



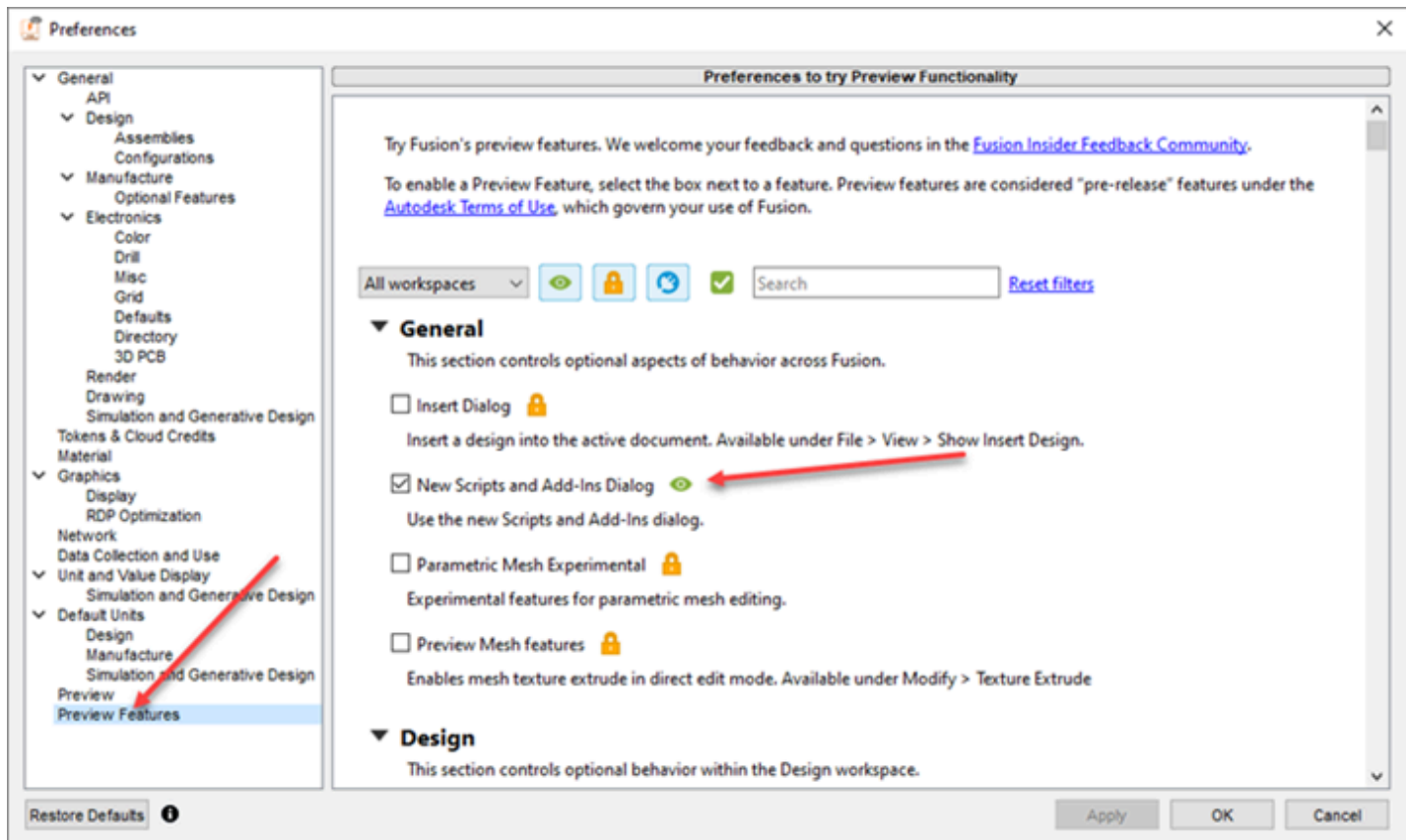
Programming Interface / [Fusion API User's Manual](#) / [Creating Scripts and Add-Ins](#)

Old Dialog

New Dialog

Managing, Running, and Creating Scripts and Add-Ins Using the New Dialog

This release introduces a new dialog for the "Scripts and Add-Ins" command. It's a preview feature, so you must enable it to make it available. To do this, open the Preferences dialog, choose "Preview Features" from the tree on the left, and select the check box beside "New Scripts and Add-Ins Dialog" in the "General" section, as shown below. Once enabled, you'll see the new dialog when you run the "Scripts and Add-Ins" command. Of course, you can switch back at any time by changing the preference setting. Some known issues are listed at the bottom of this page.



Contents

- [Working with Installed Scripts and Add-Ins](#)
 - [Filtering the List of Displayed Scripts and Add-Ins](#)
 - [Installing, Linking, and Removing Scripts and Add-Ins](#)

- [Default Folders for Scripts and Add-Ins](#)
- [User Defined Folder for Scripts and Add-Ins](#)
- [Linking Scripts and Add-Ins](#)
- [Uninstalling Scripts and Add-Ins](#)
- [Running Scripts and Add-Ins](#)
- [Creating, Editing, and Running Your First Script](#)
- [Script and Add-In Details](#)
 - [Script and Add-In Files](#)
 - [The Manifest File](#)
 - [Script Code](#)
 - [Add-In Code](#)
- [Scripts vs. Add-Ins](#)
- [Editing and Debugging](#)
- [Known Issues](#)

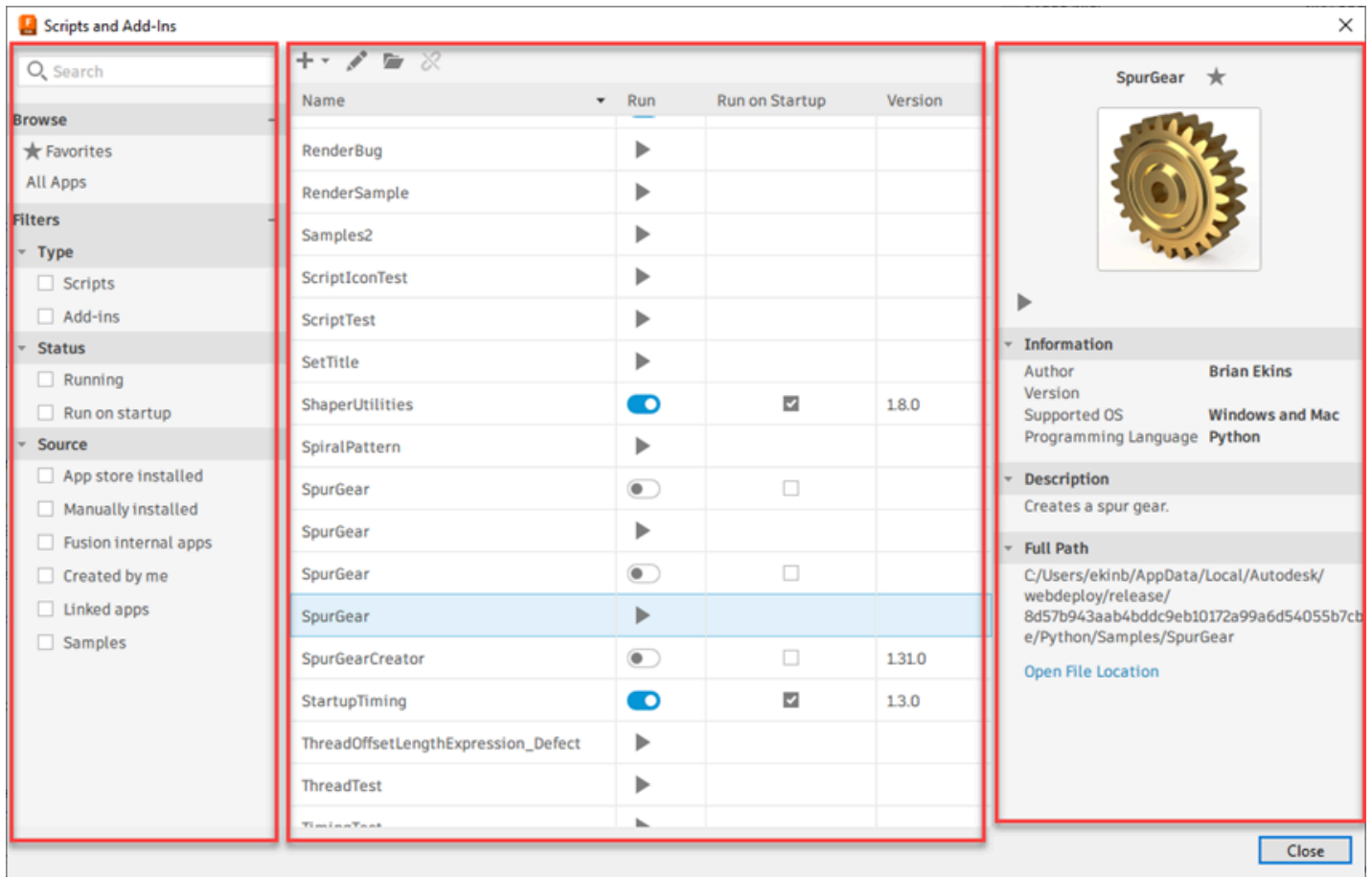
Managing Scripts and Add-Ins

Working with Installed Scripts and Add-Ins

The purpose of the Scripts and Add-Ins dialog is to provide the needed functionality to manage your scripts and add-ins. It provides the following capabilities:

- List all the currently available scripts and add-ins and show additional related information.
- Search and filter that list in different ways.
- Run a script.
- Start an add-in, see if an add-in is running, and stop a running add-in.
- Specify if an add-in should start running automatically when Fusion starts.
- See where a script or add-in is located on your machine and open a new File Explorer or Finder window in that folder.
- Create new scripts and add-ins.
- Create links to scripts and add-ins that exist anywhere on your computer.

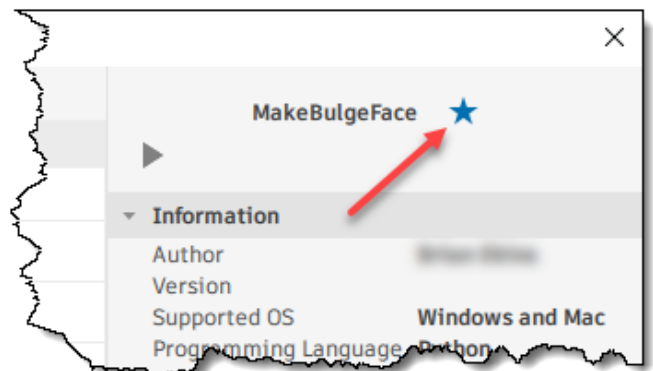
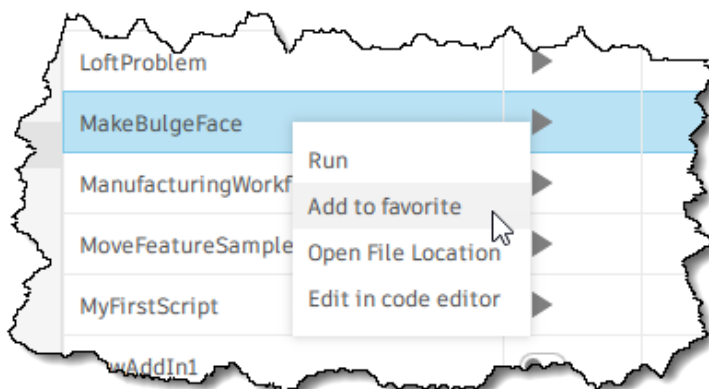
Below is the Scripts and Add-Ins dialog with the three main sections highlighted. The section to the left provides the various filters to control which scripts and add-ins are displayed. The center section displays the list of scripts and add-ins. The section on the right displays detailed information for the selected script or add-in. Each of these is described in more detail below.



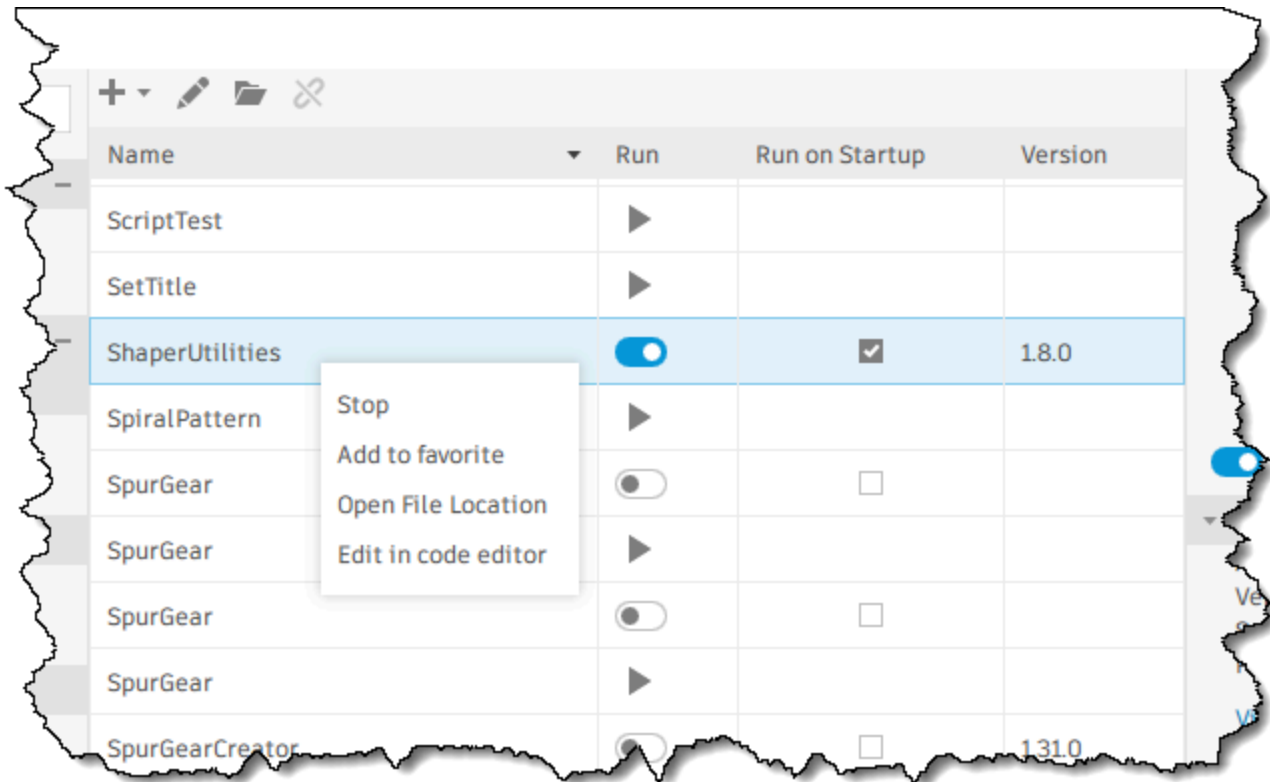
Filtering the List of Scripts and Add-Ins

The left side of the Scripts and Add-Ins dialog provides different filters for controlling what's displayed in the list in the center section of the dialog. Selecting a filter will display the scripts and add-ins with that attribute. For example, if you select "Add-ins", only add-ins will be displayed, and then if you select "Running", it will combine the filters so only the currently running add-ins will be displayed. Any number of filters can be selected at once. You can deselect individual filters to remove that filter. Selecting "All Apps" or "Favorites" at the top of the filter list removes all filters so that everything will be displayed.

A powerful filtering feature is an ability to identify any script or add-in as a "Favorite". You can set a script or add-in to be a favorite by using its context menu or by clicking on the star beside its name in the right-hand section of the dialog, as shown below. Clicking the "Favorites" option at the top of the "Browse" section on the left panel will display your favorite scripts and add-ins. You can select other filters to filter the list of favorites further.



Selecting a script or add-in in the list of scripts and add-ins will display its details in the right-hand section of the dialog. In this dialog section, you can view current state information and perform various operations for the selected script or add-in using the associated context menu, as shown below.



The first column displays the script or add-in's name. The second column indicates its current running state. From this, you can infer whether the item is a script or an add-in. The four possible icons are shown below.

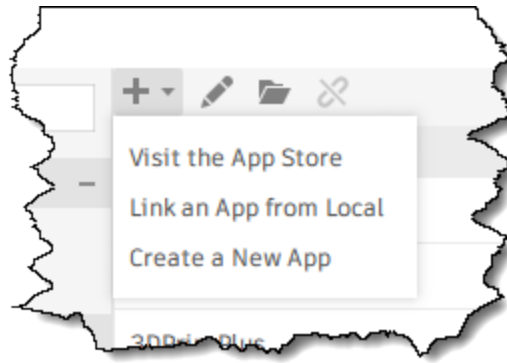
- ▶ - Indicates a script that is not running. Clicking this icon will run the script.
- - Indicates a script that is currently running. It is rare to see this state because scripts typically run and end. However, creating a script that doesn't end automatically when the run function finishes is possible.
- ⬢ - Indicates an add-in that is currently not running. Clicking this icon will start the add-in.
- ⬢ - Indicates an add-in that is running. Clicking this icon will stop the add-in.

The third column indicates if the add-in is set to run automatically on start-up. If this is checked, the add-in will be automatically started by Fusion when Fusion starts and will remain running for the entire session or until you manually stop it. This setting is only available for add-ins. The fourth column shows the version of the add-in.

The fourth column shows the version of the add-in.

A small toolbar with several commands is at the top of the list section. The button on the left displays a pop-up menu where you can go to the Autodesk App Store, link an app, or create a new app. Linking and creation are discussed in more detail below. The command with the pencil icon will open the code editor for the currently selected script or add-in to allow you to edit and debug the script or add-in. The command

with the folder icon will open the folder where the script or add-in is located using Window's File Explorer or Mac's Finder. The command with the broken paper clip will unlink a linked script or add-in.



Installing, Linking, and Removing Scripts and Add-Ins

An "installed" script or add-in is one that exists locally on your computer and that Fusion is aware of. A script or add-in is a folder that contains the code and other associated files, like icon files. Fusion automatically searches several directories on your machine for scripts and add-ins.

Default Folders for Scripts and Add-Ins

Several default folders are always searched for scripts and add-ins when Fusion starts. These folders are listed below.

- **The default path when creating new scripts. As described below, the default path can be redefined by setting a user preference. However, even if the default path is changed, this path is still searched.**

Windows - C:\Users\<userName>\AppData\Roaming\Autodesk\Autodesk Fusion 360\API\Scripts

Mac - ~/Library/Application Support/Autodesk/Autodesk Fusion 360/API/Scripts

- **The default path when creating new add-ins. As described below, the default path can be redefined by setting a user preference. However, it is still searched even if the default path has changed.**

Windows - C:\Users\<userName>\AppData\Roaming\Autodesk\Autodesk Fusion 360\API\AddIns

Mac - ~/Library/Application Support/Autodesk/Autodesk Fusion 360/API/AddIns

- **This is the location where apps downloaded from the Autodesk App Store are installed. It is a shared location where apps for all Autodesk applications, not just Fusion apps, are installed. The folder structure of apps in this location is different. Each add-in is in a .bundle folder. Even though this location is intended for Autodesk App Store apps, other applications sometimes also install in it. Still, they have to follow the rules for having the .bundle folder and a PackageContents.xml file that identifies it as a Fusion app.**

Windows - C:\Users\<userName>\AppData\Roaming\Autodesk\ApplicationPlugins<

Mac - ~/Library/Application Support/Autodesk/ApplicationPlugins

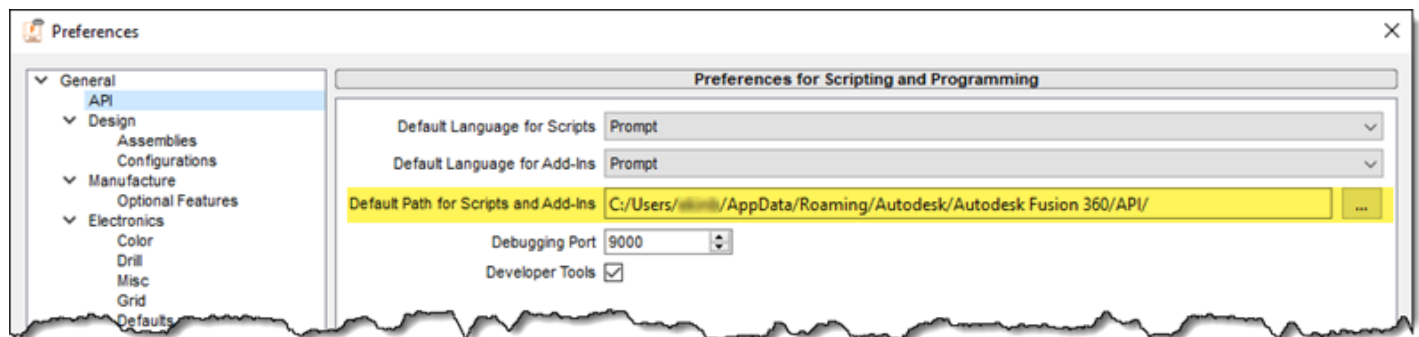
- The location is intended for apps to be installed that are not from the Autodesk App Store. Apps in this location are Fusion-specific and follow the standard folder structure, with each add-in in a folder with the same name as the add-in. You DO NOT need to use a .bundle folder for this location.

Windows - C:\Users\<userName>\AppData\Roaming\Autodesk\FusionAddinscode>

Mac - ~/Library/Application Support/Autodesk/FusionAddins

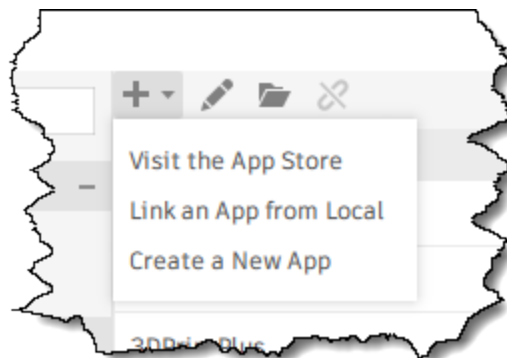
User Defined Folder for Scripts and Add-Ins

A user preference defines the path when creating a new script or add-in. The default path is shown above, but you can set this to any location using the preference shown below. Fusion will search for scripts and add-ins in this defined path in the "Scripts" and "AddIns" folders. If you change this path from the default, Fusion will continue to search the default location.



Linking Scripts and Add-Ins

Fusion also supports using a script or add-in that can exist anywhere on your computer. This is done by adding a "link" to the script or add-in folder. You create a link using the "Link an App from Local" option in the drop-down of the "+" plus command at the top of the list section, as shown below. When running the command, you can browse for any script or add-in folder on your computer. Fusion will remember it and add it to the list of scripts and add-ins. When a linked script or add-in is selected in the list, you can use the "Unlink" command at the top of the list section of the dialog to remove the script or add-in from the list and have Fusion forget about it. Unlinking does NOT delete the folder but only causes Fusion to forget about the script or add-in.



Uninstalling Scripts and Add-Ins

To permanently remove a script or add-in, delete it from your computer. If it was installed on Windows using an installer, it's best to uninstall it using the Windows tools to uninstall programs. If an installer was not used, or you're using a Mac, you can delete the folder containing the add-in by browsing to it using Window's File Explorer or Mac's Finder and deleting or moving it to the trash.

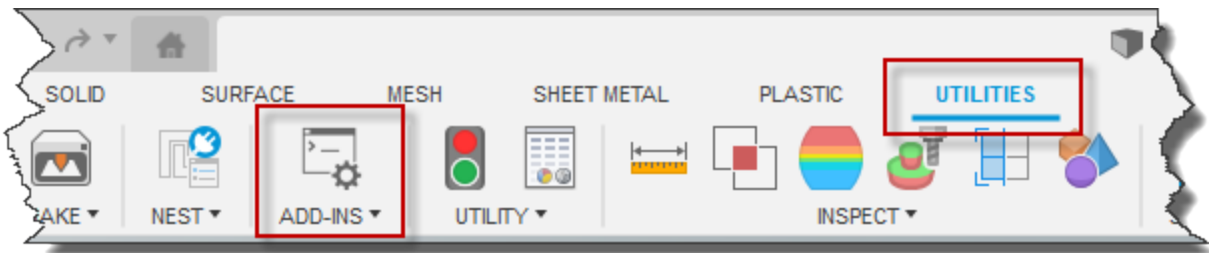
Running Scripts and Add-Ins

Through the "Scripts and Add-Ins" dialog, you can see your existing scripts and add-ins and run them. You do this by finding the script or add-in in the list and clicking either the ► or ⬅ icon beside it in the "Run" column. When running a script, the Scripts and Add-Ins dialog is immediately dismissed so the script can perform whatever action it does. When starting an add-in, the dialog remains open so you can continue to start other add-ins if desired.

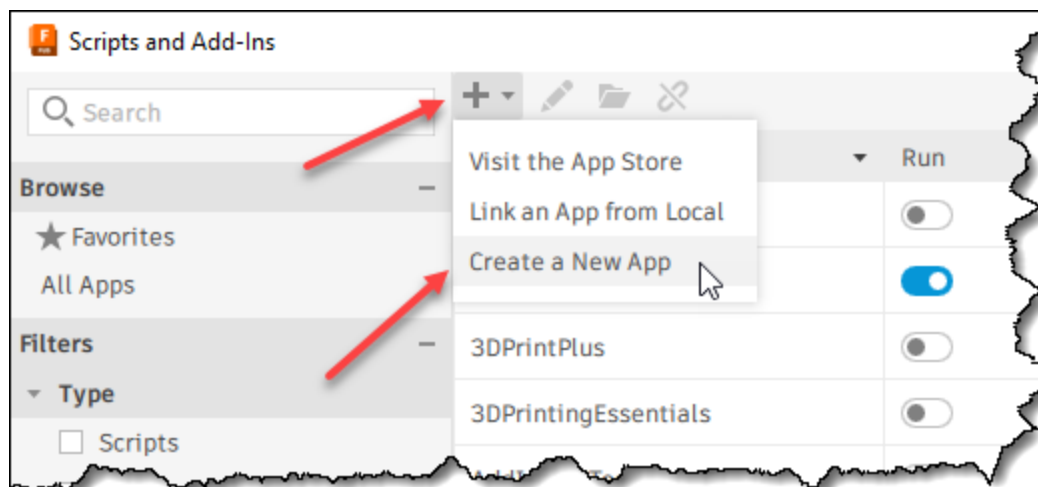
Creating, Editing, and Running Your First Script

Technically, there is not much difference between a script and an add-in. Creating, editing, and debugging them are mostly the same, so the description below applies to both. Before getting into the details, here are the basic steps to create, edit, and run a Python script or add-in. The process is very similar when creating a C++ script or add-in.

1. Run the **Scripts and Add-Ins** command from the UTILITIES tab in the toolbar, as shown below.



2. In the **Scripts and Add-Ins** dialog, click the "+" near the top of the dialog, and then choose "Create a New App" from the drop-down list, as shown below.



3. In the “New App” dialog, make sure “Script” and “Python” are selected. Enter a name for the script name, and optionally enter the “Description”, and “Author” fields and then click “Create”. This will take you back to the “Scripts and Add-Ins” dialog.

New App

General Information

Create a new ☒ **Script** ☐ Add-in

Name

Description

Author

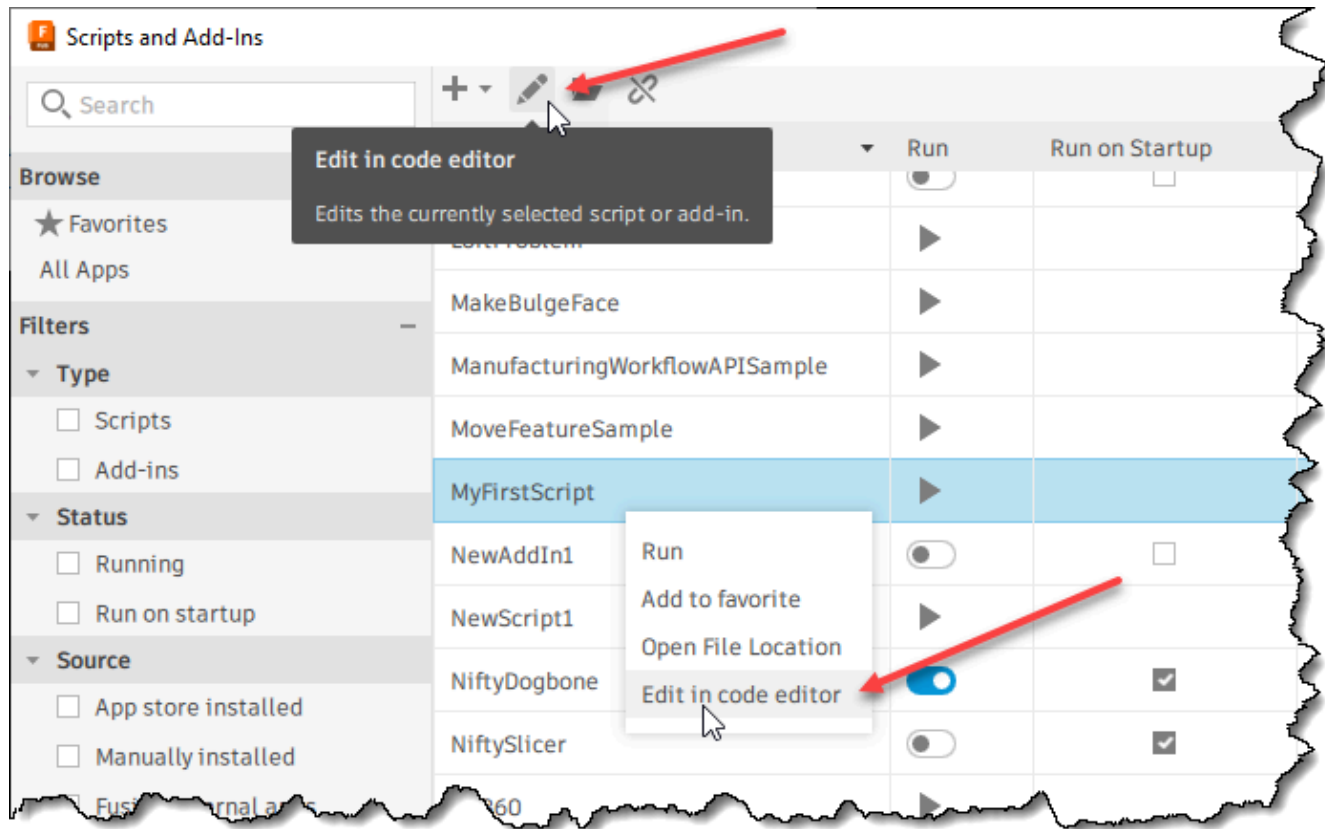
Programming Language and System

Programming Language ☐ C++ ☒ **Python**

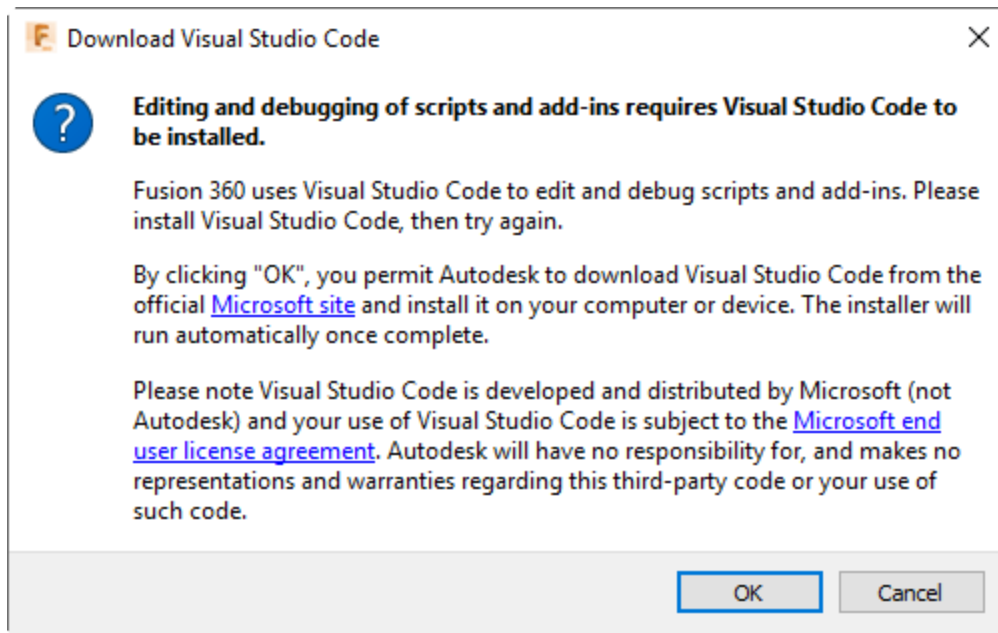
*Target Operating System

Folder Location

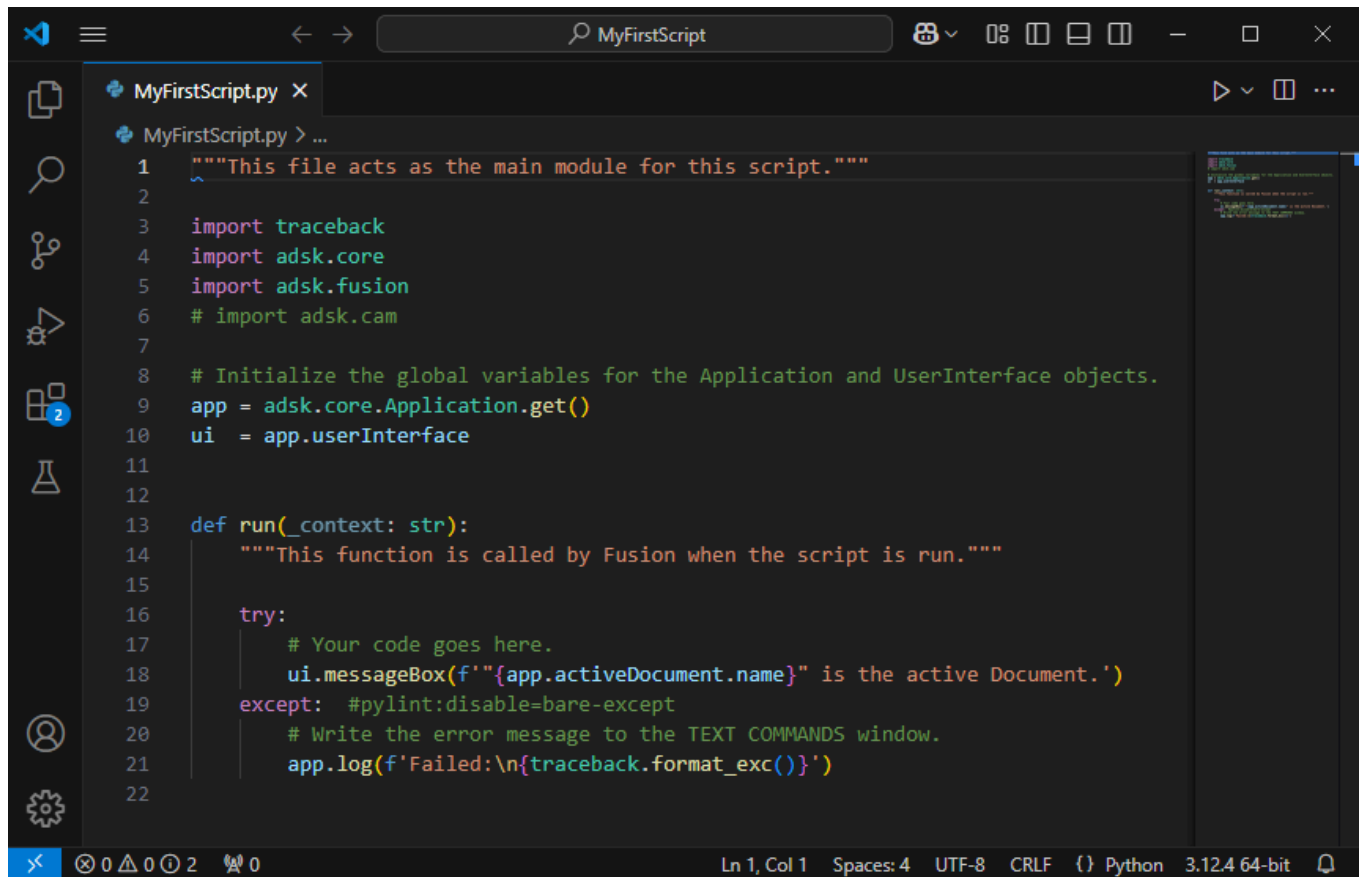
4. The new script will now be shown in the list. To edit it, use the "Edit in code editor" command from the toolbar at the top of the list section or the context menu, as shown below.



For Python programs, Fusion uses Visual Studio Code (VS Code) as the development environment. If it is not already installed when you try to edit or debug, Fusion will display the dialog below to install VS Code. You only need to do this the first time you edit any script or add-in. When VS Code finishes installing, do the Edit step again to open the script in VS Code.

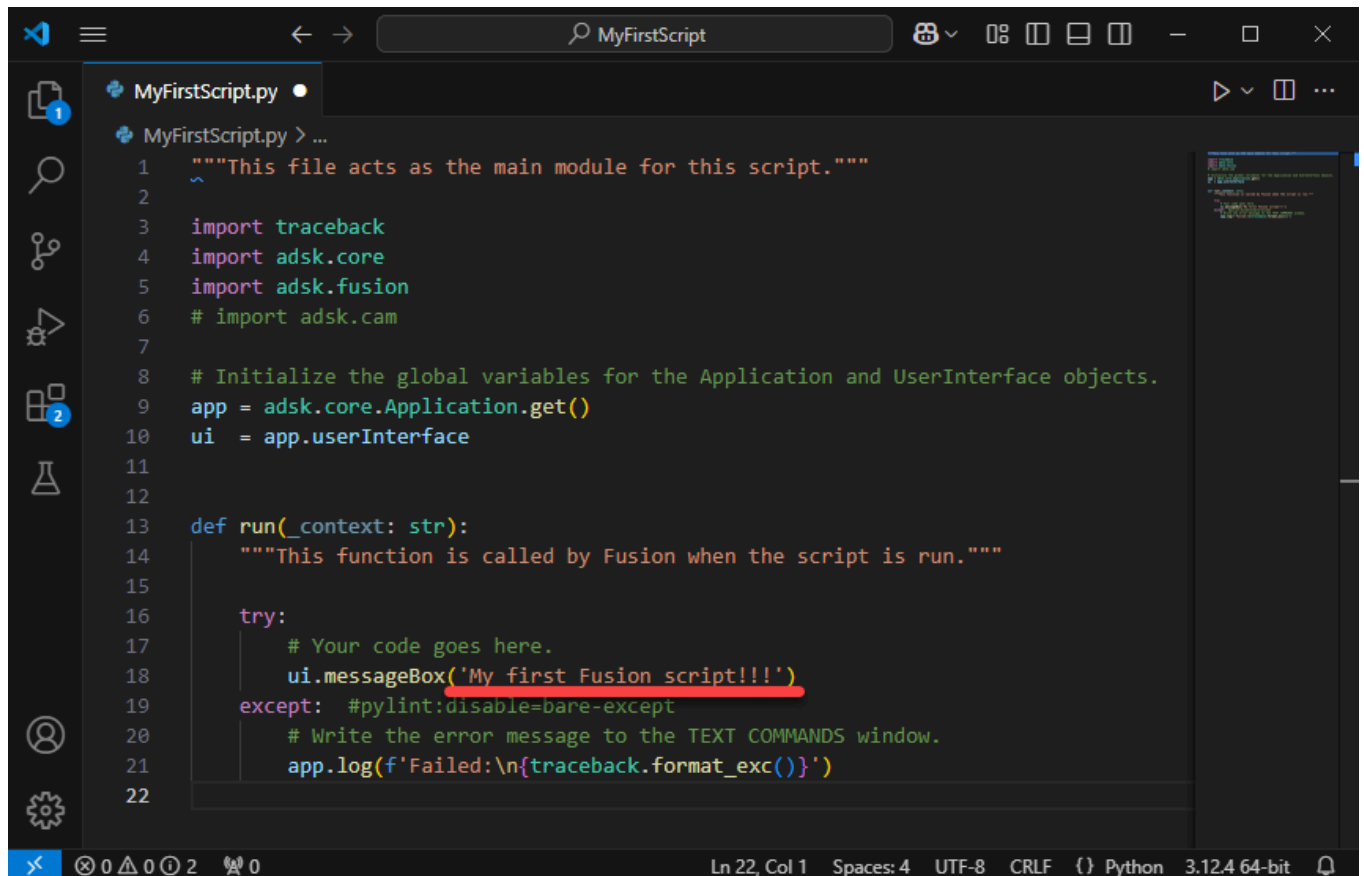


The first time you run VS Code from Fusion, you will see a window pop-up saying an extension is being installed. Fusion is installing the Python extension for VS Code. This also only needs to be done once. Finally, once everything is installed VS Code will open, as shown below.



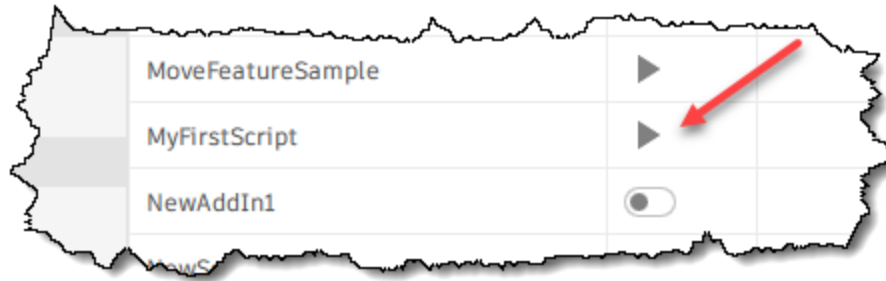
```
1 """This file acts as the main module for this script."""
2
3 import traceback
4 import adsk.core
5 import adsk.fusion
6 # import adsk.cam
7
8 # Initialize the global variables for the Application and UserInterface objects.
9 app = adsk.core.Application.get()
10 ui = app.userInterface
11
12
13 def run(_context: str):
14     """This function is called by Fusion when the script is run."""
15
16     try:
17         # Your code goes here.
18         ui.messageBox(f'"{app.activeDocument.name}" is the active Document.')
19     except: #pylint:disable=bare-except
20         # Write the error message to the TEXT COMMANDS window.
21         app.log(f'Failed:\n{traceback.format_exc()}')
22
```

5. You can now use VS Code to edit your program. For this simple example, edit the text for the messageBox to any message you would like, such as what's shown below, and save the changes.

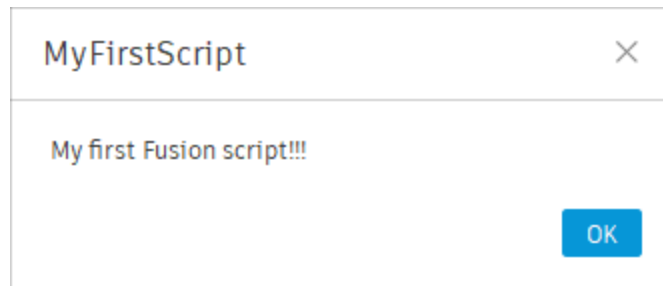


```
1 """This file acts as the main module for this script."""
2
3 import traceback
4 import adsk.core
5 import adsk.fusion
6 # import adsk.cam
7
8 # Initialize the global variables for the Application and UserInterface objects.
9 app = adsk.core.Application.get()
10 ui = app.userInterface
11
12
13 def run(_context: str):
14     """This function is called by Fusion when the script is run."""
15
16     try:
17         # Your code goes here.
18         ui.messageBox('My first Fusion script!!!')
19     except: #pylint:disable=bare-except
20         # Write the error message to the TEXT COMMANDS window.
21         app.log(f'Failed:\n{traceback.format_exc()}')
22
```

6. Congratulations, you have just written your first script. To run your script, run the **Scripts and Add-Ins** command, find your script in the list, and click the ► icon beside it.



The script will run and do whatever it is programmed to do. In this case it will display the message box shown below.

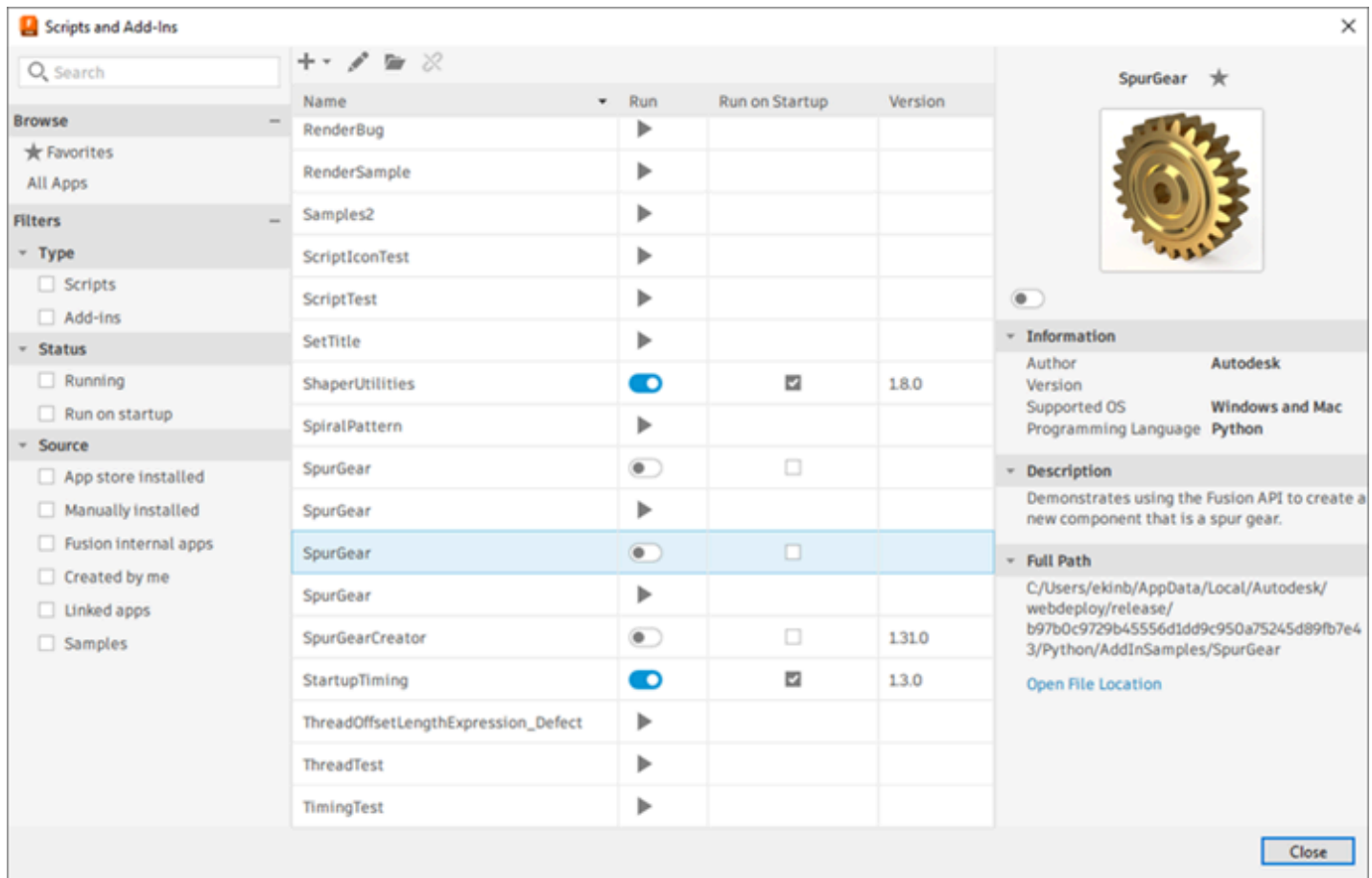


7. A development environment provides two important capabilities; code hints as you write your code to speed up development and reduce mistakes and the ability to debug your program. Debugging is very different between Python and C++. You can learn about debugging your [Python](#) and [C++](#) programs in the language specific topics.

Script and Add-In Details

Now that you have seen the basic process of creating and debugging a script, here is some more information about the details of both scripts and add-ins.

The **Scripts and Add-Ins** dialog is the main access point to scripts and add-ins for both users and programmers. It lists all known scripts and add-ins. From the list you can select a script or add-in and run or edit it.



When creating a new script or add-in, the “New App” dialog is displayed where you enter information about the script or add-in you want to create.

New App

▼ **General Information**

Create a new ☒ **Script** ☐ **Add-in**

Name

Description

Author

▼ **Programming Language and System**

Programming Language ☐ **C++** ☒ **Python**

*Target Operating System

▼ **Folder Location**

Create **Cancel**

The various settings in the dialog are described below.

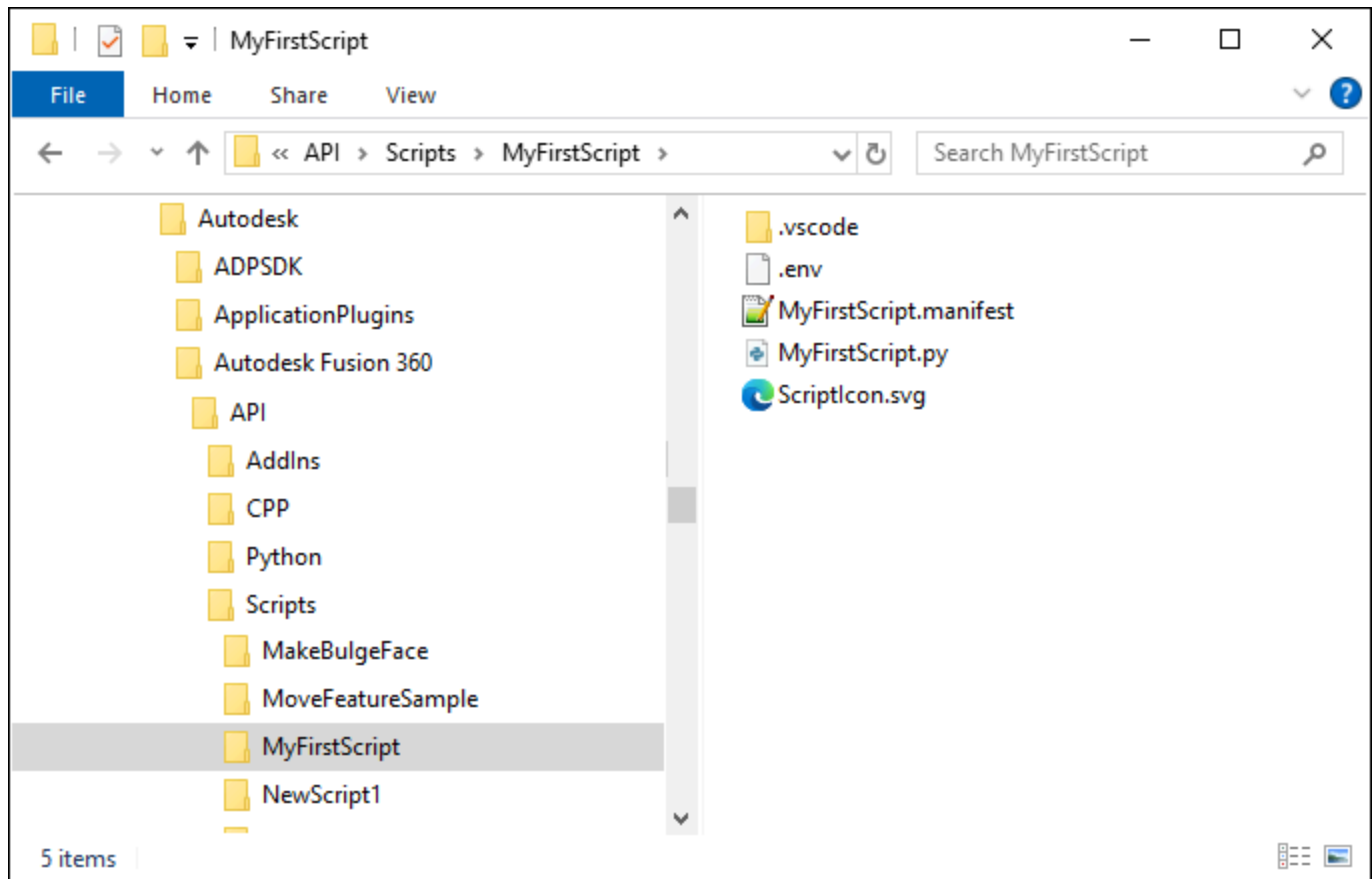
- **Create a new** – Choose whether you want to create a script or add-in.
- **Name** – This is the name of your script or add-in. This name will be used to create a new folder in the location specified by the "Folder Location", and will also be used to name the script or add-in code files.
- **Description** – An optional description of the script or add-in. The description is shown in the properties section of the "Scripts and Add-Ins" dialog.
- **Author** – You can optionally specify the author's name. The author is shown in the properties section of the "Scripts and Add-Ins" dialog.
- **Programming Language** – Choose whether you want to create a Python or C++ script or add-in.
- **Run on Startup** – This setting is add-in-specific and indicates if the add-in should be run automatically when Fusion is started. Most add-ins will want to take advantage of this capability so the commands they define will be available to the user as soon as Fusion starts.
- ~~**Version** – This is an optional setting that is add-in specific and is the version of the add-in. This is a string and can be any form of a version label, for example, "1.0.0", "2016", "R1", "V2", etc.~~
- **Target Operating System** – Indicates which operating system(s) the script or add-in should be available in. For example, if your script or add-in uses Windows-specific libraries you would set this to "Windows", so Fusion won't attempt to display or load it on a Mac.
- **Folder Location** – The location where the script or add-in will be created. When you create a new script or add-in using the dialog, a new folder with the script or add-in name is created, and the add-

in files are created in that folder. By default, this uses the default location as described earlier. You can also browse to any folder, and the script or add-in will be created there, and a link will be created so Fusion will remember it and display it in the dialog.

Scripts and add-ins can exist at any location on the machine, but it's only in the locations listed above where Fusion automatically searches for add-ins when it starts up. A script or add-in in any other location must be explicitly located using the green "+" icon near the top of the "Scripts and Add-Ins" dialog. When copying or installing an add-in onto another computer you should copy it to the location specified above so Fusion will find it automatically.

Script and Add-In Files

When a new script or add-in is created a new folder is created using the specified name and the code files are created in that folder. In addition to the code files, a couple of other files are created; a manifest file is also created that contains additional information about the script or add-in. For example, if you create a Python add-in called MyAddIn, a MyAddIn folder with the files shown below is created in ".../Autodesk/Autodesk Fusion/API/AddIns". Additional files associated with the script or add-in (icons, for example) should be added to this folder so the add-in is completely self-contained and can be "installed" by simply copying this folder to the correct location.



The Manifest File

The .manifest file contains the information specified in the "Create New Script or Add-In" dialog when you initially created the script or add-in. It also contains additional information Fusion uses to determine when it should be displayed and loaded. The manifest file has the same name as the add-in but has a .manifest extension. The file is a text file in JSON format. Shown below is an example of a typical manifest file for an add-in.

```
{
  "autodeskProduct": "Fusion",
  "type": "addin",
  "id": "",
  "author": "Roger Rabbit",
  "description": {
    "": "This is my first add-in"
  },
  "version": "",
  "runOnStartup": true,
  "supportedOS": "windows|mac",
  "editEnabled": true,
  "iconFilename": "AddInIcon.svg"
}
```

Below is a description of each of the items in the manifest.

- **autodeskProduct** – This property will always have the value “Fusion”.
- **type** – This property can be “addin” or “script” to indicate if this program is an add-in or script.
- **id** – This property is not used and can be left empty.
- **author** – This property is a string containing the name of the author. This is displayed in the “Scripts and Add-Ins” dialog.
- **description** – This is a JSON object with properties that define the add-in's description. Using the JSON format, it is defined as an object with one or more properties so that you can specify descriptions for more than one language. The example below has one property with an empty name, which is the default description and will be used for any language without a specific description. The other properties define the text for the other languages supported by Fusion using Microsoft-defined language codes.

```
"Description":{
  "":"Default description",
  "1028": "說明在中國",
  "1031": "Beschreibung auf Deutsch",
  "1033": "Description in English",
  "1034": "Descripción en Español",
  "1036": "Description en Français",
  "1040": "Descrizione in Italiano",
  "1041": "日本語での説明",
  "1042": "한국어 설명"
}
```

Below is a list of the language codes for the languages supported by Fusion.

2052 : Peoples Republic of China Chinese

1028 : Taiwan Chinese

1029 : Czech

1033 : English

1036 : French

1031 : German

1038 : Hungarian

1040 : Italian

1041 : Japanese

1042 : Korean

1045 : Polish

1046 : Brazilian Portuguese

1049 : Russian

1034 : Spanish

1055 : Turkish

- **version** – This property defines the version of the add-in and can be any string, i.e. “1.0.0”, “2016”, “R1”, “V2”, etc.
- **runOnStartup** – This property can be true or false to indicate if Fusion should automatically start this add-in when Fusion is started. By default, this is false, which is the desired setting when writing and debugging your code. However, once developed, this value is typically true.
- **supportedOS** – This property can be "windows", "mac", or "windows|mac". This defines which operating systems the add-in will load on. One example of where this is used is when an add-in uses OS-specific libraries so that the add-in won't work on any other OS. For example, if I write an add-in that uses a Windows-specific library, I can set the supportedOS to "windows" so that on a Mac, Fusion won't display the add-in in the "Scripts and Add-Ins" dialog and also won't attempt to run it on startup. Most Python scripts and add-ins should be compatible with both Mac and Windows, so this should be set to "Windows | Mac" to indicate the add-in can be loaded for both operating systems. C++ scripts and add-ins must be compiled separately for each platform, so they may more likely use this option when the developer doesn't have access to both a Windows and Mac machine to compile.
- **sourcewindows and sourcemac** –

A C++ script or add-in has two additional properties that identify the filename of the project file for both Windows and Mac. When you select the "Edit" option in the "Scripts and Add-Ins" dialog, Fusion opens the associated project file using the application associated with that file type. For example, in the example below, a .vcxproj file is specified for the sourcewindows property, so Visual Studio will be invoked since it is defined within Windows as the associated application for .vcxproj files. You can use any code editor you want by changing this file or the application associated with that file type on your computer.

```
"sourcewindows":    "NewCPPTest.vcxproj",  
"sourcemac":       "NewCPPTest.xcodeproj"
```


Notice that, except for the sourcewindows and sourcemac properties, the script or add-in's name is not specified in the manifest file. The name is defined by the name used for the main directory. The primary source file, and the manifest file also have this name. To change the name of a script or add-in, change the names of the directory and files to the new name.

Script Code

Below is the code automatically written when a new Python script is created. Notice the "run" function, which Fusion calls when executing the script. Fusion also passes in information through the "context" argument as to whether the script is being run at Fusion startup or is being loaded during a session. The context can be ignored for a script because it is always run during a Fusion session and never at startup. The run function is the entry point into your script, and once it is complete, the script is finished, and Fusion unloads it.

```
"""This file acts as the main module for this script."""

import traceback
import adsk.core
import adsk.fusion
# import adsk.cam

# Initialize the global variables for the Application and UserInterface objects.
app = adsk.core.Application.get()
ui = app.userInterface

def run(_context: str):
    """This function is called by Fusion when the script is run."""

    try:
        # Your code goes here.
        ui.messageBox(f'"{app.activeDocument.name}" is the active document.')
    except: #pylint:disable=bare-except
        # Write the error message to the TEXT COMMANDS window.
        app.log(f'Failed:\n{traceback.format_exc()}')
```

Add-In Code

The code below is a minimal add-in. Notice that it is the same as a new script except that it also contains a "stop" function. This code differs from what is created when you create a new Python add-in using the "Scripts and Add-Ins" command. That add-in has much more structure to support an add-in that supports different types of commands. You can read more about it in the [topic in the API User's Manual](#).

```
"""This file acts as the main module for this add-in."""

import traceback
```

```

import adsk.core
import adsk.fusion
# import adsk.cam

# Initialize the global variables for the Application and UserInterface objects.
app = adsk.core.Application.get()
ui = app.userInterface

def run(_context: str):
    """This function is called by Fusion when the add-in is run."""

    try:
        # Your code goes here.
        ui.messageBox(f'"{app.activeDocument.name}" is the active document.')
    except: #pylint:disable=bare-except
        # Write the error message to the TEXT COMMANDS window.
        app.log(f'Failed to run add-in:\n{traceback.format_exc()}')

def stop(context):
    """This function is called by Fusion when the add-in is terminated."""

    try:
        ui.messageBox('Stopping the add-in')

    except: #pylint:disable=bare-except
        # Write the error message to the TEXT COMMANDS window.
        app.log(f'Failed to stop add-in:\n{traceback.format_exc()}')

```

The “stop” function is called by Fusion whenever the add-in is stopped and unloaded. This can happen because the user is stopping it using the “Scripts and Add-Ins” dialog, or more typically, it is because Fusion is shutting down and all add-ins are stopped. The stop function is where the add-in can perform any needed cleanup, like removing any user-interface elements that it created.

Both the run and the stop functions have a single argument called “context” to pass additional information to the add-in, indicating the context of why the run or stop function is being called. Depending on the language, this information is passed using different types, but in all cases, it represents a set of name:value pairs. Python passes this as a Dictionary object, and C++ passes it as a string in JSON format. The following name:value pairs are currently supported.

run

Name	Value	Description
IsApplicationStartup	true or false	Indicates the add-in is being started as a result of automatic loading during Fusion startup (true) or is being loaded by the user through the “Scripts and Add-Ins” dialog (false).

stop

Name	Value	Description
IsApplicationClosing	true or false	Indicates the add-in is being shut down as a result Fusion being shut down (true) or because the user stopped it through the “Scripts and Add-Ins” dialog (false).

Scripts vs. Add-Ins

As was said earlier, there is very little technical difference between a script and an add-in. The primary difference is how they are executed and their lifetime. The user executes a script through the “Scripts and Add-Ins” command, and the execution stops immediately after the run function is complete. A script runs, and then it is done.

An add-in is typically automatically loaded by Fusion when Fusion starts up. An add-in also usually creates one or more custom commands and adds them to the user interface during startup. The add-in continues to run throughout the Fusion session, so it can react whenever the user executes any of its commands. The add-in remains running until Fusion is shut down or the user explicitly stops it through the “Scripts and Add-Ins” dialog. When it stops, it cleans up whatever user interface customization it created in its stop function.

How an add-in uses the Fusion API is not any different from a script. It is the same API, and none of the API calls are limited to scripts or add-ins. However, there are a few areas of the API that are more useful to an add-in than a script. The first is the portion that deals with working with the Fusion user interface and adding buttons or other controls to access your custom commands. For example, if you create a custom command that draws geometry in a sketch, you will want to add a new button to the Sketch panel so it will be easy for the user to find. Because an add-in can be loaded at startup, it can add its custom commands to the user interface whenever Fusion starts up, so they are always available to the user and appear as a standard Fusion command. This is described in more detail in the [User Interface](#) topic.

A second area of the API that is useful for add-ins is commands. Commands are not limited to add-ins, and there are sometimes reasons to use the command functionality within a script, but they are typically used and make more sense within an add-in. This is described in the [Commands](#) topic.

A third area of the API that is useful for add-ins is events. Various events can be subscribed to, and then Fusion will fire the event when a specific action occurs within Fusion. Because the program needs to be running to receive the event, add-ins are the logical choice since they run throughout the Fusion session.

Editing and Debugging

For more detailed information about editing and debugging your scripts and add-ins, see the language-specific topics ([Python](#) or [C++](#)). The process is different depending on which programming language you're using.

Known Issues

Here's a list of known issues that either couldn't be addressed in time for this release or are waiting for feedback to determine the best solutions.

- Starting an add-in should not dismiss the dialog.

- Opening a script or add-in using the "Open in code editor" command should dismiss the Script and Add-Ins dialog.
- Selecting the "Favorites" filter should clear all existing filters to show all favorites. Additional filters can then be selected to limit the favorites shown.
- Should the dialog remember the filters from the last time the "Scripts and Add-Ins" command was run?
- Sorting the list using the names currently considers the case of the name. An example of the result is shown below, which is not ideal, and it should ignore the case when sorting.
 - 3D Print Utility
 - A Sample Script
 - Find Part
 - Zip Files
 - another test add-in
 - script test
- The icon for a newly created script or add-in is not displayed in the "Scripts and Add-Ins" dialog until Fusion is restarted.
- Possibly remove the "Version" column from the list. It's not helpful to sort this column, and it's available in the properties section.
- Possibly remove the "Run" button from the properties section. It seems confusing to duplicate it here.
- The ability to set the version is missing in the "New App" dialog. Is this desired?
- Is there any reason to be able to choose if a script or add-in should only be shown on Windows, Mac, or both?
- Remove the ID property from the manifest and the documentation. It's not used and is confusing.



Except where otherwise noted, this work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License](#). Please see the [Autodesk Creative Commons FAQ](#) for more information.

© 2025 Autodesk Inc. All rights reserved