


Version 1.84 ([/updates](#)) is now available! Read about the new features and fixes from October.



Topics

In this article

 (<https://vscode.dev/github/microsoft/vscode-docs/blob/main/docs/editor/variables-reference.md>)

Variables Reference

Visual Studio Code supports variable substitution in Debugging ([/docs/editor/debugging](#)) and Task ([/docs/editor/tasks](#)) configuration files as well as some select settings. Variable substitution is supported inside some key and value strings in `launch.json` and `tasks.json` files using `${variableName}` syntax.

Predefined variables

The following predefined variables are supported:

- `${userHome}` - the path of the user's home folder
- `${workspaceFolder}` - the path of the folder opened in VS Code
- `${workspaceFolderBasename}` - the name of the folder opened in VS Code without any slashes (/)
- `${file}` - the current opened file
- `${fileWorkspaceFolder}` - the current opened file's workspace folder
- `${relativeFile}` - the current opened file relative to `workspaceFolder`
- `${relativeFileDirname}` - the current opened file's `dirname` relative to `workspaceFolder`
- `${fileBasename}` - the current opened file's `basename`
- `${fileBasenameNoExtension}` - the current opened file's `basename` with no file extension
- `${fileExtname}` - the current opened file's `extension`
- `${fileDirname}` - the current opened file's `folder path`
- `${fileDirnameBasename}` - the current opened file's `folder name`
- `${cwd}` - the task runner's current working directory upon the startup of VS Code
- `${lineNumber}` - the current selected line number in the active file
- `${selectedText}` - the current selected text in the active file
- `${execPath}` - the path to the running VS Code executable
- `${defaultBuildTask}` - the name of the default build task
- `${pathSeparator}` - the character used by the operating system to separate components in file paths

Hello from Seattle. Follow @code (<https://go.microsoft.com/fwlink/?LinkID=533687>)

Star 152,793

Predefined variables examples

Supposing that you have the following requirements:

Support files located at `/home/your-username/your-project/folder/file.ext` opened in your editor;
 2. The directory `/home/your-username/your-project` opened as your root workspace.
 Privacy (https://go.microsoft.com/fwlink/?LinkId=521839)
 Terms of Use (https://www.microsoft.com/legal/terms-of-use) License (/License)
 So you will have the following values for each variable:



- `$(userHome)` - `/home/your-username`
- `${workspaceFolder}` - `/home/your-username/your-project`
- `${workspaceFolderBasename}` - `your-project`
- `$(file)` - `/home/your-username/your-project/folder/file.ext`
- `$(fileWorkspaceFolder)` - `/home/your-username/your-project`
- `$(relativeFile)` - `folder/file.ext`
- `$(relativeFileDirname)` - `folder`
- `$(fileBasename)` - `file.ext`
- `$(fileBasenameNoExtension)` - `file`
- `$(fileDirname)` - `/home/your-username/your-project/folder`
- `$(fileExtname)` - `.ext`
- `$(lineNumber)` - line number of the cursor
- `$(selectedText)` - text selected in your code editor
- `$(execPath)` - location of Code.exe
- `$(pathSeparator)` - `/` on macOS or linux, `\` on Windows

Tip: Use IntelliSense inside string values for `tasks.json` and `launch.json` to get a full list of predefined variables.

Variables scoped per workspace folder

By appending the root folder's name to a variable (separated by a colon), it is possible to reach into sibling root folders of a workspace. Without the root folder name, the variable is scoped to the same folder where it is used.

For example, in a multi root workspace with folders `Server` and `Client`, a

`$(workspaceFolder:Client)` refers to the path of the `Client` root.

Environment variables

You can also reference environment variables through the `$(env:Name)` syntax (for example, `$(env:USERNAME)`).

```
{
  "type": "node",
  "request": "launch",
  "name": "Launch Program",
  "program": "${workspaceFolder}/app.js",
  "cwd": "${workspaceFolder}",
  "args": ["$(env:USERNAME)"]
}
```

Configuration variables

You can reference VS Code settings ("configurations") through `${config:Name}` syntax (for example, `${config:editor.fontSize}`).

Command variables

If the predefined variables from above are not sufficient, you can use any VS Code command as a variable through the `${command:commandID}` syntax.

A command variable is replaced with the (string) result from the command evaluation. The implementation of a command can range from a simple calculation with no UI, to some sophisticated functionality based on the UI features available via VS Code's extension API. If the command returns anything other than a string, then the variable replacement will not complete. Command variables **must** return a string.

An example of this functionality is in VS Code's Node.js debugger extension, which provides an interactive command `extension.pickNodeProcess` for selecting a single process from the list of all running Node.js processes. The command returns the process ID of the selected process. This makes it possible to use the `extension.pickNodeProcess` command in an **Attach by Process ID** launch configuration in the following way:

```
{
  "configurations": [
    {
      "type": "node",
      "request": "attach",
      "name": "Attach by Process ID",
      "processId": "${command:extension.pickNodeProcess}"
    }
  ]
}
```

When using a command variable in a `launch.json` configuration, the enclosing `launch.json` configuration is passed as an object to the command via an argument. This allows commands to know the context and parameters of the specific `launch.json` configuration when they are called.

Input variables

Command variables are already powerful but they lack a mechanism to configure the command being run for a specific use case. For example, it is not possible to pass a **prompt message** or a **default value** to a generic "user input prompt".

This limitation is solved with **input variables** which have the syntax: `${input:variableID}` . The `variableID` refers to entries in the `inputs` section of `launch.json` and `tasks.json` , where additional configuration attributes are specified. Nesting of input variables is not supported.

The following example shows the overall structure of a `tasks.json` that makes use of input variables:

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "label": "task name",
      "command": "${input:variableID}"
      // ...
    }
  ],
  "inputs": [
    {
      "id": "variableID",
      "type": "type of input variable"
      // type specific configuration attributes
    }
  ]
}
```

Currently VS Code supports three types of input variables:

- **promptString**: Shows an input box to get a string from the user.
- **pickString**: Shows a Quick Pick dropdown to let the user select from several options.
- **command**: Runs an arbitrary command.

Each type requires additional configuration attributes:

`promptString` :

- **description**: Shown in the quick input, provides context for the input.
- **default**: Default value that will be used if the user doesn't enter something else.
- **password**: Set to true to input with a password prompt that will not show the typed value.

`pickString` :

- **description**: Shown in the quick pick, provides context for the input.
- **options**: An array of options for the user to pick from.
- **default**: Default value that will be used if the user doesn't enter something else. It must be one of the option values.

An option can be a string value or an object with both a label and value. The dropdown will display **label**: **value**.

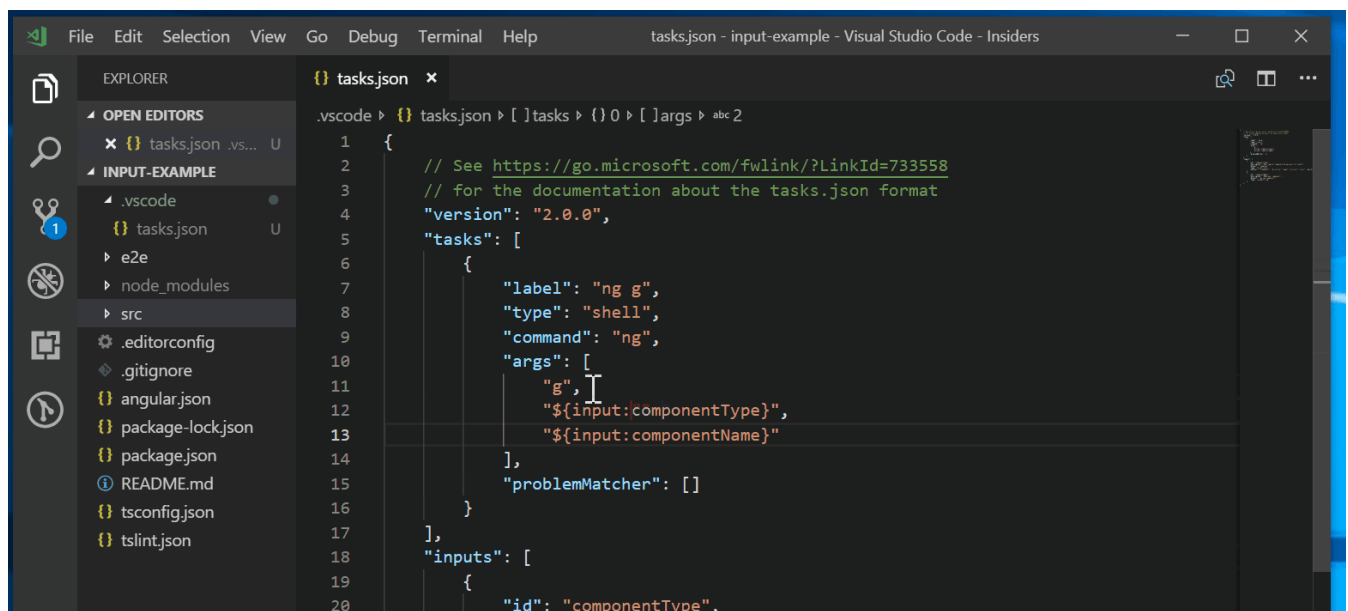
`command` :

- **command**: Command being run on variable interpolation.
- **args**: Optional option bag passed to the command's implementation.

Below is an example of a `tasks.json` that illustrates the use of `inputs` using Angular CLI:

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "label": "ng g",
      "type": "shell",
      "command": "ng",
      "args": ["g", "${input:componentType}", "${input:componentName}"]
    }
  ],
  "inputs": [
    {
      "type": "pickString",
      "id": "componentType",
      "description": "What type of component do you want to create?",
      "options": [
        "component",
        "directive",
        "pipe",
        "service",
        "class",
        "guard",
        "interface",
        "enum"
      ],
      "default": "component"
    },
    {
      "type": "promptString",
      "id": "componentName",
      "description": "Name your component.",
      "default": "my-new-component"
    }
  ]
}
```

Running the example:



```
21     "description": "What type of component do you want to create?",
22     "default": "component",
23     "type": "pickString",
24     "options": ["component", "directive", "pipe", "service", "class", "guard",
25
26   },
27   {
28     "id": "componentName",
29     "description": "Name your component.",
30     "default": "my-new-component",
31     "type": "promptString"
32   }
33 ]
```

Visual Studio Code interface showing a JSON configuration file. The editor has a dark theme. The left sidebar shows the 'OUTLINE' and 'NPM SCRIPTS' views. The status bar at the bottom indicates 'master*' and 'Initializing JS/TS language features'. The bottom right of the editor shows 'Ln 13, Col 40', 'Spaces: 4', 'UTF-8', 'LF', and 'JSON with Comments'.

The following example shows how to use a user input variable of type `command` in a debug configuration that lets the user pick a test case from a list of all test cases found in a specific folder. It is assumed that some extension provides an `extension.mochaSupport.testPicker` command that locates all test cases in a configurable location and shows a picker UI to pick one of them. The arguments for a command input are defined by the command itself.

```
{
  "configurations": [
    {
      "type": "node",
      "request": "launch",
      "name": "Run specific test",
      "program": "${workspaceFolder}/${input:pickTest}"
    }
  ],
  "inputs": [
    {
      "id": "pickTest",
      "type": "command",
      "command": "extension.mochaSupport.testPicker",
      "args": {
        "testFolder": "/out/tests"
      }
    }
  ]
}
```

Command inputs can also be used with tasks. In this example, the built-in Terminate Task command is used. It can accept an argument to terminate all tasks.

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "label": "Terminate All Tasks",
      "command": "echo ${input:terminate}",
      "type": "shell",
      "problemMatcher": []
    }
  ],
  "inputs": [
    {
      "id": "terminate",
      "type": "command",
      "command": "workbench.action.tasks.terminate",
      "args": "terminateAll"
    }
  ]
}
```

Common questions

Details of variable substitution in a debug configuration or task

Variable substitution in debug configurations or tasks is a two pass process:

- In the first pass, all variables are evaluated to string results. If a variable occurs more than once, it is only evaluated once.
- In the second pass, all variables are substituted with the results from the first pass.

A consequence of this is that the evaluation of a variable (for example, a command-based variable implemented in an extension) has **no access** to other substituted variables in the debug configuration or task. It only sees the original variables. This means that variables cannot depend on each other (which ensures isolation and makes substitution robust against evaluation order).

Is variable substitution supported in User and Workspace settings?

The predefined variables are supported in a select number of setting keys in `settings.json` files such as the `terminal.cwd`, `env`, `shell` and `shellArgs` values. Some settings (/[docs/getstarted/settings](/docs/getstarted/settings)) like `window.title` have their own variables:

```
"window.title": "${dirty}${activeEditorShort}${separator}${rootName}${separator}${appName}"
```

Refer to the comments in the Settings editor (`Ctrl+,`) to learn about setting specific variables.

Why isn't `${workspaceRoot}` documented?

The variable `${workspaceRoot}` was deprecated in favor of `${workspaceFolder}` to better align with Multi-root Workspace (</docs/editor/multi-root-workspaces>) support.

Why aren't variables in `tasks.json` being resolved?

Not all values in `tasks.json` support variable substitution. Specifically, only `command`, `args`, and `options` support variable substitution. Input variables in the `inputs` section will not be resolved as nesting of input variables is not supported.

How can I know a variable's actual value?

One easy way to check a variable's runtime value is to create a VS Code task (</docs/editor/tasks>) to output the variable value to the console. For example, to see the resolved value for `${workspaceFolder}`, you can create and run (**Terminal > Run Task**) the following simple 'echo' task in `tasks.json`:

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "label": "echo",
      "type": "shell",
      "command": "echo ${workspaceFolder}"
    }
  ]
}
```

Was this documentation helpful?

Yes

No

11/1/2023

 [Subscribe\(/feed.xml\)](/feed.xml)  [Ask questions\(https://stackoverflow.com/questions/tagged/vscode\)](https://stackoverflow.com/questions/tagged/vscode)

 [Follow @code\(https://go.microsoft.com/fwlink/?LinkID=533687\)](https://go.microsoft.com/fwlink/?LinkID=533687)

 [Request features\(https://go.microsoft.com/fwlink/?LinkID=533482\)](https://go.microsoft.com/fwlink/?LinkID=533482)

 [Report issues\(https://www.github.com/Microsoft/vscode/issues\)](https://www.github.com/Microsoft/vscode/issues)

 [Watch videos\(https://www.youtube.com/channel/UCs5Y5_7XK8HLDX0SLNwkd3w\)](https://www.youtube.com/channel/UCs5Y5_7XK8HLDX0SLNwkd3w)