

SD466922

AutoLISP Has a New Home: Get to Know Visual Studio Code

Matt Worland
ECE Design

Learning Objectives

- Learn how to implement Visual Studio Code for AutoLISP.
- Learn how to develop code efficiently with IntelliSense and Code Snippets.
- Learn how to create custom code snippets.
- Learn how to debug AutoLISP code effectively.

Description

We have a new IDE! Let's familiarize ourselves with the most significant update to AutoLISP since receiving VL- functions. In this session, you will learn how to create code efficiently using Code Snippets and IntelliSense while debugging your new or existing code in AutoCAD using Visual Studio Code. Let's take advantage of what Visual Studio Code has to offer!

Speaker

I enjoy sharing my knowledge to elevate Design and Engineering teams to take their projects to the next level. I have been customizing and programming in AutoCAD products since the mid-90s. My passion for helping others was a perfect fit as an AutoCAD All-Star Mentor and a former member of Autodesk User Group International AUGI board of directors. From Drafter to CAD Manager, I have increased the efficiencies of the engineering and design teams I worked with. I now support, train, and customize Plant related products as a Senior Applications Engineer and Developer at ECE Design. When I am not helping others or learning new tricks in the office, I enjoy mountain biking, paddle boarding, or hiking in the mountains with my wife and children.

Table of Contents

AutoLISP Has a New Home: Get to Know Visual Studio Code	1
Visual Studio Code	3
Why use Visual Studio Code.....	3
Install Wizard.....	4
Installing Manually	9
Visual Studio Interface	12
Workspace & AutoLISP Project Manager	14
Keyboard Shortcuts	15
Shortcuts to Explore	16
Adding New Shortcuts	17
IntelliSense & Code Snippets	19
Adding New Snippets	20
Snippets Structure	21
Creating Files.....	23
Formatting.....	23
Debugging.....	24
Attach	24
Launch.....	25
Define Configurations.....	25
Debug Side Bar	27
Breakpoints	28
Debug Console.....	28
Debug Commands	30
Source Control Management	31
Git Setup	31
Committing Changes.....	33
Remote Repositories.....	34
Open Source Project.....	34
Additional Information	34

Visual Studio Code

Visual Studio Code is a free, open-source licensed; cross-platform source code editor developed by Microsoft. A Stack Overflow Survey ranked VS Code as the most popular developer environment tool. As it grows in popularity, more and more extensions have been added, including the AutoCAD AutoLISP extension. While the VL IDE has many tools, it is built on old technologies and has not been updated for over a decade. While not all VL IDE functions have been added to the AutoLISP extension, these first iterations have great features to benefit your AutoLISP development.

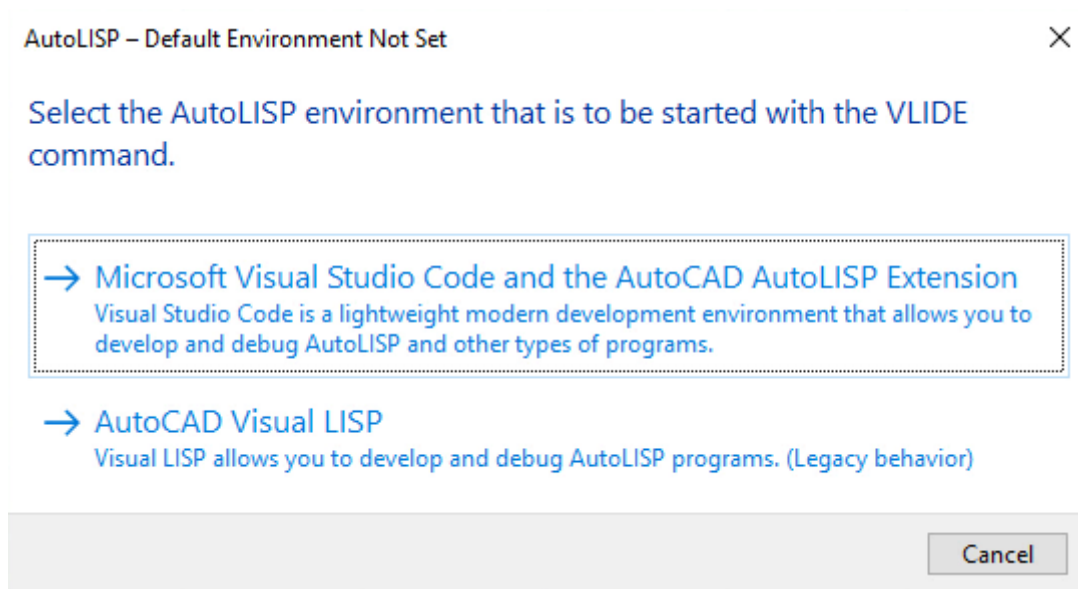
This handout covers many concepts in using Visual Studio Code. We work through installing and setting up VS Code to debugging AutoLISP files.

Why use Visual Studio Code

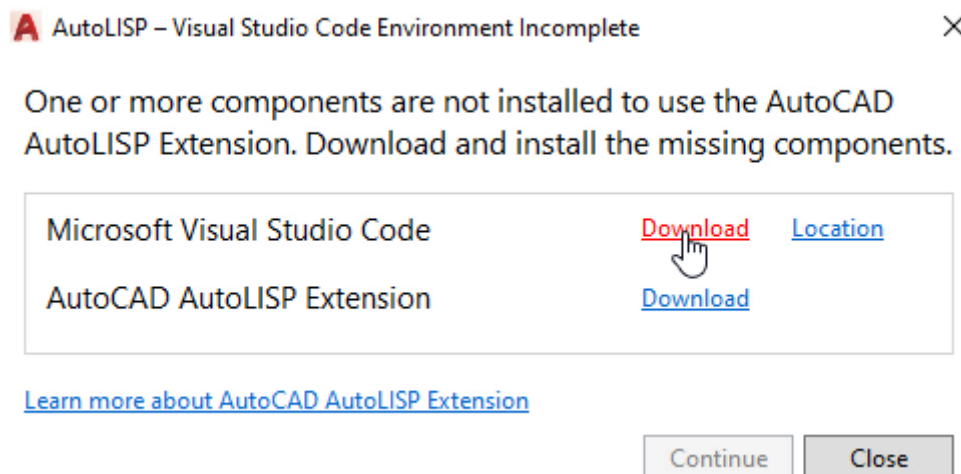
- IntelliSense & Code Snippets significantly reduce coding time!
- Keyboard Shortcuts help you navigate and update your code.
- Extensions enhance your VS Code experience.
- Source Control manages your code in a safe environment.
- The AutoLISP extension is open-source, allowing you to enhance the extension for all!
- It is the future. As stated in the help documents, the VL IDE may be removed in a future release.

Install Wizard

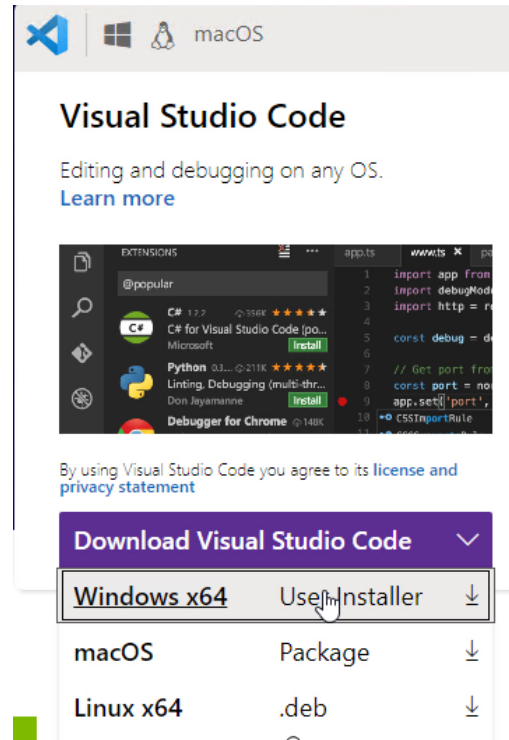
Setting up AutoCAD to use Visual Studio Code is an easy process. If you have not chosen a default editor, AutoCAD prompts you to choose one. To use Visual Studio Code, you select the top option.



Another dialog appears if you do not have Visual Studio Code and the AutoLISP Extension downloaded. Choose the download link for Visual Studio Code.



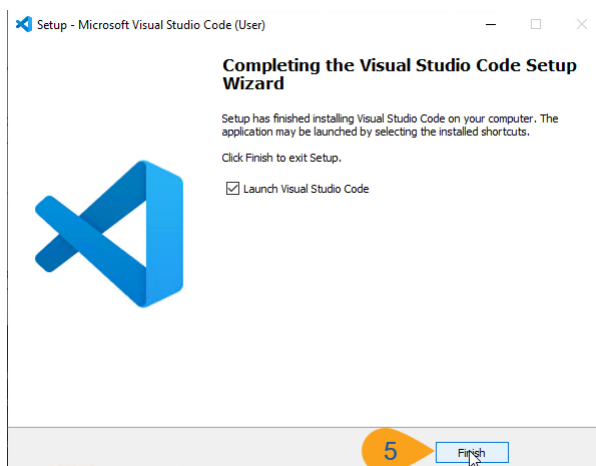
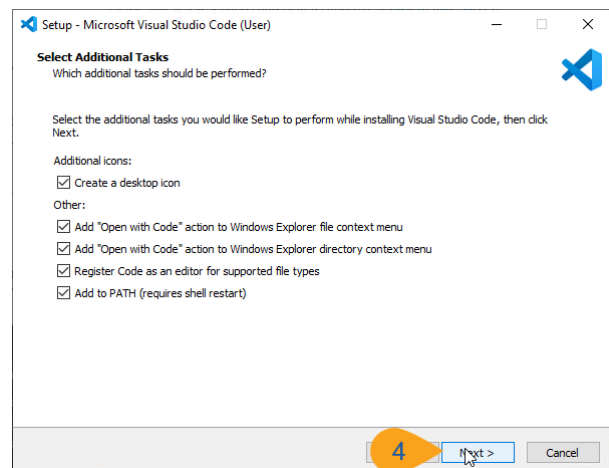
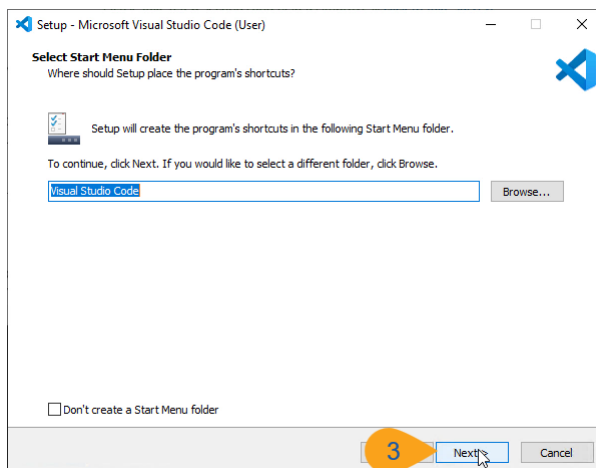
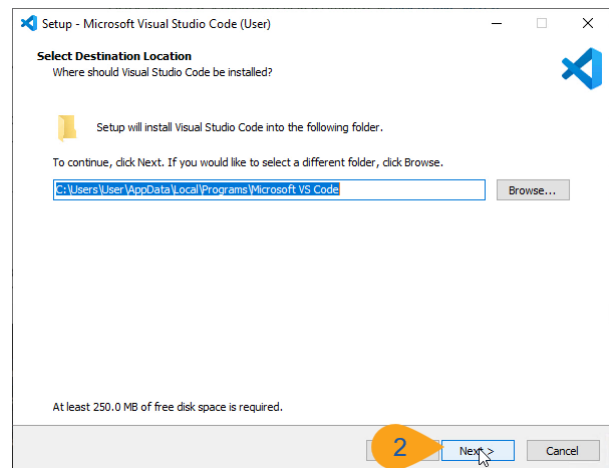
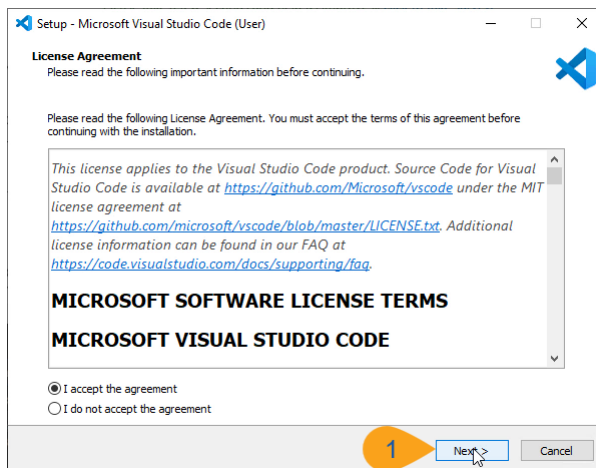
This navigates you out to Microsoft's download site. Choose the download that best fits your Operating System. Below, I show the area for downloading onto a Windows computer



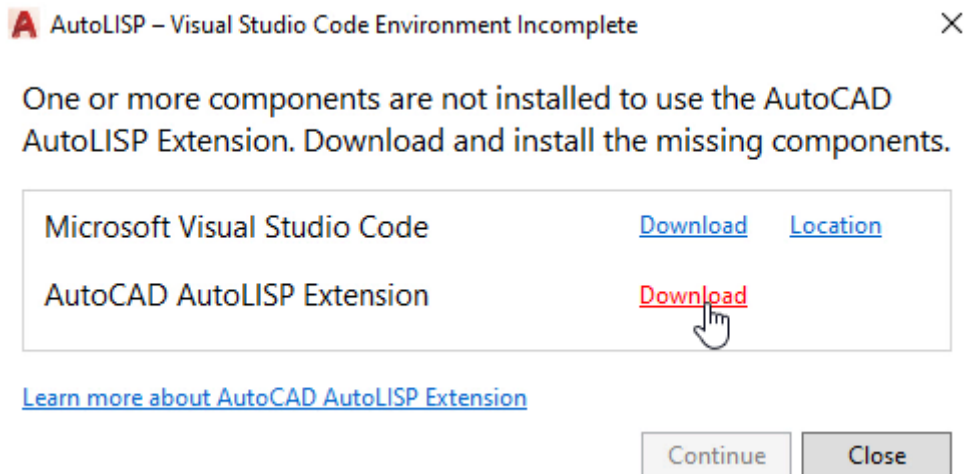
It directs you to another page to choose the correct version, architecture, and download type.



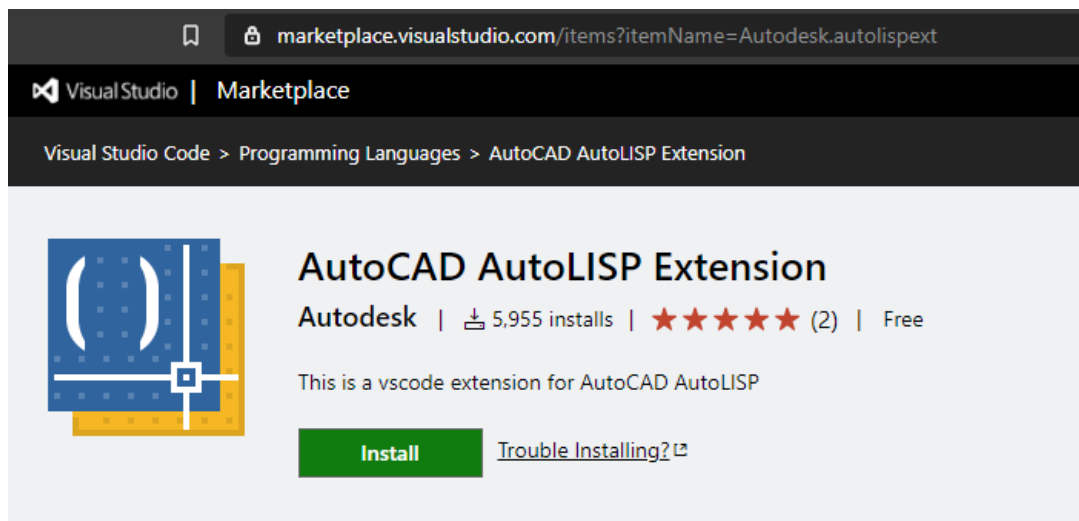
Once downloaded, follow the prompts for installation.



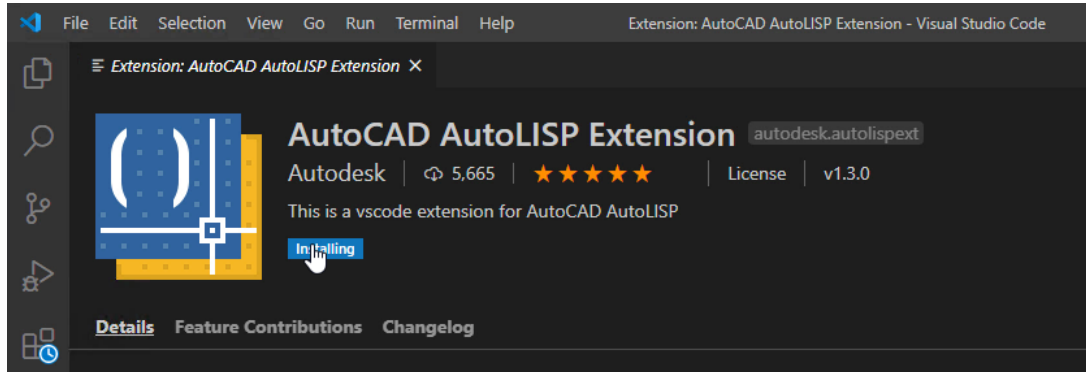
Now that VS Code is installed, we return to AutoCAD to download and install the AutoCAD AutoLISP Extension.



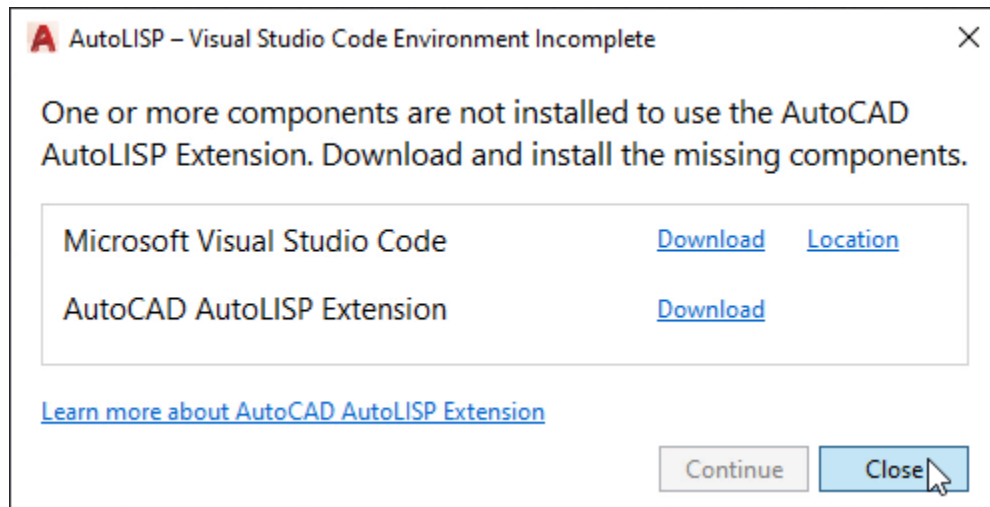
A new browser window appears with an install button.



A new prompt appears to open Visual Studio Code and you to install it there. Choose the Install button to add the extension to VS Code.



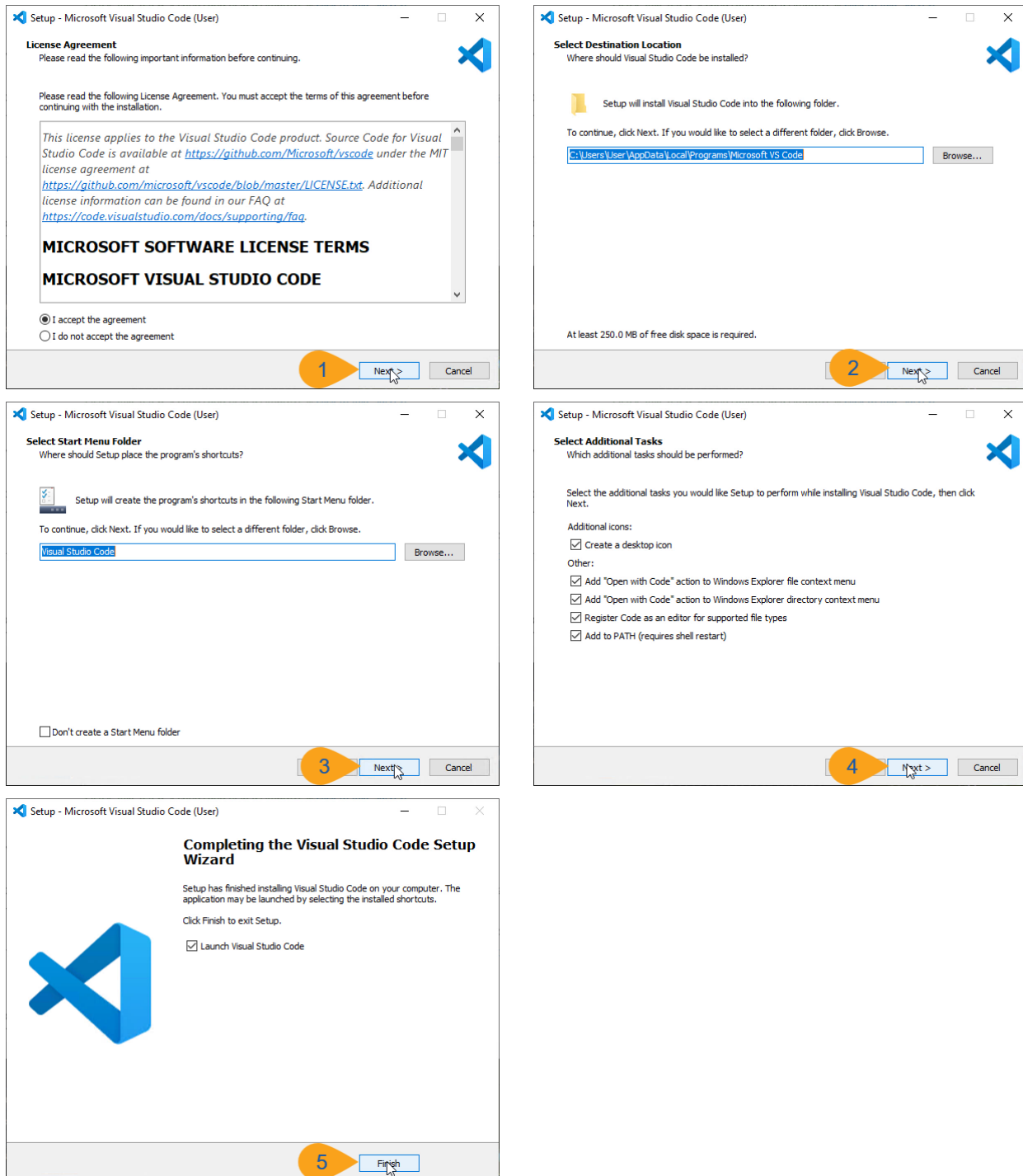
We have now installed Visual Studio Code and the AutoLISP Extension. You may close VS Code and return to AutoCAD. Here we can close out the download dialog.



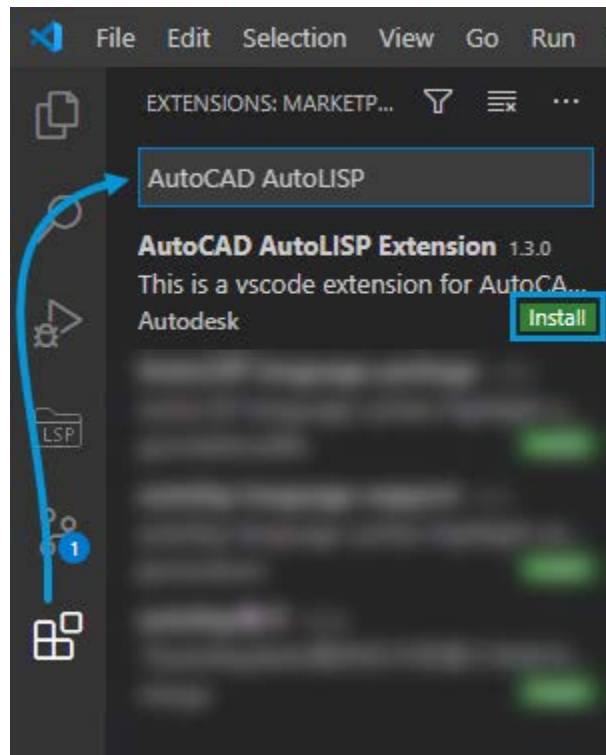
AutoCAD sets **LISPSYS = 1** when you choose VS Code for your default editor using their wizard. If you compile your code to be used in versions before AutoCAD 2021, you need to modify the variable to **LISPSYS = 2**

Installing Manually

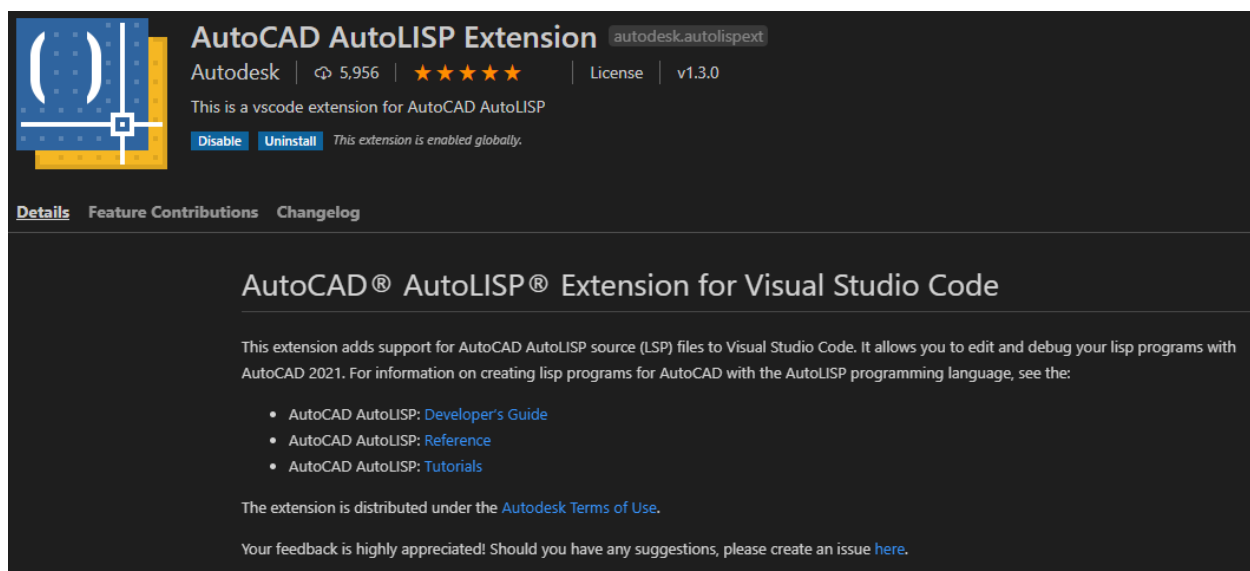
To download manually, go here: <https://aka.ms/win32-x64-user-stable>
Once downloaded, follow the prompts for installation.



After launching VS Code, you need to use the Extensions Panel **Ctrl + Shift + X** to search for and install the AutoCAD AutoLISP Extension. There are several AutoLISP extensions available; make sure to select the extension from Autodesk.



The AutoLISP Extension tab appears, and you can get additional information about using the extension, creating GitHub issues, links to the Autodesk AutoLISP help, and more.

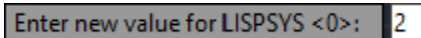


With Visual Studio Code and the AutoLISP extension installed, we need to let AutoCAD know we are ready to use our new editor. To do this, we need to change the **LISPSYS** variable.

To maintain and compile your code for use in AutoCAD 2020 and previous releases, you must use **LISPSYS = 2**

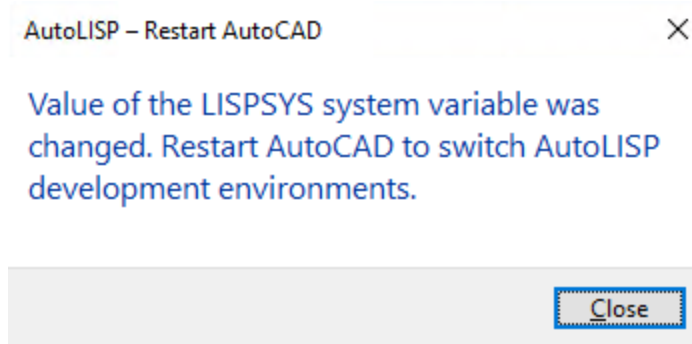
If you do not need to compile code for AutoCAD 2020 and previous releases, you can set **LISPSYS = 1**

At the AutoCAD command line, type **LISPSYS** and set the value to either **1** or **2**.



Enter new value for LISPSYS <0>: 2

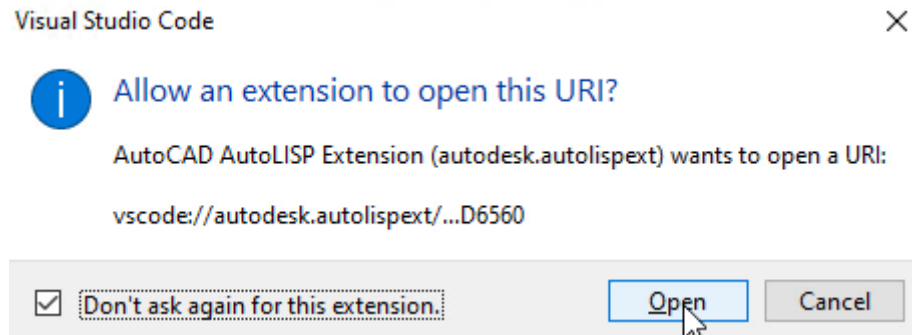
You must restart AutoCAD when switching AutoLISP development environments.



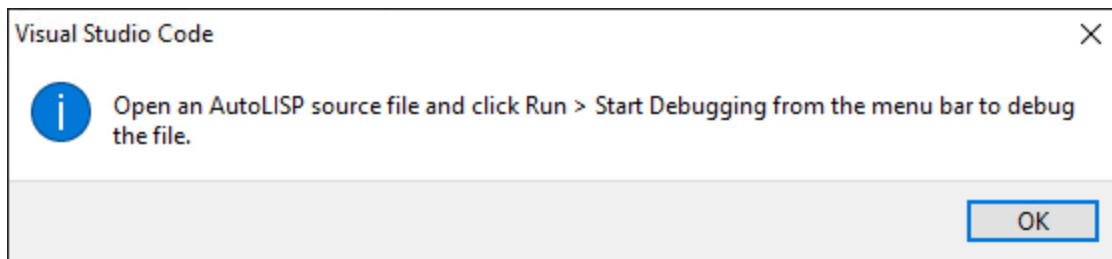
For additional information on the LISPSYS variable, please visit AutoCAD's cloud help:
<https://knowledge.autodesk.com/support/autocad/learn-explore/caas/CloudHelp/cloudhelp/2021/ENU/AutoCAD-Core/files/GUID-1853092D-6E6D-4A06-8956-AD2C3DF203A3-htm.html>

Visual Studio Interface

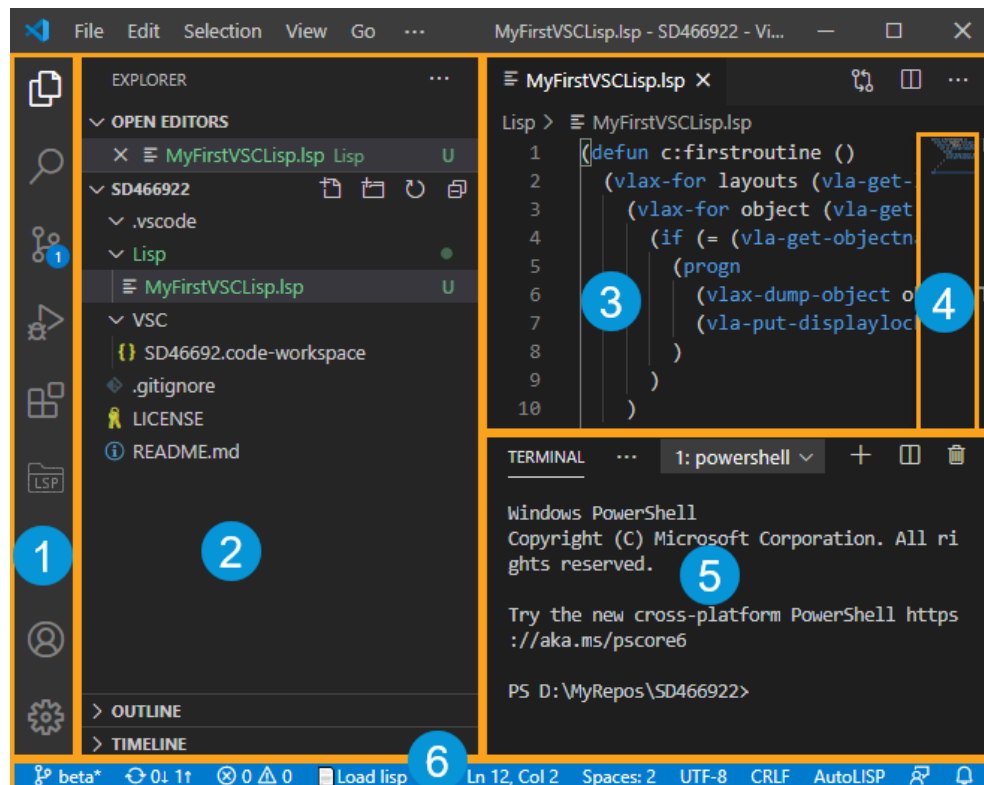
You type **VLIDE** or **VLISP** at the command line to launch Visual Studio Code from within AutoCAD. You are greeted with the dialog shown below the first time you start Visual Studio Code with the AutoCAD AutoLISP Extension installed. You may check the 'Don't ask again for this extension' box to start VS Code with an extension message each time it starts.



When you start VS Code from within AutoCAD, you receive the following message that reminds you how to debug your files.



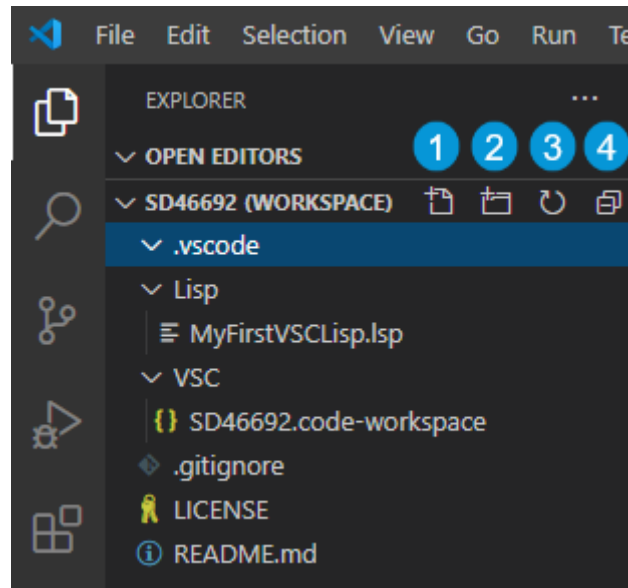
Let's review the interface of Visual Studio Code. On the next page are the main areas to get to know. Please visit <https://code.visualstudio.com/docs/getstarted/userinterface> for in-depth and up to date information on the Visual Studio Code interface.



- 1) Activity Bar: Choose the main activity like opening a file, debugging your code, or installing an extension.
- 2) Side Bar: Displays what you can do with the selected activity
- 3) Editor area: This shows the files you are editing
- 4) MiniMap: An overview of the currently opened file in the Editor Area
- 5) Panels: This shows the Debug Console, Terminal, and other panels
- 6) Status Bar: Displays the current statuses and information about the current session.

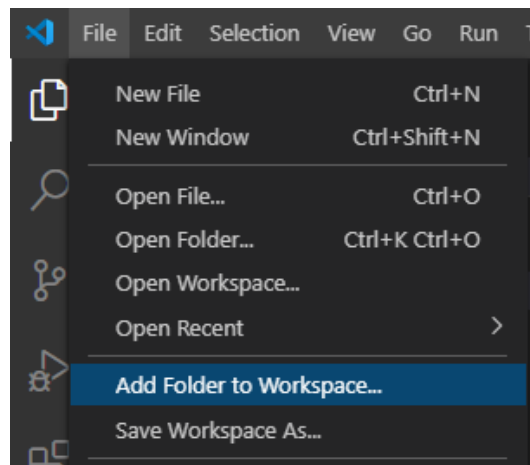
Workspace & AutoLISP Project Manager

You can manage your files using Folders, Workspaces, or AutoLISP Projects. Unless you have previous AutoLISP projects, you can use the native VS Code Folders or Workspace functionality. Using a workspace gives the additional benefits of adding other file types like DCL and managing your code in multiple folders. Visual Studio Code can also save specific settings to your workspace. This allows you to have individual VS Code settings for specific workspaces.

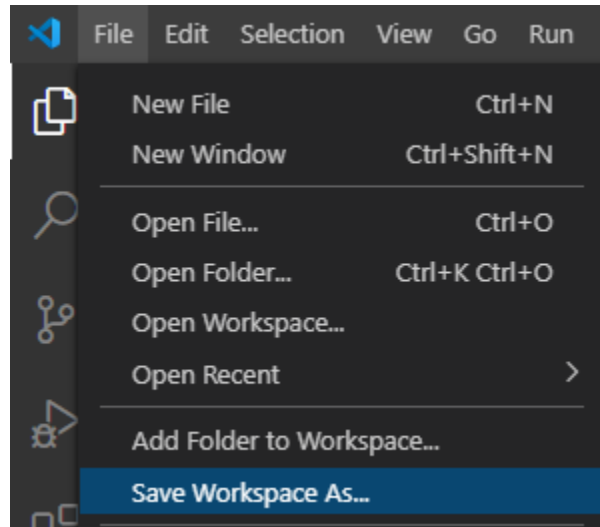


- 1) Add new files to the selected folder.
- 2) Add new folders to the selected folder
- 3) Refresh files and folders listed in the Explorer bar
- 4) Collapse all folders listed in the Explorer bar

You can add multiple folders to your workspace regardless of their root level. To do this, use the [Add Folder to Workspace...](#) command from the File Menu.



Be sure to save your workspace after adding new folders.



Keyboard Shortcuts

Keyboard shortcuts are still the fastest way to run AutoCAD. VS Code offers similar capabilities to quickly navigate and manipulate your AutoLISP files. Visual Studio Code has many built-in keyboard shortcuts. To view the list, use the keyboard shortcut **Ctrl + K, Ctrl + S**

Keyboard Shortcuts X			
Type to search in keybindings			
Command	Keybinding	When	Source
Add Cursor Above	Ctrl + Alt + UpArrow	editorTextFocus	Default
Add Cursor Below	Ctrl + Alt + DownArrow	editorTextFocus	Default
Add Cursors to Line Ends	Shift + Alt + I	editorTextFocus	Default
Add Line Comment	Ctrl + K Ctrl + C	editorTextFocus && !editorReadOnly	Default
Add Selection To Next Find Match	Ctrl + D	editorFocus	Default
Auto Fix...	Shift + Alt + .	editorTextFocus && !editorReadOnly && support...	Default
Calls: Show Call Hierarchy	Shift + Alt + H	editorHasCallHierarchyProvider	Default
Cancel Selection Anchor	Escape	editorTextFocus && selectionAnchorSet	Default
Change All Occurrences	Ctrl + F2	editorTextFocus && editorTextFocus && !editor...	Default
Change Language Mode	Ctrl + K M	—	Default
Close Window	Ctrl + Shift + W	—	Default
Close Window	Ctrl + W	!editorIsOpen && !multipleEditorGroups	Default

Use this link to view a printable list: <https://code.visualstudio.com/shortcuts/keyboard-shortcuts-windows.pdf>

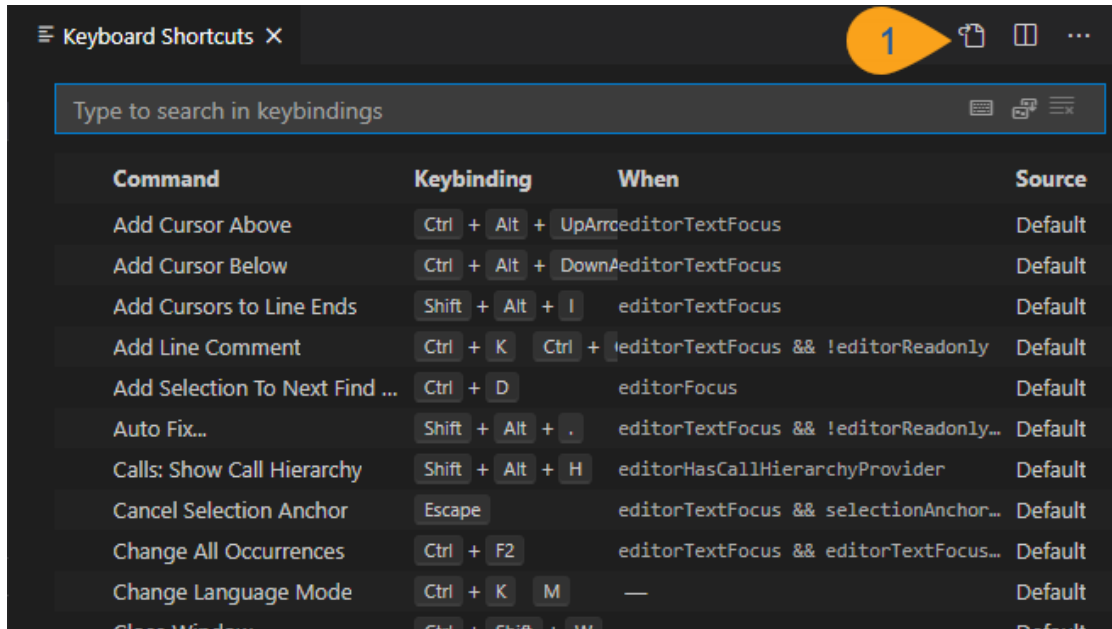
Shortcuts to Explore

There are quite a few shortcuts that are beneficial to efficiently changing your code. See the list below of shortcuts that I find very beneficial.

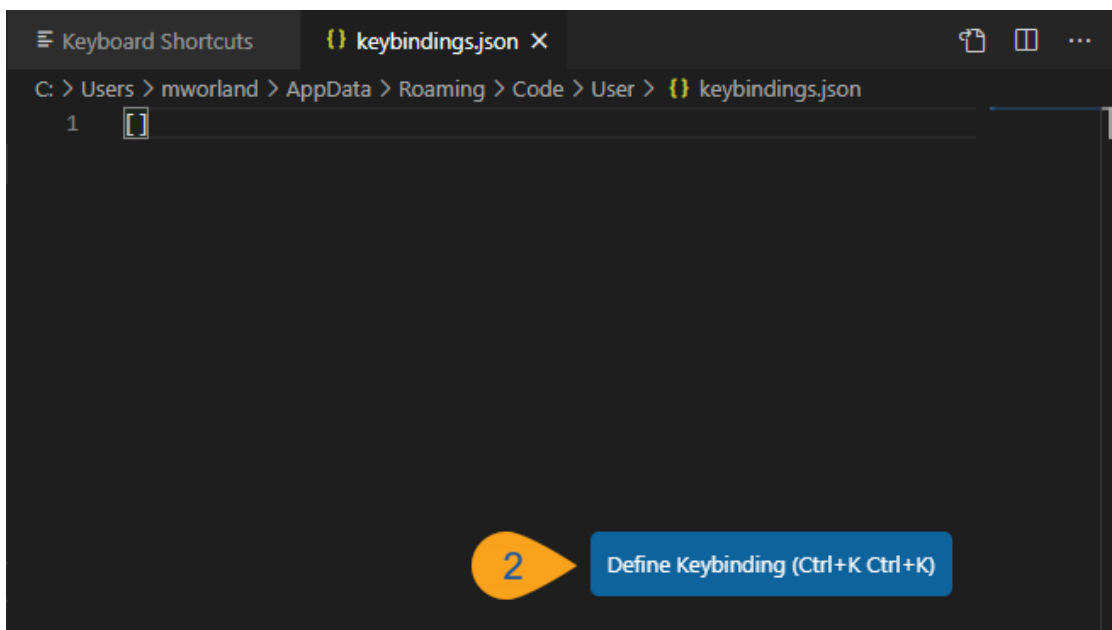
General	
Ctrl+Shift+P, F1	Show Command Palette
Ctrl+P	Quick Open, Go to File...
Basic editing	
Alt+ ↑ / ↓	Move line up/down
Shift+Alt + ↓ / ↑	Copy line up/down
Ctrl+Shift+K	Delete line
Ctrl+Enter	Insert line below
Ctrl+Shift+\	Jump to matching bracket
Ctrl+Shift+[Fold (collapse) region
Ctrl+Shift+]	Unfold (uncollapse) region
Ctrl+/	Toggle line comment
Shift+Alt+A	Toggle block comment
Multi-cursor and selection	
Alt+Click	Insert cursor
Shift+Alt+I	Insert cursor at end of each line selected
Ctrl+L	Select current line
Ctrl+Shift+L	Select all occurrences of current selection
Ctrl+F2	Select all occurrences of current word
Shift+Alt + (drag mouse)	Column (box) selection
Navigation	
Ctrl+P	Go to File...
Search and replace	
Alt+C / R / W	Toggle case-sensitive / regex / whole word
Rich languages editing	
Shift+Alt+F	Format document
Ctrl+K Ctrl+F	Format selection
File management	
Ctrl+K S	Save All
Ctrl+K Ctrl+W	Close All
Ctrl+Shift+T	Reopen closed editor
Ctrl+K P	Copy path of active file
Ctrl+K R	Reveal active file in Explorer
Display	
F11	Toggle full screen
Ctrl+K Z	Zen Mode (Esc Esc to exit)
Debug	
F9	Toggle breakpoint
F5	Start/Continue
Shift+F5	Stop
F11 / Shift+F11	Step into/out
F10	Step over

Adding New Shortcuts

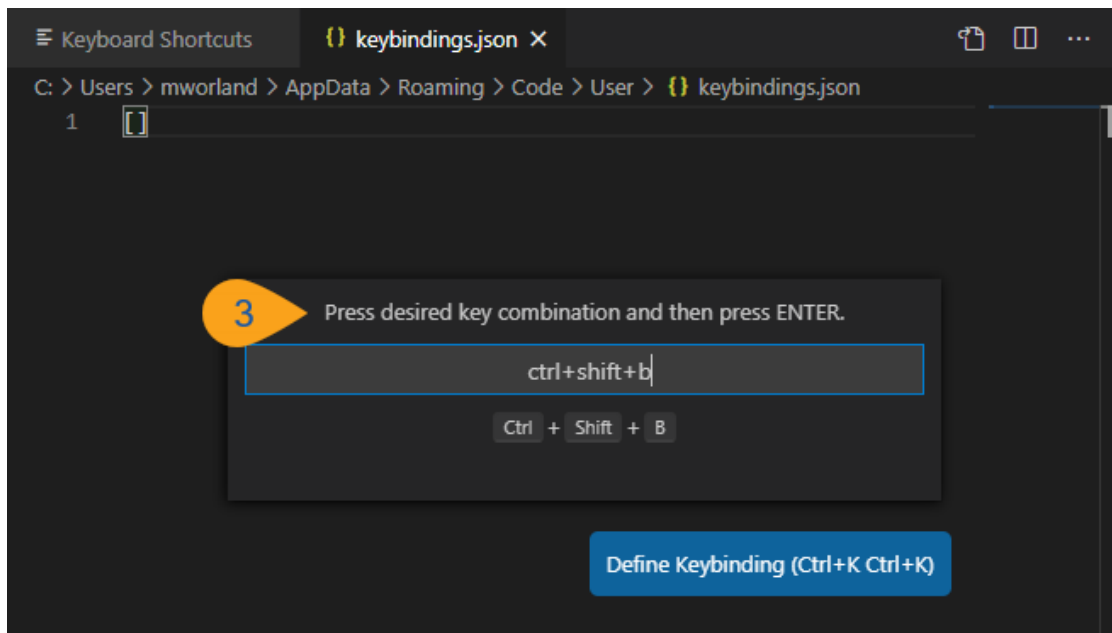
We can add new shortcuts to Visual Studio Code. Let's look at adding a new shortcut, which allows us to select all code between matching brackets. We use several shortcuts to create a new shortcut. Start with **Ctrl + K**, **Ctrl + S** to open the existing keyboard shortcuts.



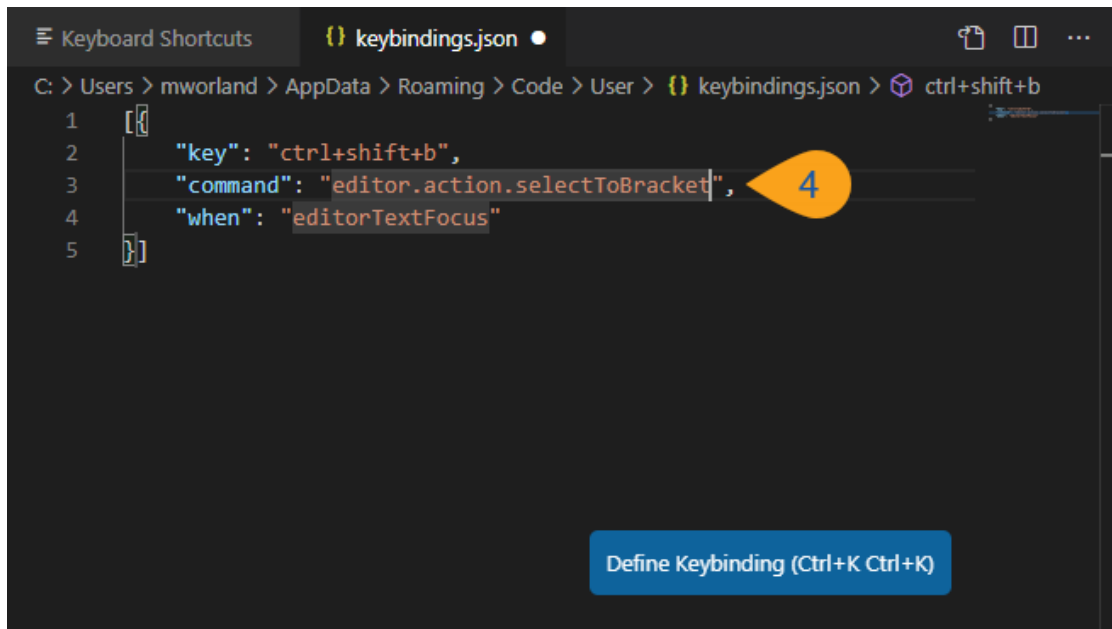
- 1.) Open your user's keybindings file and
- 2.) Press the button or **Ctrl + K**, **Ctrl + K** to define a new shortcut.



3.) Type your shortcut keys **Ctrl + Shift + B** and hit **Enter**



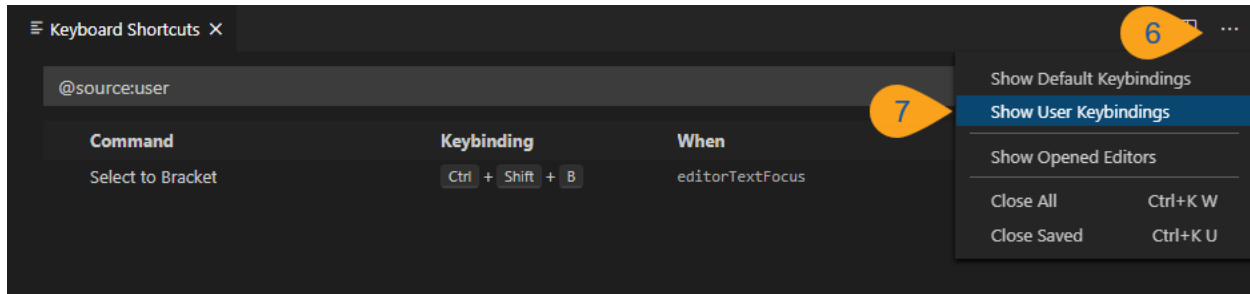
4.) Type in the new command: **editor.action.selectToBracket**



5.) Save your **keybindings.json**

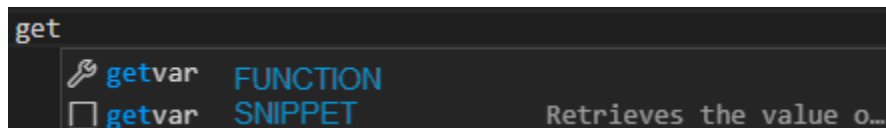
Since we are creating a new file, make sure you create a backup.

- 6.) To view your newly created Shortcut, select the *Ellipsis* button
- 7.) Then *Show User Keybindings*



IntelliSense & Code Snippets

Writing AutoLISP in VS Code is very similar to the VL IDE with some additional benefits. As you type a parenthesis, bracket, or quote, it completes the pair on the other side. It also has IntelliSense and Code Snippets. IntelliSense is like AutoCAD's AutoComplete functionality. When you start typing in the Visual Studio Code Editor, IntelliSense finds matching functions or Code Snippets. Snippets are bits of code used to reduce the amount of typing. When you find the desired function or snippet, press *Tab* to select it.



Choosing the *getvar* function only finishes typing the function *getvar*. Using Snippets not only completes the word or function, but they can also add multiple lines of code. Choosing the *getvar* snippet adds parentheses, function, quotes, and a prompt. Here is a simple example:

```
(getvar "sysvar")
```

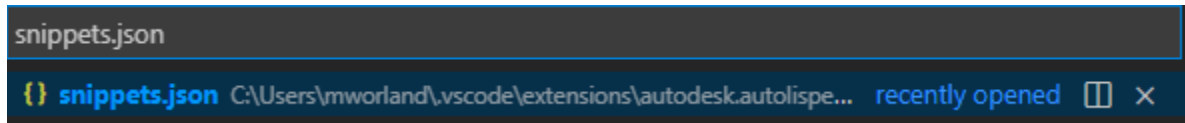
The snippet adds all the code needed for a *getvar* function and prompts you to type the variable's name.

The *if* snippet has multiple lines and prompts, as shown below:

```
(if (testexpr)
  (progn
    (thenexpr)
  )
)
```

Adding New Snippets

We can further customize VS Code by adding our snippets. The AutoCAD AutoLISP extension has its own snippets, use **Ctrl + P** to search for and open the **snippets.json** file.

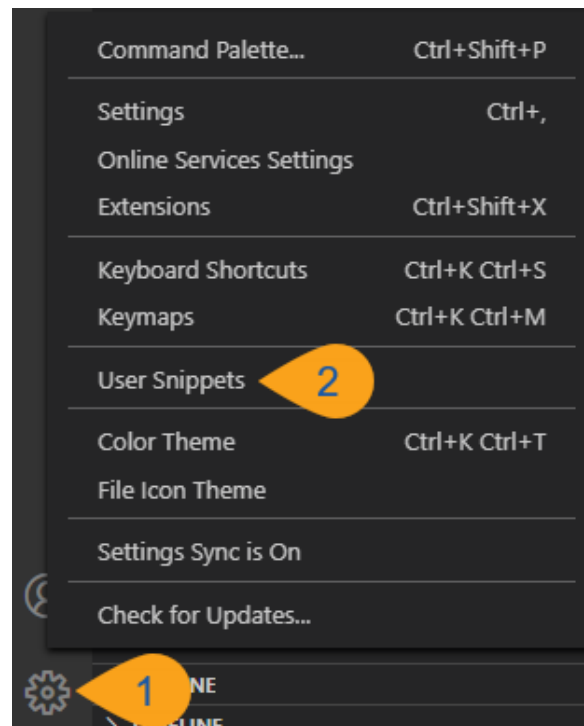


Below is a list of the out of the box snippets.

(defun	caddr	getlayer	readline
(defun c:	car	getvar	repeat
abs	chr	if	reverse
alert	circle	ifp	search
alloc	cons	inters	setq
angle	dcl	itoa	setvar
append	defun	lefttrim	ssget
apply	defun c:	line	strcat
arc	distance	nth	substr
ascii	entmake	open	tblsearch
assoc	entmod	pline	trim
atan	eq	polar	vgetlayer
atof	equal	prin1	while
atoi	expt	princ	writeline
atomsfamily	foreach	print	
boundp	getcolor	prompt	

While the help files state you can modify this file, a user file can also be updated. I try not to modify files that could be overwritten when an update is installed, but in either case, whichever file you change, make sure you back it up. Let's review how to update the User Snippets file.

Select the **Manage** button and choose **User Snippets**



Snippets Structure

```

1 // Place your snippets for autolisp here. Each snippet is defined under a snippet name and has a prefix, body and
2 // description. The prefix is what is used to trigger the snippet and the body will be expanded and inserted. Possible variables are:
3 // $1, $2 for tab stops, $0 for the final cursor position, and ${1:label}, ${2:another} for placeholders. Placeholders with the
4 // same ids are connected.
5 // Example:
6 // "Print to console": {
7 //   "prefix": "log",
8 //   "body": [
9 //     "console.log('$1');",
10 //     "$2"
11 //   ],
12 //   "description": "Log output to console"
13 // }
14
15

```

When you first open the autolisp.json file, you see a brief overview and example of creating a snippet. A snippet structure needs to be followed strictly. Make sure your brackets, quotes, colons, commas, and general layout match a previous example. In the example on the next page, we create a snippet to iterate every object in the Active Document.

The snippet's name is "*iterate active document*". This name needs to be unique regarding all other snippets. If no description is given, the name will be used as the Description when viewed in the IntelliSense prompt.

The Body of the snippet contains the code that is placed in the Editor. Each line of the Body needs to be wrapped in quotes. If you have quotes or backslashes as part of your original code, you need to escape them with a backslash. Here is an example where we escape a quote and backslash.

Use `\t` to add a Tab into your final code.

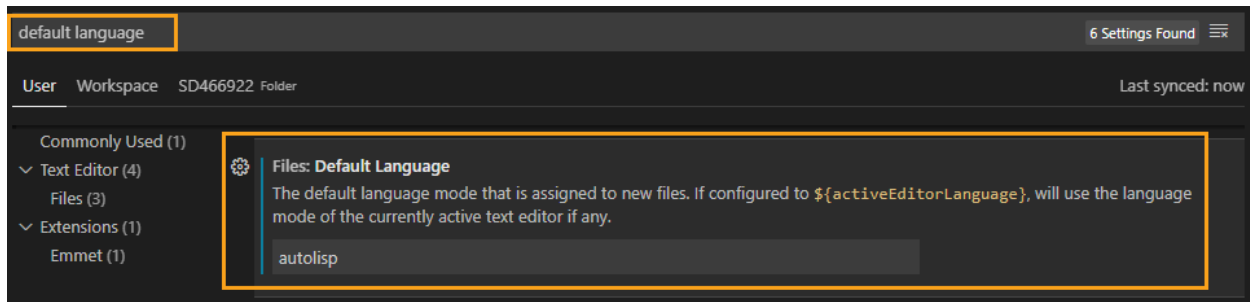
`{1:YourPrompt}`

The Description is shown when viewing the information from the IntelliSense prompt.

Page 22

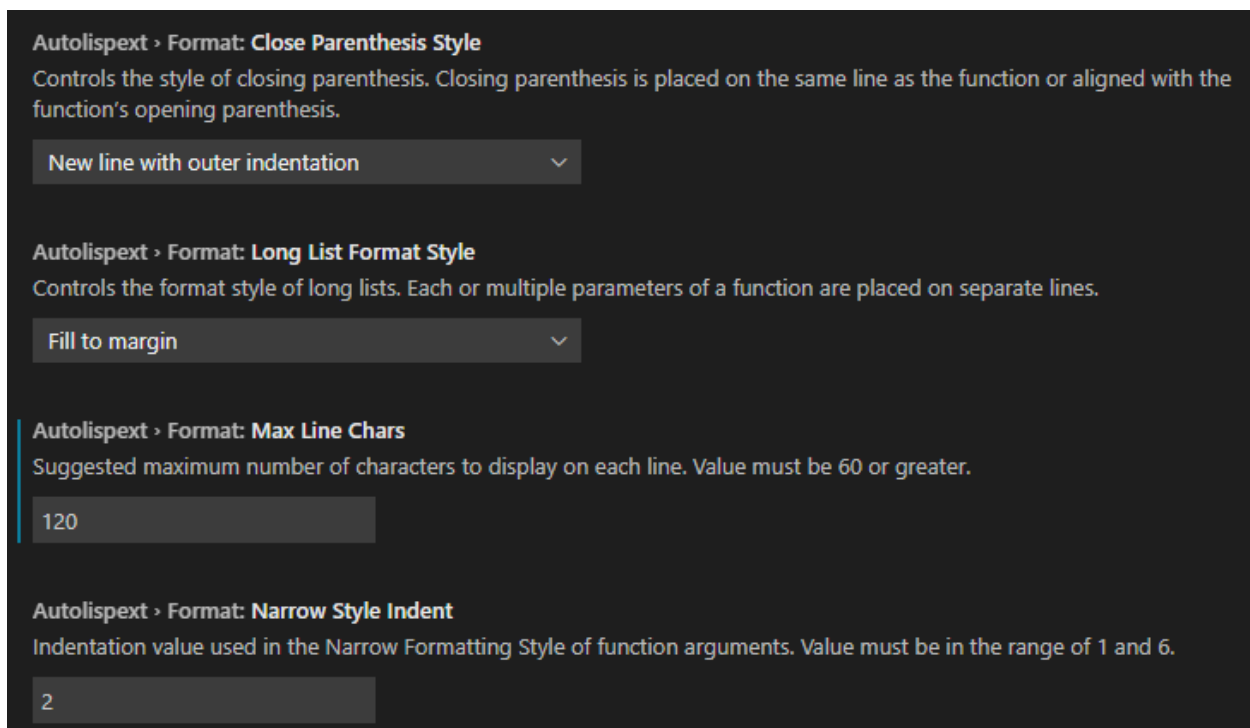
Creating Files

Visual Studio Code is an editor for many programming languages. Since it is not primarily used for developing AutoLISP files, you need to add the correct extension; .LSP, .MNL, or .DCL. Suppose your primary purpose for Visual Studio Code is to write AutoLISP. In that case, you should set the default language in the settings. To open the settings, use the keyboard shortcut **Ctrl + ,**. Then in the settings search box, search for **Default Language**. Finally, in the Default Language text box, type **autolisp**.



Formatting

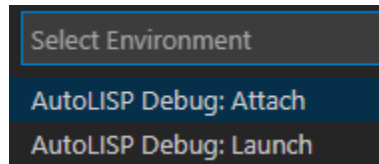
The AutoLISP Extension in Visual Studio Code has some basic formatting options.



Use **Alt + Shift + F** to format your entire file. Use **Ctrl + K**, **Ctrl + F** to format a selection. Both options are available with a right-click.

Debugging

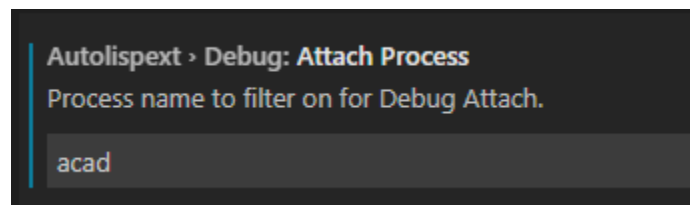
Troubleshooting and debugging code is an essential part of writing robust programs. The AutoCAD AutoLISP Extension has two out of the box debugging configurations. However, these are not automatic configurations. You need to select either the Attach or Launch configuration when using **F5** to debug without setting up an automatic configuration.



Before debugging for the first time, make sure your AutoLISP Extension settings are correct. Below we cover the needed values for these settings.

Attach

The AutoLISP Debug: Attach configuration allows you to attach the debug session to the currently running AutoCAD session.

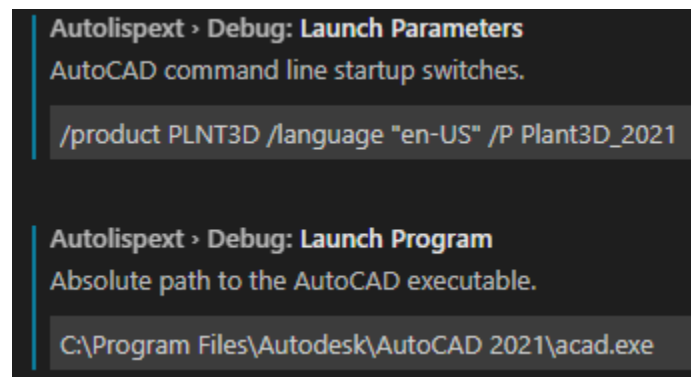


The Attach Process setting tells Visual Studio Code which process to attach itself to. This setting is case sensitive and has different values depending on your operating system:

Windows: **acad**
Mac OS: **AutoCAD**

Launch

The AutoLISP Debug: Launch configuration allows you to start a new AutoCAD session using startup switches. Startup switches change the default startup of AutoCAD. There are many switches, including, load a vertical application, choose a language, set a profile, start with a template, or run a script. You set these switches in the [Launch Parameters](#) text box. In the example below, AutoCAD Plant3D starts with the en-US language pack and the Plant3D_2021 profile.

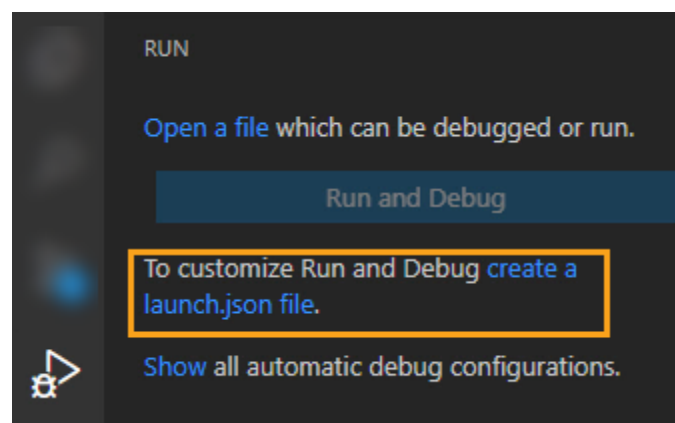


The [Launch Program](#) tells VS Code the correct AutoCAD executable to launch. Using the out of the box default Debug Configurations are recommended by Autodesk.

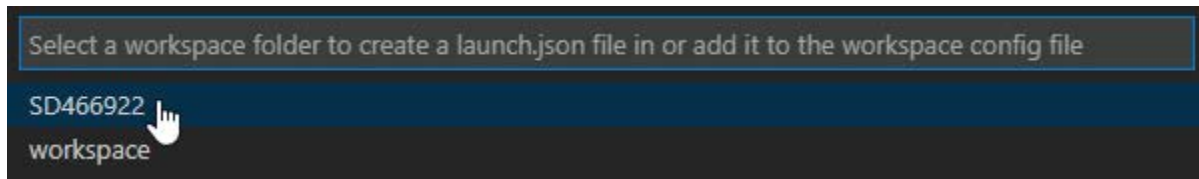
Define Configurations

In previous releases of the extension, you had to create your configurations. These were saved with your current working folder. In many cases, using the out of the box setup is sufficient. However, these configurations can still be used. They may be preferred when you need specific Launch Parameters for the current workspace. Once created, you select a default configuration to use every time you begin debugging. If needed, follow the steps below in creating workspace specific debug configurations.

Go to the Debug Side Bar, select [create a launch.json](#) file.



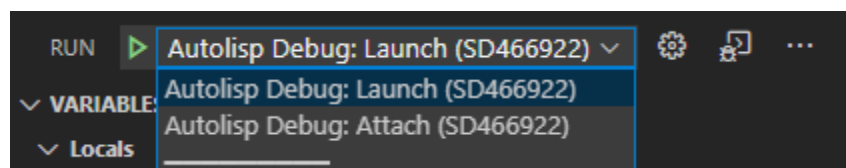
Choose a location to save the JSON data.



Now add the text between the square brackets to add a new Attach and Launch configuration:

```
"configurations": [
  {
    "type": "launchlisp",
    "request": "launch",
    "name": "Autolisp Debug: Launch",
    "attributes": {
      "path": "C:\\Program Files\\Autodesk\\AutoCAD 2021\\acad.exe",
      "params": ""
    }
  },
  {
    "type": "attachlisp",
    "request": "attach",
    "name": "Autolisp Debug: Attach",
    "attributes": {
      "process": "acad"
    }
  }
]
```

We now select the default configuration to be used with **F5**. Note: this text box was not available until creating the new launch.json file



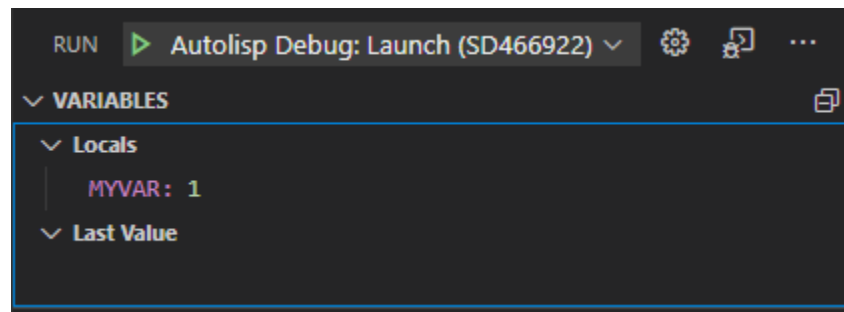
We can now use **F5** to start a debug session using our default workspace debugging configuration.

Debug Side Bar

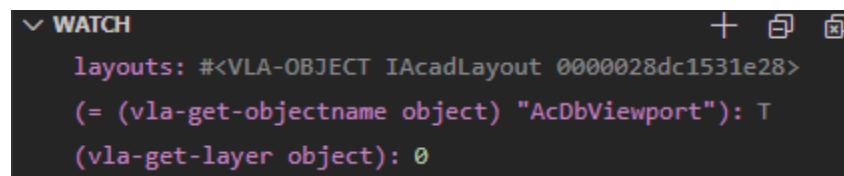
While debugging, the status bar of Visual Studio Code window turns to a different color. Your Side Bar also displays your Variables, Watch list, Call Stack, and Breakpoints.

Adding variables to your variables list when defining a function, adds them to the Variables Locals section of the Debug Side Bar.

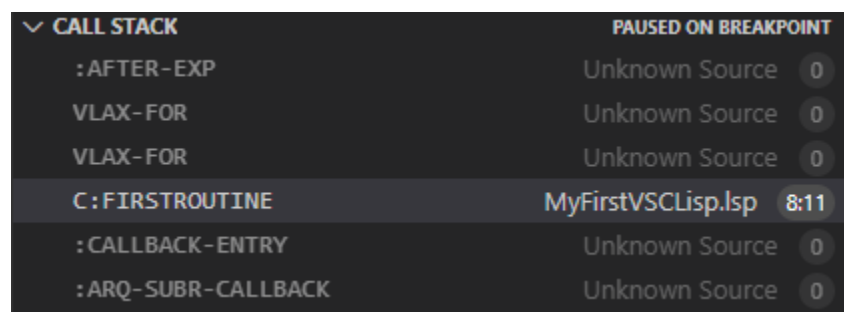
There is also a Last Value variable that shows the value of the last evaluated variable or expression.



The watch list is persistent between VS Code sessions. You add variables and expressions to the list using the **+** button and typing your text or selecting the text in the Editor, Right-Clicking, and choosing **Add to Watch**. The value is shown to the right of the text being watched. You can remove all watched items using the **X** button.



Call stack steps are shown in the Call Stack section.



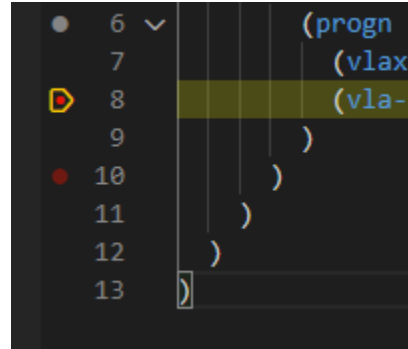
Current breakpoints are displayed in the Breakpoints list. You can add new breakpoints using the **+** button. Disable all breakpoints using the **double circle** button. Remove all breakpoints using the **X** button.



Breakpoints

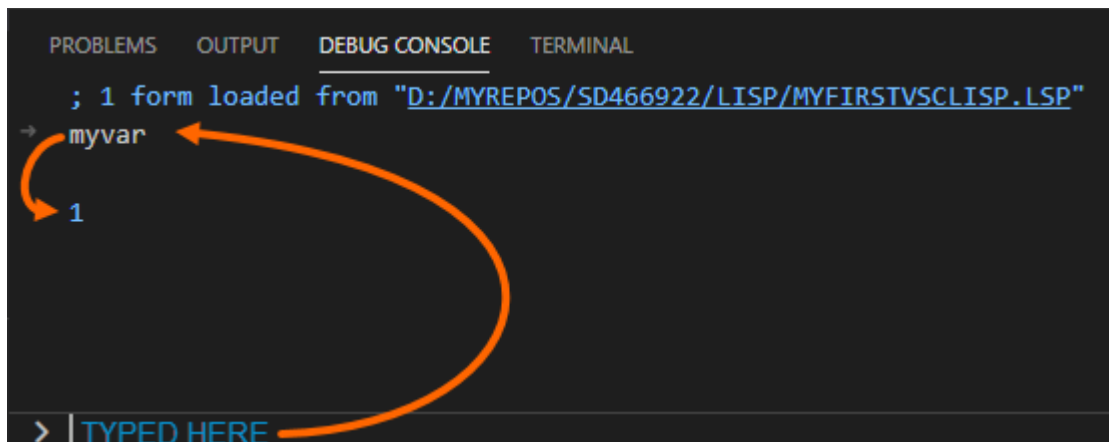
Breakpoints are essential in debugging software. When a breakpoint is hit, Visual Studio Code stops the execution of code, and you can evaluate your currently set variables. In the Editor's farthest left column, you can select a row, and Visual Studio Code adds a new breakpoint. Selecting an active breakpoint removes the breakpoint. Right clicking an active breakpoint allows you to remove or disable the breakpoint.

- = Active breakpoint
- = Disabled breakpoint
- = Future breakpoint

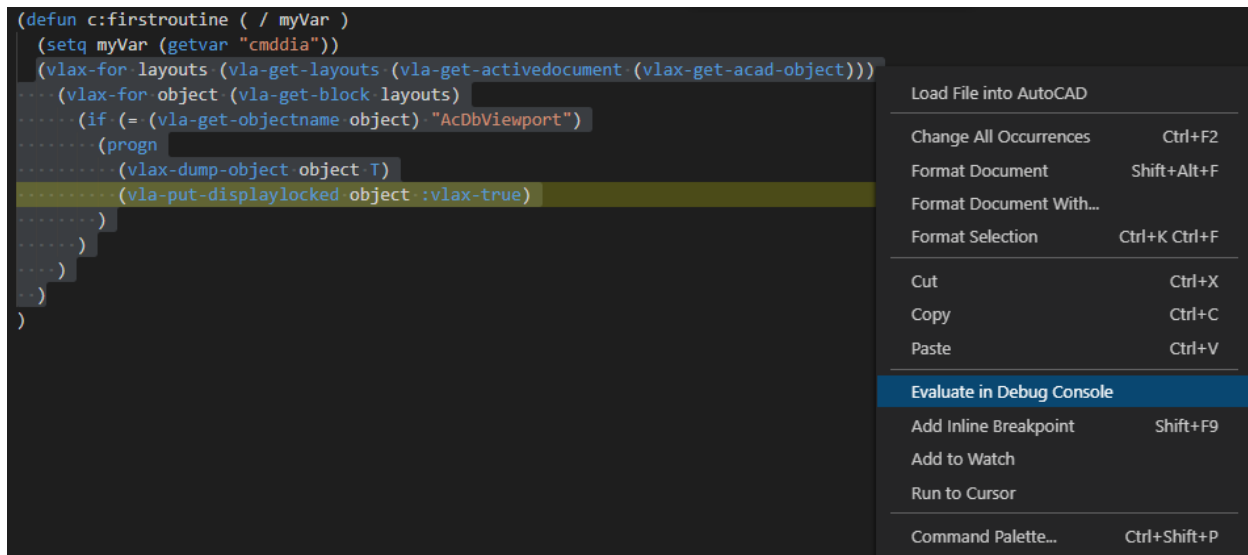


Debug Console

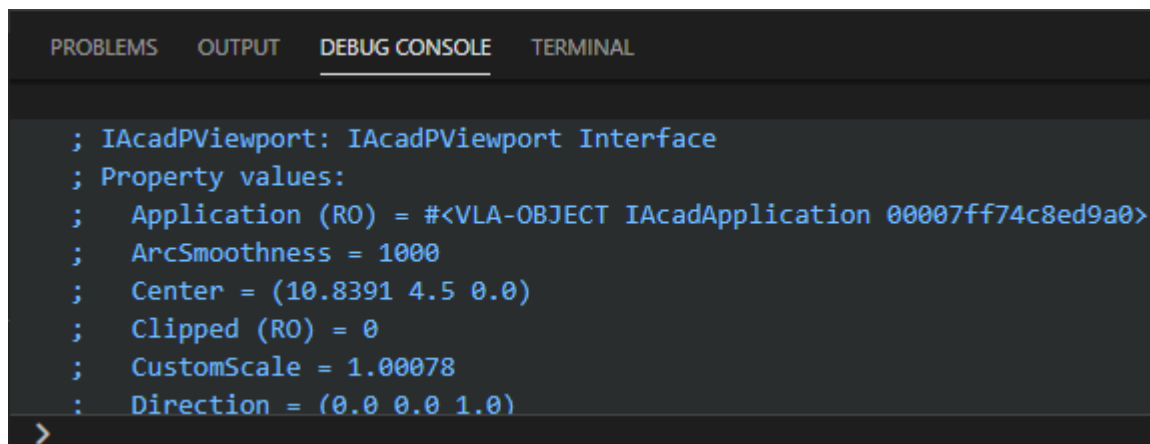
To immediately evaluate an expression or check a variable, you can use the Debug Console.



You can also evaluate selected text by right-clicking and choosing *Evaluate in Debug Console*



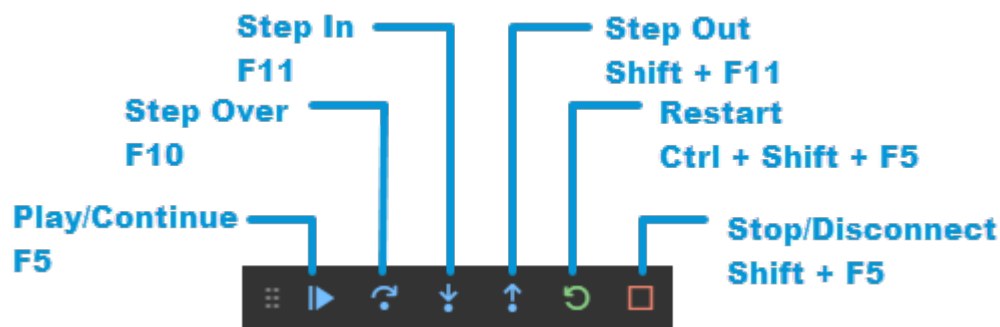
The selected code is shown in the debug console and immediately run.



Debug Commands

The debug commands become available when you begin a debug session.

F5	Start or Continue executing your debug code
F10	Use when you are confident the current line executes without error.
F11	Use when you need to evaluate each portion of a line
Shift + F11	Use when you are confident the remainder of a function executes without error.
Ctrl + Shift + F5	Use to restart your current debug session
Shift + F5	Use to end your current debug session
F9	Add a breakpoint at the current line



Source Control Management

Source Control Management provides many benefits. There are many facets of SCM and plenty of great resources to learn more about it. If you are new to Source Control Management and Git, there are some resources in the [Additional Information](#) section of this handout. You do not need to use SCM to be successful with Visual Studio Code. However, there are many benefits to introducing it to your current workflow. Working with a team of developers is much easier when you use SCM. Even if you are a lone programmer or a CAD Manager, you can take advantage of SCM. Git is the process that allows you to take snapshots of your code and add messages to the snapshot for future reference. Git snapshots (commits) allow you to compare and rollback changes. This can be great for testing new features or making changes. If you have a large change to be made, consider using a branch. A branch is like doing a save-as on a drawing to try a different design layout. You test the branch code and polish it until it is ready for production, then you merge it back. A branch is also helpful if you need several stages of development. You can have a Master branch, a beta branch, and a developer branch. Where in each branch, the code goes through different checks and balances or procedures. Let's look at setting up Git to work with Visual Studio Code.

Git Setup

With minimal setup, Visual Studio Code can link with local or remote repositories through Git. A repository is basically a folder where Git can manage the files. You need to install Git before using the Source Control section of Visual Studio Code.

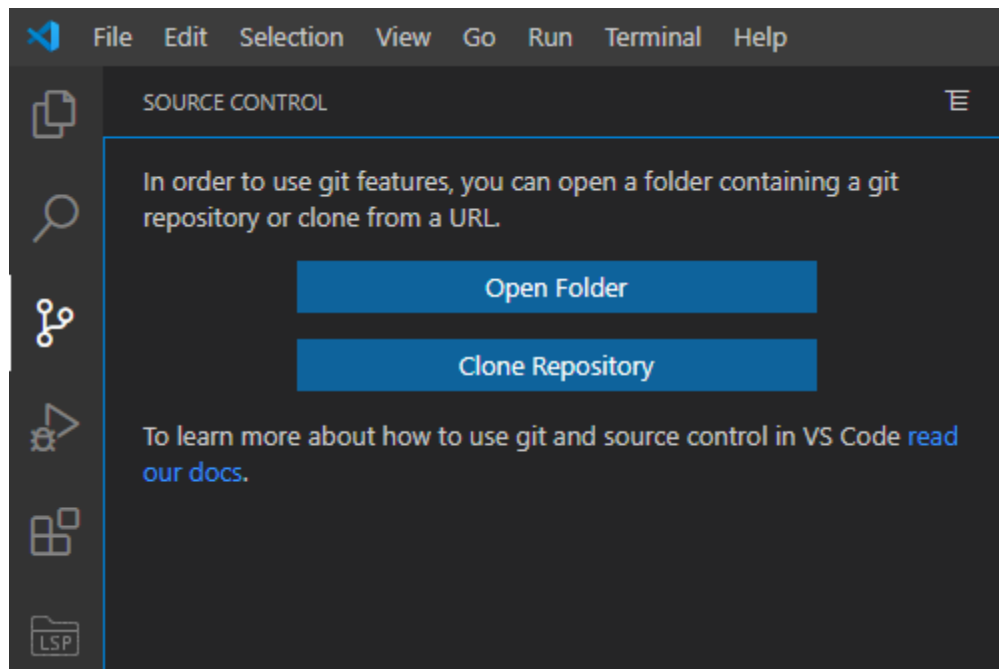
Download git from <https://git-scm.com/download>

Installing with the default options is a great place to start. Before you can commit changes, you need to set up Git to know who is making the commits. In the terminal panel, you need to add your email and user name. Use the example code below. Replace **YOURNAME** and **YOUREMAIL** with your name and email

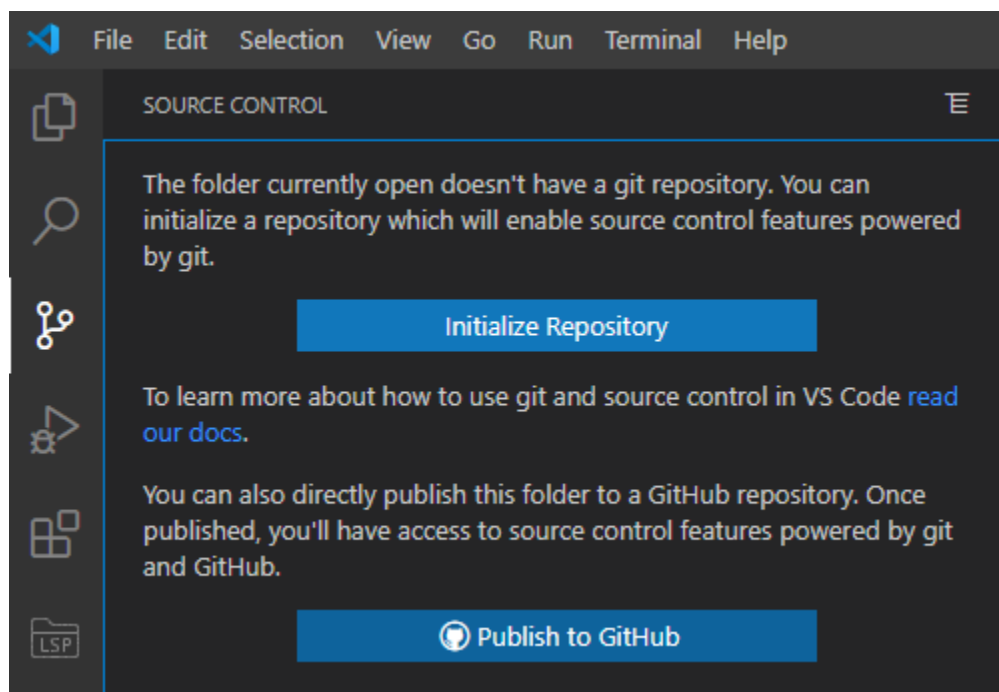
```
git config --global user.name "YOURNAME"  
git config --global user.email "YOUREMAIL"
```

After installing and setting your email and name, you are ready to select or initialize a repository.

Choose your local Git repository or Clone a remote repository to start working with it.



If the folder you chose is not initialized for Git, you can initialize it using the buttons provided.

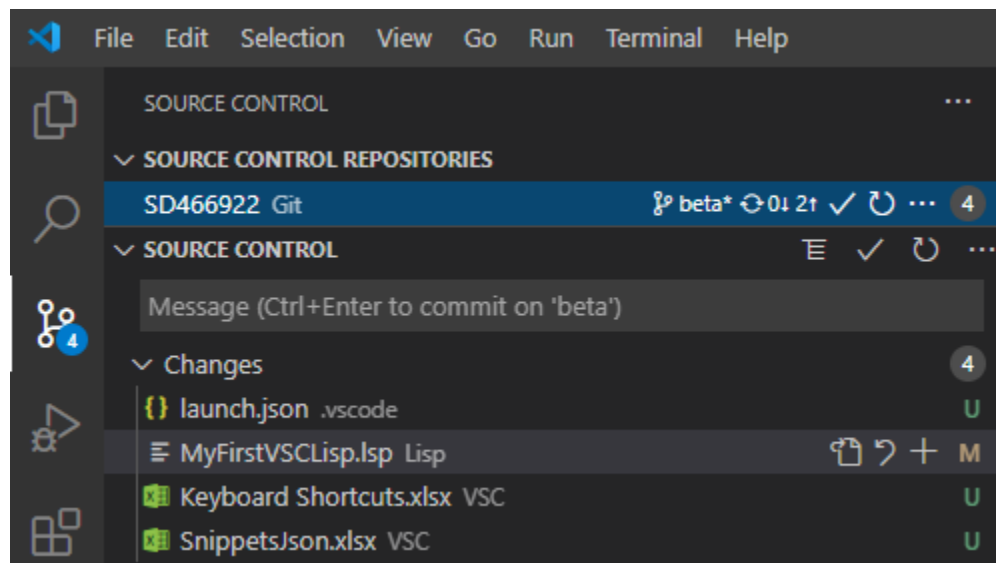


VS Code has quite a few Git related extensions and if you are looking for a great one to start with, give GitLens a try. <https://marketplace.visualstudio.com/items?itemName=eamodio.gitlens>

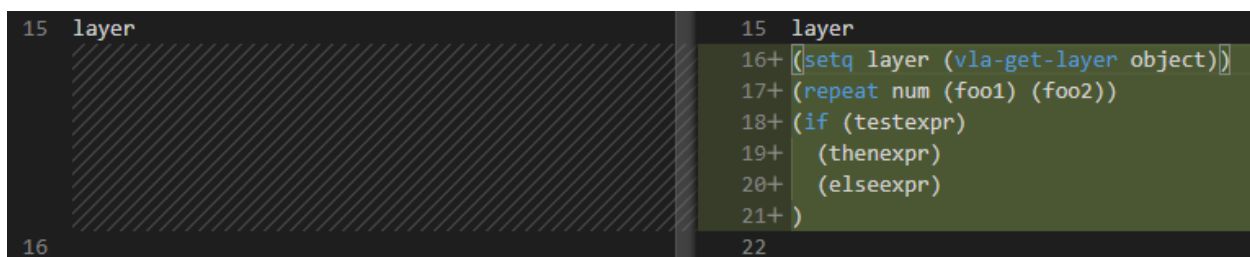
Committing Changes

Visual Studio Code shows when you have Modified or Untracked (new) files. These are marked with an M or U. While you see these notifiers, you know these changes have not been committed into the repository (repo). When you commit a change, you add a description of the change, and the snapshot is saved to the repo.

In the image below, there are three Untracked files and one Modified file. Using the checkmark commits the changes and prompts you for a description. There are many basic Git related commands in the ellipsis button for even more control.



When you commit changes to a modified file, you can see the previous code and compare it to the new code. Below, Lines 16-21 are new. This functionality is a huge time saver when trying to figure out why code no longer works. And if you wrote a detailed description of the change, you could quickly get in the mind set of a change that could have been made months or years prior.



Remote Repositories

Remote repositories are where you store your code in a remote location like a cloud provider. GitHub is a popular option that works well with Visual Studio Code. After setting up a remote repository, you clone it locally, in what is called a local repository. This allows you to work locally but backup your code to a remote location. Having a remote repository helps with backups and sharing code between developers. Using a remote repository is not required, but it provides some peace of mind knowing your code is backed up in the cloud if your computer crashes.

Source Control Management and Git are not required, but they provide many great benefits once implemented.

Open Source Project

The AutoCAD AutoLISP Extension is open-source, allowing the community to drive and enhance its functionality. GitHub stores the project, and you can contribute by following the guidelines found in the contribution guide. Below are some helpful links if you would like to contribute.

<https://adndevblog.typepad.com/autocad/2020/08/opensource-announcement-for-autocad-autolisp-extension.html>

<https://github.com/Autodesk-AutoCAD/AutoLispExt>

<https://github.com/Autodesk-AutoCAD/AutoLispExt/blob/main/CONTRIBUTING.md>

Additional Information

The Autodesk team has done a great job of documenting how to use Visual Studio Code with AutoCAD. Review the help documents found here:

<http://help.autodesk.com/view/OARX/2021/ENU/?guid=GUID-7BE00235-5D40-4789-9E34-D57685E83875>

There are many resources if you are new to Git. Check out these pages for additional information:

<https://code.visualstudio.com/docs/editor/versioncontrol>

<https://git-scm.com/doc>

<https://git-scm.com/video/what-is-git>

<https://training.github.com/downloads/github-git-cheat-sheet.pdf>

To learn more about Source Control with AutoCAD, please visit this excellent class:

<https://www.autodesk.com/autodesk-university/class/Control-Your-Code-Introduction-Source-Control-Programmers-and-CAD-Managers-2018>

This handout and other downloads are available from <https://tinyurl.com/SD466922>