: T[] | A set of items that will be rendered as actions in the message. |
| Returns | Description |
| Thenable<T> | A thenable that resolves to the selected item or <code>undefined</code> when being dismissed. |

`showInputBox(options?: InputBoxOptions): Thenable<string>`

Opens an input box to ask the user for input.

The returned value will be `undefined` if the input box was canceled (e.g. pressing ESC). Otherwise the returned value will be the string typed by the user or an empty string if the user did not type anything but dismissed the input box with OK.

| Parameter | Description |
|---------------------------|--|
| options?: InputBoxOptions | Configures the behavior of the input box. |
| Returns | Description |
| Thenable<string> | A promise that resolves to a string the user provided or to <code>undefined</code> in case of dismissal. |

`showQuickPick(items: string[] | Thenable<string[]>, options?: QuickPickOptions): Thenable<string>`

Shows a selection list.

| Parameter | Description |
|--------------------------------------|---|
| items: string[] Thenable<string[]> | An array of strings, or a promise that resolves to an array of strings. |
| options?: QuickPickOptions | Configures the behavior of the selection list. |
| Returns | Description |
| Thenable<string> | A promise that resolves to the selection or <code>undefined</code> . |

`showQuickPick<T extends QuickPickItem>(items: T[] | Thenable<T[]>, options?: QuickPickOptions): Thenable<T>`

Shows a selection list.

| Parameter | Description |
|--|---|
| items: T[] Thenable<T[]> | An array of items, or a promise that resolves to an array of items. |
| options?: QuickPickOptions | Configures the behavior of the selection list. |
| Returns | Description |
| Thenable<T> | A promise that resolves to the selected item or undefined. |

[`showTextDocument\(document: TextDocument, column?: ViewColumn, preserveFocus?: boolean\): Thenable<TextEditor>`](#)

Show the given document in a text editor. A [column](#) can be provided to control where the editor is being shown. Might change the [active editor](#).

| Parameter | Description |
|--|--|
| document: TextDocument | A text document to be shown. |
| column?: ViewColumn | A view column in which the editor should be shown. The default is the one , other values are adjusted to be Min(column, columnCount + 1) . |
| preserveFocus?: boolean | When <code>true</code> the editor will not take focus. |
| Returns | Description |
| Thenable<TextEditor> | A promise that resolves to an editor . |

[`showWarningMessage\(message: string, ...items: string\[\]\): Thenable<string>`](#)

Show a warning message.

- see - [showInformationMessage](#)

| Parameter | Description |
|--|---|
| message: string | The message to show. |
| ...items: string[] | A set of items that will be rendered as actions in the message. |
| Returns | Description |
| Thenable<string> | A thenable that resolves to the selected item or <code>undefined</code> when being dismissed. |

showWarningMessage<T extends MessageItem>(message: string, ...items: T[]): Thenable<T>

Show a warning message.

- see - [showInformationMessage](#)

| Parameter | Description |
|-----------------|---|
| message: string | The message to show. |
| ...items: T[] | A set of items that will be rendered as actions in the message. |
| Returns | Description |
| Thenable<T> | A thenable that resolves to the selected item or <code>undefined</code> when being dismissed. |

workspace

Namespace for dealing with the current workspace. A workspace is the representation of the folder that has been opened. There is no workspace when just a file but not a folder has been opened.

The workspace offers support for [listening](#) to fs events and for [finding](#) files. Both perform well and run *outside* the editor-process so that they should be always used instead of nodejs-equivalents.

Variables

rootPath: string

The folder that is open in VS Code. `undefined` when no folder has been opened.

- *readonly*

textDocuments: TextDocument[]

All text documents currently known to the system.

- *readonly*

Events

onDidChangeConfiguration: Event<void>

An event that is emitted when the [configuration](#) changed.

[`onDidChangeTextDocument: Event<TextDocumentChangeEvent>`](#)

An event that is emitted when a [text document](#) is changed.

[`onDidCloseTextDocument: Event<TextDocument>`](#)

An event that is emitted when a [text document](#) is disposed.

[`onDidOpenTextDocument: Event<TextDocument>`](#)

An event that is emitted when a [text document](#) is opened.

[`onDidSaveTextDocument: Event<TextDocument>`](#)

An event that is emitted when a [text document](#) is saved to disk.

Functions

[`applyEdit\(edit: WorkspaceEdit\): Thenable<boolean>`](#)

Make changes to one or many resources as defined by the given [workspace edit](#).

When applying a workspace edit, the editor implements an 'all-or-nothing'-strategy, that means failure to load one document or make changes to one document will cause the edit to be rejected.

| Parameter | Description |
|--------------------------------------|--|
| <code>edit: WorkspaceEdit</code> | A workspace edit. |
| Returns | Description |
| <code>Thenable<boolean></code> | A thenable that resolves when the edit could be applied. |

[`asRelativePath\(pathOrUri: string | Uri\): string`](#)

Returns a path that is relative to the workspace root.

When there is no [workspace root](#) or when the path is not a child of that folder, the input is returned.

| Parameter | Description |
|---|--|
| pathOrUri: string Uri | A path or uri. When a uri is given its fsPath is used. |
| Returns | Description |
| string | A path relative to the root or the input. |

[createFileSystemWatcher](#)(globPattern: [string](#), ignoreCreateEvents?: [boolean](#), ignoreChangeEvents?: [boolean](#), ignoreDeleteEvents?: [boolean](#)): [FileSystemWatcher](#)

Creates a file system watcher.

A glob pattern that filters the file events must be provided. Optionally, flags to ignore certain kinds of events can be provided. To stop listening to events the watcher must be disposed.

| Parameter | Description |
|--|---|
| globPattern: string | A glob pattern that is applied to the names of created, changed, and deleted files. |
| ignoreCreateEvents?: boolean | Ignore when files have been created. |
| ignoreChangeEvents?: boolean | Ignore when files have been changed. |
| ignoreDeleteEvents?: boolean | Ignore when files have been deleted. |
| Returns | Description |
| FileSystemWatcher | A new file system watcher instance. |

[findFiles](#)(include: [string](#), exclude: [string](#), maxResults?: [number](#), token?: [CancellationToken](#)): [Thenable<Uri\[\]>](#)

Find files in the workspace.

- **sample** - `findFiles('**/*.js', '**/node_modules/**', 10)`

| Parameter | Description |
|--|--|
| include: <code>string</code> | A glob pattern that defines the files to search for. |
| exclude: <code>string</code> | A glob pattern that defines files and folders to exclude. |
| maxResults?: <code>number</code> | An upper-bound for the result. |
| token?: <code>CancellationToken</code> | A token that can be used to signal cancellation to the underlying search engine. |
| Returns | Description |
| <code>Thenable<Uri[]></code> | A thenable that resolves to an array of resource identifiers. |

[getConfiguration\(section?: string\): WorkspaceConfiguration](#)

Get a configuration object.

When a section-identifier is provided only that part of the configuration is returned. Dots in the section-identifier are interpreted as child-access, like `{ myExt: { setting: { doIt: true }}}` and `getConfiguration('myExt.setting.doIt') === true`.

| Parameter | Description |
|-------------------------------------|---|
| section?: <code>string</code> | A dot-separated identifier. |
| Returns | Description |
| <code>WorkspaceConfiguration</code> | The full workspace configuration or a subset. |

[openTextDocument\(uri: Uri\): Thenable<TextDocument>](#)

Opens the denoted document from disk. Will return early if the document is already open, otherwise the document is loaded and the [open document](#)-event fires. The document to open is denoted by the `uri`. Two schemes are supported:

`file`: A file on disk, will be rejected if the file does not exist or cannot be loaded, e.g.

`'file:///Users/frodo/r.ini'`.

`untitled`: A new file that should be saved on disk, e.g.

`'untitled:/Users/frodo/new.js'`. The language will be derived from the file name.

Uris with other schemes will make this method return a rejected promise.

| Parameter | Description |
|------------------------|--|
| uri: Uri | Identifies the resource to open. |
| Returns | Description |
| Thenable<TextDocument> | A promise that resolves to a document. |

openTextDocument(fileName: string): Thenable<TextDocument>

A short-hand for `openTextDocument(Uri.file(fileName))`.

- see - [openTextDocument](#)

| Parameter | Description |
|------------------------|--|
| fileName: string | A name of a file on disk. |
| Returns | Description |
| Thenable<TextDocument> | A promise that resolves to a document. |

registerTextDocumentContentProvider(scheme: string, provider: TextDocumentContentProvider): Disposable

Register a text document content provider.

Only one provider can be registered per scheme.

| Parameter | Description |
|---------------------------------------|--|
| scheme: string | The uri-scheme to register for. |
| provider: TextDocumentContentProvider | A content provider. |
| Returns | Description |
| Disposable | A disposable that unregisters this provider when being disposed. |

saveAll(includeUntitled?: boolean): Thenable<boolean>

Save all dirty files.

| Parameter | Description |
|---------------------------|---|
| includeUntitled?: boolean | Also save files that have been created during this session. |
| Returns | Description |
| Thenable<boolean> | A thenable that resolves when the files have been saved. |

CancellationToken

A cancellation token is passed to an asynchronous or long running operation to request cancellation, like cancelling a request for completion items because the user continued to type.

To get an instance of a `CancellationToken` use a [CancellationTokenSource](#).

Properties

[`isCancellationRequested: boolean`](#)

Is `true` when the token has been cancelled, `false` otherwise.

[`onCancellationRequested: Event<any>`](#)

An [event](#) which fires upon cancellation.

CancellationTokenSource

A cancellation source creates and controls a [cancellation token](#).

Properties

[`token: CancellationToken`](#)

The cancellation token of this source.

Methods

[`cancel\(\): void`](#)

Signal cancellation on the token.

| Returns | Description |
|-------------------|-------------|
| <code>void</code> | |

[dispose\(\): void](#)

Dispose object and free resources. Will call [cancel](#).

| Returns | Description |
|----------------------|-------------|
| void | |

CharacterPair

A tuple of two characters, like a pair of opening and closing brackets.

[CharacterPair: \[string, string\]](#)

CodeActionContext

Contains additional diagnostic information about the context in which a [code action](#) is run.

Properties

[diagnostics: Diagnostic\[\]](#)

An array of diagnostics.

- *readonly*

CodeActionProvider

The code action interface defines the contract between extensions and the [light bulb](#) feature.

A code action can be any command that is [known](#) to the system.

Methods

[provideCodeActions\(document: TextDocument, range: Range, context: CodeActionContext, token: CancellationToken\): Command\[\] | Thenable<Command\[\]>](#)

Provide commands for the given document and range.

| Parameter | Description |
|---|---|
| document:
TextDocument | The document in which the command was invoked. |
| range: Range | The range for which the command was invoked. |
| context:
CodeActionContext | Context carrying additional information. |
| token:
CancellationToken | A cancellation token. |
| Returns | Description |
| Command[] Thenable<Command[]> | An array of commands or a thenable of such. The lack of a result can be signaled by returning <code>undefined</code> , <code>null</code> , or an empty array. |

CodeLens

A code lens represents a [command](#) that should be shown along with source text, like the number of references, a way to run tests, etc.

A code lens is *unresolved* when no command is associated to it. For performance reasons the creation of a code lens and resolving should be done to two stages.

- see - [CodeLensProvider.provideCodeLenses](#)
- see - [CodeLensProvider.resolveCodeLens](#)

Constructors

[`new CodeLens\(range: Range, command?: Command\): CodeLens`](#)

Creates a new code lens object.

| Parameter | Description |
|-----------------------------------|--|
| range: Range | The range to which this code lens applies. |
| command?: Command | The command associated to this code lens. |
| Returns | Description |
| CodeLens | |

Properties

command: Command

The command this code lens represents.

isResolved: boolean

true when there is a command associated.

range: Range

The range in which this code lens is valid. Should only span a single line.

CodeLensProvider

A code lens provider adds [commands](#) to source text. The commands will be shown as dedicated horizontal lines in between the source text.

Methods

provideCodeLenses(document: TextDocument, token: CancellationToken): CodeLens[] | Thenable<CodeLens[]>

Compute a list of [lenses](#). This call should return as fast as possible and if computing the commands is expensive implementors should only return code lens objects with the range set and implement [resolve](#).

| Parameter | Description |
|--|--|
| document:
TextDocument | The document in which the command was invoked. |
| token:
CancellationToken | A cancellation token. |
| Returns | Description |
| CodeLens[]
Thenable<CodeLens[]> | An array of code lenses or a thenable that resolves to such. The lack of a result can be signaled by returning <code>undefined</code> , <code>null</code> , or an empty array. |

resolveCodeLens(codeLens: CodeLens, token: CancellationToken): CodeLens | Thenable<CodeLens>

This function will be called for each visible code lens, usually when scrolling and after calls to [compute-lenses](#).

| Parameter | Description |
|---|--|
| codeLens: CodeLens | code lens that must be resolved. |
| token: CancellationToken | A cancellation token. |
| Returns | Description |
| CodeLens
Thenable < CodeLens > | The given, resolved code lens or thenable that resolves to such. |

Command

Represents a reference to a command. Provides a title which will be used to represent a command in the UI and, optionally, an array of arguments which will be passed to the command handler function when invoked.

Properties

[`arguments?: any\[\]`](#)

Arguments that the command handler should be invoked with.

[`command: string`](#)

The identifier of the actual command handler.

- see - [commands.registerCommand](#).

[`title: string`](#)

Title of the command, like `save`.

CommentRule

Describes how comments for a language work.

Properties

[`blockComment?: CharacterPair`](#)

The block comment character pair, like `/* block comment */`

[`lineComment?: string`](#)

The line comment token, like `// this is a comment`

CompletionItem

A completion item represents a text snippet that is proposed to complete text that is being typed.

- see - [CompletionItemProvider.provideCompletionItems](#)
- see - [CompletionItemProvider.resolveCompletionItem](#)

Constructors

[`new CompletionItem\(label: string\): CompletionItem`](#)

Creates a new completion item.

Completion items must have at least a [label](#) which then will be used as insert text as well as for sorting and filtering.

| Parameter | Description |
|--------------------------------|------------------------------|
| <code>label: string</code> | The label of the completion. |
| Returns | Description |
| CompletionItem | |

Properties

[`detail: string`](#)

A human-readable string with additional information about this item, like type or symbol information.

[`documentation: string`](#)

A human-readable string that represents a doc-comment.

[`filterText: string`](#)

A string that should be used when filtering a set of completion items. When `falsey` the [label](#) is used.

[`insertText: string`](#)

A string that should be inserted in a document when selecting this completion. When `falsey` the [label](#) is used.

[`kind: CompletionItemKind`](#)

The kind of this completion item. Based on the kind an icon is chosen by the editor.

[label](#): string

The label of this completion item. By default this is also the text that is inserted when selecting this completion.

[sortText](#): string

A string that should be used when comparing this item with other items. When `false` the [label](#) is used.

[textEdit](#): [TextEdit](#)

An [edit](#) which is applied to a document when selecting this completion. When an edit is provided the value of [insertText](#) is ignored.

The [range](#) of the edit must be single-line and one the same line completions where [requested](#) at.

CompletionItemKind

Completion item kinds.

Enumeration members

Class

Color

Constructor

Enum

Field

File

Function

Interface

Keyword

Method

Module

Property

Reference

Snippet

Text

Unit

Value

Variable

CompletionItemProvider

The completion item provider interface defines the contract between extensions and the [IntelliSense](#).

When computing *complete* completion items is expensive, providers can optionally implement the `resolveCompletionItem`-function. In that case it is enough to return completion items with a `label` from the `provideCompletionItems`-function. Subsequently, when a completion item is shown in the UI and gains focus this provider is asked to resolve the item, like adding [doc-comment](#) or [details](#).

Methods

`provideCompletionItems(document: TextDocument, position: Position, token: CancellationToken): CompletionItem[] | Thenable<CompletionItem[]> | CompletionList | Thenable<CompletionList>`

Provide completion items for the given position and document.

| Parameter | Description |
|--|---|
| <code>document: TextDocument</code> | The document in which the command was invoked. |
| <code>position: Position</code> | The position at which the command was invoked. |
| <code>token: CancellationToken</code> | A cancellation token. |
| Returns | Description |
| <code>CompletionItem[] Thenable<CompletionItem[]> CompletionList Thenable<CompletionList></code> | An array of completions, a completion list , or a thenable that resolves to either. The lack of a result can be signaled by returning <code>undefined</code> , <code>null</code> , or an empty array. |

resolveCompletionItem(item: CompletionItem, token: CancellationToken):
CompletionItem | Thenable<CompletionItem>

Given a completion item fill in more data, like [doc-comment](#) or [details](#).

The editor will only resolve a completion item once.

| Parameter | Description |
|---|---|
| item: CompletionItem | A completion item currently active in the UI. |
| token: CancellationToken | A cancellation token. |
| Returns | Description |
| CompletionItem Thenable<CompletionItem> | The resolved completion item or a thenable that resolves to of such. It is OK to return the given <code>item</code> . When no result is returned, the given <code>item</code> will be used. |

CompletionList

Represents a collection of [completion items](#) to be presented in the editor.

Constructors

new CompletionList(items?: CompletionItem[], isIncomplete?: boolean):
CompletionList

Creates a new completion list.

| Parameter | Description |
|--------------------------|---------------------------|
| items?: CompletionItem[] | The completion items. |
| isIncomplete?: boolean | The list is not complete. |
| Returns | Description |
| CompletionList | |

Properties

isIncomplete: boolean

This list it not complete. Further typing should result in recomputing this list.

items: CompletionItem[]

The completion items.

DecorationOptions

Represents options for a specific decoration in a [decoration set](#).

Properties

[`hoverMessage: MarkedString | MarkedString\[\]`](#)

A message that should be rendered when hovering over the decoration.

[`range: Range`](#)

Range to which this decoration is applied.

DecorationRenderOptions

Represents rendering styles for a [text editor decoration](#).

Properties

[`backgroundColor?: string`](#)

Background color of the decoration. Use `rgba()` and define transparent background colors to play well with other decorations.

[`borderColor?: string`](#)

CSS styling property that will be applied to text enclosed by a decoration.

[`borderRadius?: string`](#)

CSS styling property that will be applied to text enclosed by a decoration.

[`borderSpacing?: string`](#)

CSS styling property that will be applied to text enclosed by a decoration.

[`borderStyle?: string`](#)

CSS styling property that will be applied to text enclosed by a decoration.

[`borderWidth?: string`](#)

CSS styling property that will be applied to text enclosed by a decoration.

[`color?: string`](#)

CSS styling property that will be applied to text enclosed by a decoration.

[`cursor?: string`](#)

CSS styling property that will be applied to text enclosed by a decoration.

[`dark?: ThemableDecorationRenderOptions`](#)

Overwrite options for dark themes.

[`gutterIconPath?: string`](#)

An **absolute path** to an image to be rendered in the gutterIconPath.

[`isWholeLine?: boolean`](#)

Should the decoration be rendered also on the whitespace after the line text. Defaults to `false`.

[`letterSpacing?: string`](#)

CSS styling property that will be applied to text enclosed by a decoration.

[`light?: ThemableDecorationRenderOptions`](#)

Overwrite options for light themes.

[`outlineColor?: string`](#)

CSS styling property that will be applied to text enclosed by a decoration.

[`outlineStyle?: string`](#)

CSS styling property that will be applied to text enclosed by a decoration.

[`outlineWidth?: string`](#)

CSS styling property that will be applied to text enclosed by a decoration.

[`overviewRulerColor?: string`](#)

The color of the decoration in the overview ruler. Use `rgba()` and define transparent colors to play well with other decorations.

[`overviewRulerLane?: OverviewRulerLane`](#)

The position in the overview ruler where the decoration should be rendered.

[`textDecoration?: string`](#)

CSS styling property that will be applied to text enclosed by a decoration.

Definition

The definition of a symbol represented as one or many [locations](#). For most programming languages there is only one location at which a symbol is defined.

[`Definition: Location | Location\[\]`](#)

DefinitionProvider

The definition provider interface defines the contract between extensions and the [go to definition](#) and peek definition features.

Methods

[`provideDefinition\(document: TextDocument, position: Position, token: CancellationToken\): Definition | Thenable<Definition>`](#)

Provide the definition of the symbol at the given position and document.

| Parameter | Description |
|---|---|
| document: TextDocument | The document in which the command was invoked. |
| position: Position | The position at which the command was invoked. |
| token: CancellationToken | A cancellation token. |
| Returns | Description |
| Definition Thenable<Definition> | A definition or a thenable that resolves to such. The lack of a result can be signaled by returning <code>undefined</code> or <code>null</code> . |

Diagnostic

Represents a diagnostic, such as a compiler error or warning. Diagnostic objects are only valid in the scope of a file.

Constructors

[`new Diagnostic\(range: Range, message: string, severity?: DiagnosticSeverity\): Diagnostic`](#)

Creates a new diagnostic object.

| Parameter | Description |
|---|--|
| range: Range | The range to which this diagnostic applies. |
| message: string | The human-readable message. |
| severity?: DiagnosticSeverity | The severity, default is error . |
| Returns | Description |
| Diagnostic | |

Properties

[`code: string | number`](#)

A code or identifier for this diagnostics. Will not be surfaced to the user, but should be used for later processing, e.g. when providing [code actions](#).

[`message: string`](#)

The human-readable message.

[`range: Range`](#)

The range to which this diagnostic applies.

[`severity: DiagnosticSeverity`](#)

The severity, default is [error](#).

[`source: string`](#)

A human-readable string describing the source of this diagnostic, e.g. 'typescript' or 'super lint'.

DiagnosticCollection

A diagnostics collection is a container that manages a set of [diagnostics](#). Diagnostics are always scopes to a a diagnostics collection and a resource.

To get an instance of a `DiagnosticCollection` use [createDiagnosticCollection](#).

Properties

[`name: string`](#)

The name of this diagnostic collection, for instance `typescript`. Every diagnostic from this collection will be associated with this name. Also, the task framework uses this name when defining [problem matchers](#).

Methods

[`clear\(\): void`](#)

Remove all diagnostics from this collection. The same as calling `#set(undefined)` ;

| Returns | Description |
|-------------------|-------------|
| <code>void</code> | |

[`delete\(uri: Uri\): void`](#)

Remove all diagnostics from this collection that belong to the provided `uri`. The same as `#set(uri, undefined)` .

| Parameter | Description |
|-----------------------|------------------------|
| <code>uri: Uri</code> | A resource identifier. |
| Returns | Description |
| <code>void</code> | |

[`dispose\(\): void`](#)

Dispose and free associated resources. Calls [clear](#).

| Returns | Description |
|-------------------|-------------|
| <code>void</code> | |

[`set\(uri: Uri, diagnostics: Diagnostic\[\]\): void`](#)

Assign diagnostics for given resource. Will replace existing diagnostics for that resource.

| Parameter | Description |
|--|--|
| <code>uri: Uri</code> | A resource identifier. |
| <code>diagnostics: Diagnostic[]</code> | Array of diagnostics or <code>undefined</code> |
| Returns | Description |
| <code>void</code> | |

[`set\(entries: \[Uri, Diagnostic\[\]\]\): void`](#)

Replace all entries in this collection.

| Parameter | Description |
|---|--|
| entries: [Uri , Diagnostic []][] | An array of tuples, like <code>[[file1, [d1, d2]], [file2, [d3, d4, d5]]]</code> , or <code>undefined</code> . |
| Returns | Description |
| void | |

DiagnosticSeverity

Represents the severity of diagnostics.

Enumeration members

Error

0

Hint

3

Information

2

Warning

1

Disposable

Represents a type which can release resources, such as event listening or a timer.

Static

[`from\(...disposableLikes: {dispose: \(\) => any}\[\]\): Disposable`](#)

Combine many disposable-likes into one. Use this method when having objects with a dispose function which are not instances of Disposable.

| Parameter | Description |
|--|--|
| <code>...disposableLikes: {dispose: () => any}[]</code> | Objects that have at least a <code>dispose</code> -function member. |
| Returns | Description |
| <code>Disposable</code> | Returns a new disposable which, upon dispose, will dispose all provided disposables. |

Constructors

[`new Disposable\(callOnDispose: Function\): Disposable`](#)

Creates a new Disposable calling the provided function on dispose.

| Parameter | Description |
|--------------------------------------|-----------------------------------|
| <code>callOnDispose: Function</code> | Function that disposes something. |
| Returns | Description |
| <code>Disposable</code> | |

Methods

[`dispose\(\): any`](#)

Dispose this object.

| Returns | Description |
|------------------|-------------|
| <code>any</code> | |

DocumentFilter

A document filter denotes a document by different properties like the [language](#), the [scheme](#) of its resource, or a glob-pattern that is applied to the [path](#).

- `sample` - A language filter that applies to typescript files on disk: `{ language: 'typescript', scheme: 'file' }`
- `sample` - A language filter that applies to all package.json paths: `{ language: 'json', pattern: '**/project.json' }`

Properties

language?: string

A language id, like `typescript`.

pattern?: string

A glob pattern, like `*.{ts,js}`.

scheme?: string

A Uri `scheme`, like `file` or `untitled`.

DocumentFormattingEditProvider

The document formatting provider interface defines the contract between extensions and the formatting-feature.

Methods

provideDocumentFormattingEdits(document: TextDocument, options: FormattingOptions, token: CancellationToken): `TextEdit[] | Thenable<TextEdit[]>`

Provide formatting edits for a whole document.

| Parameter | Description |
|--|--|
| document: <code>TextDocument</code> | The document in which the command was invoked. |
| options: <code>FormattingOptions</code> | Options controlling formatting. |
| token: <code>CancellationToken</code> | A cancellation token. |
| Returns | Description |
| <code>TextEdit[] Thenable<TextEdit[]></code> | A set of text edits or a thenable that resolves to such. The lack of a result can be signaled by returning <code>undefined</code> , <code>null</code> , or an empty array. |

DocumentHighlight

A document highlight is a range inside a text document which deserves special attention. Usually a document highlight is visualized by changing the background color of its range.

Constructors

new DocumentHighlight(range: Range, kind?: DocumentHighlightKind): DocumentHighlight

Creates a new document highlight object.

| Parameter | Description |
|------------------------------|--------------------------------------|
| range: Range | The range the highlight applies to. |
| kind?: DocumentHighlightKind | The highlight kind, default is text. |
| Returns | Description |
| DocumentHighlight | |

Properties

kind: DocumentHighlightKind

The highlight kind, default is text.

range: Range

The range this highlight applies to.

DocumentHighlightKind

A document highlight kind.

Enumeration members

Read

Text

Write

DocumentHighlightProvider

The document highlight provider interface defines the contract between extensions and the word-highlight-feature.

Methods

provideDocumentHighlights(document: TextDocument, position: Position, token: CancellationToken): DocumentHighlight[] | Thenable<DocumentHighlight[]>

Provide a set of document highlights, like all occurrences of a variable or all exit-points of a function.

| Parameter | Description |
|---|--|
| document: TextDocument | The document in which the command was invoked. |
| position: Position | The position at which the command was invoked. |
| token: CancellationToken | A cancellation token. |
| Returns | Description |
| DocumentHighlight[] Thenable<DocumentHighlight[]> | An array of document highlights or a thenable that resolves to such. The lack of a result can be signaled by returning <code>undefined</code> , <code>null</code> , or an empty array. |

DocumentRangeFormattingEditProvider

The document formatting provider interface defines the contract between extensions and the formatting-feature.

Methods

```
</a>provideDocumentRangeFormatEdits(document: TextDocument, range: Range,  
options: FormattingOptions, token: CancellationToken): TextEdit\[\] | Thenable<TextEdit\[\]>  
</span>
```

Provide formatting edits for a range in a document.

The given range is a hint and providers can decide to format a smaller or larger range. Often this is done by adjusting the start and end of the range to full syntax nodes.

| Parameter | Description |
|---|--|
| document:
TextDocument | The document in which the command was invoked. |
| range: Range | The range which should be formatted. |
| options:
FormattingOptions | Options controlling formatting. |
| token:
CancellationToken | A cancellation token. |
| Returns | Description |
| TextEdit[] Thenable<TextEdit[]> | A set of text edits or a thenable that resolves to such. The lack of a result can be signaled by returning <code>undefined</code> , <code>null</code> , or an empty array. |

DocumentSelector

A language selector is the combination of one or many language identifiers and [language filters](#).

- *sample* - `let sel:DocumentSelector = 'typescript' ;`
- *sample* - `let sel:DocumentSelector = ['typescript', { language: 'json', pattern: '**/tsconfig.json' }] ;`

[DocumentSelector: string | DocumentFilter | string | DocumentFilter\[\]](#)

DocumentSymbolProvider

The document symbol provider interface defines the contract between extensions and the [go to symbol](#)-feature.

Methods

[provideDocumentSymbols\(document: TextDocument, token: CancellationToken\): SymbolInformation\[\] | Thenable<SymbolInformation\[\]>](#)

Provide symbol information for the given document.

| Parameter | Description |
|---|--|
| document: TextDocument | The document in which the command was invoked. |
| token: CancellationToken | A cancellation token. |
| Returns | Description |
| SymbolInformation[] Thenable<SymbolInformation[]> | An array of document highlights or a thenable that resolves to such. The lack of a result can be signaled by returning <code>undefined</code> , <code>null</code> , or an empty array. |

EnterAction

Describes what to do when pressing Enter.

Properties

[`appendText?: string`](#)

Describes text to be appended after the new line and after the indentation.

[`indentAction: IndentAction`](#)

Describe what to do with the indentation.

[`removeText?: number`](#)

Describes the number of characters to remove from the new line's indentation.

Event<T>

Represents a typed event.

A function that represents an event to which you subscribe by calling it with a listener function as argument.

- **sample** - `item.onDidChange(function(event) { console.log("Event happened: " + event);});`

[`\(listener: \(e: T\) => any, thisArgs?: any, disposables?: Disposable\[\]\): Disposable`](#)

A function that represents an event to which you subscribe by calling it with a listener function as argument.

A function that represents an event to which you subscribe by calling it with a listener function as argument.

| Parameter | Description |
|----------------------------|---|
| listener: (e: T) => any | The listener function will be called when the event happens. |
| thisArgs?: any | The <code>this</code> -argument which will be used when calling the event listener. |
| disposables?: Disposable[] | An array to which a <code>Disposable</code> will be added. |
| Returns | Description |
| Disposable | A disposable which unsubscribes the event listener. |

EventEmitter<T>

An event emitter can be used to create and manage an [event](#) for others to subscribe to. One emitter always owns one event.

Use this class if you want to provide event from within your extension, for instance inside a [TextDocumentContentProvider](#) or when providing API to other extensions.

Properties

[event](#): Event<T>

The event listeners can subscribe to.

Methods

[dispose\(\)](#): void

Dispose this object and free resources.

| Returns | Description |
|---------|-------------|
| void | |

[fire\(data?: T\)](#): void

Notify all subscribers of the [event](#). Failure of one or more listener will not fail this function call.

| Parameter | Description |
|--------------------------|-------------------|
| data?: T | The event object. |
| Returns | Description |
| void | |

Extension<T>

Represents an extension.

To get an instance of an [Extension](#) use [getExtension](#).

Properties

[`exports: T`](#)

The public API exported by this extension. It is an invalid action to access this field before this extension has been activated.

- *readonly*

[`extensionPath: string`](#)

The absolute file path of the directory containing this extension.

- *readonly*

[`id: string`](#)

The canonical extension identifier in the form of: [`publisher.name`](#).

- *readonly*

[`isActive: boolean`](#)

[`true`](#) if the extension has been activated.

- *readonly*

[`packageJSON: any`](#)

The parsed contents of the extension's package.json.

- *readonly*

Methods

activate(): Thenable<T>

Activates this extension and returns its public API.

| >Returns | Description |
|-------------|---|
| Thenable<T> | A promise that will resolve when this extension has been activated. |

ExtensionContext

An extension context is a collection of utilities private to an extension.

An instance of an `ExtensionContext` is provided as the first parameter to the `activate` -call of an extension.

Properties

extensionPath: string

The absolute file path of the directory containing the extension.

globalState: Memento

A memento object that stores state independent of the current opened [workspace](#).

subscriptions: {dispose}[]

An array to which disposables can be added. When this extension is deactivated the disposables will be disposed.

workspaceState: Memento

A memento object that stores state in the context of the currently opened [workspace](#).

Methods

asAbsolutePath(relativePath: string): string

Get the absolute path of a resource contained in the extension.

| Parameter | Description |
|----------------------|---|
| relativePath: string | A relative path to a resource contained in the extension. |

| Returns | Description |
|---------|------------------------------------|
| string | The absolute path of the resource. |

FileSystemWatcher

A file system watcher notifies about changes to files and folders on disk.

To get an instance of a `FileSystemWatcher` use [createFileSystemWatcher](#).

Events

[onDidChange: Event<Uri>](#)

An event which fires on file/folder change.

[onDidCreate: Event<Uri>](#)

An event which fires on file/folder creation.

[onDelete: Event<Uri>](#)

An event which fires on file/folder deletion.

Static

[from\(...disposableLikes: {dispose: \(\) => any}\[\]\): Disposable](#)

Combine many disposable-likes into one. Use this method when having objects with a dispose function which are not instances of Disposable.

| Parameter | Description |
|--|--|
| <code>...disposableLikes: {dispose: () => any}[]</code> | Objects that have at least a <code>dispose</code> -function member. |
| Returns | Description |
| <code>Disposable</code> | Returns a new disposable which, upon dispose, will dispose all provided disposables. |

Constructors

[new FileSystemWatcher\(callOnDispose: Function\): FileSystemWatcher](#)

Creates a new Disposable calling the provided function on dispose.

| Parameter | Description |
|---|-----------------------------------|
| callOnDispose: Function | Function that disposes something. |
| Returns | Description |
| FileSystemWatcher | |

Properties

[ignoreChangeEvents: boolean](#)

true if this file system watcher has been created such that it ignores change file system events.

[ignoreCreateEvents: boolean](#)

true if this file system watcher has been created such that it ignores creation file system events.

[ignoreDeleteEvents: boolean](#)

true if this file system watcher has been created such that it ignores delete file system events.

Methods

[dispose\(\): any](#)

Dispose this object.

| Returns | Description |
|---------------------|-------------|
| any | |

FormattingOptions

Value-object describing what options formatting should use.

Properties

[insertSpaces: boolean](#)

Prefer spaces over tabs.

[tabSize: number](#)

Size of a tab in spaces.

Hover

A hover represents additional information for a symbol or word. Hovers are rendered in a tooltip-like widget.

Constructors

`new Hover(contents: MarkedString | MarkedString[], range?: Range): Hover`

Creates a new hover object.

| Parameter | Description |
|--|---------------------------------------|
| contents: <code>MarkedString MarkedString[]</code> | The contents of the hover. |
| range?: <code>Range</code> | The range to which the hover applies. |
| Returns | Description |
| <code>Hover</code> | |

Properties

`contents: MarkedString[]`

The contents of this hover.

`range: Range`

The range to which this hover applies. When missing, the editor will use the range at the current position or the current position itself.

HoverProvider

The hover provider interface defines the contract between extensions and the `hover`-feature.

Methods

`provideHover(document: TextDocument, position: Position, token: CancellationToken): Hover | Thenable<Hover>`

Provide a hover for the given position and document. Multiple hovers at the same position will be merged by the editor. A hover can have a range which defaults to the word range at the position when omitted.

| Parameter | Description |
|--|--|
| document: TextDocument | The document in which the command was invoked. |
| position: Position | The position at which the command was invoked. |
| token: CancellationToken | A cancellation token. |
| Returns | Description |
| Hover Thenable < Hover > | A hover or a thenable that resolves to such. The lack of a result can be signaled by returning <code>undefined</code> or <code>null</code> . |

IndentAction

Describes what to do with the indentation when pressing Enter.

Enumeration members

`Indent`

`IndentOutdent`

`None`

`Outdent`

IndentationRule

Describes indentation rules for a language.

Properties

[`decreaseIndentPattern: RegExp`](#)

If a line matches this pattern, then all the lines after it should be unindented once (until another rule matches).

[`increaseIndentPattern: RegExp`](#)

If a line matches this pattern, then all the lines after it should be indented once (until another rule matches).

[`indentNextLinePattern?: RegExp`](#)

If a line matches this pattern, then **only the next line** after it should be indented once.

[`unIndentedLinePattern?: RegExp`](#)

If a line matches this pattern, then its indentation should not be changed and it should not be evaluated against the other rules.

InputBoxOptions

Options to configure the behavior of the input box UI.

Properties

[`password?: boolean`](#)

Set to true to show a password prompt that will not show the typed value.

[`placeHolder?: string`](#)

An optional string to show as place holder in the input box to guide the user what to type.

[`prompt?: string`](#)

The text to display underneath the input box.

[`validateInput?: \(value: string\) => string`](#)

An optional function that will be called to validate input and to give a hint to the user.

- *param* - The current value of the input box.
- *returns* - A human readable string which is presented as diagnostic message. Return `undefined`, `null`, or the empty string when 'value' is valid.

[`value?: string`](#)

The value to prefill in the input box.

LanguageConfiguration

The language configuration interfaces defines the contract between extensions and various editor features, like automatic bracket insertion, automatic indentation etc.

Properties

[characterPairSupport](#)?: {autoClosingPairs: {close: string, notIn: string[], open: string[]}[]}

Deprecated Do not use.

- *deprecated* - Will be replaced by a better API soon.

[electricCharacterSupport](#)?: {docComment: {close: string, lineStart: string, open: string}, scope: string}[]

Deprecated Do not use.

- *deprecated* - Will be replaced by a better API soon.

[brackets](#)?: CharacterPair[]

The language's brackets. This configuration implicitly affects pressing Enter around these brackets.

[comments](#)?: CommentRule

The language's comment settings.

[indentationRules](#)?: IndentationRule

The language's indentation settings.

[onEnterRules](#)?: OnEnterRule[]

The language's rules to be evaluated when pressing Enter.

[wordPattern](#)?: RegExp

The language's word definition. If the language supports Unicode identifiers (e.g. JavaScript), it is preferable to provide a word definition that uses exclusion of known separators. e.g.: A regex that matches anything except known separators (and dot is allowed to occur in a floating point number): `/(-?\d.\d\w)|([^\~!@\#@%\^&()=+[{}]\|;\:\\"\\,.\\<>\?\\s]+)/g`

Location

Represents a location inside a resource, such as a line inside a text file.

Constructors

[new Location](#)(uri: Uri, rangeOrPosition: Range | Position): Location

Creates a new location object.

| Parameter | Description |
|---|---|
| uri: Uri | The resource identifier. |
| rangeOrPosition: Range Position | The range or position. Positions will be converted to an empty range. |
| Returns | Description |
| Location | |

Properties

[range: Range](#)

The document range of this locations.

[uri: Uri](#)

The resource identifier of this location.

MarkedString

MarkedString can be used to render human readable text. It is either a markdown string or a code-block that provides a language and a code snippet. Note that markdown strings will be sanitized - that means html will be escaped.

[MarkedString: string](#) | {language: [string](#), value: [string](#)}

Memento

A memento represents a storage utility. It can store and retrieve values.

Methods

[get<T>\(key: string, defaultValue?: T\): T](#)

Return a value.

| Parameter | Description |
|----------------------------------|---|
| key: string | A string. |
| defaultValue?: T | A value that should be returned when there is no value (<code>undefined</code>) with the given key. |
| Returns | Description |
| T | The stored value, <code>undefined</code> , or the defaultValue. |

[`update\(key: string, value: any\): Thenable<void>`](#)

Store a value. The value must be JSON-stringifiable.

| Parameter | Description |
|--------------------------------------|--|
| key: string | A string. |
| value: any | A value. MUST not contain cyclic references. |
| Returns | Description |
| Thenable<void> | |

MessagelItem

Represents an action that is shown with an information, warning, or error message.

- see - [showInformationMessage](#)
- see - [showWarningMessage](#)
- see - [showErrorMessage](#)

Properties

[`title: string`](#)

A short title like 'Retry', 'Open Log' etc.

OnEnterRule

Describes a rule to be evaluated when pressing Enter.

Properties

action: EnterAction

The action to execute.

afterText?: RegExp

This rule will only execute if the text after the cursor matches this regular expression.

beforeText: RegExp

This rule will only execute if the text before the cursor matches this regular expression.

OnTypeFormattingEditProvider

The document formatting provider interface defines the contract between extensions and the formatting-feature.

Methods

provideOnTypeFormattingEdits(document: TextDocument, position: Position, ch: string, options: FormattingOptions, token: CancellationToken): TextEdit[] | Thenable<TextEdit[]>

Provide formatting edits after a character has been typed.

The given position and character should hint to the provider what range the position to expand to, like find the matching `{` when `}` has been entered.

| Parameter | Description |
|---|--|
| document: TextDocument | The document in which the command was invoked. |
| position: Position | The position at which the command was invoked. |
| ch: string | The character that has been typed. |
| options: FormattingOptions | Options controlling formatting. |
| token: CancellationToken | A cancellation token. |
| Returns | Description |
| TextEdit[] Thenable<TextEdit[]> | A set of text edits or a thenable that resolves to such. The lack of a result can be signaled by returning <code>undefined</code> , <code>null</code> , or an empty array. |

OutputChannel

An output channel is a container for readonly textual information.

To get an instance of an `outputchannel` use [createOutputChannel](#).

Properties

[`name: string`](#)

The human-readable name of this output channel.

- *readonly*

Methods

[`append\(value: string\): void`](#)

Append the given value to the channel.

| Parameter | Description |
|----------------------------|---|
| <code>value: string</code> | A string, falsy values will not be printed. |
| Returns | Description |
| <code>void</code> | |

[`appendLine\(value: string\): void`](#)

Append the given value and a line feed character to the channel.

| Parameter | Description |
|----------------------------|---|
| <code>value: string</code> | A string, falsy values will be printed. |
| Returns | Description |
| <code>void</code> | |

[`clear\(\): void`](#)

Removes all output from the channel.

| Returns | Description |
|-------------------|-------------|
| <code>void</code> | |

[`dispose\(\): void`](#)

Dispose and free associated resources.

| Returns | Description |
|---------|-------------|
| void | |

[`hide\(\): void`](#)

Hide this channel from the UI.

| Returns | Description |
|---------|-------------|
| void | |

[`show\(column?: ViewColumn, preserveFocus?: boolean\): void`](#)

Reveal this channel in the UI.

- **deprecated - This method is deprecated.**

| Parameter | Description |
|---|---|
| column?: ViewColumn | The column in which to show the channel, default in one . |
| preserveFocus?: boolean | When <code>true</code> the channel will not take focus. |
| Returns | Description |
| void | |

[`show\(preservceFocus?: boolean\): void`](#)

Reveal this channel in the UI.

| Parameter | Description |
|---------------------------------------|-------------|
| <code>preservceFocus?: boolean</code> | |
| Returns | Description |
| void | |

OverviewRulerLane

Represents different positions for rendering a decoration in an [overview ruler](#). The overview ruler supports three lanes.

Enumeration members

Center

2

Full

7

Left

1

Right

4

ParameterInformation

Represents a parameter of a callable-signature. A parameter can have a label and a doc-comment.

Constructors

new ParameterInformation(label: string, documentation?: string):
ParameterInformation

Creates a new parameter information object.

| Parameter | Description |
|------------------------|-----------------|
| label: string | A label string. |
| documentation?: string | A doc string. |
| Returns | Description |
| ParameterInformation | |

Properties

documentation: string

The human-readable doc-comment of this signature. Will be shown in the UI but can be omitted.

label: string

The label of this signature. Will be shown in the UI.

Position

Represents a line and character position, such as the position of the cursor.

Position objects are **immutable**. Use the [with](#) or [translate](#) methods to derive new positions from an existing position.

Constructors

[`new Position\(line: number, character: number\): Position`](#)

| Parameter | Description |
|--------------------------------|-------------------------------|
| <code>line: number</code> | A zero-based line value. |
| <code>character: number</code> | A zero-based character value. |
| Returns | Description |
| <code>Position</code> | |

Properties

[`character: number`](#)

The zero-based character value.

- *readonly*

[`line: number`](#)

The zero-based line value.

- *readonly*

Methods

[`compareTo\(other: Position\): number`](#)

Compare this to `other`.

| Parameter | Description |
|-----------------|--|
| other: Position | A position. |
| Returns | Description |
| number | A number smaller than zero if this position is before the given position, a number greater than zero if this position is after the given position, or zero when this and the given position are equal. |

isAfter(other: Position): boolean

Check if `other` is after this position.

| Parameter | Description |
|-----------------|--|
| other: Position | A position. |
| Returns | Description |
| boolean | <code>true</code> if position is on a greater line or on the same line on a greater character. |

isAfterOrEqual(other: Position): boolean

Check if `other` is after or equal to this position.

| Parameter | Description |
|-----------------|---|
| other: Position | A position. |
| Returns | Description |
| boolean | <code>true</code> if position is on a greater line or on the same line on a greater or equal character. |

isBefore(other: Position): boolean

Check if `other` is before this position.

| Parameter | Description |
|-----------------|--|
| other: Position | A position. |
| Returns | Description |
| boolean | <code>true</code> if position is on a smaller line or on the same line on a smaller character. |

[`isBeforeOrEqual\(other: Position\): boolean`](#)

Check if `other` is before or equal to this position.

| Parameter | Description |
|------------------------------|---|
| <code>other: Position</code> | A position. |
| Returns | Description |
| <code>boolean</code> | <code>true</code> if position is on a smaller line or on the same line on a smaller or equal character. |

[`isEqual\(other: Position\): boolean`](#)

Check if `other` equals this position.

| Parameter | Description |
|------------------------------|---|
| <code>other: Position</code> | A position. |
| Returns | Description |
| <code>boolean</code> | <code>true</code> if the line and character of the given position are equal to the line and character of this position. |

[`translate\(lineDelta?: number, characterDelta?: number\): Position`](#)

Create a new position relative to this position.

| Parameter | Description |
|--------------------------------------|--|
| <code>lineDelta?: number</code> | Delta value for the line value, default is <code>0</code> . |
| <code>characterDelta?: number</code> | Delta value for the character value, default is <code>0</code> . |
| Returns | Description |
| <code>Position</code> | A position which line and character is the sum of the current line and character and the corresponding deltas. |

[`with\(line?: number, character?: number\): Position`](#)

Create a new position derived from this position.

| Parameter | Description |
|--------------------|---|
| line?: number | Value that should be used as line value, default is the existing value |
| character?: number | Value that should be used as character value, default is the existing value |
| Returns | Description |
| Position | A position where line and character are replaced by the given values. |

QuickPickItem

Represents an item that can be selected from a list of items.

Properties

[`description: string`](#)

A human readable string which is rendered less prominent.

[`detail?: string`](#)

A human readable string which is rendered less prominent.

[`label: string`](#)

A human readable string which is rendered prominent.

QuickPickOptions

Options to configure the behavior of the quick pick UI.

Events

[`onDidSelectItem?: \(item: T | string\) => any`](#)

An optional function that is invoked whenever an item is selected.

Properties

[`matchOnDescription?: boolean`](#)

An optional flag to include the description when filtering the picks.

matchOnDetail?: boolean

An optional flag to include the detail when filtering the picks.

placeHolder?: string

An optional string to show as place holder in the input box to guide the user what to pick on.

Range

A range represents an ordered pair of two positions. It is guaranteed that `start.isBeforeOrEqual(end)`

Range objects are **immutable**. Use the `with`, `intersection`, or `union` methods to derive new ranges from an existing range.

Constructors

new Range(start: Position, end: Position): Range

Create a new range from two positions. If `start` is not before or equal to `end`, the values will be swapped.

| Parameter | Description |
|-----------------|-------------|
| start: Position | A position. |
| end: Position | A position. |
| Returns | Description |
| Range | |

new Range(startLine: number, startCharacter: number, endLine: number, endCharacter: number): Range

Create a new range from number coordinates. It is a shorter equivalent of using `new Range(new Position(startLine, startCharacter), new Position(endLine, endCharacter))`

| Parameter | Description |
|--|-------------------------------|
| startLine: number | A zero-based line value. |
| startCharacter: number | A zero-based character value. |
| endLine: number | A zero-based line value. |
| endCharacter: number | A zero-based character value. |
| Returns | Description |
| Range | |

Properties

[end: Position](#)

The end position. It is after or equal to [start](#).

- *readonly*

[isEmpty: boolean](#)

true iff `start` and `end` are equal.

[isSingleLine: boolean](#)

true iff `start.line` and `end.line` are equal.

[start: Position](#)

The start position. It is before or equal to [end](#).

- *readonly*

Methods

[contains\(positionOrRange: Position | Range\): boolean](#)

Check if a position or a range is contained in this range.

| Parameter | Description |
|---|--|
| positionOrRange: Position Range | A position or a range. |
| Returns | Description |
| boolean | true iff the position or range is inside or equal to this range. |

[`intersection\(range: Range\): Range`](#)

Intersect `range` with this range and returns a new range or `undefined` if the ranges have no overlap.

| Parameter | Description |
|------------------------------|---|
| range: Range | A range. |
| Returns | Description |
| Range | A range of the greater start and smaller end positions. Will return undefined when there is no overlap. |

[`isEqual\(other: Range\): boolean`](#)

Check if `other` equals this range.

| Parameter | Description |
|------------------------------|---|
| other: Range | A range. |
| Returns | Description |
| boolean | true when start and end are equal to start and end of this range. |

[`union\(other: Range\): Range`](#)

Compute the union of `other` with this range.

| Parameter | Description |
|------------------------------|---|
| other: Range | A range. |
| Returns | Description |
| Range | A range of smaller start position and the greater end position. |

with([start?: Position](#), [end?: Position](#)): [Range](#)

Create a new range derived from this range.

| Parameter | Description |
|----------------------------------|--|
| start?: Position | A position that should be used as start. The default value is the current start . |
| end?: Position | A position that should be used as end. The default value is the current end . |
| Returns | Description |
| Range | A range derived from this range with the given start and end position. If start and end are not different this range will be returned. |

ReferenceContext

Value-object that contains additional information when requesting references.

Properties

includeDeclaration: boolean

Include the declaration of the current symbol.

ReferenceProvider

The reference provider interface defines the contract between extensions and the [find references](#)-feature.

Methods

provideReferences([document: TextDocument](#), [position: Position](#), [context: ReferenceContext](#), [token: CancellationToken](#)): [Location\[\] | Thenable<Location\[\]>](#)

Provide a set of project-wide references for the given position and document.

| Parameter | Description |
|---|--|
| document: TextDocument | The document in which the command was invoked. |
| position: Position | The position at which the command was invoked. |
| context: ReferenceContext | |
| token: CancellationToken | A cancellation token. |
| Returns | Description |
| Location[] Thenable<Location[]> | An array of locations or a thenable that resolves to such. The lack of a result can be signaled by returning <code>undefined</code> , <code>null</code> , or an empty array. |

RenameProvider

The rename provider interface defines the contract between extensions and the [rename](#)-feature.

Methods

[`provideRenameEdits\(document: TextDocument, position: Position, newName: string, token: CancellationToken\): WorkspaceEdit | Thenable<WorkspaceEdit>`](#)

Provide an edit that describes changes that have to be made to one or many resources to rename a symbol to a different name.

| Parameter | Description |
|---|---|
| document: TextDocument | The document in which the command was invoked. |
| position: Position | The position at which the command was invoked. |
| newName: string | The new name of the symbol. If the given name is not valid, the provider must return a rejected promise. |
| token: CancellationToken | A cancellation token. |
| Returns | Description |
| WorkspaceEdit Thenable<WorkspaceEdit> | A workspace edit or a thenable that resolves to such. The lack of a result can be signaled by returning <code>undefined</code> or <code>null</code> . |

Selection

Represents a text selection in an editor.

Constructors

[new Selection\(anchor: Position, active: Position\): Selection](#)

Create a selection from two postions.

| Parameter | Description |
|----------------------------------|-------------|
| anchor: Position | A position. |
| active: Position | A position. |
| Returns | Description |
| Selection | |

[new Selection\(anchorLine: number, anchorCharacter: number, activeLine: number, activeCharacter: number\): Selection](#)

Create a selection from four coordinates.

| Parameter | Description |
|---|-------------------------------|
| anchorLine: number | A zero-based line value. |
| anchorCharacter: number | A zero-based character value. |
| activeLine: number | A zero-based line value. |
| activeCharacter: number | A zero-based character value. |
| Returns | Description |
| Selection | |

Properties

[active: Position](#)

The position of the cursor. This position might be before or after [anchor](#).

[anchor: Position](#)

The position at which the selection starts. This position might be before or after [active](#).

end: Position

The end position. It is after or equal to [start](#).

- *readonly*

isEmpty: boolean

true iff `start` and `end` are equal.

isReversed: boolean

A selection is reversed if [active.isBefore\(anchor\)](#).

isSingleLine: boolean

true iff `start.line` and `end.line` are equal.

start: Position

The start position. It is before or equal to [end](#).

- *readonly*

Methods

contains(positionOrRange: Position | Range): boolean

Check if a position or a range is contained in this range.

| Parameter | Description |
|-----------------------------------|--|
| positionOrRange: Position Range | A position or a range. |
| Returns | Description |
| boolean | true iff the position or range is inside or equal to this range. |

intersection(range: Range): Range

Intersect `range` with this range and returns a new range or `undefined` if the ranges have no overlap.

| Parameter | Description |
|--------------|---|
| range: Range | A range. |
| Returns | Description |
| Range | A range of the greater start and smaller end positions. Will return undefined when there is no overlap. |

isEqual(other: Range): boolean

Check if `other` equals this range.

| Parameter | Description |
|--------------|---|
| other: Range | A range. |
| Returns | Description |
| boolean | true when start and end are equal to start and end of this range. |

union(other: Range): Range

Compute the union of `other` with this range.

| Parameter | Description |
|--------------|---|
| other: Range | A range. |
| Returns | Description |
| Range | A range of smaller start position and the greater end position. |

with(start?: Position, end?: Position): Range

Create a new range derived from this range.

| Parameter | Description |
|----------------------------------|--|
| start?: Position | A position that should be used as start. The default value is the current start . |
| end?: Position | A position that should be used as end. The default value is the current end . |
| Returns | Description |
| Range | A range derived from this range with the given start and end position. If start and end are not different this range will be returned. |

SignatureHelp

Signature help represents the signature of something callable. There can be multiple signatures but only one active and only one active parameter.

Properties

[activeParameter: number](#)

The active parameter of the active signature.

[activeSignature: number](#)

The active signature.

[signatures: SignatureInformation\[\]](#)

One or more signatures.

SignatureHelpProvider

The signature help provider interface defines the contract between extensions and the [parameter hints](#)-feature.

Methods

[provideSignatureHelp\(document: TextDocument, position: Position, token: CancellationToken\): SignatureHelp | Thenable<SignatureHelp>](#)

Provide help for the signature at the given position and document.

| Parameter | Description |
|---|---|
| document: TextDocument | The document in which the command was invoked. |
| position: Position | The position at which the command was invoked. |
| token: CancellationToken | A cancellation token. |
| Returns | Description |
| SignatureHelp
Thenable < SignatureHelp > | Signature help or a thenable that resolves to such. The lack of a result can be signaled by returning <code>undefined</code> or <code>null</code> . |

SignatureInformation

Represents the signature of something callable. A signature can have a label, like a function-name, a doc-comment, and a set of parameters.

Constructors

[`new SignatureInformation\(label: string, documentation?: string\):`](#)
[`SignatureInformation`](#)

Creates a new signature information object.

| Parameter | Description |
|---|-----------------|
| label: string | A label string. |
| documentation?: string | A doc string. |
| Returns | Description |
| <code>SignatureInformation</code> | |

Properties

[`documentation: string`](#)

The human-readable doc-comment of this signature. Will be shown in the UI but can be omitted.

[`label: string`](#)

The label of this signature. Will be shown in the UI.

[parameters: ParameterInformation\[\]](#)

The parameters of this signature.

StatusBarItem

Represents the alignment of status bar items.

Enumeration members

Left

Right

StatusBarItem

A status bar item is a status bar contribution that can show text and icons and run a command on click.

Properties

[alignment: StatusBarAlignment](#)

The alignment of this item.

- *readonly*

[color: string](#)

The foreground color for this entry.

[command: string](#)

The identifier of a command to run on click. The command must be [known](#).

[priority: number](#)

The priority of this item. Higher value means the item should be shown more to the left.

- *readonly*

[text: string](#)

The text to show for the entry. You can embed icons in the text by leveraging the syntax:

```
My text $(icon-name) contains icons like $(icon'name) this one.
```

Where the icon-name is taken from the [octicon](#) icon set, e.g. `light-bulb`, `thumbsup`, `zap` etc.

[`tooltip: string`](#)

The tooltip text when you hover over this entry.

Methods

[`dispose\(\): void`](#)

Dispose and free associated resources. Call [hide](#).

| Returns | Description |
|-------------------|-------------|
| <code>void</code> | |

[`hide\(\): void`](#)

Hide the entry in the status bar.

| Returns | Description |
|-------------------|-------------|
| <code>void</code> | |

[`show\(\): void`](#)

Shows the entry in the status bar.

| Returns | Description |
|-------------------|-------------|
| <code>void</code> | |

SymbolInformation

Represents information about programming constructs like variables, classes, interfaces etc.

Constructors

[`new SymbolInformation\(name: string, kind: SymbolKind, range: Range, uri?: Uri, containerName?: string\): SymbolInformation`](#)

Creates a new symbol information object.

| Parameter | Description |
|--|---|
| name: string | The name of the symbol. |
| kind: SymbolKind | The kind of the symbol. |
| range: Range | The range of the location of the symbol. |
| uri?: Uri | The resource of the location of symbol, defaults to the current document. |
| containerName?: string | The name of the symbol containing the symbol. |
| Returns | Description |
| SymbolInformation | |

Properties

[containerName: string](#)

The name of the symbol containing this symbol.

[kind: SymbolKind](#)

The kind of this symbol.

[location: Location](#)

The location of this symbol.

[name: string](#)

The name of this symbol.

SymbolKind

A symbol kind.

Enumeration members

[Array](#)

[Boolean](#)

[Class](#)

Constant

Constructor

Enum

Field

File

Function

Interface

Key

Method

Module

Namespace

Null

Number

Object

Package

Property

String

Variable

TextDocument

Represents a text document, such as a source file. Text documents have [lines](#) and knowledge about an underlying resource like a file.

Properties

fileName: string

The file system path of the associated resource. Shorthand notation for [TextDocument.uri.fsPath](#). Independent of the uri scheme.

- *readonly*

[isDirty](#): boolean

true if there are unpersisted changes.

- *readonly*

[isUntitled](#): boolean

Is this document representing an untitled file.

- *readonly*

[languageId](#): string

The identifier of the language associated with this document.

- *readonly*

[lineCount](#): number

The number of lines in this document.

- *readonly*

[uri](#): Uri

The associated URI for this document. Most documents have the **file**-scheme, indicating that they represent files on disk. However, some documents may have other schemes indicating that they are not available on disk.

- *readonly*

[version](#): number

The version number of this document (it will strictly increase after each change, including undo/redo).

- *readonly*

Methods

[getText\(range?: Range\)](#): string

Get the text of this document. A substring can be retrieved by providing a range. The range will be [adjusted](#).

| Parameter | Description |
|-------------------------------|--|
| range?: Range | Include only the text included by the range. |
| Returns | Description |
| string | The text inside the provided range or the entire text. |

[`getWordRangeAtPosition\(position: Position\): Range`](#)

Get a word-range at the given position. By default words are defined by common separators, like space, -, _, etc. In addition, per language custom [word definitions](#) can be defined.

The position will be [adjusted](#).

| Parameter | Description |
|------------------------------------|--|
| position: Position | A position. |
| Returns | Description |
| Range | A range spanning a word, or <code>undefined</code> . |

[`lineAt\(line: number\): TextLine`](#)

Returns a text line denoted by the line number. Note that the returned object is *not* live and changes to the document are not reflected.

| Parameter | Description |
|------------------------------|----------------------------------|
| line: number | A line number in [0, lineCount). |
| Returns | Description |
| TextLine | A line . |

[`lineAt\(position: Position\): TextLine`](#)

Returns a text line denoted by the position. Note that the returned object is *not* live and changes to the document are not reflected.

The position will be [adjusted](#).

- see - [TextDocument.lineAt](#)

| Parameter | Description |
|------------------------------------|--------------------------|
| position: Position | A position. |
| Returns | Description |
| TextLine | A line . |

[`offsetAt\(position: Position\): number`](#)

Converts the position to a zero-based offset.

The position will be [adjusted](#).

| Parameter | Description |
|------------------------------------|----------------------------|
| position: Position | A position. |
| Returns | Description |
| number | A valid zero-based offset. |

[`positionAt\(offset: number\): Position`](#)

Converts a zero-based offset to a position.

| Parameter | Description |
|--------------------------------|------------------------------------|
| offset: number | A zero-based offset. |
| Returns | Description |
| Position | A valid position . |

[`save\(\): Thenable<boolean>`](#)

Save the underlying file.

| Returns | Description |
|---|---|
| Thenable<boolean> | A promise that will resolve to true when the file has been saved. |

[`validatePosition\(position: Position\): Position`](#)

Ensure a position is contained in the range of this document.

| Parameter | Description |
|------------------------------------|---|
| position: Position | A position. |
| Returns | Description |
| Position | The given position or a new, adjusted position. |

[`validateRange\(range: Range\): Range`](#)

Ensure a range is completely contained in this document.

| Parameter | Description |
|------------------------------|---|
| range: Range | A range. |
| Returns | Description |
| Range | The given range or a new, adjusted range. |

TextDocumentChangeEvent

An event describing a transactional [document](#) change.

Properties

[`contentChanges: TextDocumentContentChangeEvent\[\]`](#)

An array of content changes.

[`document: TextDocument`](#)

The affected document.

TextDocumentContentChangeEvent

An event describing an individual change in the text of a [document](#).

Properties

[`range: Range`](#)

The range that got replaced.

[`rangeLength: number`](#)

The length of the range that got replaced.

text: string

The new text for the range.

TextDocumentContentProvider

A text document content provider allows to add readonly documents to the editor, such as source from a dll or generated html from md.

Content providers are [registered](#) for a [uri-scheme](#). When a uri with that scheme is to be [loaded](#) the content provider is asked.

Events

onDidChange?: Event<Uri>

An event to signal a resource has changed.

Methods

provideTextDocumentContent(uri: Uri, token: CancellationToken): string | Thenable<string>

Provide textual content for a given uri.

The editor will use the returned string-content to create a readonly [document](#). Resources allocated should be released when the corresponding document has been [closed](#).

| Parameter | Description |
|---------------------------|--|
| uri: Uri | An uri which scheme matches the scheme this provider was registered for. |
| token: CancellationToken | A cancellation token. |
| Returns | Description |
| string Thenable<string> | A string or a thenable that resolves to such. |

TextEdit

A text edit represents edits that should be applied to a document.

Static

[`delete\(range: Range\): TextEdit`](#)

Utility to create a delete edit.

| Parameter | Description |
|------------------------------|-------------------------|
| range: Range | A range. |
| Returns | Description |
| TextEdit | A new text edit object. |

[`insert\(position: Position, newText: string\): TextEdit`](#)

Utility to create an insert edit.

| Parameter | Description |
|------------------------------------|---|
| position: Position | A position, will become an empty range. |
| newText: string | A string. |
| Returns | Description |
| TextEdit | A new text edit object. |

[`replace\(range: Range, newText: string\): TextEdit`](#)

Utility to create a replace edit.

| Parameter | Description |
|---------------------------------|-------------------------|
| range: Range | A range. |
| newText: string | A string. |
| Returns | Description |
| TextEdit | A new text edit object. |

Constructors

[`new TextEdit\(range: Range, newText: string\): TextEdit`](#)

Create a new TextEdit.

| Parameter | Description |
|---------------------------------|-------------|
| range: Range | A range. |
| newText: string | A string. |
| Returns | Description |
| TextEdit | |

Properties

[newText: string](#)

The string this edit will insert.

[range: Range](#)

The range this edit applies to.

TextEditor

Represents an editor that is attached to a [document](#).

Properties

[document: TextDocument](#)

The document associated with this text editor. The document will be the same for the entire lifetime of this text editor.

[options: TextEditorOptions](#)

Text editor options.

[selection: Selection](#)

The primary selection on this text editor. Shorthand for `TextEditor.selections[0]`.

[selections: Selection\[\]](#)

The selections in this text editor. The primary selection is always at index 0.

[viewColumn: ViewColumn](#)

The column in which this editor shows. Will be `undefined` in case this isn't one of the three main editors, e.g an embedded editor.

Methods

[`edit\(callback: \(editBuilder: TextEditorEdit\) => void\): Thenable<boolean>`](#)

Perform an edit on the document associated with this text editor.

The given callback-function is invoked with an [edit-builder](#) which must be used to make edits. Note that the edit-builder is only valid while the callback executes.

| Parameter | Description |
|---|--|
| <code>callback: (editBuilder: TextEditorEdit) => void</code> | A function which can make edits using an edit-builder . |
| Returns | Description |
| <code>Thenable<boolean></code> | A promise that resolves with a value indicating if the edits could be applied. |

[`hide\(\): void`](#)

Hide the text editor.

- *deprecated* - **This method is deprecated.** Use the command 'workbench.action.closeActiveEditor' instead. This method shows unexpected behavior and will be removed in the next major update.

| Returns | Description |
|-------------------|-------------|
| <code>void</code> | |

[`revealRange\(range: Range, revealType?: TextEditorRevealType\): void`](#)

Scroll as indicated by `revealType` in order to reveal the given range.

| Parameter | Description |
|--|---|
| <code>range: Range</code> | A range. |
| <code>revealType?: TextEditorRevealType</code> | The scrolling strategy for revealing <code>range</code> . |
| Returns | Description |
| <code>void</code> | |

[`setDecorations\(decorationType: TextEditorDecorationType, rangesOrOptions: Range\[\] | DecorationOptions\[\]\): void`](#)

Adds a set of decorations to the text editor. If a set of decorations already exists with the given [decoration type](#), they will be replaced.

- see - [createTextEditorDecorationType](#).

| Parameter | Description |
|--|--|
| decorationType: TextEditorDecorationType | A decoration type. |
| rangesOrOptions: Range [] DecorationOptions [] | Either ranges or more detailed options . |
| Returns | Description |
| void | |

[show\(column?: ViewColumn\): void](#)

Show the text editor.

- *deprecated* - **This method is deprecated.** Use [window.showTextDocument](#) instead.
This method shows unexpected behavior and will be removed in the next major update.

| Parameter | Description |
|-------------------------------------|--|
| column?: ViewColumn | The column in which to show this editor. |
| Returns | Description |
| void | |

TextEditorDecorationType

Represents a handle to a set of decorations sharing the same styling options in a [text editor](#).

To get an instance of a [TextEditorDecorationType](#) use [createTextEditorDecorationType](#).

Properties

[key: string](#)

Internal representation of the handle.

- *readonly*

Methods

[dispose\(\): void](#)

Remove this decoration type and all decorations on all text editors using it.

| Returns | Description |
|---------|-------------|
| void | |

TextEditorEdit

A complex edit that will be applied in one transaction on a TextEditor. This holds a description of the edits and if the edits are valid (i.e. no overlapping regions, document was not changed in the meantime, etc.) they can be applied on a [document](#) associated with a [text editor](#).

Methods

[`delete\(location: Range | Selection\): void`](#)

Delete a certain text region.

| Parameter | Description |
|---|---|
| location: Range Selection | The range this operation should remove. |
| Returns | Description |
| void | |

[`insert\(location: Position, value: string\): void`](#)

Insert text at a location. You can use `\r\n` or `\n` in `value` and they will be normalized to the current [document](#). Although the equivalent text edit can be made with [replace](#), [insert](#) will produce a different resulting selection (it will get moved).

| Parameter | Description |
|------------------------------------|---|
| location: Position | The position where the new text should be inserted. |
| value: string | The new text this operation should insert. |
| Returns | Description |
| void | |

[`replace\(location: Position | Range | Selection, value: string\): void`](#)

Replace a certain text region with a new value. You can use `\r\n` or `\n` in `value` and they will be normalized to the current [document](#).

| Parameter | Description |
|--|--|
| location: Position Range Selection | The range this operation should remove. |
| value: string | The new text this operation should insert after removing <code>location</code> . |
| Returns | Description |
| void | |

TextEditorOptions

Represents a [text editor's options](#).

Properties

[`insertSpaces?: boolean | string`](#)

When pressing Tab insert `n` spaces. When getting a text editor's options, this property will always be a boolean (resolved). When setting a text editor's options, this property is optional and it can be a boolean or `"auto"`.

[`tabSize?: number | string`](#)

The size in spaces a tab takes. This is used for two purposes:

- the rendering width of a tab character;
- the number of spaces to insert when `insertSpaces` is true. When getting a text editor's options, this property will always be a number (resolved). When setting a text editor's options, this property is optional and it can be a number or `"auto"`.

TextEditorOptionsChangeEvent

Represents an event describing the change in a [text editor's options](#).

Properties

[`options: TextEditorOptions`](#)

The new value for the [text editor's options](#).

[`textEditor: TextEditor`](#)

The [text editor](#) for which the options have changed.

TextEditorRevealType

Represents different [reveal](#) strategies in a text editor.

Enumeration members

Default

InCenter

InCenterIfOutsideViewport

TextEditorSelectionChangeEvent

Represents an event describing the change in a [text editor's selections](#).

Properties

selections: Selection[]

The new value for the [text editor's selections](#).

textEditor: TextEditor

The [text editor](#) for which the selections have changed.

TextEditorViewColumnChangeEvent

Represents an event describing the change of a [text editor's view column](#).

Properties

textEditor: TextEditor

The [text editor](#) for which the options have changed.

viewColumn: ViewColumn

The new value for the [text editor's view column](#).

TextLine

Represents a line of text, such as a line of source code.

TextLine objects are **immutable**. When a [document](#) changes, previously retrieved lines will not represent the latest state.

Properties

[`firstNonWhitespaceCharacterIndex: number`](#)

The offset of the first character which is not a whitespace character as defined by `/\s/`.

- *readonly*

[`isEmptyOrWhitespace: boolean`](#)

Whether this line is whitespace only, shorthand for

`TextLine.firstNonWhitespaceCharacterIndex === TextLine.text.length`.

- *readonly*

[`lineNumber: number`](#)

The zero-based line number.

- *readonly*

[`range: Range`](#)

The range this line covers without the line separator characters.

- *readonly*

[`rangelIncludingLineBreak: Range`](#)

The range this line covers with the line separator characters.

- *readonly*

[`text: string`](#)

The text of this line without the line separator characters.

- *readonly*

ThemableDecorationRenderOptions

Represents theme specific rendering styles for a [text editor decoration](#).

Properties

[`backgroundColor?: string`](#)

Background color of the decoration. Use `rgba()` and define transparent background colors to play well with other decorations.

`borderColor?: string`

CSS styling property that will be applied to text enclosed by a decoration.

`borderRadius?: string`

CSS styling property that will be applied to text enclosed by a decoration.

`borderSpacing?: string`

CSS styling property that will be applied to text enclosed by a decoration.

`borderStyle?: string`

CSS styling property that will be applied to text enclosed by a decoration.

`borderWidth?: string`

CSS styling property that will be applied to text enclosed by a decoration.

`color?: string`

CSS styling property that will be applied to text enclosed by a decoration.

`cursor?: string`

CSS styling property that will be applied to text enclosed by a decoration.

`gutterIconPath?: string`

An **absolute path** to an image to be rendered in the gutterIconPath.

`letterSpacing?: string`

CSS styling property that will be applied to text enclosed by a decoration.

`outlineColor?: string`

CSS styling property that will be applied to text enclosed by a decoration.

`outlineStyle?: string`

CSS styling property that will be applied to text enclosed by a decoration.

`outlineWidth?: string`

CSS styling property that will be applied to text enclosed by a decoration.

`overviewRulerColor?: string`

The color of the decoration in the overview ruler. Use `rgba()` and define transparent colors to play well with other decorations.

[`textDecoration?: string`](#)

CSS styling property that will be applied to text enclosed by a decoration.

Uri

A universal resource identifier representing either a file on disk or another resource, like untitled resources.

Static

[`file\(path: string\): Uri`](#)

Create an URI from a file system path. The `scheme` will be `file`.

| Parameter | Description |
|---------------------------|----------------------------|
| <code>path: string</code> | A file system or UNC path. |
| Returns | Description |
| <code>Uri</code> | A new Uri instance. |

[`parse\(value: string\): Uri`](#)

Create an URI from a string. Will throw if the given value is not valid.

| Parameter | Description |
|----------------------------|-----------------------------|
| <code>value: string</code> | The string value of an Uri. |
| Returns | Description |
| <code>Uri</code> | A new Uri instance. |

Properties

[`authority: string`](#)

Authority is the `www.msft.com` part of `http://www.msft.com/some/path?query#fragment`. The part between the first double slashes and the next slash.

[`fragment: string`](#)

Fragment is the `fragment` part of `http://www.msft.com/some/path?query#fragment`.

[`fsPath: string`](#)

The string representing the corresponding file system path of this URI.

Will handle UNC paths and normalize windows drive letters to lower-case. Also uses the platform specific path separator. Will *not* validate the path for invalid characters and semantics. Will *not* look at the scheme of this URI.

[`path: string`](#)

Path is the `/some/path` part of `http://www.msft.com/some/path?query#fragment`.

[`query: string`](#)

Query is the `query` part of `http://www.msft.com/some/path?query#fragment`.

[`scheme: string`](#)

Scheme is the `http` part of `http://www.msft.com/some/path?query#fragment`. The part before the first colon.

Methods

[`toJSON\(\): any`](#)

Returns a JSON representation of this Uri.

| Returns | Description |
|------------------|-------------|
| <code>any</code> | An object. |

[`toString\(\): string`](#)

Returns a canonical representation of this URI. The representation and normalization of a URI depends on the scheme.

| Returns | Description |
|---------------------|---|
| <code>string</code> | A string that is the encoded version of this Uri. |

ViewColumn

Denotes a column in the VS Code window. Columns are used to show editors side by side.

Enumeration members

One

1

Three

3

Two

2

WorkspaceConfiguration

Represents the workspace configuration. The workspace configuration is always a merged view of the configuration of the current [workspace](#) and the installation-wide configuration.

Methods

[`get<T>\(section: string, defaultValue?: T\): T`](#)

Return a value from this configuration.

| Parameter | Description |
|-------------------------------|--|
| <code>section: string</code> | Configuration name, supports <i>dotted</i> names. |
| <code>defaultValue?: T</code> | A value should be returned when no value could be found, is <code>undefined</code> . |
| Returns | Description |
| <code>T</code> | The value <code>section</code> denotes or the default. |

[`has\(section: string\): boolean`](#)

Check if this configuration has a certain value.

| Parameter | Description |
|------------------------------|---|
| <code>section: string</code> | configuration name, supports <i>dotted</i> names. |
| Returns | Description |
| <code>boolean</code> | <code>true</code> iff the section doesn't resolve to <code>undefined</code> . |

WorkspaceEdit

A workspace edit represents textual changes for many documents.

Properties

[size: number](#)

The number of affected resources.

- *readonly*

Methods

[delete\(uri: Uri, range: Range\): void](#)

Delete the text at the given range.

| Parameter | Description |
|--------------|------------------------|
| uri: Uri | A resource identifier. |
| range: Range | A range. |
| Returns | Description |
| void | |

[entries\(\): \[Uri, TextEdit\[\]\]\[\]](#)

Get all text edits grouped by resource.

| Returns | Description |
|---------------------|--|
| [Uri, TextEdit[][]] | An array of [Uri, TextEdit[]] -tuples. |

[get\(uri: Uri\): TextEdit\[\]](#)

Get the text edits for a resource.

| Parameter | Description |
|------------|-------------------------|
| uri: Uri | A resource identifier. |
| Returns | Description |
| TextEdit[] | An array of text edits. |

[has\(uri: Uri\): boolean](#)

Check if this edit affects the given resource.

| Parameter | Description |
|-----------|--|
| uri: Uri | A resource identifier. |
| Returns | Description |
| boolean | true if the given resource will be touched by this edit. |

[`insert\(uri: Uri, position: Position, newText: string\): void`](#)

Insert the given text at the given position.

| Parameter | Description |
|--------------------|------------------------|
| uri: Uri | A resource identifier. |
| position: Position | A position. |
| newText: string | A string. |
| Returns | Description |
| void | |

[`replace\(uri: Uri, range: Range, newText: string\): void`](#)

Replace the given range with given text for the given resource.

| Parameter | Description |
|-----------------|------------------------|
| uri: Uri | A resource identifier. |
| range: Range | A range. |
| newText: string | A string. |
| Returns | Description |
| void | |

[`set\(uri: Uri, edits: TextEdit\[\]\): void`](#)

Set (and replace) text edits for a resource.

| Parameter | Description |
|------------------------------------|-------------------------|
| uri: Uri | A resource identifier. |
| edits: TextEdit [] | An array of text edits. |
| Returns | Description |
| void | |

WorkspaceSymbolProvider

The workspace symbol provider interface defines the contract between extensions and the [symbol search](#)-feature.

Methods

`provideWorkspaceSymbols(query: string, token: CancellationToken): SymbolInformation[] | Thenable<SymbolInformation[]>`

Project-wide search for a symbol matching the given query string. It is up to the provider how to search given the query string, like substring, indexOf etc.

| Parameter | Description |
|--|--|
| query: string | A non-empty query string. |
| token: CancellationToken | A cancellation token. |
| Returns | Description |
| SymbolInformation [] Thenable<SymbolInformation[]> | An array of document highlights or a thenable that resolves to such. The lack of a result can be signaled by returning <code>undefined</code> , <code>null</code> , or an empty array. |

Complex Commands API

This document lists a set of complex commands that we consider stable. They are called complex commands because they require parameters and often return a value. You can use the commands in conjunction with the `executeCommand` API.

The following is a sample of how to preview a HTML document:

```
let uri = Uri.parse('file:///some/path/to/file.html');
let success = await commands.executeCommand('vscode.previewHtml', uri);
```

Commands

`vscode.executeWorkspaceSymbolProvider` - Execute all workspace symbol provider.

- *query* Search string
- *(returns)* A promise that resolves to an array of SymbolInformation-instances.

`vscode.executeDefinitionProvider` - Execute all definition provider.

- *uri* Uri of a text document
- *position* Position of a symbol
- *(returns)* A promise that resolves to an array of Location-instances.

`vscode.executeHoverProvider` - Execute all hover provider.

- *uri* Uri of a text document
- *position* Position of a symbol
- *(returns)* A promise that resolves to an array of Hover-instances.

`vscode.executeDocumentHighlights` - Execute document highlight provider.

- *uri* Uri of a text document
- *position* Position in a text document
- *(returns)* A promise that resolves to an array of DocumentHighlight-instances.

`vscode.executeReferenceProvider` - Execute reference provider.

- *uri* Uri of a text document
- *position* Position in a text document
- *(returns)* A promise that resolves to an array of Location-instances.

`vscode.executeDocumentRenameProvider` - Execute rename provider.

- *uri* Uri of a text document
- *position* Position in a text document
- *newName* The new symbol name
- *(returns)* A promise that resolves to a WorkspaceEdit.

`vscode.executeSignatureHelpProvider` - Execute signature help provider.

- *uri* Uri of a text document
- *position* Position in a text document
- *(returns)* A promise that resolves to SignatureHelp.

`vscode.executeDocumentSymbolProvider` - Execute document symbol provider.

- *uri* Uri of a text document
- *(returns)* A promise that resolves to an array of SymbolInformation-instances.

`vscode.executeCompletionItemProvider` - Execute completion item provider.

- *uri* Uri of a text document
- *position* Position in a text document
- *(returns)* A promise that resolves to a CompletionList-instance.

`vscode.executeCodeActionProvider` - Execute code action provider.

- *uri* Uri of a text document
- *range* Range in a text document
- *(returns)* A promise that resolves to an array of CompletionItem-instances.

`vscode.executeCodeLensProvider` - Execute completion item provider.

- *uri* Uri of a text document
- *(returns)* A promise that resolves to an array of Commands.

`vscode.executeFormatDocumentProvider` - Execute document format provider.

- *uri* Uri of a text document
- *options* Formatting options
- *(returns)* A promise that resolves to an array of TextEdits.

`vscode.executeFormatRangeProvider` - Execute range format provider.

- *uri* Uri of a text document
- *range* Range in a text document
- *options* Formatting options
- *(returns)* A promise that resolves to an array of TextEdits.

`vscode.executeFormatOnTypeProvider` - Execute document format provider.

- *uri* Uri of a text document
- *position* Position in a text document
- *ch* Character that got typed
- *options* Formatting options
- *(returns)* A promise that resolves to an array of TextEdits.

`vscode.previewHtml` - Preview an html document.

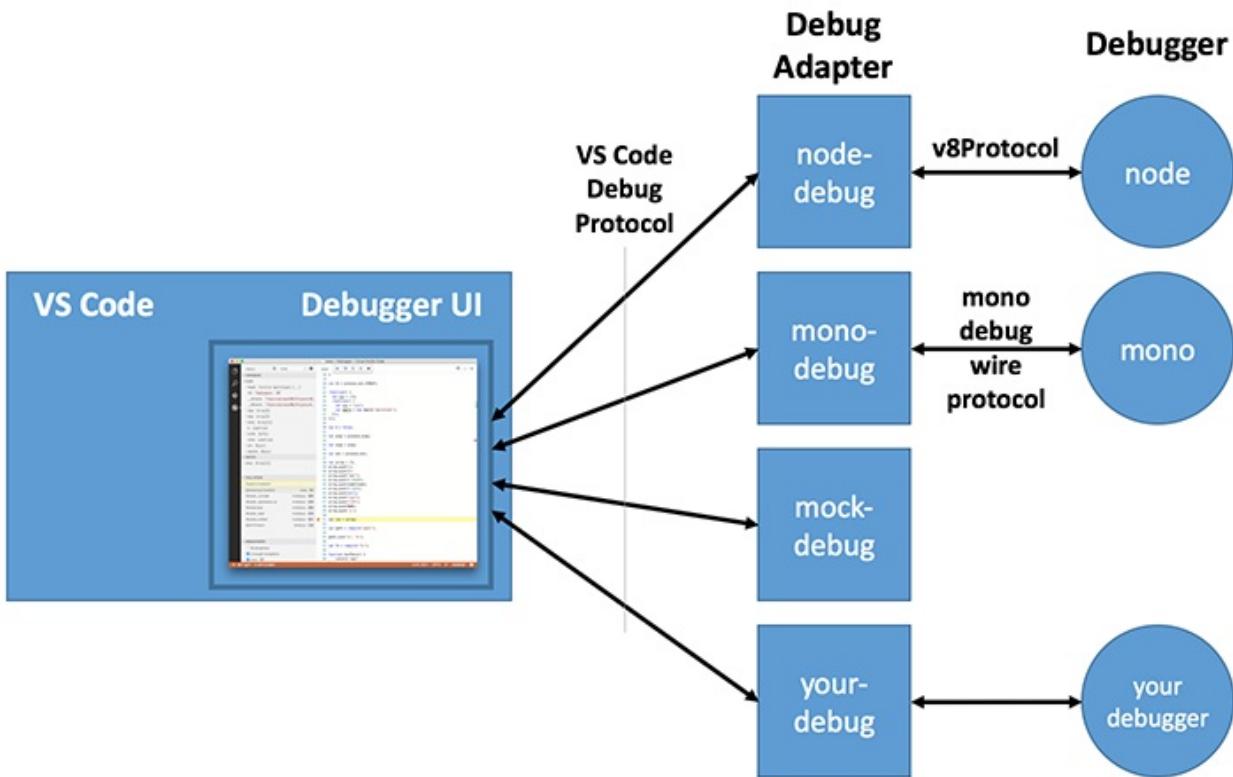
- *uri* Uri of the document to preview.
- *column* (optional) Column in which to preview.

`editor.action.showReferences` - Show references at a position in a file

- *uri* The text document in which to show references
- *position* The position at which to show
- *locations* An array of locations.

Debugging API

Since Visual Studio Code implements a language agnostic debug UI, it does not communicate directly with real debuggers but instead talks to so-called *debug adapters* through an abstract wire protocol, the *VS Code Debug Protocol* (or CDP for short).



Extensibility of the debug component of VS Code is currently limited to adding new debug adapters. So it is not (yet) possible to extend the debugger UI in similar ways as for example the editor component of VS Code.

Debug Adapter

A debug adapter is a standalone executable that talks to a real debugger and translates between the abstract CDP and the concrete protocol of the debugger. Since a debug adapter can be implemented in the language that is best suited for a given debugger backend, the wire protocol is more important than the API of a particular client library that implements that protocol.

You can find the protocol specification expressed as a TypeScript definition file in the GitHub repository [vscode-debugadapter-node](#). It shows the detailed structure of the CDP protocol requests, responses and events. The protocol is also available as the NPM module [vscode-](#)

[debugprotocol](#) .

We have already implemented client libraries for CDP in TypeScript and C#, but only the JavaScript/TypeScript client library is already available as an NPM module [vscode-debugadapter-node](#) . You can find the C# client library in the [Mono Debug](#) repository.

The following debugger extension projects can serve as examples for how to implement debug adapters:

| GitHub Project | Description | Implementation Language |
|----------------------------|-------------------------------|-------------------------|
| Mock Debug | A 'fake' debugger | TypeScript/JavaScript |
| Node Debug | The built-in Node.js debugger | TypeScript/JavaScript |
| Mono Debug | A simple C# debugger for Mono | C# |

The VS Code Debug Protocol in a Nutshell

In this section we will give a high-level overview of the interaction between VS Code and a debug adapter. This should help you in your implementation of a debug adapter based on CDP.

When a debug session starts, VS Code launches the debug adapter executable and talks to it through `stdin` and `stdout`. VS Code sends an **initialize** request to configure the adapter with information about the path format (native or URI) and whether line and column values are 0 or 1 based. If your adapter is derived from the TypeScript or C# default implementation [DebugSession](#) , you don't have to handle the initialize request yourself.

Depending on the 'request' attribute used in the launch configuration created by the user, VS Code either sends a **launch** or an **attach** request. For **launch** the debug adapter has to launch a runtime or program so that it can be debugged. If the program can interact with the user through `stdin/stdout`, it is important that the debug adapter launches the program in an interactive terminal or console. For **attach** the debug adapter has to attach or connect to an already running program.

Since arguments for both requests are highly dependent on a specific debug adapter implementation, the CDP does not prescribe any arguments. Instead VS Code passes all arguments from the user's launch configuration to the **launch** or **attach** requests. A schema for IntelliSense and hover information for these attributes can be contributed in the [package.json](#) of the debug adapter extension. This will guide the user when creating or editing launch configurations.

Since VS Code persists breakpoints on behalf of the debug adapter, it has to register the breakpoints with the debug adapter when a session starts. Since VS Code does not know when is a good time for this, the debug adapter is expected to send an **initialize** event to VS Code to announce that it is ready to accept breakpoint configuration requests.

VS Code will then send all breakpoints by calling these breakpoint configuration requests:

- **setBreakpoints** for every source file with breakpoints,
- **setFunctionBreakpoints** if the debug adapter supports function breakpoints,
- **setExceptionBreakpoints** if the debug adapter supports any exception options,
- **configurationDoneRequest** to indicate the end of the configuration sequence.

So don't forget to send the *initialize* event when you are ready to accept breakpoints. Otherwise persisted breakpoints are not restored.

The *setBreakpoint* request sets all breakpoints that exist for a file (so it is not incremental). A simple implementation of this semantics in the debug adapter is to clear all breakpoints for a file and then set the breakpoints specified in the request. *setBreakpoints* and *setFunctionBreakpoints* are expected to return the 'actual' breakpoints and VS Code updates the UI dynamically if a breakpoint could not be set at the requested position and was moved by the debugger backend.

Whenever the program stops (on program entry, because a breakpoint was hit, an exception occurred, or the user requested execution to be paused), the debug adapter has to send a **stopped** event with the appropriate reason and thread id. Upon receipt VS Code will first request the **threads** (see below), and then the **stacktrace** (a list of stack frames) for the thread mentioned in the stopped event. If the user then drills into the stack frame, VS Code first requests the **scopes** for a stack frame, and then the **variables** for a scope. If a variable is itself structured, VS Code requests its properties through additional *variables* requests. This leads to the following hierarchy:

```

Threads
  Stackframes
    Scopes
      Variables
      ...
      Variables
  
```

The VS Code debug UI supports multiple threads (but you are probably not aware of this if you are only using the Node.js debugger). Whenever VS Code receives a **stopped** or a **thread** event, VS Code requests all **threads** that exist at that point in time and displays them if there are more than one. If only one thread is detected, the VS Code UI stays in single thread mode. **Thread** events are optional but a debug adapter can send them to force VS Code to update the threads UI dynamically even when not in a stopped state.

After a successful **launch** or **attach** VS Code requests the baseline of currently existing threads with the **threads** request and then starts to listen for **thread** events to detect new or terminated threads. Even if your debug adapter does not support multiple threads, it must implement the **threads** request and return a single (dummy) thread. The id of this thread must be used in all requests where a thread id is required, e.g. **stacktrace**, **pause**, **continue**, **next**, **stepIn**, and **stepOut**.

VS Code terminates a debug session with the **disconnect** request. If the debug target was 'launched' *disconnect* is expected to terminate the target program (even forcefully if necessary). If the debug target has been 'attached' initially, *disconnect* should detach it from the target (so that it will continue to run). In both cases and in the case that the target terminated normally or crashed the debug adapter must fire a **terminated** event. After receiving a response from the *disconnect* request, VS Code will terminate the debug adapter.

Next Steps

To learn more about VS Code extensibility model, try these topics:

- [Example Debuggers](#) - See a working 'mock' debugger example
- [Extension API Overview](#) - Learn about the full VS Code extensibility model.
- [Extension Manifest File](#) - VS Code package.json extension manifest file reference
- [Contribution Points](#) - VS Code contribution points reference

Common Questions

Nothing yet

语言

- 概述
- Javascript
- JSON
- HTML(暂无)
- CSS, Sass and Less
- TypeScript
- Markdown
- C++
- Java(暂无)
- PHP
- Python
- Go(暂无)
- Dockerfile
- T-SQL(暂无)
- C#

语言 Languages

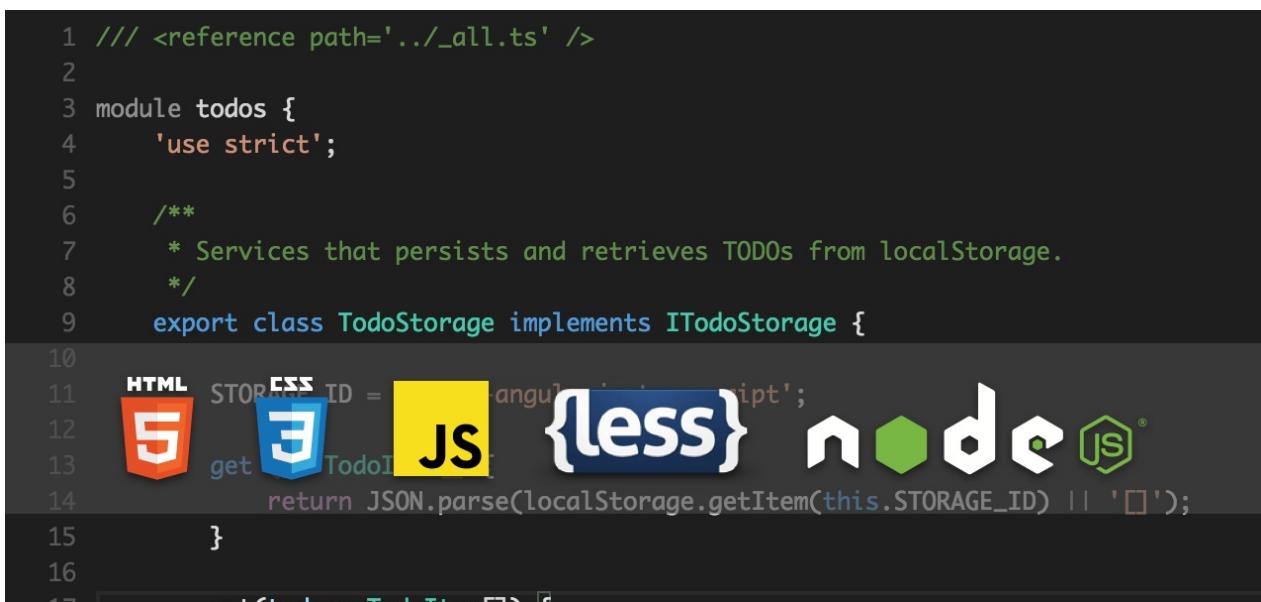
支持哪些语言 What Languages are Supported

In Visual Studio Code, we have support for many languages out of the box and more through language extensions available on the [VS Code Marketplace](#).

对于VS Code, 我们已经内置支持了很多语言（开箱即用）。更多的语言，将利用语言扩展，通过[VS Code 市场](#)得到支持。

Tip: You can also add support for your favorite language through TextMate colorizers. See [Colorizers](#) to learn how to integrate TextMate .tmLanguage syntax files into VS Code.

小贴士：你还可以通过TextMate的colorizers，来添加支持你最喜欢的语言。参考[Colorizers](#)来学习如何通过TextMate的.tmLanguage语法文件集成VS Code。



```

1 /// <reference path='../_all.ts' />
2
3 module todos {
4     'use strict';
5
6     /**
7      * Services that persists and retrieves TODOS from localStorage.
8      */
9     export class TodoStorage implements ITodoStorage {
10
11         ID = 'angular-todo';
12
13         getTodoItem(id: string): string {
14             return JSON.parse(localStorage.getItem(this.ID) || '[]');
15         }
16     }
17 }
```

The richness of support varies across the different languages. The table below provides a brief description of Visual Studio Code's various languages features. Click on any linked item to get an overview of how to use VS Code in the context of that language.

支持的丰富性因语言的不同而不同。下表提供的VS Code的各种语言功能的简要说明。点击任何链接项目获得如何在该语言的上下文中使用VS代码的概述。

| 功能 Features | 语言 Languages |
|--|--|
| 语法着色，括号匹配 Syntax coloring, bracket matching | Batch, Clojure, Coffee Script, Dockerfile, F#, Go, Jade, Java, HandleBars, Ini, Lua, Makefile, Objective-C, Perl, PowerShell, Python, R, Razor, Ruby, Rust, SQL, Visual Basic, XML |
| + 代码片断 Snippets | Groovy, Markdown, Swift |
| + 智能感知，语言分析，提纲
IntelliSense, linting, outline | C++, CSS, HTML, JavaScript, JSON, Less, PHP, Sass |
| + 重构，查找所有的引用
Refactoring, find all references | TypeScript, C# |

语言扩展 Language Extensions

The community is providing VS Code language support for nearly any modern programming language. To see if there are extensions for the language you're interested in, simply bring up the extension Marketplace and filter on the language name. Type

`kb(workbench.action.showCommands)` , 'ext inst' to bring up the extension Marketplace dropdown and then type the language name to filter the results.

社区提供几乎任何现代编程语言 VS Code 语言支持。要看看是否有您感兴趣的语种扩展，只需调出的语种名的扩展名市场和过滤器。键入 `KB (workbench.action.showCommands)` , 'ext inst' 调出扩展市场下拉框，然后键入语种名称来筛选结果。



You can also browse the VS Code Marketplace directly to look for [supported languages](#).

您也可以直接浏览VS code 市场寻找 [被支持的语言](#)。

为选中的文件更改语言 Changing the Language for the Selected File

In VS Code, we default the language support for a file based on its filename extension. However at times you may wish to change language modes, to do this click on the language indicator - which is located on the right hand of the status bar. This will bring up the Command Palette for Select Language Mode.

在VS Code，我们默认基于文件的扩展名来对应语言支持。但是有时候你可能希望改变语言模式，点击语言指示器 - 这是位于状态栏的右边。这将弹出选择语言模式的命令面板。



Tip: You can persist file associations with the `files.associations` setting.

小贴士：通过 `files.associations` setting，你可以保持文件的关联。

For example, the setting below adds the `.myphp` file extension to the `php` language:

例如，下面添加的是设置 `.myphp` 文件扩展名为 `php` 语言：

```
"files.associations": {  
    "*.myphp": "php"  
}
```

下一步 Next Steps

Now you know that VS Code has support for the languages you care about. Read on...

现在你知道VS Code支持了对你关心的语言。阅读它吧.....

- 提升编辑体验 [Editing Evolved](#) - 语言分析，智能感知，高亮显示，查看，跳转到定义，及其他 Lint, IntelliSense, Lightbulbs, Peek and Goto Definition and more
- 调试 [Debugging](#) - 这是VS Code真正的亮点This is where VS Code really shines
- 定制 [Customization](#) - 主题，设置和键盘绑定 themes, settings and keyboard bindings

常见问题 Common Questions

Q: Can I contribute my own language service?

A: Yes you can! Check out the [example language server](#) in the [Extending Visual Studio Code](#) documentation.

Q: 我可以贡献自己的语言服务吗？

A: Yes you can! Check out the [example language server](#) in the [Extending Visual Studio Code](#) documentation.

Q: Can I map additional file extensions to a language?

Q: 我可以附加文件扩展名映射到一个语言吗？

A: Yes, with the `files.associations` [setting](#) you can map file extensions to an existing language either globally or per workspace.

A: 是的。通过 `files.associations` [setting](#) 您可以为全局或每个工作区的文件扩展名映射到现有的语言。

Here is an example that will associate more file extensions to the PHP language:

下面是将文指定的件扩展名关联到PHP语言的例子：

```
"files.associations": {  
    "*.php4": "php",  
    "*.php5": "php"  
}
```

You can also configure full file paths to languages if needed. The following example associates all files in a folder `somefolder` to PHP:

如果需要，您还可以配置完整的文件路径到指定的语言。下面的示例是将文件夹 `somefolder` 中所有的文件关联到PHP：

```
"files.associations": {  
    "**/somefolder/*.*": "php"  
}
```

Note that the pattern is a [glob pattern](#) that will match on the full path of the file if it contains a `/` and will match on the file name otherwise.

请注意，此模式是一个[全局模式](#)。如果它包含了 `/` 将匹配文件的完整路径。否则将匹配文件名。

JavaScript

Rich Editing Support 丰富的编辑支持

Visual Studio Code uses the TypeScript language service to make authoring JavaScript easy. In addition to syntactical features like format, format on type and outlining, you also get language service features such as **Peek**, **Go to Definition**, **Find all References**, and **Rename Symbol**.

Visual Studio Code 使用 TypeScript 语言服务以简单地创作 JavaScript。除了一般的语法内容例如格式、对于类型的格式化、缩略图以外，你还将获得其他语言服务内容包括一瞥，转到定义，找到所有引用，以及重命名符号。

JavaScript Projects (`jsconfig.json`) JavaScript 项目 (`jsconfig.json`)

VS Code's JavaScript support can operate in two different modes:

VS Code 的 JavaScript 支持可以在两种模式下运行：

- **File Scope - no `jsconfig.json`:** In this mode, JavaScript files opened in Visual Studio Code are treated as independent units. As long as a file `a.js` doesn't reference a file `b.ts` explicitly (either using `/// reference directives` or `CommonJS modules`), there is no common project context between the two files.
- **文件范围 - 没有 `jsconfig.json` :** 在这一模式下，在 Visual Studio Code 中打开的 JavaScript 文件被视为独立的单元。只要一个文件“`a.js`”没有显性的引用一个文件“`b.js`”（无论是使用 `/// reference directives` 还是使用 `CommonJS modules`），两个文件之间没有共同的项目内容。
- **Explicit Project - with `jsconfig.json`:** A JavaScript project is defined via a `jsconfig.json` file. The presence of such a file in a directory indicates that the directory is the root of a JavaScript project. The file itself can optionally list the files belonging to the project, the files to be excluded from the project, as well as compiler options (see below).
- **显性项目 - 包含 `jsconfig.json` :** 一个 JavaScript 项目通过“`jsconfig.json`”文件进行定义。在一个目录下存在这样一个文件即表明这是一个 JavaScript 项目的根目录。这个文件本身可以可选择的列出包含在这个项目的文件，也可以包含项目中剔除的文件，以及编译器选

项。（见下）

The JavaScript experience is much better when you have a `jsconfig.json` file in your workspace that defines the project context. For this reason, we provide a hint to create a `jsconfig.json` file when you open a JavaScript file in a fresh workspace. The `jsconfig.json` file corresponds to a TypeScript project `tsconfig.json` file with the attribute `allowJS` implicitly set to `true`. If no `files` attribute is present, then this defaults to including all files in the containing directory and subdirectories. When a `files` attribute is specified, only those files are included.

当在你的工作空间中有一个“`jsconfig.json`”文件以定义项目内容时，JavaScript的体验会更佳。因此，当你在一个新的工作空间中打开JavaScript文件时，我们会提供一个建立“`jsconfig.json`”文件的提示。当“`jsconfig.json`”文件和TypeScript项目的 `tsconfig.json`文件协同工作时，其`allowJS`属性隐性地设为真。如果`file`属性不存在，则默认地将目录及子目录下的所有文件包括进徐昂目中，如果`file`属性存在，则按照他的值包括。

Make sure that you place the `jsconfig.json` at the root of your JavaScript project and not just at the root of your workspace. Below is a `jsconfig.json` file which defines the JavaScript `target` to be `ES6` and the `exclude` attribute excludes the `node_modules` folder.

请确定你讲“`jsconfig.json`”文件放在了你的JavaScript项目的根目录下，而不仅仅是你的工作空间根目录。以下是一个“`jsconfig.json`”文件，他定义了JavaScript的“`target`”为“`ES6`”，以及用“`exclude`”属性排除了“`node_modules`”文件夹。

```
{  
  "compilerOptions": {  
    "target": "ES6"  
  },  
  "exclude": [  
    "node_modules"  
  ]  
}
```

Here is an example with an explicit `files` attribute.

本案例包括了一个显性的“`files`”属性。

```
{  
  "compilerOptions": {  
    "target": "ES6"  
  },  
  "files": [  
    "src/app.js"  
  ]  
}
```

The `files` attribute cannot be used in conjunction with the `exclude` attribute. If both are specified, the `files` attribute takes precedence.

“files”属性不能与“exclude”属性同时出现，如果两者都有的话，“文件”属性优先度较高。

In more complex projects, you may have more than one `jsconfig.json` file defined inside a workspace, as illustrated in below for a project with a `client` and `server` folder, that are a separate project context:

在更复杂的项目中，你也许在一个工作空间中会有不止一个“`jsconfig.json`”文件，正如如下所示的一个项目，他包含了一个“`client`”文件夹和一个“`server`”文件夹，他们是分开的项目环境。



Excludes 排除

Whenever possible, you should exclude folders with JavaScript files that are not part of the source code for your project.

任何时候，你都应该将并非你项目源代码的JavaScript文件所在的文件夹排除出去。

Note: If you do not have a `jsconfig.json` in your workspace, VS Code will by default exclude the `node_modules` folder and the folder defined by the `out` attribute.

注：如果在你的工作空间没有“`jsconfig.json`”文件，VS Code会默认地排除“`node_modules`”文件夹以及被“`out`”属性定义的文件夹。

Below is a table mapping common project components to their installation folders which are recommended to exclude:

以下表格表明了一个常见项目的构件，以及安装文件夹中推荐排除的：

| Component | folder to exclude |
|------------------------------|--|
| node | exclude the <code>node_modules</code> folder |
| webpack , webpack-dev-server | exclude the content folder, e.g., <code>dist</code> . |
| bower | exclude the <code>bower_components</code> folder |
| ember | exclude the <code>tmp</code> and <code>temp</code> folders |
| jspm | exclude the <code>jspm_packages</code> folder |

| 构件 | 排除的文件夹 |
|------------------------------|---|
| node | 排除“ <code>node_modules</code> ”文件夹 |
| webpack , webpack-dev-server | 排除内容文件夹，例如“ <code>dist</code> ” |
| bower | 排除“ <code>bower_components</code> ”文件夹 |
| ember | 排除“ <code>tmp</code> and <code>temp</code> ”文件夹 |
| jspm | 排除“ <code>jspm_packages</code> ”文件夹 |

When your JavaScript project is growing too large, it is often because of library folders like `node_modules` . If VS Code detects that your project is growing too large, it will prompt you to edit the `exclude` list.

当你的JavaScript项目变得过大的时候，其原因经常是因为库文件夹比如说“`node_modules`” 。如果VS Code发现你的项目过大时，他会建议你编辑“`exclude`”名单。

Tip: Sometimes changes to configuration, such as adding or editing a `jsconfig.json` file are not picked up correctly. Running the **Reload Java Script** command should reload the project and pick up the changes.

Tip: 有时会对于配置的改变，比如添加或者编辑“`jsconfig.json`”文件会发生错误。进行 **Reload Java Script** (重新载入Java Script) 命令来重新载入项目，并且更新改动。

jsconfig Options jsconfig 选项

Below are jsconfig options to configure the JavaScript language support.

以下是jsconfig选项，他用来配置JavaScript语言支持功能。

| Option | Description |
|---|---|
| <code>noLib</code> | Do not include the default library file (<code>lib.d.ts</code>) |
| <code>target</code> | Specifies which default library (<code>lib.d.ts</code>) to use. The values are "ES3", "ES5", "ES6". |
| <code>experimentalDecorators</code> | Enables experimental support for proposed ES decorators. |
| <code>allowSyntheticDefaultImports</code> | Allow default imports from modules with no default export. This does not affect code emit, just typechecking. |

| 选项 | 描述 |
|---|--|
| <code>noLib</code> | 不要包含默认的库文件 (<code>lib.d.ts</code>) |
| <code>target</code> | 指明使用哪一个默认库 (<code>lib.d.ts</code>)。可选的值包括“ES3”，“ES5”，“ES6” |
| <code>experimentalDecorators</code> | 为制定的ES指示器启用实验性的支持 |
| <code>allowSyntheticDefaultImports</code> | 允许对没有默认出口的模块进行默认输入。这一选项不会影响代码发布，只是字面检查 |

IntelliSense 智能感知

The JavaScript Support uses different strategies to provide IntelliSense.

JavaScript支持使用不同的策略进行智能感知。

IntelliSense based on type inference 基于类型推理的智能感知

JavaScript uses the same inference as TypeScript to determine the type of a value.

JavaScript使用与TypeScript相同的推理机制来判断一个值的类型。

The following patterns are also recognized:

以下的模式也是被承认的：

- "**ES3-style**" classes, specified using a constructor function and assignments to the `prototype` property.
- **CommonJS-style** module patterns, specified as property assignments on the `exports` object, or assignments to the `module.exports` property.
- "**ES3-style**" 使用构造函数并且对原型属性赋值，以此进行规定的类。

- **CommonJS**风格的模块，对出口对象属性赋值或是通过赋值“`module.exports`”属性，以此进行规定的类。

The **AMD** (Asynchronous Module Definition) module pattern is currently not supported.

AMD (异步模块定义) 模块模式现在不被支持。

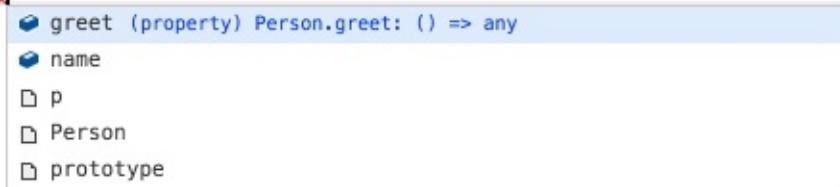
IntelliSense offers both inferred proposals and the global identifiers of the project. The inferred symbols are presented first, followed by the global identifiers (with the document icon), as you can see in the image below.

智能感知既可以提供推断的建议，也可以提供这一项目中使用的全局标识符。其中推断的将首先显示，然后是全局标识符（伴随着文档图标），如下图所示。

```

1 function Person(name) {
2     this.name = name;
3 }
4 Person.prototype.greet = function greet() {
5     return this.name;
6 }
7 var p = new Person('joe');
8 p.

```



JSDoc annotations JSDoc 注释

Where type inference does not provide the desired type information, (or just for documentation purposes), type information may be provided explicitly via **JSDoc** annotations.

当类型推断无法提供想要的类型信息时，（或者仅仅是为了文档目的），类型信息可能通过 **JSDoc**注释明确地提供。

This [document](#) describes the **JSDoc** annotations currently supported.

这一[文档](#)描述了现在支持的**JSDoc**注释。

TypeScript definition file TypeScript 定义文件

You can also get IntelliSense for libraries through the use of type definition `.d.ts` files. [DefinitelyTyped](#) is a repository of typings files for all major JavaScript libraries and environments. The typings are easily managed using [Typings](#), the TypeScript Definition manager.

你也可以通过使用类型定义 `.d.ts` 文件获得智能感知对库的支持。[DefinitelyTyped](#)是一个包含了现主要JavaScript库和环境类型文件的文档库。这些类型文件可以简单地使用TypeScript定义管理器，[Typings](#)，进行管理。

For example `typings install --ambient node` installs all the typings for the built-in Node.js modules. If your project has a `jsconfig.json` file, then make sure that `typings` is contained in the project context defined by the location of the `jsconfig.json` file. If you have no `jsconfig.json`, then you need to manually add a `/// reference` to the `.d.ts` from each JavaScript file.

比如说，

Tip: When you want to use ES6 style imports but the typings do not yet use ES6 style exports, then set the [TypeScript compiler option](#) `allowSyntheticDefaultImports` to true.

```
{
  "compilerOptions": {
    "target": "ES6",
    "module": "commonjs",
    "allowSyntheticDefaultImports": true
  },
  "exclude": [
    "node_modules"
  ]
}
```

Mixed TypeScript and JavaScript projects

It is now possible to have mixed TypeScript and JavaScript projects. Existing JavaScript code using the **CommonJS** module format, may be imported and consumed by TypeScript code using the **ECMAScript 2015** module syntax. Conversely, TypeScript code written to provide a well-defined API contract for a service, may be referenced by JavaScript code that is written to call that service, thus providing rich IntelliSense at design time.

To enable JavaScript inside a TypeScript project, you can set the `allowJs` property to `true` in the TypeScript project's `tsconfig.json` file.

Compiling JavaScript down-level

One of the key features TypeScript provides is the ability to use the latest JavaScript language features, and emit code that can execute in JavaScript runtimes that don't yet understand those newer features. With JavaScript using the same language service, it too

can now take advantage of this same feature.

The TypeScript compiler `tsc` can down-level compile JavaScript files from ES6 to another language level. Configure the `jsconfig.json` with the desired options and then use the `-p` argument to make `tsc` use your `jsconfig.json` file, e.g. `tsc -p jsconfig.json` to down-level compile.

The following compiler options in `jsconfig.json` apply when `tsc` is used for down level compiling of ES6 JavaScript to an older version:

| Option | Description |
|-----------------------|---|
| module | Specify module code generation. The values are "commonjs", "system", "umd", "amd", "es6", "es2015" |
| diagnostics | Show diagnostic information. |
| emitBOM | Emit a UTF-8 Byte Order Mark (BOM) in the beginning of output files. |
| inlineSourceMap | Emit a single file with source maps instead of having a separate file. |
| inlineSources | Emit the source alongside the sourcemaps within a single file; requires --inlineSourceMap to be set. |
| jsx | Specify JSX code generation: "preserve" or "react". |
| reactNamespace | Specifies the object invoked for createElement and __spread when targeting 'react' JSX emit. |
| mapRoot | Specifies the location as an uri in a string where debugger should locate map files instead of generated locations. |
| noEmit | Do not emit output. |
| noEmitHelpers | Do not generate custom helper functions like __extends in compiled output. |
| noEmitOnError | Do not emit outputs if any type checking errors were reported. |
| noResolve | Do not resolve triple-slash references or module import targets to the input files. |
| outFile | Concatenate and emit output to single file. |
| outDir | Redirect output structure to the directory. |
| removeComments | Do not emit comments to output. |
| rootDir | Specifies the root directory of input files. Use to control the output directory structure with --outDir. |
| sourceMap | Generates corresponding '.map' file. |
| sourceRoot | Specifies the location where debugger should locate JavaScript files instead of source locations. |
| stripInternal | 'do not emit declarations for code that has an '@internal' annotation. |
| watch | Watch input files. |
| emitDecoratorMetadata | Emit design-type metadata for decorated declarations in source. |
| noImplicitUseStrict | Do not emit "use strict" directives in module output. |

JavaScript Formatting

VS Code provides several formatting settings for JavaScript. They can all be found in the `javascript.format settings` name space.

```
// Defines space handling after a comma delimiter
"javascript.format.insertSpaceAfterCommaDelimiter": boolean,

// Defines space handling after a semicolon in a for statement
"javascript.format.insertSpaceAfterSemicolonInForStatements": boolean,

// Defines space handling after a binary operator
"javascript.format.insertSpaceBeforeAndAfterBinaryOperators": boolean,

// Defines space handling after keywords in control flow statement
"javascript.format.insertSpaceAfterKeywordsInControlFlowStatements": boolean,

// Defines space handling after function keyword for anonymous functions
"javascript.format.insertSpaceAfterFunctionKeywordForAnonymousFunctions": boolean,

// Defines space handling after opening and before closing non empty parenthesis
"javascript.format.insertSpaceAfterOpeningAndBeforeClosingNonemptyParenthesis": boolean
'

// Defines space handling after opening and before closing non empty brackets
"javascript.format.insertSpaceAfterOpeningAndBeforeClosingNonemptyBrackets": boolean,

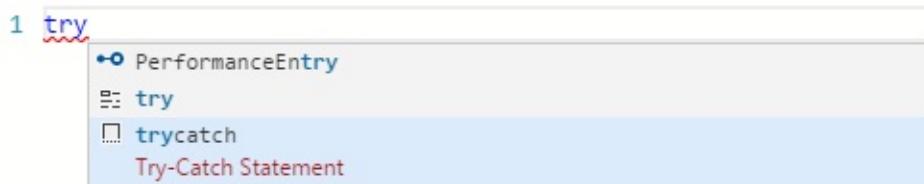
// Defines whether an open brace is put onto a new line for functions or not
"javascript.format.placeOpenBraceOnNewLineForFunctions": boolean,

// Defines whether an open brace is put onto a new line for control blocks or not
"javascript.format.placeOpenBraceOnNewLineForControlBlocks": boolean,
```



Snippets for JavaScript

VS Code has several built-in snippets that will come up as you type or you can press `kb(editor.action.triggerSuggest)` (**Trigger Suggest**) and you will see a context specific list of suggestions.



Selecting the snippet with `kbstyle(Tab)` results in:

```
1 try {  
2     |  
3 } catch (error) {  
4  
5 }
```

Tip: You can add in your own User Defined Snippets for JavaScript. See [User Defined Snippets](#) to find out how.

Run Babel inside VS Code

The **Babel** transpiler turns ES6 files into readable ES5 JavaScript with Source Maps. You can easily integrate **Babel** into your workflow by adding this code to your `tasks.json` file (located under the workspace's `.vscode` folder). The `isBuildCommand` switch makes this task the `Task: Run Build Task` gesture. `isWatching` tells VS Code not to wait for this task to finish. To learn more, go to [Tasks](#).

```
{  
    "version": "0.1.0",  
    "command": "${workspaceRoot}/node_modules/.bin/babel",  
    "isShellCommand": true,  
    "tasks": [  
        {  
            "args": ["src", "--out-dir", "lib", "-w", "--source-maps"],  
            "taskName": "watch",  
            "suppressTaskName": true,  
            "isBuildCommand": true,  
            "isWatching": true  
        }  
    ]  
}
```

Once you have added this, you can start **Babel** with the `kb(workbench.action.tasks.build)` (**Run Build Task**) command and it will compile all files from the `src` directory into the `lib` directory.

JSX and React Native

VS Code supports **JSX** and **React Native**. To get IntelliSense for **React/JSX**, install the typings for `react-global` by running `typings install --ambient react-global` from the terminal. To get IntelliSense for **React Native**, run `typings install --ambient react-native`.

React Native examples often use the experimental **Object Rest/Spread** operator. This is not yet supported by VS Code. If you want to use it, it is recommended that you disable the built-in syntax checking (see below).

To enable ES6 import statements for **React Native**, you need to set the

`allowSyntheticDefaultImports` compiler option to `true`. This tells the compiler to create synthetic default members and you get IntelliSense. **React Native** uses **Babel** behind the scenes to create the proper run-time code with default members. If you also want to do debugging of **React Native** code then you can install the [React Native Extension](#).

Disable Syntax Validation when using non ES6 constructs

Some users want to use syntax constructs like the proposed Object Rest/Spread Properties. However, these are currently not supported by VS Code's JavaScript support and are flagged as errors. For users who still want to use these future features, we provide the `javascript.validate.enable` setting. With `javascript.validate.enable: false` you disable all built-in syntax checking. If you do this, we recommend that you use a linter like [ESLint](#) to validate your code. Since the JavaScript support doesn't understand ES7 constructs, features like IntelliSense might not be fully accurate.

JavaScript Linters (ESLint, JSHint)

VS Code provides support for [ESLint](#) and [JSHint](#) via [extensions](#). If enabled, the JavaScript code is validated as you type and reported problems can be navigated to and fixed inside VS Code.

To enable one of the linters, do the following:

- Install the corresponding linter globally or inside the workspace folder that contains the JavaScript code to be validated. For example, using `npm install -g eslint` or `npm install -g jshint`, respectively.
- Install the [ESLint](#) or [JSHint](#) extension. The linter is enabled after installation. You can disable a linter via the corresponding settings `"eslint.enable": true` or `"jshint.enable": true`, respectively.
- Use a `.eslintrc.json` or `.jshintrc` file in the root of your workspace to configure the linter. You can use `eslint --init` to create an initial version of the `.eslintrc.json` file.

Tip: You get IntelliSense and hovering inside the `.eslintrc.json` and the `.jshintrc` files.

It is recommended that you enable the linter rules that warn about undefined and unused variables.

In JSHint:

```
"undef": true,  
"unused": true,
```

In ESLint:

```
"no-undef": 1,  
"no-unused-vars": 1,
```

Next Steps

Read on to find out about:

- [TypeScript](#) - VS Code has great support for TypeScript which brings structure and strong typing to your JavaScript code, without compromising the good parts.

Common Questions

Q: Can I debug minified/uglified JavaScript?

A: Yes, you can.

JSON

JSON is a data format that is common in configuration files like `package.json` or `project.json`. We also use it extensively in VS Code for our configuration files. When opening a file that ends with `.json`, VS Code provides the following set of features that make it simpler to write or modify the file's content.

JSON是一种数据格式，常见于配置文件如 `package.json` 或 `project.json`。我们在VS Code中也广泛地使用它作为配置文件。当打开与结尾的文件 `.json` 时，VSCode提供了，能更简单地编写或修改文件内容的功能集。

JSON 注释 JSON Comments

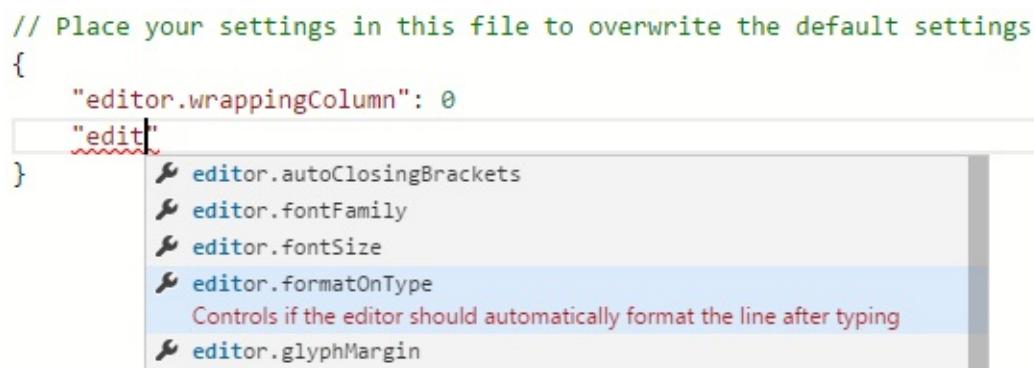
Comments in JSON are an extension to JSON specification that is supported by VS Code. You can use single line (`//`) as well as block comments (`/ /`) as used in JavaScript.

JSON中的JSON是一种JSON规范的扩展，是被VS Code支持的。在使用JavaScript时，使用单行 (`//`) 以及块 (`/ /`) 注释。

智能感知和验证 IntelliSense & Validation

For properties and values (`kb(editor.action.triggerSuggest)`), both for JSON data with and without schema, we offer up suggestions as you type with IntelliSense. We also perform structural and value verification based on an associated JSON schema giving you red squiggly lines.

对于属性和值(`kb(editor.action.triggerSuggest)`)，无论JSON数据是否是使用模式(schema)的JSON数据，智能感知在你输入时，都提供建议。基于关联的JSON模式，还进行了结构和值验证并显示红色标记。



包及项目依赖 Package and Project Dependencies

We also offer IntelliSense for specific value sets such as package and project dependencies in `package.json`, `project.json` and `bower.json`.

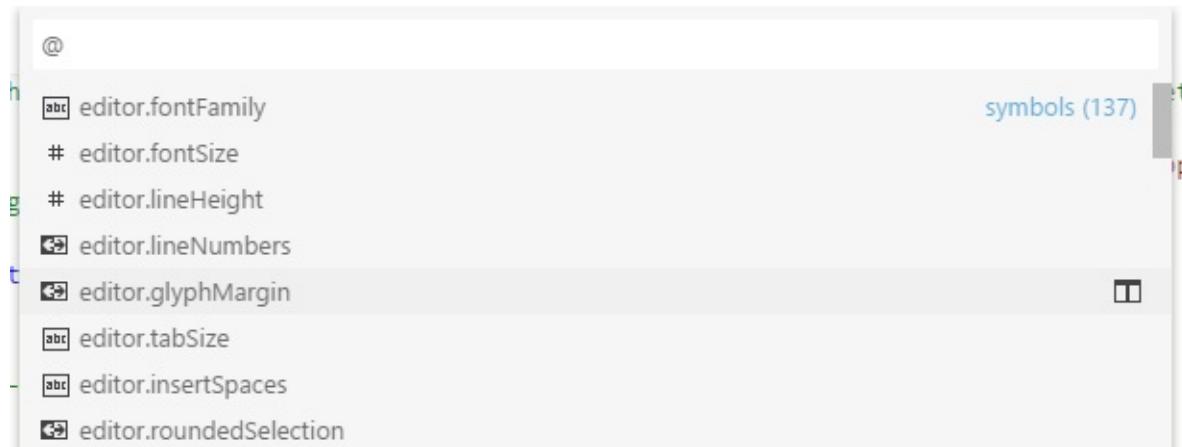
通过 `package.json`, `project.json` 和 `bower.json`，我们还提供智能感知特定值集，如包和项目依赖

快速导航 Quick Navigation

JSON files can get pretty large and we support quick navigation to properties

`kb(workbench.action.gotoSymbol)` (**Go to Symbol**) with the Command Palette.

JSON文件可以相当大。通过命令面板的(**Go to Symbol**)，我们支持快速导航到属性 `kb(workbench.action.gotoSymbol)`



悬浮 Hovers

When you hover over properties and values for JSON data with or without schema, we will provide additional context.

当你将鼠标悬停在JSON数据的属性和值时，我们将提供更多的上下文。无论JSON数据是否有模式（schema）。

```
1 // Place your settings in this file to overwrite the default settings
2 {   Controls if the editor should automatically format the line after typing
3     "editor.formatOnType": false
4 }
```

格式化 Formatting

You can format your JSON document (or just a part of it) using `kb(editor.action.format)` or **Format** from the context menu.

使用 `kb(editor.action.format)` 或上下文菜单的**Format**，你能够格式化你的JSON文档。

JSON模式和设置 JSON Schemas & Settings

To understand the structure of JSON files, we use [JSON schemas](#). JSON schemas describe the shape of the JSON file, as well as value sets, default values, and descriptions.

要了解的 JSON 文件结构，我们使用了 [JSON模式](#)。JSON模式描述了 JSON 文件的构成，以及值集、默认值和说明。

Servers like [JSON Schema Store](#) provide schemas for most of the common JSON based configuration files. However, schemas can also be defined in a file in the VS Code workspace, as well as the VS Code settings files.

像 [JSON模式存储](#) 服务，提供最常见的基于配置文件的JSON模式。尽管如此，模式也可以在 VSCode 工作区的文件，以及在VS Code 设置文件中被定义。

The association of a JSON file to a schema can be done either in the JSON file itself using the `$schema` attribute, or in the User or Workspace [Settings](#) ([File > Preferences > User Settings](#) or [Workspace Settings](#)) under the property `json.schemas`.

一个JSON文件可以关联到一个模式。通过该JSON文件本身使用相应 `$schema` 属性，或者通过用户或工作区 [设置](#)(文件 > 首选项 > 用户设置 或 工作区设置)下的属性 `json.schemas`。

VS Code extensions can also define schemas and schema mapping. That's why VS Code already knows about the schema of some well known JSON files such as `package.json`, `bower.json` and `tsconfig.json`.

VS Code 扩展也可以定义模式到模式的映射。这就是为什么VS Code已经知道了一些知名的 JSON文件，如 `package.json`，`bower.json` 和 `tsconfig.json` 的模式。

在JSON中映射 Mapping in the JSON

In the following example, the JSON file specifies that its contents follow the [CoffeeLint](#) schema.

下面的例子是，指定一个JSON文件的内容遵循[CoffeeLint](#) 模式

```
{
  "$schema": "http://json.schemastore.org/coffeelint",
  "line_endings": "unix"
}
```

在用户设置中映射 Mapping in the User Settings

The following excerpt from the User Settings shows how `.babelrc` files are mapped to the `babelrc` schema located on <http://json.schemastore.org/babelrc>.

下面，从用户设置中摘录的内容显示如何将 `.babelrc` 文件映射到位于 <http://json.schemastore.org/babelrc> 的 `babelrc` 的模式。

```
"json.schemas": [
  {
    "fileMatch": [
      "./.babelrc"
    ],
    "url": "http://json.schemastore.org/babelrc"
  },
  ...
]
```

Tip: Additionally to defining a schema for `.babelrc`, also make sure that `.babelrc` is associated to the JSON language mode. This is also done in the settings using the `files.association` array setting.

小贴士: 此外，以定义一个模式 `.babelrc`，也要确保 `.babelrc` 关联到JSON语言。这也 可以使用 `files.associative` 数组设置的设置来完成。

Tip: For an overview on settings, see [User and Workspace Settings](#).

小贴士: 有关设置的概述，参考[用户和工作区设置](#)。

在工作区中映射到模式 Mapping to a Schema in the Workspace

To map a schema that is located in the workspace, use a relative path. In this example, a file in the workspace root called `myschema.json` will be used as the schema for all files ending with `.foo.json`.

要映射位于工作区中的模式，请使用相对路径。在这个例子中，位于工作空间中的根的 `myschema.json` 文件，被用来当作所有 `.foo.json` 结尾的文件的模式。

```
"json.schemas": [
  {
    "fileMatch": [
      "/*.foo.json"
    ],
    "url": "./myschema.json"
  },
}
```

在设置中定义映射到模式 Mapping to a Schema Defined in Settings

To map a schema that is defined in the User or Workspace Settings, use the `schema` property. In this example, a schema is defined that will be used for all files named `.myconfig`.

要映射在用户或工作区设置中定义一个模式，使用 `schema` 属性。在这个例子中，定义一个模式用于所有 `.myconfig` 的文件。

```
"json.schemas": [
  {
    "fileMatch": [
      "./.myconfig"
    ],
    "schema": {
      "type": "object",
      "properties": {
        "name" : {
          "type": "string",
          "description": "The name of the entry"
        }
      }
    }
  },
}
```

在扩展中映射一个模式 Mapping a Schema in an Extension

Schemas and schema associations can also be defined by an extension. Check out the [jsonValidation contribution point](#).

模式和模式关联也可以通过一个扩展来定义，查看[jsonValidation contribution point](#).

下一步 Next Steps

Read on to find out about:

- [Customization](#) - Customize VS Code to work the way you want

请仔细阅读，了解：

- [定制](#) - 根据你所希望的工作方式，自定义 VS Code

VS Code对HTML的相关 HTML Programming in VS Code

当你在VS Code中编辑HTML文件时，你将会获得所有应有的支持及一些更多的小玩意。

When editing HTML files in Visual Studio Code you get all of the basics and a few more things :)

智能感知 IntelliSense

当你在编写HTML的时候，VS Code会提供一些智能感知的建议。下图中你可以看到一个包括对`</div>`标签闭合的基于当前语境的建议列表。

As you type in HTML, we offer suggestions via HTML IntelliSense. In the image below you can see a suggested HTML element closure `</div>` as well as a context specific list of suggested elements.

```

3   <div class="container body-content docs blog">
4     <div class="row">
5
6   <!-->
7   <!--> /div
8   <!--> a
9     If the a element has an href attribute, then it represents a hyperlink (a h...
10    <!--> abbr
11    <!--> address

```

VS Code同样提供对于元素，标签，一些值（在HTML5中定义的），Ionic和AngularJS中标签的支持。任何时候只要按下 `kb(editor.action.triggerSuggest)` 就可以了。

We also offer up suggestions for elements, tags, some values (as defined in HTML 5), Ionic and AngularJS tags. You can trigger suggestions at any time by pressing
`kb(editor.action.triggerSuggest)` .

HTML格式化 Format HTML

为了改进你的HTML代码的格式，可以按 `kb(editor.action.format)` 并选中区域来进行重新格式化。

小贴士：对于HTML格式化的设置详见[用户与工作空间](#).

To improve the formatting of your HTML code press `kb(editor.action.format)` and the selected area will be reformatted.

Tip: Configure the HTML formatter settings in the [User and Workspace Settings](#).

支持Emmet Emmet snippets

我们提供Emmet代码拓展，只要按 `kb(editor.emmet.action.expandAbbreviation)` 就可以了。

```
demo.html C:\myhtml
17
18 <!-- Type Emmet syntax and then press Tab -->
19 |
20
21
22
23
24
```

小贴士:Emmet语法详见 [Emmet cheat sheet](#)

我们同样支持 [用户定义代码段](#).

We support Emmet snippet expansion, simply press

`kb(editor.emmet.action.expandAbbreviation)` .

```
demo.html C:\myhtml
17
18 <!-- Type Emmet syntax and then press Tab -->
19 |
20
21
22
23
24
```

Tip: See the HTML section of the [Emmet cheat sheet](#) for valid abbreviations.

We also support [User Defined Snippets](#).

接下来 Next Steps

看看关于

- [CSS, Less 和 Sass](#) - VS Code对CSS（包括Less和Sass）有着顶级的支持。

Read on to find out about:

- [CSS, Less and Sass](#) - VS Code has first class support for CSS including Less and Sass.

CSS, Sass and Less

VS Code 内置对样式表编辑的支持，对于 CSS `.css`, Sass `.scss` 和 Less `.less`。这些支持包括：

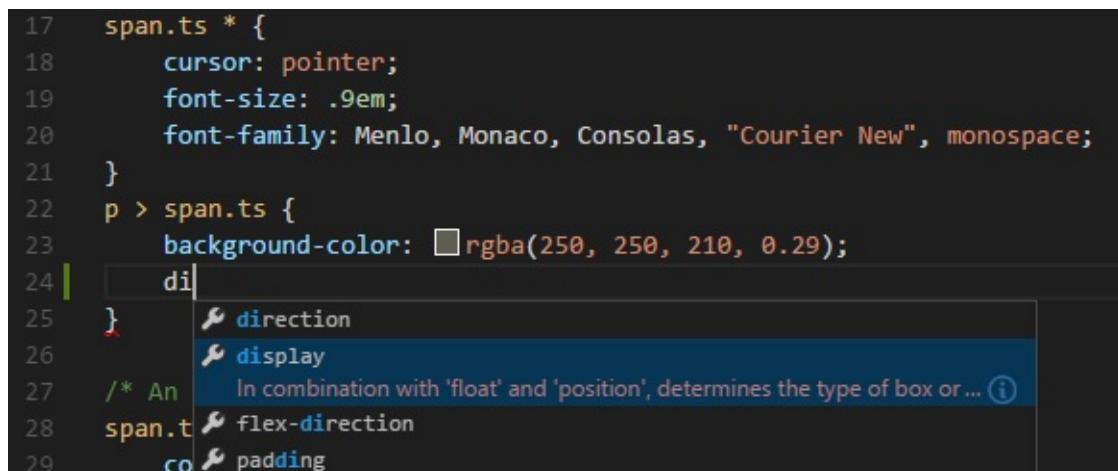
Visual Studio Code has built-in support for editing style sheets in CSS `.css`, Sass `.scss` and Less `.less`. This support includes:

智能感知 IntelliSense

我们提供对选择器，属性以及值的支持。按 `kb(editor.action.triggerSuggest)` 会得到一个具体列表。

We have support for selectors, properties and values. Use

`kb(editor.action.triggerSuggest)` to get a list of context specific options.



A screenshot of the Visual Studio Code editor showing a CSS file. The cursor is at the end of the word 'di' in a selector. A dropdown menu shows suggestions for CSS properties: 'direction', 'display', 'flex-direction', and 'padding'. The 'display' option is highlighted. The code in the file is as follows:

```

17   span.ts * {
18     cursor: pointer;
19     font-size: .9em;
20     font-family: Menlo, Monaco, Consolas, "Courier New", monospace;
21   }
22 p > span.ts {
23   background-color: #rgba(250, 250, 210, 0.29);
24   di| ...
25 } ...
26 /* An ...
27 span.t ...
28 co ...
29

```

建议中包括大量的文档，其中包括一个对浏览器兼容性的建议列表。如果对选中项想看完整描述按 `kb(toggleSuggestionDetails)`（哎哎为什么我的快捷键不好用。。2016-5-14）。

Proposals contain extensive documentation, including a list of browsers that support the property. To see the full description text of the selected entry, use

`kb(toggleSuggestionDetails)`.

Emmet语法 Emmet snippets

按 `kb(editor.emmet.action.expandAbbreviation)` 拓展当前缩写

小贴士：Emmet对CSS的支持详见 [Emmet cheat sheet](#).

我们同样支持 [用户定义代码段](#).

Press `kb(editor.emmet.action.expandAbbreviation)` to expand the current abbreviation.

Tip: See the CSS section of the [Emmet cheat sheet](#) for valid abbreviations.

We also support [User Defined Snippets](#).

语法高亮及颜色浏览 **Syntax coloring & Color preview**

当你编辑CSS样式时，VS Code提供语法高亮以及对你所设置的颜色浏览。

As you type, we provide syntax highlighting as well as in context preview of colors.

```

34  /* A reference to a type */
35  span.ts span.type-ref {
36      color: #rgb(175, 0, 219) !important;
37  }
38
39  /* Signature details */
40  div.signature > table {
41      border-collapse: collapse;
42      border: thin #darkgray solid;
43      width: 60%;
44  }

```

语法验证 **Syntax Verification & Linting**

该功能对 CSS 版本 \leq 2.1, Sass 版本 \leq 3.2 and Less 版本 \leq 1.7有效。

注意: 你可以在通过在用户定义代码段 [设置](#)来禁用VS Code对于CSS,Sass或者Less的默认语法验证。

```
"css.validate": false
```

We support CSS version \leq 2.1, Sass version \leq 3.2 and Less version \leq 1.7.

Note: You can disable VS Code's default CSS, Sass or Less validation by setting the corresponding `.validate` User or Workspace [setting](#) to false.

```
"css.validate": false
```

转到文件的符号信息 **Goto symbol in file**

按 `kb(workbench.action.gotoSymbol)` 即可。

Simply press `kb(workbench.action.gotoSymbol)`.

Hovers

鼠标停留在选择器上将会得到一个通过CSS规则匹配的HTML代码片段。

Hovering over a selector or property will provide an HTML snippet that is matched by the CSS rule.

```

45
46  <div class="signature">
47    <table>
48      ...
49      <th>
50  div.signature > table th {
51    background-color: lightgrey;
52    text-align: left;
53  }

```

声明跳转及引用查找 **Goto Declaration and Find References**

在同一个文件里支持Sass和Less的变量实现此功能。

注意:对于跨文件的引用 ('imports') 是无效的.

This is supported for Sass and Less variables in the same file.

Note: Cross file references ('imports') are not resolved.

将Sass和Less编译成CSS **Transpiling Sass and Less into CSS**

VS Code可以通过自带的[task runner](#)将Sass和Less编译为CSS。我们可以用这个功能将 `.scss` 或者 `.less` 文件编译为 `.css`，这个功能对一个简单的Sass/Less文件有效。

VS Code can integrate with Sass and Less transpilers through our integrated [task runner](#). We can use this to transpile `.scss` or `.less` files into `.css` files. Let's walk through transpiling a simple Sass/Less file.

第一步：安装一个**Sass**或者**Less**的编译器。Step 1: Install a **Sass or Less transpiler**

对此，可以使用[node-sass](#) 或者 [less](#) 这两个Node.js 的模块。

注意: 如果你没有 [Node.js](#) 以及没有安装 [NPM](#) 包管理器, 你得装。在你的系统安装 [Node.js](#). Node Package Manager (NPM) 会随着Node.js安装. 你只要打开一个新的命令行终端输入

```
npm install -g node-sass less
```

For this walkthrough, let's use either the [node-sass](#) or [less](#) Node.js module.

Note: If you don't have [Node.js](#) and the [NPM](#) package manager already installed, you'll need to do so for this walkthrough. [Install Node.js for your platform](#). The Node Package Manager (NPM) is included in the Node.js distribution. You'll need to open a new terminal (command prompt) for `npm` to be on your PATH.

```
npm install -g node-sass less
```

第二步：新建一个简单的**Sass**或者**Less**文件。 Step 2: Create a simple **Sass or Less** file

在VS Code打开一个空的文件夹并且创建 `styles.scss` 或者 `styles.less` 文件. 然后把下面的代码放入文件。

(如果是less文件，把 `$padding` 改成 `@padding` 即可。)

```
$padding: 6px;

nav {
  ul {
    margin: 0;
    padding: $padding;
    list-style: none;
  }

  li { display: inline-block; }

  a {
    display: block;
    padding: $padding 12px;
    text-decoration: none;
  }
}
```

注意:这是个非常简单的例子。这就是为什么源代码在两种文件类型中几乎是相同的。

Open VS Code on an empty folder and create a `styles.scss` or `styles.less` file. Place the following code in that file:

```
$padding: 6px;

nav {
    ul {
        margin: 0;
        padding: $padding;
        list-style: none;
    }

    li { display: inline-block; }

    a {
        display: block;
        padding: $padding 12px;
        text-decoration: none;
    }
}
```

For the Less version of the above file, just change `$padding` to `@padding`.

Note: This is a very simple example, which is why the source code is almost identical between both file types. In more advanced scenarios, the syntaxes and constructs will be much different.

第三步：创建**Create tasks.json Step 3: Create tasks.json**

下一步是进行任务配置。按 `kb(workbench.action.showCommands)` 打开命令面板。按Enter选择“任务：配置任务运行程序”，在选择对话框中，选择“其他(Others)”。

这将会在工作空间的 `.vscode` 文件夹创建一个 `tasks.json` 示例文件。这个文件包含了一个可以执行任意命令的样例。我们可以简单修改下编译配置。

```
// Sass configuration
{
    "version": "0.1.0",
    "command": "node-sass",
    "isShellCommand": true,
    "args": ["styles.scss", "styles.css"]
}
```

```
// Less configuration
{
  "version": "0.1.0",
  "command": "lessc",
  "isShellCommand": true,
  "args": ["styles.less", "styles.css"]
}
```

VS Code 将 `node-sass` or `lessc` 当作一个外部任务处理器：将 Sass/Less files 编译为 CSS 文件。我们需要的执行命令就是 `node-sass styles.scss styles.css` 或者 `lessc styles.less styles.css`。

The next step is to set up the task configuration. To do this open the **Command Palette** with `kb(workbench.action.showCommands)` and type in **Configure Task Runner**, press `kbstyle(Enter)` to select it. In the selection dialog that shows up, select `Others`.

This will create a sample `tasks.json` file in the workspace `.vscode` folder. The initial version of file has an example to run an arbitrary command. We will simply modify that configuration for transpiling Less/Sass instead:

```
// Sass configuration
{
  "version": "0.1.0",
  "command": "node-sass",
  "isShellCommand": true,
  "args": ["styles.scss", "styles.css"]
}
```

```
// Less configuration
{
  "version": "0.1.0",
  "command": "lessc",
  "isShellCommand": true,
  "args": ["styles.less", "styles.css"]
}
```

VS Code interprets `node-sass` or `lessc` as an external task runner exposing exactly one task: the transpiling of Sass/Less files into CSS files. The command we run is `node-sass styles.scss styles.css` or `lessc styles.less styles.css`.

第四步：编译 Step 4: Run the Build Task

因为这个文件只有这一个任务，你可以直接按 `kb(workbench.action.tasks.build)` (**Run Build Task**) 来编译。样例文件不应该出现任何编译问题，所以只会生成相应的 `styles.css`。As this is the only task in the file, you can execute it by simply pressing

`kb(workbench.action.tasks.build)` (**Run Build Task**)。The sample Sass/Less file should not have any compile problems, so by running the task all that happens is a corresponding `styles.css` file is created.

自动化Sass/Less编译 Automating Sass/Less compilation

你也可以试试往前再走一步——用VS Code进行自动化Sass/Less编译。我们用和以前相同的任务执行（task runner），但是要做些修改。

Let's take things a little further and automate Sass/Less compilation with VS Code. We can do so with the same task runner integration as before, but with a few modifications.

第一步：安装Gulp和一些插件。 Step 1: Install Gulp and some plug-ins

我们用[Gulp](#)来创建任务使Sass/Less自动化编译。同样用到[gulp-sass](#)这个插件。Less的话用[gulp-less](#)这个插件。

先安装 `gulp` (不要用 `-g`)

```
npm install gulp gulp-sass gulp-less
```

注意: `gulp-sass` 和 `gulp-less` 是Gulp对于 `node-sass` 和 `lessc` 这两个我们以前用过的模块的插件。也有许多其他Gulp的Sass和Less插件可供选择，以及Grunt的插件。

We will use [Gulp](#) to create a task that will automate Sass/Less compilation. We will also use the [gulp-sass](#) plug-in to make things a little easier. The Less plug-in is [gulp-less](#).

We need to install `gulp` locally (no `-g` switch):

```
npm install gulp gulp-sass gulp-less
```

Note: `gulp-sass` and `gulp-less` are Gulp plug-ins for the `node-sass` and `lessc` modules we were using before. There are many other Gulp Sass and Less plug-ins you can use, as well as plug-ins for Grunt.

第二步：创建个简单的Gulp任务 Step 2: Create a simple Gulp task

在VS Code打开之前的文件夹(包含 `styles.scss` / `styles.less` 和在 `.vscode` 文件夹下的 `tasks.json`)并且在根目录下创建 `gulpfile.js`。然后把下面的代码复制进去。

```
// Sass configuration
var gulp = require('gulp');
var sass = require('gulp-sass');

gulp.task('sass', function() {
  gulp.src('*.*scss')
    .pipe(sass())
    .pipe(gulp.dest(function(f) {
      return f.base;
    }))
});

gulp.task('default', ['sass'], function() {
  gulp.watch('*.*scss', ['sass']);
})
```

```
// Less configuration
var gulp = require('gulp');
var less = require('gulp-less');

gulp.task('less', function() {
  gulp.src('*.*less')
    .pipe(less())
    .pipe(gulp.dest(function(f) {
      return f.base;
    }))
});

gulp.task('default', ['less'], function() {
  gulp.watch('*.*less', ['less']);
})
```

这样会发生什么呢？

1. 我们 `default` 的gulp任务在启动时首先运行过去的 `sass` 或者 `less` 任务。
2. 会监听到之后工作空间根目录下任何Sass/Less文件的变化。例如在VS Code打开当前文件夹。（留个坑回来填，这句话好像理解的不对。）
3. 一旦源文件有变动，就会用我们的编译器自动生成编译后的版本。例如 `gulp-sass` , `gulp-less` .
4. We now have a set of CSS files, each named respectively after their original Sass/Less file. We then put these files in the same directory.

Open VS Code on the same folder from before (contains `styles.scss` / `styles.less` and `tasks.json` under the `.vscode` folder), and create `gulpfile.js` at the root.

Place the following code in the `gulpfile.js` file:

```
// Sass configuration
var gulp = require('gulp');
var sass = require('gulp-sass');

gulp.task('sass', function() {
  gulp.src('*.*scss')
    .pipe(sass())
    .pipe(gulp.dest(function(f) {
      return f.base;
    }));
});

gulp.task('default', ['sass'], function() {
  gulp.watch('*.*scss', ['sass']);
})
```

```
// Less configuration
var gulp = require('gulp');
var less = require('gulp-less');

gulp.task('less', function() {
  gulp.src('*.*less')
    .pipe(less())
    .pipe(gulp.dest(function(f) {
      return f.base;
    }));
});

gulp.task('default', ['less'], function() {
  gulp.watch('*.*less', ['less']);
})
```

What is happening here?

1. Our `default` gulp task first runs the `sass` or `less` task once when it starts up.
2. It then watches for changes to any Sass/Less file at the root of our workspace, for example the current folder open in VS Code.
3. It takes the set of Sass/Less files that have changed and runs them through our respective compiler, for example `gulp-sass`, `gulp-less`.
4. We now have a set of CSS files, each named respectively after their original Sass/Less file. We then put these files in the same directory.

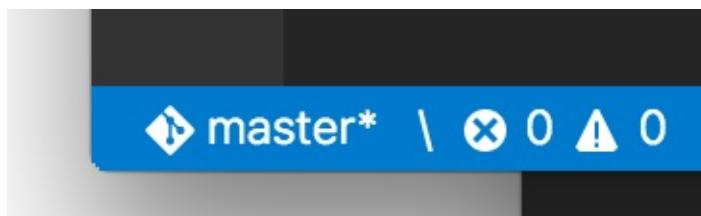
Step 3: Modify the configuration in tasks.json for watching

To complete the tasks integration with VS Code, we will need to modify the task configuration from before, to set a watch on the default Gulp task we just created. Your tasks configuration should now look like this:

```
{
  "version": "0.1.0",
  "command": "gulp",
  "isShellCommand": true,
  "tasks": [
    {
      "taskName": "default",
      "isBuildCommand": true,
      "showOutput": "always",
      "isWatching": true
    }
  ]
}
```

Step 4: Run the Build Task

Again, as this is the only task in the file you can execute it by simply pressing `kb(workbench.action.tasks.build)` (**Run Build Task**). But this time, we've set a watch so the Status Bar should indicate that on the left-hand side.



At this point, if you create and/or modify other Less/Sass files, you will see the respective CSS files generated and/or changes reflected on save. You can also enable [Auto Save](#) to make things even more streamlined.

If you want to stop the watch, you can press `kb(workbench.action.tasks.build)` again and click **Terminate Running Task** in the message box. Or you can use the **Command Palette** with `kb(workbench.action.showCommands)` and find the terminate command there.

Customizing CSS, Sass and Less Settings

You can configure the following lint warnings as [User and Workspace Settings](#).

The `validate` setting allows you turn off the built-in validation. You would do this if you rather use a different linter.

| Id | Description | Default |
|----------------------------|--|----------------|
| <code>css.validate</code> | Enables or disables all css validations | true |
| <code>less.validate</code> | Enables or disables all less validations | true |
| <code>sass.validate</code> | Enables or disables all sass validations | true |

To configure an option for CSS, use `css.lint.` as the prefix to the id; for Sass and Less, use `less.lint.` and `sass.lint.`.

Set a setting to `warning` or `error` if you want to enable lint checking, use `ignore` to disable it. Lint checks are performed as you type.

| Id | Description | Default |
|---------------------------------------|--|----------------|
| <code>validate</code> | Enables or disables all validations | true |
| <code>compatibleVendorPrefixes</code> | When using a property with a vendor-specific prefix (for example <code>-webkit-transition</code>), make sure to also include all other vendor-specific properties eg. <code>-moz-transition</code> , <code>-ms-transition</code> and <code>-o-transition</code> | ignore |
| <code>vendorPrefix</code> | When using a property with a vendor-specific prefix for example <code>-webkit-transition</code> , make sure to also include the standard property if it exists eg. <code>transition</code> | warning |
| <code>duplicateProperties</code> | Warn about duplicate properties in the same ruleset | ignore |
| <code>emptyRules</code> | Warn about empty rulesets | warning |
| <code>importStatement</code> | Warn about using an <code>import</code> statement as import statements are loaded sequentially which has a negative impact on web page performance | ignore |
| <code>boxModel</code> | Do not use <code>width</code> or <code>height</code> when using <code>padding</code> or <code>border</code> | ignore |
| <code>universalSelector</code> | Warn when using the universal selector <code>*</code> as it is known to be slow and should be avoided | ignore |
| <code>zeroUnits</code> | Warn when having zero with a unit e.g. <code>0em</code> as zero does not need a unit. | ignore |

| | | |
|---------------------------------|--|---------|
| fontFaceProperties | Warn when using <code>@font-face</code> rule without defining a <code>src</code> and <code>font-family</code> property | warning |
| hexColorLength | Warn when using hex numbers that don't consist of three or six hex numbers | error |
| argumentsInColorFunction | Warn when an invalid number of parameters in color functions e.g. <code>rgb</code> | error |
| unknownProperties | Warn when using an unknown property | warning |
| ieHack | Warn when using an IE hack <code>*propertyName</code> or <code>_propertyName</code> | ignore |
| unknownVendorSpecificProperties | Warn when using an unknown vendor-specific property | ignore |
| propertyIgnoredDueToDisplay | Warn when using a property that is ignored due to the display. For example with <code>display: inline</code> , the <code>width</code> , <code>height</code> , <code>margin-top</code> , <code>margin-bottom</code> , and <code>float</code> properties have no effect. | warning |
| important | Warn when using <code>!important</code> as it is an indication that the specificity of the entire CSS has gotten out of control and needs to be refactored. | ignore |
| float | Warn when using <code>float</code> as floats lead to fragile CSS that is easy to break if one aspect of the layout changes. | ignore |
| idSelector | Warn when using selectors for an id <code>#id</code> as selectors should not contain IDs because these rules are too tightly coupled with the HTML. | ignore |

接下来... Next Steps

看看关于...

Read on to find out about:

- 任务：配置运行程序 - 深入挖掘这个能帮助你将Sass和Less编译成CSS。
- 提升编辑体验 - 了解编辑器对语言提供的丰富的功能集，正如CSS。
- HTML - 除了CSS，HTML在VS Code中同样被很好的支持。

- [Configure Tasks](#) - Dig into Tasks to help you transpile your Sass and Less to CSS.
- [Editing Evolved](#) - Find out about the rich set of features the editor offers for languages such as CSS.
- [HTML](#) - CSS is just the start, HTML is also very well supported in VS Code.

常见问题解答 **Common Questions**

Q: VS Code 提供拾色器吗

A: 不提供。

Q: VS Code 支持基于Sass的缩进语法吗？

A: 不支持。

Q: Do you provide a color selector?

A: No, this is currently not supported.

Q: Do you support the indentation based Sass syntax (.sass) ?

A: No, not yet.

Editing TypeScript

TypeScript is a typed superset of JavaScript that compiles to plain JavaScript. It offers classes, modules, and interfaces to help you build robust components. A language specification can be found [here](#).

TypeScript是JavaScript的超集，它编译出的文件也是JavaScript。它支持类，模块，接口，更易于构建组件。说明文档地址：[Github](#)。

VS Code's TypeScript support can operate in two different modes:

VS Code's TypeScript 支持2种操作模式

- **File Scope:** in this mode TypeScript files opened in Visual Studio Code are treated as independent units. As long as a file `a.ts` doesn't reference a file `b.ts` explicitly (either using [/// reference directives](#) or external modules) there is no common project context between the two files.
- 文件域：在VSCode中，文件浏览模式的TypeScript文件是以单个文件的形式存在的，所以不会存在 `a.ts` 引用 `b.ts` 的情况。
- **Explicit Project:** a TypeScript project is defined via a `tsconfig.json` file. The presence of such a file in a directory indicates that the directory is the root of a TypeScript project. The file itself lists the files belonging to the project as well as compiler options. Details about the `tsconfig.json` file can be found [here](#).
- 明确项目：通过一个 `tsconfig.json` 文件定义的TypeScript项目，这个文件所在的目录就是项目的根目录。这个文件列举出了项目的所有文件和编译选项。关于 `tsconfig.json` 的详细信息可以[在这里](#)看到。

Tip: We recommend that you use explicit projects over file scope projects. Since explicit projects list the files belonging to a project language, features like `Find All References` `kb(editor.action.referenceSearch.trigger)` consider the project scope and not the file scope only.

提示：我们更推荐大家使用项目而不是文件，因为明且的项目将会地处属于这个项目的语言文件，就像 `Find All References` `kb(editor.action.referenceSearch.trigger)` 将会考虑项目的范围而不仅仅是文件范围。

tsconfig.json

Typically the first step in any new TypeScript project is to add in a `tsconfig.json` file. This defines the TypeScript project settings such as the compiler options and the files that should be included. To do this, open up the folder where you want to store your source and add in a new file named `tsconfig.json`. Once in this file IntelliSense will help you along the way.

首先我们都会为每个新的TypeScript项目添加一个 `tsconfig.json` 文件，它定义了项目的配置，比如编译选项和应该包含的文件，为此，打开你想储存源文件的文件夹并添加一个名为 `tsconfig.json` 文件。只要在这个文件智能感应将会一直帮助你。

`tsconfig.json`

```
1 {
2   compilerOptions
3     Instructs the TypeScript compiler how to compile .ts files
4   files
```

A simple `tsconfig.json` looks like this for ES5, **CommonJS modules** and source maps:

一个简单 `ES5 tsconfig.json` 应该像这样，包含了**CommonJS 模块** 和 source maps：

```
{
  "compilerOptions": {
    "target": "es5",
    "module": "commonjs",
    "sourceMap": true
  }
}
```

Now when you create a `.ts` file as part of the project we will offer up rich editing experiences and syntax validation.

当你创建项目的一个 `.ts` 文件，我们将会提供给你丰富的编辑体验和语法检查。

Transpiling TypeScript into JavaScript

将TypeScript转化为JavaScript

VS Code integrates with `tsc` through our integrated **task runner**. We can use this to transpile `.ts` files into `.js` files. Let's walk through transpiling a simple TypeScript Hello World program.

通过我们集成的**task runner**来集成VSCode和 `tsc` 。让我们通过Hello World程序来了解下转化过程。

Step 1: Create a simple TS file

第一步：创建一个TS文件

Open VS Code on an empty folder and create a `HelloWorld.ts` file, place the following code in that file...

打开VSCode，在一个空目录下创建一个 `HelloWorld.ts` 文件，在文件内加入以下代码...

```
class Startup {
    public static main(): number {
        console.log('Hello World');
        return 0;
    }
}

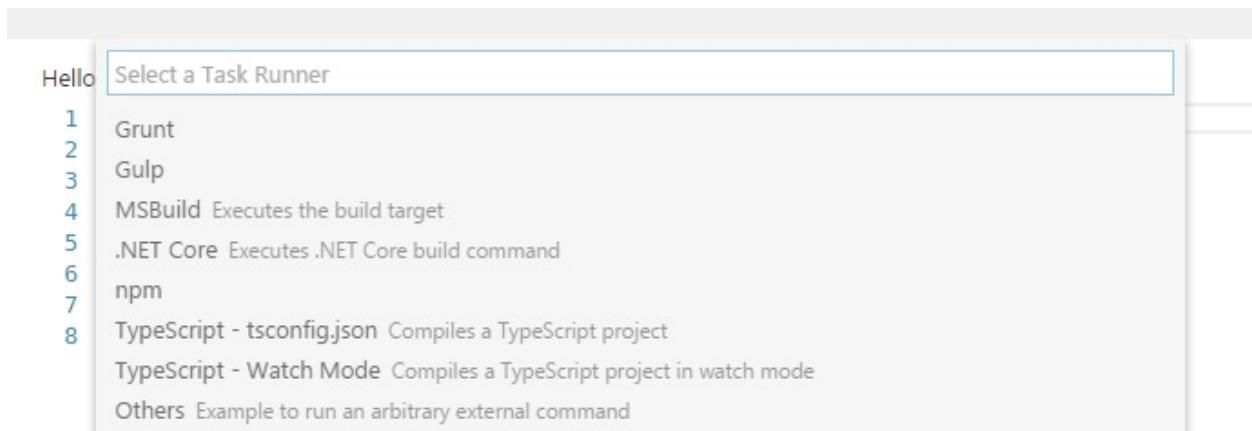
Startup.main();
```

Step 2: Create tasks.json

第二步：创建task.json文件

The next step is to set up the task configuration. To do this open the **Command Palette** with `kb(workbench.action.showCommands)` and type in **Configure Task Runner**, press `kbstyle(Enter)` to select it. This shows a selection box with templates you can choose from:

下一步就是创建任务配置。首先要用 `kb(workbench.action.showCommands)` 打开**Command Palette** 写入**Configure Task Runner**，按下 `kbstyle(Enter)` 选择。这里显示了你可以选择的模板复选框。



Select `TypeScript - tsconfig.json`. This will create a `tasks.json` file in the workspace `.vscode` folder.

选择 `TypeScript - tsconfig.json`。这将会在工作空间的 `.vscode` 目录下创建一个 `task.json` 文件。

The content of the tasks.json file looks like this:

task.json文件的内容像这样：

```
{
    // See http://go.microsoft.com/fwlink/?LinkId=733558
    // for the documentation about the tasks.json format
    "version": "0.1.0",
    "command": "tsc",
    "isShellCommand": true,
    "args": ["-p", "."],
    "showOutput": "silent",
    "problemMatcher": "$tsc"
}
```

Tip: While the template is there to help with common configuration settings, IntelliSense is available for the `tasks.json` file as well to help you along. Use

`kb(editor.action.triggerSuggest)` to see the available settings.

提示：这个模版有助于常用配置的设置，task.json的智能感应可以很好的帮到你，使用 `kb(editor.action.triggerSuggest)` 去查看可用配置。

Under the covers we interpret `tsc` as an external task runner exposing exactly one task: the compiling of TypeScript files into JavaScript files. The command we run is: `tsc -p .`

在这里，我们的解读 `tsc` 是一个外部的task runner执行了这样的任务：将TypeScript文件编译为JavaScript文件。执行的命令为：`tsc -p .`

Tip: If you don't have the TypeScript compiler installed, you can [get it here](#).

提示：如果你没有安装TypeScript编译器，你可以点击[这里](#)。

Step 3: Run the Build Task

第三步：执行构建任务

As this is the only task in the file, you can execute it by simply pressing

`kb(workbench.action.tasks.build)` (**Run Build Task**). At this point you will see an additional file show up in the file list `HelloWorld.js`.

因为这是在文件的唯一任务，你可以直接按下 `kb(workbench.action.tasks.build)` (**Run Build Task**)执行。这时候你将会看到文件列表额外多了一个 `HelloWorld.js` 文件。

The example TypeScript file did not have any compile problems, so by running the task all that happened was a corresponding `HelloWorld.js` and `HelloWorld.js.map` file was created.

TypeScript的示例文件不会有任何的编译问题，所以运行该任务将会生成一个相应的 `HelloWorld.js` 并创建一个 `HelloWorld.js.map` 文件。

If you have [Node.js](#) installed, you can run your simple Hello World example by opening up a terminal and running:

如果你安装了 [Node.js](#)，你可以打开一个的终端并执行如下命令来运行你的Hello World示例：

```
node HelloWorld.js
```

Tips You can also run the program using VS Code's Run/Debug feature. Details about running and debugging node apps in VS Code can be found [here](#)

提示：你也可以使用VS Code's的执行/调试功能运行代码。[这里可以看到在VS Code执行和调试node应用的细节。](#)

Step 4: Reviewing Build Issues

第四步：检查构建问题

Unfortunately, most builds don't go that smoothly and the result is often some additional information. For instance, if there was a simple error in our TypeScript file, we may get the following output from `tsc` :

不幸的是，大多情况下构建都不会太顺利并且结果将会返回额外的信息。比如，如果一个简单的错误在我们的TypeScript文件里面，`tsc` 可能会使我们得到下面的输出：

```
HelloWorld.ts(3,17): error TS2339: Property 'logg' does not exist on type 'Console'.
```

This would show up in the output window (which can be opened using `kb(workbench.action.output.toggleOutput)`) and selecting Tasks in the output view dropdown. We parse this output for you and highlight detected problems in the Status Bar.

信息将会在输出窗口显示出来(它可以用 `kb(workbench.action.output.toggleOutput)` 打开)还可以在输出视口的下拉列表内选择任务，我们会将输出解析并在状态栏内高亮检测出的问题。



You can click on that icon to get a list of the problems and navigate to them.

你可以通过点击图标得到问题列表并导航到他们。

```

Hello !
1   [ts] Property 'logg' does not exist on type 'Console'.
2   HelloWorld.ts(3,17)
3     return 0;
4   ...
5 }
6
8 Startup.main();

```

You can also use the keyboard to open the list `kb(workbench.action.showErrorsWarnings)`.

你也可以使用键盘打开问题列表 `kb(workbench.action.showErrorsWarnings)`

Tip: Tasks offer rich support for many actions. Check the [Tasks](#) topic for more information on how to configure them.

提示: Tasks提供了很多动作的支持，[Tasks](#) 将得到很多配置他们的信息。

Goto Symbol & Show All Symbols

跳转到符号 & 显示所有符号

`kb(workbench.action.gotoSymbol)` : lists all defined symbols of the current open TypeScript and lets you navigate in it.

`kb(workbench.action.gotoSymbol)` :列出了当前打开TypeScript的所有定义的符号并让能自动导航到那里。

`kb(workbench.action.showAllSymbols)` : lets you search all symbols defined in the current project or file scope. You need to have a TypeScript file open in the active editor.

`kb(workbench.action.showAllSymbols)` :让您可以搜索在当前项目或文件的范围内定义的所有符号。在活动的编辑区你需要有一个打开的TypeScript文件。

Format Code

格式代码

`kb(editor.action.format)` : formats the currently selected code, or the whole document if no code is selected.

`kb(editor.action.format)` : 格式当前选中的代码，如果没有代码选中将会格式化整个文档。

JSDoc Support

JSDoc 支持

VS Code offers **JSDoc** support for TypeScript. Besides syntax coloring, we help you enter **JSDoc** comments. Simply type `/**` and it will auto insert the closing `*/`. Pressing `kbstyle(Enter)` inside a **JSDoc** block will indent the next line and auto insert a `*`.

VS Code为TypeScript提供**JSDoc**支持。除了语法着色，还加入了**JSDoc**注释。只需要输入`/**`将会自动添加`*/`。在**JSDoc**块内按下`kbstyle(Enter)`将会在下一行缩进并且自动添加`*`。

JavaScript Source Map Support

JavaScript Source Map 支持

TypeScript debugging supports JavaScript source maps. Enable this by setting the `sourceMaps` attribute to `true` in the project's launch configuration file `launch.json`. In addition, you can specify a TypeScript file with the `program` attribute.

TypeScript 调试支持JavaScript source maps。开启方式是将配置文件 `launch.json` 的 `sourceMaps` 属性设置为 `true`。另外，你也可以指定TypeScript文件的 `program` 变量。

To generate source maps for your TypeScript files, compile with the `--sourcemap` option or set the `sourceMap` property in the `tsconfig.json` file to `true`.

要生成TypeScript文件的source maps，需要使用 `--sourcemap` 选项编译或者设置 `tsconfig.json` 文件的 `sourceMap` 属性为 `true`。

In-lined source maps (a source map where the content is stored as a data URL instead of a separate file) are also supported, although in-lined source is not yet supported.

In-lined source maps (一个内容存储在URL而不是以单独文件存储的source map)也被支持，虽然in-lined source还未被支持。

Setting a different outDir for generated files

设置为生成的文件设置不同的输出目录

If generated (transpiled) JavaScript files do not live next to their source, you can help the VS Code debugger locate them by specifying the `outDir` directory in the launch configuration. Whenever you set a breakpoint in the original source, VS Code tries to find the generated source, and the associated source map, in the `outDir` directory.

如果生成的JavaScript文件没有出现在源代码旁边，你可以通过VS Code调试器的launch配置中输出目录。无论什么时候你都可以在输出目录中生成的源代码或者关联的source map中打断点。

Hiding Derived JavaScript Files

隐藏派生的JavaScript Files

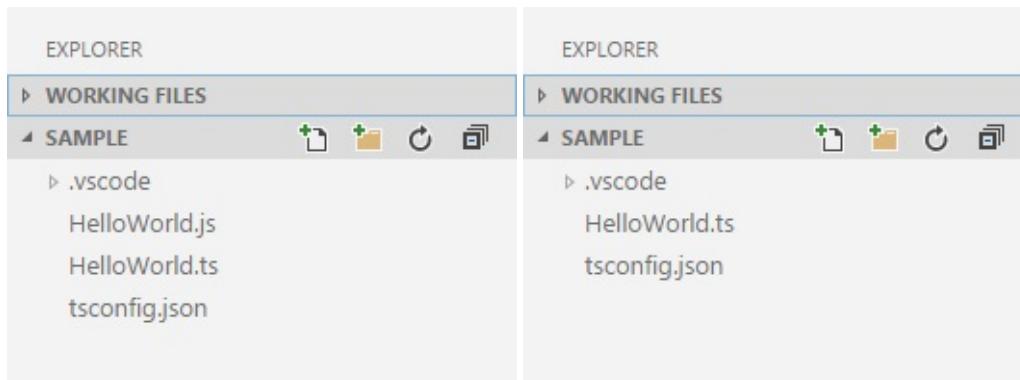
When you are working with TypeScript, you often don't want to see generated JavaScript files in the explorer or in search results. VS Code offers filtering capabilities with a `files.exclude` setting ([File > Preferences > Workspace Settings](#)) and you can easily create an expression to hide those derived files:

当你使用TypeScript工作的时候，常常不希望在资源管理器或者搜索结果中看到生成的JavaScript文件。VS Code通过`files.exclude` setting ([File > Preferences > Workspace Settings](#)) 提供了文件过滤功能，你可以轻松创建隐藏这些导出的文件的表达式。

```
"/**/*.js": { "when": "$(basename).ts"}
```

This pattern will match on any JavaScript file (`/**/*.js`) but only if a sibling TypeScript file with the same name is present. The file explorer will no longer show derived resources for JavaScript if they are compiled to the same location.

这个模式将会匹配所有的JavaScript文件(`/**/*.js`)但是只显示相同名字的兄弟TypeScript文件，如果他们被编译到一起文件资源管理器不会显示派生的JavaScript资源。



Mixed TypeScript and JavaScript projects

TypeScript和JavaScript混合项目

It is now possible to have mixed TypeScript and JavaScript projects. To enable JavaScript inside a TypeScript project, you can set the `allowJs` property to `true` in the `tsconfig.json`.

现在已经可能存在TypeScript和JavaScript混合的项目。要开启JavaScript在TypeScript项目，你可以在 `tsconfig.json` 中设置 `allowJs` 属性为 `true`

Tip: The `tsc` compiler does not detect the presence of a `jsconfig.json` file automatically. Use the `-p` argument to make `tsc` use your `jsconfig.json` file, e.g.
`tsc -p jsconfig.json`.

提示：`tsc` 不会自动检测 `jsconfig.json` 文件是否存在，使用 `tsc` 的 `-p` 参数去生成自己的 `jsconfig.json` 文件，如 `tsc -p jsconfig.json`。

Using Newer TypeScript Versions

使用新版本的TypeScript

VS Code ships with a recent stable version of TypeScript in the box. If you want to use a newer version of TypeScript, you can define the `typescript.tsdk` setting (**File > Preferences > User/Workspace Settings**) pointing to a directory containing the TypeScript `tsserver.js` and the corresponding `lib.*.d.ts` files. The directory path can be absolute or relative to the workspace directory. By using a relative path, you can easily share this workspace setting with your team and use the latest TypeScript version (`npm install typescript@next`). Refer to this [blog post](#) for more details on how to install the nightly builds of TypeScript.

VS Code 附带了最近稳定版本的TypeScript。如果你想要使用最近版本的TypeScript，你可以定义一个 `typescript.tsdk` 环境 (**File > Preferences > User/Workspace Settings**) 指定一个包含TypeScript `tsserver.js` 和对应的 `lib.*.d.ts` 文件的目录。

Next Steps

下一步

OK, read on to find out about:

好，继续阅读深入了解：

- [JavaScript](#) - we have several JavaScript specific features in VS Code
- [Tasks](#) - we used tasks to transpile your TS file. Read more to find out what else tasks can do
- [Editing Evolved](#) - dig into multi-cursor, snippets and more
- [Debugging](#) - we support debugging TypeScript Node.js apps

Common Questions

通常都会遇到的问题

Q: How do I resolve a TypeScript "Cannot compile external module" error?

问题：我怎么解决TypeScript "Cannot compile external module" 的错误？

A: If you get that error, resolve it by creating a `tsconfig.json` file in the root folder of your project. The `tsconfig.json` file lets you control how Visual Studio Code compiles your TypeScript code. For more information, see the [typescript.json overview](#).

回答 如果你遇到这个错误，可以通过在项目的根目录下创建一个 `tsconfig.json` 文件的方式解决。你可以用过 `tsconfig.json` 文件控制VS Code怎样编译你的TypeScript代码。获取更多信息，请参阅[typescript.json overview](#)。

Due to a current limitation, you must restart VS Code after adding the `tsconfig.json` file.

由于当前有局限性，在添加 `tsconfig.json` 后你必需重启VS Code。

Markdown and VS Code - Markdown与VS Code

Working with Markdown in Visual Studio Code can be pretty fun and there are a number of Markdown specific features that will help you be more productive.

利用VS Code写就Markdown文档将很有乐趣并享有许多的Markdown写作特性以增强你的生产力。

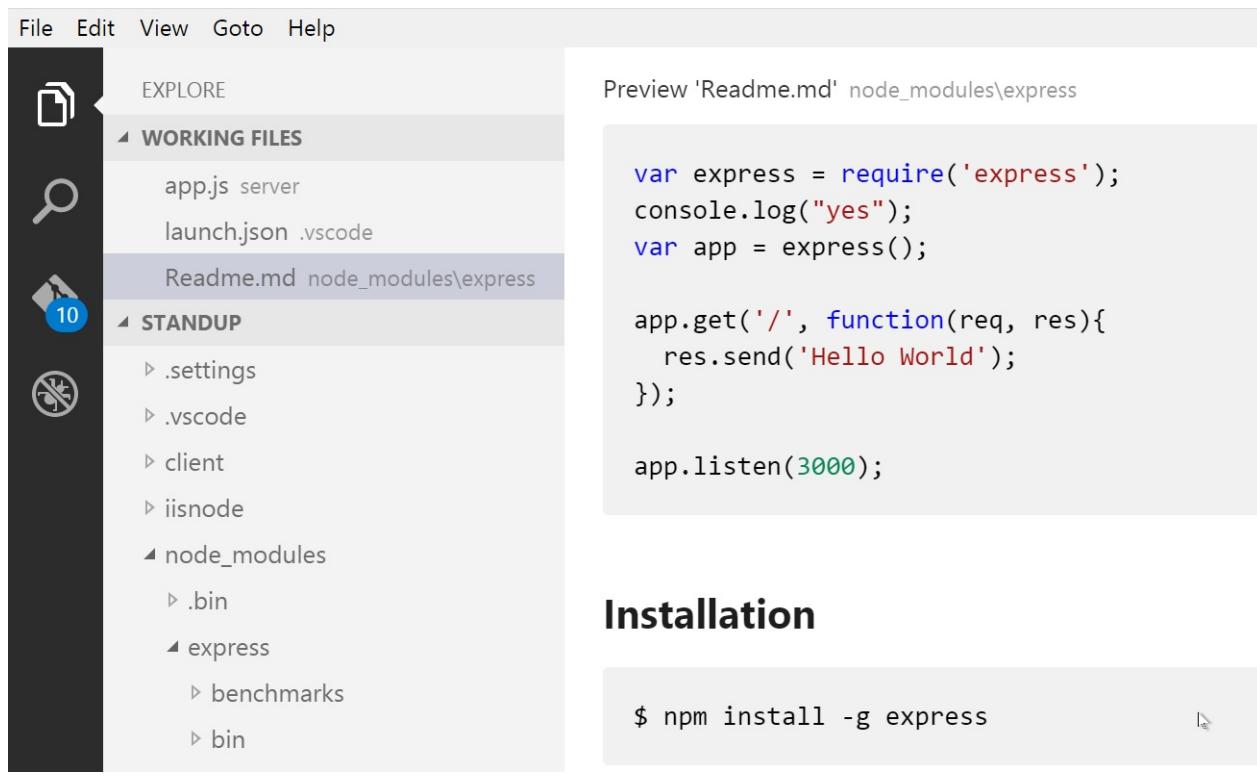
Markdown Preview - Markdown预览模式

VS Code supports Markdown files out of the box. You just start writing Markdown text, save the file with the .md extension and then you can toggle the visualization of the editor between the code and the preview of the Markdown file; obviously, you can also open an existing Markdown file and start working with it. To switch between views you just have to press `kb(workbench.action.markdown.togglePreview)` in the editor. You can view the preview side-by-side (`kb(workbench.action.markdown.openPreviewSideBySide)`) with the file you are editing and see changes reflected in real-time as you edit.

VS Code原生支持对于Markdown的写作，你只需运行VS Code并写作Markdown文档并保存为.md后缀，之后你将可切换Markdown文本的渲染模式，以脚本模式或可视化模式，显然你也可以从打开一个Markdown文本作为编辑的起点，仅需要按下 `kb(workbench.action.markdown.togglePreview)` 便可以切换渲染模式，同时你可以通过按下 `kb(workbench.action.markdown.openPreviewSideBySide)` 开启对比模式，在同屏中观察渲染结果和脚本，当然这两种操作均在编辑器内进行。

Here is an example with a very simple file.

这是一个简单的文件示例



Tip: You can also click on the icon on the top right of the preview window to switch back and forth between source and preview mode.

注：你也可以通过点击渲染窗口顶部右方的相应按键去切换脚本或渲染模式

Installation

```
$ npm install -g express
```



Using your own CSS - 使用你自己的CSS

By default, we use a CSS style for the preview that matches the style of VS Code. If you want to use your own CSS for the Markdown preview, update the `"markdown.styles": []` setting with the comma-separated list of URL(s) for your style sheet(s).

默认的，我们使用CSS风格的预览模式用以适应VS Code的风格。如果你想用自己的Markdown预览风格，请使用逗号分隔的URLs升级 `"markdown.styles": []` `setting`。

For instance, in the screen shot above we used a custom CSS to change the default font for the page and changed the color for the H1 title.

举个例子，在文中的预览截图即为应用了我们自己定制的CSS样式，其中修改了默认字体和H1标题的颜色

Here is the relevant CSS:

这是相关的实现用CSS：

```
body {  
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;  
}  
  
h1 {  
    color: cornflowerblue;  
}
```

Use **File > Preferences > Workspace Settings** to bring up the workspace `settings.json` file and make this update:

```
// Place your settings in this file to overwrite default and user settings.  
{  
    "markdown.styles": [  
        "Style.css"  
    ]  
}
```

Snippets for Markdown

There are several built-in Markdown snippets included in VS Code - simply press `kb(editor.action.triggerSuggest)` (Trigger Suggest) and we will give you a context specific list of suggestions.

Tip: You can add in your own User Defined Snippets for Markdown. Take a look at [User Defined Snippets](#) to find out how.

Compiling Markdown into HTML

VS Code can integrate with Markdown compilers through our integrated [task runner](#). We can use this to compile `.md` files into `.html` files. Let's walk through compiling a simple Markdown document.

Step 1: Install a Markdown compiler

For this walkthrough, we will use the popular [Node.js](#) module, [marked](#).

```
npm install -g marked
```

Note: There are many Markdown compilers to choose from beyond [marked](#), such as [markdown-it](#). Pick the one that best suits your needs and environment.

Step 2: Create a simple MD file

Open VS Code on an empty folder and create a `sample.md` file.

Note: You can open a folder with VS Code by either selecting the folder with **File > Open Folder...** or navigating to the folder and typing `code .` at the command line.

Place the following source code in that file:

```
Hello Markdown in VS Code!
=====
This is a simple introduction to compiling Markdown in VS Code.

Things you'll need:

* [node](https://nodejs.org)
* [marked](https://www.npmjs.com/package/marked)
* [tasks.json](./docs/editor/tasks.md)

## Section Title

> This block quote is here for your information.
```

Step 3: Create tasks.json

The next step is to set up the task configuration file `tasks.json`. To do this, open the **Command Palette** with `kb(workbench.action.showCommands)` and type in **Configure Task Runner**, press `kbstyle(Enter)` to select it.

It will present a list of possible `tasks.json` templates to choose from. Select `Others` since we want to run an external command.

This generate a `tasks.json` file in your workspace `.vscode` folder with the following content:

```
{
  // See http://go.microsoft.com/fwlink/?LinkId=733558
  // for the documentation about the tasks.json format
  "version": "0.1.0",
  "command": "echo",
  "isShellCommand": true,
  "args": ["Hello World"],
  "showOutput": "always"
}
```

Since we want to use **marked** to compile the Markdown file, we change the contents as follows:

```
{  
    // See http://go.microsoft.com/fwlink/?LinkId=733558  
    // for the documentation about the tasks.json format  
    "version": "0.1.0",  
    "command": "marked",  
    "isShellCommand": true,  
    "args": ["sample.md", "-o", "sample.html"],  
    "showOutput": "always"  
}
```

Tip: While the sample is there to help with common configuration settings, IntelliSense is available for the `tasks.json` file as well to help you along. Use `kb(editor.action.triggerSuggest)` to see the available settings.

Under the covers, we interpret **marked** as an external task runner exposing exactly one task: the compiling of Markdown files into HTML files. The command we run is `marked sample.md -o sample.html`.

Step 4: Run the Build Task

As this is the only task in the file, you can execute it by simply pressing `kb(workbench.action.tasks.build)` (**Run Build Task**). At this point, you should see an additional file show up in the file list `sample.html`.

The sample Markdown file did not have any compile problems, so by running the task all that happened was a corresponding `sample.html` file was created.

Automating Markdown compilation

Let's take things a little further and automate Markdown compilation with VS Code. We can do so with the same task runner integration as before, but with a few modifications.

Step 1: Install Gulp and some plug-ins

We will use [Gulp](#) to create a task that will automate Markdown compilation. We will also use the [gulp-markdown](#) plug-in to make things a little easier.

```
npm install -g gulp gulp-markdown
```

Note: gulp-markdown is a Gulp plug-in for the `marked` module we were using before. There are many other Gulp Markdown plug-ins you can use, as well as plug-ins for Grunt.

Step 2: Create a simple Gulp task

Open VS Code on the same folder from before (contains `sample.md` and `tasks.json` under the `.vscode` folder), and create `gulpfile.js` at the root.

Place the following source code in that file:

```
var gulp = require('gulp');
var markdown = require('gulp-markdown');

gulp.task('markdown', function() {
    return gulp.src('**/*.md')
        .pipe(markdown())
        .pipe(gulp.dest(function(f) {
            return f.base;
        }));
});

gulp.task('default', function() {
    gulp.watch('**/*.md', ['markdown']);
});
```

What is happening here?

1. We are watching for changes to any Markdown file in our workspace, i.e. the current folder open in VS Code.
2. We take the set of Markdown files that have changed, and run them through our Markdown compiler, i.e. `gulp-markdown`.
3. We now have a set of HTML files, each named respectively after their original Markdown file. We then put these files in the same directory.

Step 3: Modify the configuration in tasks.json for watching

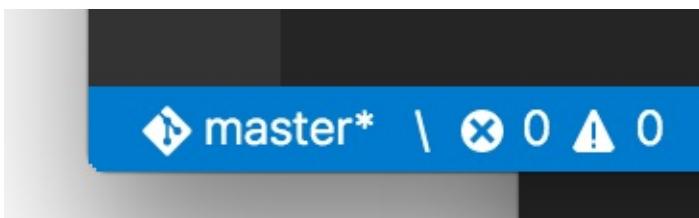
To complete the tasks integration with VS Code, we will need to modify the task configuration from before to set a watch on the default Gulp task we just created.

Your tasks configuration should now look like this:

```
{
  "version": "0.1.0",
  "command": "gulp",
  "isShellCommand": true,
  "tasks": [
    {
      "taskName": "default",
      "isBuildCommand": true,
      "showOutput": "always",
      "isWatching": true
    }
  ]
}
```

Step 4: Run the gulp Build Task

Again, as this is the only task in the file you can execute it by simply pressing `kb(workbench.action.tasks.build)` (**Run Build Task**). But this time, we've set a watch so the Status Bar should indicate that on the left-hand side.



At this point, if you create and/or modify other Markdown files, you will see the respective HTML files generated and/or changes reflected on save. You can also enable [Auto Save](#) to make things even more streamlined.

If you want to stop the watch, you can press `kb(workbench.action.tasks.build)` again and click **Terminate Running Task** in the message box. Or you can use the **Command Palette** with `kb(workbench.action.showCommands)` and find the terminate command there.

Next Steps

Read on to find out about:

- [Customization](#) - Dig into additional settings such as word wrap and User Defined Snippets.
- [CSS, Less and Sass](#) - Want to edit your CSS? VS Code has great support for CSS, Less and Sass editing.

Common Questions

Q: Is there spell checking?

A: Not in VS Code out of the box but there are [spell checking extensions](#). Be sure to check the [VS Code Marketplace](#) to look for useful extensions to help with your workflow.

Q: Does VS Code support GitHub Flavored Markdown?

A: We are using the [marked](#) library with the `gfm` option set to `true`.

Q: In the walkthrough above, I didn't find the Configure Task Runner command in the Command Palette?

A: You may have opened a file in VS Code rather than a folder. You can open a folder by either selecting the folder with **File > Open Folder...** or navigating to the folder and typing `code .` at the command line.

C/C++ for VS Code (预览)(Preview)

C/C++ support for Visual Studio Code is provided today as a preview of our work to enable cross-platform C and C++ development using VS Code on Windows, Linux, and OS X. Our focus in this preview release is code editing and navigation support for C and C++ code everywhere that VS Code runs, as well as debugging on Linux (Ubuntu 14.04 64-bit) and OS X (see *Known limitations* below).

今天作为我们工作的预览，C/C++ support for VS Code 被提供出来。它能够在Windows，Linux和OS X上跨平台开发C和C++。我们这个预览版的重点是代码编辑和在任何运行VS Code中的C 和 C++ 的代码导航，以及在Linux(Ubuntu 14.04 64-bit)和OS X下的调试（参见下文*Known limitations*）。

If you just want a lightweight tool to edit your C++ files VS Code has you covered wherever you are, but if you want the best possible experience for your existing Visual C++ projects or debugging on Windows, we recommend you use a version of Visual Studio such as [Visual Studio Community](#).

如果你只是一个轻量级的工具来编辑你的C++文件，无论你在哪里VSCode都已经被覆盖。但如果你想在Windows下调试您现有的Visual C ++项目，最佳的体验，我们推荐您使用[Visual Studio社区]版本的Visual Studio。

Because we're still shaping the C++ experience in VS Code, now is a great time to [provide bug reports, feature requests, and feedback](#), and for those of you who use Linux or OS X as your development environment to [get engaged](#) with the Visual Studio team.

因为在VS Code中，我们还在积累 C++ 的经验。现在是一个[提供错误报告，请求功能和反馈](#)的最好的时间。并请那些在使用Linux或OS X作为开发环境的人，[参与](#)Visual Studio团队的工作。

安装 Installing C++ Installing C++ support

C++ language support is an optional [install from the Marketplace](#). Or, just install it from VS Code by launching the **Quick Open** (`kb(workbench.action.quickOpen)`) and then entering the command **ext install cpptools**.

C++ 语言支持是一个可选安装，[从Marketplace](#)。或者，仅需要从VS Code启动**Quick Open** (`kb(workbench.action.quickOpen)`)然后输入名令**ext install cpptools**来安装它。

On Linux, there's an additional step that installs dependencies necessary for debugging support. When VS Code restarts after installing the extension, a script installs the [dotnet cli](#) dependency. Because elevated permissions are needed to install this package, you'll be prompted for your password in the terminal where the script is running. If you'd rather perform these last steps yourself, you can close the terminal now, then enter the commands yourself (these steps must be completed to enable debugging support.) For more information on these commands, see *Manual Installation for the C++ Debugger extension* in the [README](#).

在Linux上，为了支持调试，而必需额外的安装依赖。当VS Code安装扩展后重新启动后，脚本会安装[dotnet cli](#) 依赖。因为需要提升权限才能安装该软件包，你会被提示在脚本运行的终端输入密码。如果您愿意执行这些步骤，最后你可以关闭终端，然后输入你自己命令开始工作（为了启用调试支持，这些步骤必须完成。）。有关这些命令的详细信息，请参考手动安装C++调试扩展 中的[README](#).

On OS X, additional install steps need to be completed manually to enable debugging on OS X. See *Manual Installation for the C++ Debugger extension* in the [README](#).

在OS X上，对于在OS X上调试，额外的安装步骤需要手工完成。请参考手动安装C++调试扩展 中的[README](#).

Navigating code

Search for Symbols

You can search for symbols in the current file or workspace to navigate your code more quickly.

To search for a symbol in the current file, press `kb(workbench.action.gotoSymbol)`, then enter the name of the symbol you're looking for. A list of potential matches will appear and be filtered as you type. Choose from the list of matches to navigate to its location.

The screenshot shows a code editor interface with a dark theme. On the left, there is a file named "main.cpp". In the center, a code editor window displays the following C++ code:

```
1 #include <iostream>
2 #include <vector>
3 #include <string>
4 #include <memory>
5 #include <functional>
6 // ...
7 #include <asio.hpp>
8 #include <asio/strand.hpp>
9 #include <asio/asyncReadStream.hpp>
10 #include <asio/asyncWriteStream.hpp>
11 #include <asio/ssl.hpp>
12 using namespace web::http;
13 using namespace web::http::client;
14 using namespace concurrency::streams;
15 using namespace web;
16
17 enum class GuessResult
18 {
19     TooLow = -1,
20     Goldilocks = 0,
21     TooHigh = 1
22 };
23
24 const int MIN = 0;
25 const int MAX = 1;
26 class GuessGame
27 {
28 public:
29     GuessGame(std::string address, std::string userName, int maxNumber) :
30         guessServer(http::uri(address)),
31         userName(userName),
32         validatedRange {0, maxNumber}
```

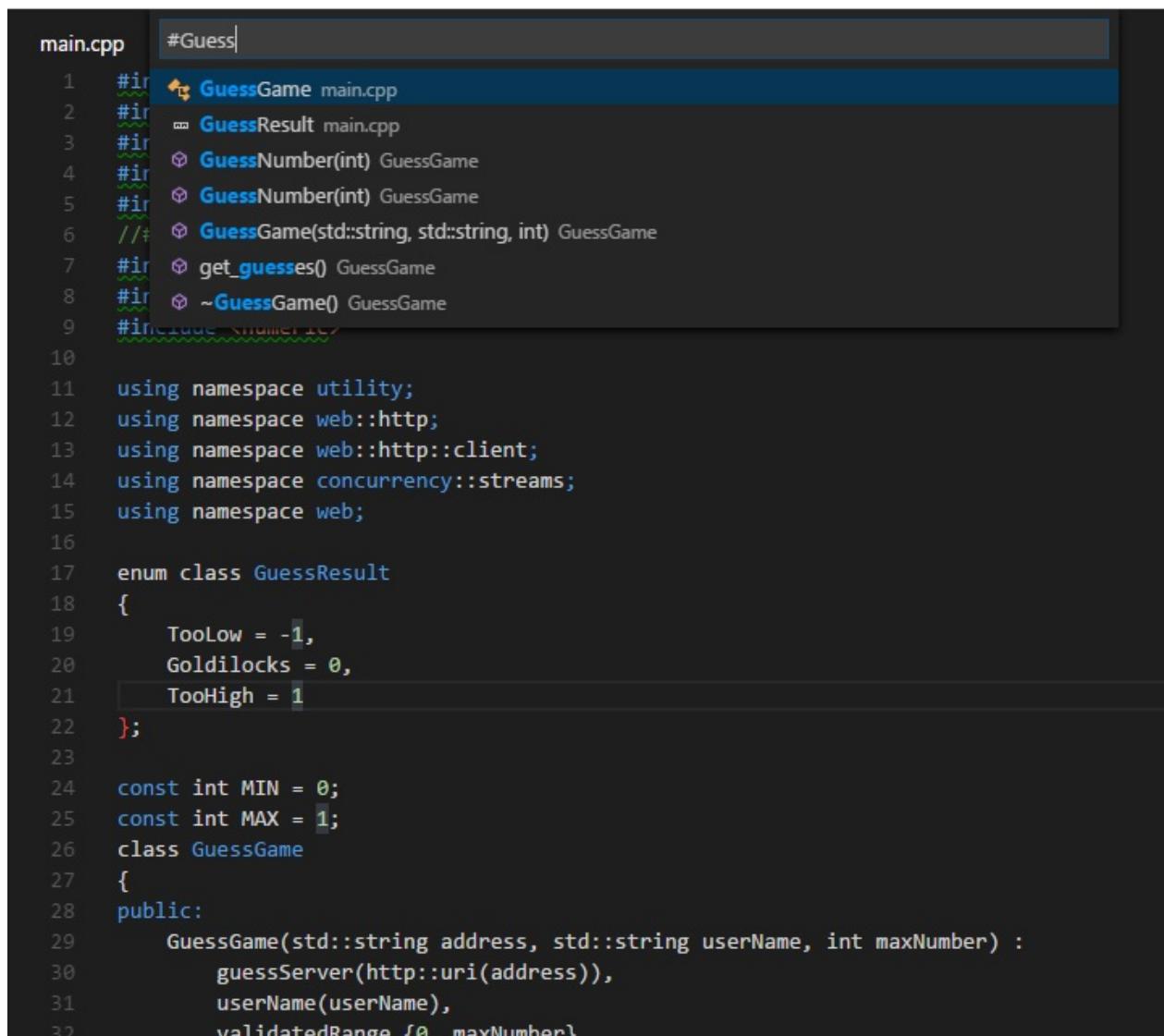
A floating search results panel titled "@Guess" is overlaid on the code editor. It contains a list of symbols found in the current workspace, with one item highlighted in blue. The list includes:

- ~GuessGame() GuessGame
- get_guesses() GuessGame
- guesses GuessGame
- GuessGame
- GuessGame(std::string, std::string, int) GuessGame
- GuessNumber(int) GuessGame
- GuessNumber(int) GuessGame
- GuessResult
- guessServer GuessGame

On the right side of the search panel, the text "symbols (9)" is displayed.

To search for a symbol in the current workspace, start by pressing

`kb(workbench.action.showAllSymbols)` instead, then enter the name of the symbol. A list of potential matches will appear as before. If you choose a match that was found in a file that's not already open, the file will be opened before navigating to the match's location.



```

main.cpp #Guess
1 #include <iostream>
2 #include "GuessResult.h"
3 #include "GuessNumber.h"
4 #include "GuessGame.h"
5 #include "GuessGame.h"
6 //#
7 #include "GuessGame.h"
8 #include "GuessGame.h"
9 #include <utility>
10
11 using namespace utility;
12 using namespace web::http;
13 using namespace web::http::client;
14 using namespace concurrency::streams;
15 using namespace web;
16
17 enum class GuessResult
18 {
19     TooLow = -1,
20     Goldilocks = 0,
21     TooHigh = 1
22 };
23
24 const int MIN = 0;
25 const int MAX = 1;
26 class GuessGame
27 {
28 public:
29     GuessGame(std::string address, std::string userName, int maxNumber) :
30         guessServer(http::uri(address)),
31         userName(userName),
32         validatedRange {0, maxNumber}

```

Alternatively, you can search for symbols by accessing these commands through the **Command Palette** if you prefer. Use **Quick Open** (`kb(workbench.action.quickopen)`) then enter the '@' command to search the current file, or the '#' command to search the current workspace. `kb(workbench.action.gotoSymbol)` and `kb(workbench.action.showAllSymbols)` are just shortcuts for the '@' and '#' commands, respectively, so everything works the same.

Peek Definition

You can take a quick look at how a symbol was defined by using the Peek Definition feature. This feature displays a few lines of code near the definition inside a peek window so you can take a look without navigating away from your current location.

To peek at a symbol's definition, place your cursor on the symbol anywhere its used in your code and then press `kb(editor.action.previewDeclaration)`. Alternatively, you can choose **Peek Definition** from the context menu (right-click, then choose **Peek Definition**).

```

136     std::string address = "http://20.0.0.124:44551";
137     if (argc >= 4)
138     {
139         address = argv[3];
140     }
141     address.append("/game");
142
143     GuessGame game(address, username, maxNo);

```

```

22 };
23
24 const int MIN = 0;
25 const int MAX = 1;
26 class GuessGame
27 {
28 public:
29     GuessGame(std::string address, std::string userName, int
30             maxNumber) :
31         guessServer(http::uri(address)),
32         userName(userName),
33         validatedRange {0, maxNumber}
34     {}
35     ~GuessGame() {}
36
37     void Start();

```

```

144     GuessResult result = GuessResult::Goldilocks;
145
146
147

```

Spaces: 4 Ln 179, Col 1 UTF-8 LF C++ Win32 😊

Currently, the C/C++ extension doesn't parse code in a way that helps it distinguish between competing definitions based on how the symbol is used. These competing definitions arise when the symbol defines different things in different contexts, such as occurs with overloaded functions, classes and their constructors, and other situations. When this happens, each of the competing definitions are listed in the right-hand side of the peek window with the source code of the current selection displayed on the left.

With the peek window open, you browse the list of competing definitions to find the one you're interested in. If you want to navigate to the location of one of the definitions just double-click the definition you're interested in, or by double-clicking anywhere in the source code displayed on the left-hand side of the peek window.

Go to Definition

You can also quickly navigate to where a symbol is defined by using the Go to Definition feature.

To go to a symbol's definition, place your cursor on the symbol anywhere its used in your code and then press `kb(editor.action.gotoDeclaration)`. Alternatively, you can choose **Go to Definition** from the context menu (right-click, then choose **Go to Definition**). When there's only one definition of the symbol, you'll navigate directly to its location, otherwise the competing definitions are displayed in a peek window as described in the previous section and you have to choose the definition that you want to go to.

Debugging

Debugging is supported on Linux (Ubuntu 14.04 64-bit) and OS X (see Known limitation below).

Preparing your launch.json file for debugging

Before you can debug your app you'll need to set a few things up. Navigate to the Debug View (click the debug icon in the toolbar on the left-hand side of the VS Code window) then in the **Debug Panel**, click the **Settings** icon and select `c++ Launch (GDB)`. This opens the `launch.json` file for editing.

```

launch.json .vscode
1  {
2      "version": "0.2.0",
3      "configurations": [
4          {
5              "name": "C++ Launch (GDB)",
6              "type": "miengine",
7              "request": "launch",
8              "launchOptionType": "Local",
9              "miDebuggerPath": "/usr/bin/gdb",
10             "targetArchitecture": "x64",
11             "program": "<enter program name, for example ${workspaceRoot}/a.out>",
12             "args": [],
13             "stopAtEntry": false,
14             "cwd": "${workspaceRoot}",
15             "environment": []
16         },
17         {
18             "name": "C++ Attach (GDB)",
19             "type": "miengine",
20             "request": "launch",
21             "launchOptionType": "Local",
22             "miDebuggerPath": "/usr/bin/gdb",
23             "targetArchitecture": "x64",
24             "program": "<enter program name, for example ${workspaceRoot}/a.out>",
25             "args": [],
26             "stopAtEntry": false,
27             "cwd": "${workspaceRoot}",
28             "environment": [],
29             "processId": "<enter program's process ID>"
30         }
31     ]
32 }

```

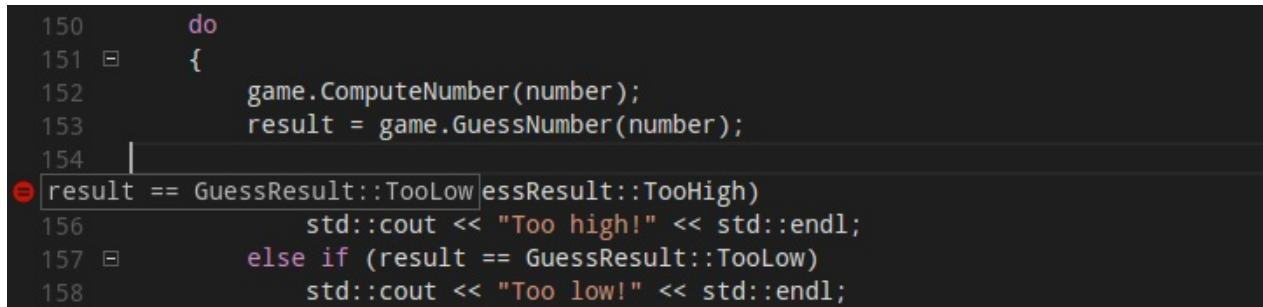
This file, `launch.json`, contains configurations that tell the debugger how to interact with your app. Two configurations are included by default -- one that defines the properties for launching your app under GDB from VS Code, and another that defines the properties for attaching GDB to a process that's already running. Note that launching your app under GDB is not currently supported on OS X, for now you have to use Attach to debug OS X apps.

At the minimum, you'll need to update the 'program' property to contain the program name and path, but you can modify other properties as well. You can view a tooltip that describes each property and its possible values by placing your cursor over a property. For more information about the properties inside the `launch.json` file and how to use them, see the VS Code [debugging documentation](#).

After your `launch.json` file is configured you're ready to start debugging, but remember that VS Code won't rebuild your program when you make changes to it between debugging sessions unless you also create a `task.json` file to invoke the build and set it as the `preLaunchTask` property in the `launch.json` file

Conditional Breakpoints

Conditional breakpoints enable you to break execution on a particular line of code only when the value of the conditional is true. To set a conditional breakpoint, right-click on an existing breakpoint and select **Edit Breakpoint**, this opens a small peek window where you can enter the condition that must evaluate to true in order for the breakpoint to activate and break execution.



A screenshot of a code editor showing a C++ file. Line 154 contains a conditional breakpoint: `else if (result == GuessResult::TooLow)`. The line is highlighted with a red rectangle, and the word "Breakpoint" is visible in the status bar at the bottom. The code is as follows:

```
150     do
151     {
152         game.ComputeNumber(number);
153         result = game.GuessNumber(number);
154     }
155     Breakpoint if (result == GuessResult::TooLow) else if (result == GuessResult::TooHigh)
156         std::cout << "Too high!" << std::endl;
157     else if (result == GuessResult::TooLow)
158         std::cout << "Too low!" << std::endl;
```

In the editor, conditional breakpoints are indicated by a breakpoint symbol that has a black equals sign inside of it. You can place the cursor over a conditional breakpoint to show its condition.

Function Breakpoints

Function breakpoints enable you to break execution at the beginning of a function rather than on a particular line of code. To set a function breakpoint, on the **Debug Panel**, right click inside the **Breakpoints** pane, then choose **Add Function Breakpoint** and enter the name of the function on which you want to break execution.

Expression evaluation

VS Code supports expression evaluation in several contexts:

- You can type an expression into the **Watch** pane and it will be evaluated each time a breakpoint is hit.
- You can type an expression into the **Debug Console** and it will be evaluated only once.
- You can evaluate any expression that appears in your code while you're stopped at a breakpoint.

Note that expressions in the Watch Pane take effect in the application being debugged; an expression that modifies the value of a variable will modify that variable for the duration of the program.

Core Dump debugging

The C/C++ extension for VS Code also has the ability to debug using a memory dump. To debug using a memory dump, open your launch.json file for editing and add the `coreDumpPath` property to the **C++ Launch** configuration, setting its value to be a string containing the path to the core dump. This will even work for multi-threaded programs and x86 programs being debugged on an x64 machine.

GDB and MI commands

You can execute GDB or MI commands directly through the debug console with the `-exec` command, but be careful -- executing GDB commands directly in the debug console is untested and might crash VS Code in some cases. For more information on debugging with VS Code, see this introduction to [debugging in VS Code](#).

Other Debugging Features

- Unconditional breakpoints
- Watch window
- Call stack
- Stepping

Known limitations

Symbols and Code Navigation

All platforms:

- Because the extension doesn't parse function bodies, Peek Definition and Go to Definition don't work for symbols defined inside the body of a function.

Debugging

Windows:

- Debugging is not currently supported on Windows.

Linux:

- Ubuntu 14.04 64-bit is the only version of Linux supported by the script that performs additional install steps on Linux. Other versions of Linux might work if you perform these steps manually, but you might need to modify them for your version of Linux. For more information on these steps, see *Manual Installation for the C++ Debugger extension* in the [README](#).

- GDB needs elevated permissions in order to attach to a process. When using *attach to process*, you need to provide your password before the debugging session can begin.

OS X:

- Additional install steps need to be completed manually to enable debugging on OS X. See *Manual Installation for the C++ Debugger extension* in the [README](#).
- *Launch process* is not currently supported on OS X.
- No additional terminal is provided for programs that already display a terminal, and the GDB shell is not available for those applications.

Next Steps

Read on to find out about:

- [Editing Evolved](#) - find out more about advanced editing features
- [Tasks](#) - use tasks to build your project and more
- [Debugging](#) - find out how to use the debugger with your project

Common Questions

Q: Which versions of Linux support debugging?

A: In this release our Linux install script targets Ubuntu 14.04 64-bit, therefore its the only version of Linux that officially supports debugging. Other versions of Linux might work if you perform the steps found in the script, but you might need to modify them for your version of Linux. For more information on these steps, see *Manual Installation for the C++ Debugger extension* in the [README](#).

Q: Why do I need provide my password to complete installation of the extension on Linux?

A: VS Code takes a dependency on [dotnet cli](#) to enable debugging support on Linux. Root access is required to install the dotnet cli package. Normally these steps are performed by a script we've provided, but you can perform the steps manually if you prefer not to run the script with elevated permissions.

Q: My project won't load.

A: VS Code doesn't currently support C++ project files, instead it considers a directory of your choosing to be the workspace of your project. Source code files inside that directory and its sub-directories are part of the workspace.

Q: IntelliSense isn't working.

A: In this release, IntelliSense isn't supported. We plan to enable this and other features in future releases.

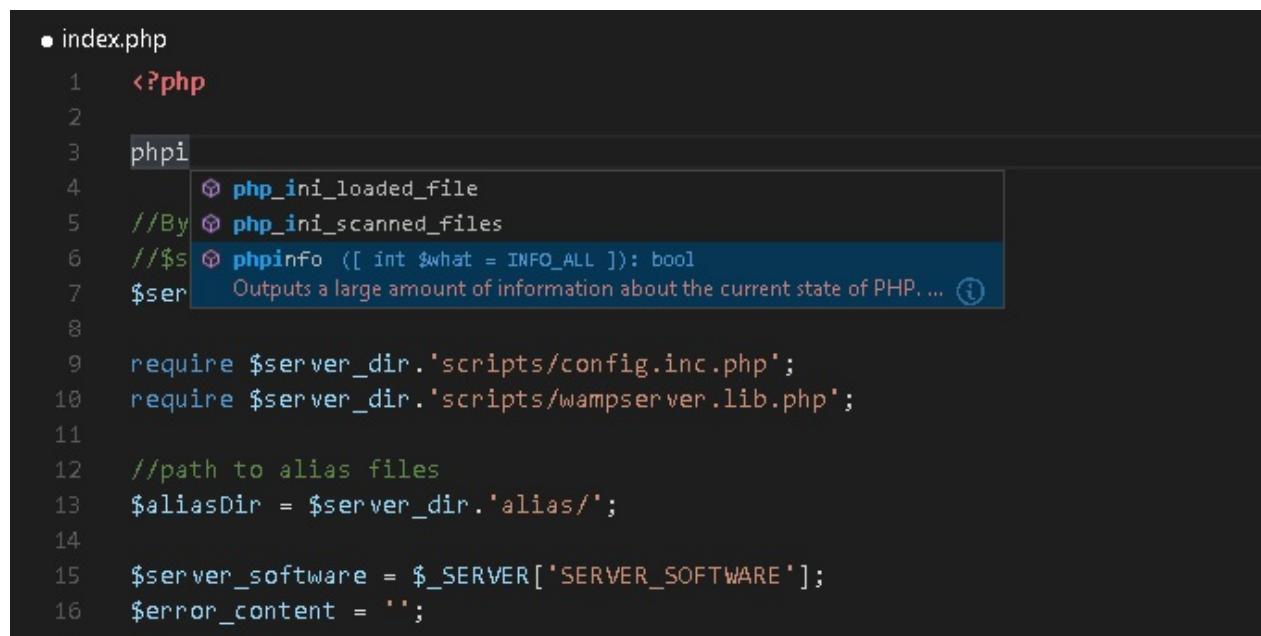
Q: How do I build/run my project?

A: VS Code supports tasks that you can configure to build your application, and natively understands the output of MSBuild, CSC, and XBuild. For more information, see the [Tasks](#) documentation.

VS Code 中的 PHP 编程 PHP Programming in VS Code

对于 PHP 开发来说，Visual Studio Code 是一个非常不错的编辑器。你可以体验到诸如语法高亮和括号匹配、代码提示以及 snippet 等特性。不仅如此，你还可以通过 VS Code 的社区获得更多功能性的扩展。

Visual Studio Code is a great editor for PHP development. You get features like syntax highlighting and bracket matching, IntelliSense, and snippets out of the box and you can add more functionality through community created VS Code [extensions](#).



A screenshot of the Visual Studio Code interface showing a PHP file named index.php. The code includes basic setup and configuration. At line 7, the variable '\$phpinfo' is being typed, and a tooltip is displayed, providing information about the function: 'Outputs a large amount of information about the current state of PHP...'. This illustrates the IDE's built-in PHP support and its ability to provide context-sensitive documentation.

```

• index.php
1 <?php
2
3 phpi
4 //By
5 //$
6 //By
7 $ser
8
9 require $server_dir.'scripts/config.inc.php';
10 require $server_dir.'scripts/wampserver.lib.php';
11
12 //path to alias files
13 $aliasDir = $server_dir.'alias/';
14
15 $server_software = $_SERVER['SERVER_SOFTWARE'];
16 $error_content = '';
17

```

代码片段 Snippets

Visual Studio Code 为 PHP 提供了一套通用的 snippet 集合。你只需要输入 `kb(editor.action.triggerSuggest)` 就可以得到与之上下文环境匹配的列表。

Visual Studio Code includes a set of common snippets for PHP. To access these, hit `kb(editor.action.triggerSuggest)` to get a context specific list.

snip.php

```

1 $arrayName = array('' => ,
2     abstract
3     acos
4     acosh
5     addslashes
6     addslashes
7     and
8     array
9     array
  Array initializer
10    array_change_key_case
11    array_chunk
12    array_combine
13    array_count_values

```

检查 Linting

VS Code 使用 PHP 官方的 `linter(php -l)` 来进行 PHP 语言的诊断。这意味着 VS Code 可以自动同步 PHP 官方 Linter 的更新。

VS Code uses the official PHP linter (`php -l`) for PHP language diagnostics. This allows VS Code to stay current with PHP linter improvements.

这里提供了三条配置项来配置 PHP linter:

There are three [settings](#) to control the PHP linter:

- `php.validate.enable` : 用于配置是否启用 PHP linting 功能。默认是启用的（Enabled）。
- `php.validate.enable` : controls whether to enable PHP linting at all. Enabled by default.
- `php.validate.executablePath` : 磁盘上的 PHP 可执行路径。如果 PHP 可执行路径不在系统路径上，则要自己手动配置该项。
- `php.validate.executablePath` : points to the PHP executable on disk. Set this if the PHP executable is not on the system path.
- `php.validate.run` : 用于配置 linter 的功能是通过“保存”(`value: "onSave"`) 触发还是“输入”(`value: "onType"`) 触发。默认是“保存”触发。
- `php.validate.run` : controls whether the validation is triggered on save (`value: "onSave"`) or on type (`value: "onType"`). Default is on save.

```
//----- PHP Configuration options -----  
  
// Whether php validation is enabled or not.  
"php.validate.enable": true,  
  
// Points to the php executable.  
"php.validate.executablePath": null,  
  
// Whether the linter is run on save or on type.  
"php.validate.run": "onSave"
```

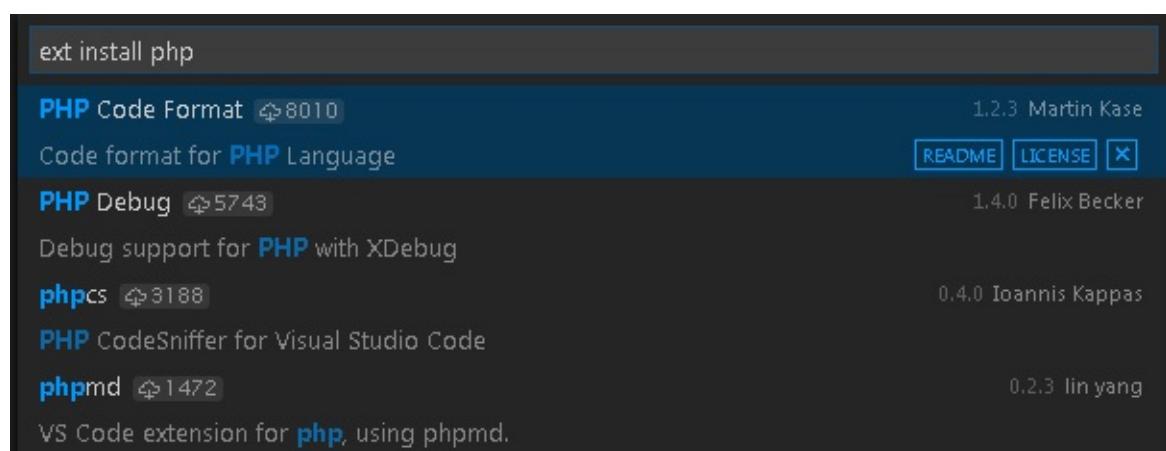
设置PHP的可执行路径，打开**User or Workspace Settings**然后添加 `php.validate.executablePath` : To set the PHP executable path, open your **User or Workspace Settings** and add the `php.validate.executablePath` :

```
{  
  "php.validate.executablePath": "c:/php/php.exe"  
}
```

扩展 Extensions

在[VS Code 市场](#)上你可以找到许多PHP相关的可用扩展，并且还有更多扩展正在陆续地再开发当中。你可以通过在VS Code中执行**Extensions: Install Extension**命令(`kb(workbench.action.showCommands)` 输入 `ext install`)来得到所有的可用扩展列表，再输入“PHP”即可显示与PHP相关的扩展列表

There are many PHP language extensions available on the [VS Code Marketplace](#) and more are being created. You can search for PHP extensions from within VS Code by running the **Extensions: Install Extension** command (`kb(workbench.action.showCommands)` and type `ext install`) then filter the extensions drop down list by typing `php` .



调试 Debugging

VS Code 支持通过 **XDebug** 来调试 PHP，[PHP Debug 扩展](#)。根据扩展的指令来配置 **XDebug** 使 XDebug 与 VS Code 协同工作。

PHP debugging with **XDebug** is supported through a [PHP Debug extension](#). Follow the extension's instructions for configuring **XDebug** to work with VS Code.

下一步 Next Steps

阅读以下相关内容：

Read on to find out about:

- [扩展市场](#) - 浏览其他已分享的扩展
- [Extension Marketplace](#) - Browse the extensions others have shared
- [调试](#) - 学习更多 VS Code 调试相关内容
- [Debugging](#) - Learn more about VS Code debugging

VS Code 对 Python 的支持

VS Code 通过扩展对 Python 充分支持。市场上流行的扩展对代码补全、linting、调试、代码格式化、代码片段等等提供了支持。

[下载 VS Code](#) - 如果您还未下载 VS Code，那就快为您的平台（Windows，Mac，Linux）安装一个吧。

安装 Python 扩展

VS Code 是一个只包含基本特性的轻量编辑器。通过安装其中一个流行的Python扩展插件，即可让 VS Code 添加对 Python 的语言支持。

1. 选择一个扩展。
2. 在命令面板 `kb(workbench.action.showCommands)` 输入 `ext install` 安装插件。

小贴士: 上示的扩展插件是动态获取的。点击上面的扩展插件名称可阅读描述和评论，判断哪个扩展最适合你。详情见 [市场](#).

本文档中的例子将使用 Don Jayamanne 流行的全部特性 [Python 扩展](#).

代码补全

Python 扩展支持代码补全和智能提示。智能提示 是一系列特性的通用术语，包括借助你所有文件以及内置或第三方模块进行代码智能补全（上下文方法和变量提示）。

快速查看方法、类名和文档。

小贴士：按下快捷键 `kb(editor.action.triggerSuggest)` 触发代码补全。

Linting

Linting 用于分析 Python 代码的潜在错误。使用 VS Code 可以快速导航到代码中错误或警告的部分。

小贴士：Don Jayamanne 的 [Python 扩展](#) 为您提供了三种不同的linter选择 - [Pylint](#), [Pep8](#), 和 [Flake8](#). 详情见 [wiki](#) 。

调试

告别“`print`”语句调试！您可以设置断点，检阅数据，以及使用调试控制台，来调试不同类型的Python应用程序（包括多线程、web和远程应用程序）。

小贴士：按照 [wiki](#) 给出的指令进行调试，包括设置你的 `launch.json` 调试配置和常见故障排除。

小贴士：想了解更多关于 VS Code 的调试信息，可见 [调试文档](#)。

代码片段

代码片段将把生产力提升到更高一个层次。您可以配置 [自己的代码片段](#) 或使用扩展提供的片段。

小贴士：使用快捷键 `kb(editor.action.triggerSuggest)`，代码片段将和代码补全出现在相同的地方。

配置

您需要安装 [扩展](#) 和 [Python](#)。其他依赖项是可选的，取决于您想使用的特性。在 [扩展 README](#) 中了解更多需求。

下一阶段

- 安装扩展 - Python 扩展可在 [市场](#) 获得。
- 基础功能 - 了解更多 VS Code 编辑器的强大功能。
- 代码导航 - 更快捷地找到相应的源代码。

常见问题

Q: 为什么 `linting` 不能正常运作？

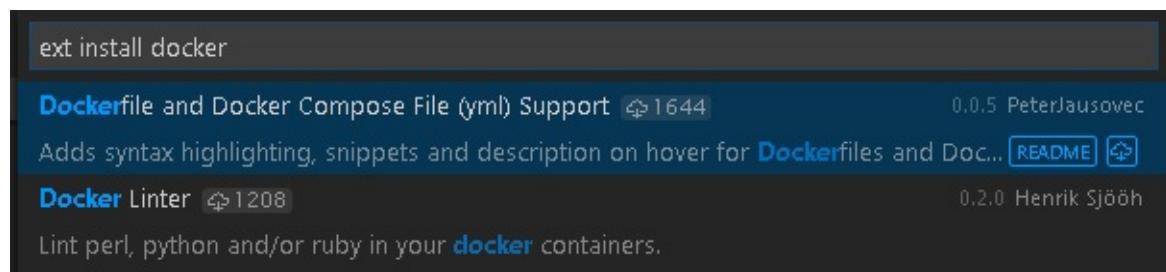
A: 首先，确保您已安装相应的扩展。其次，许多扩展依赖了外部的包，您需要使用 Python 包管理器，比如 [pip](#) 或 [easy_install](#)，来安装 [required packages](#)。您可以在 [这里](#) 阅读更多关于 `linting` 的信息。

使用Docker工作

Docker 是现今十分热门的容器引擎，可以让你轻松地打包、部署和使用应用程序以及服务。无论你是一个经验丰富的Docker开发者还是刚刚开始学习它，Visual Studio Code都可以让你轻松地创造 Dockerfile 和 docker-compose.yml 两个文件到你的开发目录中。

安装Docker扩展插件

VS Code通过插件的方式支持Docker的使用。安装这一扩展插件，只需要按下 `kb(workbench.action.showCommands)`，然后输入"ext install"并且运行**Extensions: Install Extension**命令来获得目前支持的插件列表。现在输入docker搜索所需插件然后选择 **Dockerfile and Docker Compose File (yml) Support** 插件。



Dockerfiles

通过Docker，你可以指定一系列的命令，通过它们在 Dockerfile 中建立镜像。一个 Dockerfile 是包含着一系列安装指令的文本脚本。

VS Code 很清楚Dockerfiles的结构以及可以使用的指令集，这意味着当你使用VS Code编辑这些文件时它可以给予你很多的经验指导。

1. 在你的工作目录中创建一个新的文件命名为 Dockerfile
2. 按下 `kb(editor.action.triggerSuggest)` 来获得 Dockerfile 中命令的补全

dockerfile

- ADD
ADD
- CMD
- COPY
- ENTRYPOINT
- ENV
- EXPOSE
- FROM
- LABEL
- MAINTAINER
- ONBUILD
- RUN
- USER

3. 按下 `kbstyle(Tab)` 在段落中不同的区域移动。比如说，在 `COPY` 部分你可以输入 `source`，接着按下 `kbstyle(Tab)` 移动到 `dest` 部分。

dockerfile

```
1 FROM node:latest  
2 COPY source dest
```

除了编辑 Dockerfile 时的各种功能，当你放置鼠标在一个Docker命令上的时候，Visual Studio Code将会提供关于这个命令的描述。比如说，当你的鼠标放到 WORKDIR 上面的时候你将可以看到以下描述。

```
dockerfile
Sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions
that follow it in the Dockerfile.

3 WORKDIR /src
4 ENV NODE_ENV=DEVELOPMENT
5 EXPOSE 3000
6 ENTRYPOINT npm start
```

想要获取更多关于Dockerfiles的信息，可以进入在docker.com上面的Dockerfile best practices

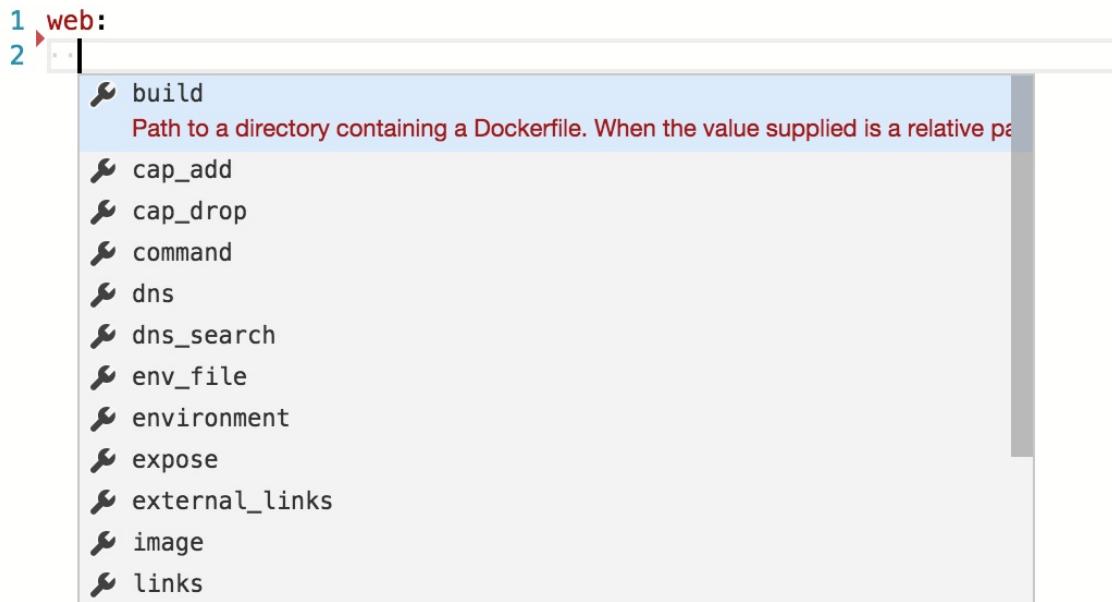
Docker compose

Docker Compose让你可以通过Docker定义以及运行多容器应用。你可以通过一个叫做 `docker-compose.yml` 的文件来定义容器的外形。

对于 `docker-compose.yml` , Visual Studio Code的功能同样也是十分丰富的。它可以为合法的 Docker compose指令提供IntelliSense, 以及帮助你查询Docker Hub找到适合的镜像。

1. 在你的工作目录中创建一个名为 `docker-compose.yml` 的新文件
2. 定义一个新的服务成为 `web`:
3. 在第二行, 通过 `kb(editor.action.triggerSuggest)` 引入IntelliSense来查看所有合法的指令列表

`docker-compose.yml`



1. 对于 `image` 指令, 你可以再次输入 `kb(editor.action.triggerSuggest)` 来完成, 而且VS Code会帮你在Docker Hub上查询公开的镜像。

docker-compose.yml



```
1 web:  
2 image:  
  centos [Official] 1066 stars  
    The official build of CentOS.  
  microsoft/aspnet  
  mongo  
  mysql  
  nginx  
  node  
  postgres  
  redis  
  ubuntu  
  wordpress
```

VS Code 第一次使用会根据一些元数据比如说star的数量和描述去为你显示一系列热门的镜像。如果你继续输入，VS code会查询Docker Hub的索引去找到更加符合的镜像，包括搜索公开的profiles。比如说，搜索 Microsoft 会显示所有微软的镜像。

docker-compose.yml



```
1 web:  
2 image: microsoft  
3 ..  
4 ..
```

microsoft/aspnet [Automated] 213 stars
 ASP.NET is an open source server-side Web application framework
 microsoft/azure-cli

用C#进行工作 Working with C#

在VS Code中对C#的支持是针对.NET跨平台开发框架的（DNX）（见“用APS.NET Core和VS Code工作”这篇相关文章）。我们致力于将VS Code作为一个优秀的跨平台C#开发的编辑器。例如让很多Unity游戏开发的厂家乐于使用VS Code来代替MonoDevelop IDE。

The C# support in VS Code is optimized for cross-platform .NET development (DNX) (see [working with ASP.NET Core and VS Code](#) for another relevant article). Our focus with VS Code is to be a great editor for cross-platform C# development. For instance, many Unity game developers enjoy using VS Code in place of the MonoDevelop IDE.

VS Code支持使用Mono进行C#应用跨平台开发调试的功能。（详见 [Mono Debugging](#)）。

We support debugging of C# apps cross-platform via Mono (see [Mono Debugging](#)).

但是因此很多标准的C#工程不能被VS Code所识别。比如ASP.NET MVC应用就是无法识别的一种。在这种情况下如果你只是单纯的想用一个轻量级工具来编辑文件，那么VS Code够用了。如果你对这些项目想有尽可能好的体验，并且通常基于Windows系统开发，我们推荐你还是用[Visual Studio社区版（Community）](#)。

Due to this focus many standard C# project types are not recognized by VS Code. An example of a non-supported project type is an ASP.NET MVC Application. In these cases if you simply want to have a lightweight tool to edit a file - VS Code has you covered. If you want the best possible experience for those projects and development on Windows in general, we recommend you use [Visual Studio Community](#).

安装对C#的支持Installing C# support

C#的支持是可选的（[点此从Marketplace安装](#)）。也可通过VS Code里的命令面板输入（拓展：安装拓展）**Extensions: Install Extension**来搜索“C#”（或者按F1并输入‘ext install’）并在下拉菜单里选择。如果你已经有了一个C#的项目，VS Code也会在你打开一个C#文件不久后提示你安装拓展。

C# language support is an optional [install from the Marketplace](#). You can install it from within VS Code by searching for 'C#' in the **Extensions: Install Extension** dropdown (`kb(workbench.action.showCommands)` and type `ext install`) or if you already have a project with C# files, VS Code will prompt you to install the extension as soon as you open a C# file.

Roslyn and OmniSharp (前者是C#的编译器，后者是对C#的自动补全及智能提示。)

VS Code通过使用 [Roslyn](#) 和 [OmniSharp](#) 来提供更好的C#体验。我们支持以下两种项目。

- DNX环境下的工程（DNX projects）
- MS解决方案下的工程（MSBuild projects）

Visual Studio Code uses the power of [Roslyn](#) and [OmniSharp](#) to offer an enhanced C# experience. We offer support for both:

- DNX projects
- MSBuild projects

在启动最佳匹配项目时，可自动加载，但也可以手动选择项目。状态栏将会显示已经加载了什么项目。并且也允许你选择不同的项目。只需要你点击状态栏项目图标并选择更换项目。下图显示有一个项目已经被打开。

On startup the best matching projects are loaded automatically but you can also choose your projects manually. The status bar will show what projects have been loaded and also allows you to select a different set of projects. To do so, click on the status bar projects item and select *Change projects*.... In the image below a single project has been picked up:



可选项包括：

- 选择一个 `project.json` 文件，将会打开一个DNX项目并且VS Code将会加载与之相关的项目。*will load that project plus the referenced projects.*
- 选择一个 `*.sln` 文件会打开一个MS解决方案，它会加载有关 `*.csproj` 的工程以及同级或子代 `project.json` 文件，但是不会打开解决方案中其他工程中的文件。
- 选择一个 `folder` 将会使VS Code搜索 `*.sln` 以及 `project.json` 文件并且VS Code将会尝试将它们全部加载。

Once the project is loaded the enhanced experiences light up...

The available options include:

- Select a `project.json` file will open a DNX-project and VS Code will load that project plus the referenced projects.
- Select a `*.sln` file opens a MSBuild-project. It will load the referenced `*.csproj` projects and sibling or descendant `project.json` files but no other project files that are referenced from the solution file.
- Select a `folder` will make VS Code scan for `*.sln` and `project.json` files and VS Code will attempt to load them all.

Once the project is loaded the enhanced experiences light up...

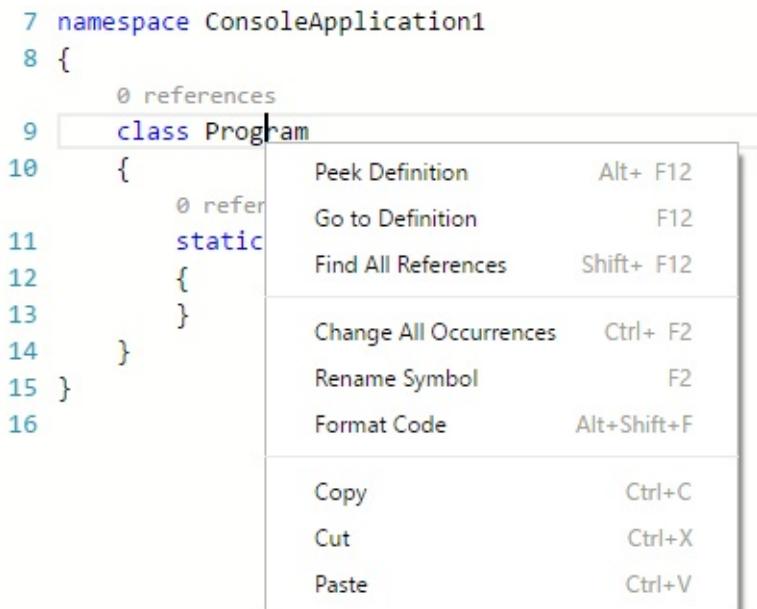
C#的编辑特性 **Editing Evolved**

你会有许多关于C#和编辑器的发现，例如代码格式，智能提示，代码重构等等。

如果想知道全部对于VS Code在代码编辑时的特点，请戳[Editing Evolved](#)这篇文章。

接下来列出一些亮点...

There is a lot to discover with C# and the editor, such as format on type, IntelliSense, the rename-refactoring, etc.



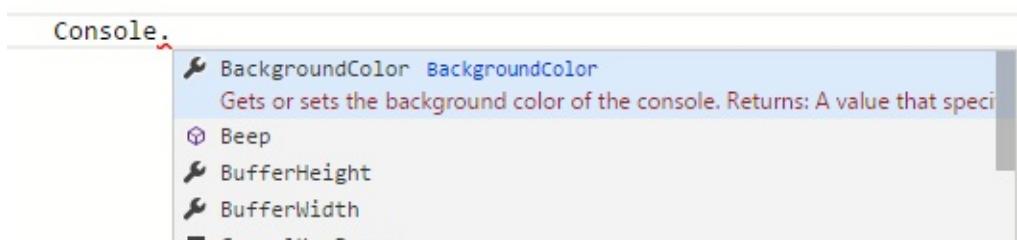
For a full description of our editing features go to the [Editing Evolved](#) documentation.

Here are a few highlights...

智能感知 **IntelliSense**

只有在你按下 `kb(editor.action.triggerSuggest)` 的时候才会得到语境下的具体建议。

IntelliSense just works hit `kb(editor.action.triggerSuggest)` at any time to get context specific suggestions.

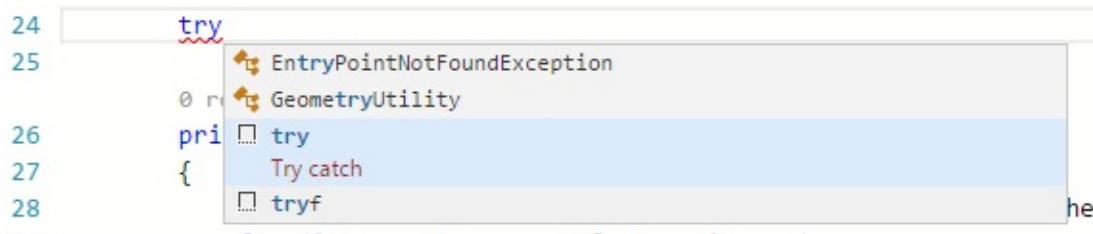


C# 的代码片段 Snippets for C#

在VS Code里内置了几种代码片段，在你输入或者按 `kb(editor.action.triggerSuggest)` (触发建议) 并且VS Code会给你一个基于语境的建议列表。

提示：你可以添加C#的自定义代码片段，详见[自定义代码片段](#)。

We have several built-in snippets included in VS Code that will come up as you type or you can press `kb(editor.action.triggerSuggest)` (Trigger Suggest) and we will give you a context specific list of suggestions.

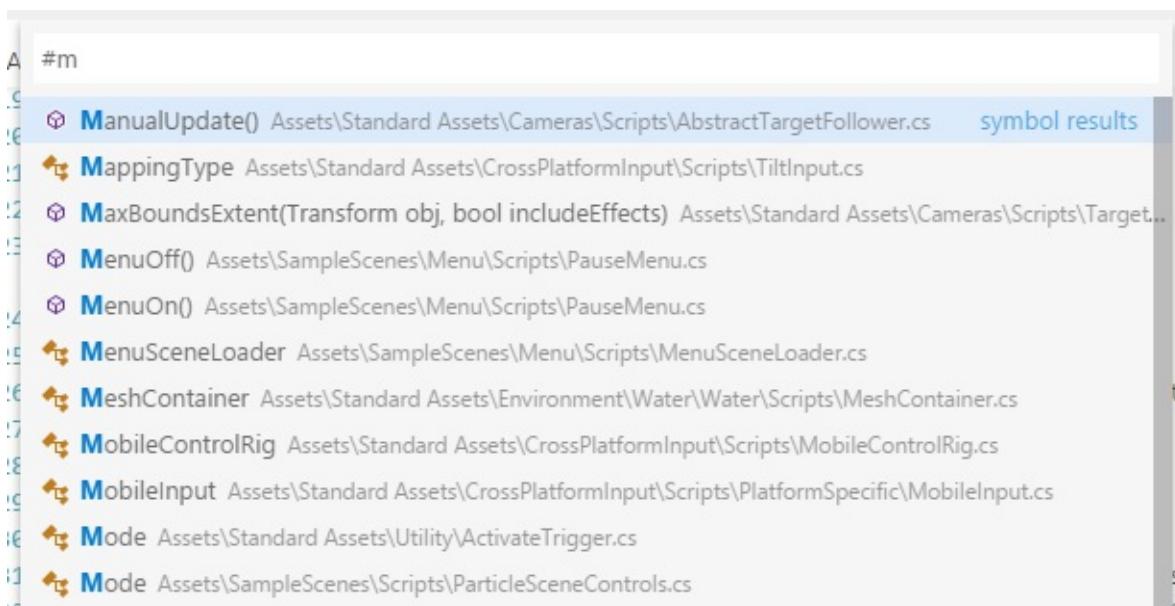


Tip: You can add in your own User Defined Snippets for C#. Take a look at [User Defined Snippets](#) to find out how.

代码追踪 Search for Symbols

VS Code同样有一些超出编辑器的特点。其中一个能力就是代码追踪。按 `kb(workbench.action.showAllSymbols)` 并开始输入，你将会看到一个相关列表。选择你想要的一项，你就会被带到它的具体位置。

There are also features outside the editor. One is the ability to search for symbols from wherever you are. Hit `kb(workbench.action.showAllSymbols)` , start typing, and see a list of matching C# symbols. Select one and you'll be taken straight to its code location.



CodeLens

另外一个很酷的特性就是能看到方法的引用次数。通过点击定义会在Peek视图看到引用信息。

Note: 由于性能原因，只有在对象中定义的方法才会被计入。例如 `equals` 和 `hashCode` 将不会得到引用信息。

提示：你可以通过 `editor.referenceInfos` 关闭引用次数提示，详见 [setting](#)

Another cool feature is the ability to see the number of references to a method directly above the method. Click on the reference info to see the references in the Peek view. This reference information updates as you type.

Note: Methods defined in `object` , such as `equals` and `hashCode` do not get reference information due to performance reasons.

```

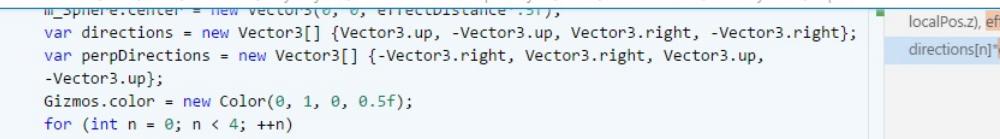
 2 references
9   public float effectAngle = 15;
 2 references
10  public float effectWidth = 1;
 3 references
11  public float effectDistance = 10;
 1 reference
12  public float force = 10;
13
 7 references
14  private Collider[] m_Cols;
 8 references
15  private SphereCollider m_Sphere;
--
```

Tip: You can turn off references information with the `editor.referenceInfos` [setting](#).

查找引用/速览定义 Find References/Peek Definition

你可以通过点击一个对象的定义来找到它的使用地点而不必离开当前语境。反过来你也可以在行间查看对象的定义。

You can click on the references of an object to find the locations of its use in place without losing context. This same experience works in reverse where you can Peek the definition of an object and see it inline without leaving your location.



```
AfterburnerPhysicsForce.cs Assets\Standard Assets\ParticleSystems\Scripts
10     public float effectWidth = 1;

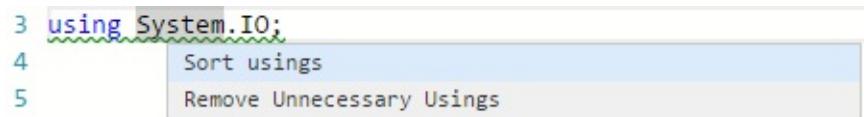
AfterburnerPhysicsForce.cs c:\Users\Public\Documents\Unity Projects\Standard Assets Example Project\Assets\Standard Assets\ParticleSystems\Scripts
33         m_Sphere.Center = new Vector3(0, 0, effectDistance * .5f);
54         var directions = new Vector3[] {Vector3.up, -Vector3.up, Vector3.right, -Vector3.right};
55         var perpDirections = new Vector3[] {-Vector3.right, Vector3.right, Vector3.up,
56             -Vector3.up};
57         Gizmos.color = new Color(0, 1, 0, 0.5f);
58         for (int n = 0; n < 4; ++n)
59         {
60             Vector3 origin = transform.position + transform.rotation*directions[n]
61             *effectWidth*0.5f;
62             Vector3 direction =
63                 transform.TransformDirectionQuaternion.AngleAxis(effectAngle, perpDirections[n])
64                 *Vector3.forward);
65             Gizmos.DrawLine(origin, origin + direction*m_Sphere.radius*2);
66         }
67     }

3 references
11     public float effectDistance = 10;
```

快速修复/建议 Quick Fixes / Suggestions

在VS Code里提供一些基础的快速修复功能。你会看到一个小灯泡然后点击它，或者按 `kb(editor.action.quickFix)` 提供一个简单列表提供修复或者修复意见。

There are some basic quick fixes supported in VS Code. You will see a lightbulb and clicking on it, or pressing `kb(editor.action.quickFix)` provides you with a simple list of fixes/suggestions.



接下来 Next Steps

看看关于：

- [ASP.NET Core 开发](#) - 如何安排并采用跨平台框架.NET
 - [编辑特性（Editing Evolved）](#) - 看看更多优秀的编辑特性。
 - [Tasks](#) - use tasks to build your project and more
 - [调试相关（Debugging）](#) - 看看如何在你的工程里使用调试功能。

Read on to find out about:

- [ASP.NET Core Development](#) - get up and running with cross-platform .NET
 - [Editing Evolved](#) - find out more about advanced editing features
 - [Tasks](#) - use tasks to build your project and more
 - [Debugging](#) - find out how to use the debugger with your project

常见问题 Common Questions

Q: 为啥我的项目无法加载。

A: VS Code 只支持有限的工程种类（主要是ASP.NET Core）。如果想要对.NET平台所有种类工程的支持，还是推荐你使用 [Visual Studio社区版（Community）](#)。

Q: 为啥智能感知不好用。

A: 这通常是因为当前项目的类型不受支持的结果。你可以看看状态栏左下角的OmniSharp的火焰提示。

Q: 我该怎么建立/运行我的工程？

A: VS Code 支持任务的建立以及对于MSBuild，CSC,XBuild的理解。详情请查阅 [Tasks](#) 文档。

Q: My Project won't load.

A: VS Code only supports a limited set of project types (primarily ASP.NET Core). For full .NET project support, we suggest you use [Visual Studio Community](#).

Q: IntelliSense is not working.

A: This is typically as a result of the current project type not being supported. You can see an indication in the OmniSharp flame in the bottom left hand side of the status bar.

Q: How do I build/run my project?

A: VS Code supports tasks for build and natively understand the output of MSBuild, CSC, XBuild. Find out more in the [Tasks](#) documentation.

运行时

- nodejs
- ASPnet5
- unity
- office

Node.js Applications with VS Code

用VS Code开发Node.js应用

[Node.js](#) is a platform for building fast and scalable server applications using JavaScript. Node.js is the runtime and [NPM](#) is the Package Manager for Node.js modules.

[Node.js](#)是一个使用JavaScript开发和部署快速且规模化的服务端应用的平台。Node.js是运行时，而[NPM](#)是Node.js模块的包管理器。

To get started, [install Node.js for your platform](#). The Node Package Manager is included in the Node.js distribution. You'll need to open a new terminal (command prompt) for `npm` to be on your PATH.

开始之前，你需要先安装一个适合你的操作系统的[Node.js](#)。Node包管理器（即npm）已经内置在了这个Node.js发行版中。你需要打开终端（控制台）设置 npm 的环境变量。

Tip! You can download both the TypeScript and JavaScript versions of the sample application created in this walkthrough from the [vscode-samples](#) repository.

Tip! 你可以从[vscode-samples](#)这个仓库里下载JavaScript和TypeScript的示例程序。

Express

[Express](#) is a very popular application framework for building and running Node.js applications. You can scaffold a new Express application using the Express Generator tool, which is typically installed globally on your computer.

[Express](#)是一个用于构建和运行Node.js应用的非常流行的框架。你可以用Express Generator Tool去创建一个新的Node.js应用，这个脚手架通常全局安装在你的计算机上。

```
npm install -g express-generator
```

We can now scaffold a new Express application called `myExpressApp`.

我们可以创建一个新的名为 `myExpressApp` 的Express应用

```
express myExpressApp
```

This creates a new folder called `myExpressApp` with the contents of your application. To install all of the application's dependencies, go to the new folder and execute `npm install`:

这条命令将创建一个新的叫做 `myExpressApp` 的目录，里面是你的应用。为了安装这个应用所有的依赖项，你需要进入这个新创建的目录然后运行：`npm install`

```
cd myExpressApp  
npm install
```

At this point, we should test that our application runs. The generated Express application has a `package.json` file which includes a `start` script to run `node ./bin/www`. This will start the Node.js application running.

这时候，你应该测试一下你的应用能否跑起来。Express应用都有一个 `package.json` 文件，里面包含了 `start` 脚本，用来运行 `node ./bin/www`。这将让你的Node.js应用跑起来。

From a terminal in the Express application folder, run:

在这个Express应用的目录里打开一个终端，运行：

```
npm start
```

The Node.js web server will start and you can browse to `http://localhost:3000` to see the running application.

Node.js Web服务器将会开启，你可以通过访问 `http://localhost:3000` 看到这个跑起来的应用。

```

myExpressApp — node — 69x23
```
-- etag@1.6.0 (crc@3.2.1)
-- proxy-addr@1.0.8 (forwarded@0.1.0, ipaddr.js@1.0.1)
-- send@0.12.3 (destroy@1.0.3, ms@0.7.1, mime@1.3.4)
-- accepts@1.2.7 (negotiator@0.1.2)
-- type-is@1.6.2 (media-type@0.1.0)
jade@1.9.2 node_modules/jade
-- character-parser@1.2.1
-- void-elements@2.0.1
-- commander@2.6.0
-- mkdirp@0.5.1 (minimist@0.2.8)
-- with@4.0.3 (acorn-glob@0.2.0)
-- constantinople@3.0.1 (method-override@2.3.0)
-- transformers@2.1.0 (pako@0.2.9)
~/src/myExpressApp$ npm start
> myExpressApp@0.0.0 start
> node ./bin/www
GET / 200 213.699 ms - 176
GET /stylesheets/style.css 200 173.699 ms - 176
GET /favicon.ico 404 33.500 ms
```

```

Express
Welcome to Express

Great Code Editing Experiences

极好的代码编辑体验

Close the browser and from a terminal in the `myExpressApp` folder, stop the Node.js server by pressing `kbstyle(CTRL+C)`.

关掉你的浏览器，在 `myExpressApp` 目录的终端里用 `kbstyle(CTRL+C)` 停止Node.js服务器。

Now launch VS Code:

现在打开VS Code:

```
code .
```

Tip: You can open files or folders directly from the command line. The period `'.'` refers to the current folder, therefore VS Code will start and open the `myExpressApp` folder.

Tip: 你可以直接通过命令行打开文件或者目录。这时候，`'.'`指向整个当前目录，因此VS Code将会运行，并且打开"myExpressApp"目录。

The [Node.js](#) and [Express](#) documentation does a great job explaining how to build rich applications using the platform and framework. Visual Studio Code will make you more productive developing these types of applications by providing great code editing and navigation experiences.

[Node.js](#) 和 [Express](#) 的文档非常清晰的阐述了如何用这两者去构建富应用。而Visual Studio Code会通过提供极好的编辑体验和提示体验，来帮助你提高对这些富应用的生产力。

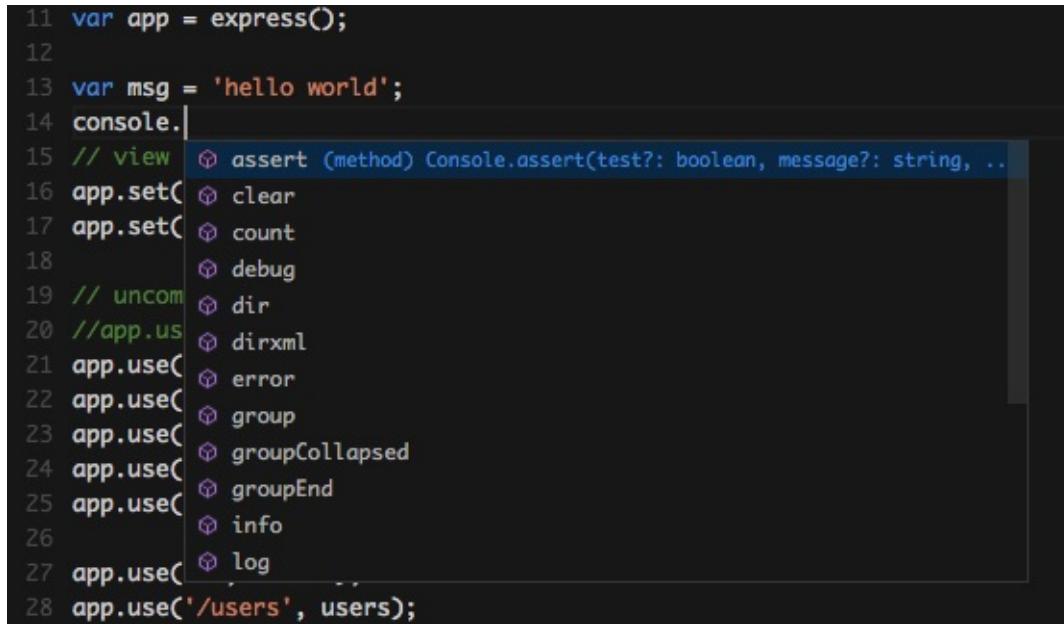
VS Code uses the TypeScript compiler to drive its JavaScript language service, which means we can take advantage of what the compiler can infer about your code. For example, let's create a simple string variable in `app.js` and send the contents of the string to the console.

VS Code 使用 TypeScript 编译器去驱动其 JavaScript 语言服务，这意味着我们可以利用 TypeScript 编译器的能力去分析、推断你的代码。举个栗子，我们可以创建一个简单的字符串变量，然后将这个字符串在控制台中打印出来。

```
var msg = 'hello world';
console.log(msg);
```

Note that when you typed `console.` IntelliSense on the `console` object was automatically presented to you. When editing JavaScript files, VS Code will automatically provide you with IntelliSense for the DOM.

注意，当你打出了 `console.` 的时候，`console` 对象的智能提示就自动的出现在你眼前了。在编辑JavaScript程序的时候，VS Code 将自动为你提供DOM的智能提示。



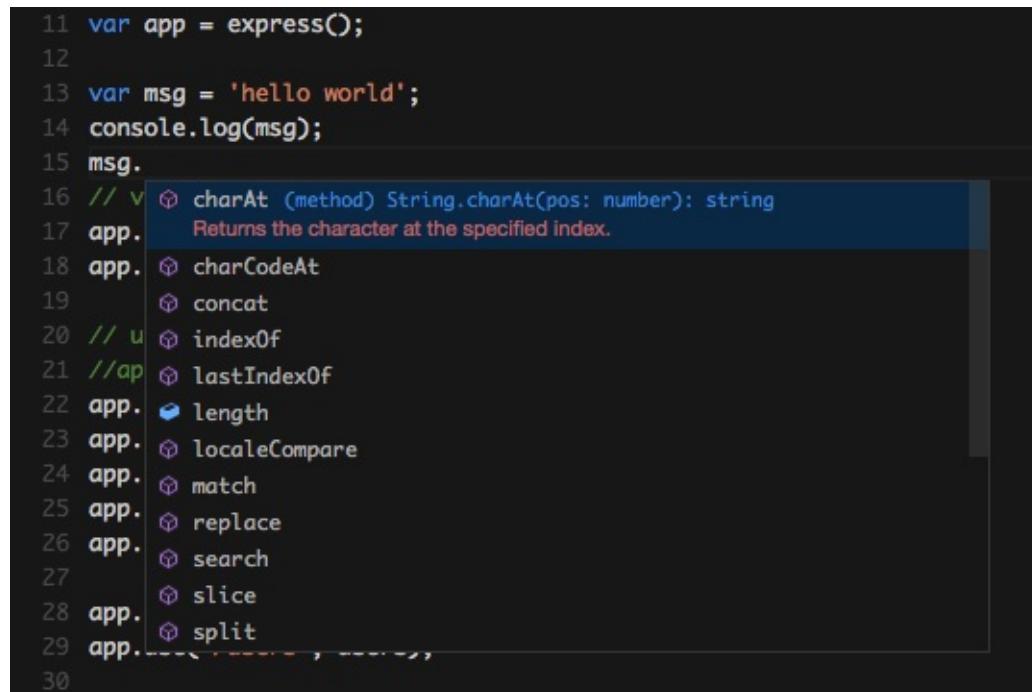
A screenshot of the Visual Studio Code editor showing code completion for the `console` object. The code being typed is:

```
11 var app = express();
12
13 var msg = 'hello world';
14 console.|
```

The cursor is at the end of `console.`. A dropdown menu shows various methods available on the `console` object, including `assert`, `clear`, `count`, `debug`, `dir`, `dirxml`, `error`, `group`, `groupCollapsed`, `groupEnd`, `info`, and `log`.

Also notice that VS Code knows that `msg` is a string based on the initialization to `'hello world'`. Type `msg.` to bring up IntelliSense and you'll see all of the string functions available on `msg`.

另外你可以注意到，VS Code知道 `msg` 是一个字符串，因为你之前用 `'hello world'` 来初始化了它。输入 `msg.` 可以呼出智能提示，你会看到 `msg` 的所有可用的字符串方法。



A screenshot of the Visual Studio Code interface showing code completion (Intellisense) for a variable named `msg`. The code is a simple script using the Express.js framework:

```

11 var app = express();
12
13 var msg = 'hello world';
14 console.log(msg);
15 msg.
16 // v ⚡ charAt (method) String.charAt(pos: number): string
17 app. Returns the character at the specified index.
18 app. ⚡ charCodeAt
19 ⚡ concat
20 // u ⚡ indexOf
21 // ap ⚡ lastIndexOf
22 app. ⚡ length
23 app. ⚡ localeCompare
24 app. ⚡ match
25 app. ⚡ replace
26 app. ⚡ search
27 ⚡ slice
28 app. ⚡ split
29 app. ⚡ substr, ...
30

```

The cursor is at the dot after `msg.`, and a tooltip is displayed for the `charAt` method, which is highlighted in blue. The tooltip contains the method name, its description ("Returns the character at the specified index."), and its signature (`(method) String.charAt(pos: number): string`). Other methods like `charCodeAt`, `concat`, `indexOf`, etc., are listed below it.

Adding a `jsconfig.json` Configuration File

添加 `jsconfig.json` 配置文件

You can give even more hints to Visual Studio Code through a configuration file for the workspace (the root folder). Add a new file and name it `jsconfig.json` with the following contents:

你可以利用工作空间（项目的根目录）中的配置文件来为VS Code添加更多的提示。创建一个名叫 `jsconfig.json` 的新文件，内容如下：

```
{
  "compilerOptions": {
    "target": "es5",
    "module": "commonjs"
  }
}
```

The presence of this file lets VS Code know that it should treat all the files under this root as part of the same project. The specific `compilerOptions` tells VS Code you are writing ES5 compliant code and you want to use the [CommonJS module system](#).

这个配置文件可以让VS Code把根目录下的所有文件当做同一个项目的一部分来对待。`compilerOptions` 字段告诉了VS Code：你在编写兼容ES5规范的代码，并且使用[CommonJS模块系统](#).

Typings

VS Code can use TypeScript definition files (for example `node.d.ts`) to provide metadata to VS Code about the JavaScript based frameworks you are consuming in your application. Because TypeScript definition files are written in TypeScript, they can express the data types of parameters and functions, allowing VS Code to provide not only a rich IntelliSense experience, but also warnings when an API is being used incorrectly.

VS Code能够使用TypeScript定义文件（例如 `node.d.ts`），为VS Code提供你的应用所使用的框架的元数据。由于TypeScript定义文件是用TypeScript编写的，所以它们可以分辨参数数据的类型和函数的类型，这允许了VS Code不仅提供智能提示功能，还可以在你错误的使用了API的时候发出警告。

[Typings](#), the type definition manager for TypeScript, makes it easy to search for and install TypeScript definition files into your workspace. This tool can download the requested definitions from a variety of sources, including the [DefinitelyTyped repository](#). As we did with the express generator, we will install Typings globally using NPM so that you can use the tool in any application you create.

[Typings](#)是TypeScript的类型定义管理器，它可以帮助我们更容易地搜索和在工作空间中安装TypeScript定义文件。这个工具可以从一个海量源中下载定义文件，包括[DefinitelyTyped repository](#)。就像我们之前安装Express生成器一样，我们将用NPM全局安装Typings，这样你就可以在你的任何应用中使用这个工具。

```
npm install -g typings
```

Tip: Typings has a number of options for configuring where and how definition files are downloaded, from the terminal run `typings --help` for more information.

Tip: Typings有许多的配置选项，用于配置定义文件在哪里、通过何种方式下载。在终端中运行：`typings --help` 可以看到更多信息。

Go back to the file `app.js` and notice that if you hover over the Node.js global object `__dirname`, VS Code does not know the type and displays `any`.

回到 `app.js` 这个文件，你会注意到，当鼠标悬浮在Node.js的全局对象 `_dirname` 上的时候，VS Code并不知道它的类型，并且显示 `any`。

Now, using the Typings command line, pull down the Node and Express definition files.

现在，使用Typings的命令行，拉取Node和Express的定义文件。

```
typings install node --ambient
typings install express serve-static express-serve-static-core --ambient
```

Tip: You can download multiple definition files by combining them on the command line, as you can see from the Express typings above. We need to install the typings for Express and also its references.

Tip: 你可以在一行命令中安装多个类型定义文件，就像上面我们做的那样。我们不仅需要为Express安装类型定义文件，还要为它的引用安装。

Notice how VS Code now understands what `_dirname` is, based on the metadata from the `node.d.ts` file. Even more exciting, you can get full IntelliSense against the Node.js framework. For example, you can require `http` and get full IntelliSense against the `http` class as you type in Visual Studio Code.

你会发现利用 `node.s.ts` 文件，VS Code现在已经明白了 `_dirname` 是什么类型的。更让人激动的是，你可以在使用Node.js框架的情况下获得完整的智能提示了，当你 `require` 一个 `http` 模块的时候，你刚在VS Code里输入，一个完整的智能提示就会出现在 `http` 旁边。

```
7 var cookieParser = require('cookie-parser');
8 var bodyParser = require('body-parser');
9 var http = require('http');
10 http.|
```

```
11   ↗ Agent
12 var r ↗ createClient
13 var u ↗ createServer (function) http.createServer(requestListener?: (request: Request, response: Response) => void, options?: ServerOptions): Server;
14   ↗ get
15 var a ↗ globalAgent
16   ↗ request
17 // vi ↗ STATUS_CODES
18 app.set('views', path.join(__dirname, 'views'));
19 app.set('view engine', 'jade');
```

You can also write code that references modules in other files. For example, in `app.js` we require the `./routes/index` module, which exports an `Express.Router` class. If you bring up IntelliSense on `routes` , you can see the shape of the `Router` class.

你也可以编写代码，引用在其他文件中的模块试试。举个栗子，在 `app.js` 中，我们require了 `./routes/index` 模块，这个模块输出了一个 `Express.Router` 的类。如果你在 `routes` 上呼出智能提示，你就会看到 `Router` 类的大致结构。

```

13 var msg = 'hello world';
14 console.log(msg);
15
16 routes.
17   ↘ all (property) e.IRouter<'express'.Router>.all: 'express'.IRouterMai
18 // view Special-cased "all" method, applying the given route 'path', middleware, and callb
19 app.set ↗ apply
20 app.set ↗ arguments
21 ↗ bind
22 // unco ↗ call
23 //app.u ↗ caller
24 app.use ↗ delete
25 app.use ↗ get
26 app.use ↗ length
27 app.use ↗ options
28 app.use ↗ param
29 ↗ patch
30 app.use` , -----,
31 app.use('/users', users);
32

```

Debugging your Node Application

为你的Node应用 Debug

In order to run and debug your Node.js application from within VS Code, you need to configure how the application will be started. To do this, click on the Debug icon in the View Bar on the left of Visual Studio Code.

为了在VS Code中运行和调试你的Node.js应用，你需要设置应该以何种方式被启动。在VS Code左侧的View这一栏中点击Debug图标。



Click on the Configure gear icon at the top of the Debug view to create a default `launch.json` file and select "Node.js" as the Debug Environment. This configuration file lets you specify how to start the application, what arguments to pass in, the working directory, and more. When the file is first created, VS Code will look in `package.json` for a `start` script and will use that value as the `program` (which in this case is `${workspaceRoot}/bin/www`) for the `Launch` configuration. A second `Attach` configuration is also created to show you how to attach to a running Node application.

在Debug视图上点击齿轮形的图标，创建一个默认的 `launch.json` 文件，选择“Node.js”作为调试环境。这个配置文件让你能够设置如何去启动应用、启动的参数、工作目录等等。当文件第一次被创建的时候VS Code会在 `package.json` 中寻找 `start` 脚本，并且会把 `start` 的值当

做 `Launch` 配置文件的 `program` (当前情况下是 `${workspaceRoot}/bin/www`) 。第二个名为 `Attach` 的配置文件也会被创建，用于配置连接到运行的Node应用的方式。

```

1  {
2    "version": "0.2.0",
3    "configurations": [
4      {
5        "name": "Launch",
6        "type": "node",
7        "request": "launch",
8        "program": "${workspaceRoot}/bin/www",
9        "stopOnEntry": false,
10       "args": [],
11       "cwd": "${workspaceRoot}",
12       "preLaunchTask": null,
13       "runtimeExecutable": null,
14       "runtimeArgs": [
15         "-nolazy"
16       ],
17       "env": {
18         "NODE_ENV": "development"
19       },
20       "externalConsole": false,
21       "sourceMaps": false,
22       "outDir": null
23     },
24     {
25       "name": "Attach",
26       "type": "node",
27       "request": "attach",
28       "port": 5858,
29       "address": "localhost",
30       "restart": false,
31       "sourceMaps": false,
32       "outDir": null,
33       "localRoot": "${workspaceRoot}",
34       "remoteRoot": null
35     }
36   ]
37 }

```

Take the defaults for everything else. If you do not have `Auto Save` on, save the file by pressing `kb(workbench.action.files.save)` , and make sure `Launch` is selected in the configuration dropdown at the top of the Debug view. Open `app.js` and set a breakpoint on the line of code we wrote earlier `var msg = 'hello world';` by clicking in the gutter to the left of the line number. Press `kb(workbench.action.debug.start)` to start debugging the application. VS Code will start the server in a new terminal and hit the breakpoint we set. From there you can inspect variables, create watches, and step through your code.

对于其余的选项，使用默认配置即可。如果你没有开启 `自动保存`，通过输入 `kb(workbench.action.files.save)` 来保存文件，并且确保Debug视图中的下拉菜单里的 `Launch` 被正确的选择了。打开 `app.js` 在 `var msg = 'hello world';` 的行号处设置一个断点。输入 `kb(workbench.action.debug.start)`，开始调试应用。VS Code将会在一个新的终端里开启服务器并在我们设置的断点处暂停。这里你可以审查变量、创建watcher，以及分部调试代码。

```

app.js /
1 //<reference path="typings/node/node.d.ts"/>
2
3 var express = require('express');
4 var path = require('path');
5 var favicon = require('serve-favicon');
6 var logger = require('morgan');
7 var cookieParser = require('cookie-parser');
8 var bodyParser = require('body-parser');
9
10
11 var routes = require('./routes/index');
12 var users = require('./routes/users');
13
14 var app = express();
15
16 // view engine setup
17 app.set('views', path.join(__dirname, 'views'));
18 app.set('view engine', 'jade');
19
20 // uncomment after placing your favicon in /public

```

Extensions

扩展

The community is continually developing more and more valuable extensions for Node.js. Here are some popular extensions that you might find useful.

VS Code的社区正在为Node.js不断开发越来越多的有价值的扩展。这里有一些你可能会觉得有用的，同时也很流行的扩展。

- [View Node Package](#) - Open a Node.js package repository/documentation straight from VS Code. (直接用VS Code打开Node.js包的仓库/文档)
- [JavaScript \(ES6\) code snippets](#) - Snippets for JavaScript in ES6 syntax. (支持ES6语法的JavaScript代码高亮插件)
- [ESLint](#) - Integrates ESLint into VS Code. (在VS Code中集成了ESLint)
- [JSHint](#) - Integrates JSHint into VS Code. (在VS Code中集成了ESHint)
- [Add JSDoc comments](#) - Adds **JSDoc** @param and @return tags for selected function signatures in JS and TS. (用于添加包含@param和@return的注释)
- [Prettify JSON](#) - Prettify ugly JSON inside VS Code. (在VS Code里实现JSON的格式化和压缩)
- [Beautify](#) - This extension enables running **js-beautify** in VS Code. (这个插件允许在VS Code中运行js-beautify)

Next Steps

下一步

There is much more to explore with Visual Studio Code, please try the following topics:

Visual Studio Code还有很多地方可以去探索，可以看看下面的几个话题：

- [Debugging](#) - This is where VS Code really shines
- [Editing Evolved](#) - Lint, IntelliSense, Lightbulbs, Peek and Goto Definition and more
- [ASP.NET Core](#) - End to end sample showing off our ASP.NET Core and .NET Core support with a sample app
- [Tasks](#) - Running tasks with Gulp, Grunt and Jake. Showing Errors and Warnings

ASP.NET Core with VS Code

Note: ASP.NET Core and DNX (the .NET Execution Environment) on OS X and Linux are in an early Beta/Preview state. We recommend following the [ASP.NET Home project](#) on GitHub for the latest information.

ASP.NET Core and DNX

[ASP.NET Core/DNX](#) is a lean .NET stack for building modern cloud and web apps that run on OS X, Linux, and Windows. It has been built from the ground up to provide an optimized development framework for apps that are either deployed to the cloud or run on-premises. It consists of modular components with minimal overhead, so you retain flexibility while constructing your solutions.

Installing ASP.NET Core and DNX

Installation for each platform (Windows, OS X and Linux) is slightly different and covered in detail at [ASP.NET Getting Started](#).

Getting Started

If you don't have an existing ASP.NET DNX application, here's a nice tutorial to [Create an ASP.NET web app in VS Code](#). It will walk you through installing prerequisites and scaffolding out a web application.

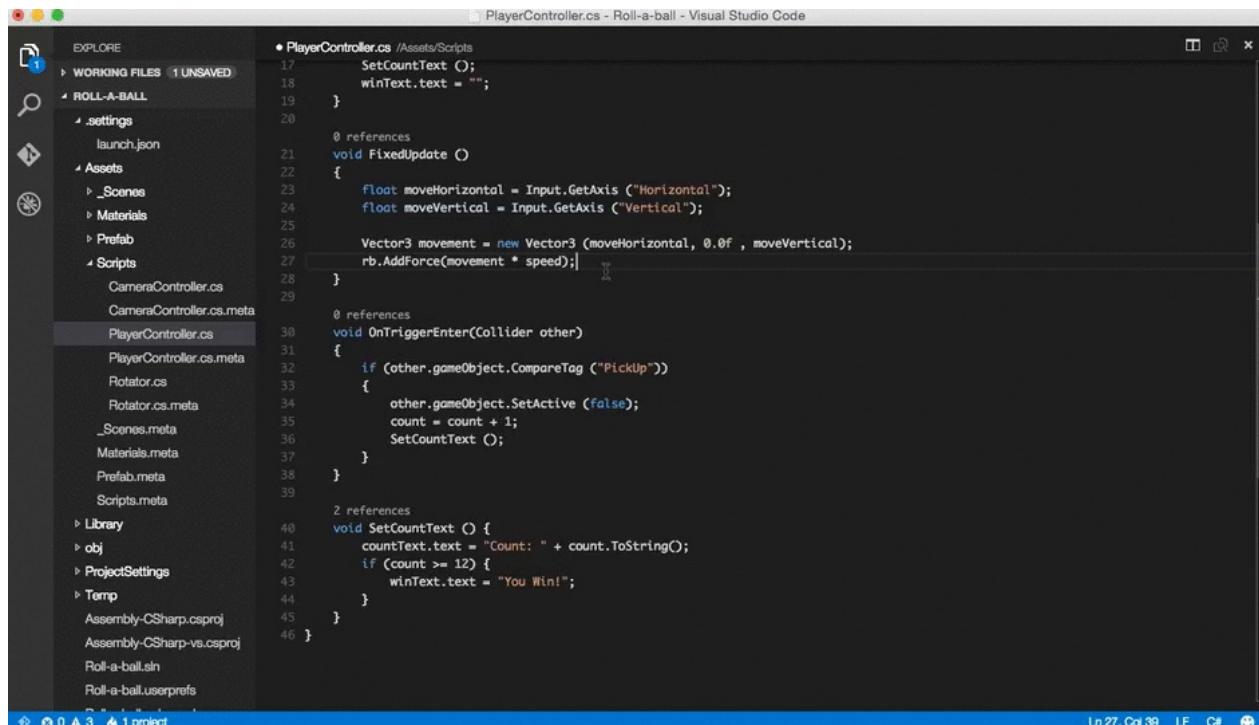
Next Steps

We hope this gets you started building ASP.NET Core applications. Try these things next:

- [Editing Evolved](#) - Lint, IntelliSense, Lightbulbs, Peek and Go to Definition and more
- [Working with C#](#) - Learn about the great C# support you'll have when working on your ASP.NET application.
- [Tasks](#) - Running tasks with Gulp, Grunt and Jake. Showing Errors and Warnings

Unity Development with VS Code

Visual Studio Code can be a great companion to Unity for editing and debugging C# files. All of the [C#](#) features are supported and more. In the screen below you can see code colorization, bracket matching, IntelliSense, CodeLens and that's just the start.



Read on to find out how to configure Unity and your project to get the best possible experience.

Note: VS Code uses a more recent version of Mono than that included with Unity. If you get OmniSharp errors, you may need to update your Mono version. See this [FAQ topic](#) for additional details.

Connecting Unity and VS Code

The easiest way to get going is to leverage a [Unity plug-in](#) maintained by [@Reapazor](#). This plug-in streamlines the integration process significantly by performing the following tasks:

1. Sets VS Code as the default editor - opening a script now opens it in VS Code
2. Configures Unity to pass file and line numbers and reuse the existing window - so VS Code opens in the correct context
3. Scrubs the Unity project file to ensure that OmniSharp can work with it - to get the best editing experience

4. Configures VS Code to ignore certain Unity file types - removing clutter from the VS Code file explorer
5. Configures a `launch.json` file with the correct debug port - to enable debugging. However, installing the [Debugger for Unity](#) is recommended to get debugging support.

Step 1: Download the plug-in code

Open up a console and do a clone of the repo to get the plug-in source code.

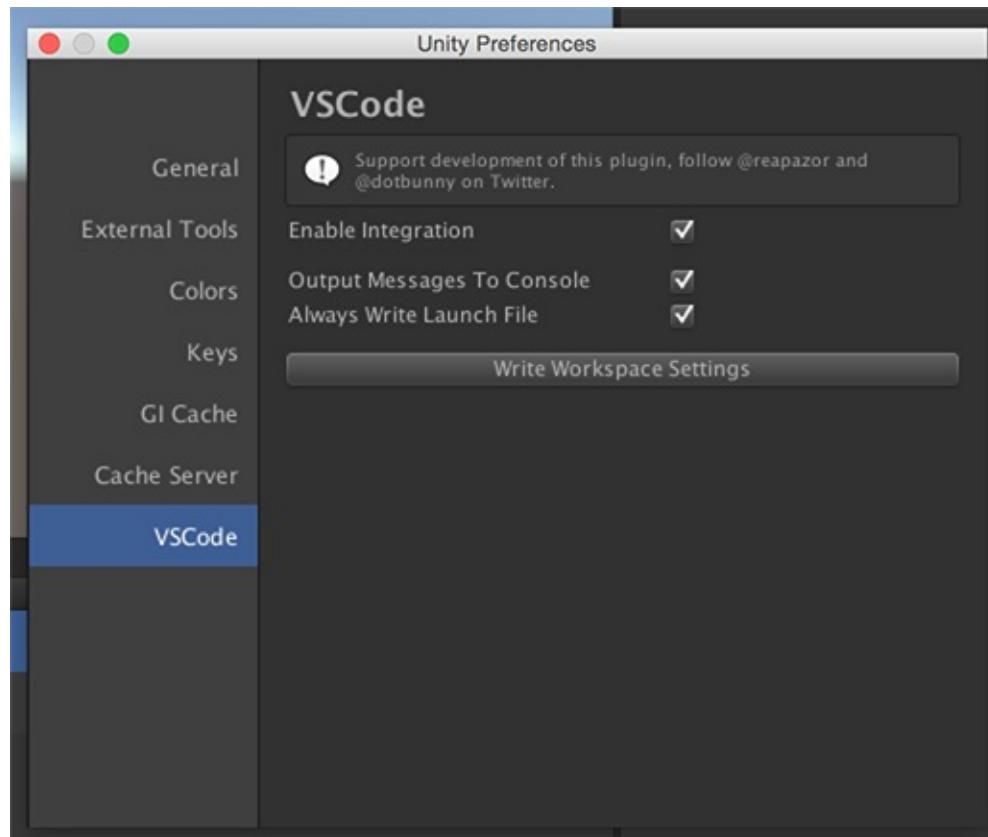
```
git clone https://github.com/dotBunny/vsCode.git
```

Step 2: Add the plug-in to your project

Go to the folder where you downloaded the plug-in source code and copy the `Plugins\Editor\dotBunny` folder to your Unity project.

Tip: You may need to create a `Plugins` folder. Typically this should be stored under `Assets`.

To turn on the use of the provided integration, you will need to go to `unity Preferences` and select the newly created `vscode` tab.



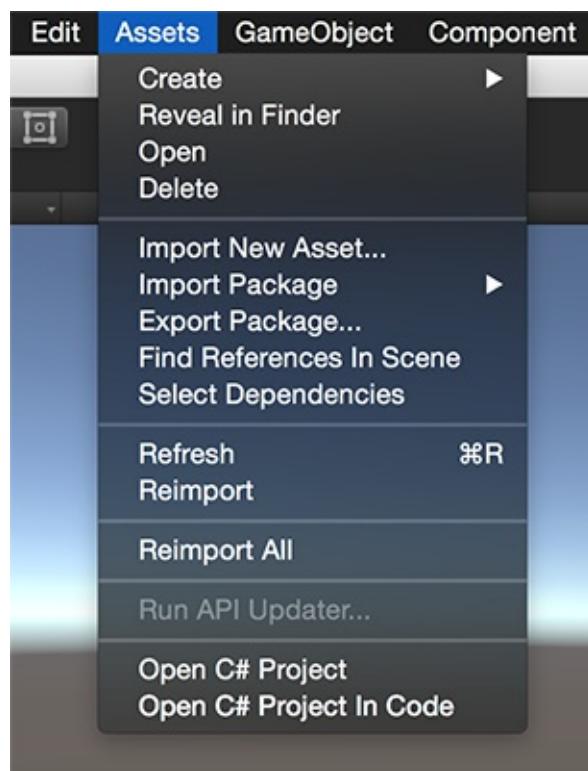
Toggle on `Enable Integration` and you are ready to get going.

Additionally, you can toggle `Output Messages To Console` which will echo output to the Unity console e.g. whether the debug port was found. This is useful for debugging any issues with the integration.

Clicking the `Write Workspace Settings` button will configure the workspace settings to filter out non-code assets created by Unity e.g. `.meta` files.

Step 3: Open the Project

Now available in the `Assets` menu is `Open C# Project In Code`. This will open the project in VS Code in the context of the root folder of your project. This enables VS Code to find your Unity project's solution file.



Tip: You probably want to leave a VS Code window open with the project context established. That way when you open a file from Unity it will have all the required context.

Editing Evolved

With the solution file selected, you are now ready to start editing with VS Code. Here is a list of some of the things you can expect:

- Syntax Highlighting
- Bracket matching

- IntelliSense
- Snippets
- CodeLens
- Peek
- Go-to Definition
- Code Actions/Lightbulbs
- Go to symbol
- Hover

Two topics that will help you are [Editing Evolved](#) and [C#](#). In the image below, you can see VS Code showing hover context, peeking references and more.

The screenshot shows a code editor window for a file named `LandingGear.cs`. The code defines a private enum `GearState` with two values: `Lowered` and `Raised`. The editor highlights the `m_State` variable in several places where it is used as a condition in if statements. A floating reference panel on the right side of the screen lists various ways the `m_State` variable is used throughout the project, such as assignments to `m_State` and comparisons against `GearState.Lowered` and `GearState.Raised`.

```

8
6 references UnityStandardAssets.Vehicles.Aeroplane.GearState
9 private enum GearState
LandingGear.cs c:\Users\Public\Documents\Unity Projects\Standard Assets Example Project\Assets\Standard Assets\Vehicles\Aircraft\Scripts
37
38     // Update is called once per frame
39     0 references
40     private void Update()
41     {
42         if (m_State == GearState.Lowered && m_Plane.Altitude >
43             raiseAtAltitude && m_Rigidbody.velocity.y > 0)
44         {
45             m_State = GearState.Raised;
46         }
47
48         if (m_State == GearState.Raised && m_Plane.Altitude <
49             lowerAtAltitude && m_Rigidbody.velocity.y < 0)
50         {
51             m_State = GearState.Lowered;
52         }

```

Extensions

The community is continually developing more and more valuable extensions for Unity. Here are some popular extensions that you might find useful. You can find more extensions in the VS Code [Extension Marketplace](#).

- [Debugger for Unity](#) - Debug your Unity projects in VS Code.
- [Unity Tools](#) - This extension adds extra functionality such as integrating Unity documentation and the Unity Asset Store with VS Code.

Next Steps

Read on to learn more about:

- [Editing Evolved](#) - find out more about the evolved editing features

- [Debugging](#) - how to use the debugger with your project
- [C#](#) - learn about the C# support in VS Code

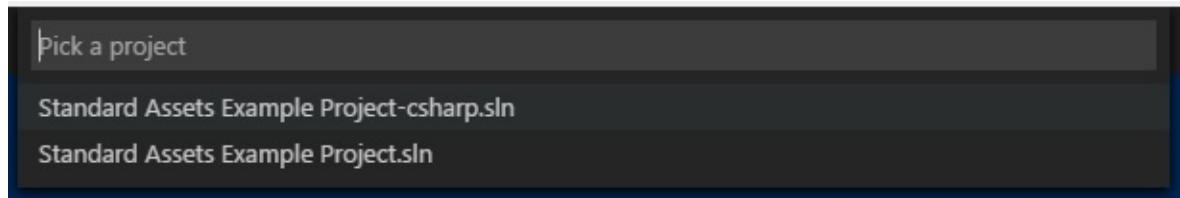
Common Questions

Q: I don't have IntelliSense.

A: You need to ensure that your solution is open in VS Code (not just a single file). Open the folder with your solution and you usually will not need to do anything else. If for some reason VS Code has not selected the right solution context, you can change the selected project by clicking on the OmniSharp flame icon on the status bar.

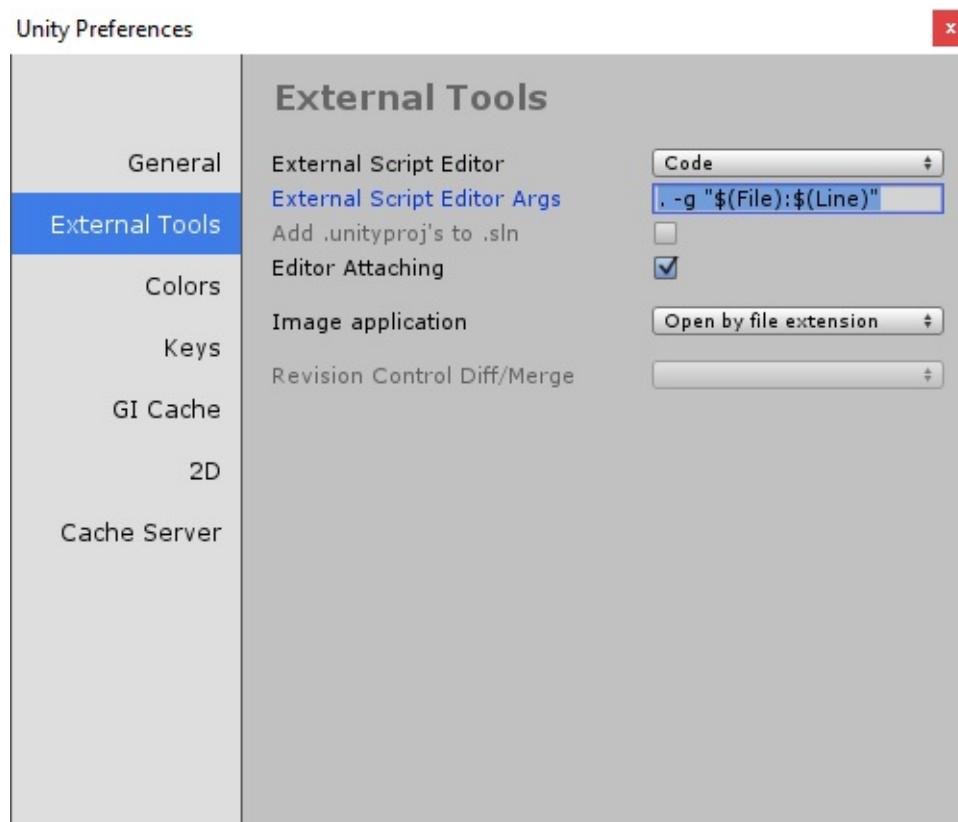


Choose the `-csharp` version of the solution file and VS Code will light up.



Q: How can I manually configure Editor Settings?

A: To set VS Code as the External Editor for Unity go to the **Edit > Preferences...** menu. From there move to the **External Tools** tab where you should see a screen like the following:



Click the browse button and set the VS Code executable (under `Program Files (x86)\Microsoft\VS Code\Code.exe`) as the External Script Editor.

This will enable Unity to launch VS Code whenever you open a script.

Q: I want to set the command line arguments when I launch VS Code.

A: The workflow between Unity and VS Code is much smoother if you reopen an existing editor window and pass in context so that the editor lands on the exact line number and column location (for example to navigate to a compile bug).

Note: This option is only available in the Windows version of Unity. For Mac users, the Unity plug-in described above handles setting up the command line arguments.

To configure Unity to pass this data to VS Code, set the Editor Args to be `. -g "$(File):$(Line)"`. You can see this step completed in the image above.

Q: How can I change the file exclusions?

A: Unity creates a number of additional files that can clutter your workspace in VS Code. You can easily hide these so that you can focus on the files you actually want to edit.

To do this, add the following JSON to your [workspace settings](#).

```
// Configure glob patterns for excluding files and folders.
"files.exclude": {
    "**/.git": true,
    "**/.DS_Store": true,
    "**/*.meta": true,
    "**/*.*.meta": true,
    "**/*.unity": true,
    "**/*.unityproj": true,
    "**/*.*.mat": true,
    "**/*.*.fbx": true,
    "**/*.*.FBX": true,
    "**/*.*.tga": true,
    "**/*.*.cubemap": true,
    "**/*.*.prefab": true,
    "**/Library": true,
    "**/ProjectSettings": true,
    "**/Temp": true
}
```

As you can see below this will clean things up a lot...

| Before | After |
|---|---|
| <ul style="list-style-type: none"> ▲ Scripts <ul style="list-style-type: none"> CameraSwitch.cs CameraSwitch.cs.meta LevelReset.cs LevelReset.cs.meta ParticleSceneControls.cs ParticleSceneControls.cs.meta PlaceTargetWithMouse.cs PlaceTargetWithMouse.cs.meta SlowMoButton.cs SlowMoButton.cs.meta | <ul style="list-style-type: none"> ▲ Scripts <ul style="list-style-type: none"> CameraSwitch.cs LevelReset.cs ParticleSceneControls.cs PlaceTargetWithMouse.cs SlowMoButton.cs |

Q: VS Code did not go the correct position in my file.

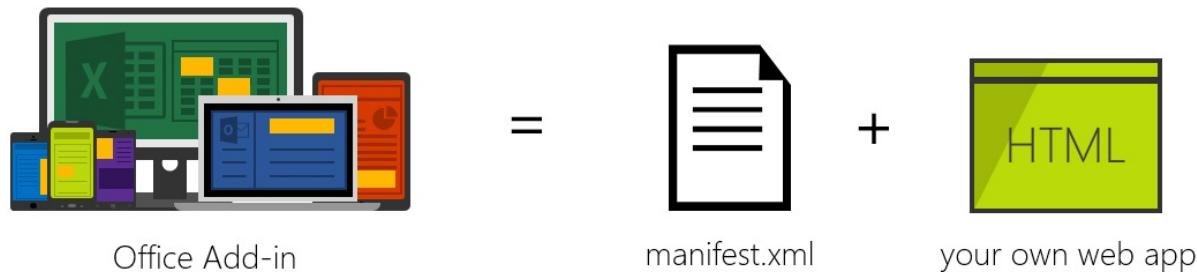
A: For OS X, make sure you have installed the Unity plug-in described above. For Windows, ensure you have set the additional command line arguments when you launch VS Code from Unity.

Q: How can I debug Unity?

A: Install the [Debugger for Unity](#) extension.

Office Add-ins with VS Code

[Office Add-ins](#) run inside an Office application and can interact with the contents of the Office document using the rich JavaScript API.



Under the hood, an Office Add-in is simply a web app that you can host anywhere. Using a `manifest.xml` file, you tell the Office application where your web app is located and how you want it to appear. The Office application takes care of hosting it within Office.

Prerequisites

To run the Yeoman Office Add-in generator, you need a few things:

- NPM
 - Bower
 - Yeoman
 - Yeoman Office generator
 - Gulp
 - TSD

These should all be installed globally which you can do with a single command:

```
npm install -g bower yo generator-office gulp tsd
```

Before running the generator, we recommend you first create a subfolder and run the generator from there. Yeoman generators typically create files in the current working directory where it is run. The following command will create a new folder and navigate into it:

```
mkdir myaddin && cd $
```

Create an Office Add-in

Once you have a place to put your Add-in, you can now create the Add-in. Use the Yeoman generator [office](#) to create one of three Add-ins: mail, content or task pane.

The generator is designed to be run from within the directory where you want to scaffold the project so ensure you set the current directory appropriately.

To run the generator:

yo office

The generator will prompt you for the Add-in name, relative folder where the project should be created, the type of Add-in and the technology you want to use to create the Add-in.

ac@Andrews-MBP ~/Dev/Scratch/myaddin yo office



- ? Project name (display name): My First Add-in
- ? Root folder of project? Default to current directory (/Users/ac/Dev/Scratch/myaddin), or specify relative path from current (src / public):
- ? Office project type: Mail Add-in (read & compose forms)
- ? Technology to use: HTML, CSS & JavaScript
- ? Supported Outlook forms: (Press <space> to select)
 - >● E-Mail message – read form
 - E-Mail message – compose form
 - Appointment – read form
 - Appointment – compose form

Update the Add-in Manifest.xml

This is done using an XML file that tells the Office application about the Add-in including details such as:

- Type of Add-in (e.g. Mail / Task Pane / Content)
 - Name & description of the Add-in
 - Permissions the Add-in requires
 - URL of the web app hosting the Add-in

Open the project in Visual Studio Code by entering the following on the command line from within the same folder where you ran the generator:

```
code .
```

```

manifest.xml - myaddin - Visual Studio Code
EXPLORE manifest.xml
WORKING FILES
MYADDIN
app
bower_components
content
images
node_modules
scripts
.bowerrc
bower.json
gulpfile.js
manifest.xml
package.json

1 <?xml version="1.0" encoding="UTF-8"?>
2 <OfficeApp xmlns="http://schemas.microsoft.com/office/appforoffice/1.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:type="TaskPaneApp">
3   <Id>c50be0c3-ce79-4252-933c-f75725b9a326</Id>
4   <Version>1.0.0.0</Version>
5   <ProviderName>[Provider name]</ProviderName>
6   <DefaultLocale>en-US</DefaultLocale>
7   <DisplayName DefaultValue="My Office Project" />
8   <Description DefaultValue="[Task pane Add-in description]" />
9   <Hosts>
10    <Host Name="Workbook" />
11    <Host Name="Presentation" />
12    <Host Name="Project" />
13    <Host Name="Document" />
14  </Hosts>
15  <DefaultSettings>
16    <SourceLocation
      DefaultValue="https://localhost:8443/app/home/home.html" />
17  </DefaultSettings>
18  <Permissions>ReadWriteDocument</Permissions>
19 </OfficeApp>

```

Open the `manifest.xml` file that was created by the `office` generator and locate the `SourceLocation` node. Update this URL to the URL where you will host the Add-in.

Tip: If you are using an Azure Web App as the host, the URL will look something like `https://[name-of-your-web-app].azurewebsites.net/[path-to-add-in]`. If you are using the self-hosted option listed above, it will be `https://localhost:8443/[path-to-add-in]`.

Hosting Your Office Add-in Development

Office Add-ins must be served via HTTPS; the Office application will not load a web app as an Add-in if it is HTTP. To develop, debug and host the Add-in locally, you need a way to create and serve a web app locally using HTTPS.

Self-Hosted HTTPS Site

One option is to use the [gulp-webserver](#) plug-in. The Office generator will add this to the `gulpfile.js` as a task named `serve-static` for the project that's generated.

Start the self-hosted webserver using the following statement:

```
gulp serve-static
```

This will start a HTTPS server at <https://localhost:8443>.

Tip: You can also run the task from within VS Code by pressing

```
kb(workbench.action.showCommands)
```

 and then typing `Run Task` followed by

```
kbstyle(Enter)
```

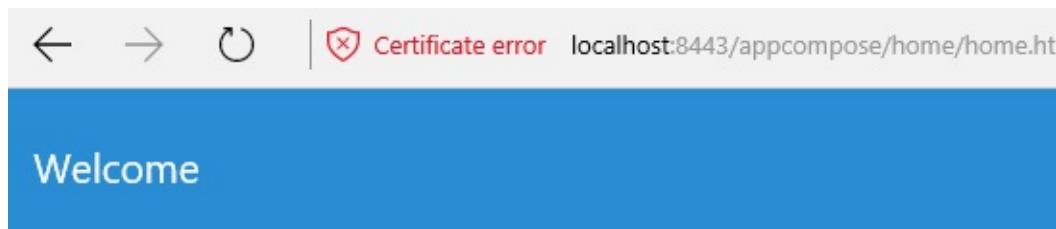
 which will list all available tasks. Selecting `serve-static` and pressing

```
kbstyle(Enter)
```

 will execute the task.

Running the Add-in

At this point, you can point your browser at the Add-in URL and see it running. Assuming you hosted it locally, just type that URL into your browser.



Add home screen content here.

For example:

```
Set subject    Get subject    Add yourself to recipients list
```

[Find more samples online...](#)

Tip: The generated Add-in comes with a self-signed certificate and key; you will want to add these to your trusted authority list of certificates so your browser does not issue certificate warnings like you see in the image above.

Refer to the [gulp-webserver](#) documentation if you want to use your own self-signed certificates.

Refer to [this KB article #PH18677](#) for instructions on how to trust a certificate in OS X Yosemite.

Now that you've tested your Add-in locally, let's add it into Office.

Use VS Code to Develop Your Office Add-in!

VS Code is a great tool to help you develop your custom Office Add-ins regardless if they are for Outlook, Word, Excel, PowerPoint and run in the web clients, Windows clients, iOS clients or on OS X!

JavaScript Project Support

The Office generator will create a `jsconfig.json` file when it creates your project. This is the file that VS Code will use to infer all the JavaScript files within your project and save you from having to include the repetitive `/// <reference path="..../App.js" />` directives.

Learn more about the `jsconfig.json` file on the [JavaScript language](#) page.

JavaScript IntelliSense Support

In addition, even if you are writing plain JavaScript, VS Code can use TypeScript type definition files (`*.d.ts`) to provide additional IntelliSense support. The Office generator adds a `tsd.json` file to the created files with references to all third-party libraries used by the project type you selected.

All you have to do after creating the project using the Yeoman Office generator is run the following command to download the referenced type definition files:

```
tsd install
```

Learn more about the JavaScript IntelliSense support provided by VS Code with TypeScript on the [JavaScript language](#) page.

JavaScript Peek Definition

You can also get details on objects, properties and methods you are referencing within your Office Add-in using VS Code capabilities like Peek Definition, Go to Definition and Find all References by simply right-clicking in any JavaScript file.

Learn more about the Rich Editing Support in VS Code on the [JavaScript language](#) page.

Debugging your Office Add-in

VS Code does not currently support client-side debugging. To debug your client-side Add-in, you can use the Office web clients and open the browser's developer tools and debug the Add-in just like any other client-side JavaScript application.

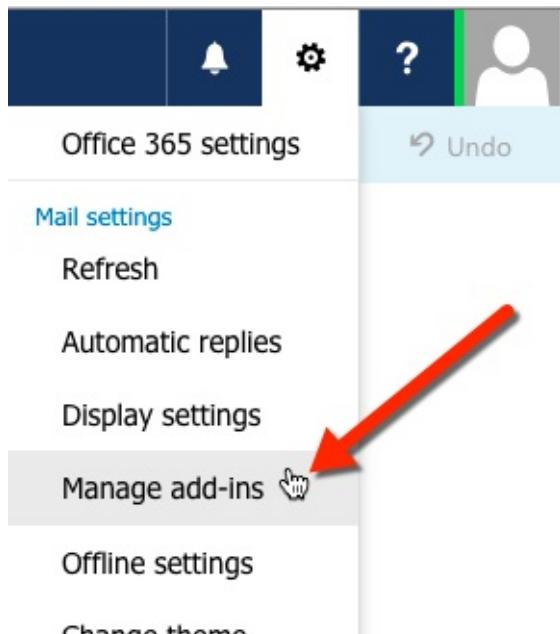
If you are using Node.js or ASP.NET Core for server-side logic that supports your Office Add-in, refer to the [Debugging](#) page to configure VS Code for debugging either of these runtimes.

Install the Add-in

Office Add-ins must be installed, or registered, with the Office application in order to load. This is done using the `manifest.xml` file you modified earlier.

Side Loading Mail Add-ins

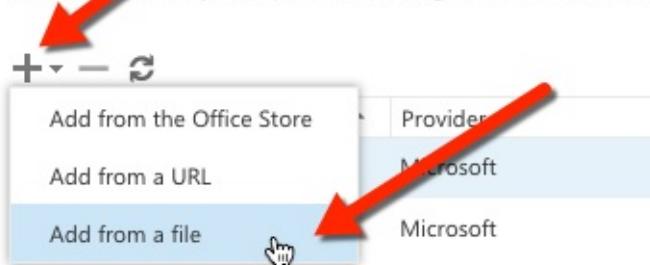
Mail Add-ins can be installed within the Outlook Web App. Browse to your Outlook Web App (<https://mail.office365.com>) and login. Once logged in, click the gear icon in the top-right section and select **Manage add-ins**:



On the **Manage add-ins** page, select the + icon and then select **Add from a file**.

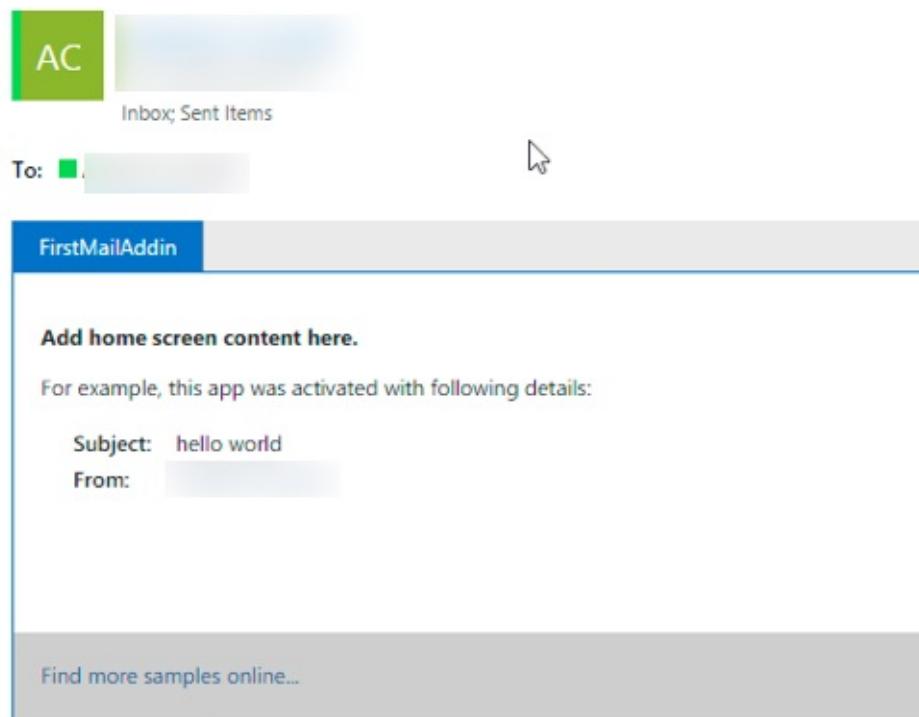
Manage add-ins

Add-ins are built by third parties and bring additional features to your Office apps.

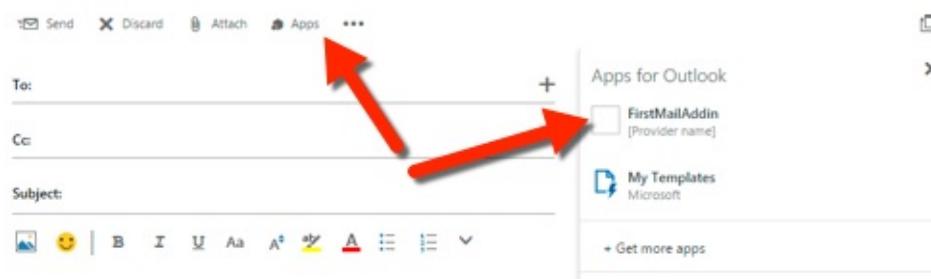


Locate the `manifest.xml` file for your custom Add-in and install it, accepting all prompts when installing it.

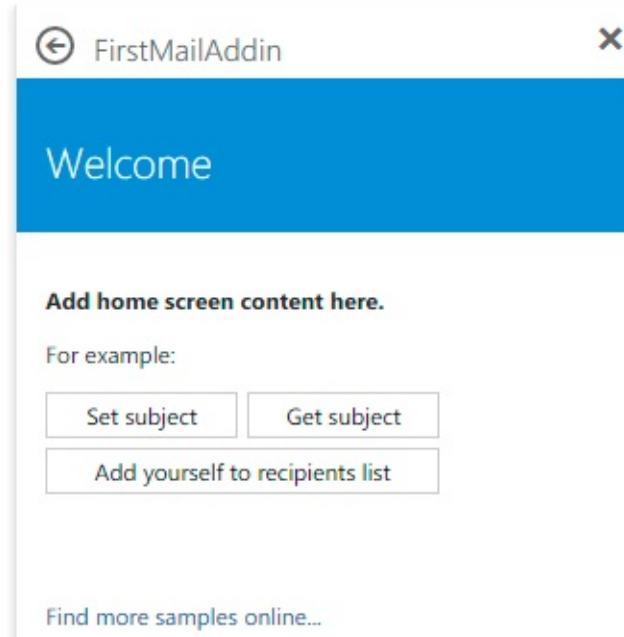
Once that's done, select an existing email and you will see a horizontal bar below the email header that includes the Add-in:



Next try creating an email, click the **Add-ins** or **Apps** menu item to get the Task Pane to appear:

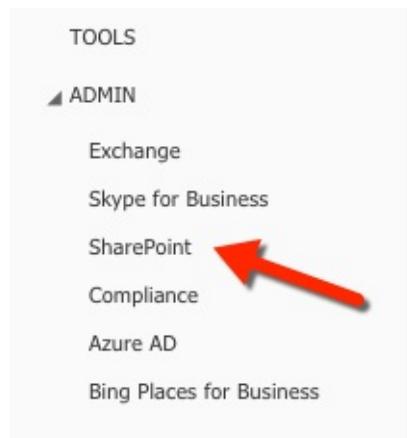


Select the Add-in and see it appear in the Task Pane:



Deploying Add-ins to the Office 365 Tenancy's App Catalog

All Office Add-ins (including Mail Add-ins) can be installed from your Office 365 tenancy's App Catalog site. Log in to your [Office 365 Portal](#). In the left-hand navigation, towards the bottom, select the **Admin / SharePoint** option:



From the **SharePoint Admin Center**, select the **Apps** option in the left-hand menu and then select the **App Catalog**. On the **App Catalog** page, select the **Apps for Office** option and upload the `manifest.xml` file.

BROWSE PAGE



Home

App Catalog

- Recent
- Apps for SharePoint
- Apps for Office** →
- App Requests
- Site Contents



Distribute apps for SharePoint

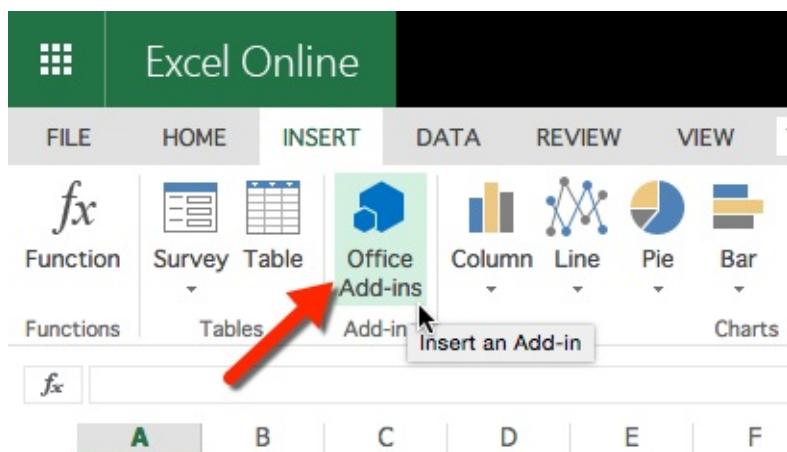


Distribute apps for Office

Install Content & Task Pane Add-ins in Word / Excel / PowerPoint

Depending on the type of Add-in you created, you can add it to one of the Office applications. Task Pane and Content Add-ins can be installed in Word, Excel & PowerPoint. Mail Add-ins can be installed in Outlook.

To install an Add-in within an Office application, select the **Insert** tab and click the **Office Add-ins** button, as shown here using the Excel Web App:



Using the Office Add-ins dialog you can select Add-ins you've uploaded to your Office 365 tenancy's App Catalog (*listed under My Organization*) or acquire Add-ins from the Office Store.

Next Steps

Check out the other pages on the VS Code site to find out how you can leverage more capabilities of the editor when creating custom Office Add-ins:

- [Language Overview](#) - You can write Office Add-ins in many languages. Find out what VS Code has to offer.
- [The Basics](#) - Just starting out with VS Code? This is worth reviewing.
- [Editing Evolved](#) - Review all the ways VS Code can help you in editing.
- [Node.js](#) - Find out more about our Node.js support.

Common Questions

Q: Can I create an Office Add-in with the generator and use VS Code regardless of the language or client-side framework?

A: Yes, you can. You can use pure HTML, Angular, Ember, React, Aurelia... anything you like!

Q: Can I use TypeScript to create my Office Add-in?

A: Absolutely and VS Code has great support for [TypeScript](#)!