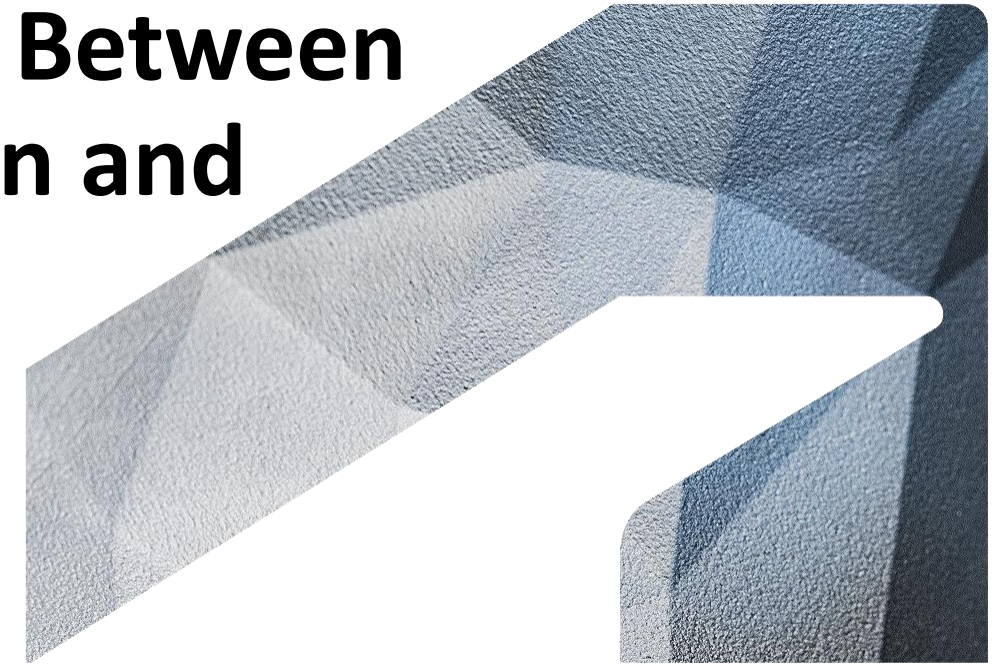
AU 2023

# Bridging the Gap Between iLogic Automation and Inventor Add-Ins

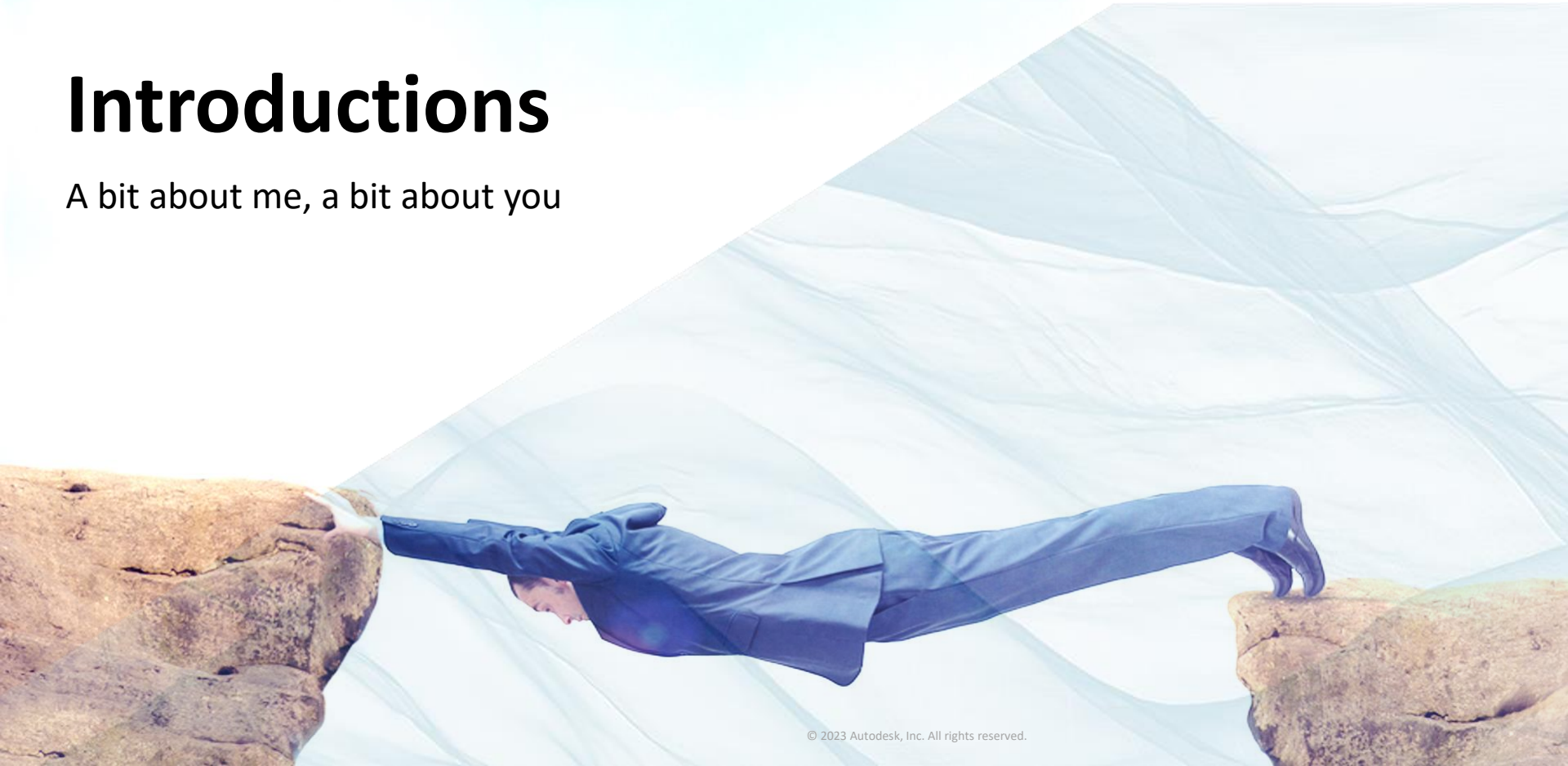
MFG601910



Curtis Waguespack
Automation Solution Consultant Team D3

# Introductions

A bit about me, a bit about you



# About Me

## Curtis Waguespack [wag] [əs] [pak]

Expert Elite Member

- I've used Inventor and AutoCAD for over 20 years, designing a number of manufactured products.
- This has included a great deal of Inventor automation with iLogic.
- I started out modifying LISP routines and writing VBA for AutoCAD.
- I currently work as an Automation Solution Consultant, where I help professionals like yourself automate their designs.
- I teach iLogic classes, and in the past, I have taught general Inventor and AutoCAD classes.
- Additionally, I've authored and co-authored multiple editions of the **Mastering Autodesk Inventor** book.


# About You

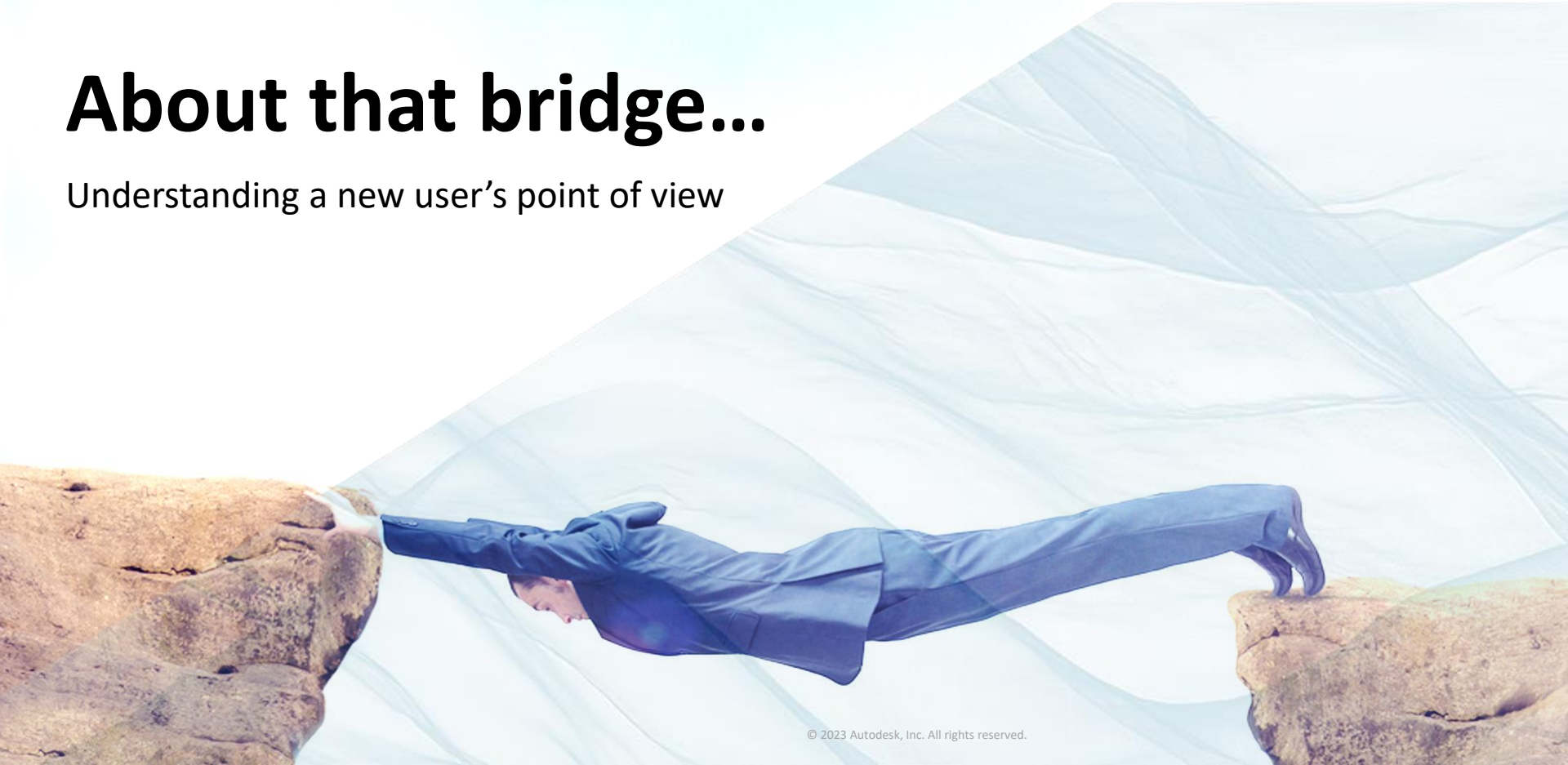
The Intended Audience

**This presentation is aimed at anyone who has:**

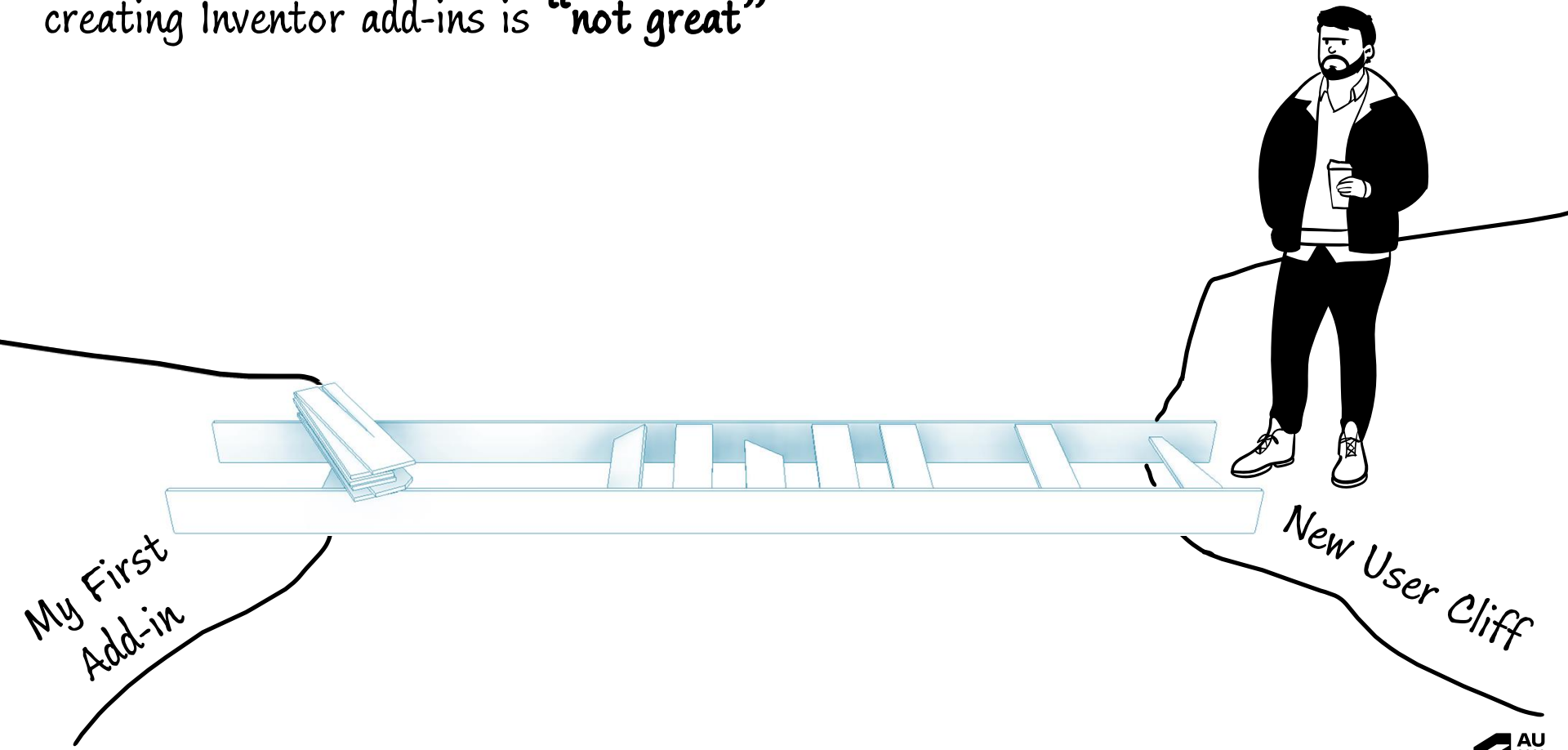
- existing iLogic rules in place
- run up against the limitations present within the automation tools offered with iLogic
- struggled to take things to the next level and create Inventor add-ins in the past
- concerns about having to start over with Inventor automation

# About that bridge…

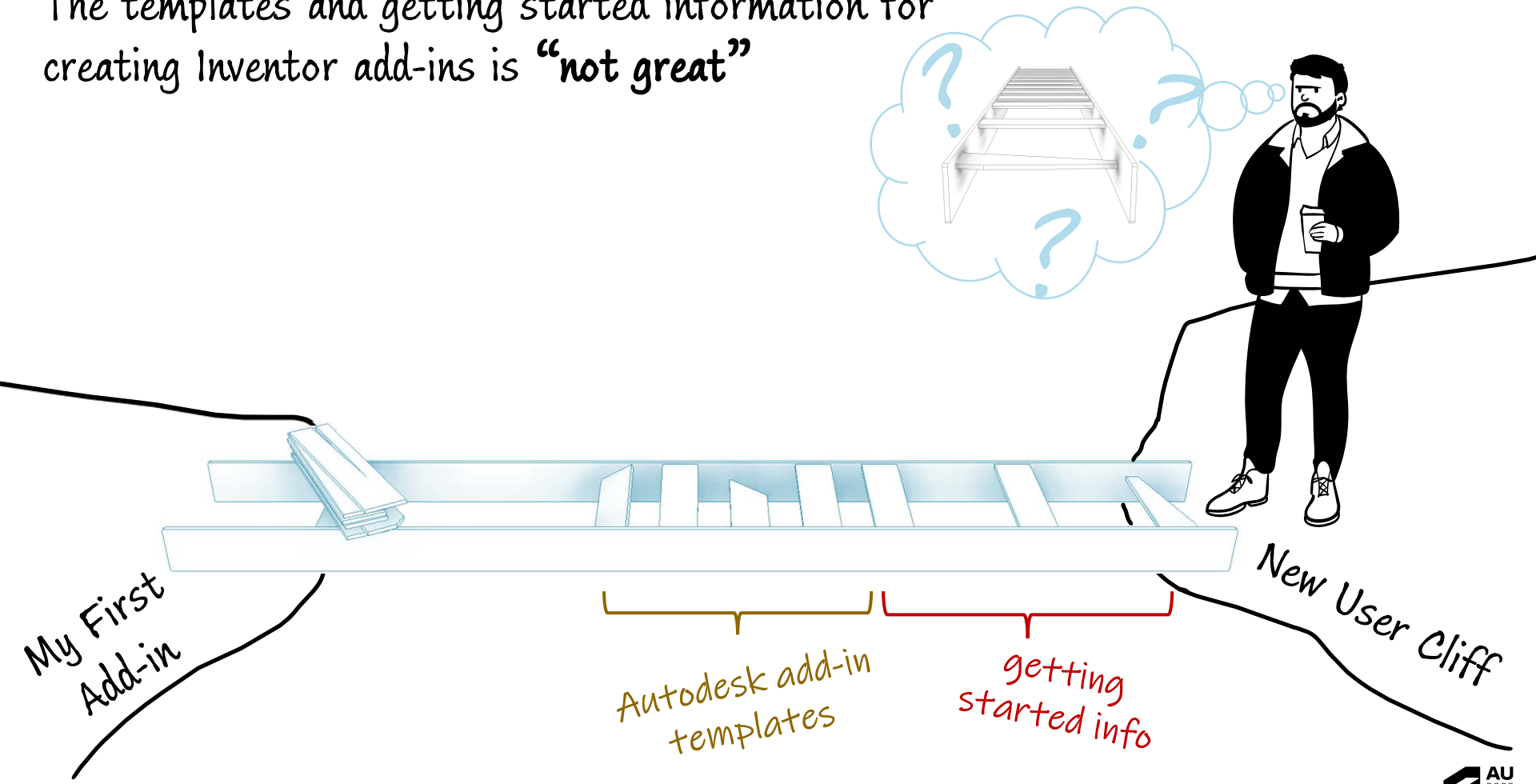
Understanding a new user’s point of view



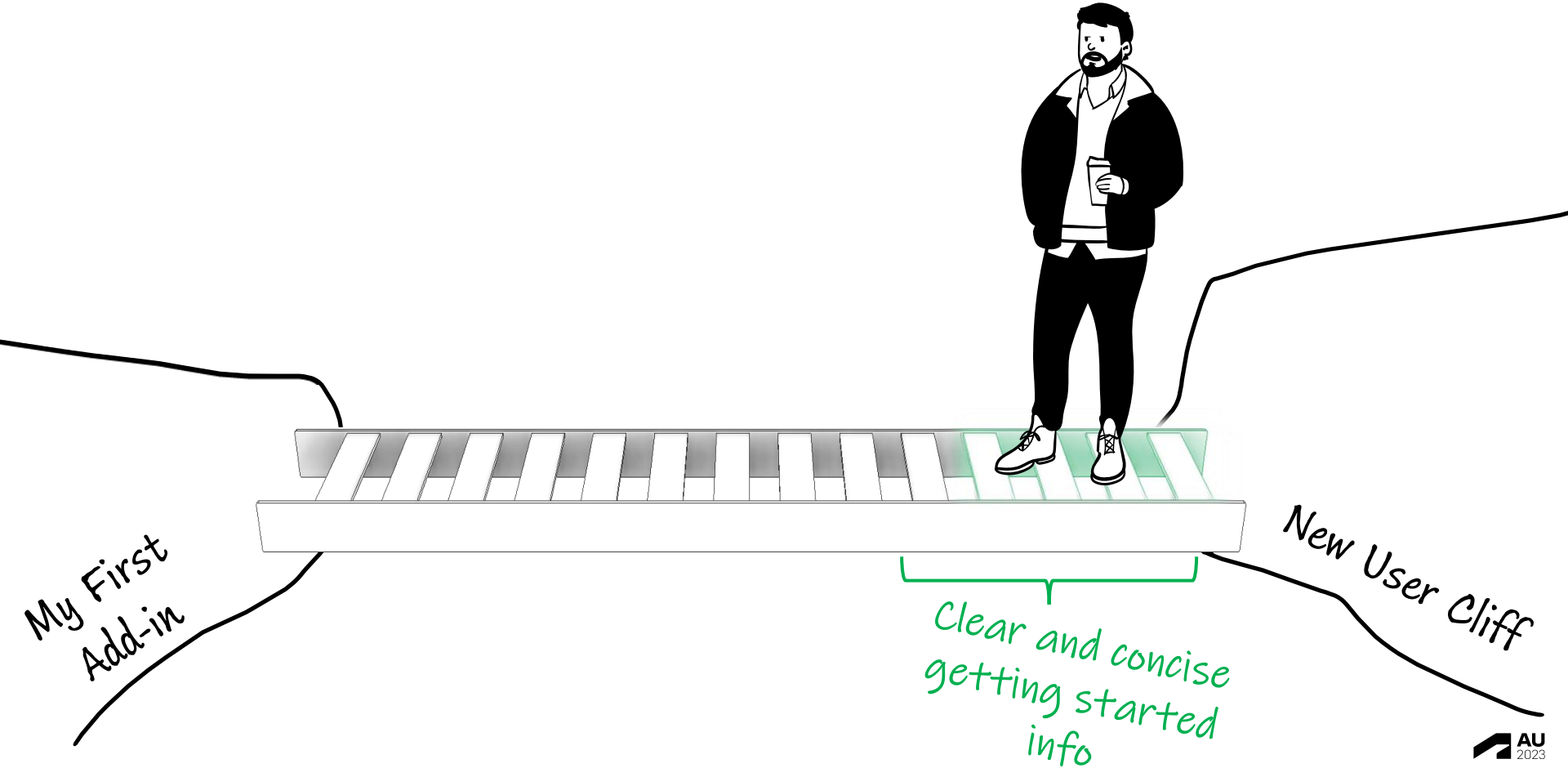
The templates and getting started information for creating Inventor add-ins is **“not great”**



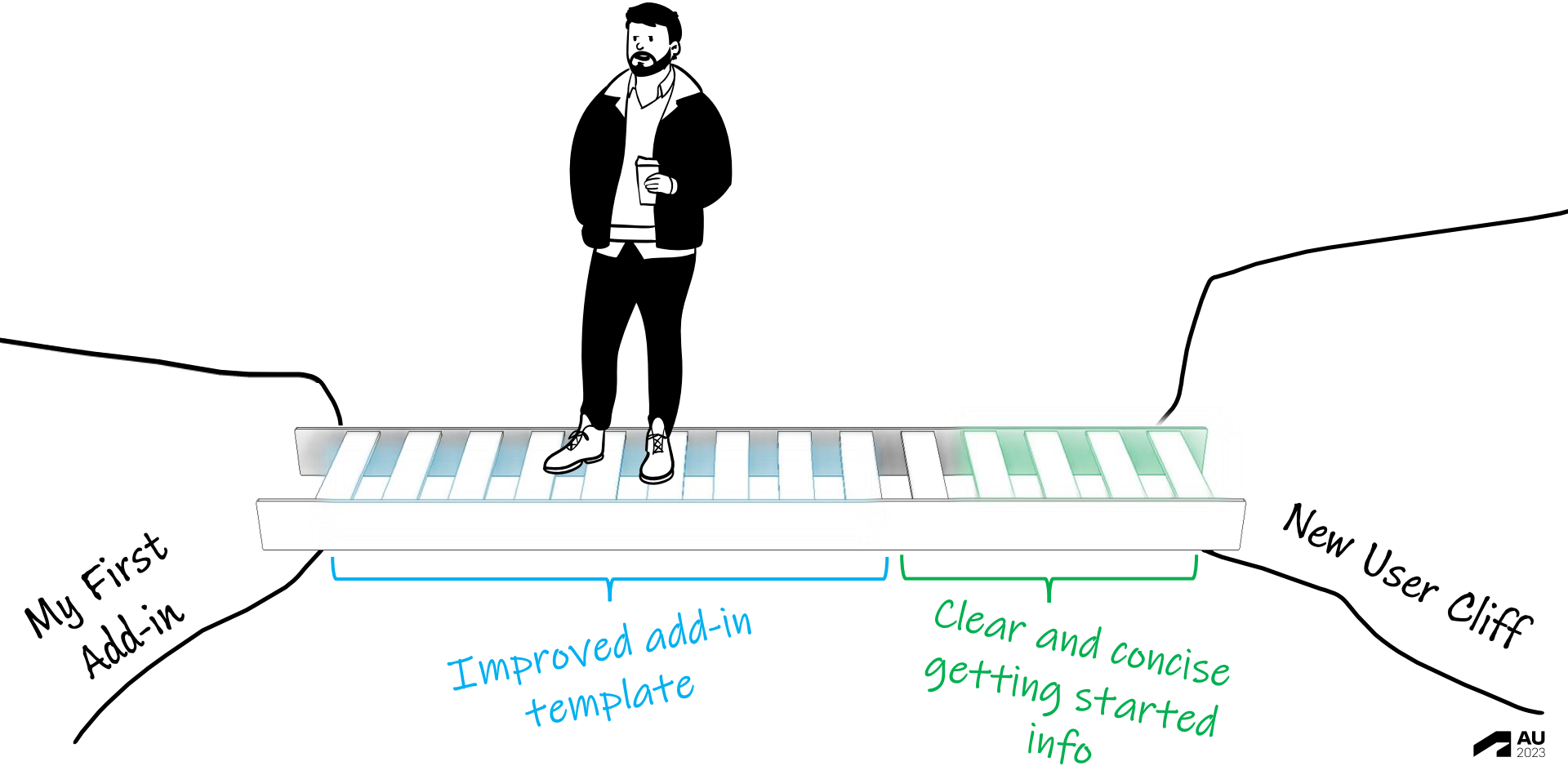
The templates and getting started information for creating Inventor add-ins is **“not great”**



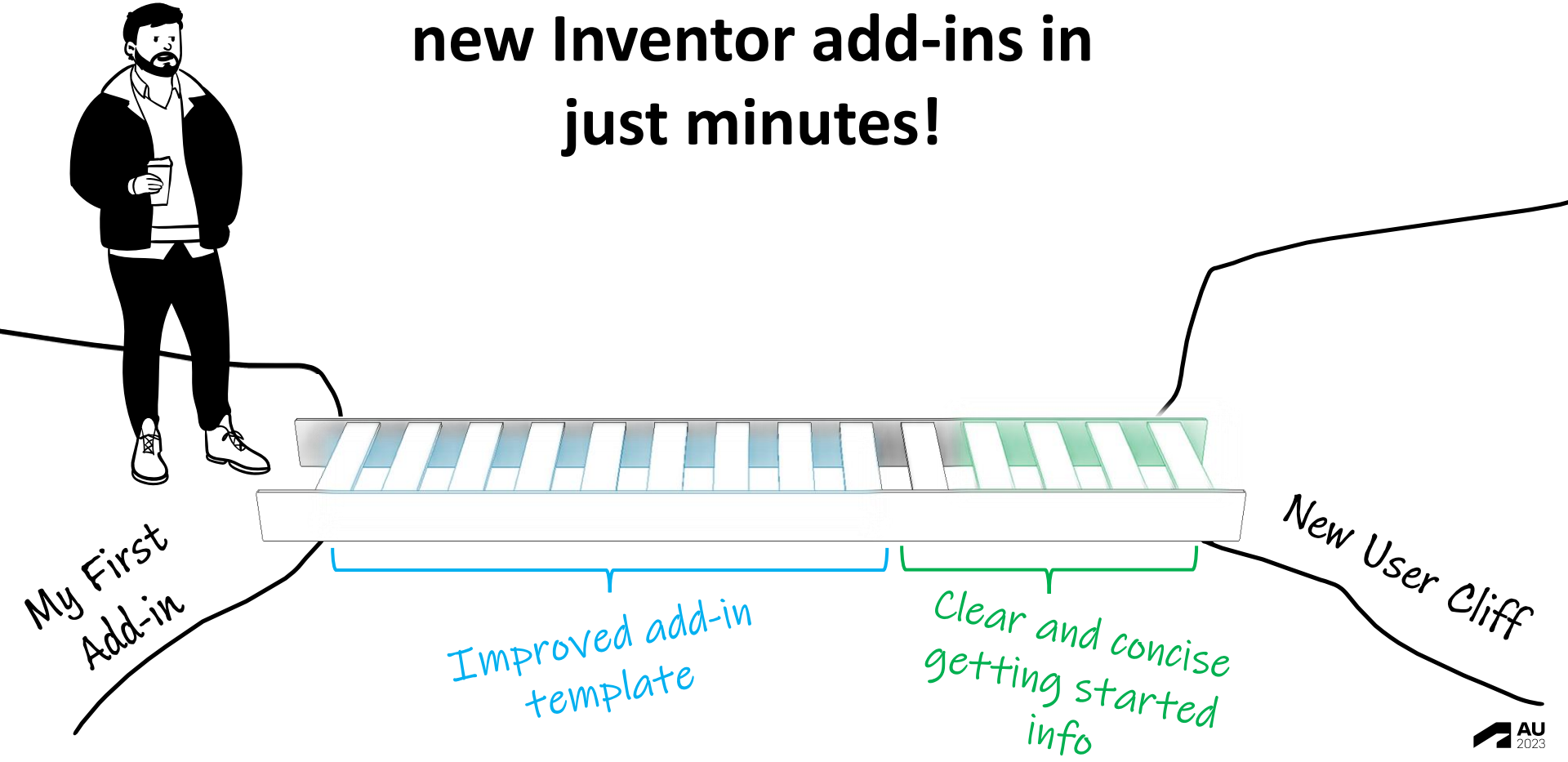
We're going to bridge this gap together.



We're going to bridge this gap together.



# Goal: be able to create new Inventor add-ins in just minutes!



# Why would we want to move to add-ins?

Why not just use iLogic?

## Motivations And Challenges

# Motivations

- Better Debugging Environment
- Would like to offer users a more organized & congruent interface & experience
- More control over events. Example: Trigger on the edit of a Parts List
- Would like to use C#

# Challenges

- Don’t have the time to start over completely.
- Fall back on what we’ve been doing.
- Don’t understand how to get started.
- Can’t get the add-in templates to load.
- How do we use the add-in templates?

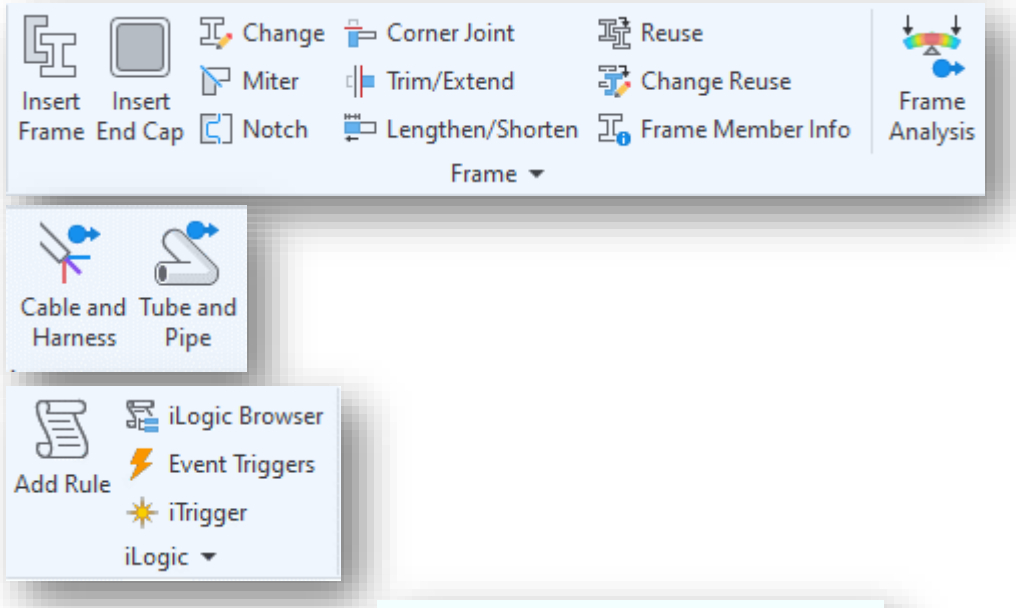
AU 2023

# Understanding Inventor Add-ins

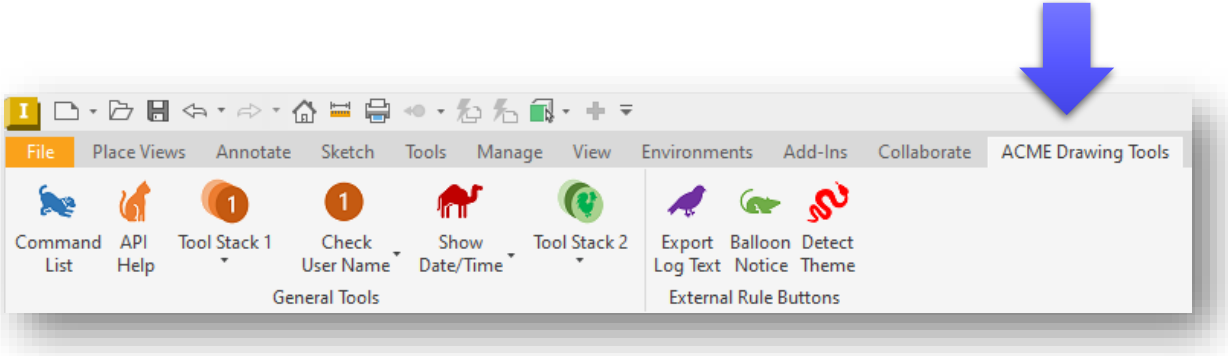

- Also called a plug-in ( the 2 terms are interchangeable)
- Written and debugged in Visual Studio
  - Visual Studio is a stand-alone IDE application
    - IDE – Integrated Development Environment
- The Add-in is compiled to a DLL file and loaded into the Inventor process on startup of the Inventor application
- Add-ins are integrated into the Inventor’s user interface
- Add-ins are integrated with all of Inventor’s events
- Add-ins are typically written in VB.net or C#

# Examples of Add-ins you might be familiar with

- Frame Generator
- Cable and Harness
- Tube and Pipe
- iLogic
- Inventor App Store downloads



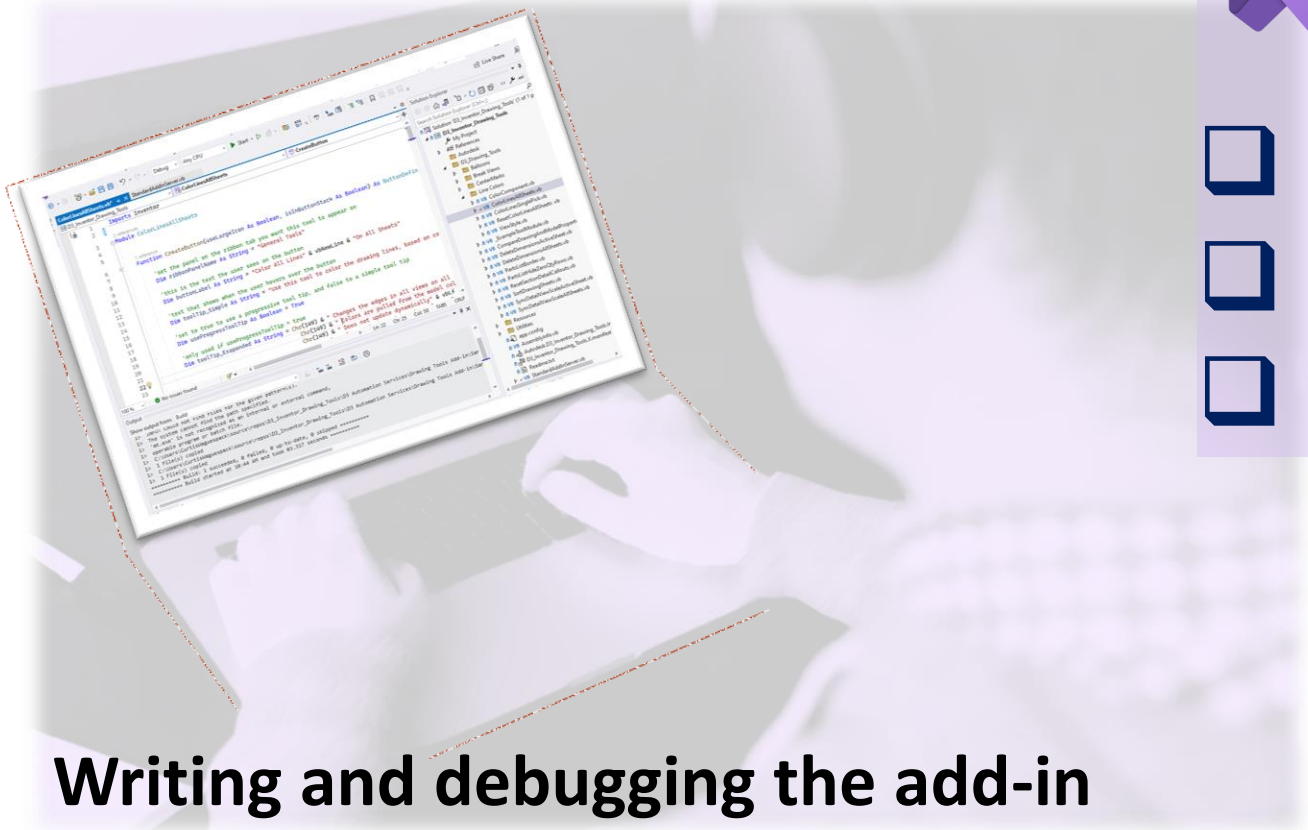
# Custom add-in




- A custom “enterprise” tab for your tools in each environment
- Buttons to run code from the add-in
- Buttons to run external iLogic rules

# How do we create an add-in?

# How do we create an add-in?

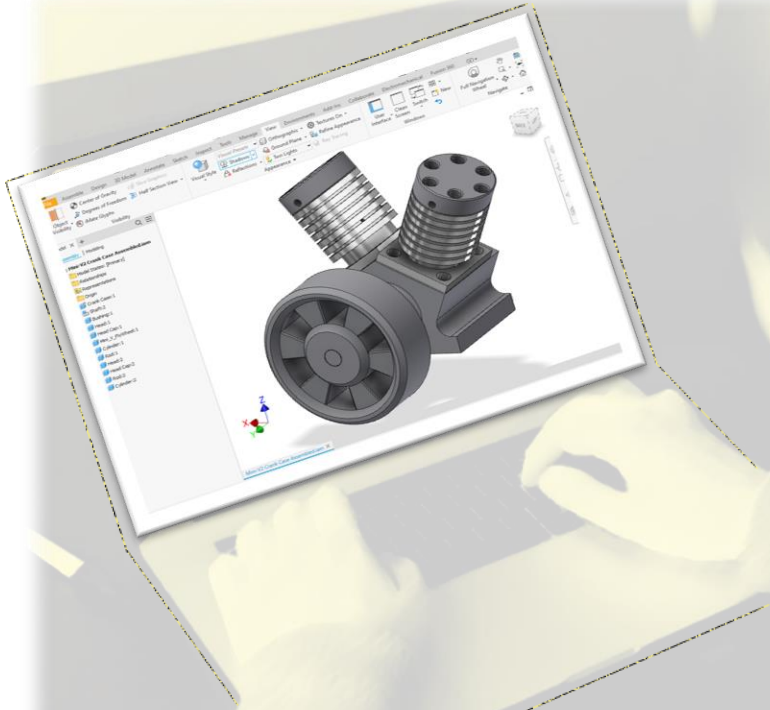


Visual Studio

- ❑ Write Code
- ❑ Debug
- ❑ Compile

## Writing and debugging the add-in

# How do we create an add-in?



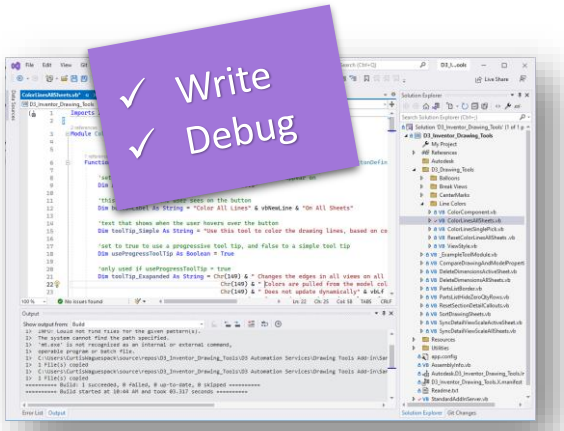
Inventor

- ❑ Start Inventor
- ❑ Make sure the Add-in Loaded
- ❑ Use Add-in

## Loading and using the add-in

# Writing the add-in

Visual Studio




Autodesk Application Plugins Folder

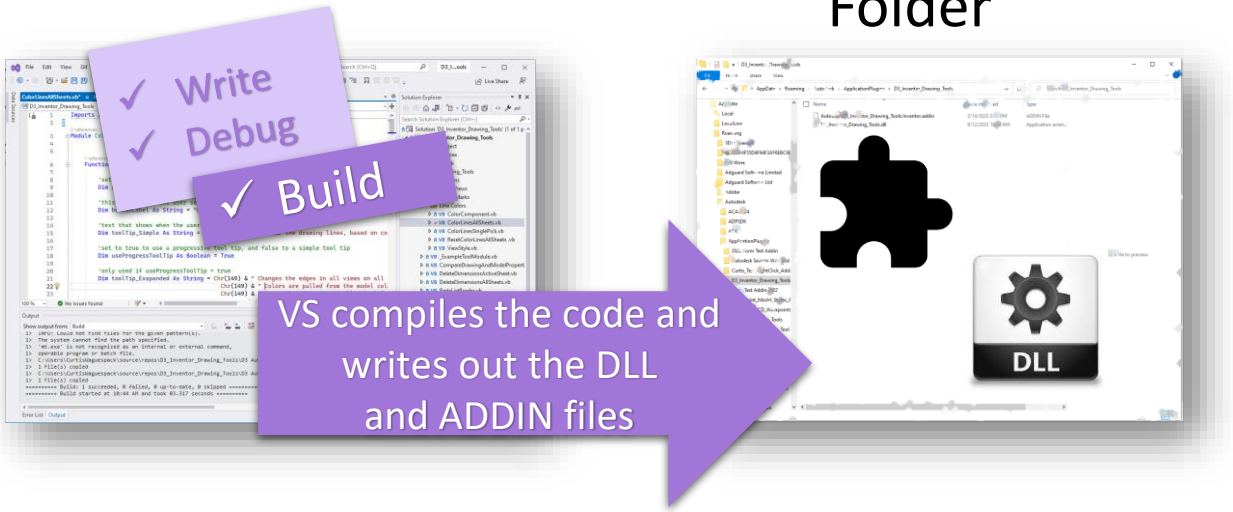
Autodesk Inventor

# The hand off

Visual Studio

Autodesk Application Plugins Folder

Autodesk Inventor

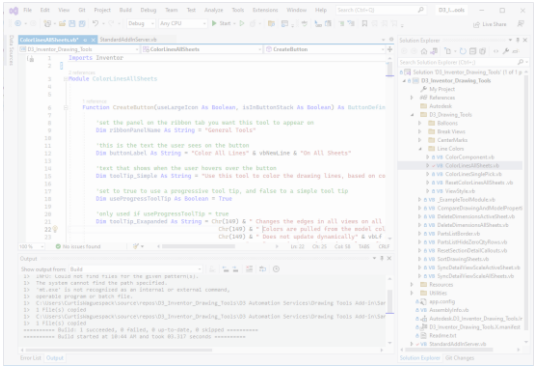


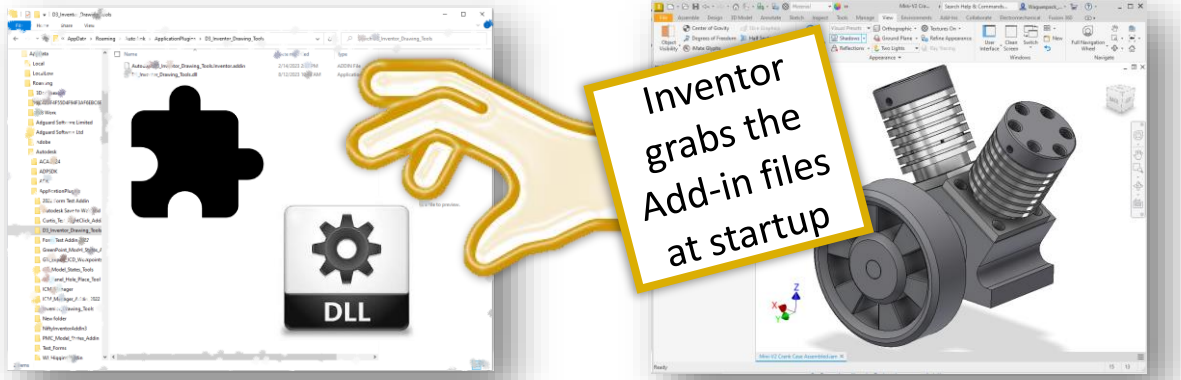
# Loading the add-in

Visual Studio

Autodesk Application Plugins Folder

Autodesk Inventor



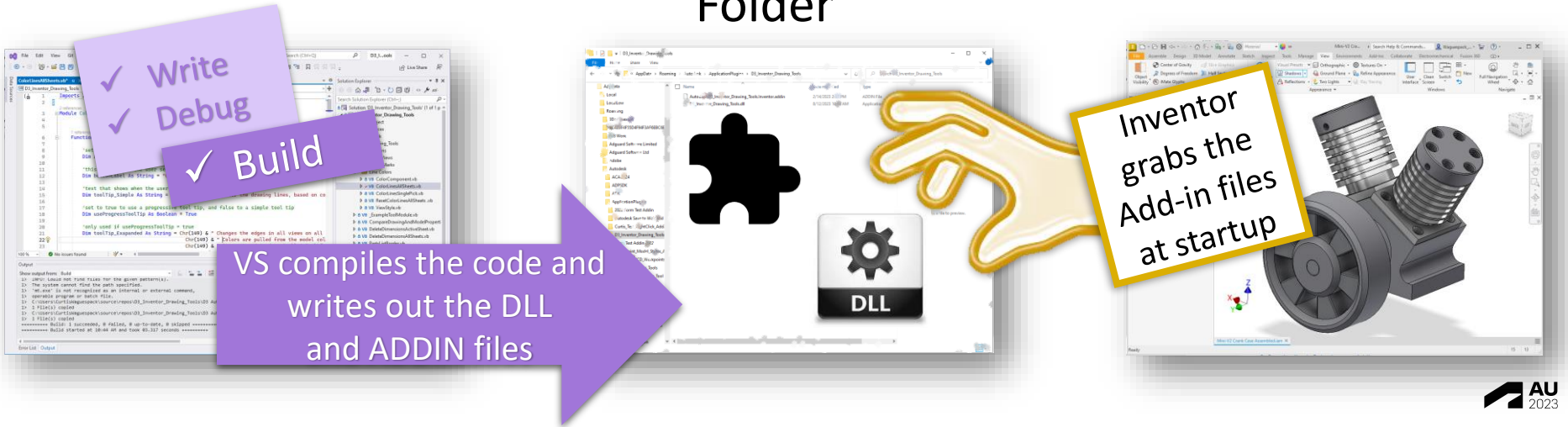


# Overall Process

Visual Studio

Autodesk Application Plugins Folder

Autodesk Inventor



# Getting Started

Installing Visual Studio and the Inventor SDK tools

*SDK = Software Development Kit

# Steps to get started…

… with Visual Studio and the Inventor Software Development Kit

| 1 | 2 | 3 |
| --- | --- | --- |
| Install Visual Studio | Install the Inventor SDK | Ensure the Inventor Add-in Templates are Loaded |

# Can we install the Inventor SDK before Visual Studio?

2 ⟷ 1

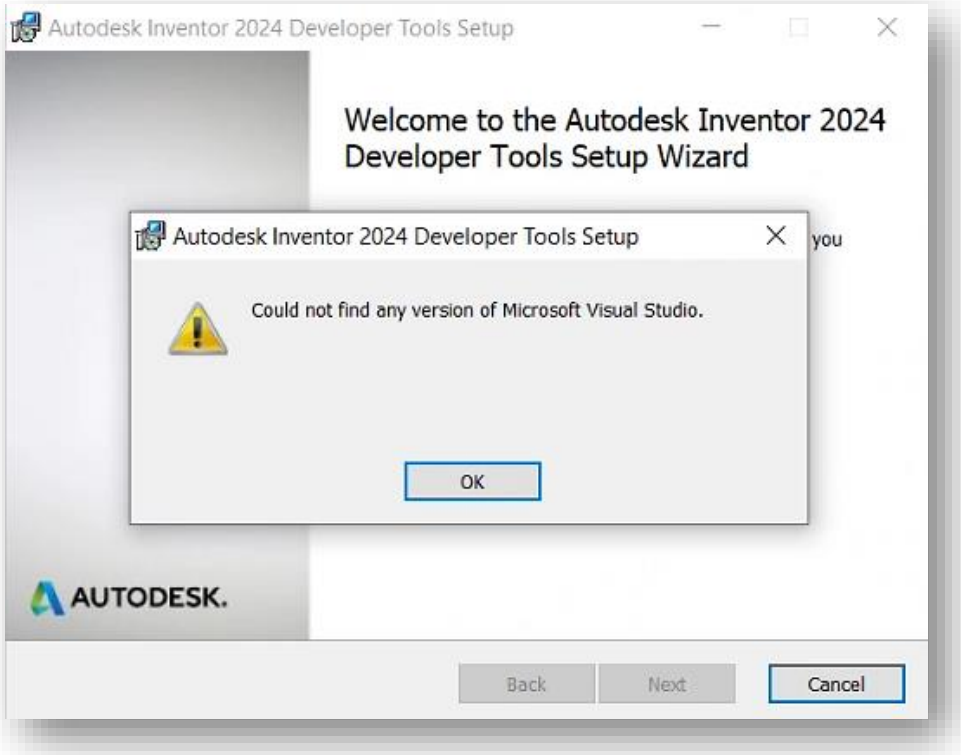
3

Install Visual Studio

Install the Inventor SDK

Ensure the Inventor Add-in Templates are Loaded

# Can we install the Inventor SDK before Visual Studio?

# Steps to get started…

… with Visual Studio and the Inventor SDK

- [ ] Install Visual Studio
- [ ] Extract the Inventor SDK ( software development kit)
- [ ] Ensure the Inventor Add-In Templates show up in Visual Studio

# Install Visual Studio

## Install Visual Studio

Article • 05/16/2023 • 20 contributors

Feedback

In this article

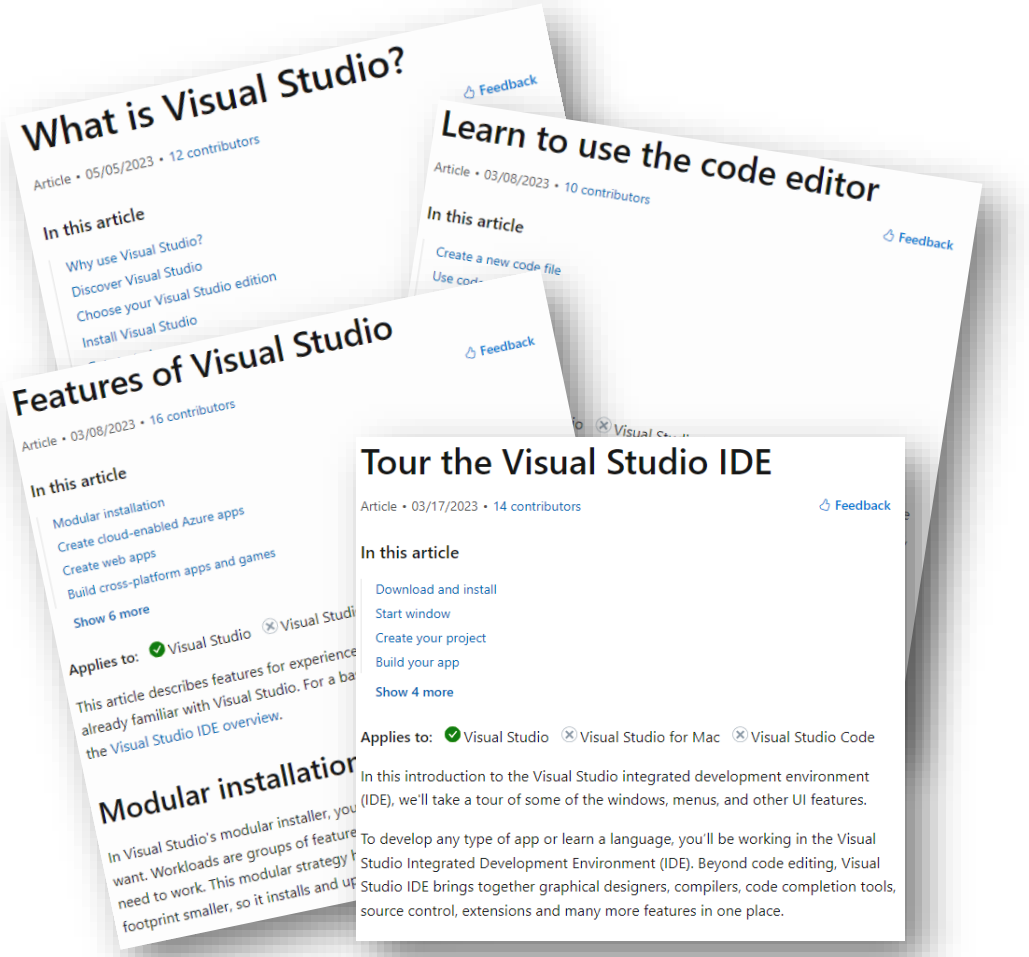
- Step 1 - Make sure your computer is ready for Visual Studio
- Step 2 - Determine which version and edition of Visual Studio to install
- Step 3 - Initiate the installation
- Step 4 - Choose workloads
- Step 5 - Choose individual components (optional)
- Step 6 - Install language packs (optional)
- Step 7 - Select the installation location (optional)
- Step 8 - Start developing
- Support or troubleshooting
- See also

Show less

Applies to: Visual Studio Visual Studio for Mac Visual Studio Code

Welcome to Visual Studio 2022! In this version, it's easy to choose and install just the features you need.

https://learn.microsoft.com/en-us/visualstudio/ide/?view=vs-2022



# Which version of Visual Studio should I install?

**Visual Studio Professional**

- Full version
- Purchased from Microsoft
- Full Use Licensing

**Visual Studio Community**

- Full Version
- Free from Microsoft
- Limited Use Licensing

**Visual Studio Express**

- Limited version
- Free from Microsoft
- Not suitable for Inventor add-ins

# Steps to get started…

… with Visual Studio and the Inventor SDK

- [x] Install Visual Studio
- [ ] Extract the Inventor SDK ( software development kit)
- [ ] Ensure the Inventor Add-In Templates show up in Visual Studio

# Steps to get started…

… with Visual Studio and the Inventor SDK

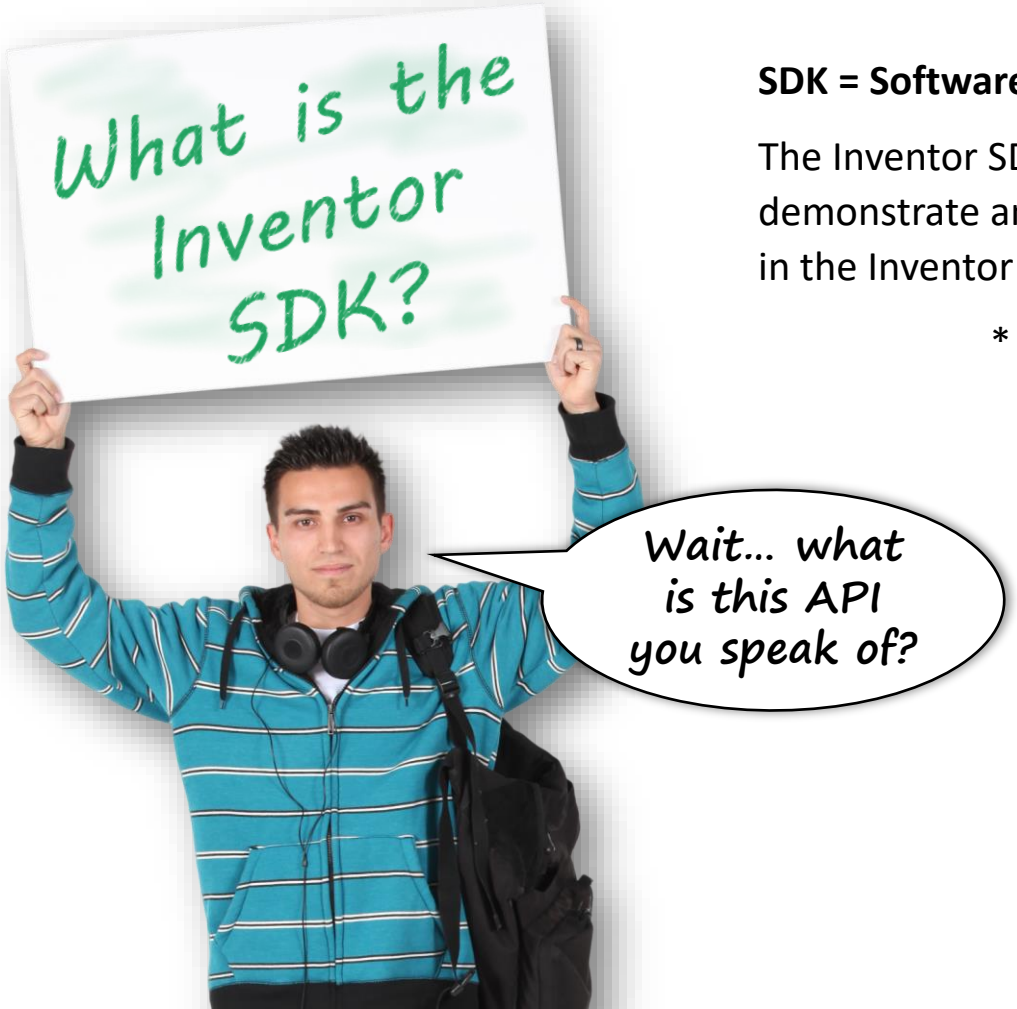
- [x] Install Visual Studio
- [ ] Extract the Inventor SDK ( software development kit)
- [ ] Ensure the Inventor Add-In Templates show up in Visual Studio


**SDK = Software Development Kit**


**SDK = Software Development Kit**

The Inventor SDK includes samples and tools to demonstrate and describe the functionality that is available in the Inventor API*

* API = Application Programming Interface




**SDK = Software Development Kit**

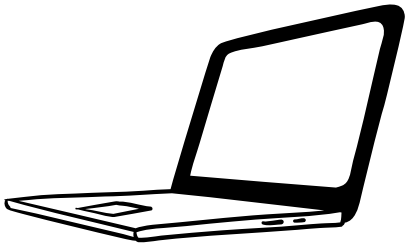
The Inventor SDK includes samples and tools to demonstrate and describe the functionality that is available in the Inventor API*

* API = Application Programming Interface

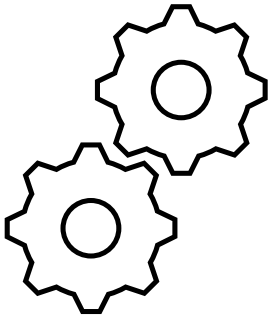
# Understanding the API

# Understanding what an API is  (Web Page Example)

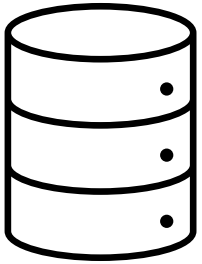
User Interface



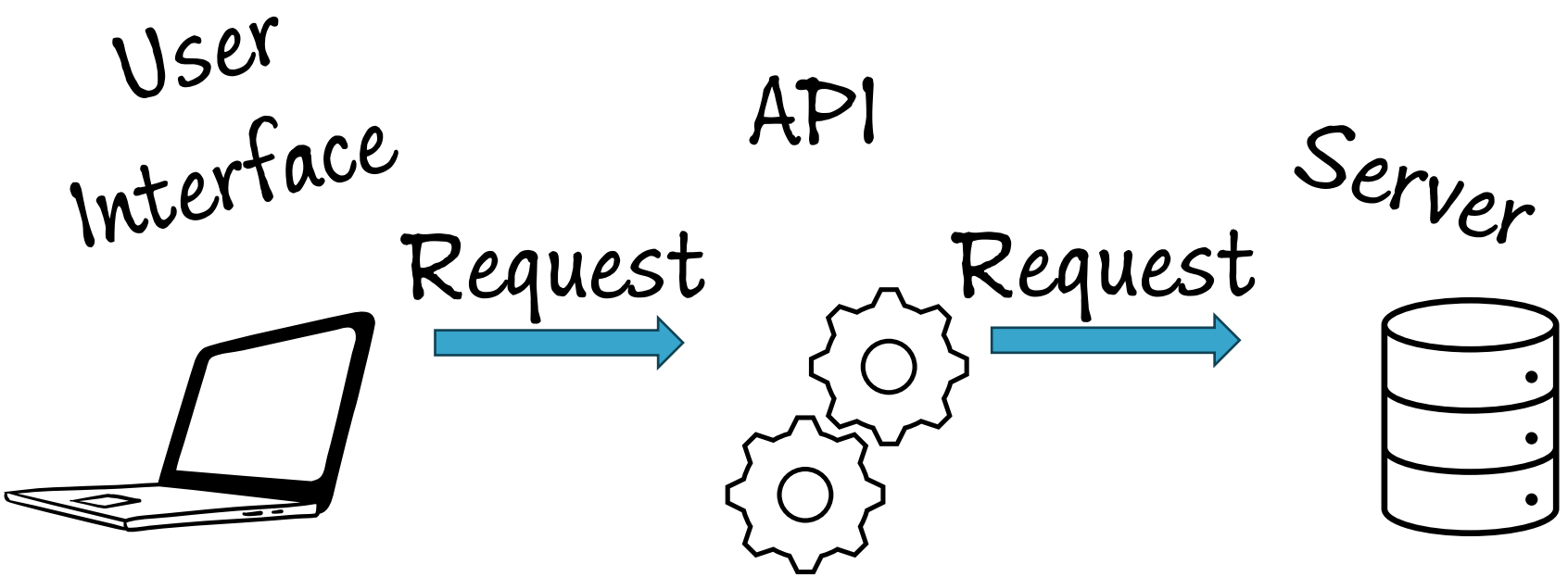
API



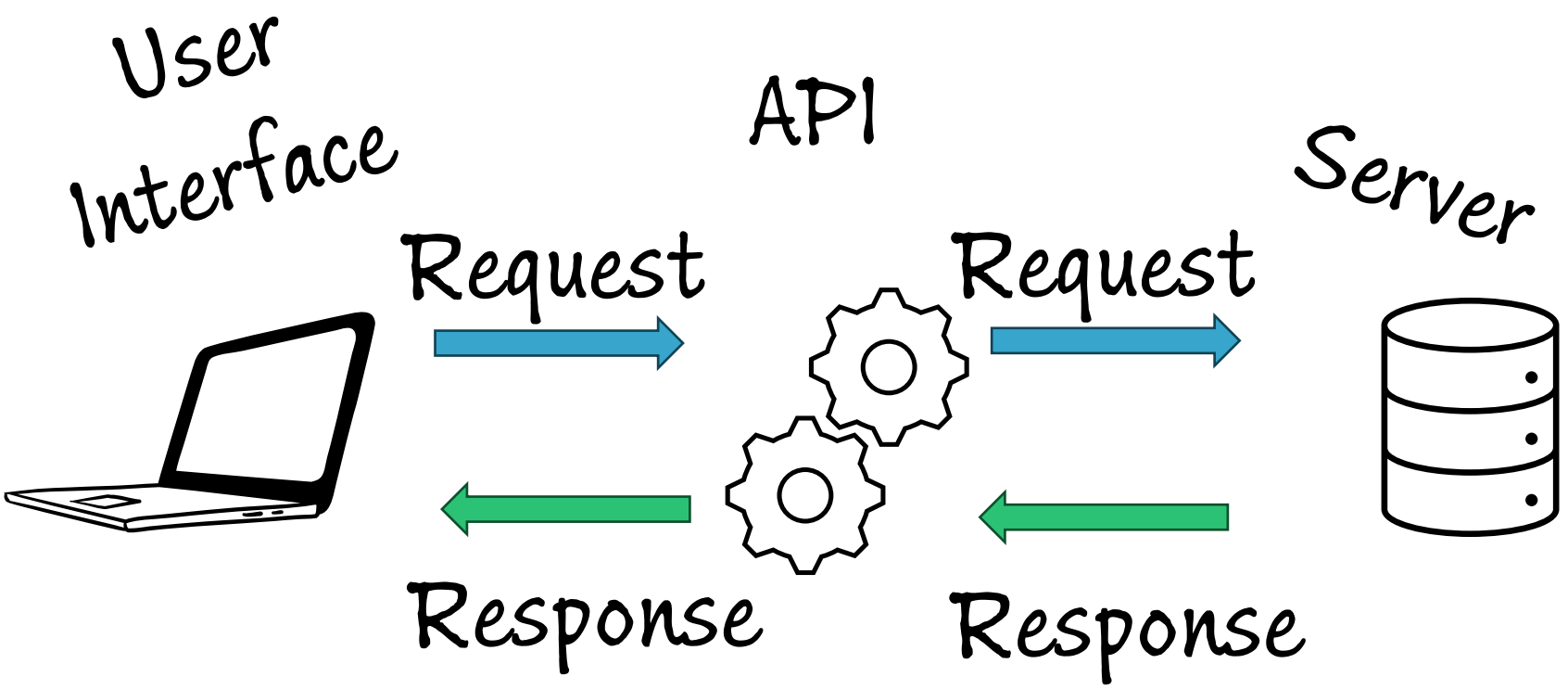
Server



# Understanding what an API is (Web Page Example)

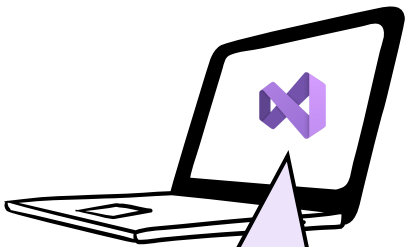


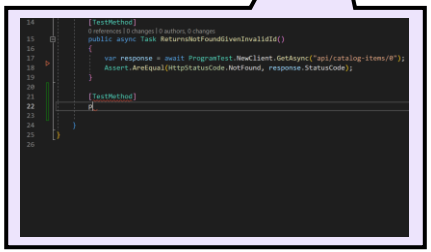
# Understanding what an API is (Web Page Example)



# Understanding what an API is (Inventor Add-in Example)

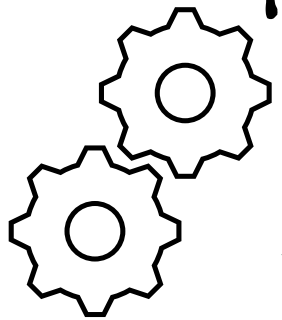
User Add-in

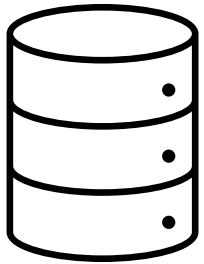




Request

Request





Response

Response

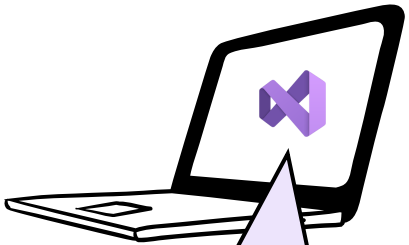
# Understanding what an API is (Inventor Add-in Example)

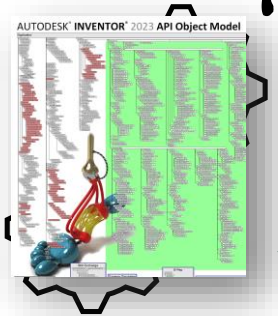
User Add-in

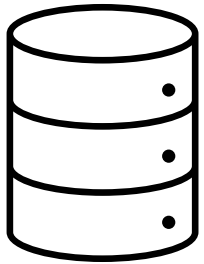
Inventor API

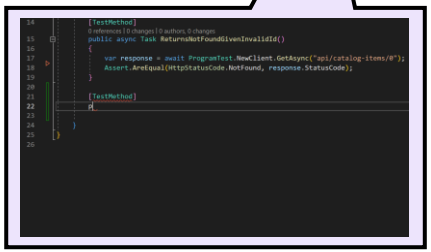
Request

Request







Response

Response

# Understanding what an API is (Inventor Add-in Example)

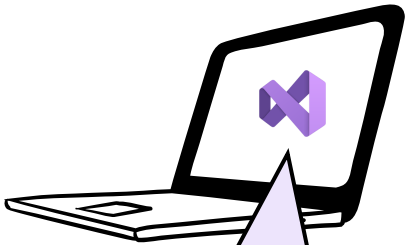
User Add-in

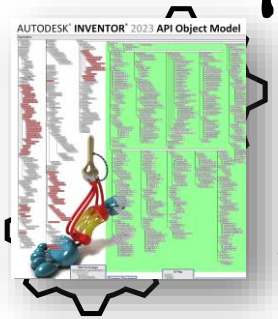
Inventor API

Application

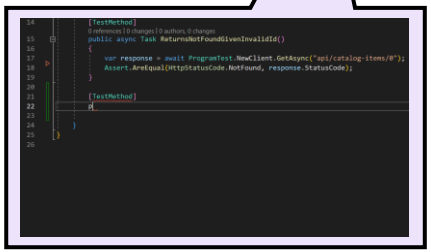
Request

Request





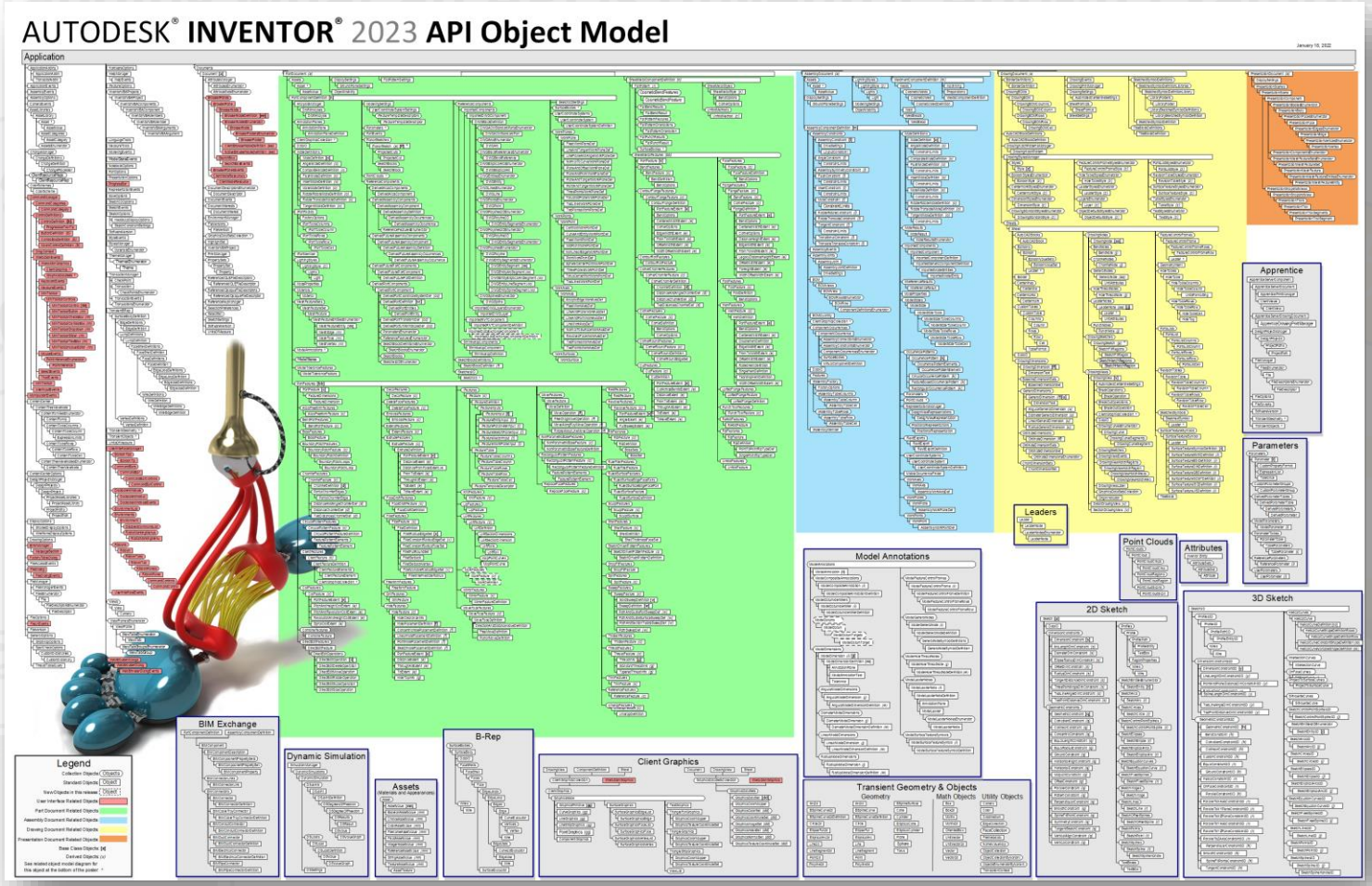


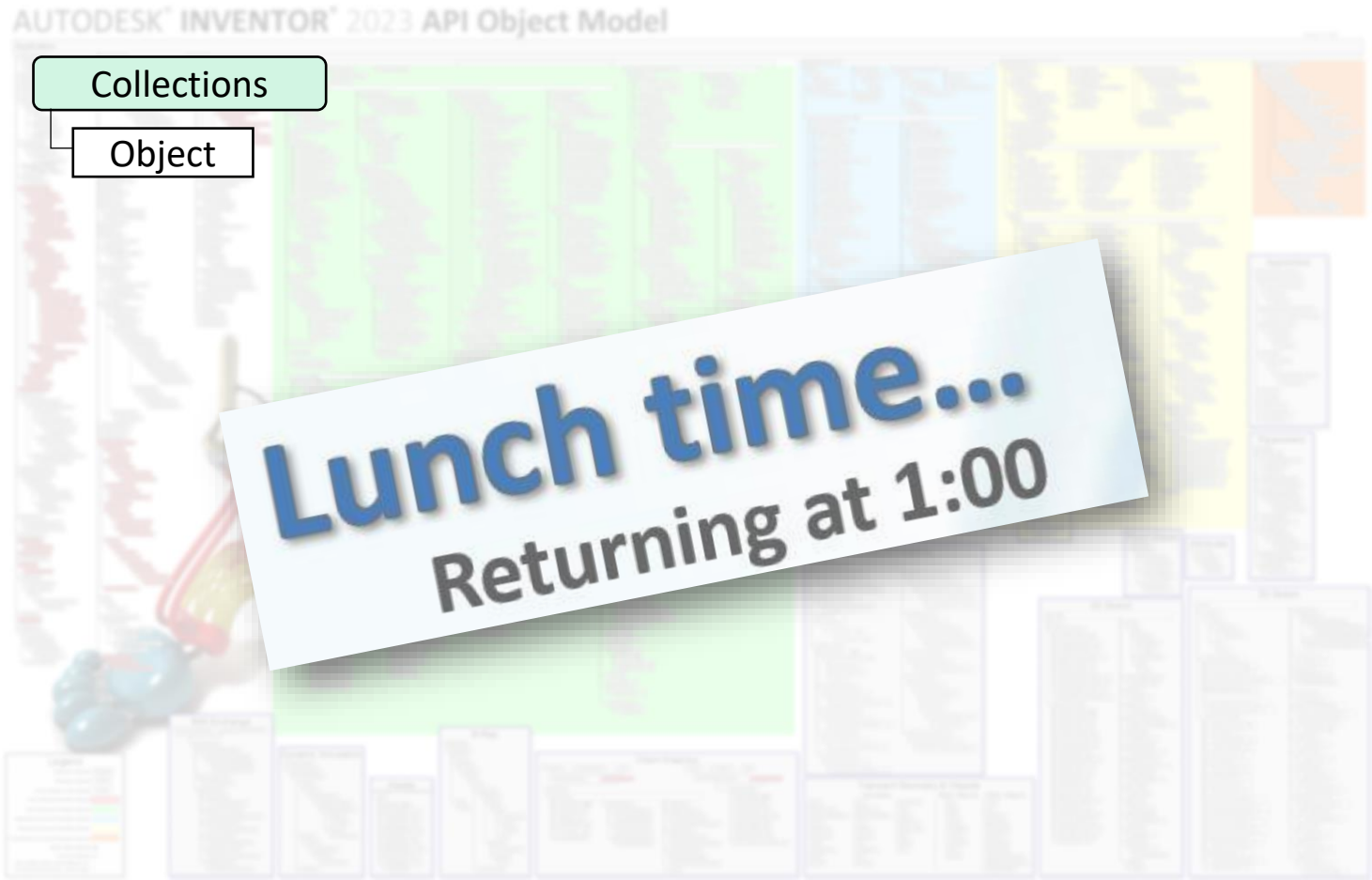


Response

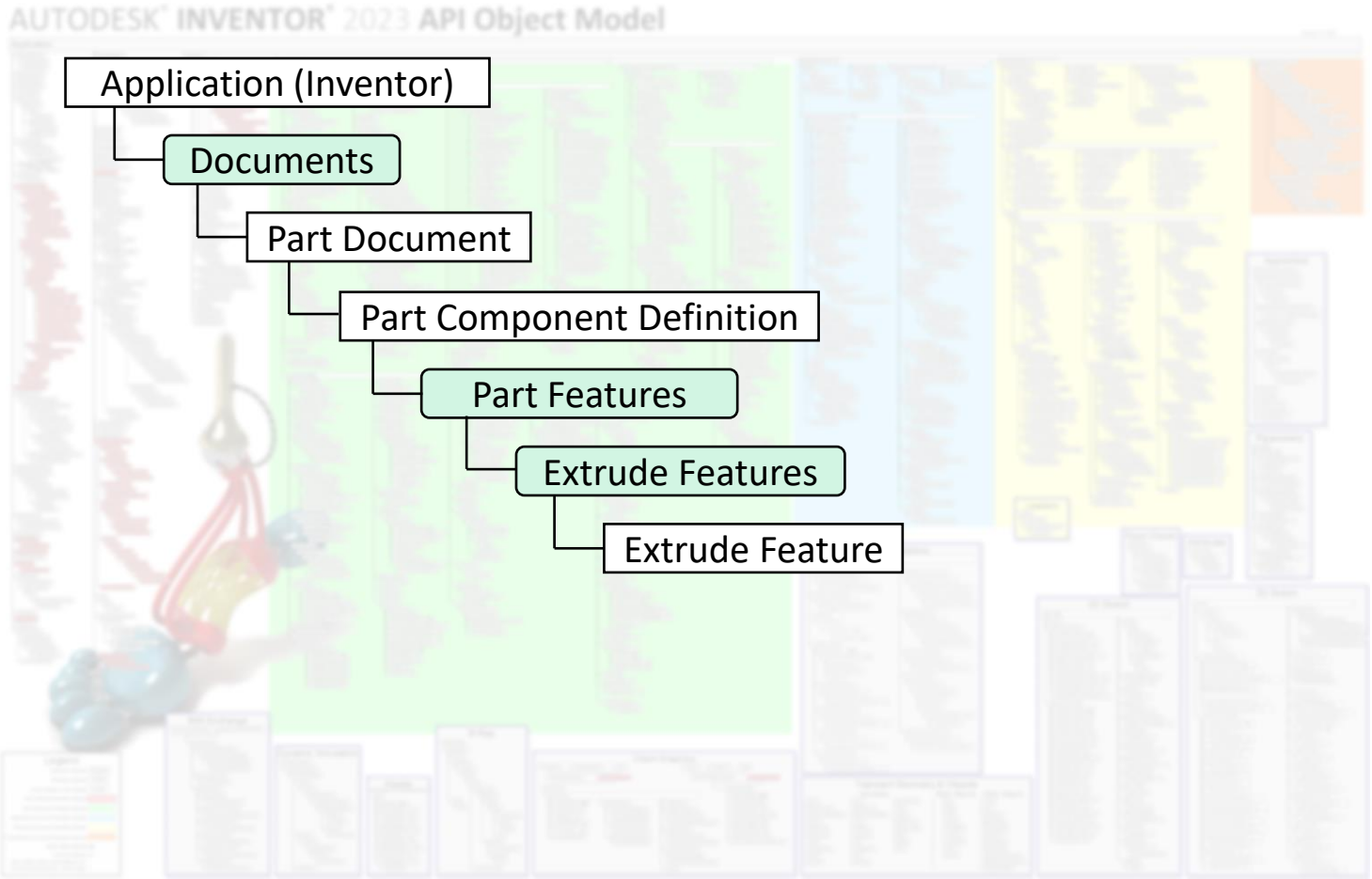
Response

# Understanding the API



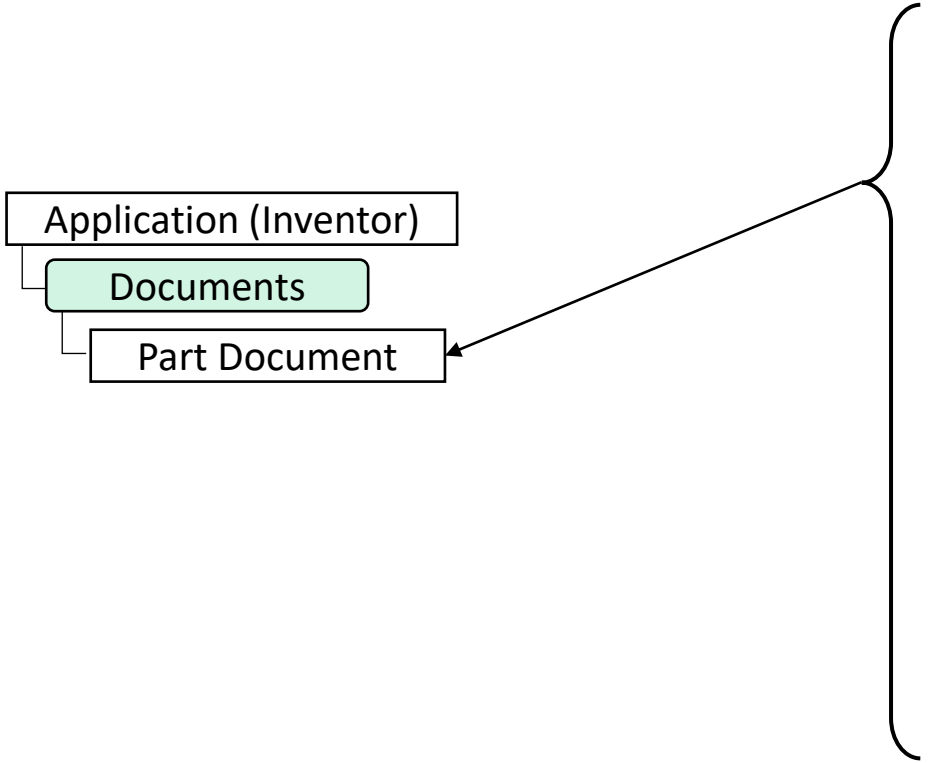


AUTODESK® INVENTOR® 2023 API Object Model
Collections
Object
Lunch time...
Returning at 1:00

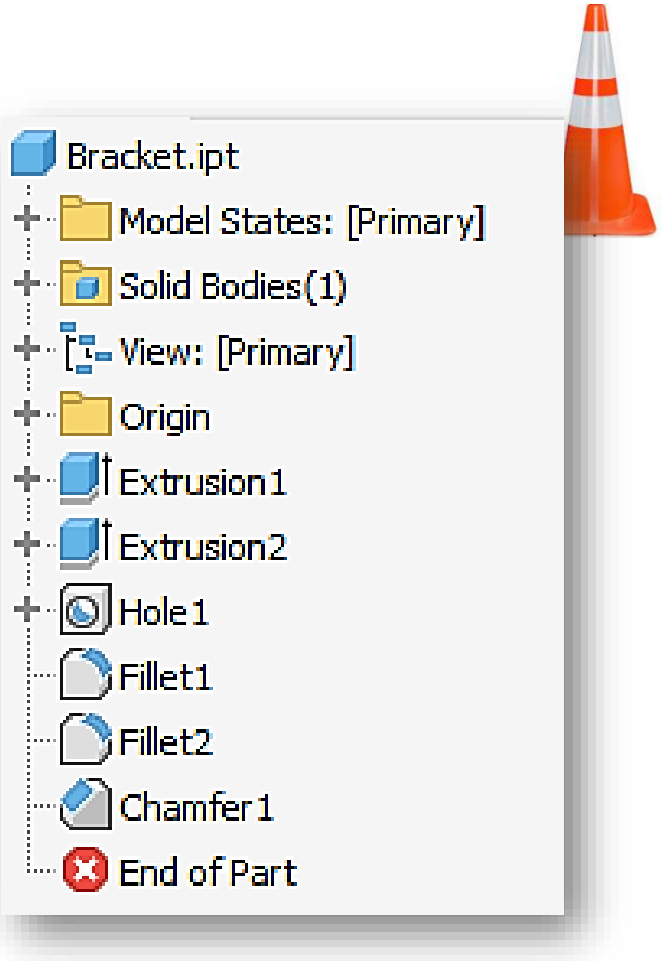
# Understanding the API



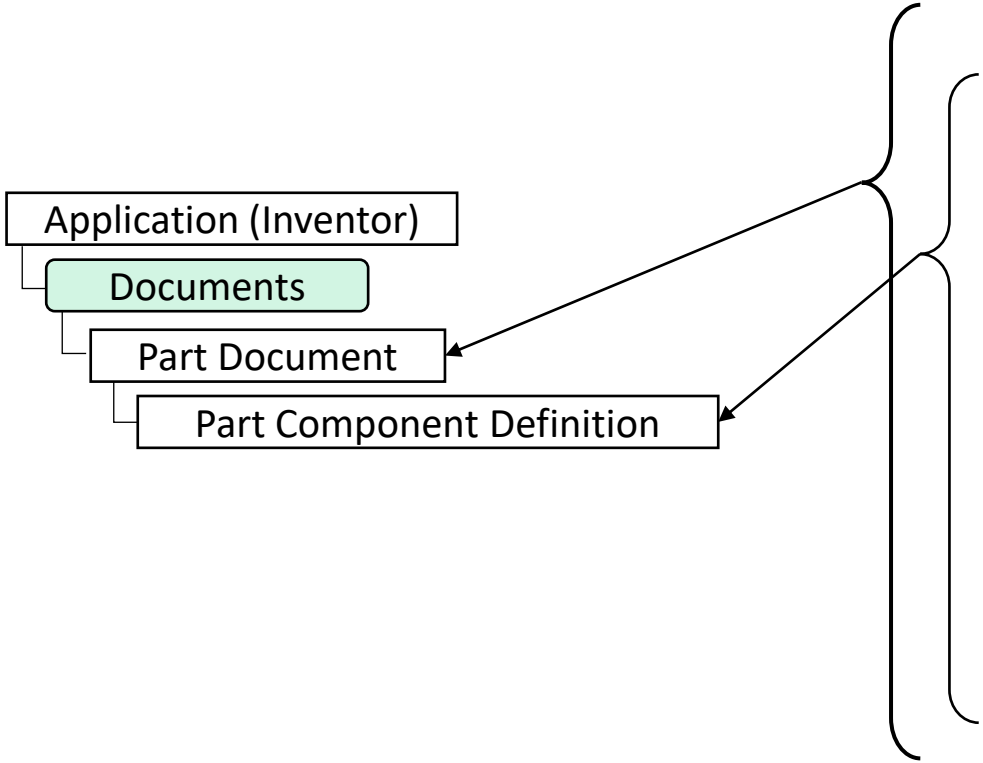

- Application (Inventor)
  - Documents
    - Part Document
      - Part Component Definition
        - Part Features
          - Extrude Features
            - Extrude Feature

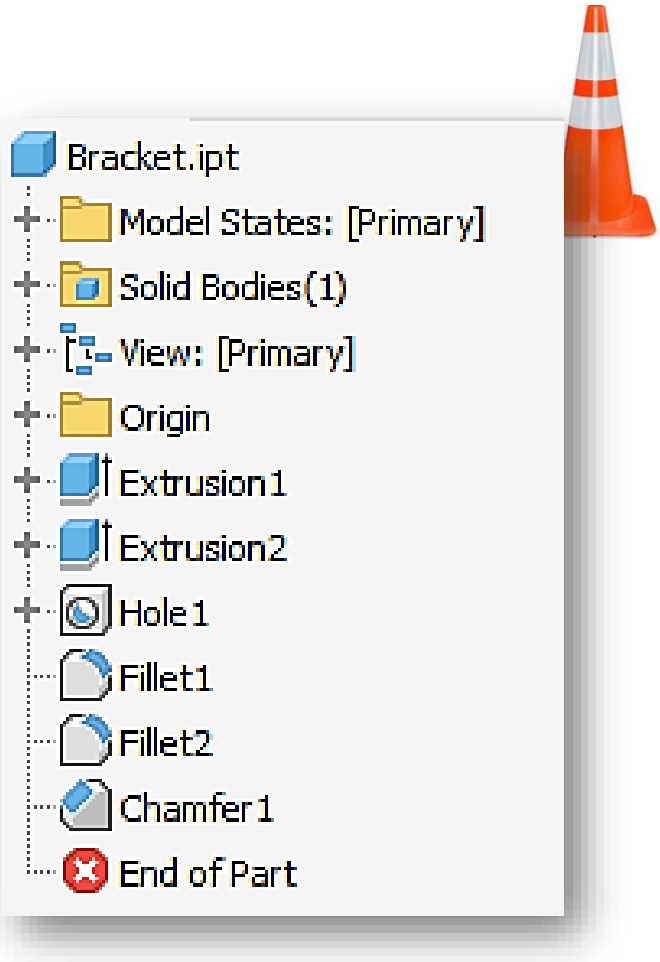
# API Object Model of a part



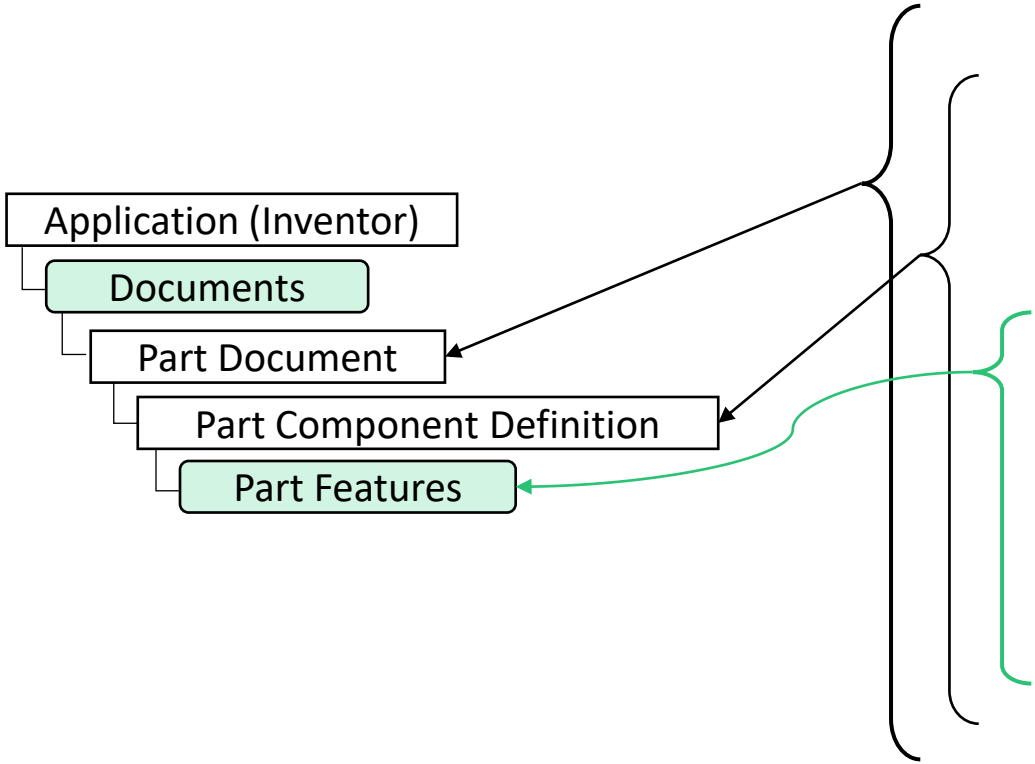



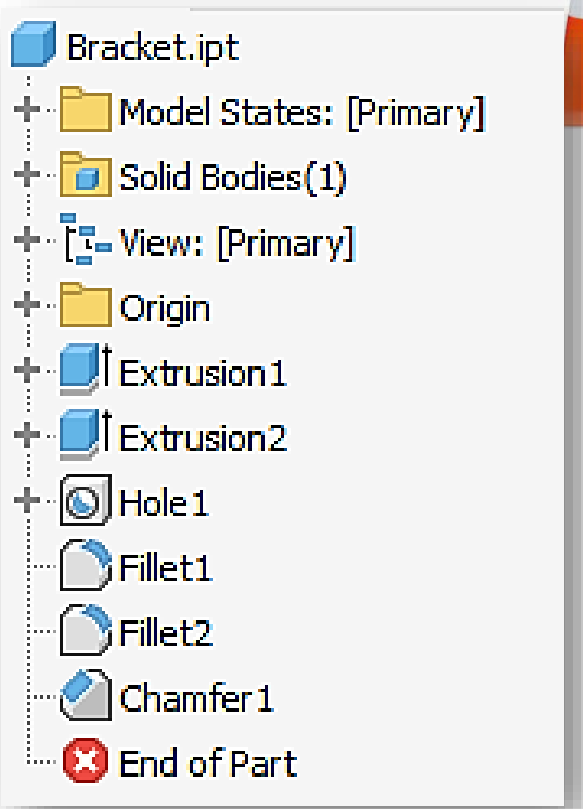
# API Object Model of a part



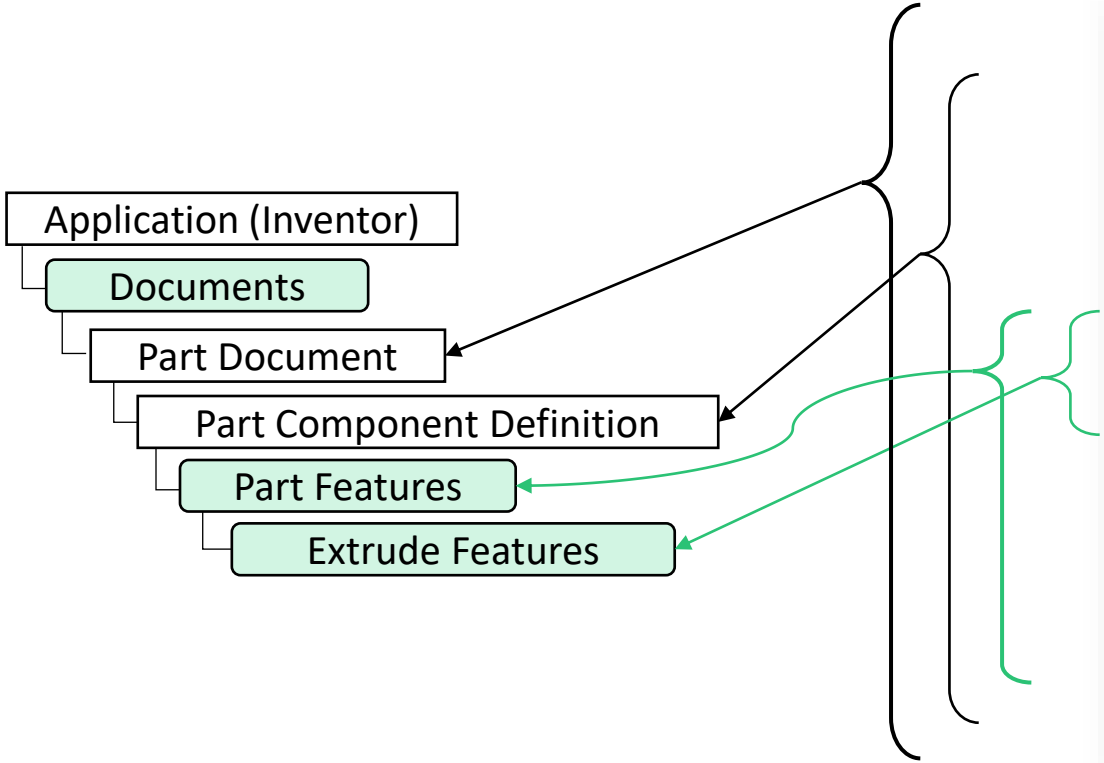



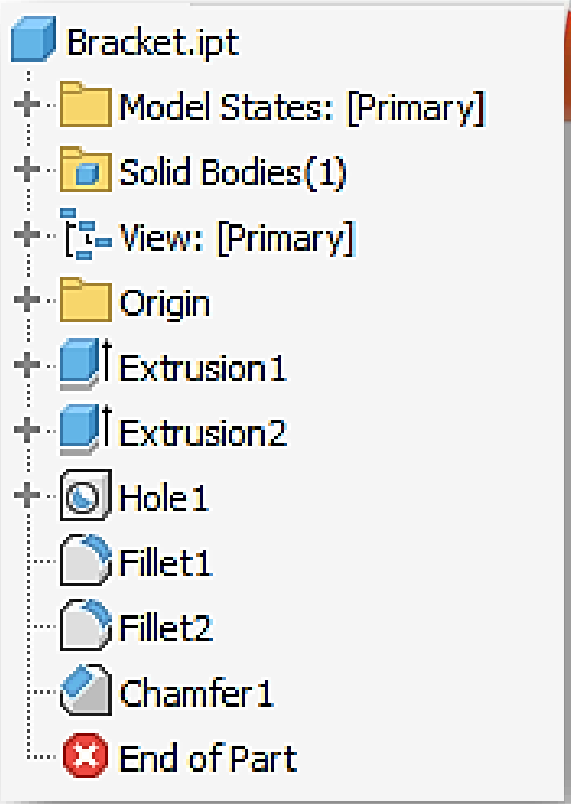
# API Object Model of a part



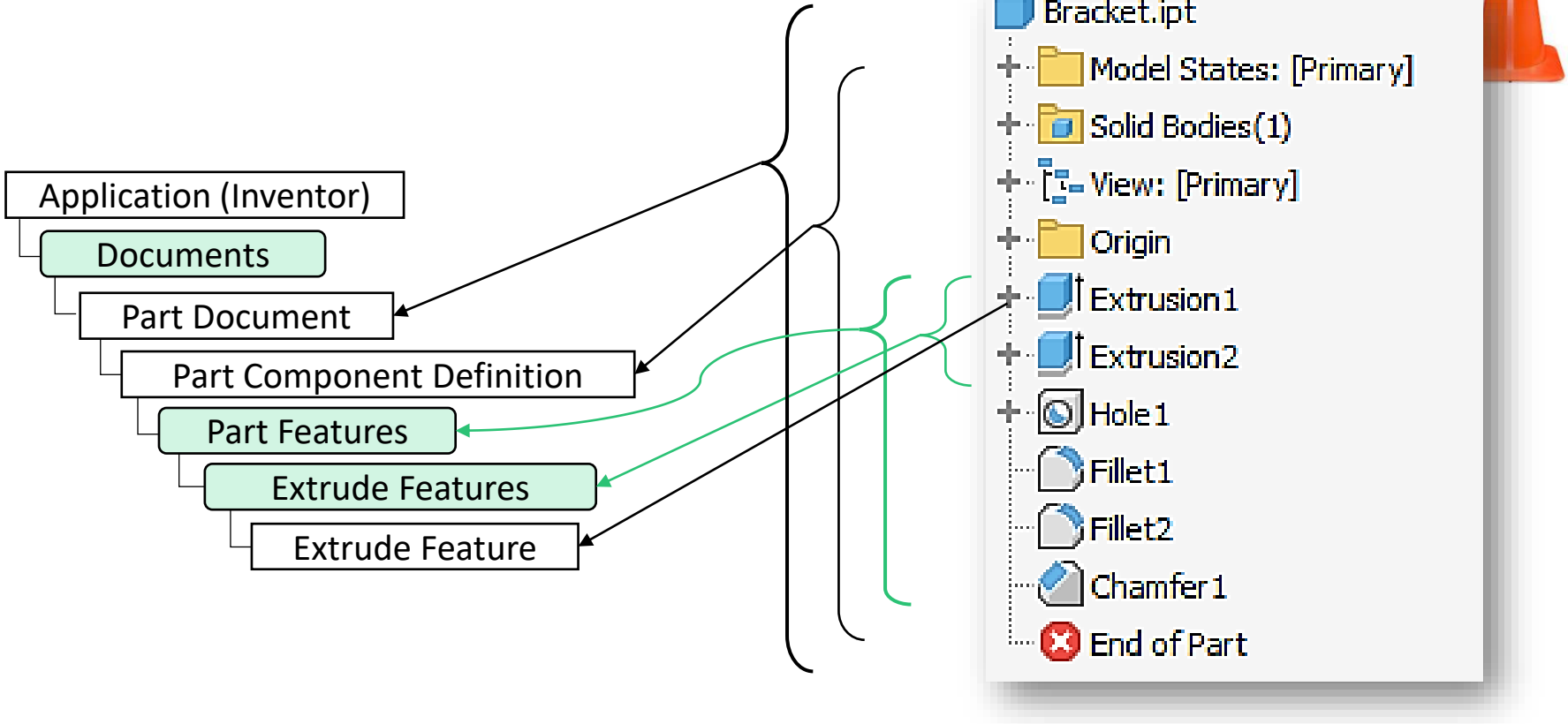



# API Object Model of a part

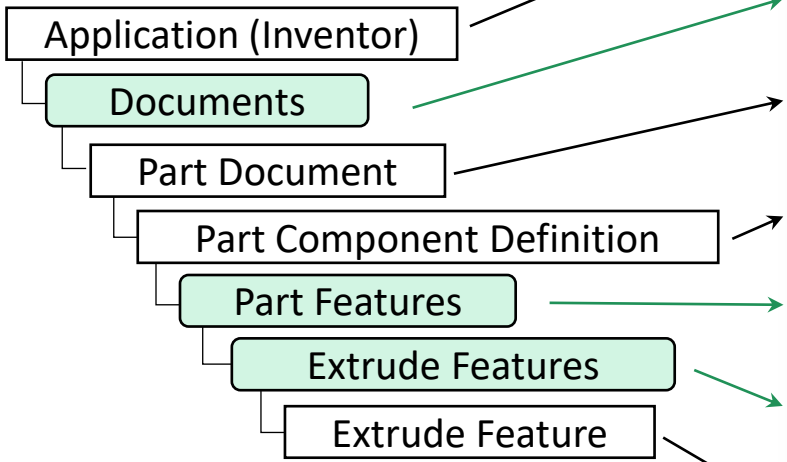


# API Object Model of a part



# Code Comparison




```
1   Dim InventorApp As Inventor.Application
2   InventorApp = ThisApplication
3
4   Dim Docs As Documents
5   Docs = InventorApp.Documents
6
7   Dim Doc As PartDocument
8   Doc = Docs.VisibleDocuments.Item(1)
9
10  Dim ComponentDef As ComponentDefinition
11  ComponentDef = Doc.ComponentDefinition
12
13  Dim Features As PartFeatures
14  Features = ComponentDef.Features
15
16  Dim Extrusions As ExtrudeFeatures
17  Extrusions = Features.ExtrudeFeatures
18
19  Dim Extrude1 As ExtrudeFeature
20  Extrude1 = Extrusions.Item("Extrusion1")
```

iLogic Example

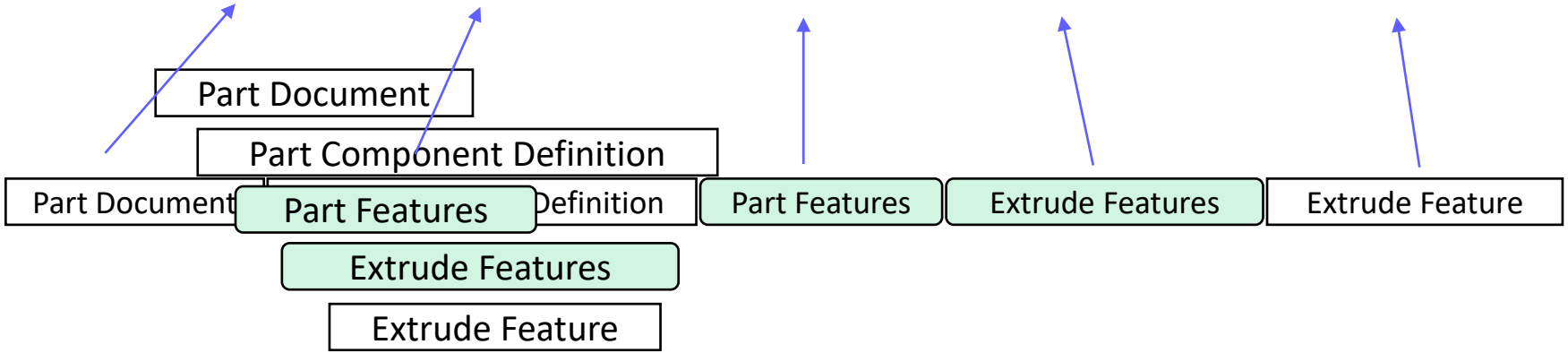
# Using a single line to get the named Extrusion

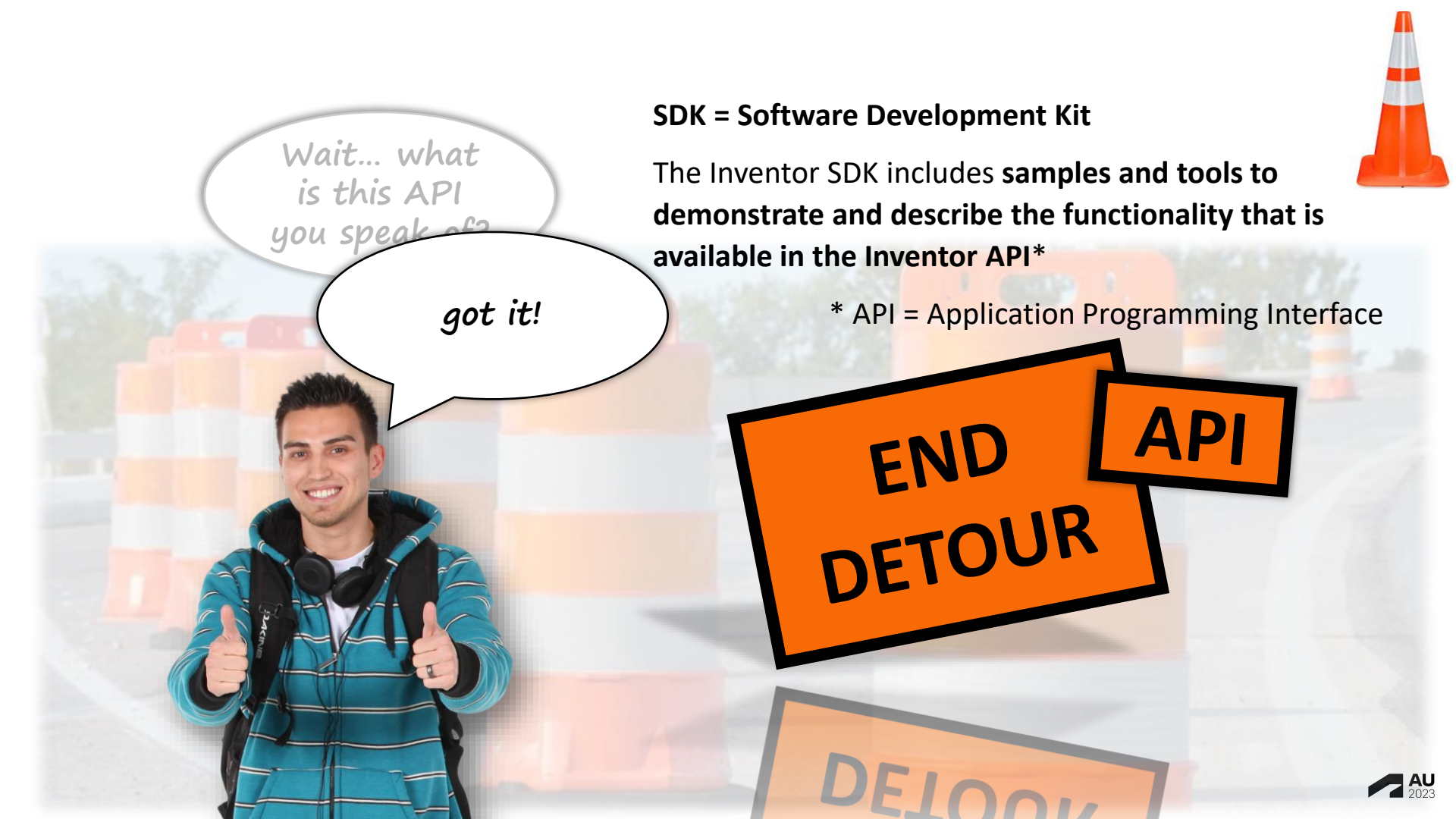
```
Extrude1 = Doc.ComponentDefinition.Features.ExtrudeFeatures.Item("Extrusion1")
```

- Part Document
  - Part Component Definition
    - Part Features
      - Extrude Features
        - Extrude Feature

# Using a single line to get the named Extrusion

```
Extrude1 = Doc.ComponentDefinition.Features.ExtrudeFeatures.Item("Extrusion1")
```






**SDK = Software Development Kit**

The Inventor SDK includes **samples and tools to demonstrate and describe the functionality that is available in the Inventor API***

* API = Application Programming Interface


**SDK = Software Development Kit**

The Inventor SDK includes **samples and tools to demonstrate and describe the functionality that is available in the Inventor API***

* API = Application Programming Interface


**SDK = Software Development Kit**

The Inventor SDK includes **samples and tools to demonstrate and describe the functionality that is available in the Inventor API***

* API = Application Programming Interface

The SDK installs with Inventor, but we need to “unpack” it


**SDK = Software Development Kit**

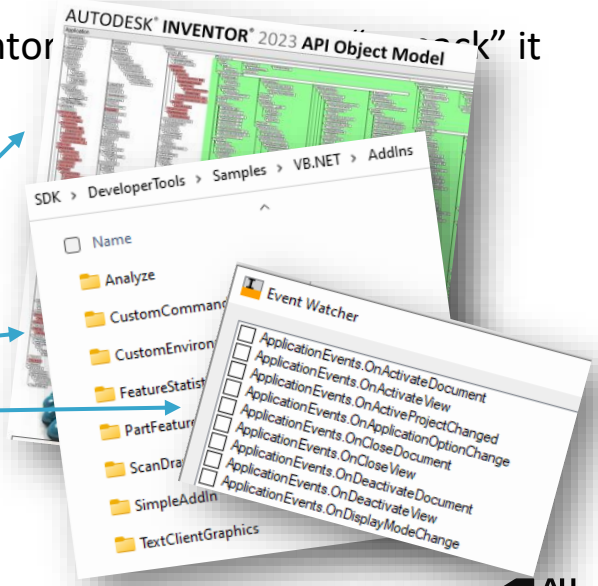
The Inventor SDK includes **samples and tools to demonstrate and describe the functionality that is available in the Inventor API***

* API = Application Programming Interface

The SDK installs with Inventor [illegible]ck" it

The SDK includes:

- Developer Tools
  - Documentation
  - Samples
  - Tools
  - Add-in Templates
- User Tools



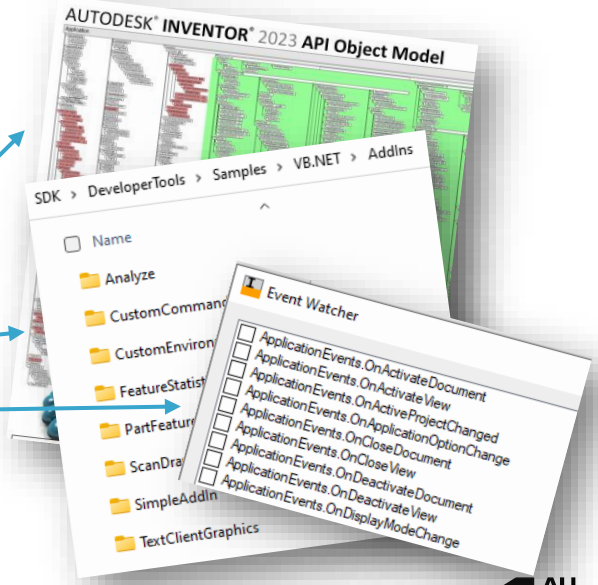

**SDK = Software Development Kit**

The Inventor SDK includes **samples and tools to demonstrate and describe the functionality that is available in the Inventor API***

* API = Application Programming Interface

The SDK includes:

- Developer Tools
  - Documentation
  - Samples
  - Tools
  - **Add-in Templates**
- User Tools

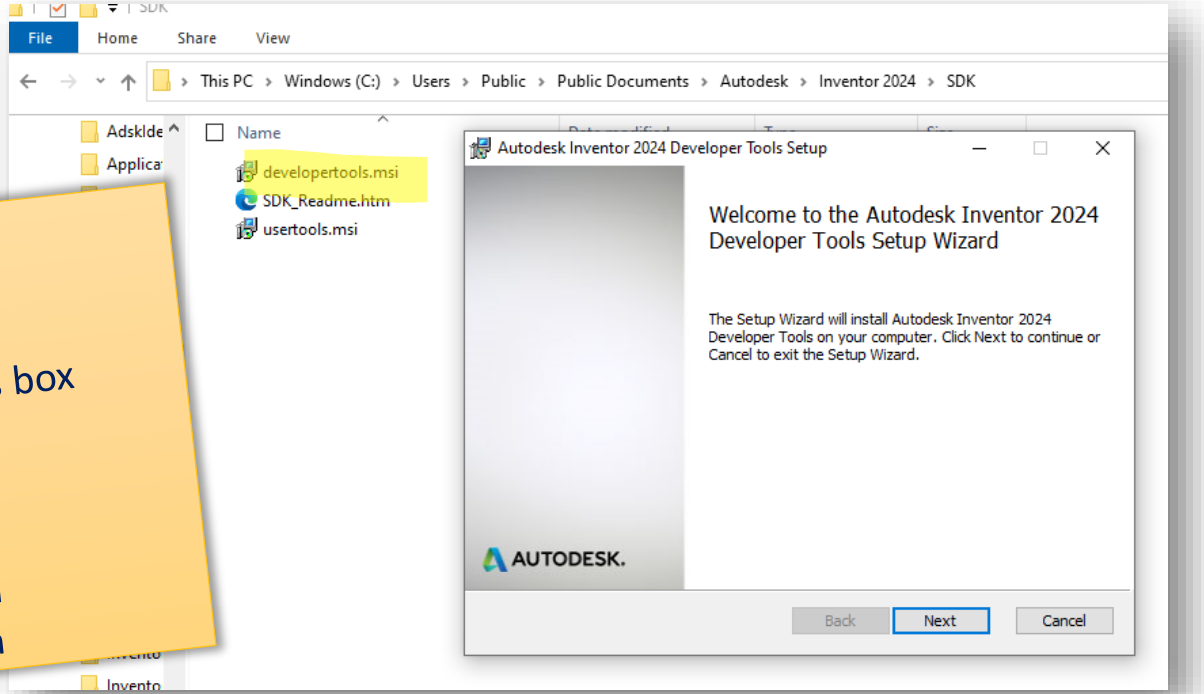


# Unpacking/installing the Inventor SDK files

**( Software Developer Kit )**

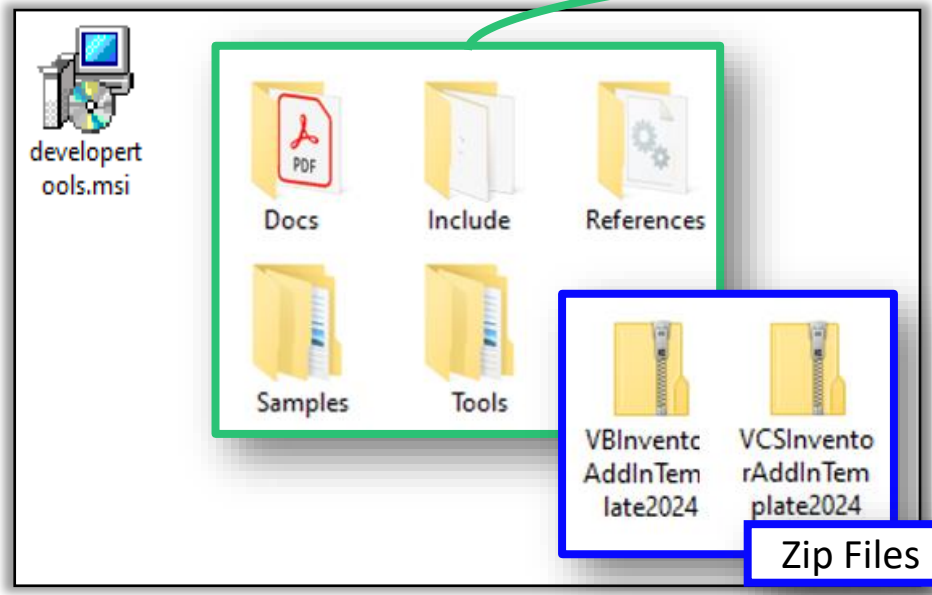
In Windows Explorer go to: C:\Users\Public\Documents\Autodesk\Inventor 2024\SDK

- Double-click the **developertools.msi** file
- Click the **Next** button
- Check the Accept Terms box
- Click the **Next** button
- Choose the path
- Click the **Next** button
- Click the **Install** button
- Click the **Finish** button



# developertools.msi Contents

**Contents of developertools.msi are unpacked into two Locations**




C:\Users\Public\Documents\Autodesk**Inventor 2024**\SDK\

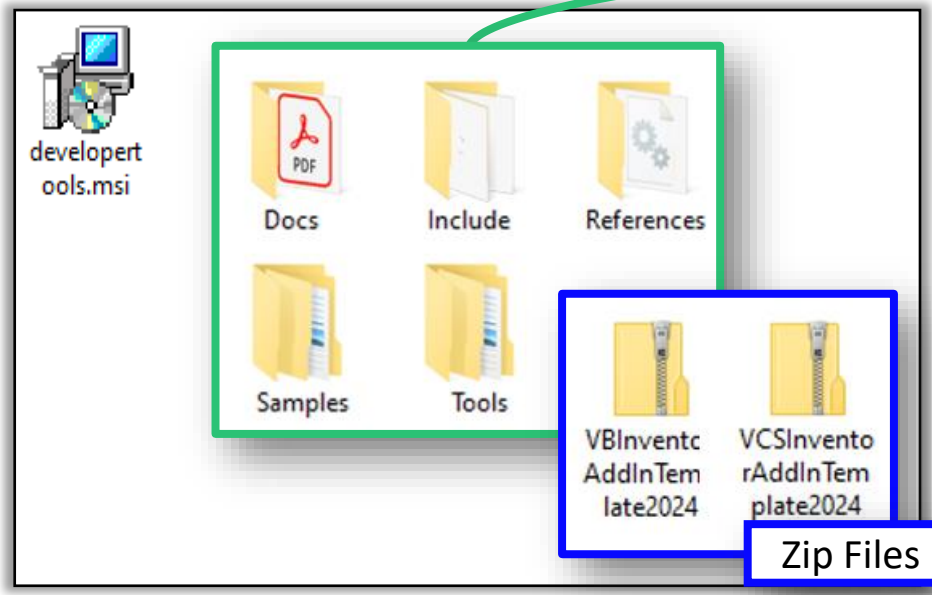
DeveloperTools

C:\Users\CurtisWaguespack\Documents**Visual Studio 2022**\Templates\

ProjectTemplates

# developertools.msi Contents

**Contents of developertools.msi are unpacked into two Locations**




C:\Users\Public\Documents\Autodesk**Inventor 2024**\SDK\

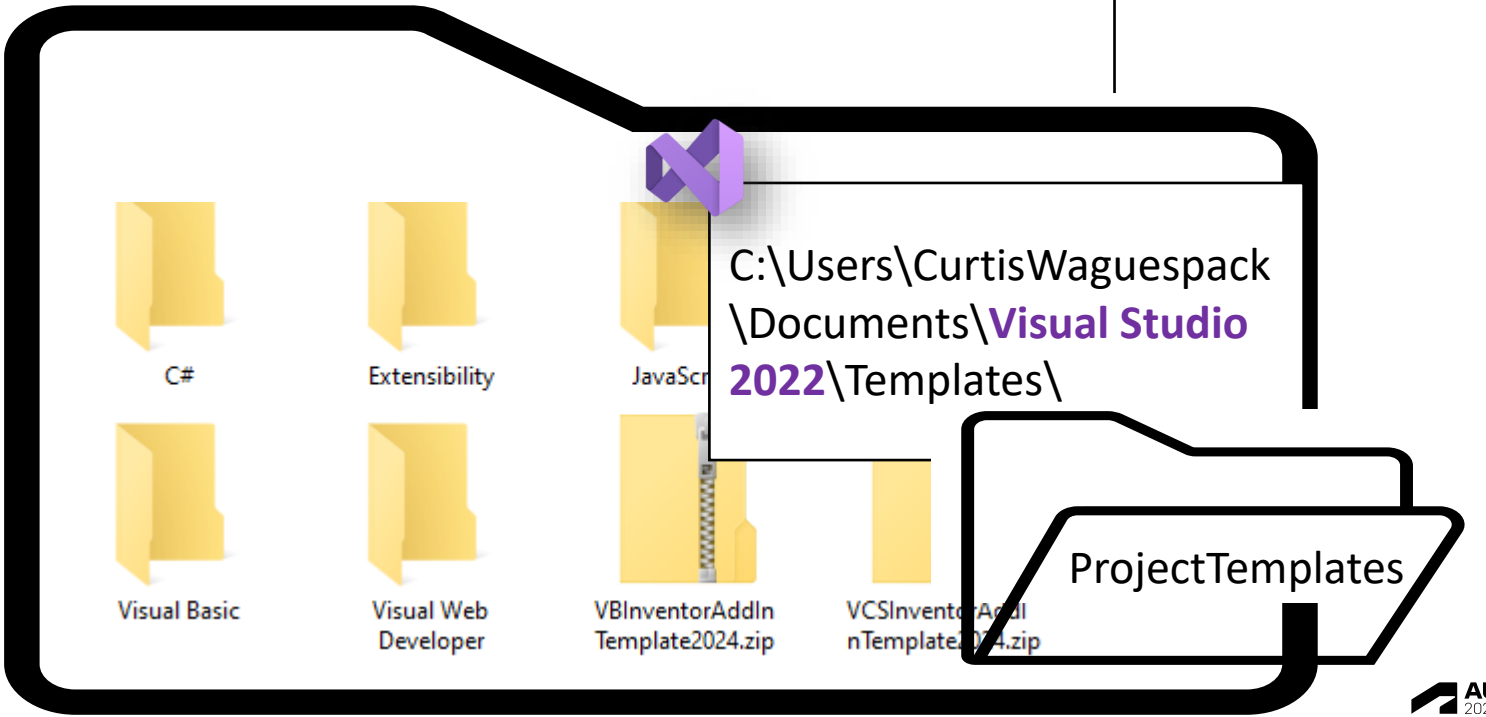
DeveloperTools

C:\Users\CurtisWaguespack\Documents**Visual Studio 2022**\Templates\

ProjectTemplates

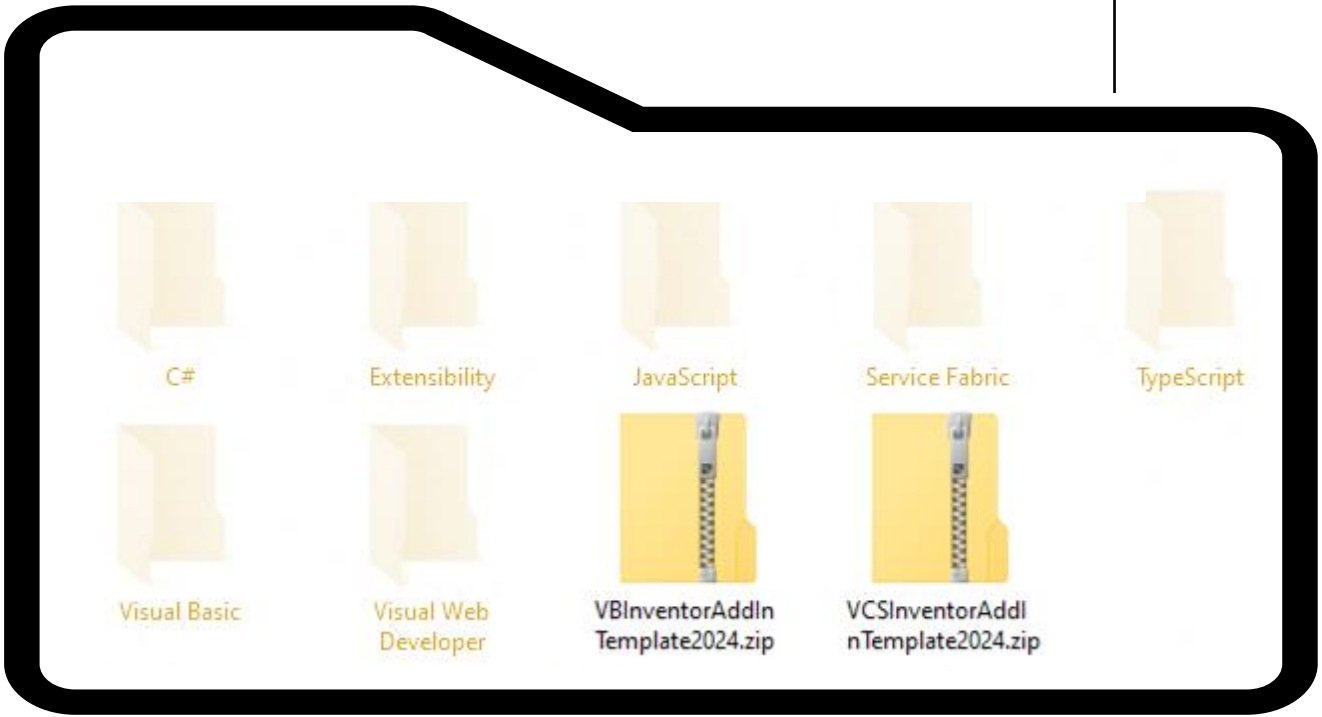
# Add-in Templates

C:\Users\CurtisWaguespack\Documents**Visual Studio 2022**\Templates\ProjectTemplates



# Add-in Templates

C:\Users\CurtisWaguespack\Documents**Visual Studio 2022**\Templates\ProjectTemplates



# Steps to get started…

… with Visual Studio and the Inventor SDK

- [x] Install Visual Studio
- [x] Extract the Inventor SDK ( software development kit)
- [ ] Ensure the Inventor Add-In Templates show up in Visual Studio

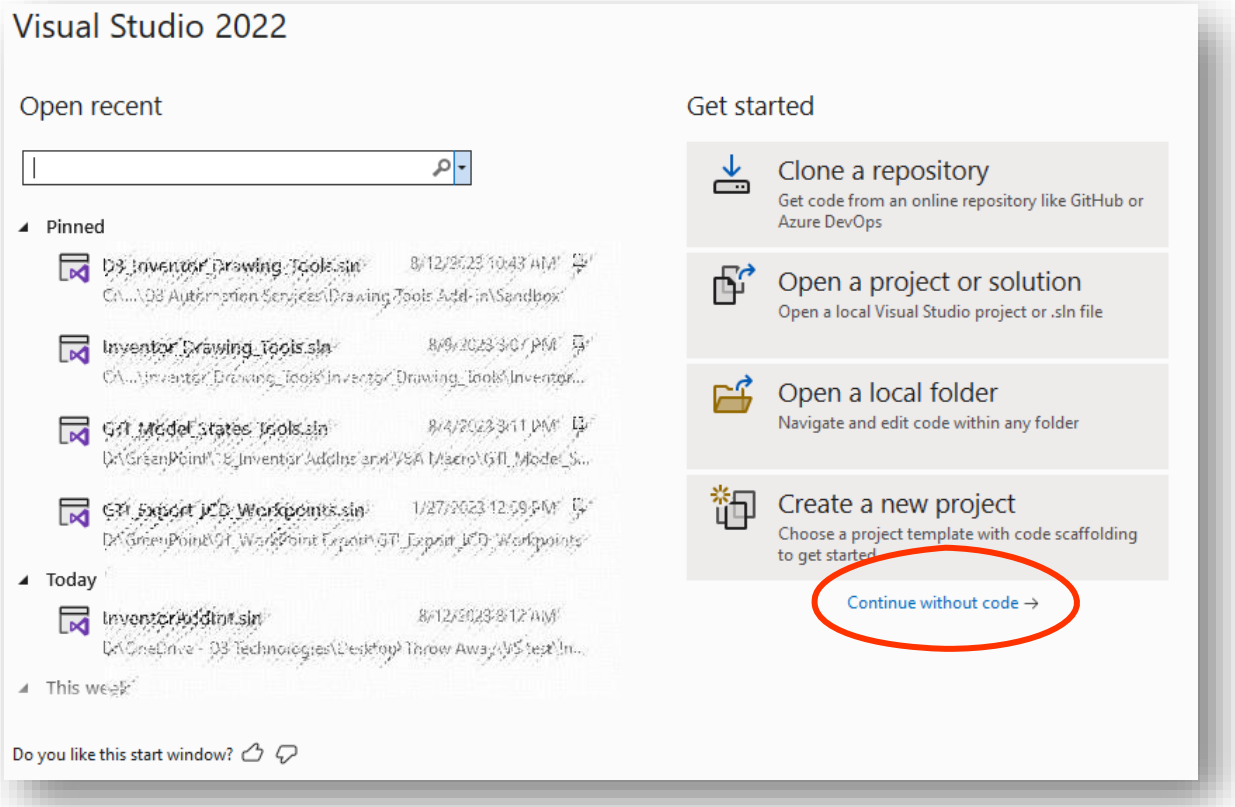
# Steps to get started…

… with Visual Studio and the Inventor SDK

- [x] Install Visual Studio
- [x] Extract the Inventor SDK ( software development kit)
- [ ] Ensure the Inventor Add-In Templates show up in Visual Studio

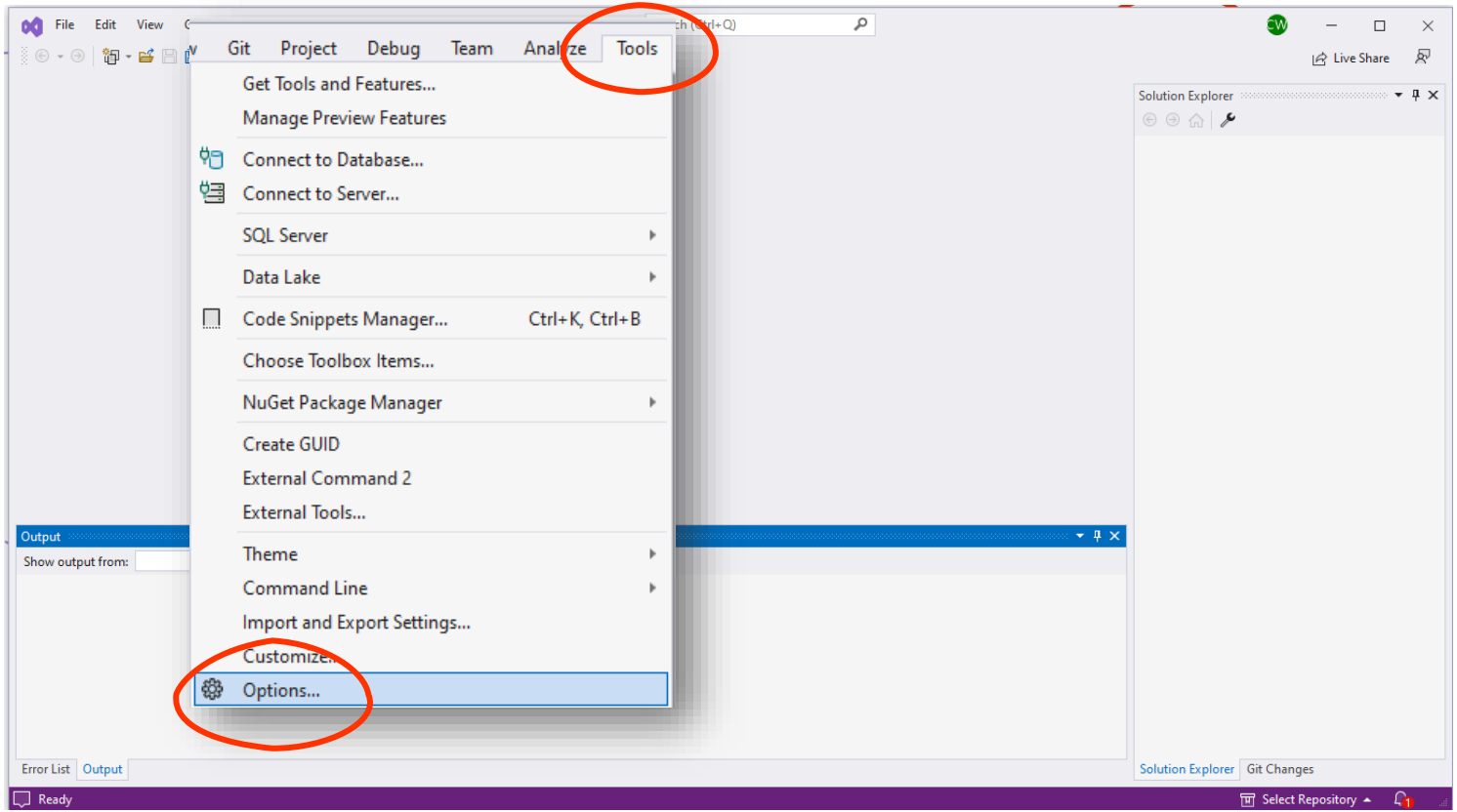
# Check/set the template location in Visual Studio

**Open Visual Studio > click Continue without code**



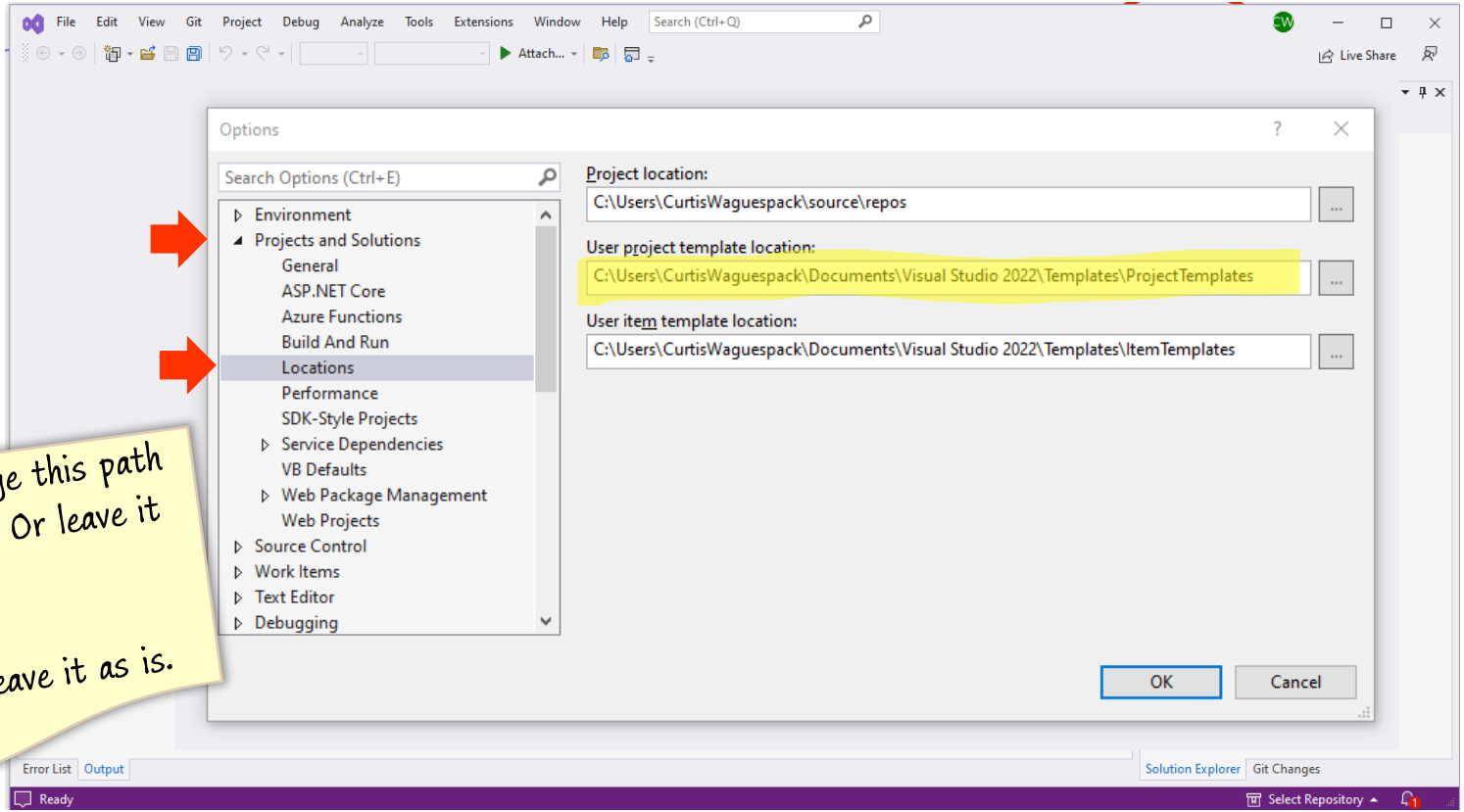
# Check/set the template location in Visual Studio

## Visual Studio > Tools tab > Options



# Check/set the template location in Visual Studio

## Visual Studio > Tools tab > Options > Projects and Solutions > Locations

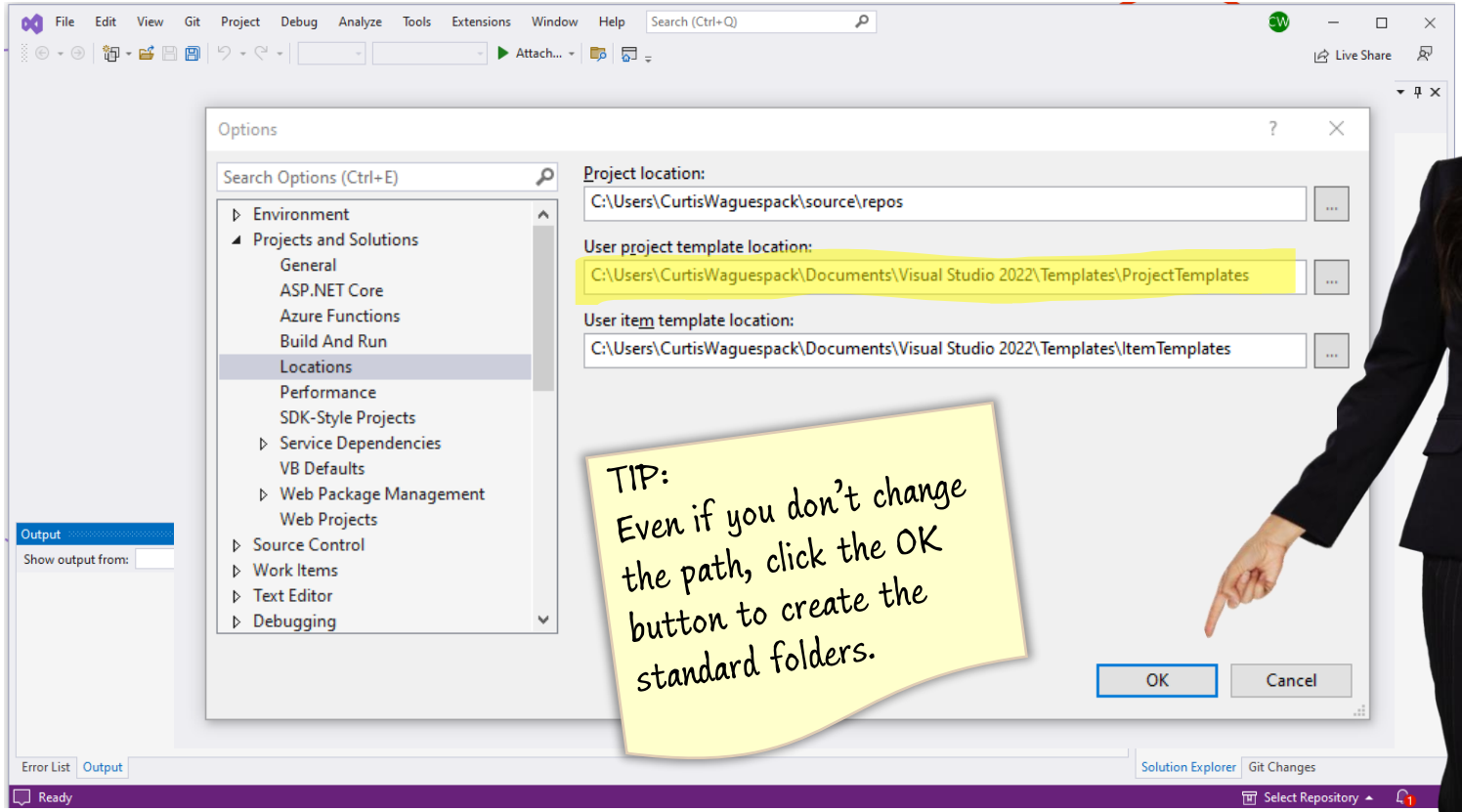



You can change this path if you prefer. Or leave it as is.

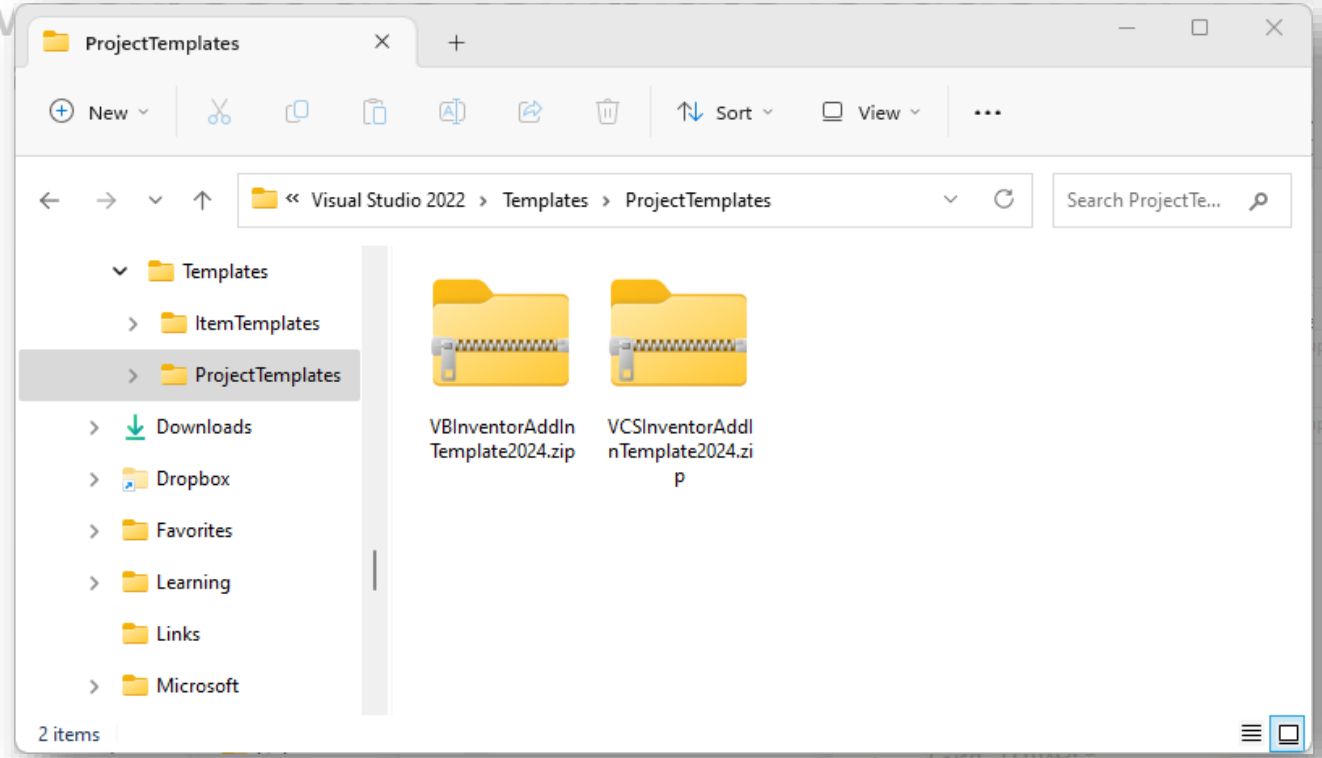
I generally leave it as is.

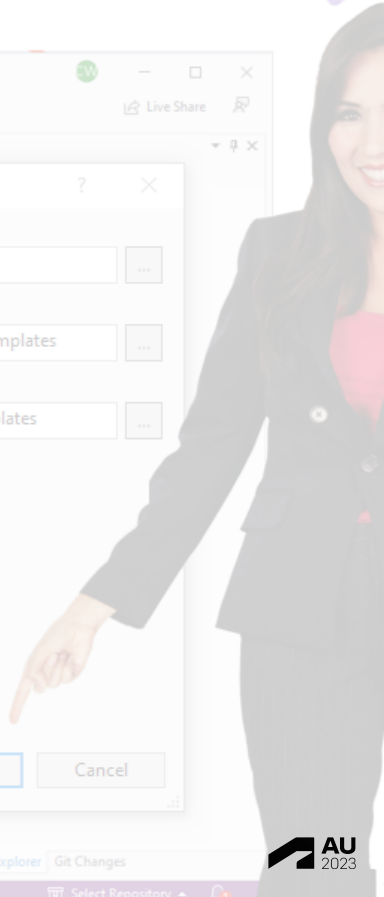
# Check/set the template location in Visual Studio

## Visual Studio > Tools tab > Options > Projects and Solutions > Locations

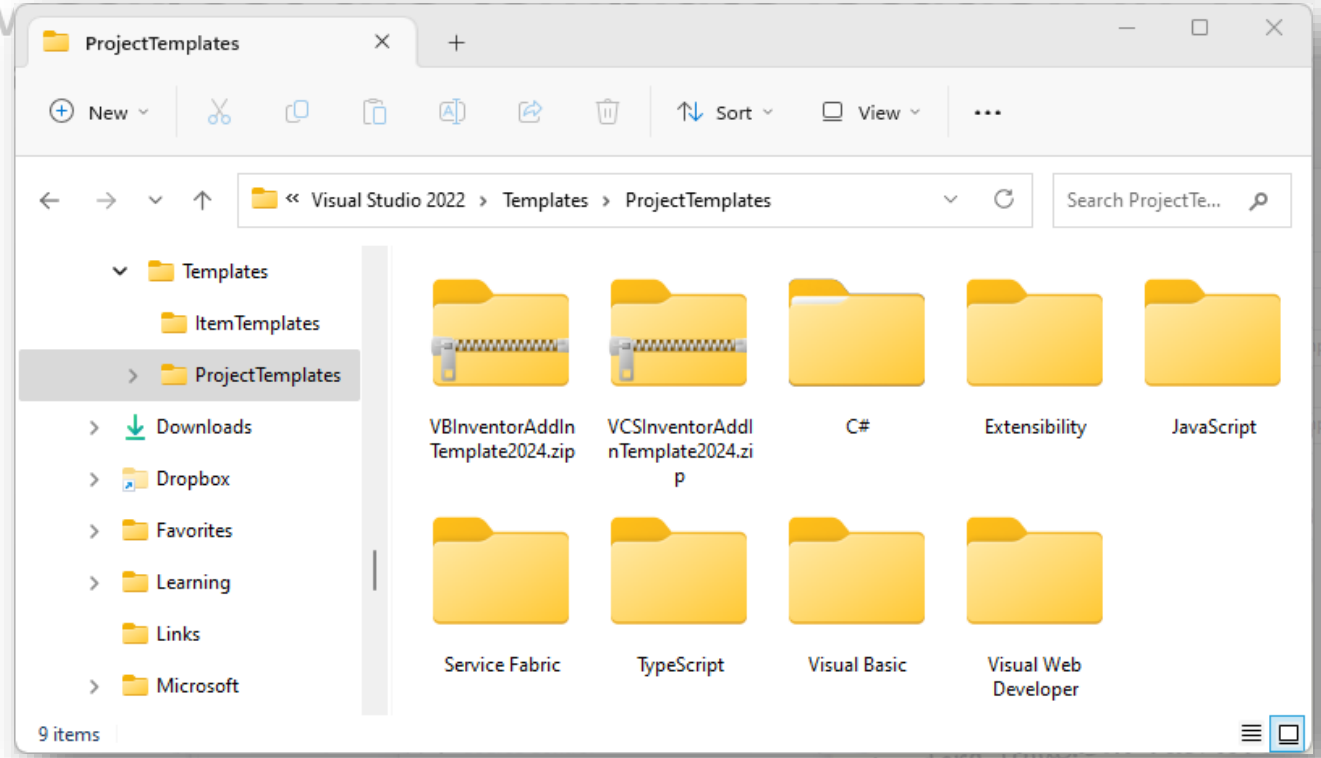


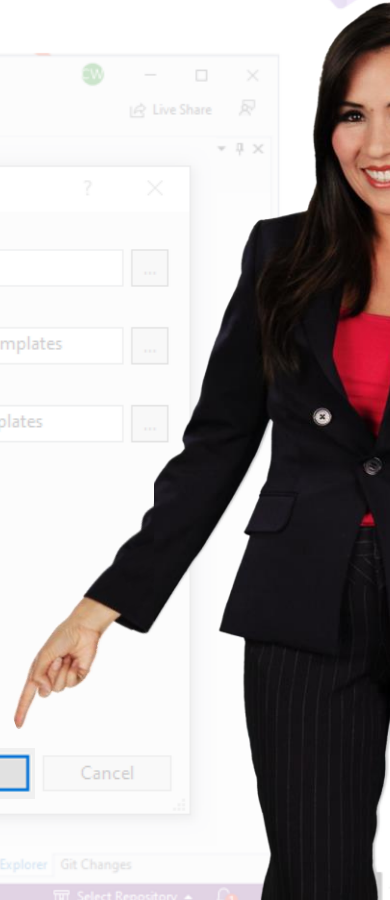
# Check/set the template location in Visual Studio



# Check/set the template location in Visual Studio



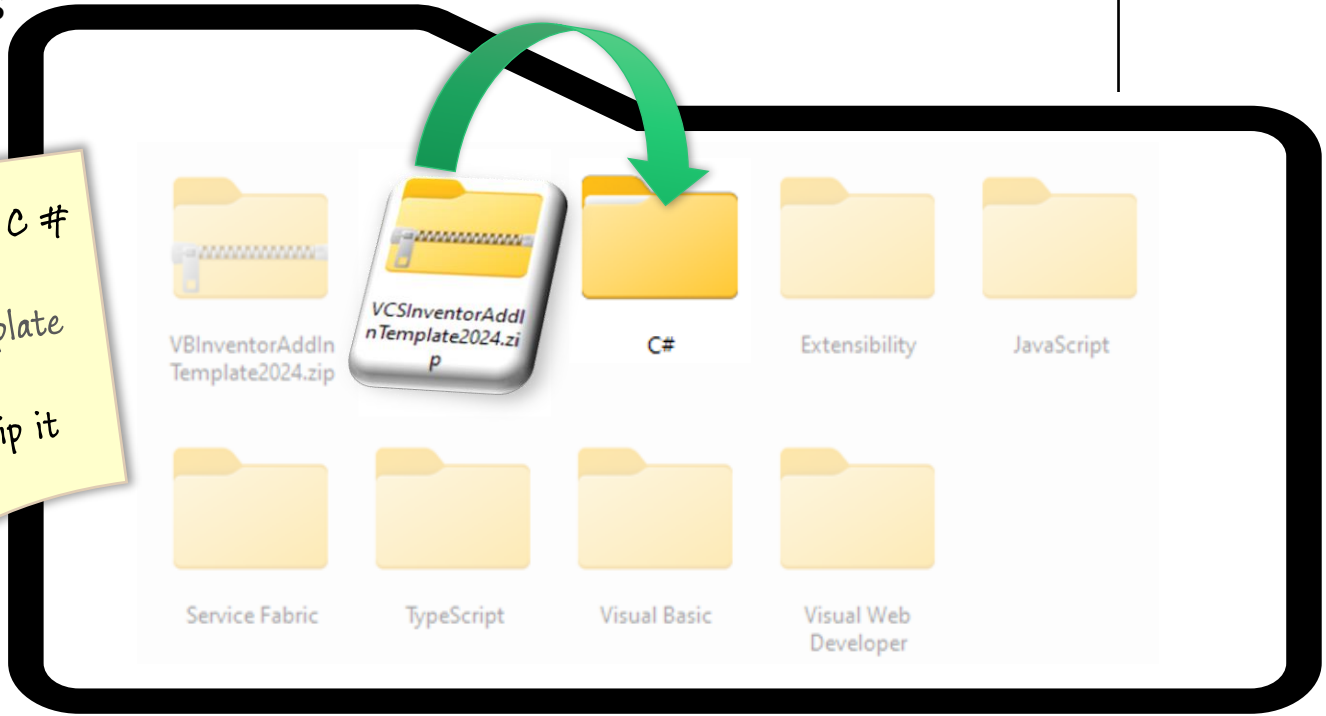



# Add-in Templates

C:\Users\CurtisWaguespack\Documents**Visual Studio 2022**\Templates\ProjectTemplates

Move the zip file to the C # folder.
VCSInventorAddinTemplate

You don't need to unzip it

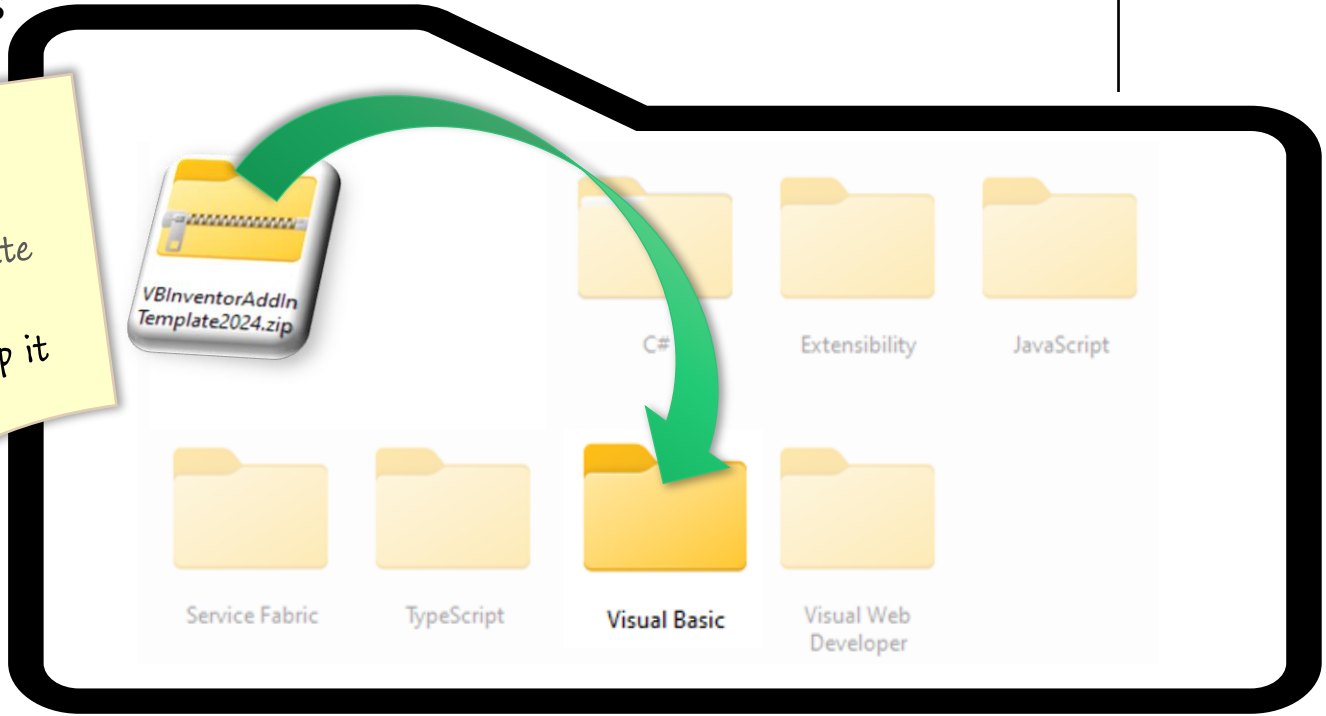


# Add-in Templates

C:\Users\CurtisWaguespack\Documents**Visual Studio 2022**\Templates\ProjectTemplates

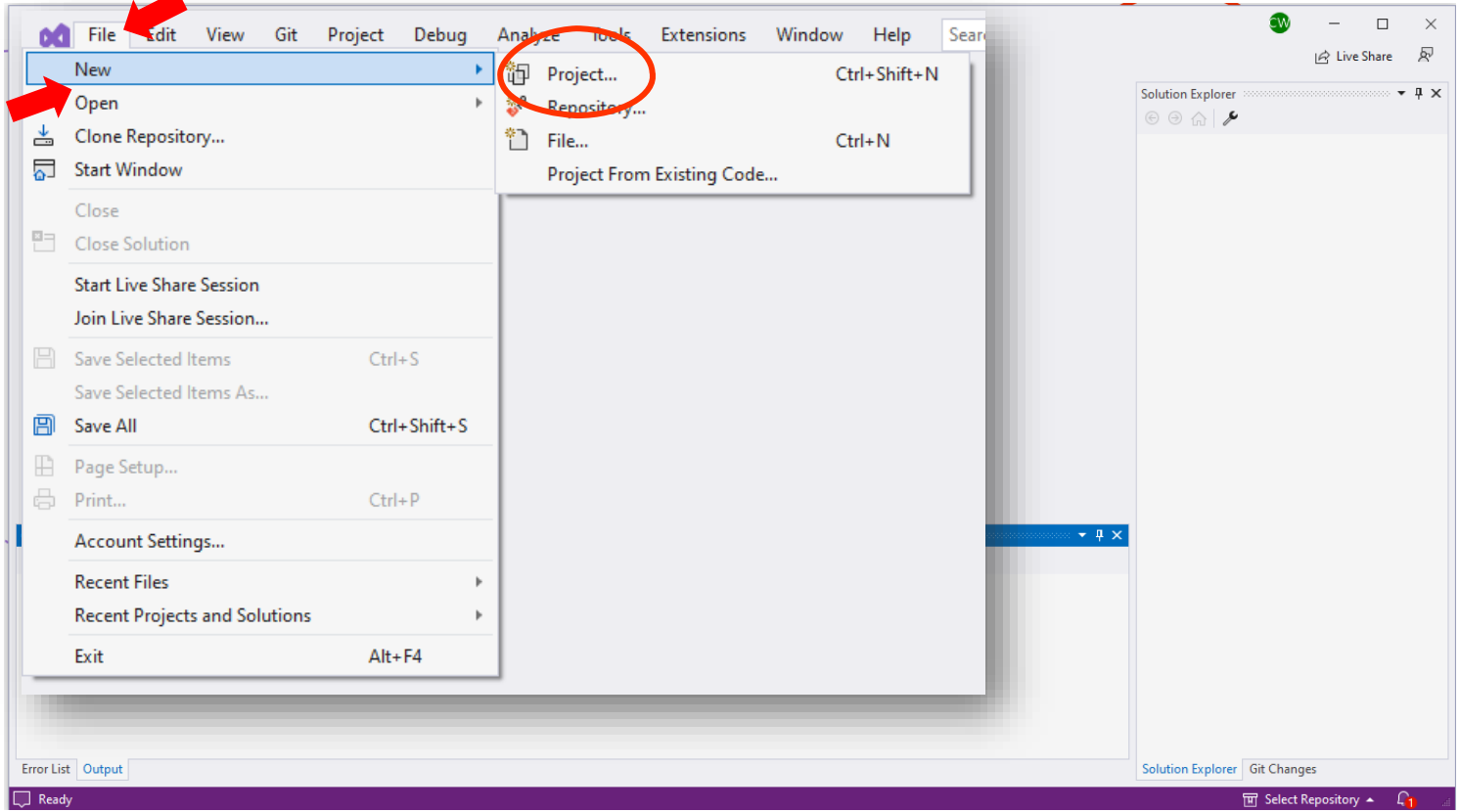
Move the zip file to the Visual Basic folder.
**VB**InventorAddinTemplate

You don't need to unzip it



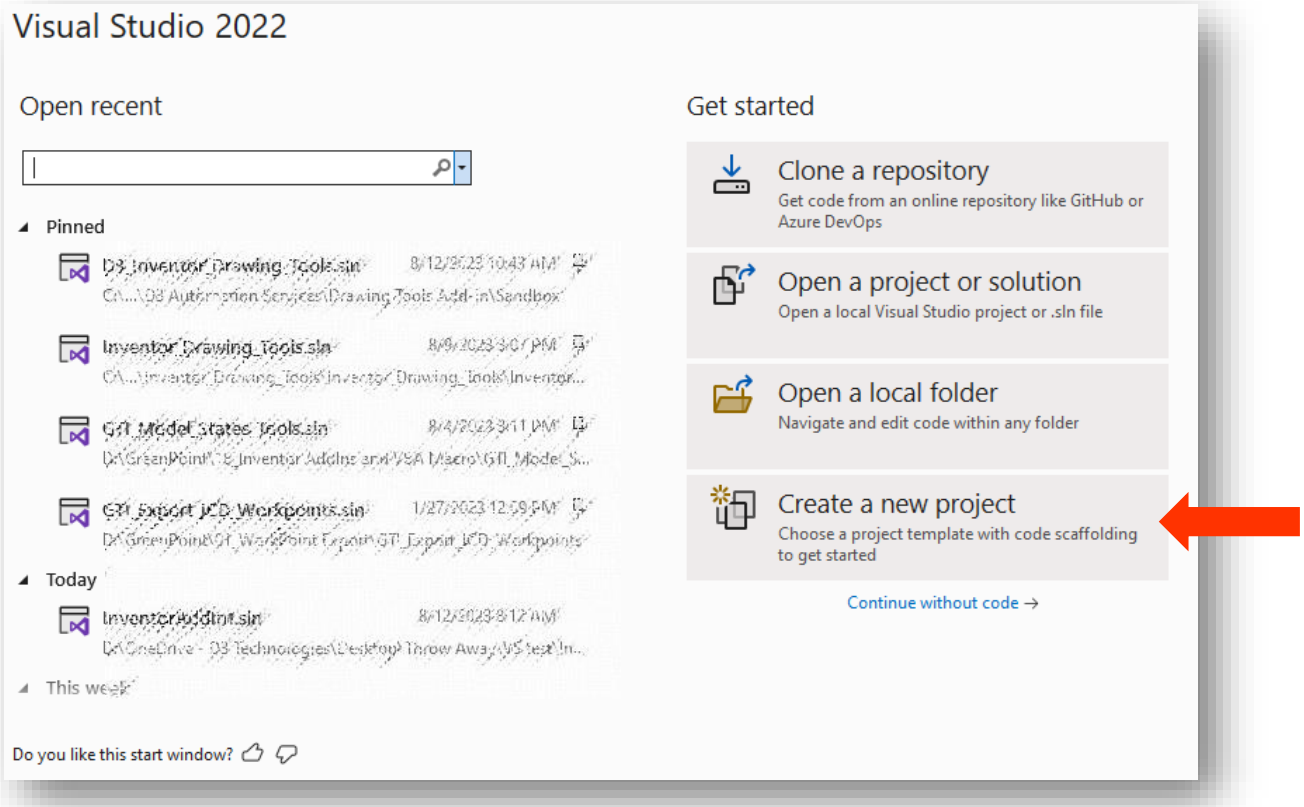
# Check/set the template location in Visual Studio

**Visual Studio > File tab > New > Project**



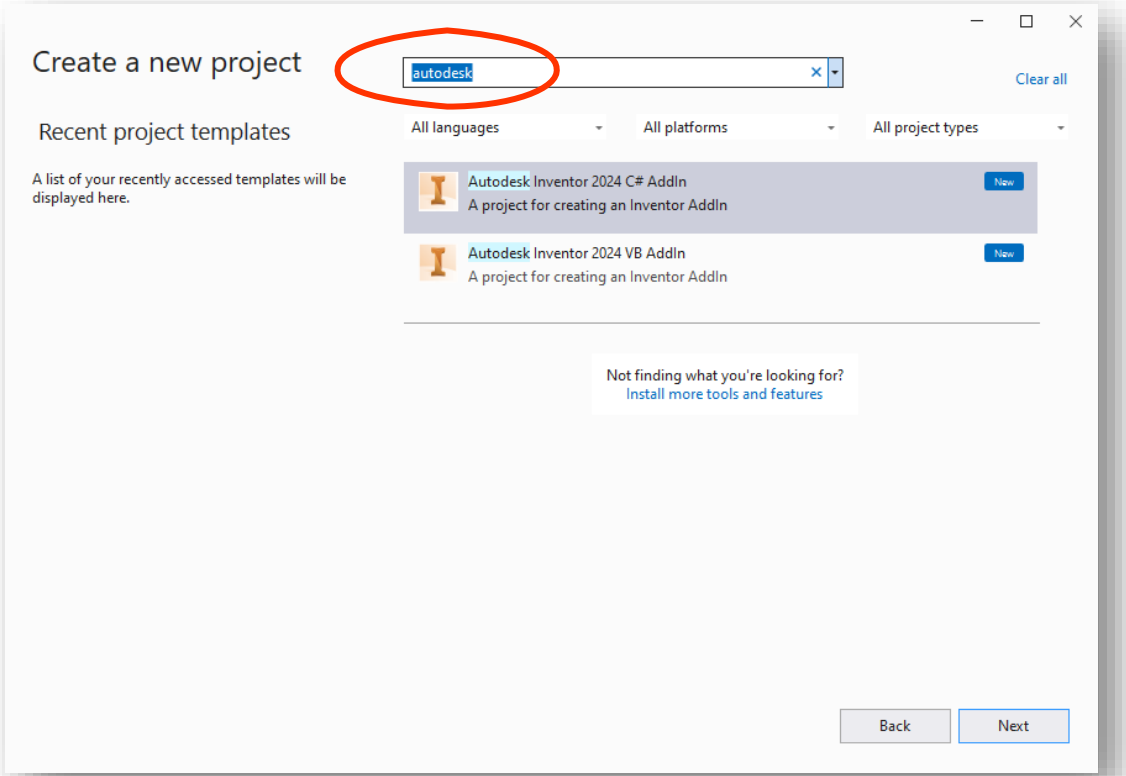
# Ensure Add-In Templates show up in Visual Studio

## Click Create a New Project



# Ensure Add-In Templates show up in Visual Studio

- Type **Autodesk** or **Inventor** in the search bar
- You should see the templates show up, as shown in this image.



# Now you're ready to create an add-in!

- [x] Install Visual Studio
- [x] Extract the Inventor SDK ( software development kit)
- [x] Ensure the Inventor Add-In Templates show up in Visual Studio

# Now you're ready to create an add-in!

Except you're not!

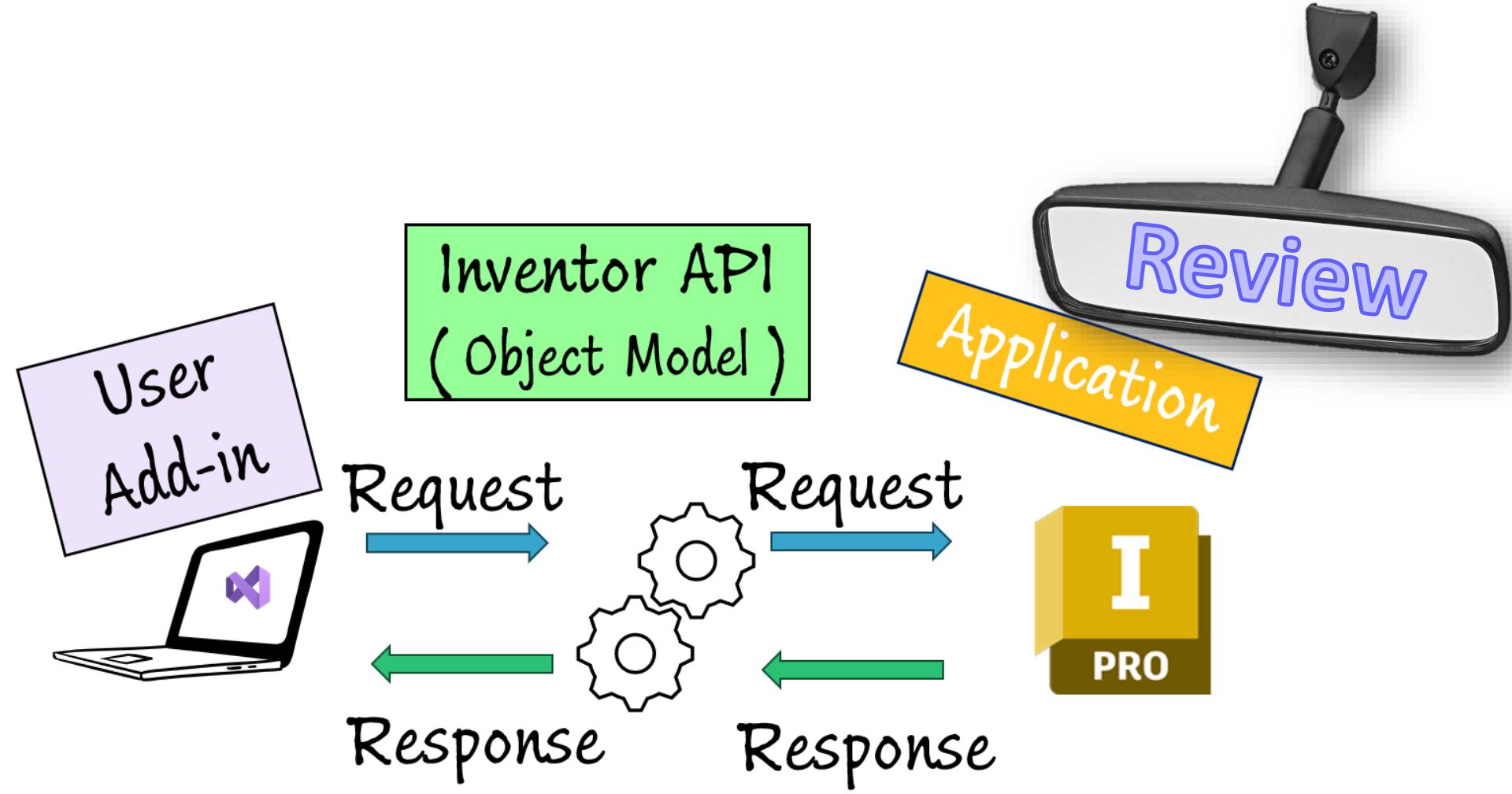
- Install Visual Studio
- Extract the Inventor SDK ( software development kit)
- Ensure the Inventor Add-In Templates show up in Visual Studio

# Quick Review

# Steps to get started…

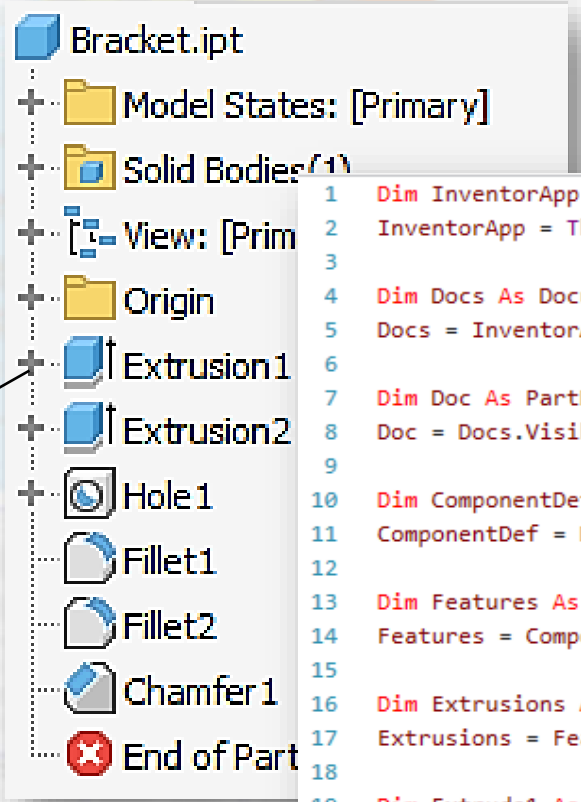
… with Visual Studio and the Inventor Software Development Kit


| 1 | 2 | 3 |
|---|---|---|
| Install Visual Studio | Install the Inventor SDK | Ensure the Inventor Add-in Templates are Loaded |

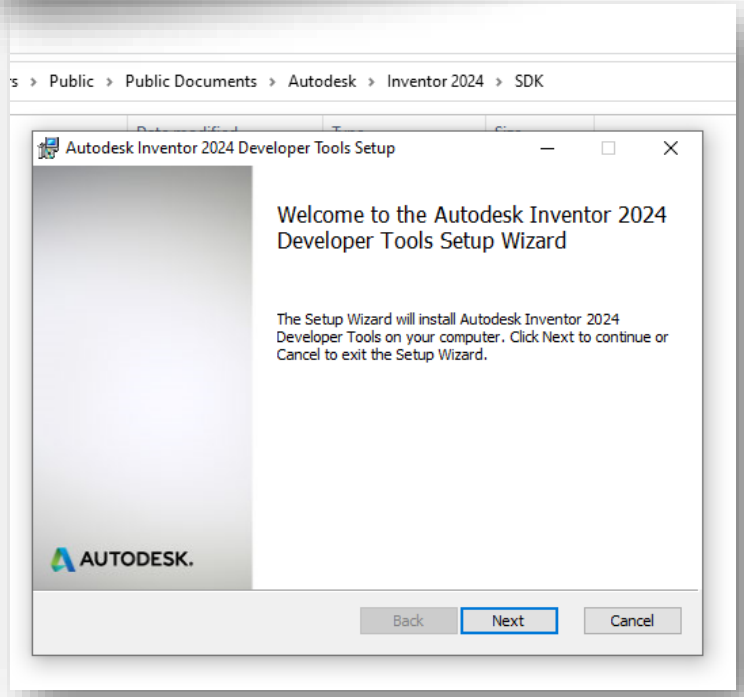

Review
Application
Inventor API
( Object Model )
User
Add-in
Request
Request
Response
Response
PRO

# Review

- Application (Inventor)
  - Documents
    - Part Document
      - Part Component Definition
        - Part Features
          - Extrude Features
            - Extrude Feature




```
1   Dim InventorApp As Inv
2   InventorApp = ThisAppl
3
4   Dim Docs As Documents
5   Docs = InventorApp.Doc
6
7   Dim Doc As PartDocumer
8   Doc = Docs.VisibleDocu
9
10  Dim ComponentDef As Co
11  ComponentDef = Doc.Com
12
13  Dim Features As PartFe
14  Features = ComponentDe
15
16  Dim Extrusions As Extr
17  Extrusions = Features.
18
19  Dim Extrude1 As Extruc
20  Extrude1 = Extrusions.
```

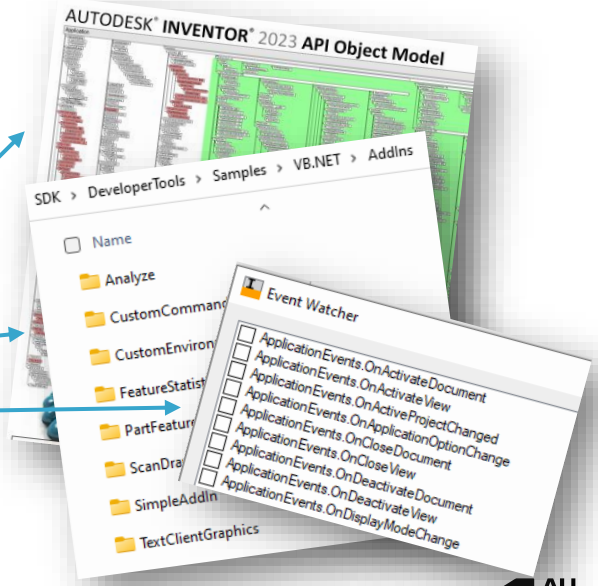

**SDK = Software Development Kit**

The Inventor SDK includes **samples and tools to demonstrate and describe the functionality that is available in the Inventor API***

* API = Application Programming Interface

The SDK includes:

- Developer Tools
  - Documentation
  - Samples
  - Tools
  - Add-in Templates
- User Tools



# An Improved Add-in Template

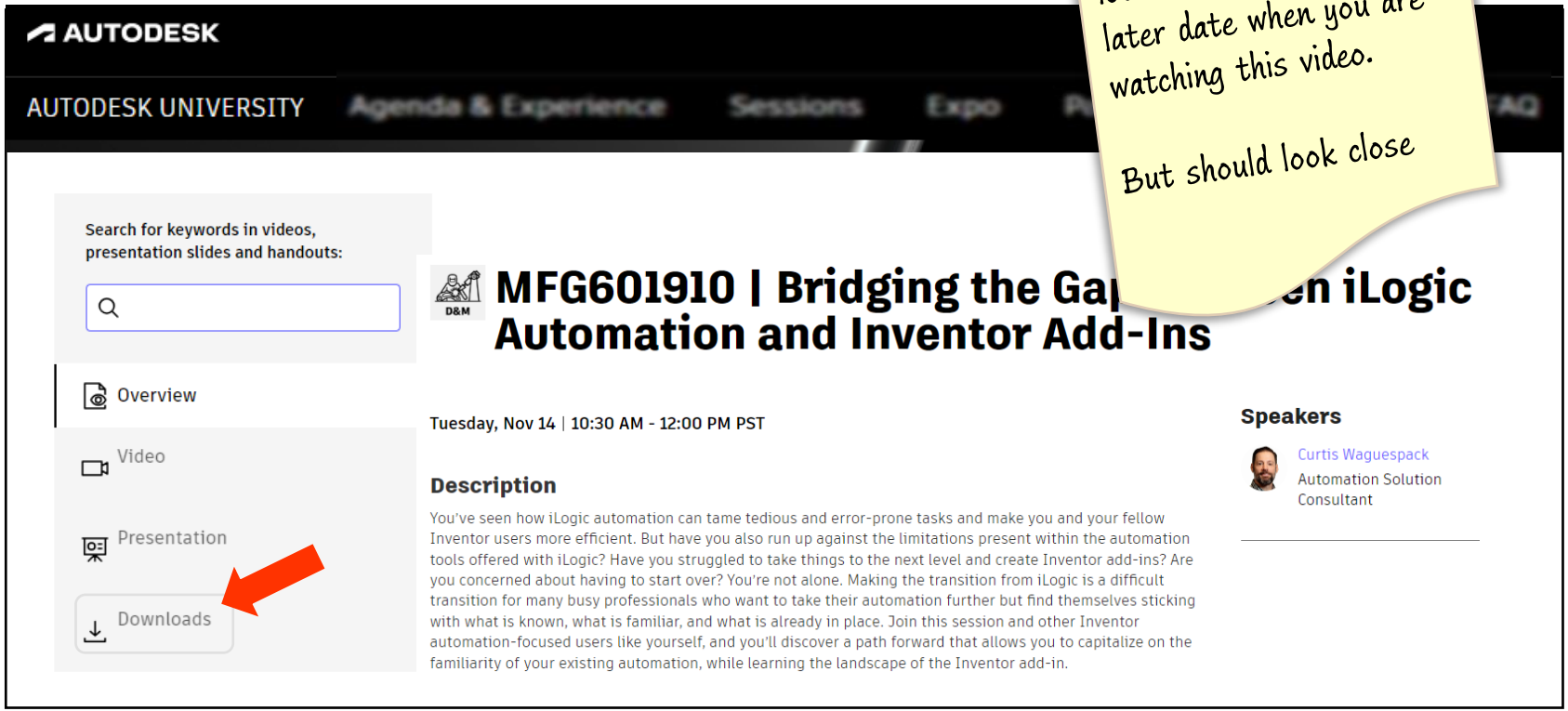
Downloading the files

# Getting the custom templates

## Download the Class Materials

The Autodesk University page might or might not look like this at some later date when you are watching this video.

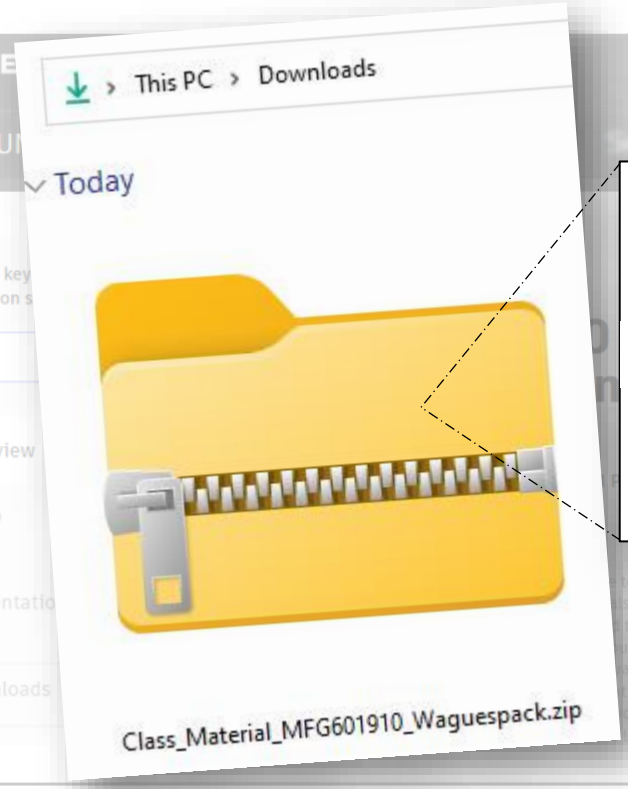
But should look close

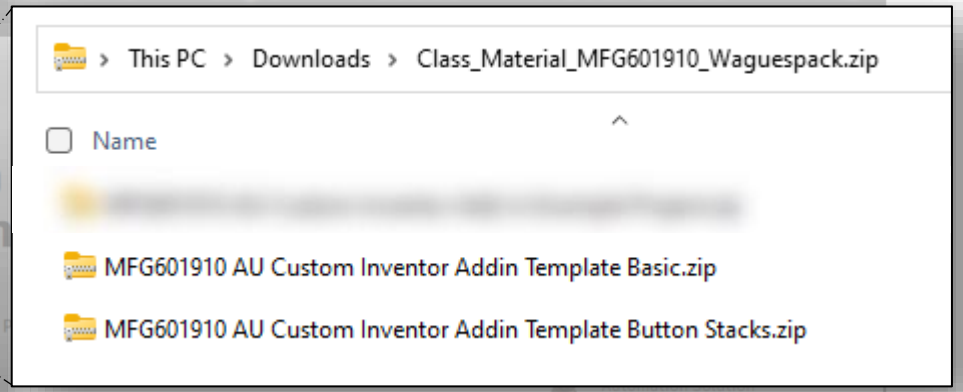


# Finding the add-in files

**Find the Templates within the class materials**

Exact file count and names might vary by the time this recording is released.



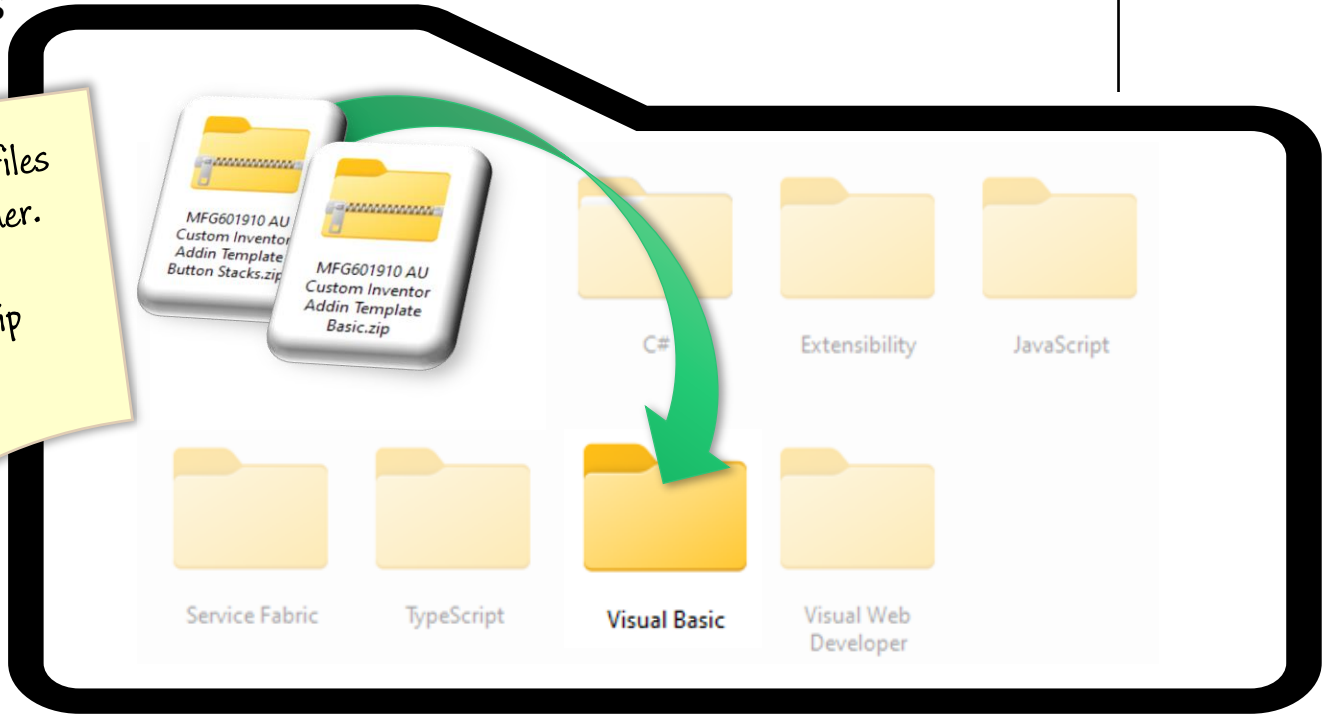



# Add-in Templates

C:\Users\CurtisWaguespack\Documents**Visual Studio 2022**\Templates\ProjectTemplates

Move the template zip files to the Visual Basic folder.

You don't need to unzip them.

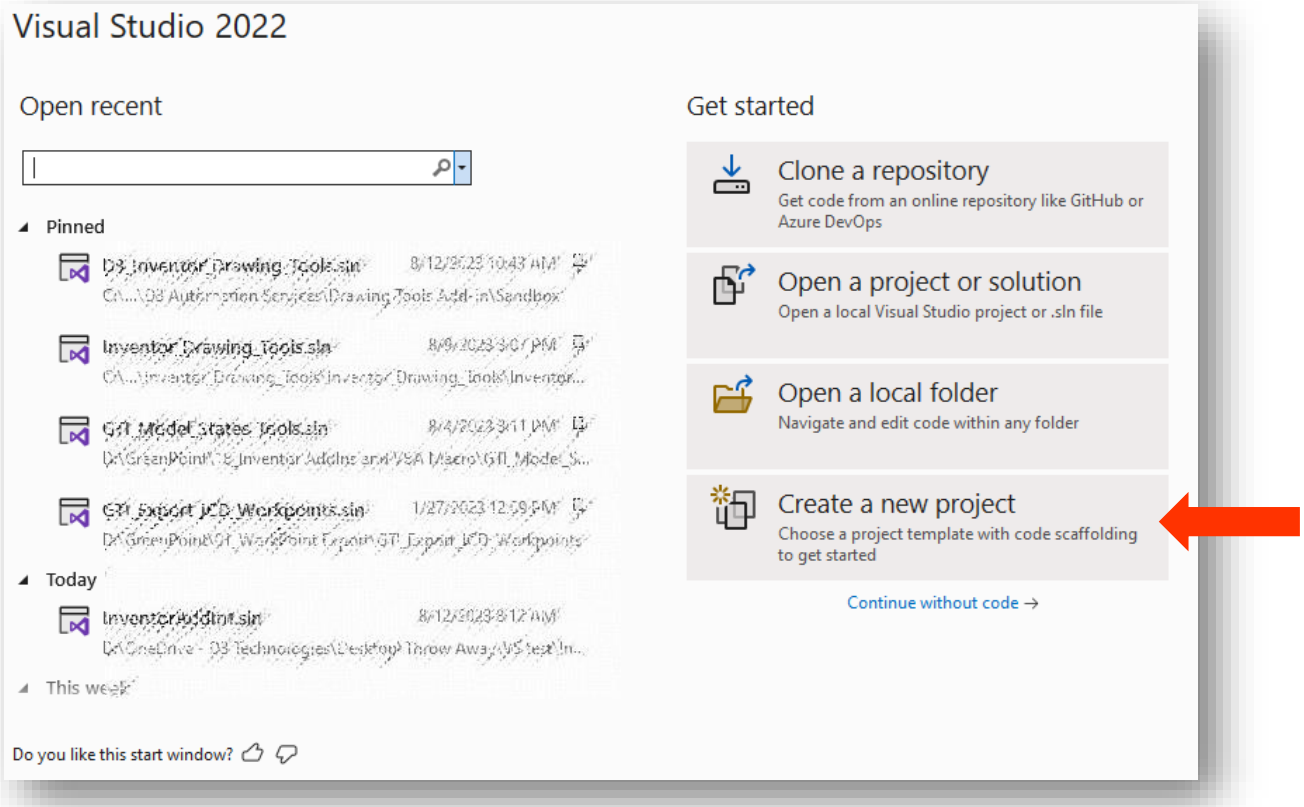


# Creating an Add-in from an improved template

Getting Started

# Creating an Add-in

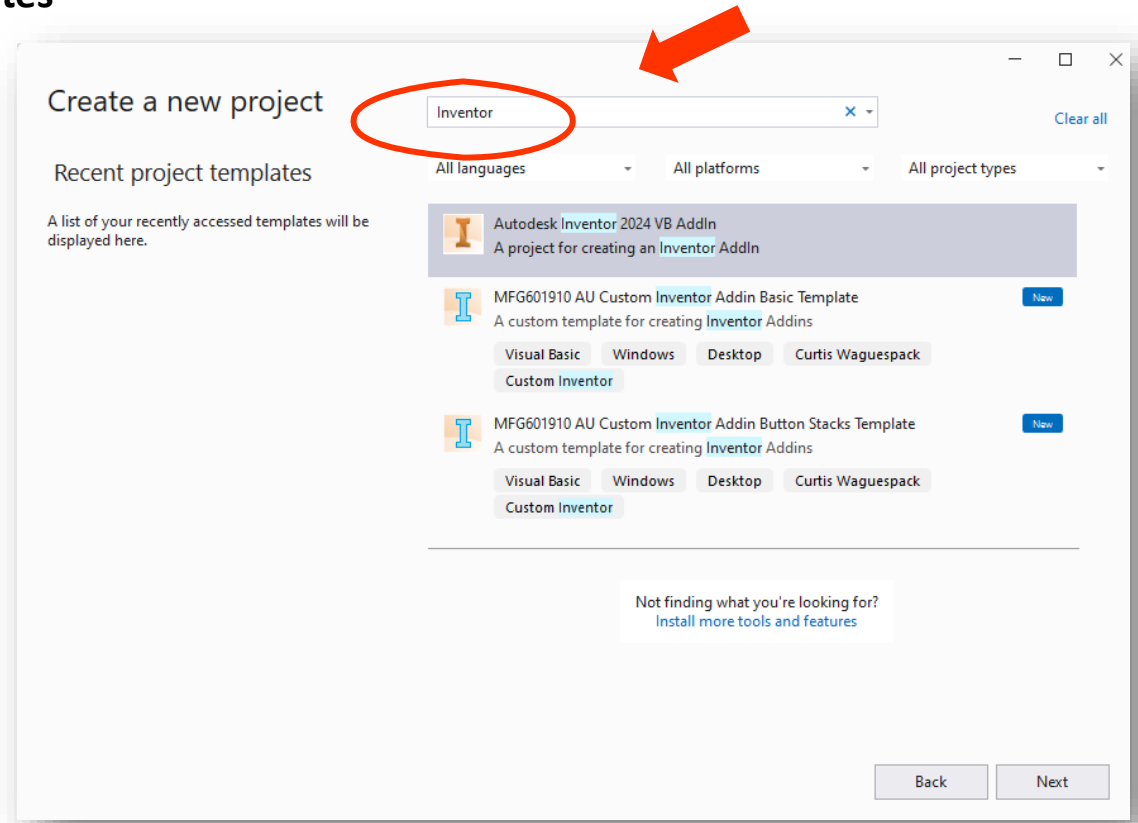
## Click Create a New Project



# Creating an Add-in

## Find the Inventor Add-in Templates

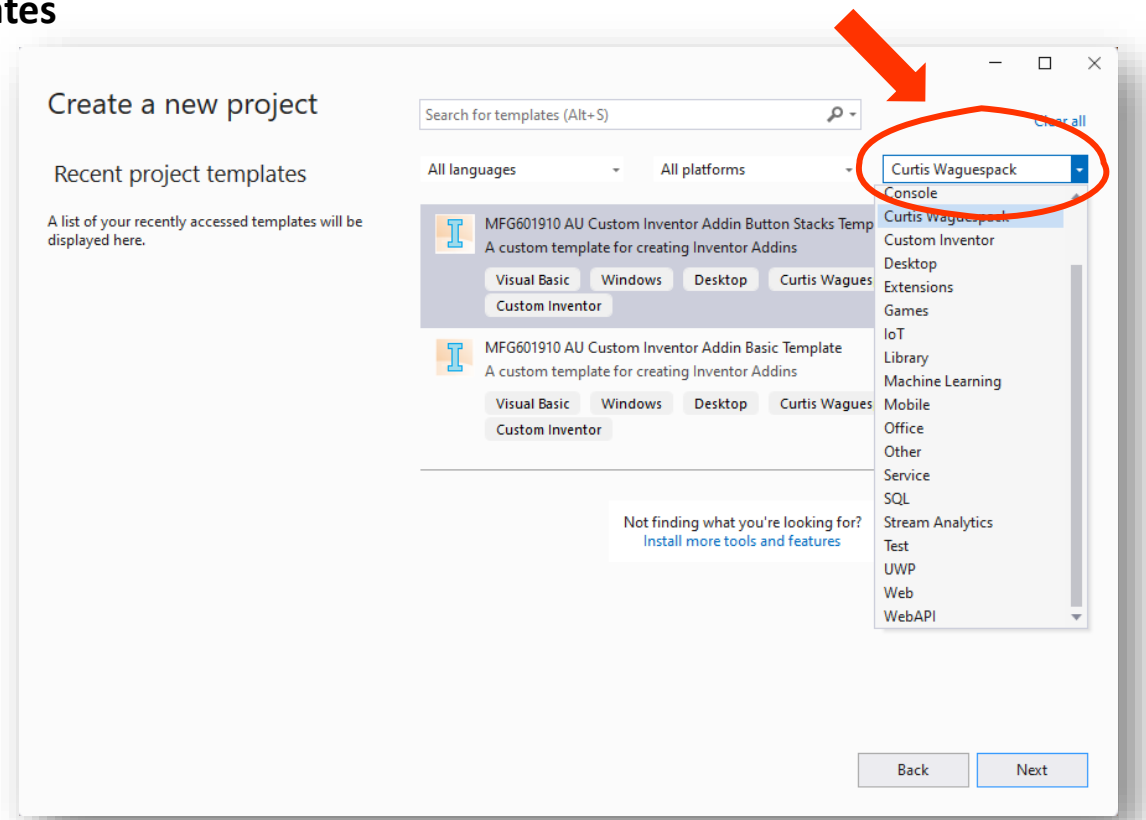
- Type “Inventor” in the search bar to filter for just the templates you want to use
- You should see the templates show up, similar to this image.



# Creating an Add-in

## Find the Inventor Add-in Templates

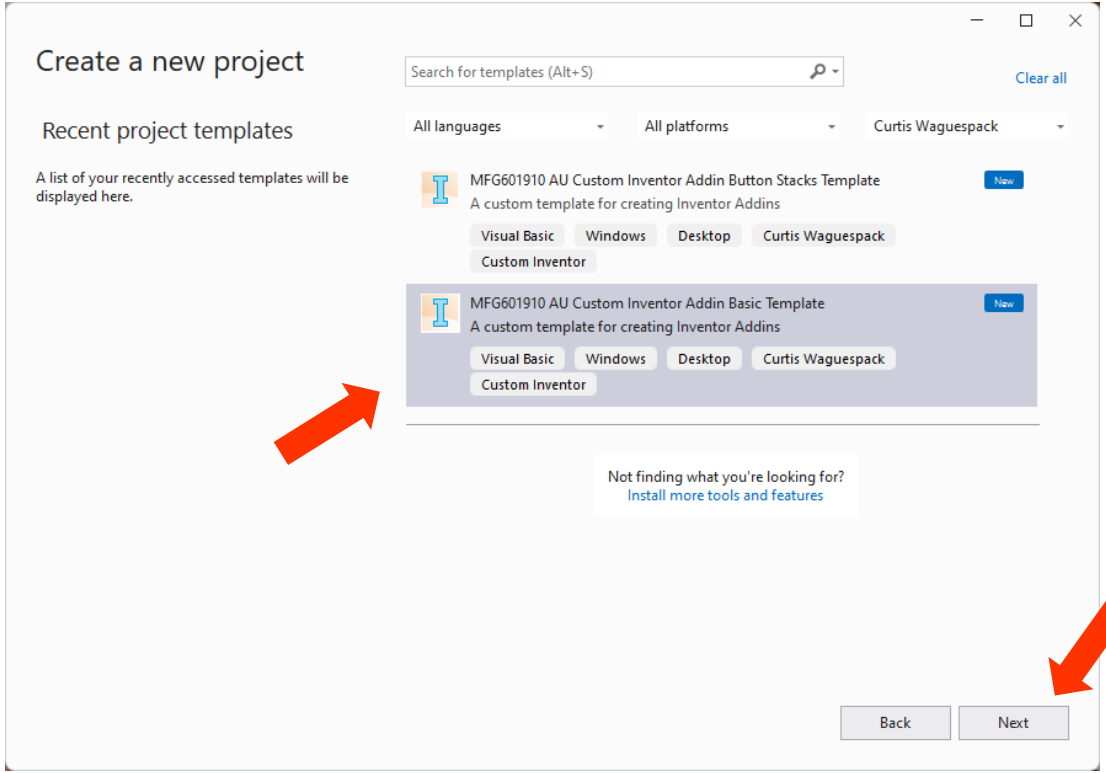
- Or you can use the filter drop-down to filter for just the add-ins provided with the class materials by selecting my name from the list



# Creating an Add-in

## Find the Inventor Add-in Templates

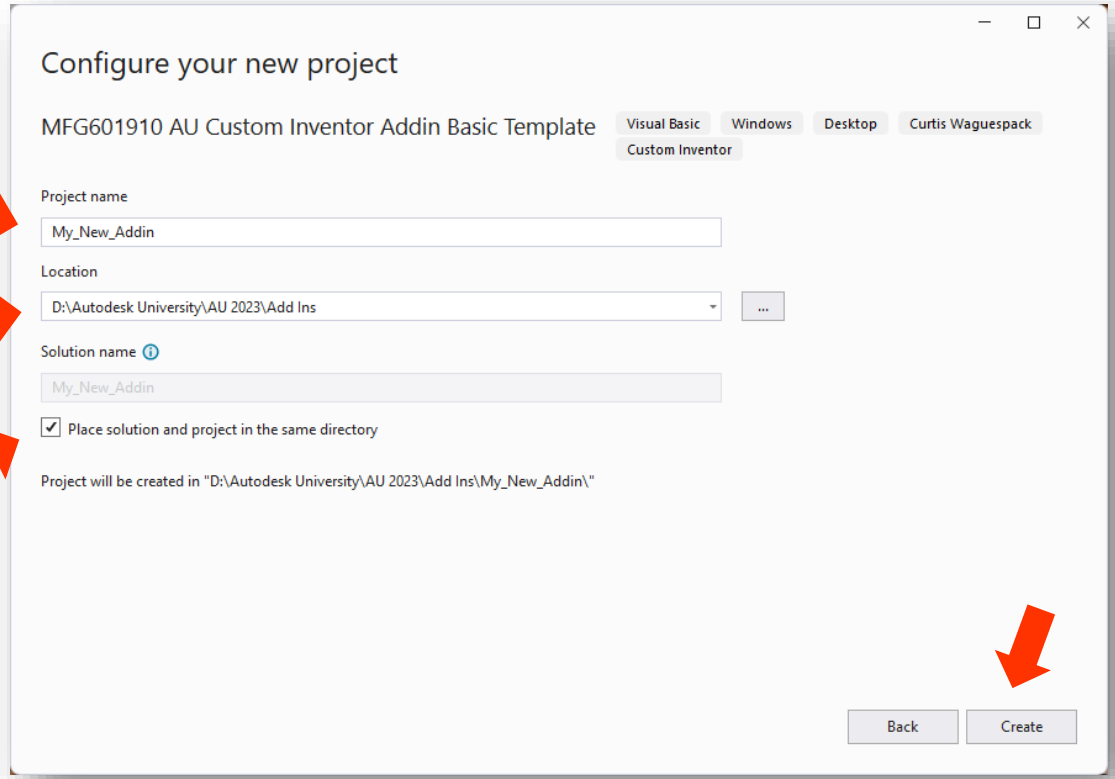
- Select the template called: **MFG601910 AU Custom Inventor Addin Template Basic**
- Click the **Next** button



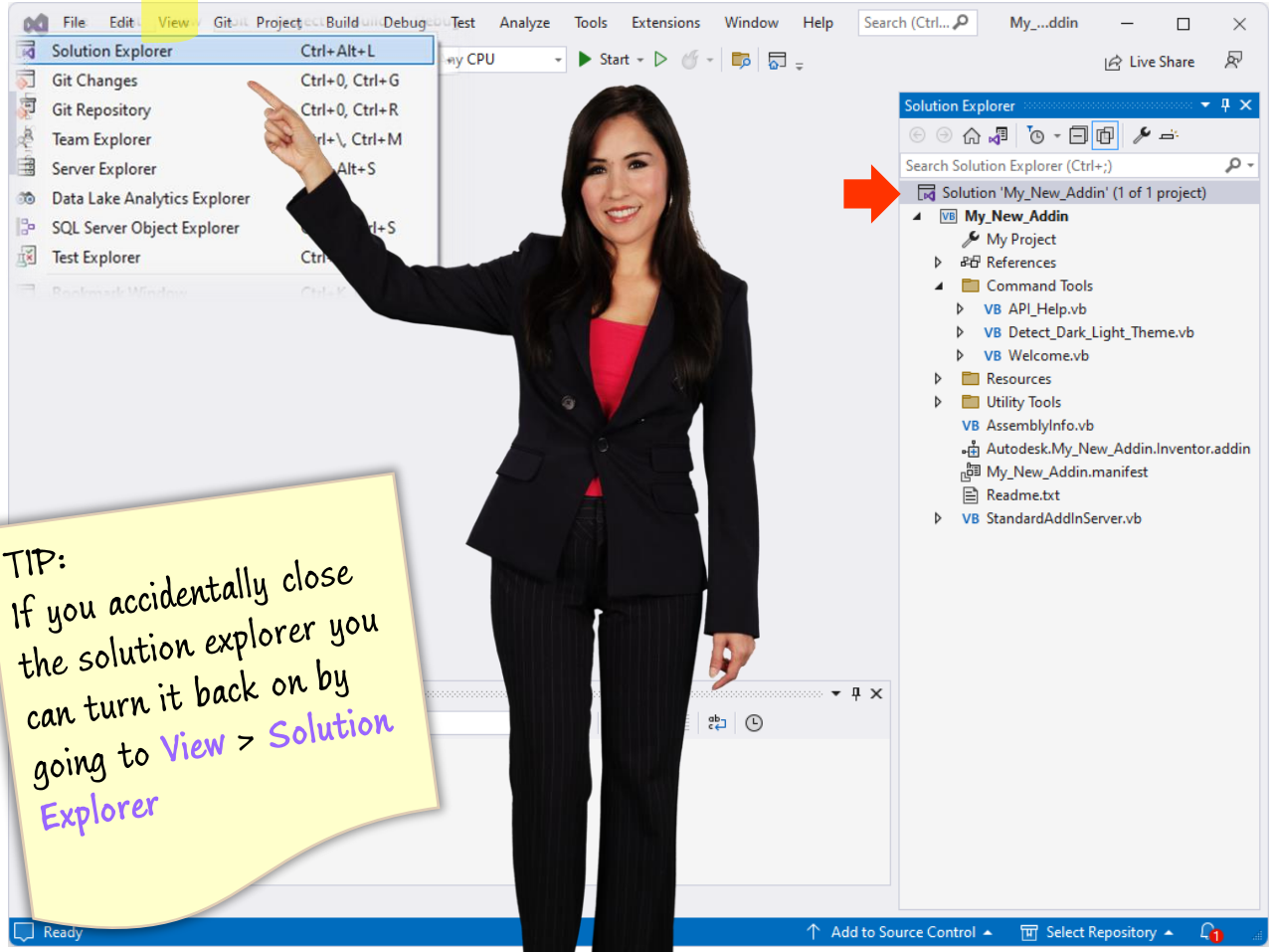
# Creating an Add-in

## Find the Inventor Add-in Templates

- Enter the project name
  - Do not use spaces!
  - Do not use dashes!
  - Spaces and dashes in the project name will cause the add-in not to load.
- Select the location where you want to store your add-in code files
- Optionally select the “Place solution and project in the same directory” checkbox
- Click Create to create the Project

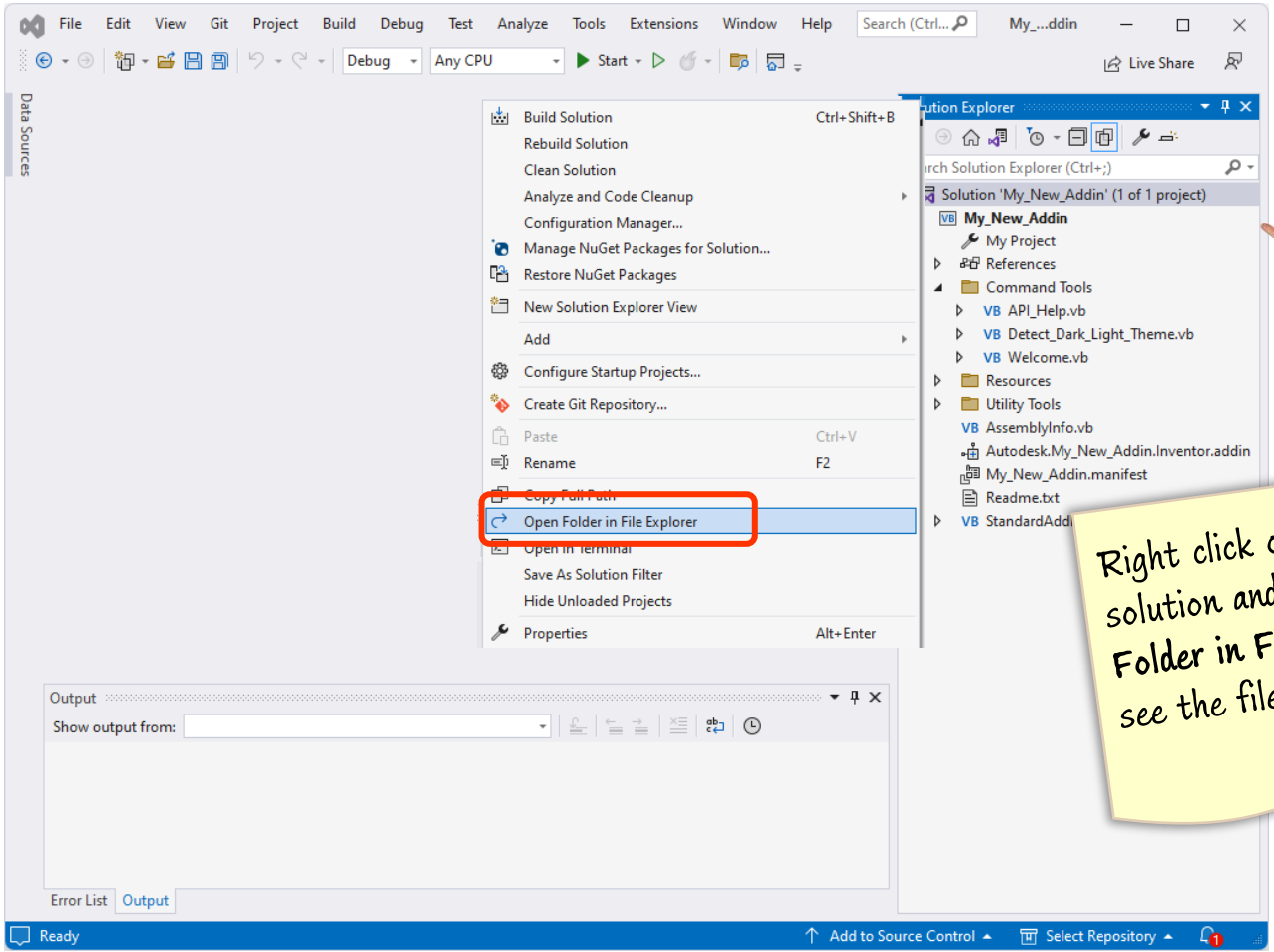


# Creating an Add-in




- The solution is shown in the Solution Explorer

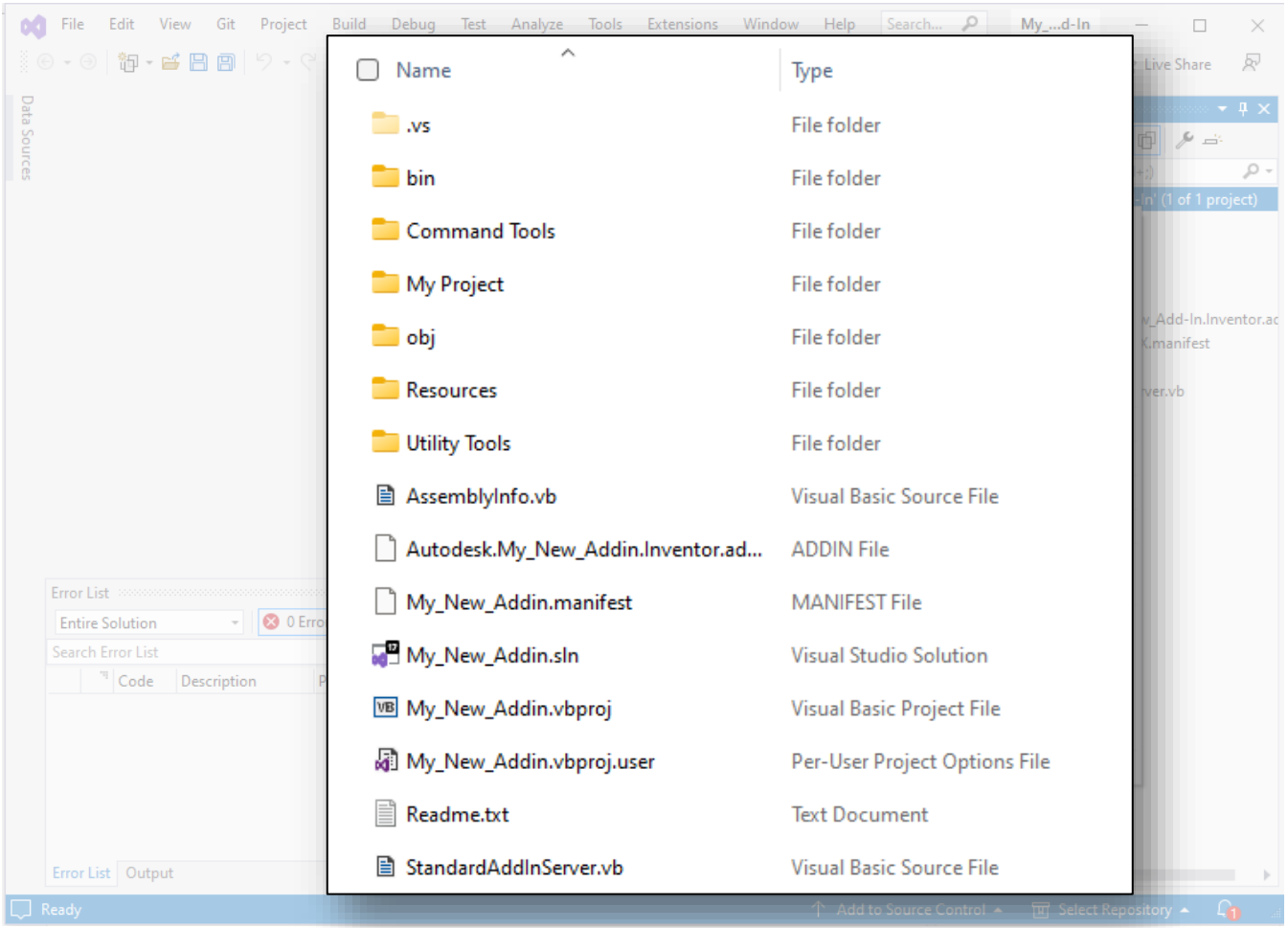
# Creating an Add-in




- The solution is shown in the Solution Explorer

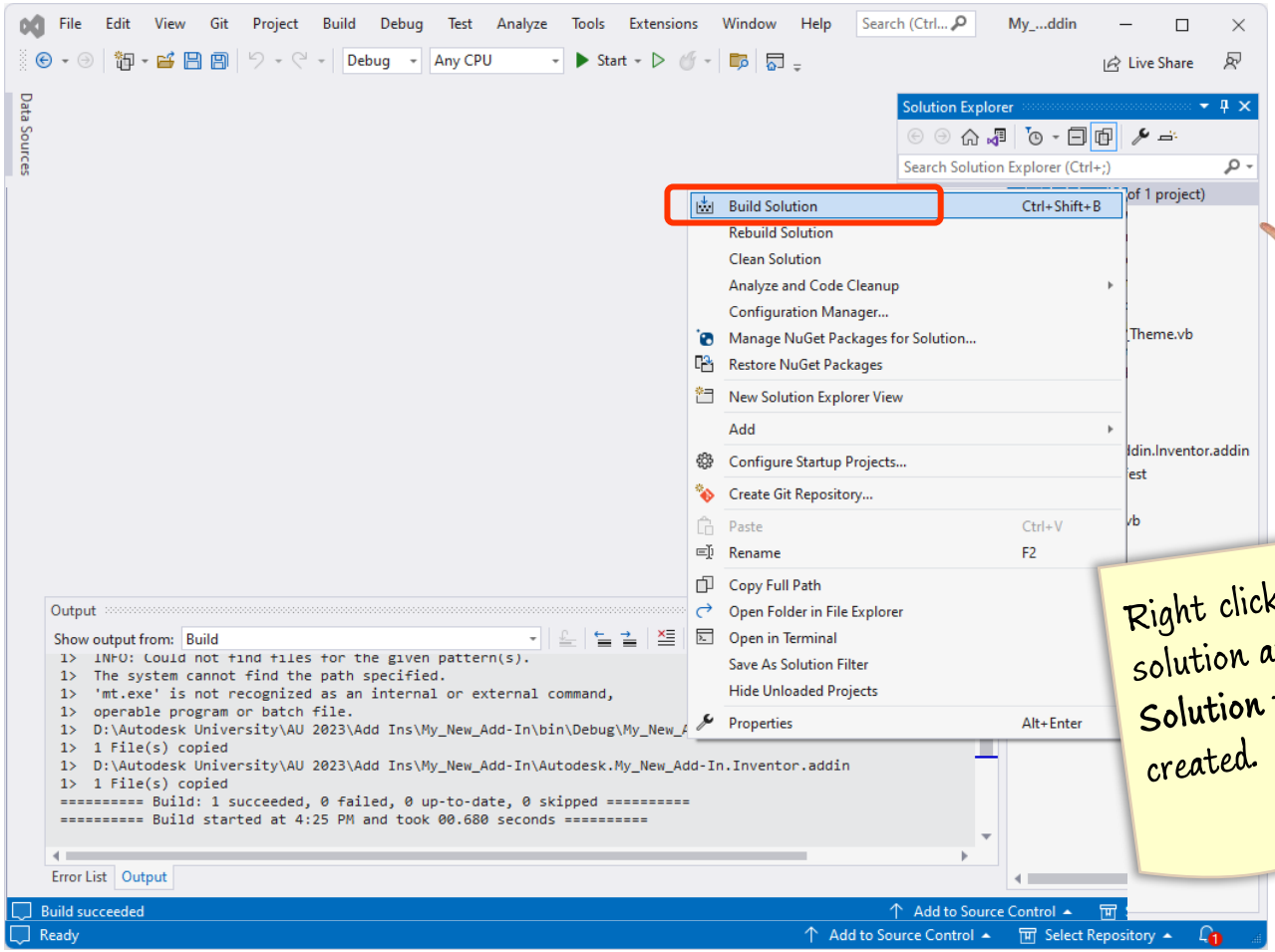
Right click on the solution and choose Open Folder in File Explorer to see the files created.

# Creating an Add-in




- The folder will contain files and folders similar to image shown here

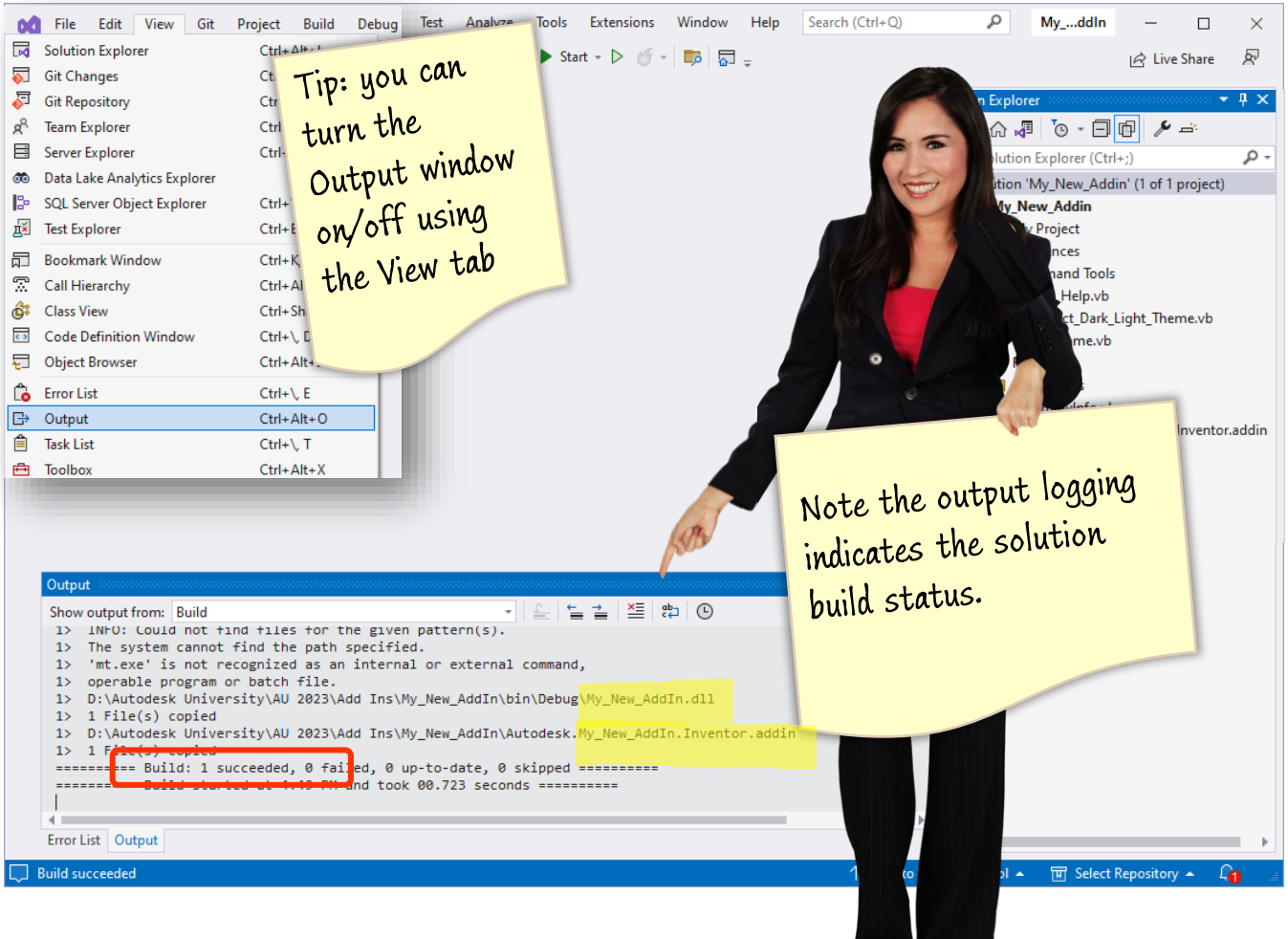
# Creating an Add-in




- The solution is shown in the Solution Explorer

Right click on the solution and choose **Build Solution** to see the files created.

# Creating an Add-in



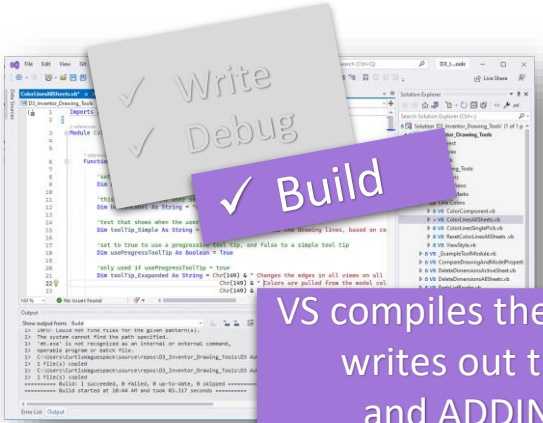

- The Build results are shown in the Output window

# Overall Process

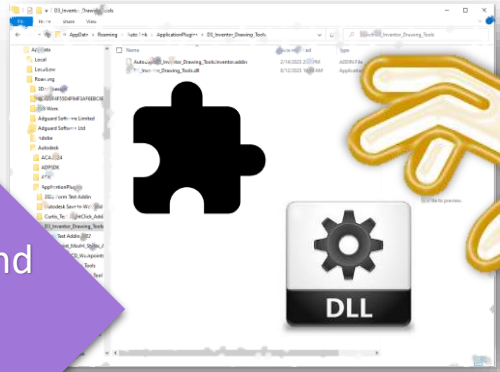

Visual Studio

Autodesk Application Plugins Folder

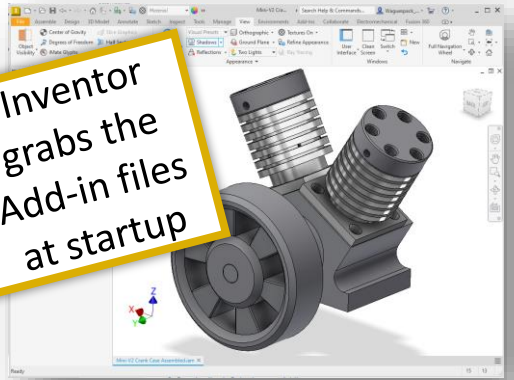
Autodesk Inventor




VS compiles the code and writes out the DLL and ADDIN files

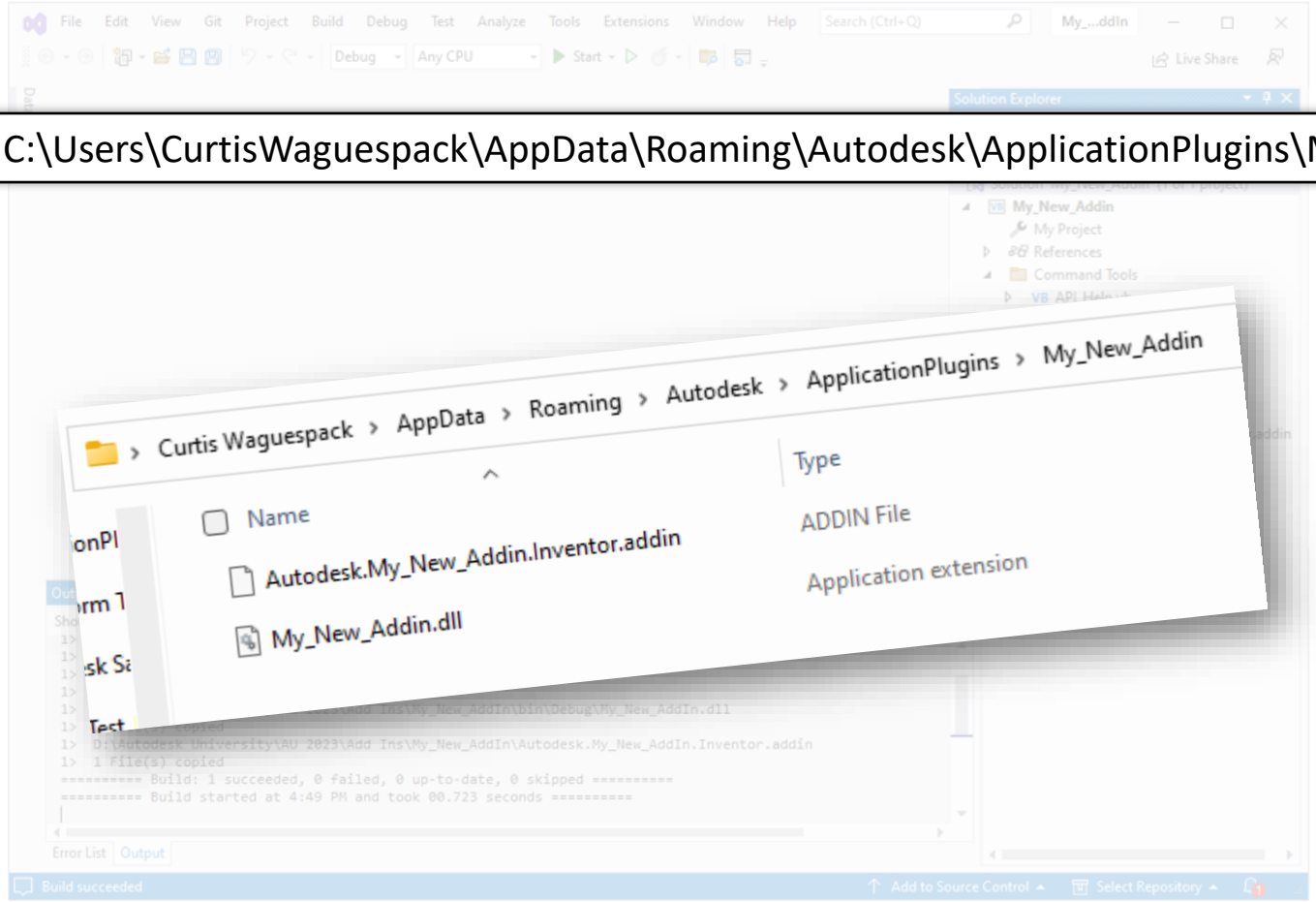



Inventor grabs the Add-in files at startup



# Creating an Add-in

C:\Users\CurtisWaguespack\AppData\Roaming\Autodesk\ApplicationPlugins\My_New_Addin



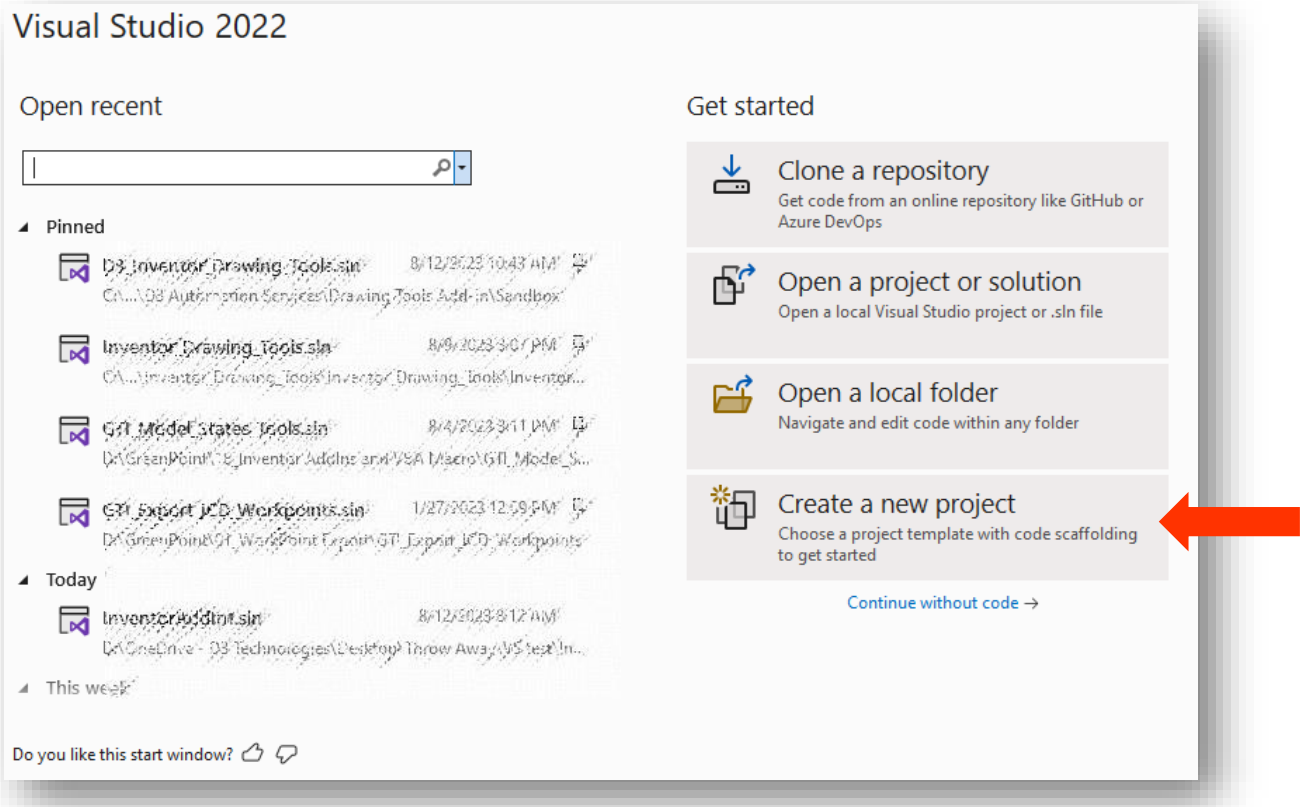

- Browse to the ApplicationPluggins folder and observe the files created during the Build

# Creating an Add-in from an improved template

Getting Started

# Creating an Add-in

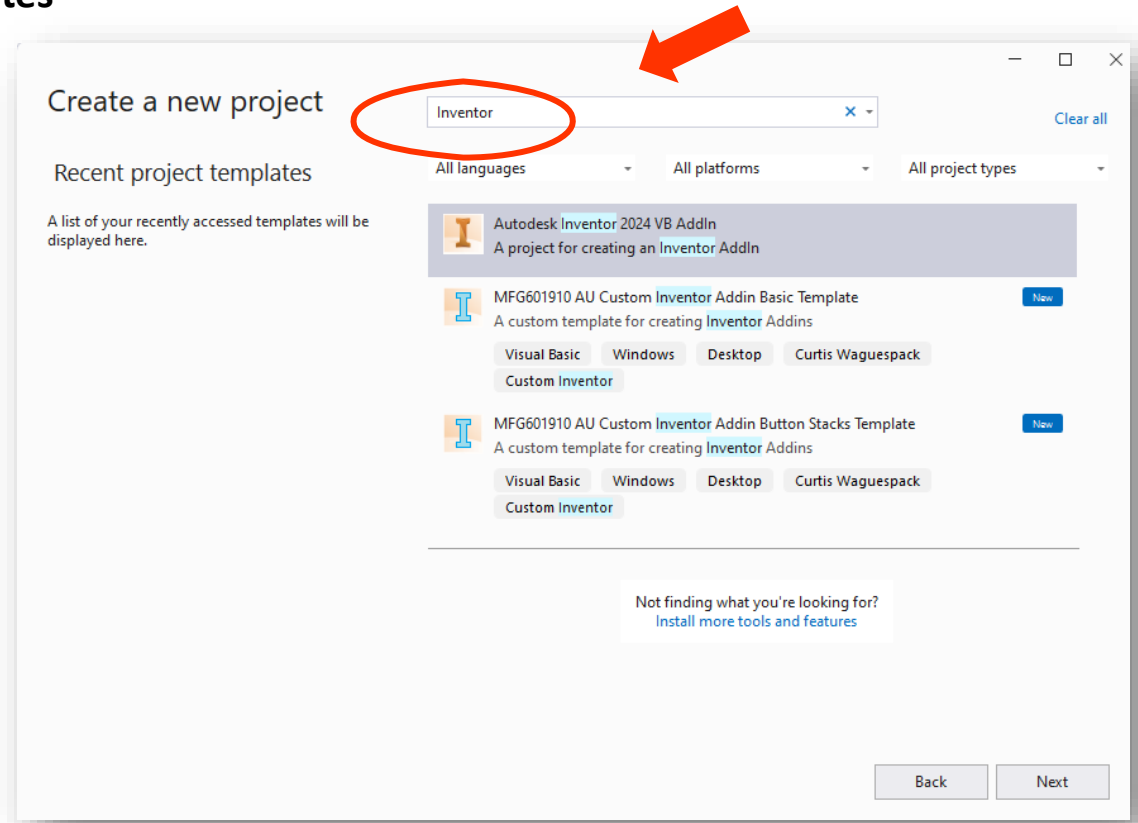
## Click Create a New Project



# Creating an Add-in

## Find the Inventor Add-in Templates

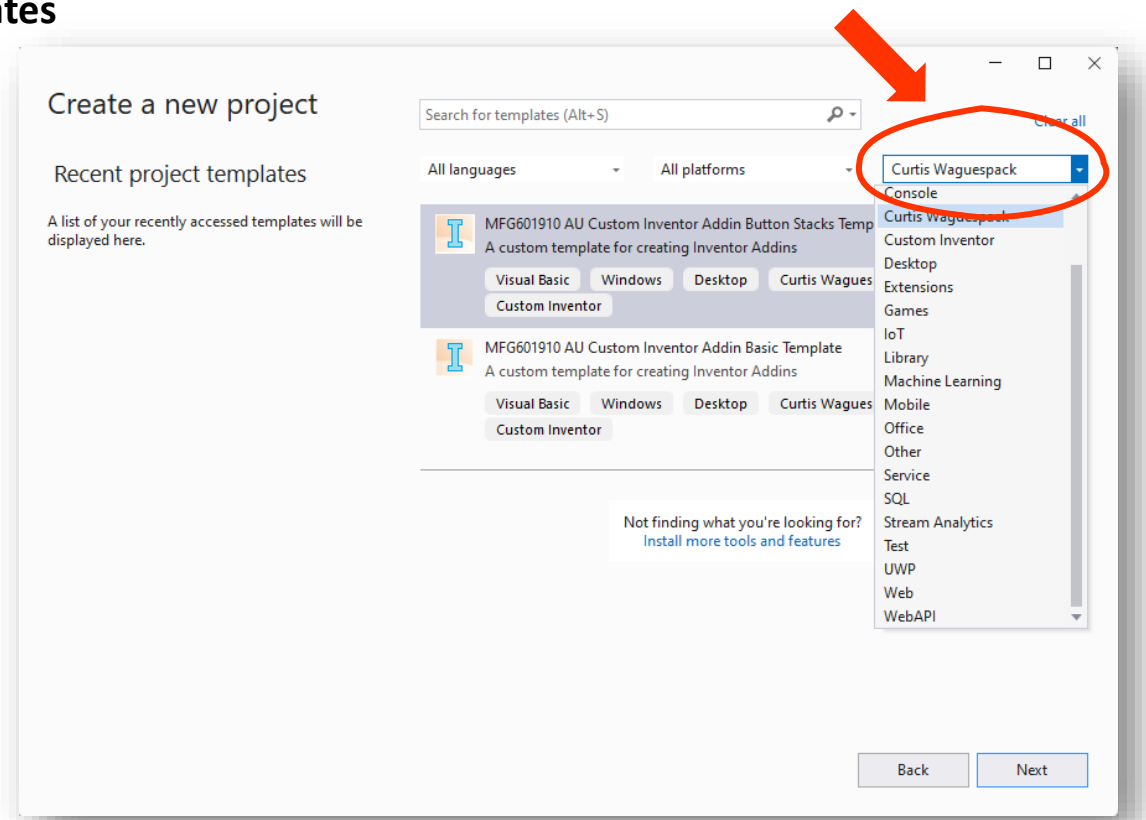
- Type “Inventor” in the search bar to filter for just the templates you want to use
- You should see the templates show up, similar to this image.



# Creating an Add-in

## Find the Inventor Add-in Templates

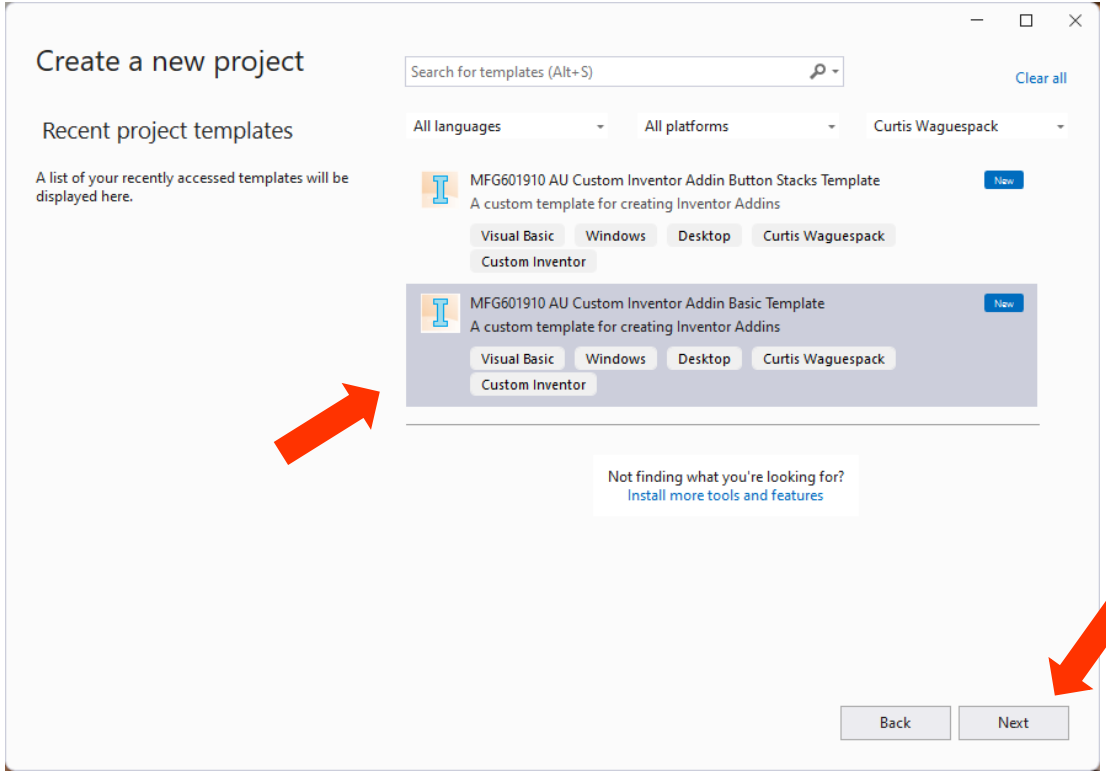
- Or you can use the filter drop-down to filter for just the add-ins provided with the class materials by selecting my name from the list



# Creating an Add-in

## Find the Inventor Add-in Templates

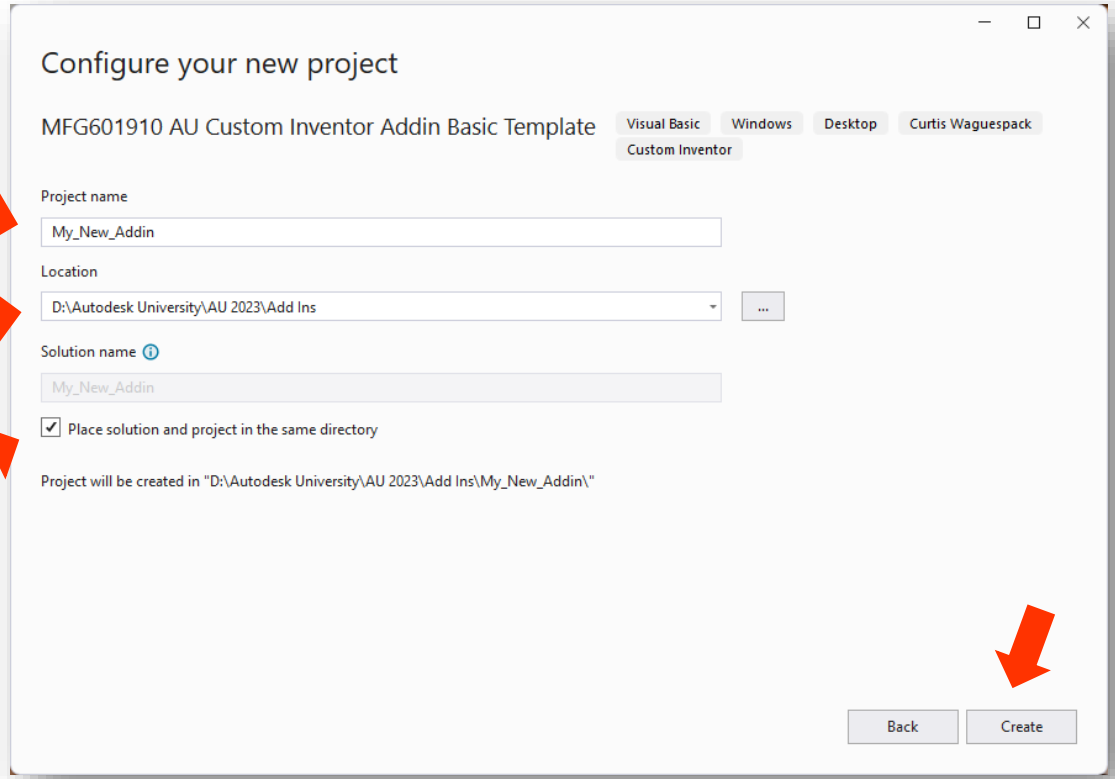
- Select the template called: **MFG601910 AU Custom Inventor Addin Template Basic**
- Click the **Next** button



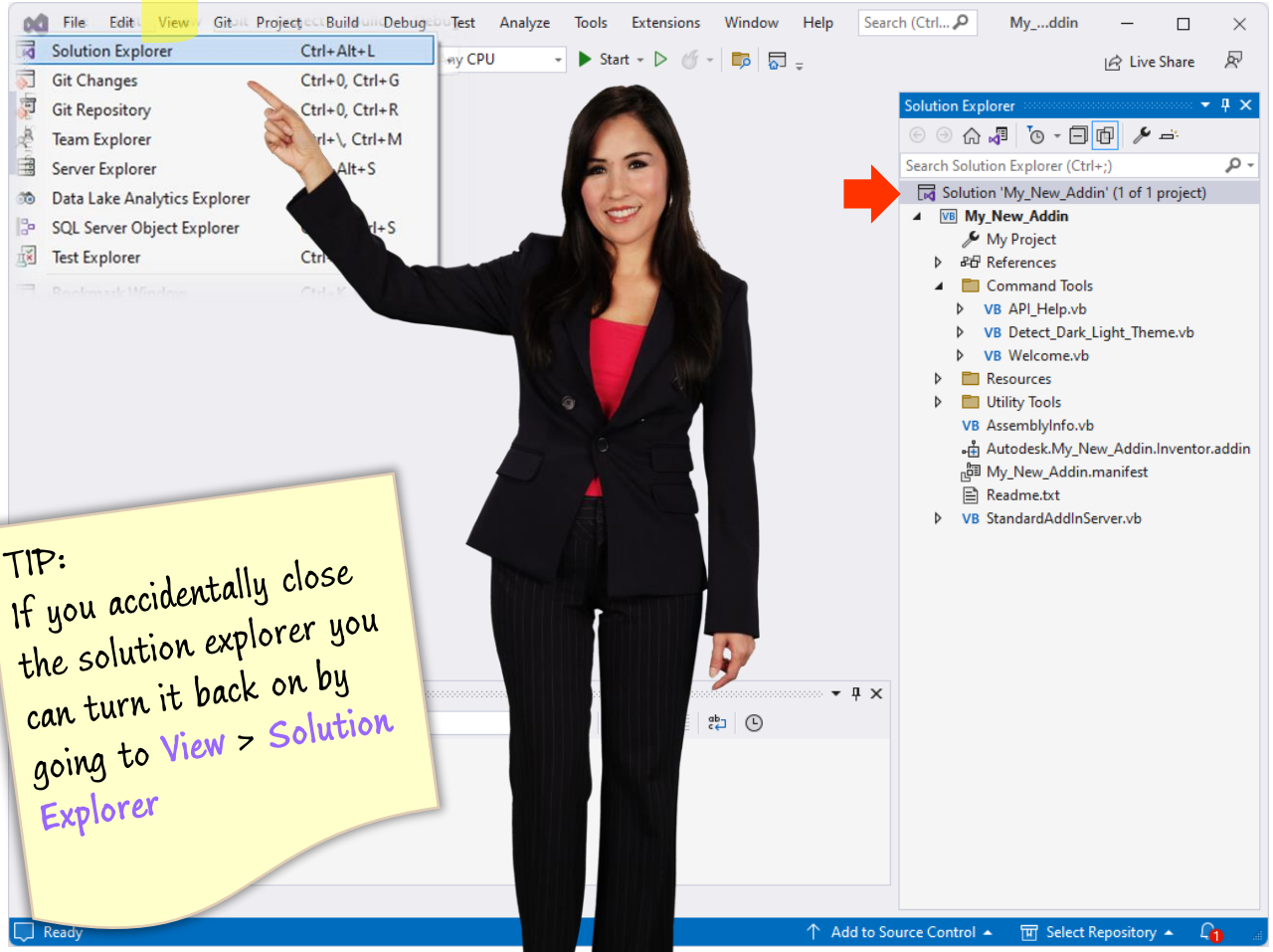
# Creating an Add-in

## Find the Inventor Add-in Templates

- Enter the project name
  - Do not use spaces!
  - Do not use dashes!
  - Spaces and dashes in the project name will cause the add-in not to load.
- Select the location where you want to store your add-in code files
- Optionally select the “Place solution and project in the same directory” checkbox
- Click Create to create the Project

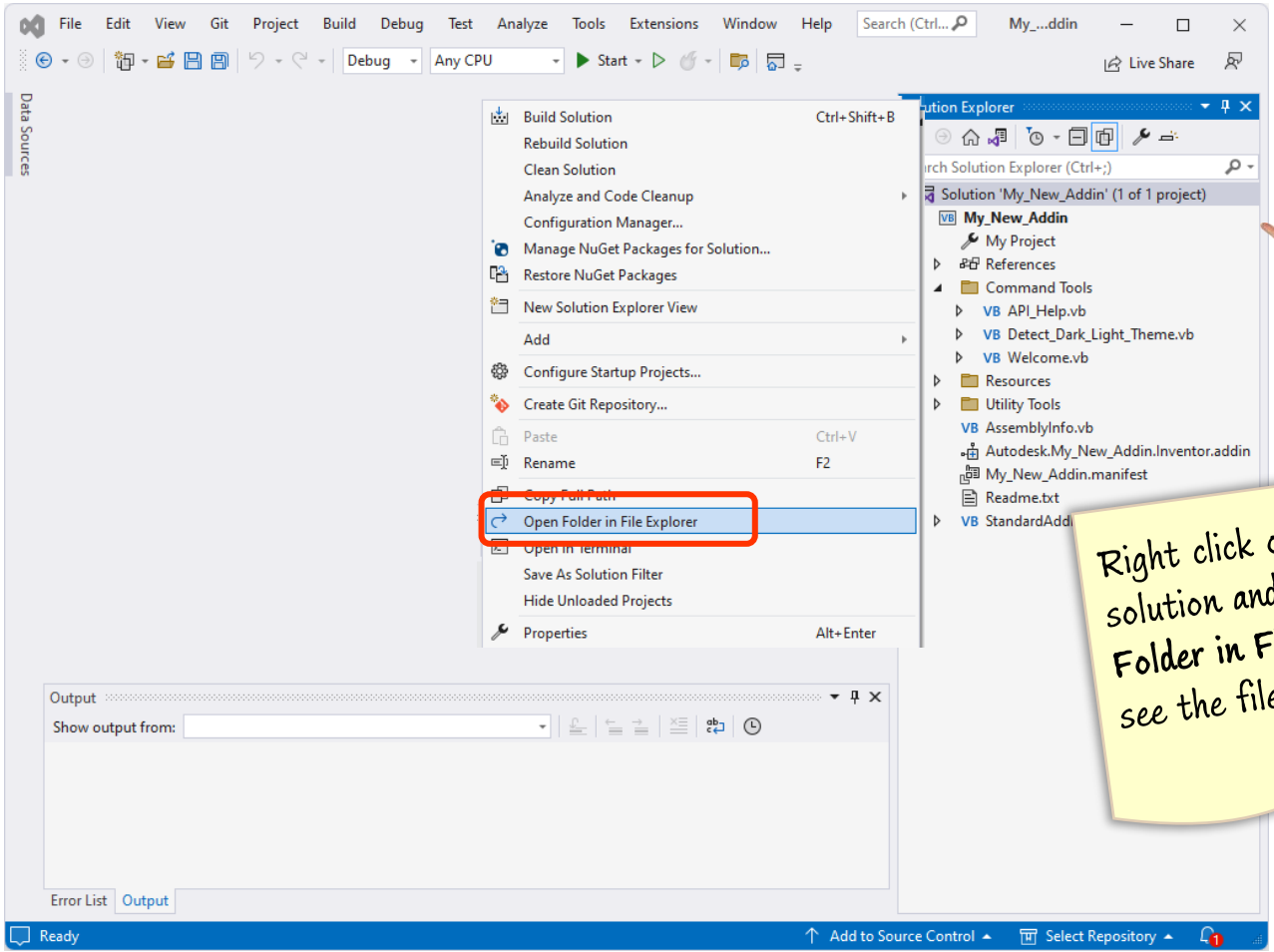


# Creating an Add-in




- The solution is shown in the Solution Explorer

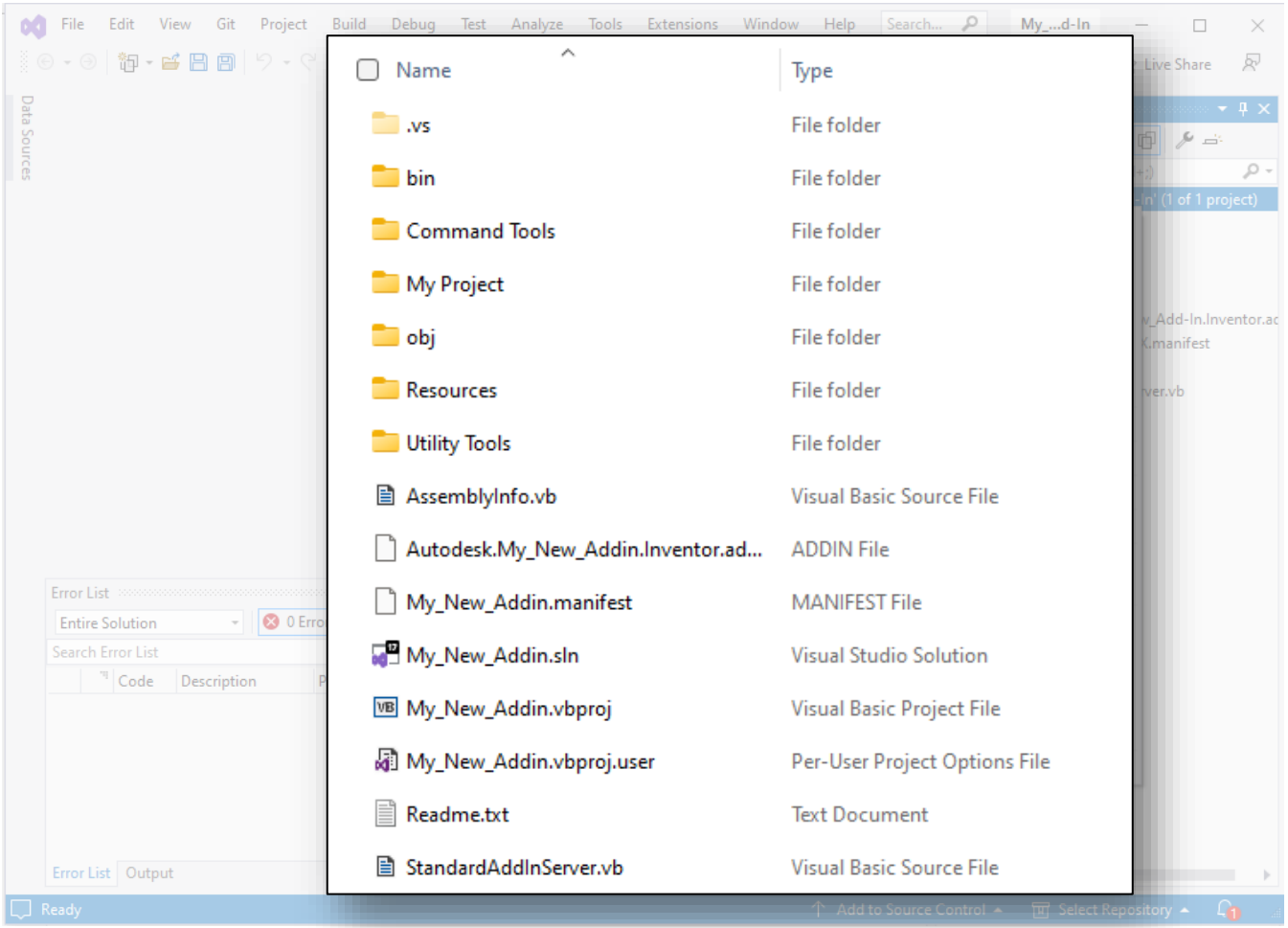
# Creating an Add-in




- The solution is shown in the Solution Explorer

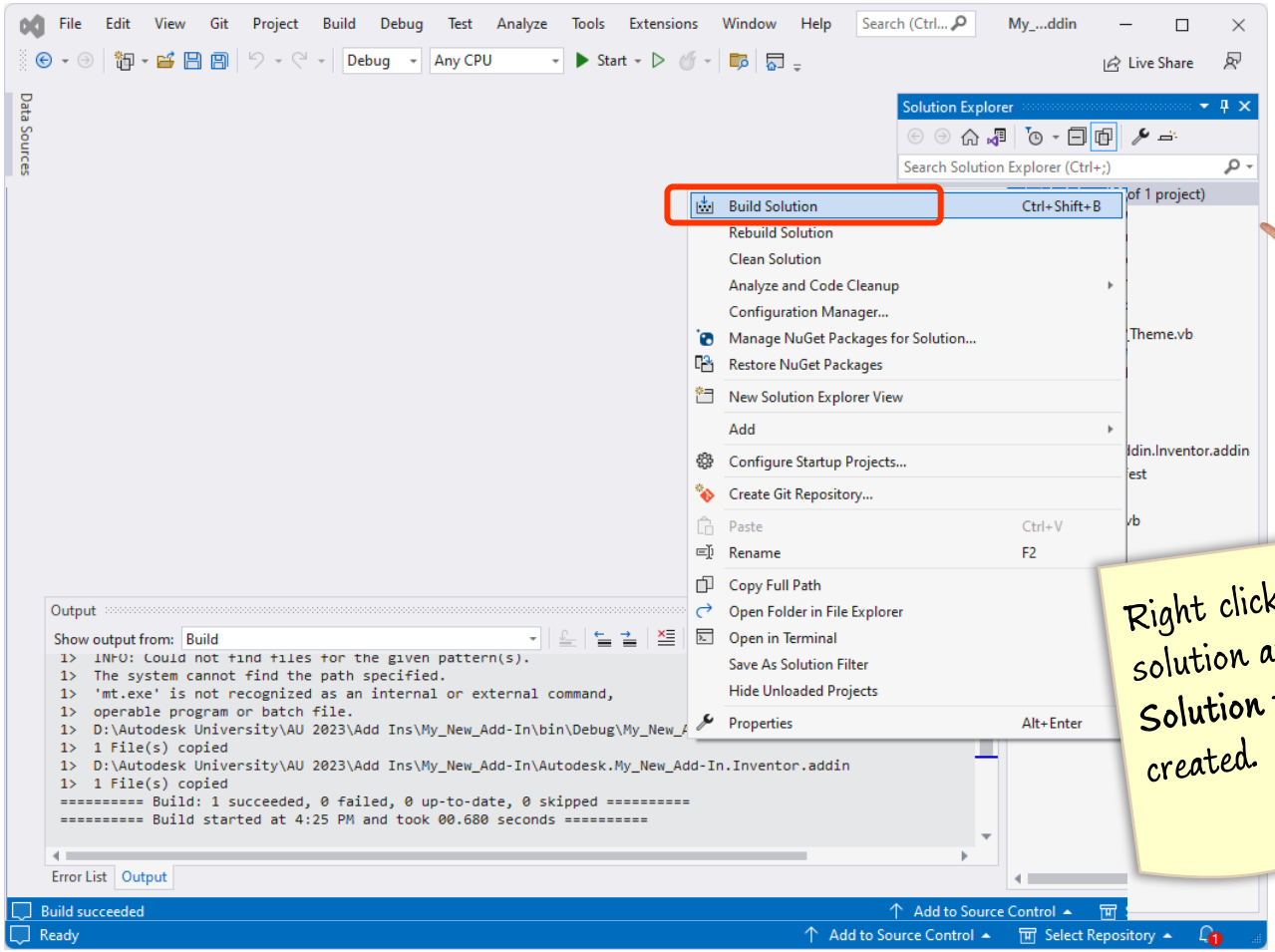
Right click on the solution and choose Open Folder in File Explorer to see the files created.

# Creating an Add-in




- The folder will contain files and folders similar to image shown here

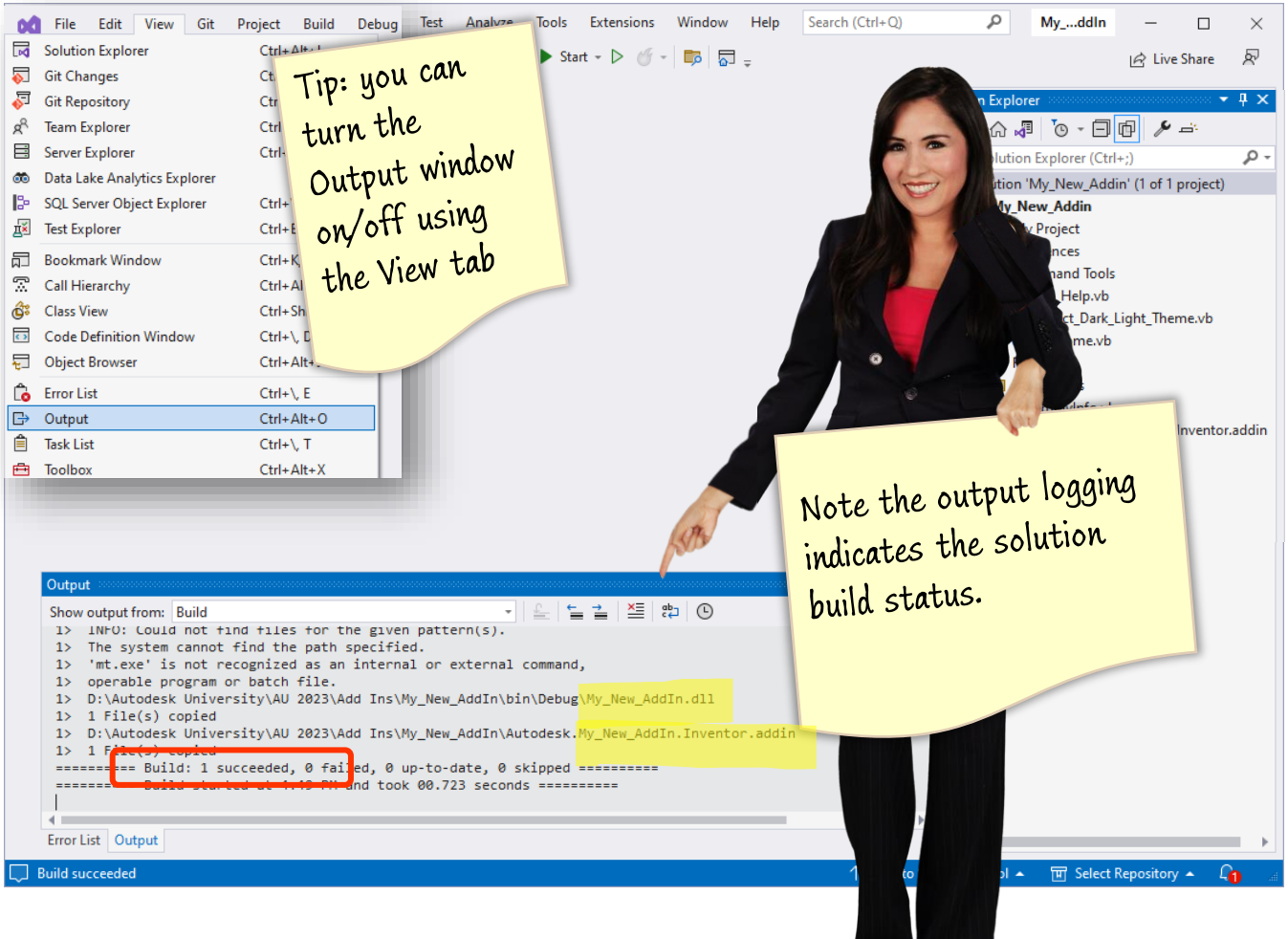
# Creating an Add-in




- The solution is shown in the Solution Explorer

Right click on the solution and choose **Build Solution** to see the files created.

# Creating an Add-in



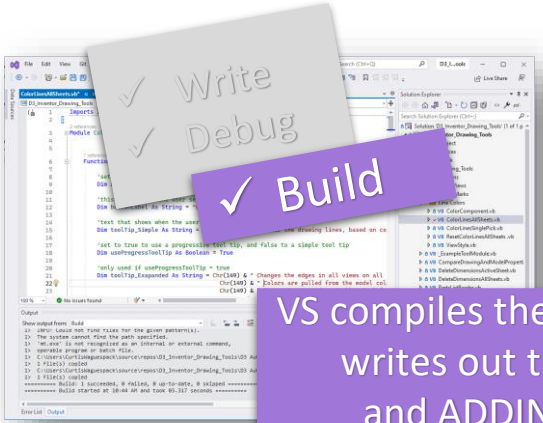

- The Build results are shown in the Output window

# Overall Process

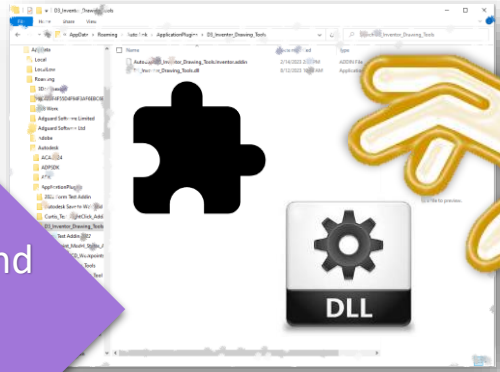

Visual Studio

Autodesk Application Plugins Folder

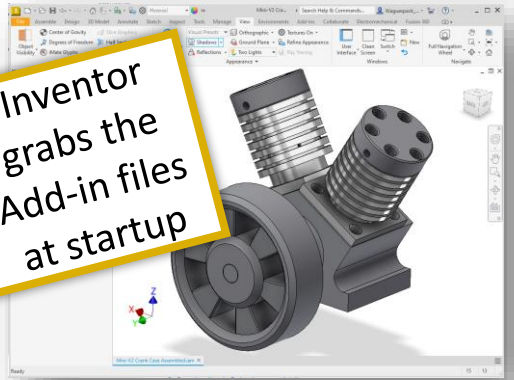
Autodesk Inventor




VS compiles the code and writes out the DLL and ADDIN files

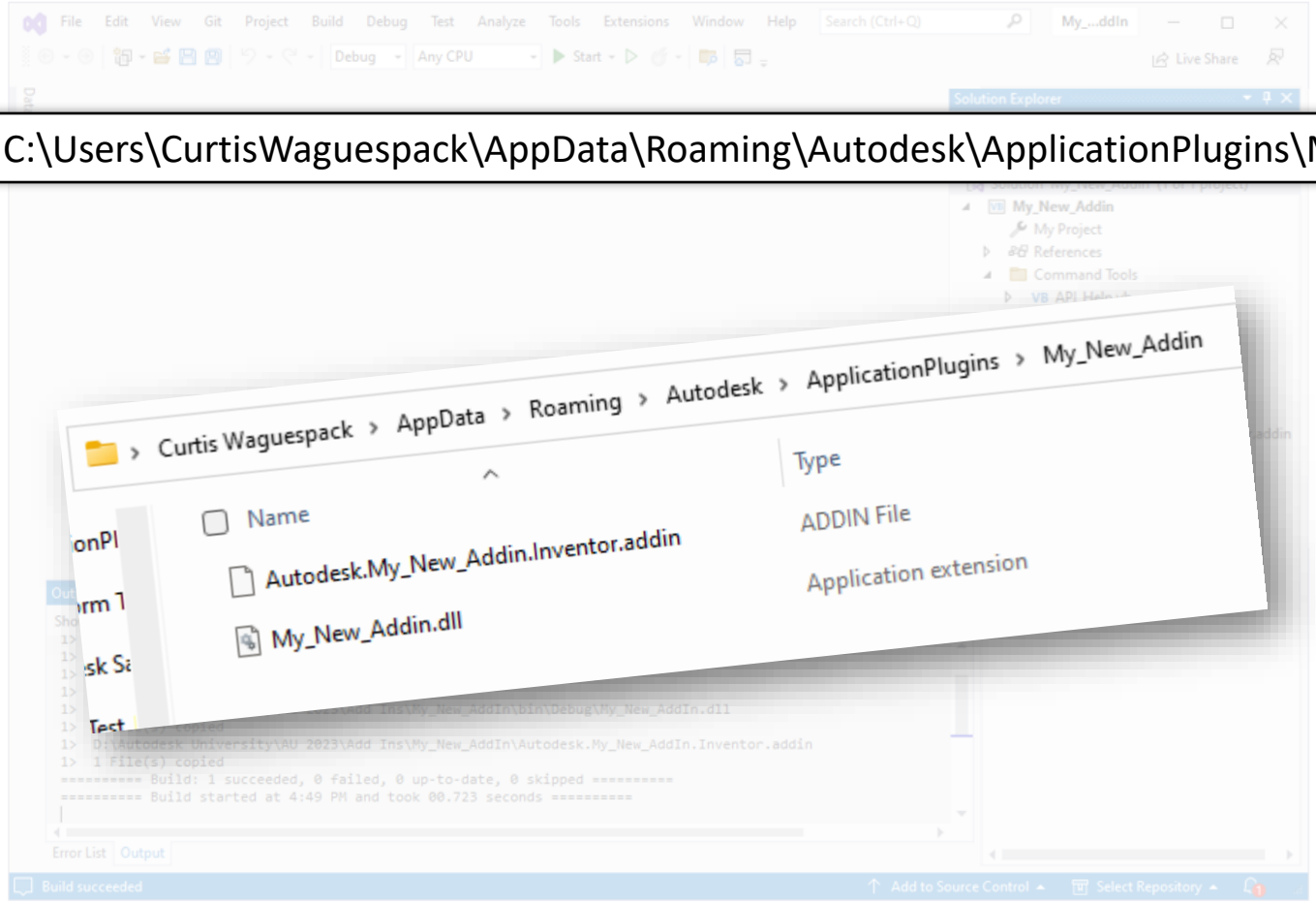



Inventor grabs the Add-in files at startup



# Creating an Add-in

C:\Users\CurtisWaguespack\AppData\Roaming\Autodesk\ApplicationPlugins\My_New_Addin

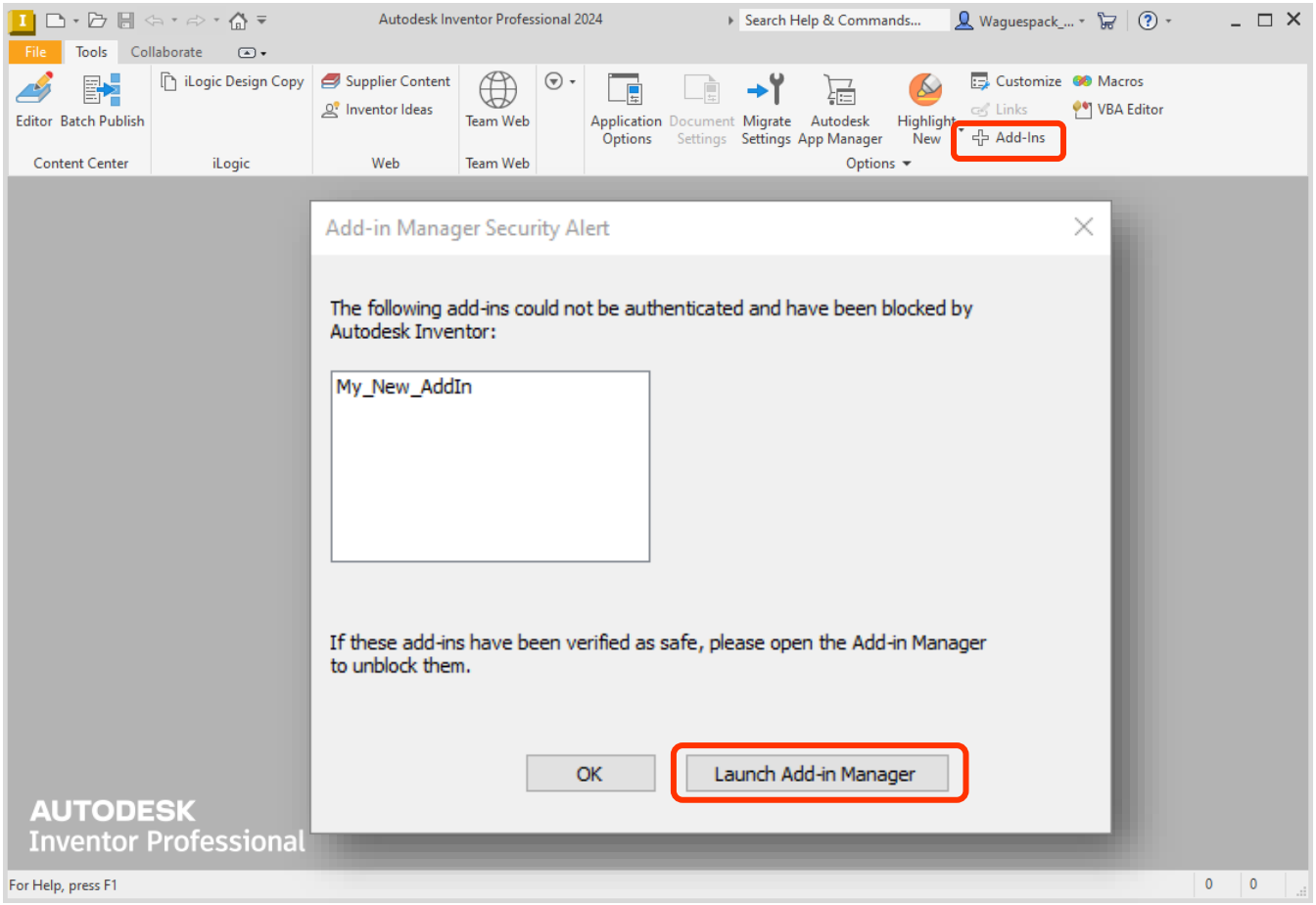



- Browse to the ApplicationPluggins folder and observe the files created during the Build

# Loading the add-in

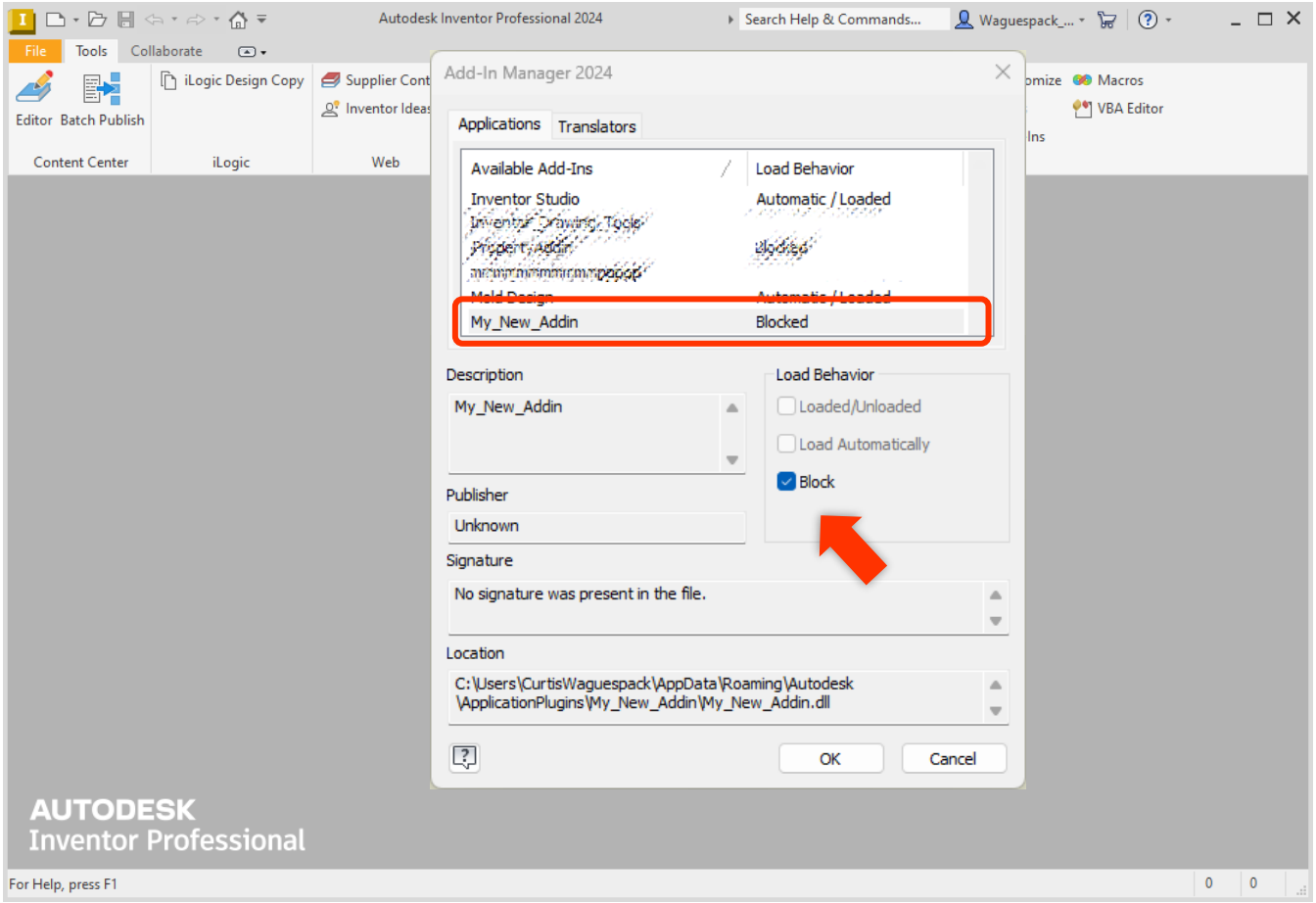
Working in Inventor

# Loading an Add-in



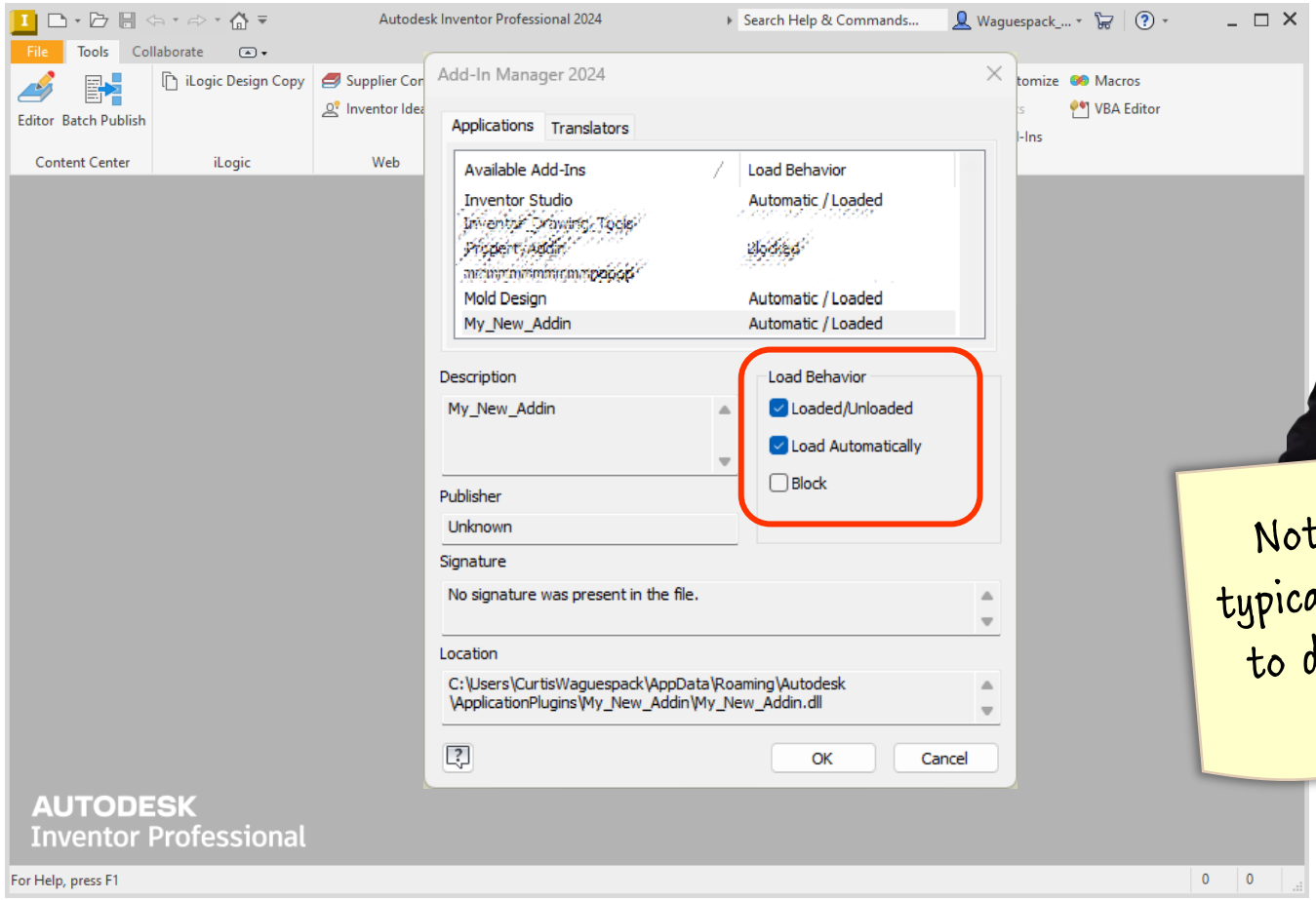

- Open Inventor.
- Note that you will be greeted by a message informing you that your add-in was blocked from loading
- Click the Launch Add-In Manager button or got to Tools > Add-ins

# Loading an Add-in




- With the add-in selected, un-check the **Block** checkbox
- And then check the **Loaded/Unloaded**
- Ad the **Load Automatically** check box

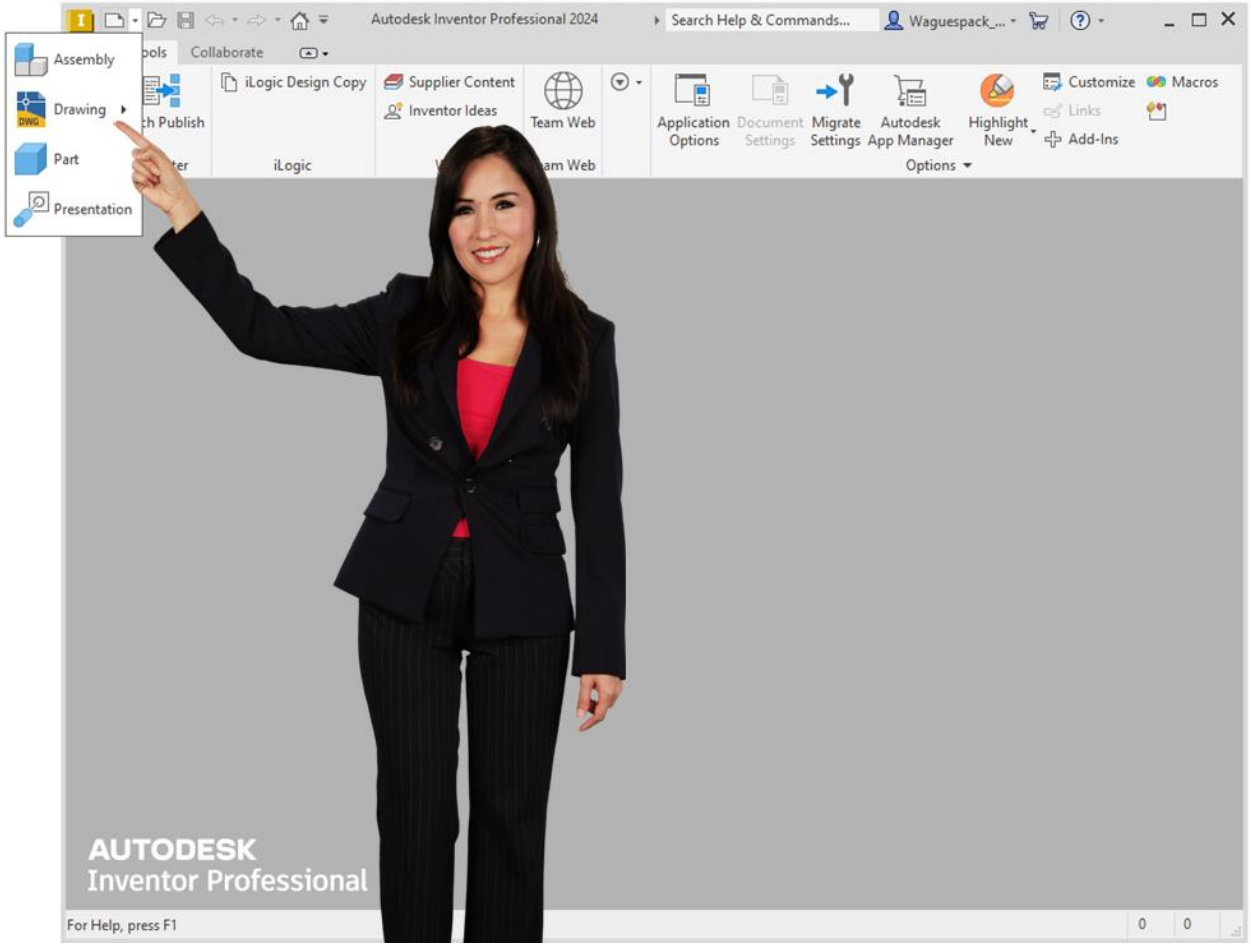
# Loading an Add-in




# Using the add-in

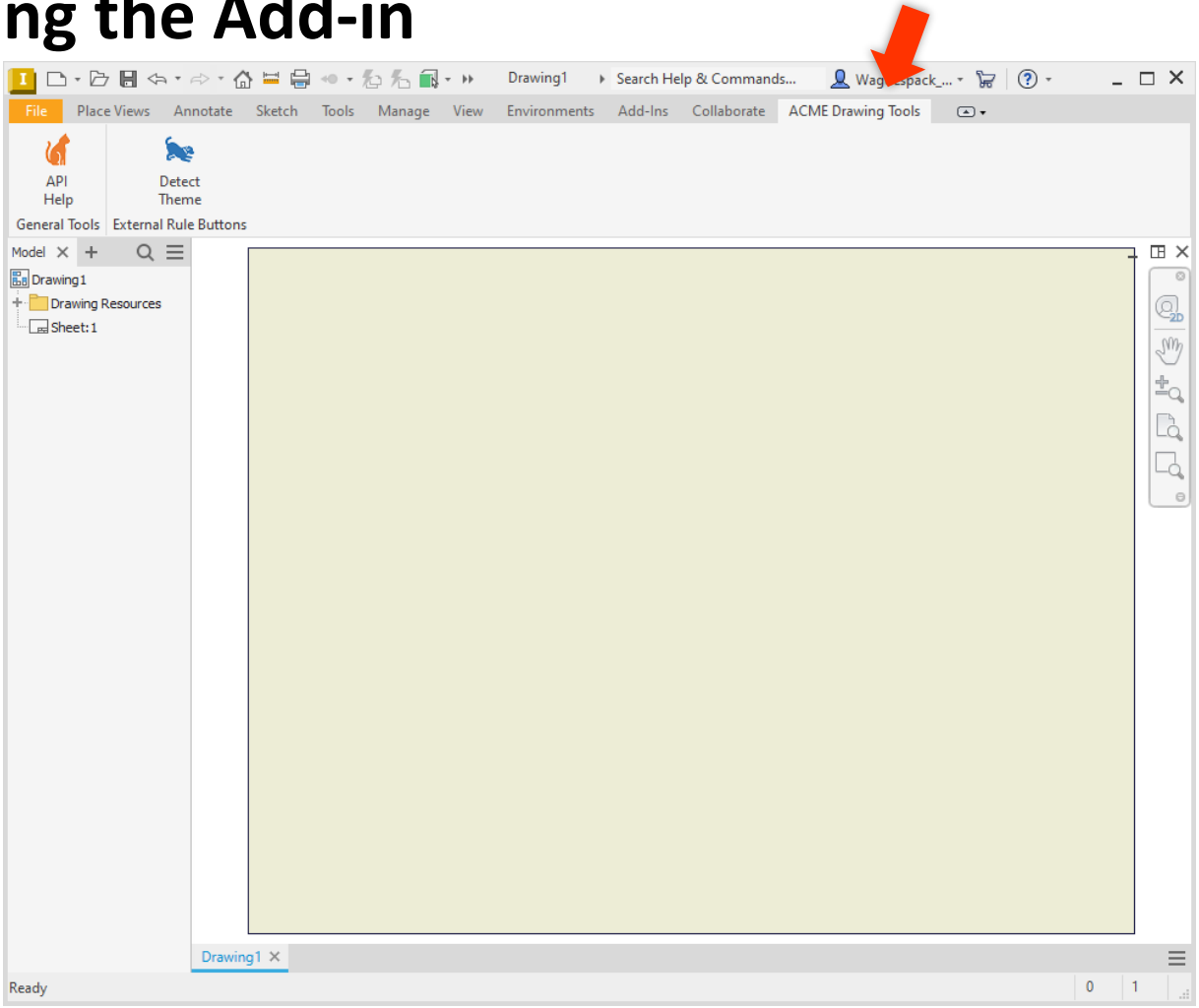
Working in Inventor

# Using the Add-in



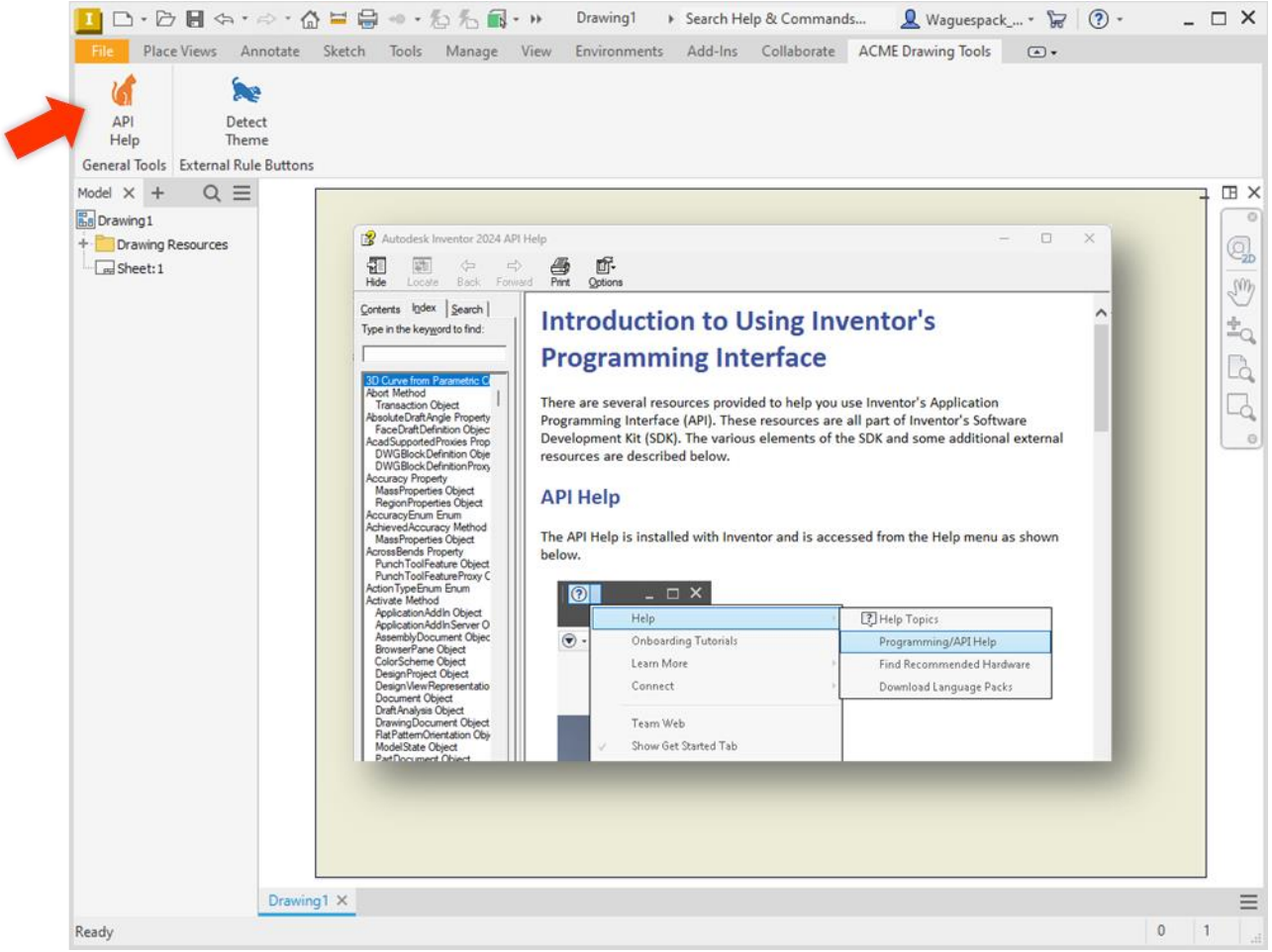

- Create a new drawing file from a template

# Using the Add-in



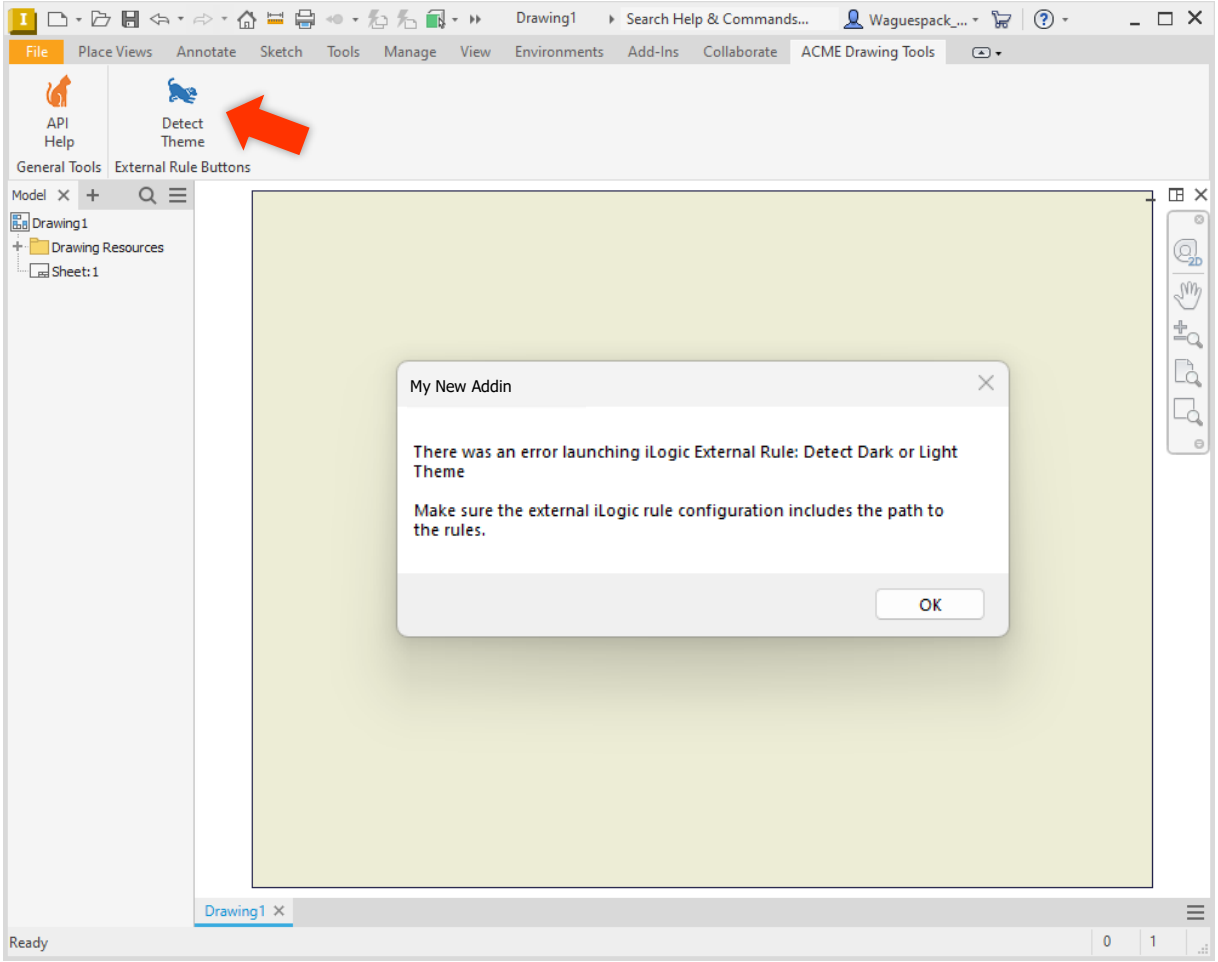

- Switch to the **ACME Drawing Tool** tab
- Note the 2 buttons found on this tab

# Using the Add-in



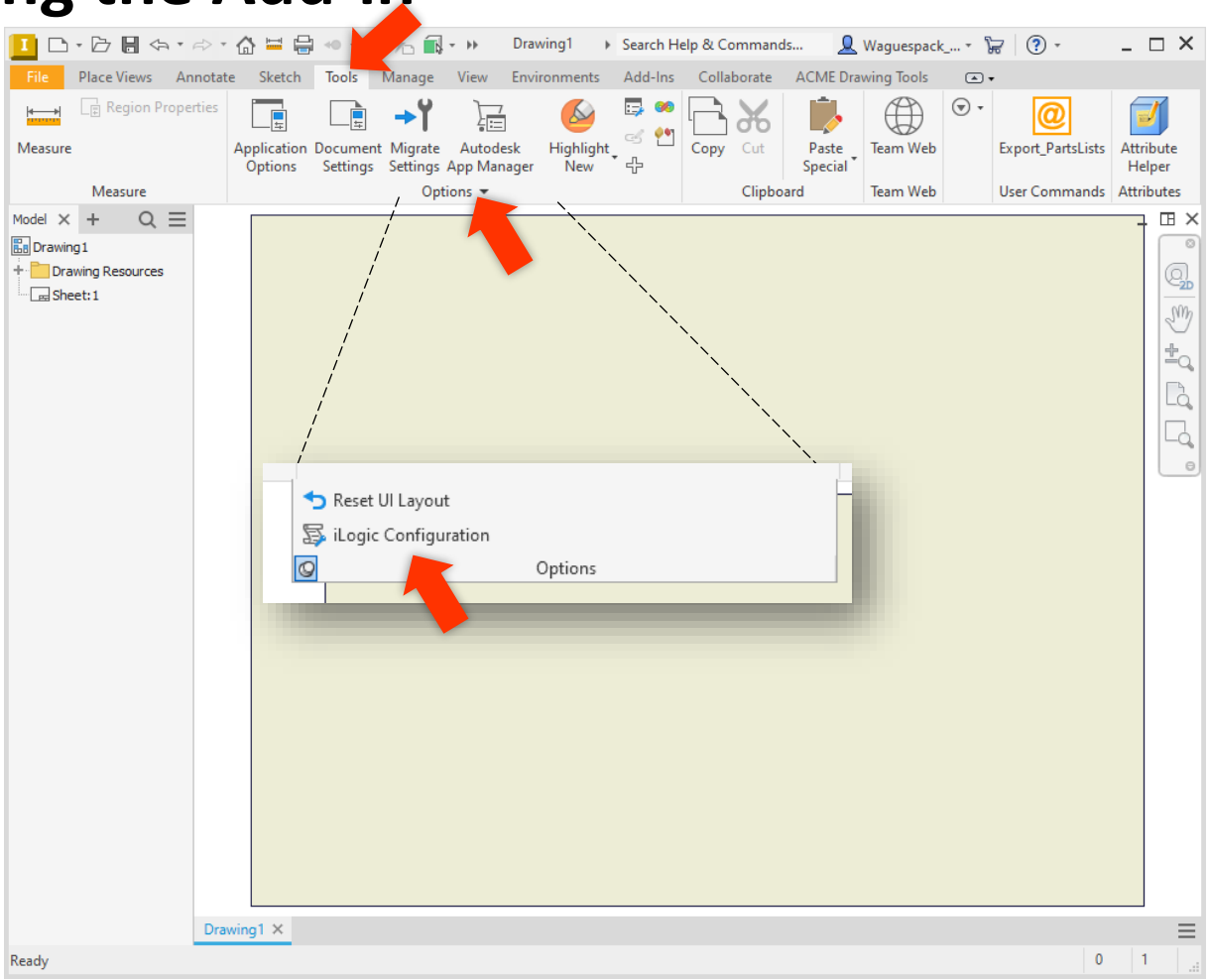

- Click the button called **API Help**
- This will open the Inventor API Help in a separate window
- This was done with code written in the add-in
- You can close the window if you like

# Using the Add-in



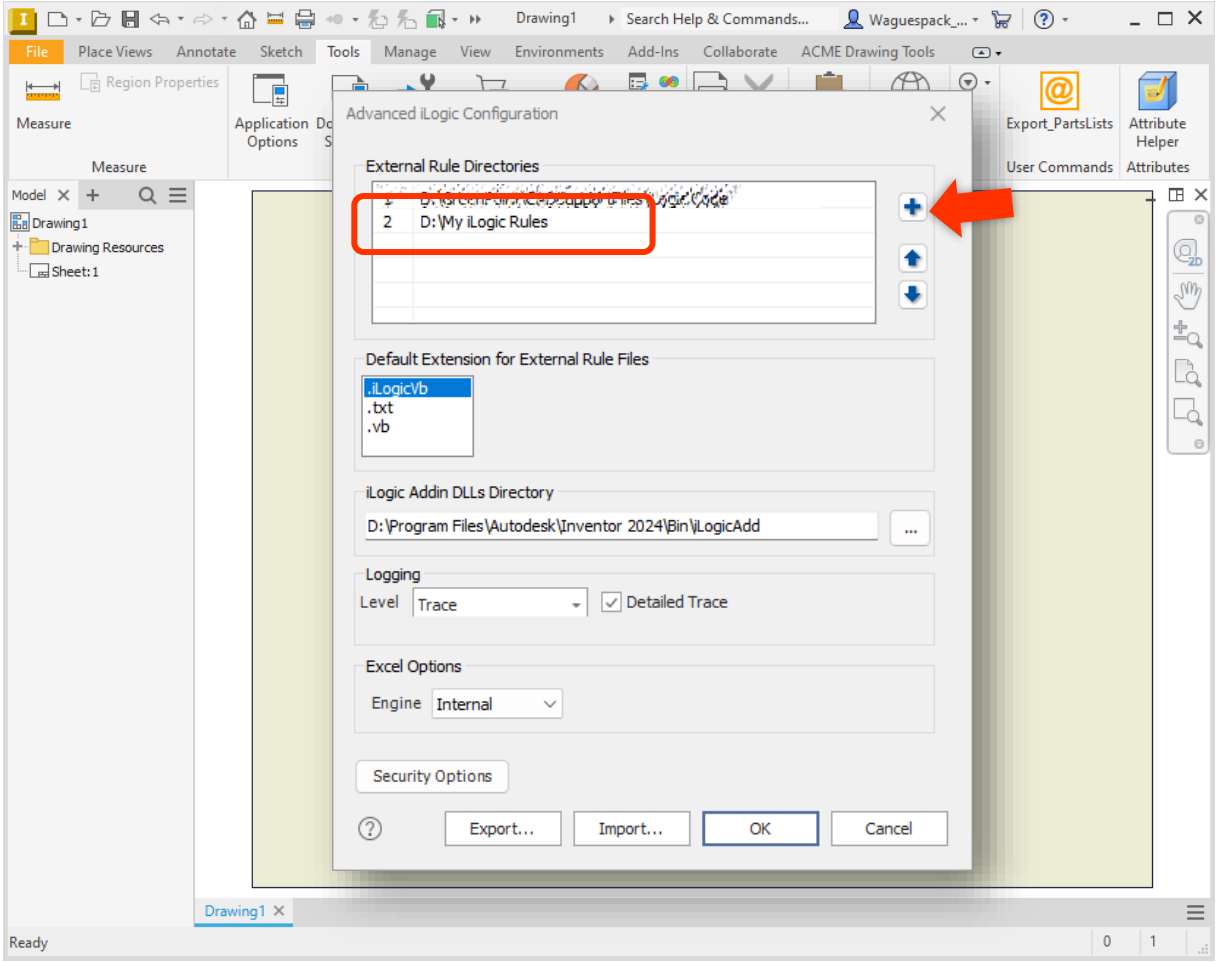

- Click the button called **API Help**
- This will open the Inventor API Help in a separate window
- This was done with code written in the add-in
- You can close the window if you like

# Using the Add-in



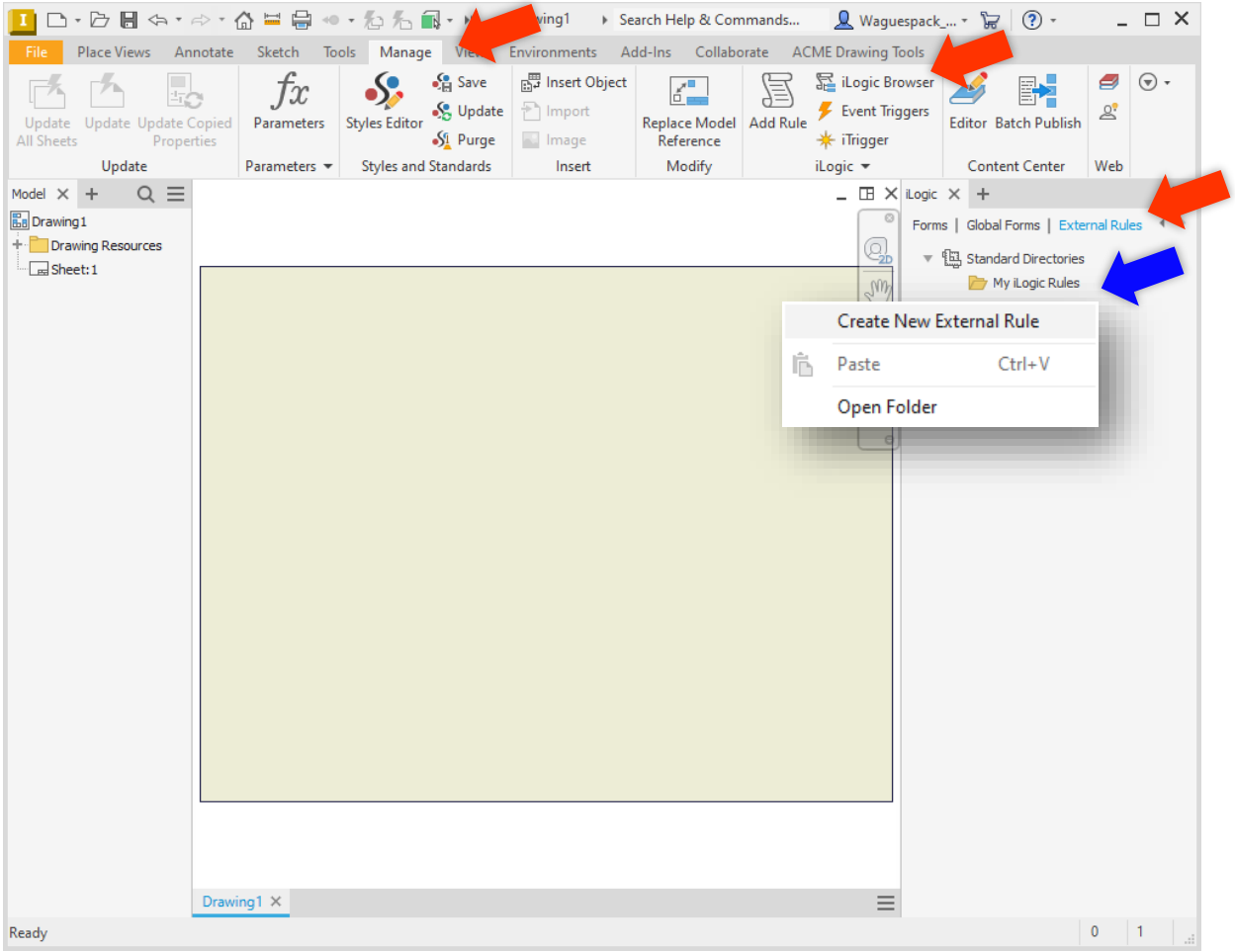

- Switch to the **Tools** tab and click the **Options** button
- Click the **iLogic Configuration** button as shown

# Using the Add-in



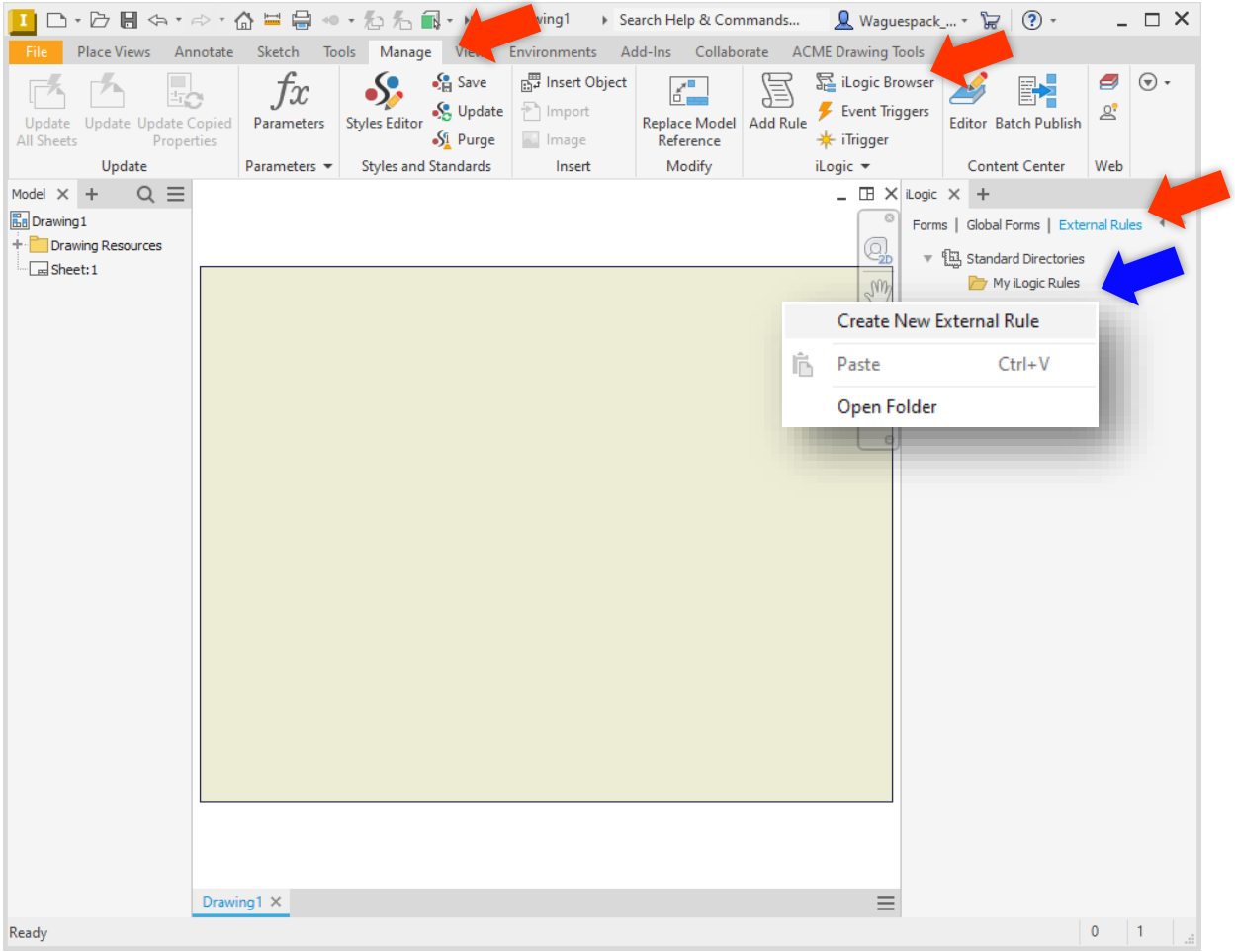

- If you have an External Rule directory already specified you can just use it
- If not use the **+** button to add a new path of your choice
- Click the **OK** button when done

# Using the Add-in



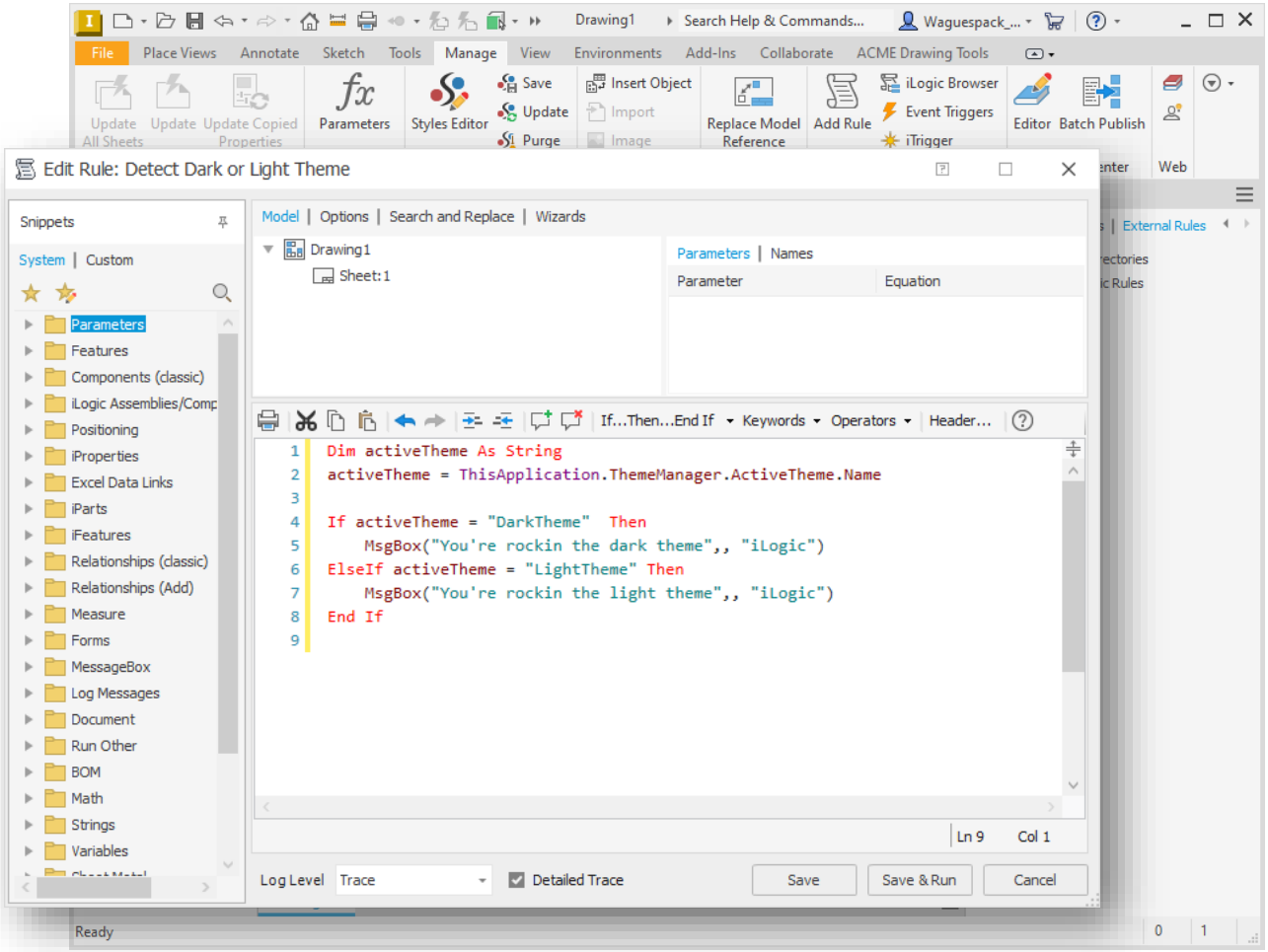

- On the **Manage** tab, click the **iLogic Browser** button
- Switch to the External Rules tab and note the folder you specified in the iLogic Configuration
- Right click on the folder and choose **Create New External Rule**

# Using the Add-in



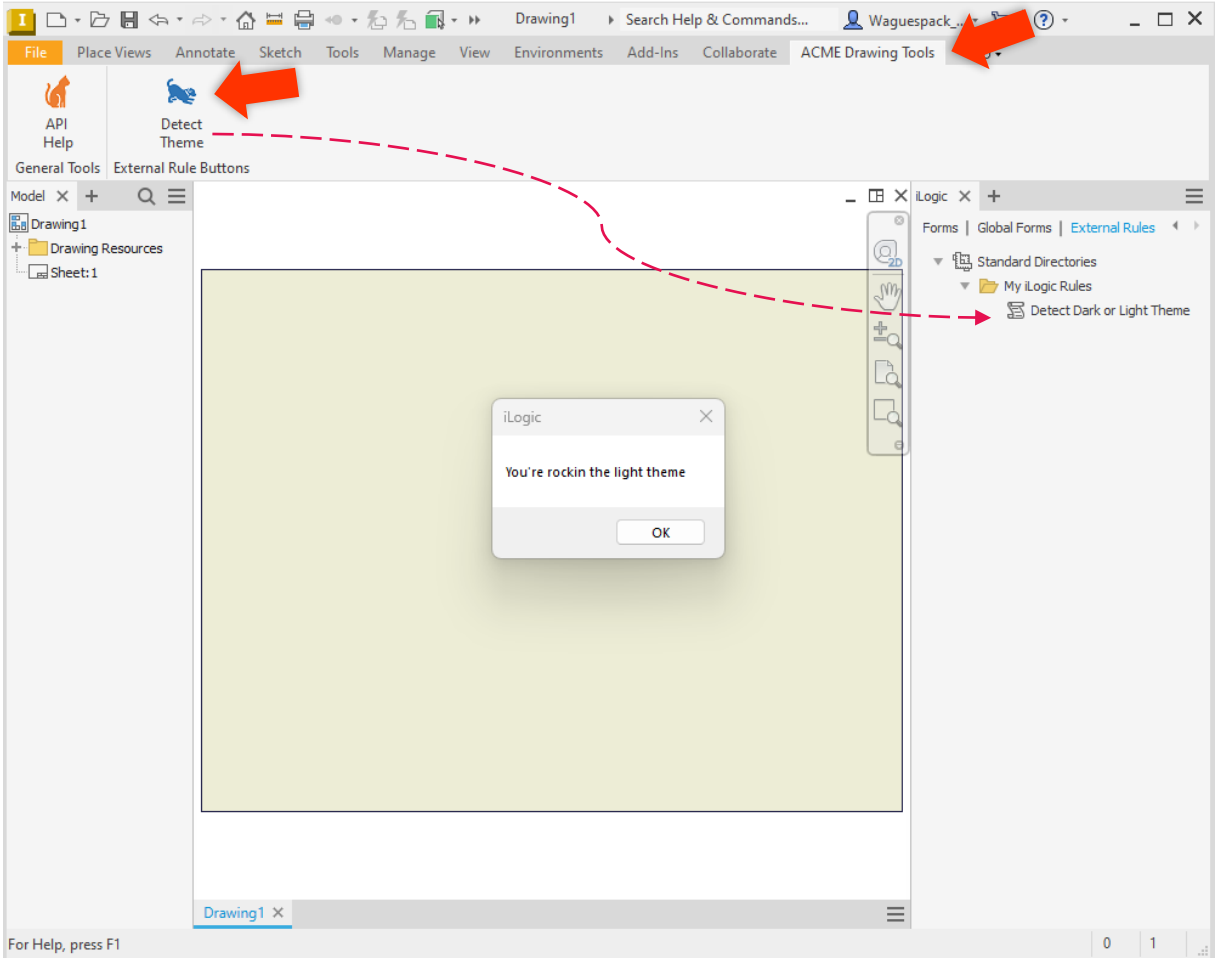

- On the **Manage** tab, click the **iLogic Browser** button
- Switch to the External Rules tab and note the folder you specified in the iLogic Configuration
- Right click on the folder and choose **Create New External Rule**

# Using the Add-in



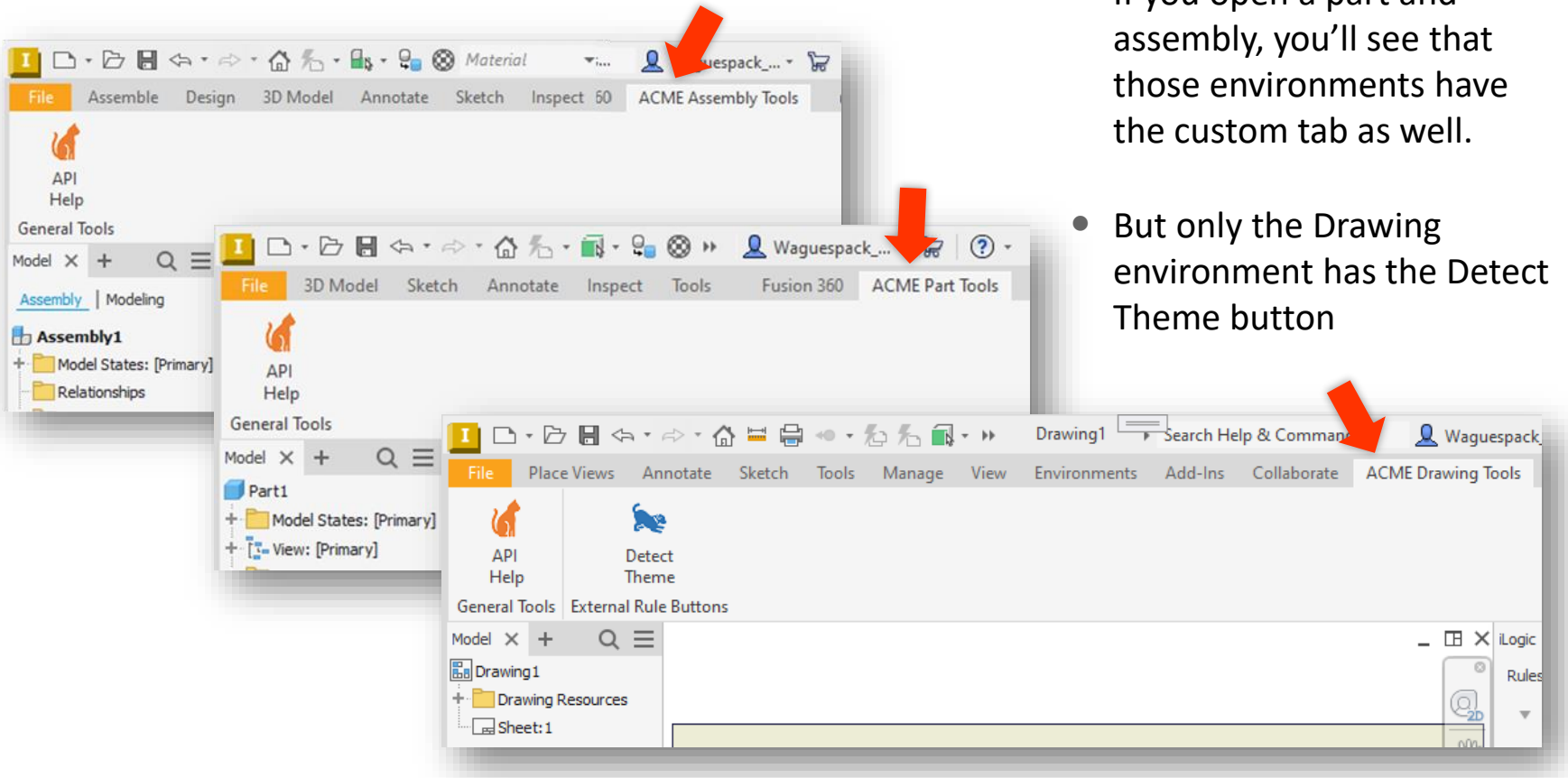

- Name the rule **Detect Dark or**
- Enter the code as shown
- Click the **Save** button, then the **Close** button
- Note that **this rule is available in the downloaded materials** if you'd prefer just to copy it.

# Using the Add-in




- Switch back to the **ACME Drawing Tools** tab
- Click the **Detect Theme** button
- Note that the add-in button runs the external iLogic rule

# Using the Add-in



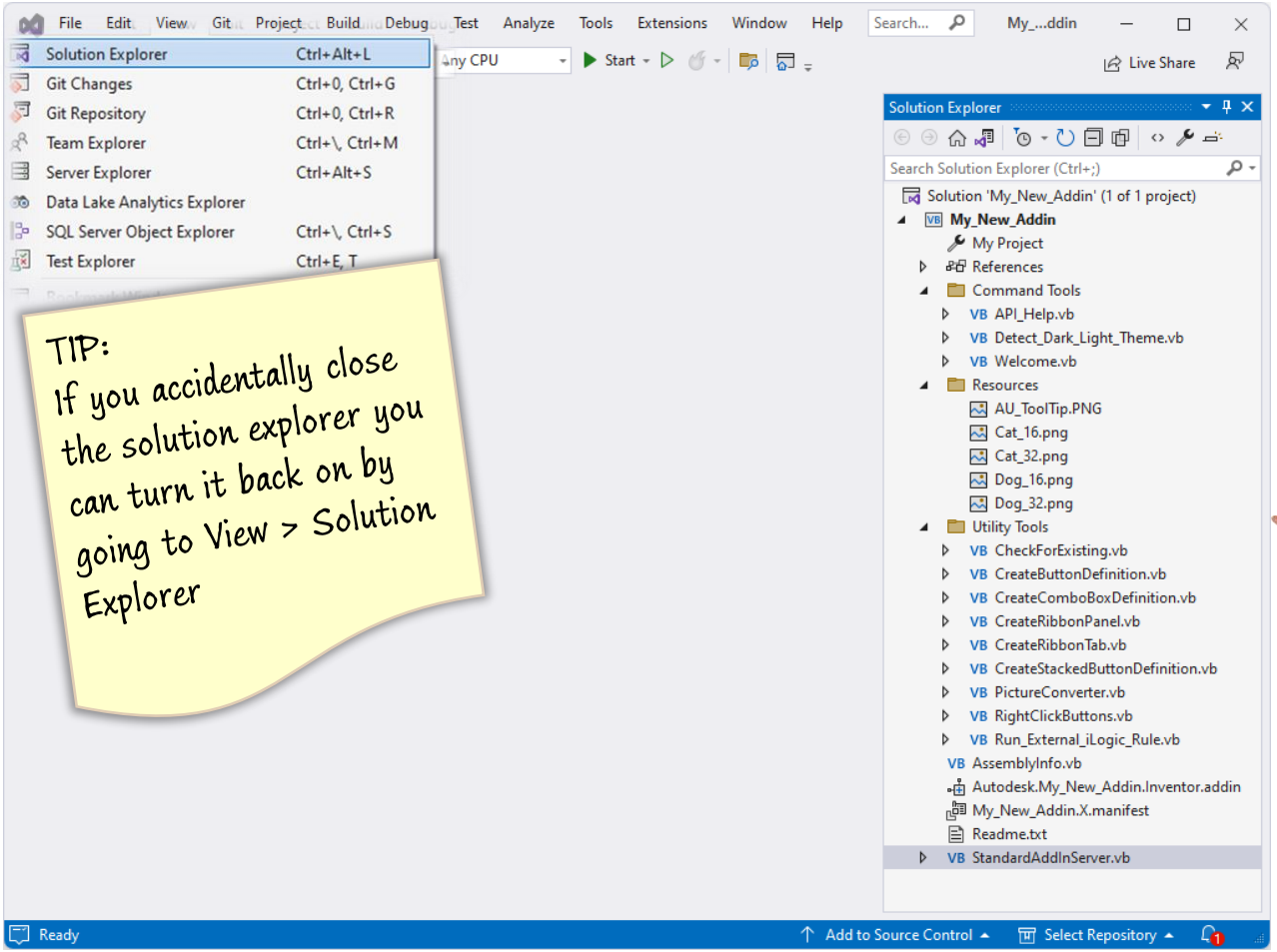

- If you open a part and assembly, you'll see that those environments have the custom tab as well.
- But only the Drawing environment has the Detect Theme button

AU 2023

# Understanding the Add-in Template Structure

Working in Visual Studio

# Understanding the Add-in Template Structure




TIP:
If you accidentally close the solution explorer you can turn it back on by going to View > Solution Explorer

- Back in Visual Studio, let's examine the file/folder structure of the template soluiton

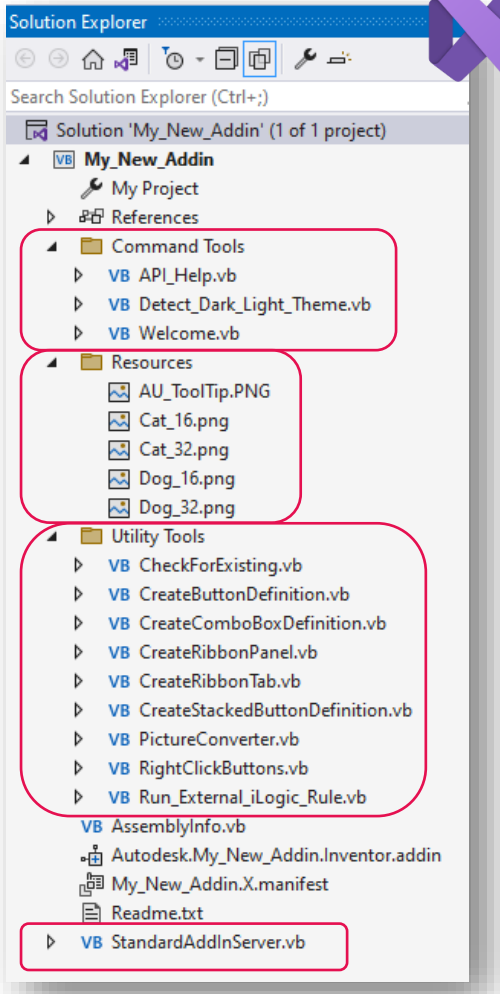
# Understanding the Add-in Template Structure

Command Tool Modules →

Resources →

Utility Modules →

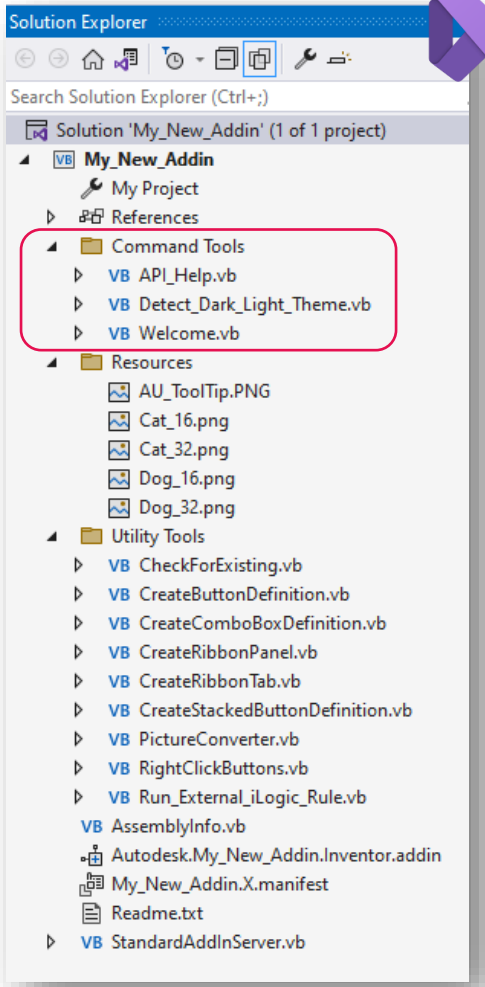
Standard Add-in Server →



# Understanding the Add-in Template Structure

- One module for each button.
  - Function to define the button
  - Sub to house the code that is run when the button is clicked

Command Tool Modules →



# Understanding the Add-in Template Structure

API_Help.vb* | My_New_Addin | API_Help | CreateButton

```
1   Imports Inventor
2
    4 references
3   Module API_Help
4
5       ''' <summary> Creates a button Button Definition Define the label, icons, tool t ...
13
14      'This function is where the button is defined
        3 references
15      Function CreateButton(environment As String, CustomDrawingTab As RibbonTab, ribbonPanel As RibbonPanel,
16                            useLargeIcon As Boolean, isInButtonStack As Boolean) As ButtonDefinition
17
18          'get the images to use for the button
19          Dim largeIcon As IPictureDisp = PictureDispConverter.ToIPictureDisp(My.Resources.Cat_32)
20          Dim standardIcon As IPictureDisp = PictureDispConverter.ToIPictureDisp(My.Resources.Cat_16)
21          Dim toolTipImage As IPictureDisp = Nothing
22
23          'this is the text the user sees on the button
24          Dim buttonLabel As String = "API " & vbLf & "Help"
25
26          'text that displays when the user hovers over the button
27          Dim toolTip_Simple As String = "Opens the API Help"
28          Dim toolTip_Expanded As String = Nothing
29
30
31      Progressive ToolTip
47
48          Dim buttonDef As ButtonDefinition
49          buttonDef = CreateButtonDefintion.CreateButtonDef(environment, CustomDrawingTab, ribbonPanel, useLa
50                                                            isInButtonStack, useProgressToolTip,
51                                                            buttonLabel, toolTip_Simple, toolTip_Expanc
52                                                            standardIcon, largeIcon, toolTipImage)
53
54          Return buttonDef
55
56      End Function
77
78  End Module
79
```

- One module for each button.
  - Function to define the button
  - Sub to house the code that is run when the button is clicked

# Understanding the Add-in Template Structure

API_Help.vb* | My_New_Addin | API_Help | CreateButton

```vb
Imports Inventor

4 references
Module API_Help

    ''' <summary> Creates a button Button Definition Define the label, icons, tool t ...

    'This function is where the button is defined
    3 references
    Function CreateButton(environment As String, CustomDrawingTab As RibbonTab, ribbonPanel As RibbonPanel,
                          useLargeIcon As Boolean, isInButtonStack As Boolean) As ButtonDefinition

        'get the images to use for the button
        Dim largeIcon As IPictureDisp = PictureDispConverter.ToIPictureDisp(My.Resources.Cat_32)
        Dim standardIcon As IPictureDisp = PictureDispConverter.ToIPictureDisp(My.Resources.Cat_16)
        Dim toolTipImage As IPictureDisp = Nothing

        'this is the text the user sees on the button
        Dim buttonLabel As String = "API " & vbLf & "Help"

        'text that displays when the user hovers over the button
        Dim toolTip_Simple As String = "Opens the API Help"
        Dim toolTip_Expanded As String = Nothing


    Progressive ToolTip

        Dim buttonDef As ButtonDefinition
        buttonDef = CreateButtonDefintion.CreateButtonDef(environment, CustomDrawingTab, ribbonPanel, useLargeIcon,
                                                          isInButtonStack, useProgressToolTip,
                                                          buttonLabel, toolTip_Simple, toolTip_Expanded,
                                                          standardIcon, largeIcon, toolTipImage)

        Return buttonDef

    End Function


End Module
```

- One module for each button.
  - Function to define the button
  - Sub to house the code that is run when the button is clicked

# Understanding the Add-in Template Structure

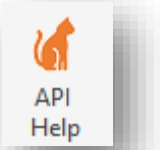
API_Help.vb*

My_New_Addin | API_Help | CreateButton

```
1   Imports Inventor
2
    4 references
3   Module API_Help
4
5       ''' <summary> Creates a button Button Definition Define the label, icons, tool t ...
13
14      'This function is where the button is defined
        3 references
15      Function CreateButton(environment As String, CustomDrawingTab As RibbonTab, ribbonPanel As RibbonPanel,
16                            useLargeIcon As Boolean, isInButtonStack As Boolean) As ButtonDefinition
17
18          'get the images to use for the button
19          Dim largeIcon As IPictureDisp = PictureDispConverter.ToIPictureDisp(My.Resources.Cat_32)
20          Dim standardIcon As IPictureDisp = PictureDispConverter.ToIPictureDisp(My.Resources.Cat_16)
21          Dim toolTipImage As IPictureDisp = Nothing
22
23          'this is the text the user sees on the button
24          Dim buttonLabel As String = "API " & vbLf & "Help"
25
26          'text that displays when the user hovers over the button
27          Dim toolTip_Simple As String = "Opens the API Help"
28          Dim toolTip_Expanded As String = Nothing
29
30
31      Progressive ToolTip
47
48          Dim buttonDef As ButtonDefinition
49          buttonDef = CreateButtonDefintion.CreateButtonDef(environment, CustomDrawingTab, ribbonPanel, useLargeIcon,
50                                                            isInButtonStack, useProgressToolTip,
51                                                            buttonLabel, toolTip_Simple, toolTip_Expanded,
52                                                            standardIcon, largeIcon, toolTipImage)
53
54          Return buttonDef
55
56      End Function


77
78  End Module
79
```

Specify the icons



# Understanding the Add-in Template Structure

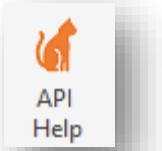
API_Help.vb*

My_New_Addin | API_Help | CreateButton

```
1   Imports Inventor
2
    4 references
3   Module API_Help
4
5       ''' <summary> Creates a button Button Definition Define the label, icons, tool t ...
13
14      'This function is where the button is defined
        3 references
15      Function CreateButton(environment As String, CustomDrawingTab As RibbonTab, ribbonPanel As RibbonPanel,
16                            useLargeIcon As Boolean, isInButtonStack As Boolean) As ButtonDefinition
17
18          'get the images to use for the button
19          Dim largeIcon As IPictureDisp = PictureDispConverter.ToIPictureDisp(My.Resources.Cat_32)
20          Dim standardIcon As IPictureDisp = PictureDispConverter.ToIPictureDisp(My.Resources.Cat_16)
21          Dim toolTipImage As IPictureDisp = Nothing
22
23          'this is the text the user sees on the button
24          Dim buttonLabel As String = "API " & vbLf & "Help"
25
26          'text that displays when the user hovers over the button
27          Dim toolTip_Simple As String = "Opens the API Help"
28          Dim toolTip_Expanded As String = Nothing
29
30
31      Progressive ToolTip
47
48          Dim buttonDef As ButtonDefinition
49          buttonDef = CreateButtonDefintion.CreateButtonDef(environment, CustomDrawingTab, ribbonPanel, useLargeIcon,
50                                                            isInButtonStack, useProgressToolTip,
51                                                            buttonLabel, toolTip_Simple, toolTip_Expanded,
52                                                            standardIcon, largeIcon, toolTipImage)
53
54          Return buttonDef
55
56      End Function


77
78  End Module
79
```

Specify the Button Display Name



# Understanding the Add-in Template Structure

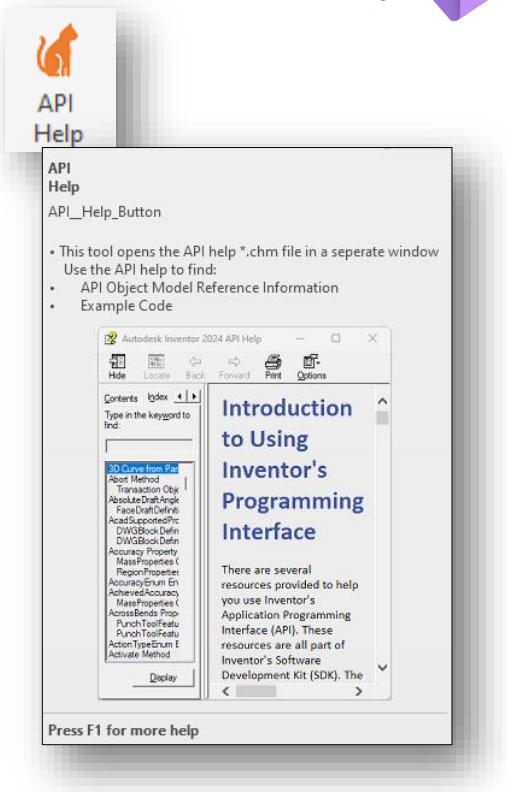
API_Help.vb*

My_New_Addin | API_Help | CreateButton

```
1   Imports Inventor
2
    4 references
3   Module API_Help
4
5       ''' <summary> Creates a button Button Definition Define the label, icons, tool t ...
13
14      'This function is where the button is defined
        3 references
15      Function CreateButton(environment As String, CustomDrawingTab As RibbonTab, ribbonPanel As RibbonPanel,
16                            useLargeIcon As Boolean, isInButtonStack As Boolean) As ButtonDefinition
17
18          'get the images to use for the button
19          Dim largeIcon As IPictureDisp = PictureDispConverter.ToIPictureDisp(My.Resources.Cat_32)
20          Dim standardIcon As IPictureDisp = PictureDispConverter.ToIPictureDisp(My.Resources.Cat_16)
21          Dim toolTipImage As IPictureDisp = Nothing
22
23          'this is the text the user sees on the button
24          Dim buttonLabel As String = "API " & vbLf & "Help"
25
26          'text that displays when the user hovers over the button
27          Dim toolTip_Simple As String = "Opens the API Help"
28          Dim toolTip_Expanded As String = Nothing
29
30
31      Progressive ToolTip
47
48          Dim buttonDef As ButtonDefinition
49          buttonDef = CreateButtonDefintion.CreateButtonDef(environment, CustomDrawingTab, ribbonPanel, useLargeIcon,
50                                                            isInButtonStack, useProgressToolTip,
51                                                            buttonLabel, toolTip_Simple, toolTip_Expanded,
52                                                            standardIcon, largeIcon, toolTipImage)
53
54          Return buttonDef
55
56      End Function


77
78  End Module
79
```

Define the Tool Tip



# Understanding the Add-in Template Structure

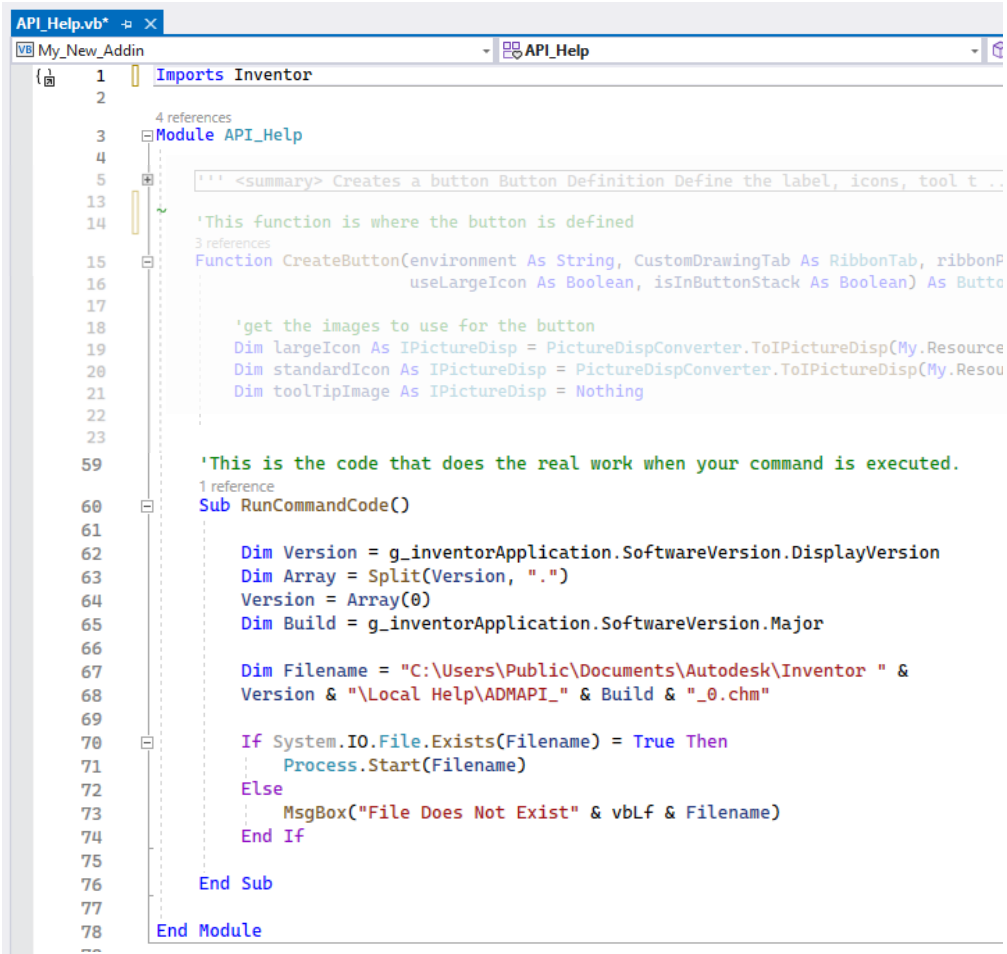
API_Help.vb* | My_New_Addin | API_Help | CreateButton

```
1   Imports Inventor
2
    4 references
3   Module API_Help
4
5       ''' <summary> Creates a button Button Definition Define the label, icons, tool t ...
13
14      'This function is where the button is defined
        3 references
15      Function CreateButton(environment As String, CustomDrawingTab As RibbonTab, ribbonPanel As RibbonPanel,
16                            useLargeIcon As Boolean, isInButtonStack As Boolean) As ButtonDefinition
17
18          'get the images to use for the button
19          Dim largeIcon As IPictureDisp = PictureDispConverter.ToIPictureDisp(My.Resources.Cat_32)
20          Dim standardIcon As IPictureDisp = PictureDispConverter.ToIPictureDisp(My.Resources.Cat_16)
21          Dim toolTipImage As IPictureDisp = Nothing
22
23          'this is the text the user sees on the button
24          Dim buttonLabel As String = "API " & vbLf & "Help"
25
26          'text that displays when the user hovers over the button
27          Dim toolTip_Simple As String = "Opens the API Help"
28          Dim toolTip_Expanded As String = Nothing
29
30
31      Progressive ToolTip
47
48          Dim buttonDef As ButtonDefinition
49          buttonDef = CreateButtonDefintion.CreateButtonDef(environment, CustomDrawingTab, ribbonPanel, useLargeIcon,
50                                                            isInButtonStack, useProgressToolTip,
51                                                            buttonLabel, toolTip_Simple, toolTip_Expanded,
52                                                            standardIcon, largeIcon, toolTipImage)
53
54          Return buttonDef
55
56      End Function


77
78  End Module
79
```

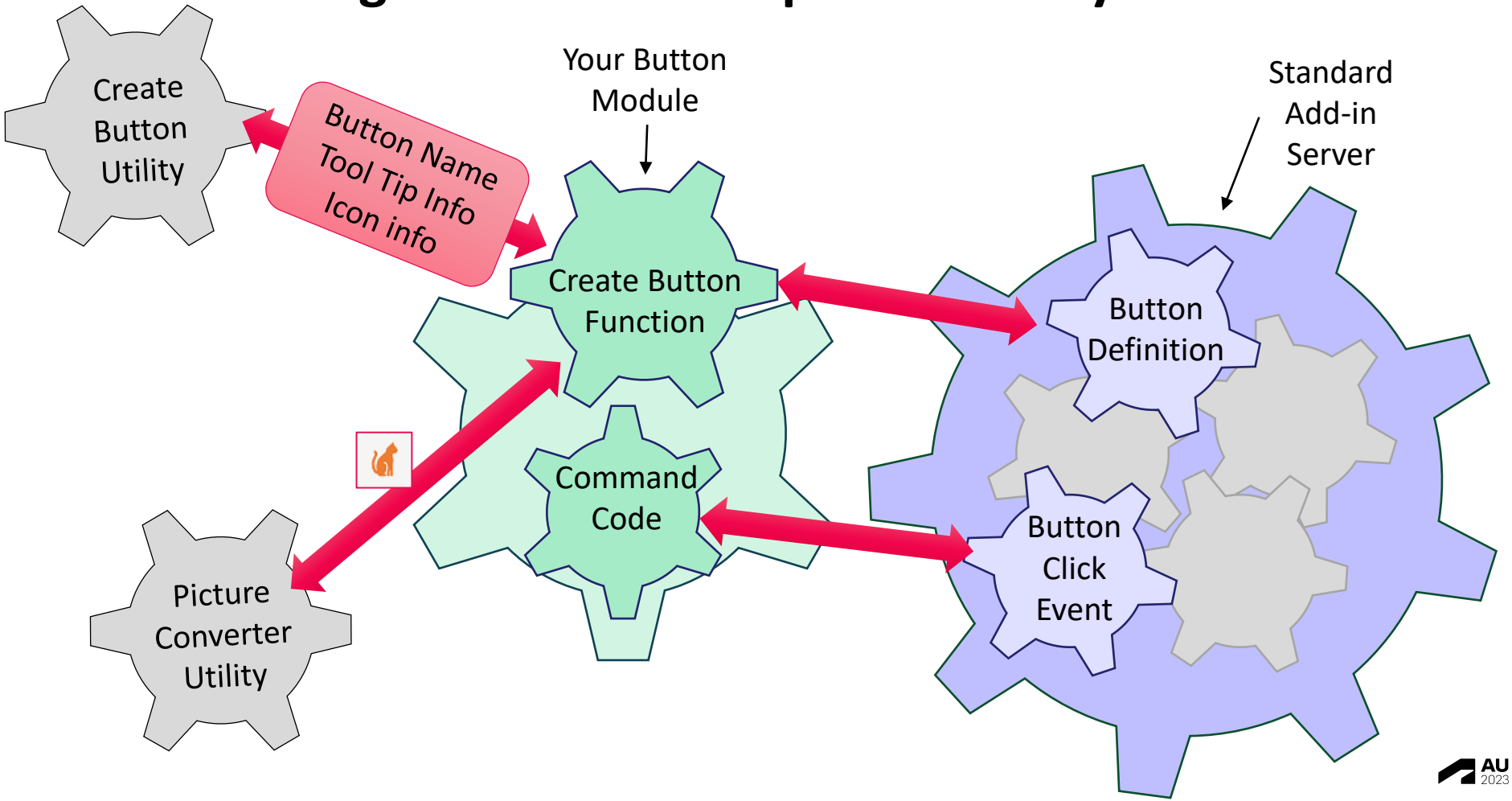
Pass the information to the utility module

# Understanding the Add-in Template Structure

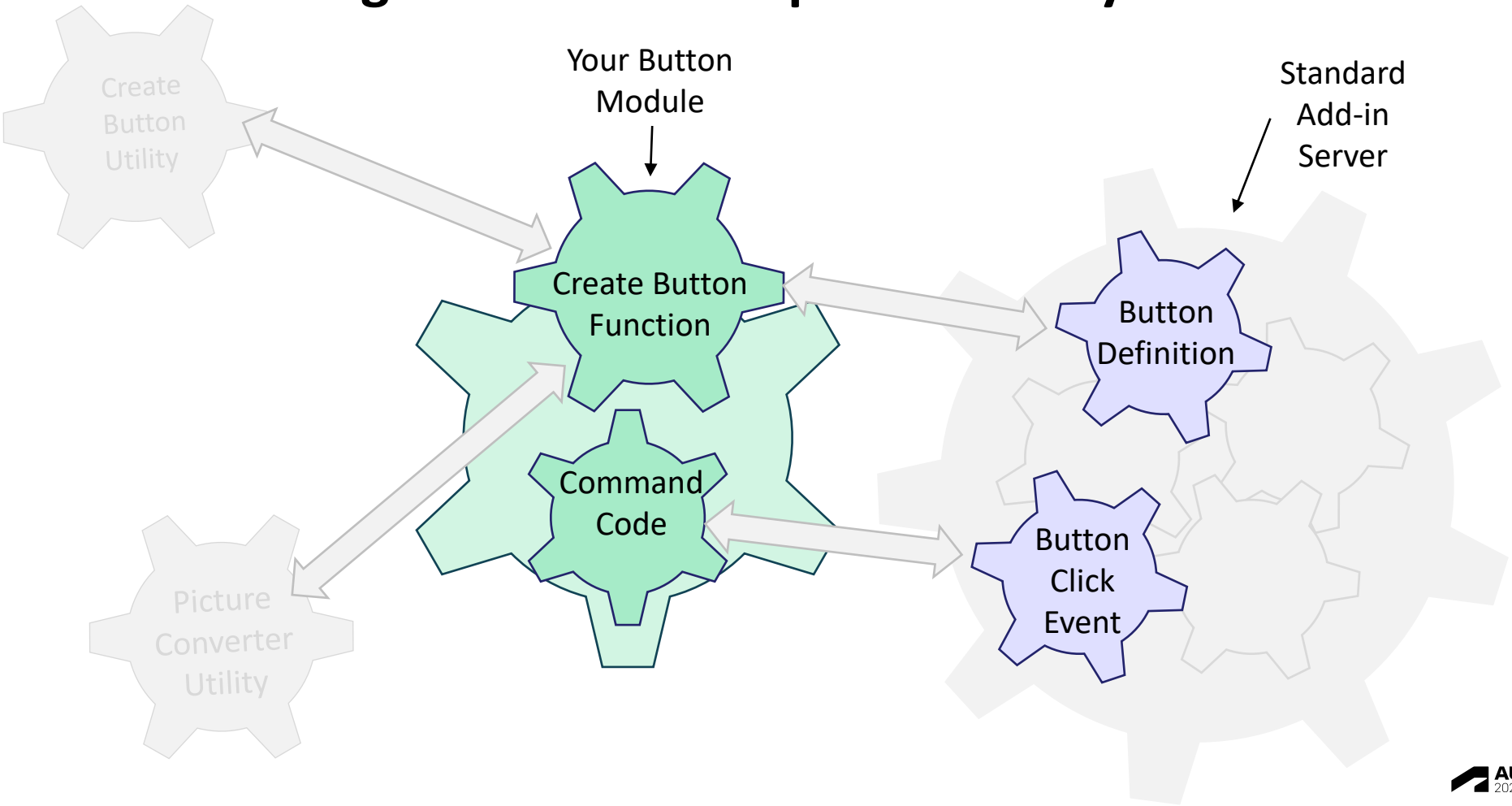



- One module for each button.
  - Function to define the button
  - Sub to house the code that is run when the button is clicked

# Understanding the Add-in Template Visually

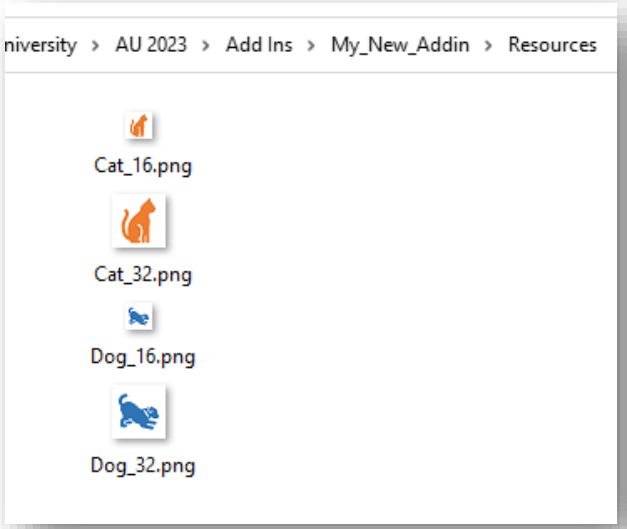


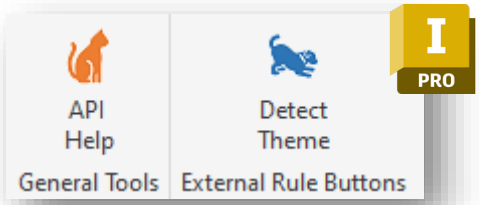
# Understanding the Add-in Template Visually



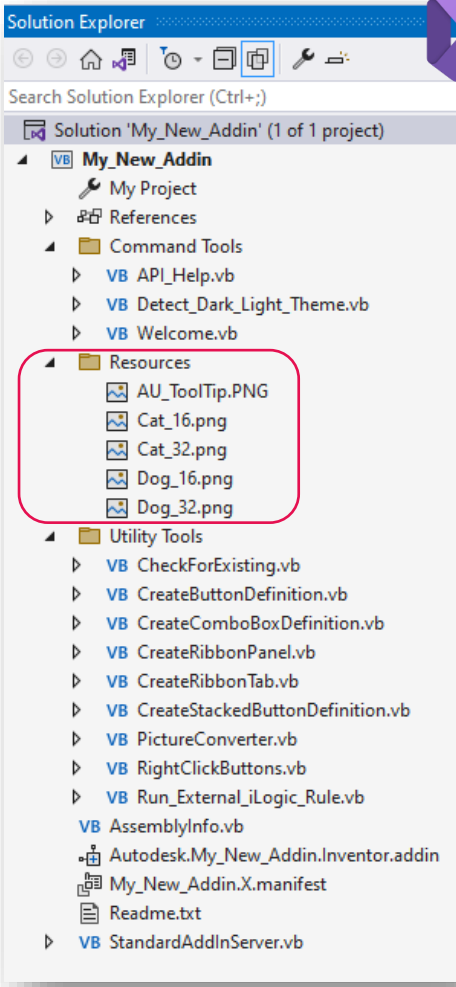
# Understanding the Add-in Template Structure

- A folder that holds the pictures used for the button icons




Resources →



# Adding a button image

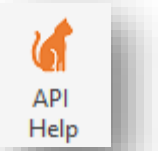
API_Help.vb*

My_New_Addin | API_Help | CreateButton

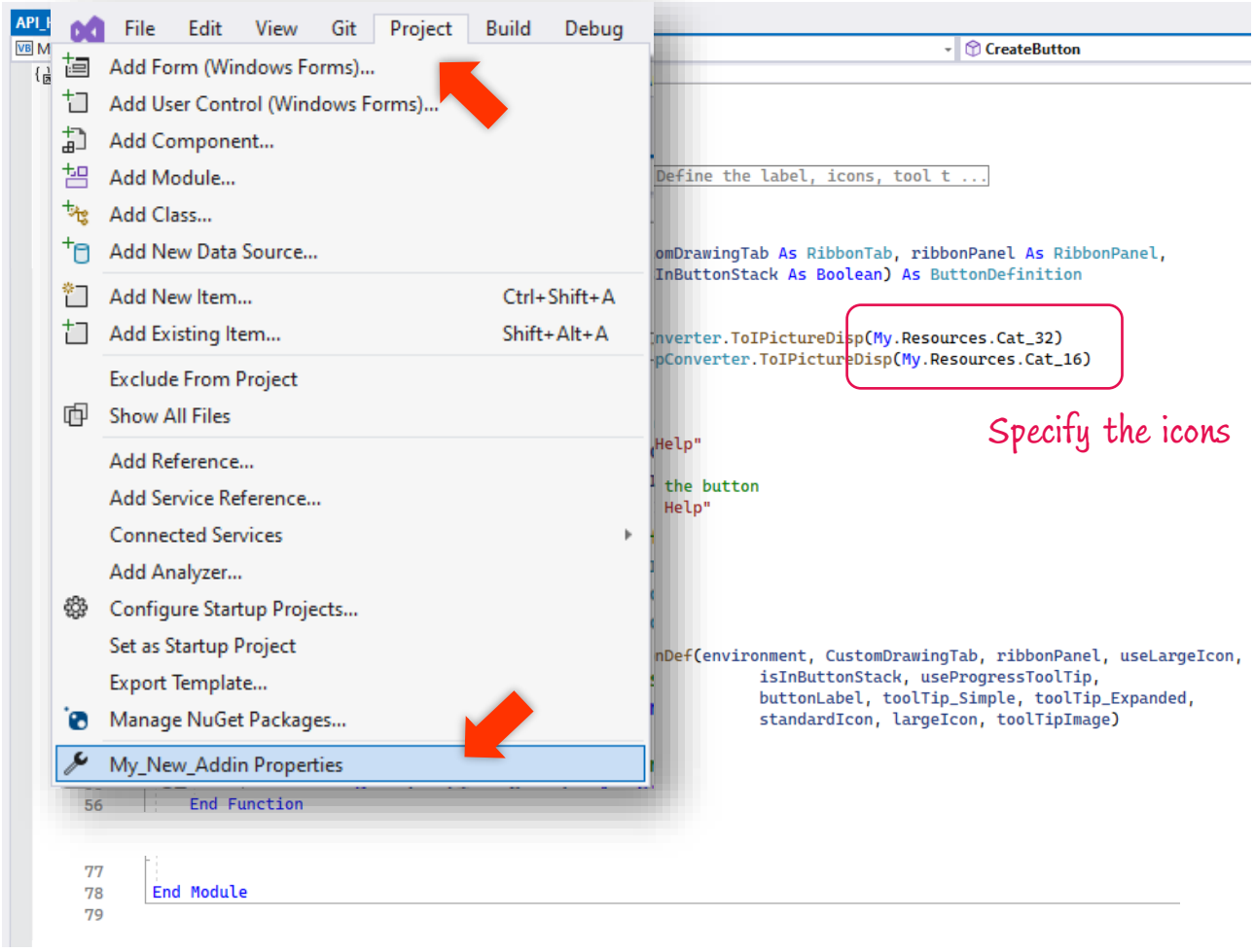
```
1   Imports Inventor
2
    4 references
3   Module API_Help
4
5       ''' <summary> Creates a button Button Definition Define the label, icons, tool t ...
13
14      'This function is where the button is defined
        3 references
15      Function CreateButton(environment As String, CustomDrawingTab As RibbonTab, ribbonPanel As RibbonPanel,
16                            useLargeIcon As Boolean, isInButtonStack As Boolean) As ButtonDefinition
17
18          'get the images to use for the button
19          Dim largeIcon As IPictureDisp = PictureDispConverter.ToIPictureDisp(My.Resources.Cat_32)
20          Dim standardIcon As IPictureDisp = PictureDispConverter.ToIPictureDisp(My.Resources.Cat_16)
21          Dim toolTipImage As IPictureDisp = Nothing
22
23          'this is the text the user sees on the button
24          Dim buttonLabel As String = "API " & vbLf & "Help"
25
26          'text that displays when the user hovers over the button
27          Dim toolTip_Simple As String = "Opens the API Help"
28          Dim toolTip_Expanded As String = Nothing
29
30
31      Progressive ToolTip
47
48          Dim buttonDef As ButtonDefinition
49          buttonDef = CreateButtonDefintion.CreateButtonDef(environment, CustomDrawingTab, ribbonPanel, useLargeIcon,
50                                                            isInButtonStack, useProgressToolTip,
51                                                            buttonLabel, toolTip_Simple, toolTip_Expanded,
52                                                            standardIcon, largeIcon, toolTipImage)
53
54          Return buttonDef
55
56      End Function


77
78  End Module
79
```

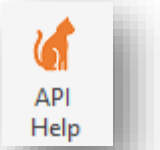
Specify the icons



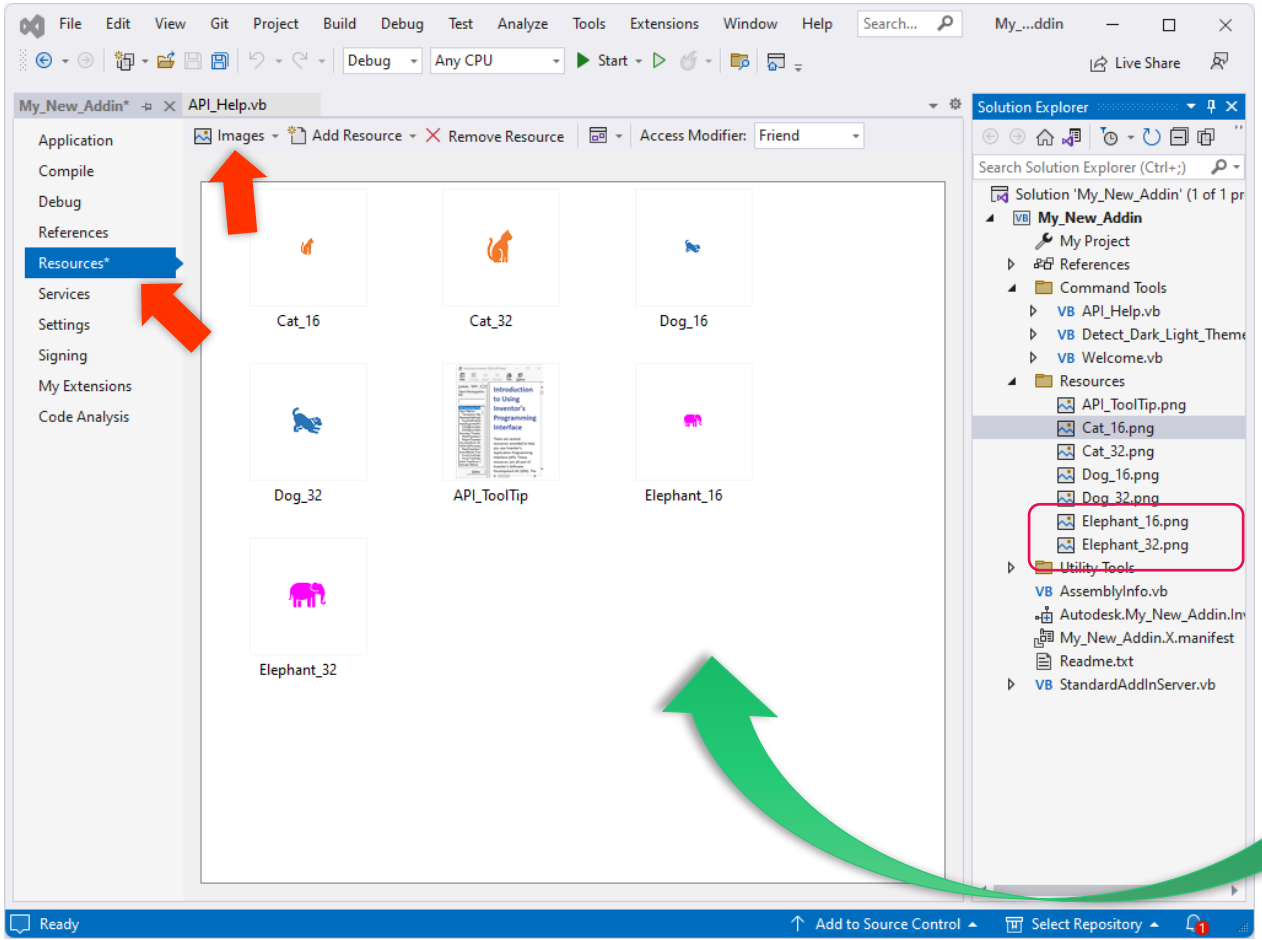
# Adding a button image



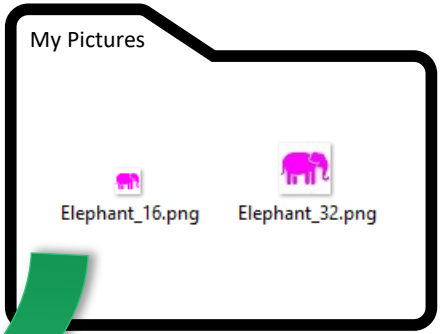

Specify the icons



# Understanding the Add-in Template Structure




To add images, we simply drag and drop the images on the Resources pane



# Understanding the Add-in Template Structure

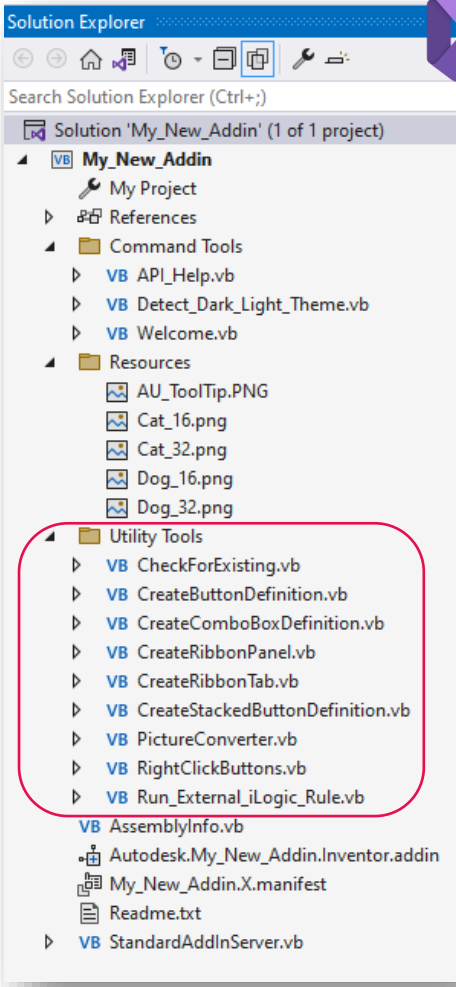
- Modules that are called over and over from the Command Tools and the StandardAddInServer

```
Dim buttonDef As ButtonDefinition
buttonDef = CreateButtonDefintion.CreateButtonDef(environment, CustomDrawingTab, ribbonPanel, useLargeIcon,
                                                  isInButtonStack, useProgressToolTip,
                                                  buttonLabel, toolTip_Simple, toolTip_Expanded,
                                                  standardIcon, largeIcon, toolTipImage)
```

```
'get the images to use for the button
Dim largeIcon As IPictureDisp = PictureDispConverter.ToIPictureDisp(My.Resources.Cat_32)
Dim standardIcon As IPictureDisp = PictureDispConverter.ToIPictureDisp(My.Resources.Cat_16)
```

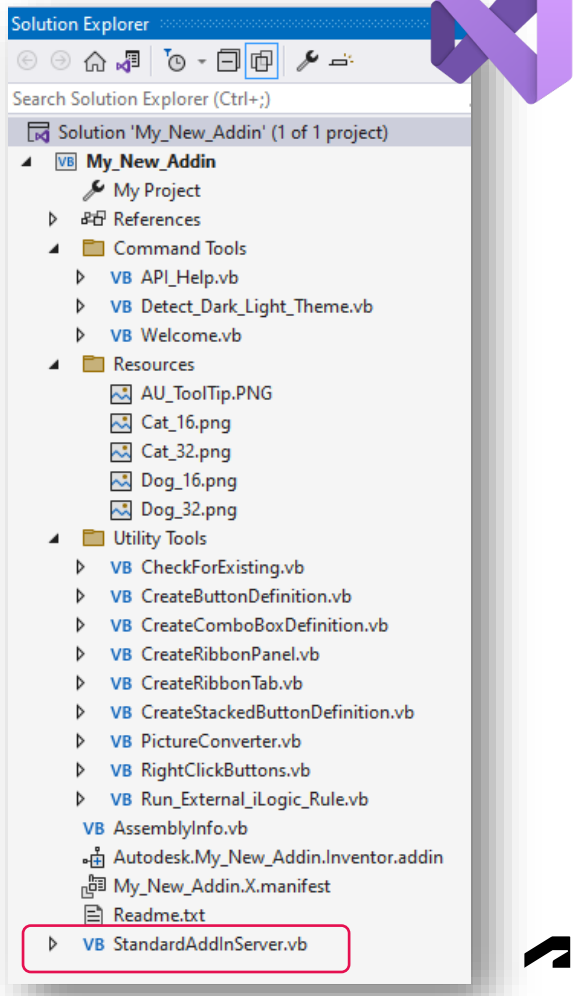
```
Run_External_iLogic_Rule.RunExternalRule("Detect Dark or Light Theme")
```

Utility Modules →



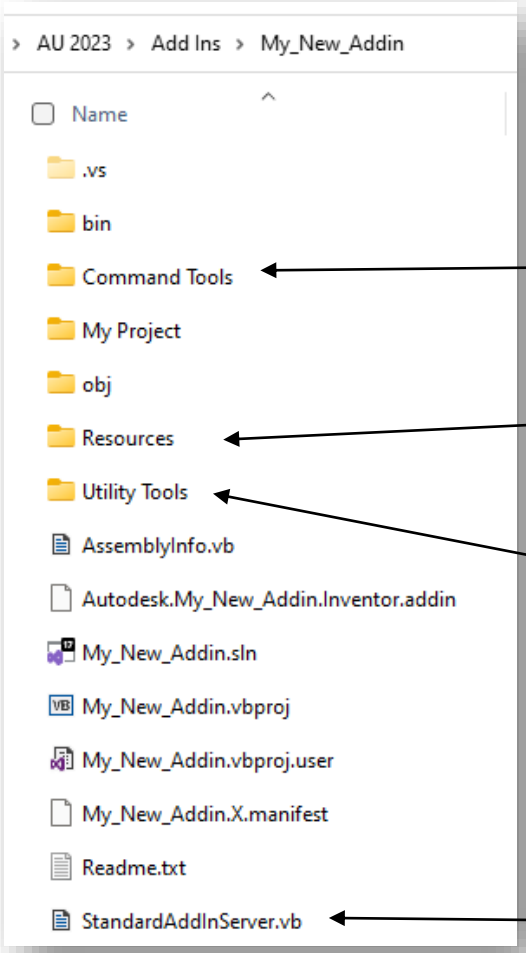
# Understanding the Add-in Template Structure

- The primary file that defines the add-in
- This is what Inventor calls when it loads the add-in
- This is what connects our command modules to Inventor
- This is what handles the events:
  - Document OnOpen
  - Document OnSave
  - Button OnExecute
  - … and so on.




Standard Add-in Server →

# Understanding the Add-in Template Structure

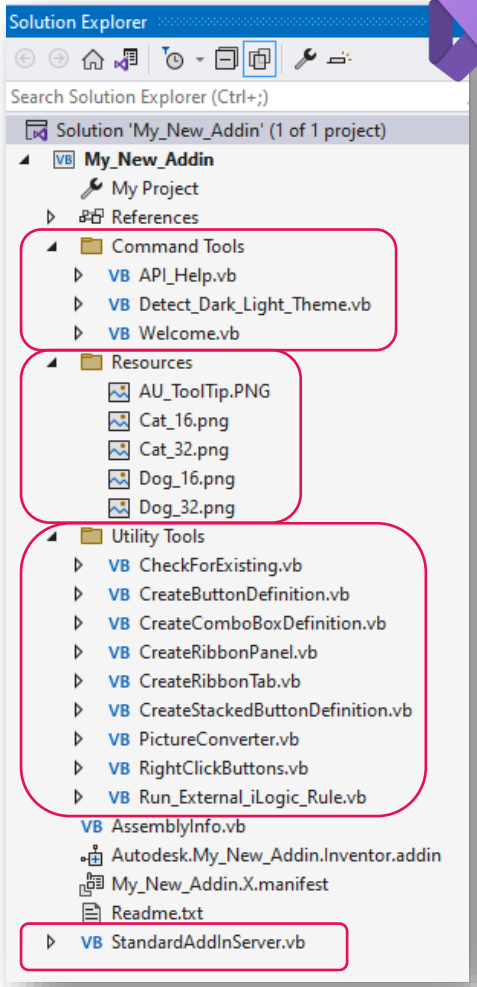



Command Tool Modules

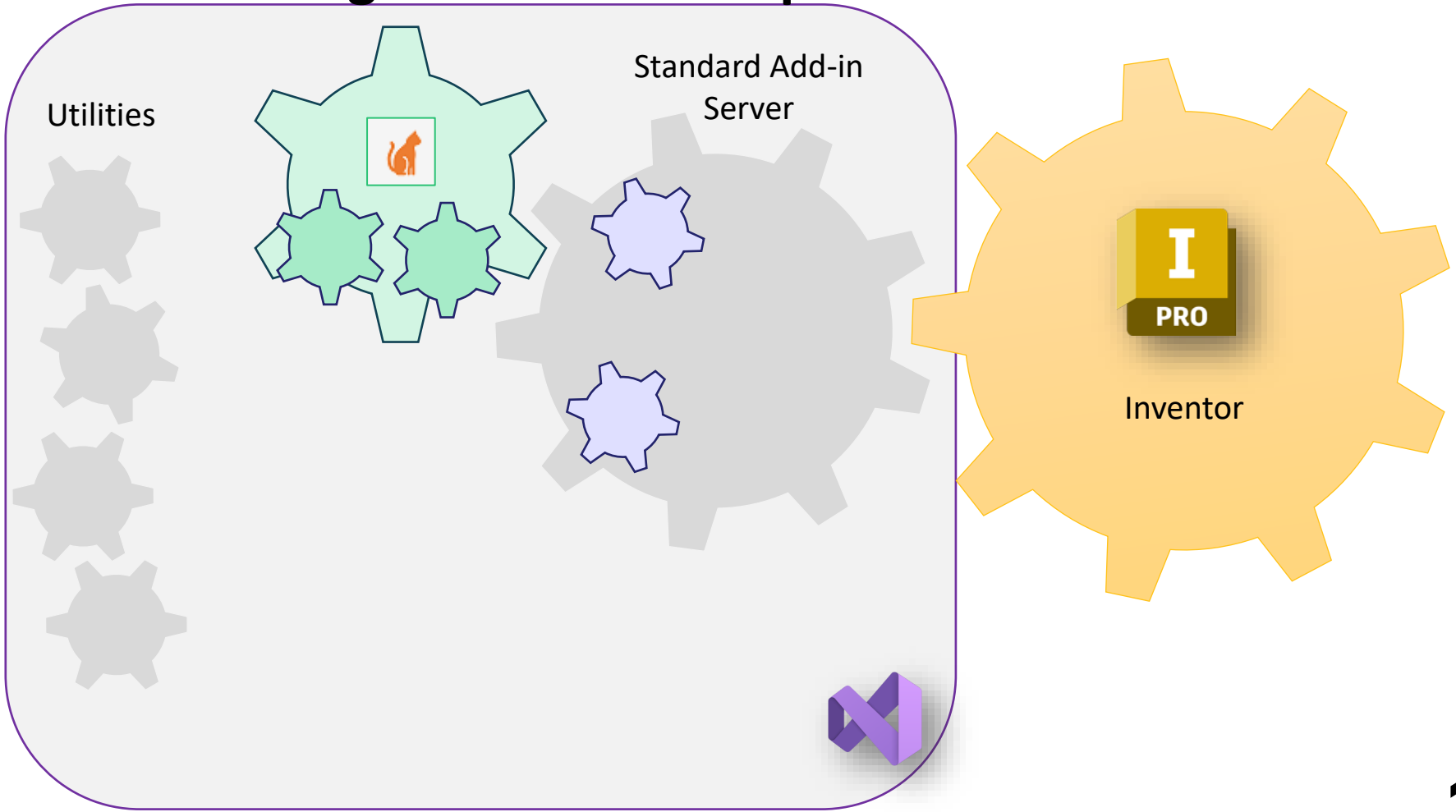
Resources

Utility Modules

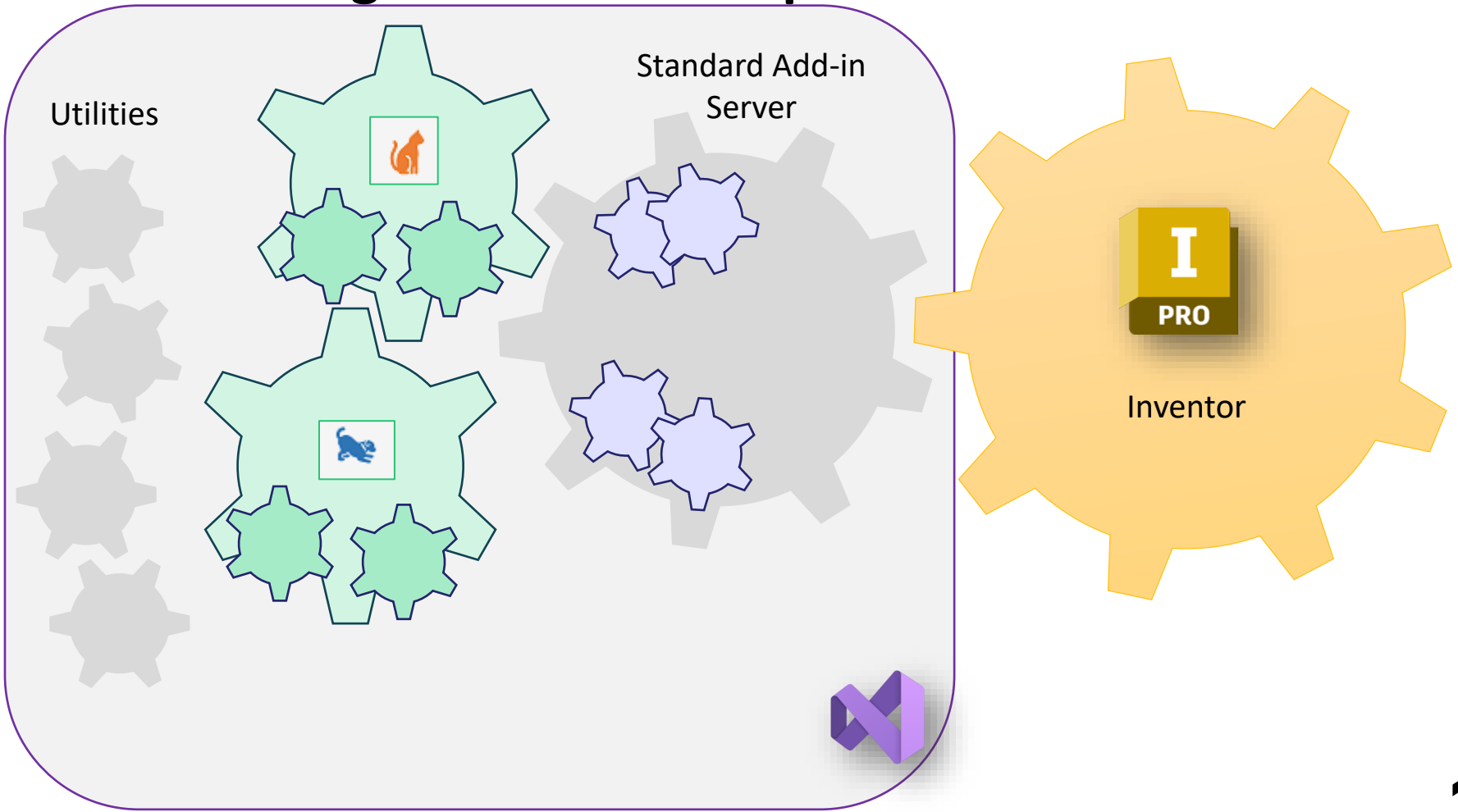
Standard Add-in Server



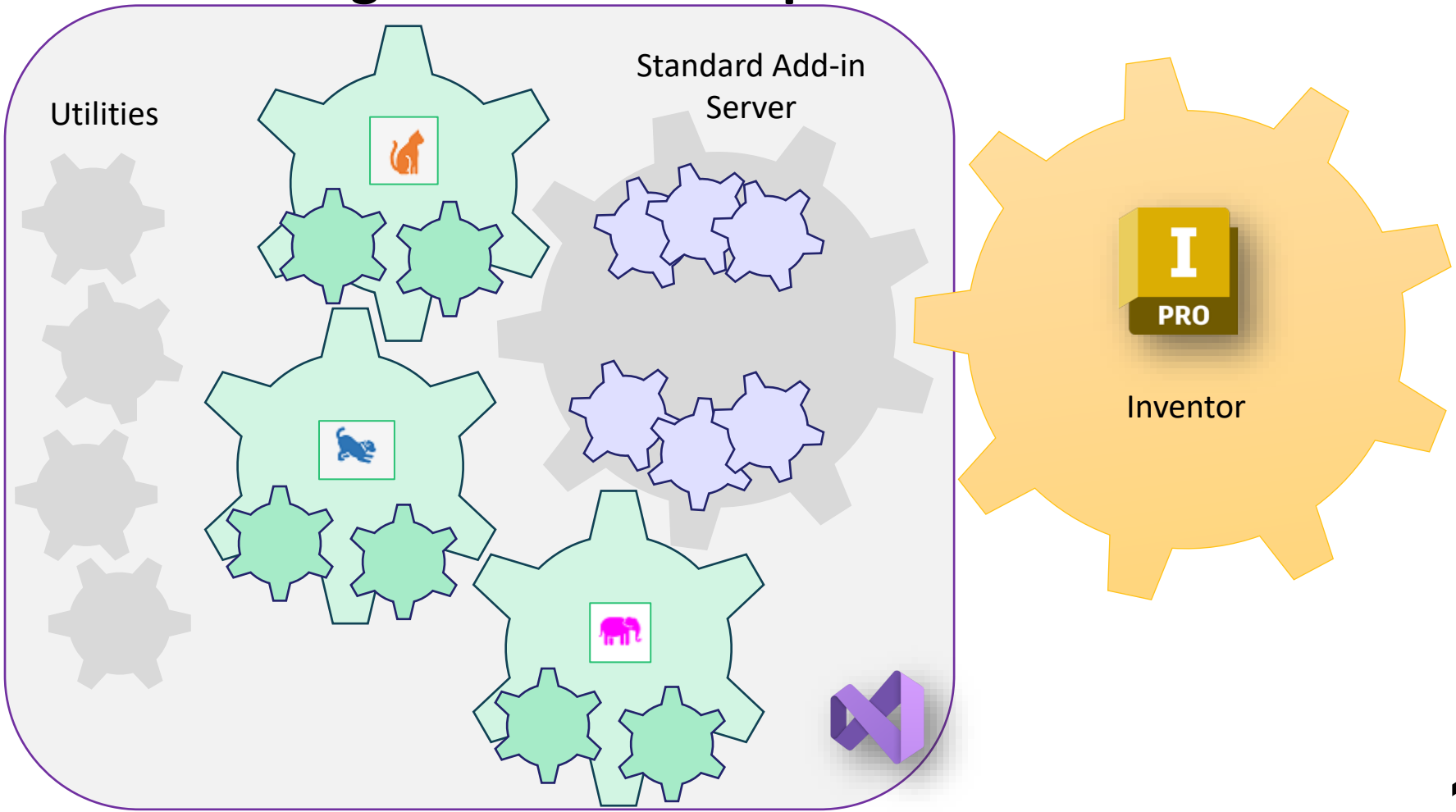
# Understanding the Add-in Template Structure



# Understanding the Add-in Template Structure



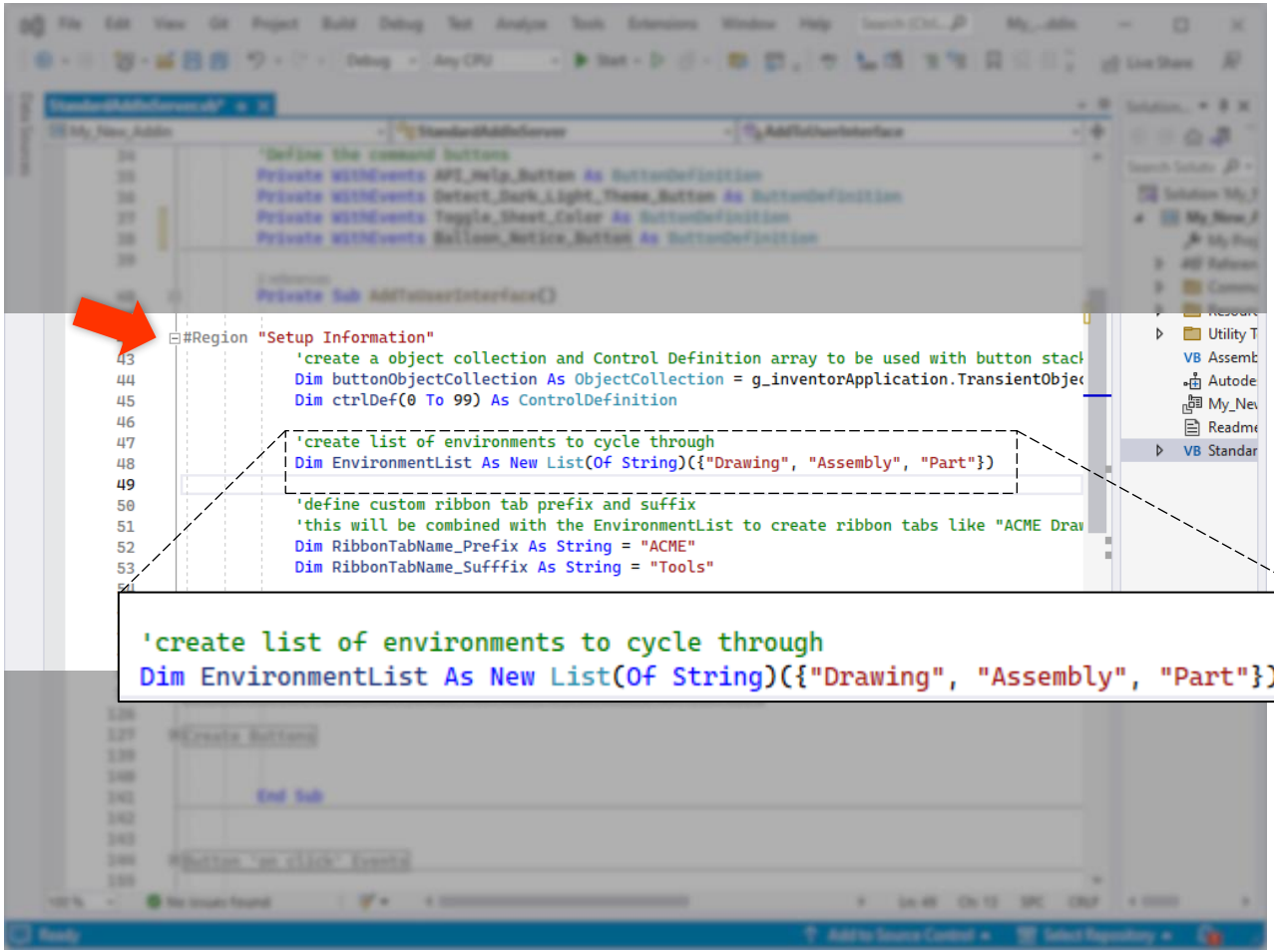
# Understanding the Add-in Template Structure



# Modifying the Add-in

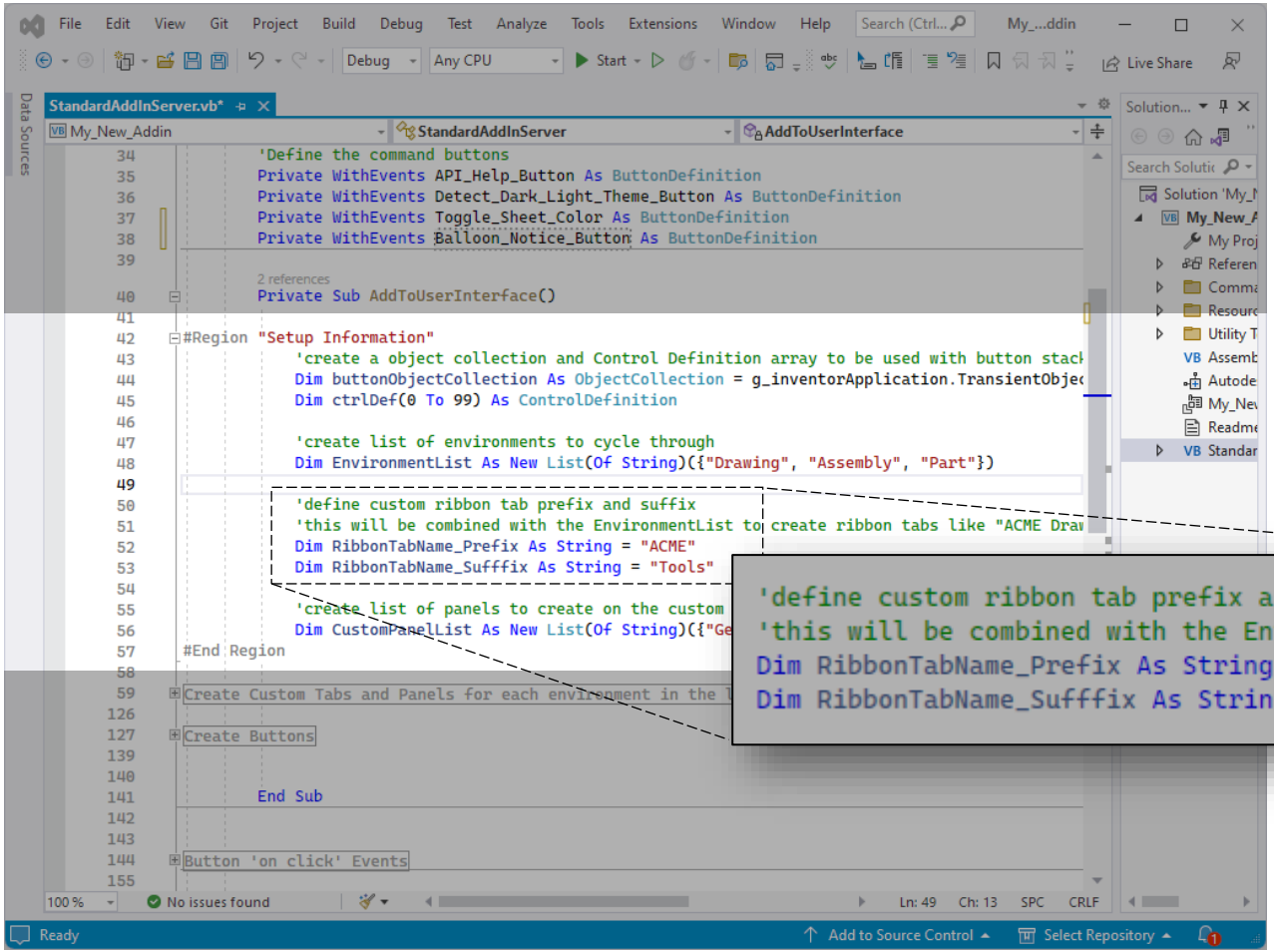
Working in Visual Studio

# Setup Information



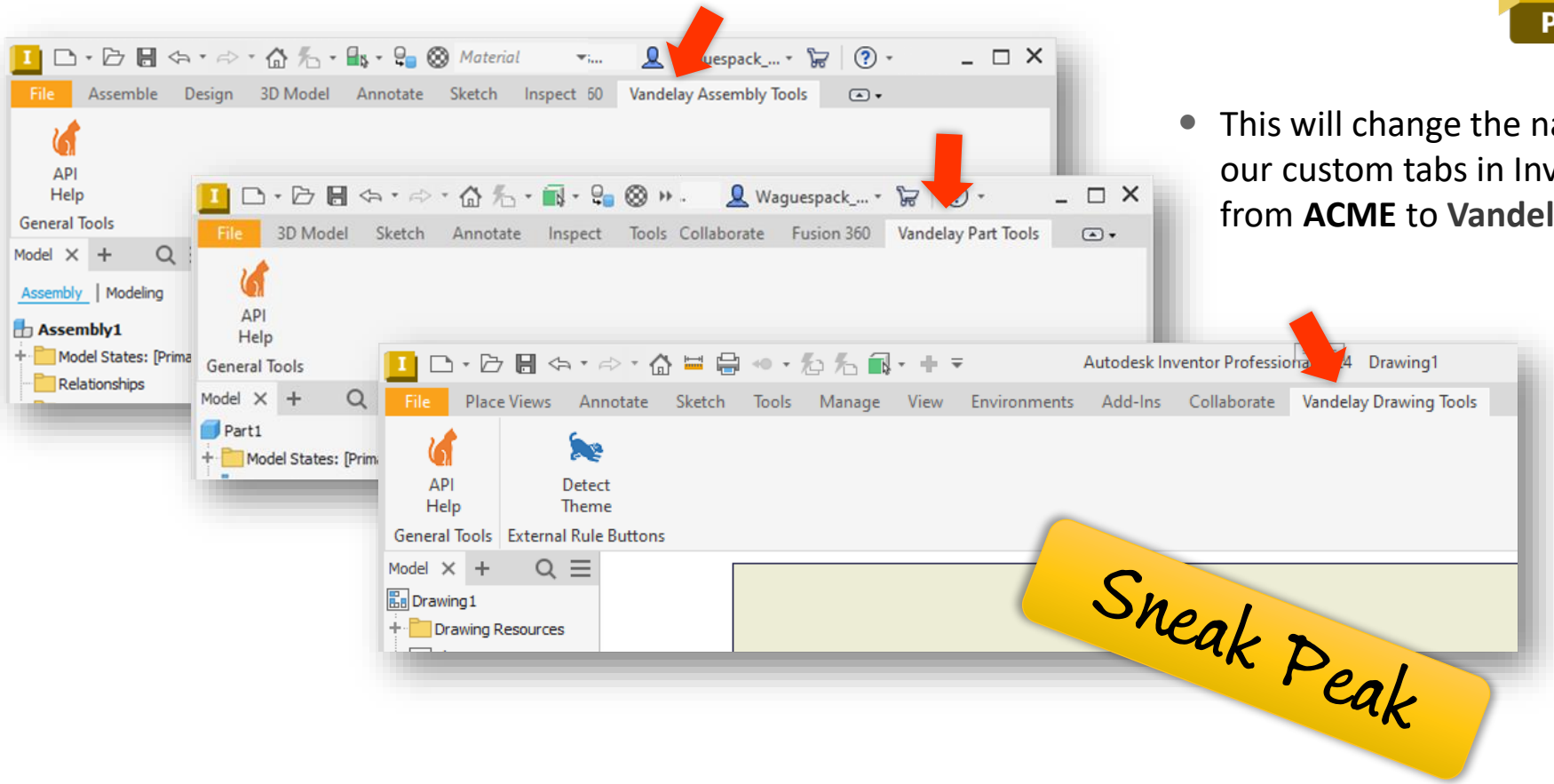

- We double click the **StandardAddinServer** file in the Solution Explorere to open it for edits.
- If we expand the **Setup Information** region, we can see where the custom tabs come from
- If we wanted the custom tab in only the Drawing and Part environments, we could modify the EnvironmentList and remove Assembly

# Setup Information



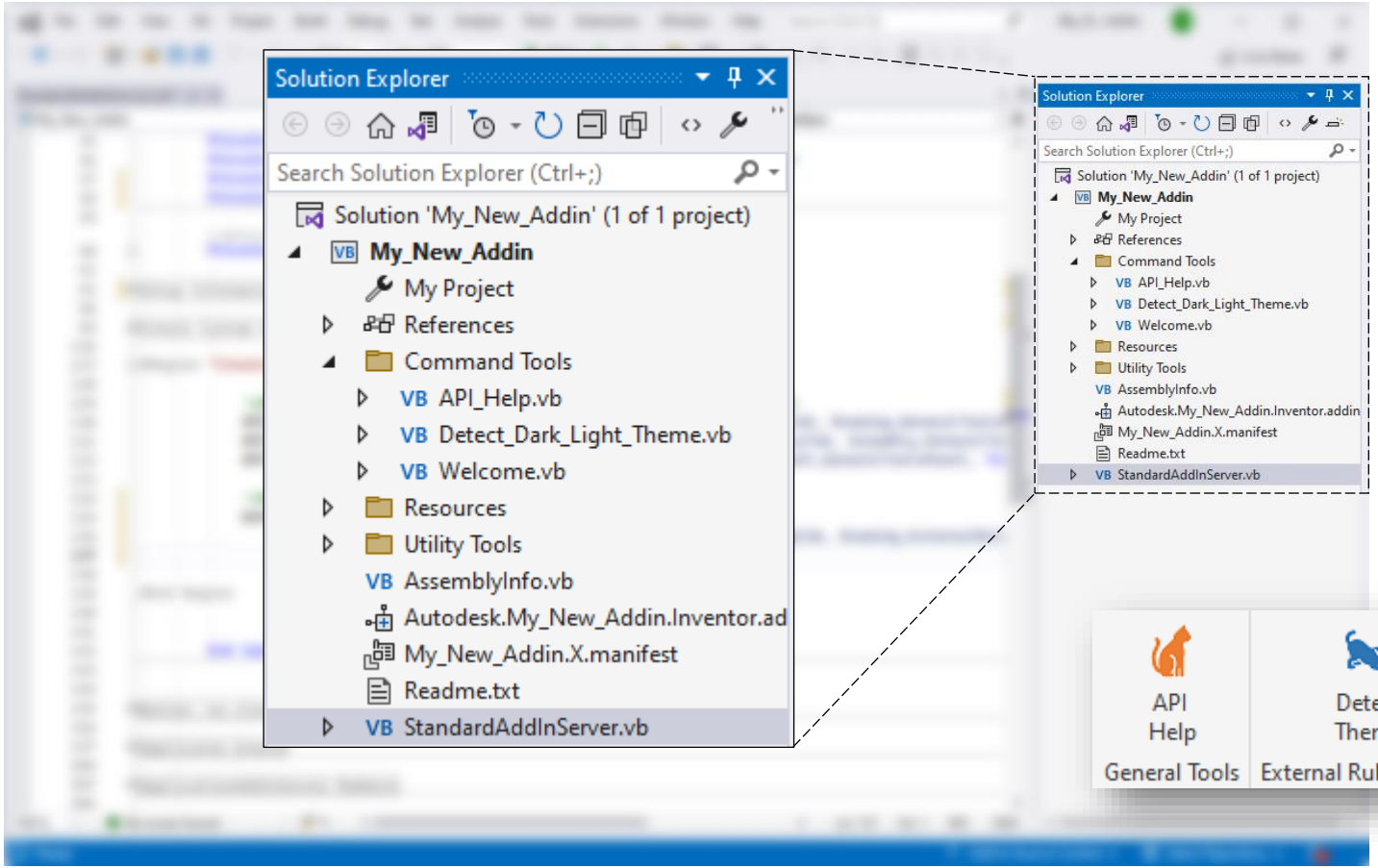

- We can see where the name of the tab is set here.
- We'll change that from **ACME** to **Vandelay**

# Renamed Custom Tab



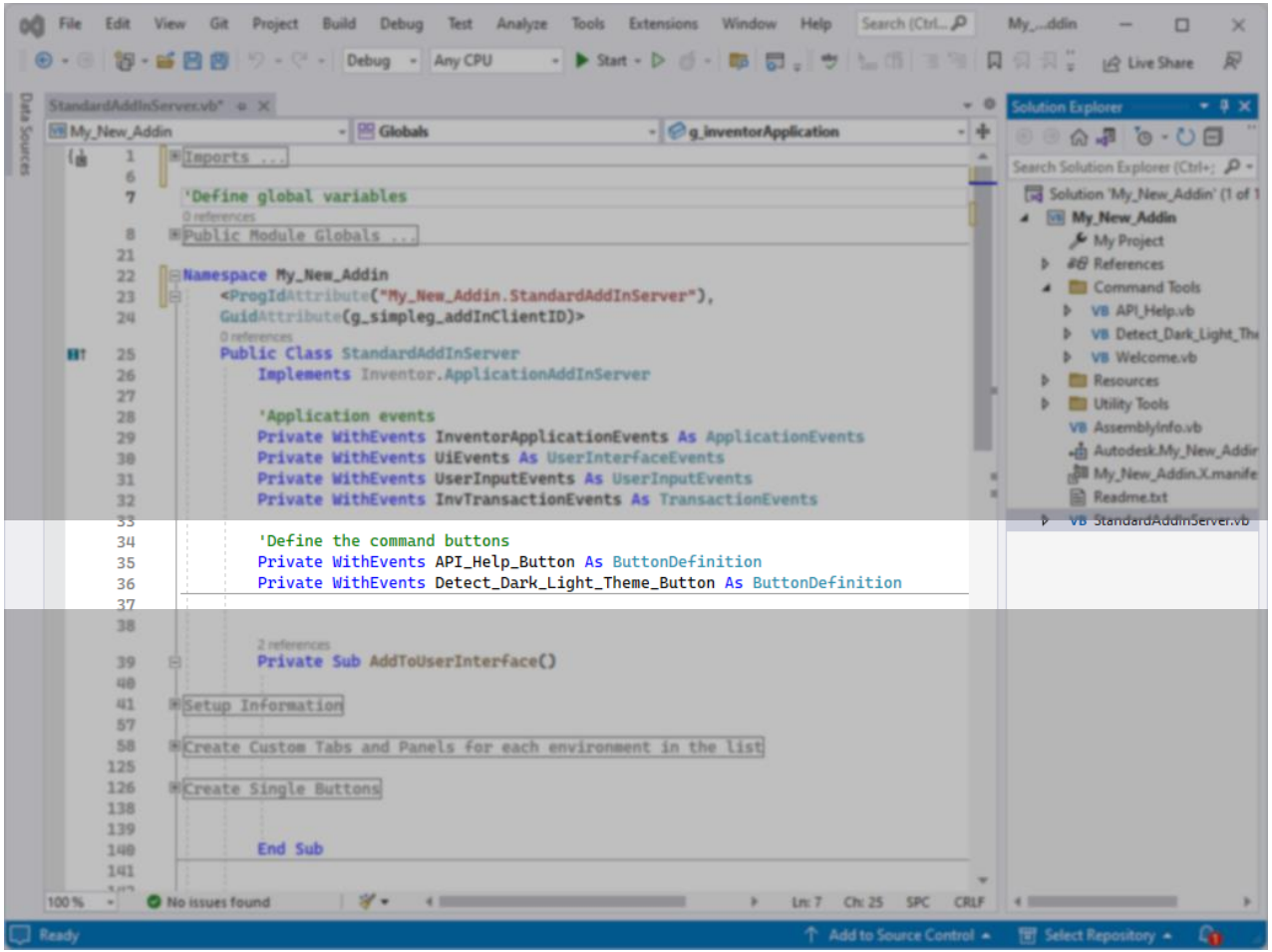

- This will change the name of our custom tabs in Inventor from **ACME** to **Vandelay**

# Modifying the Add-in



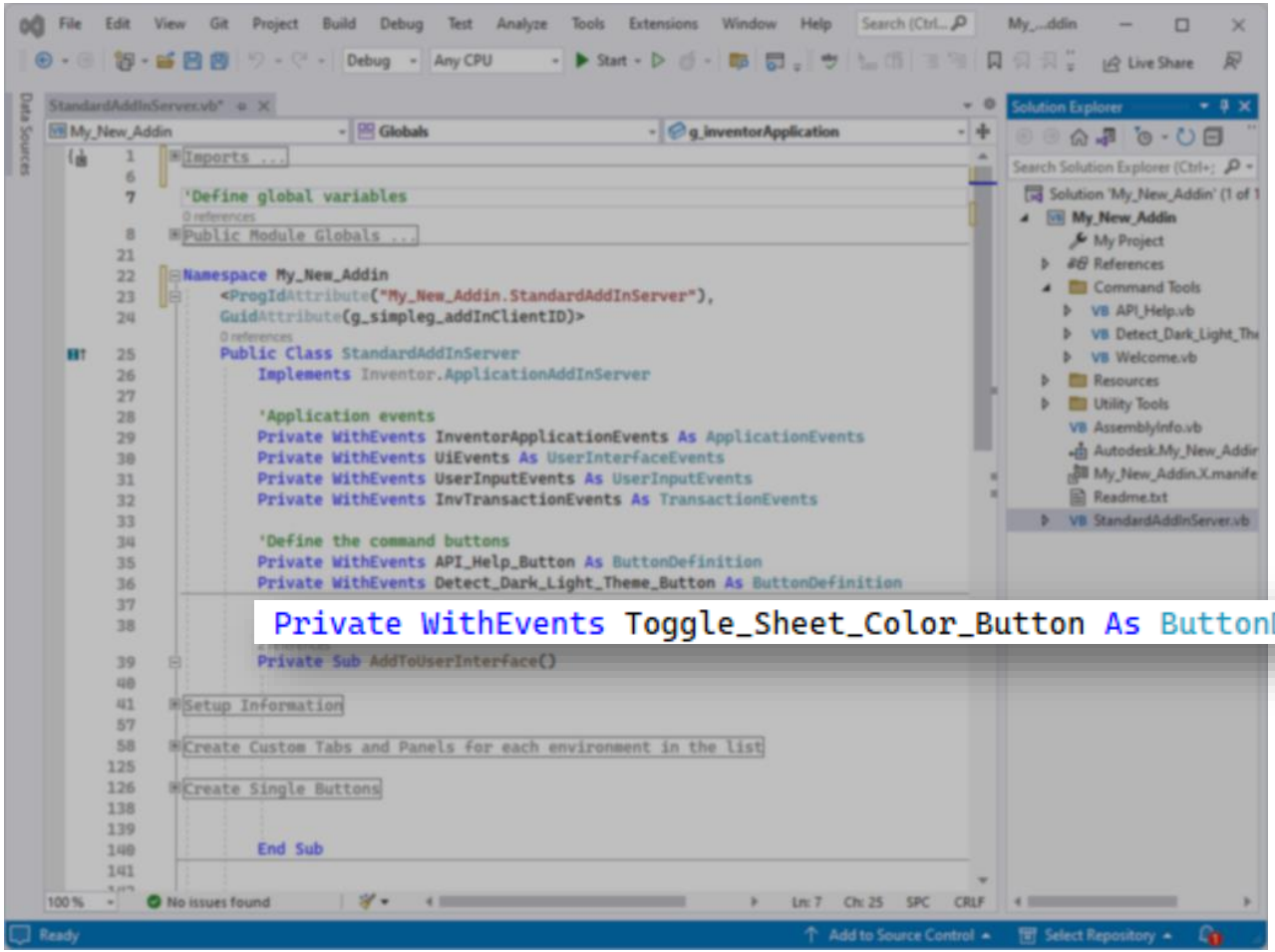

- If we expand the **Create Buttons** region, we can see where the buttons are created

# Modifying the Add-in



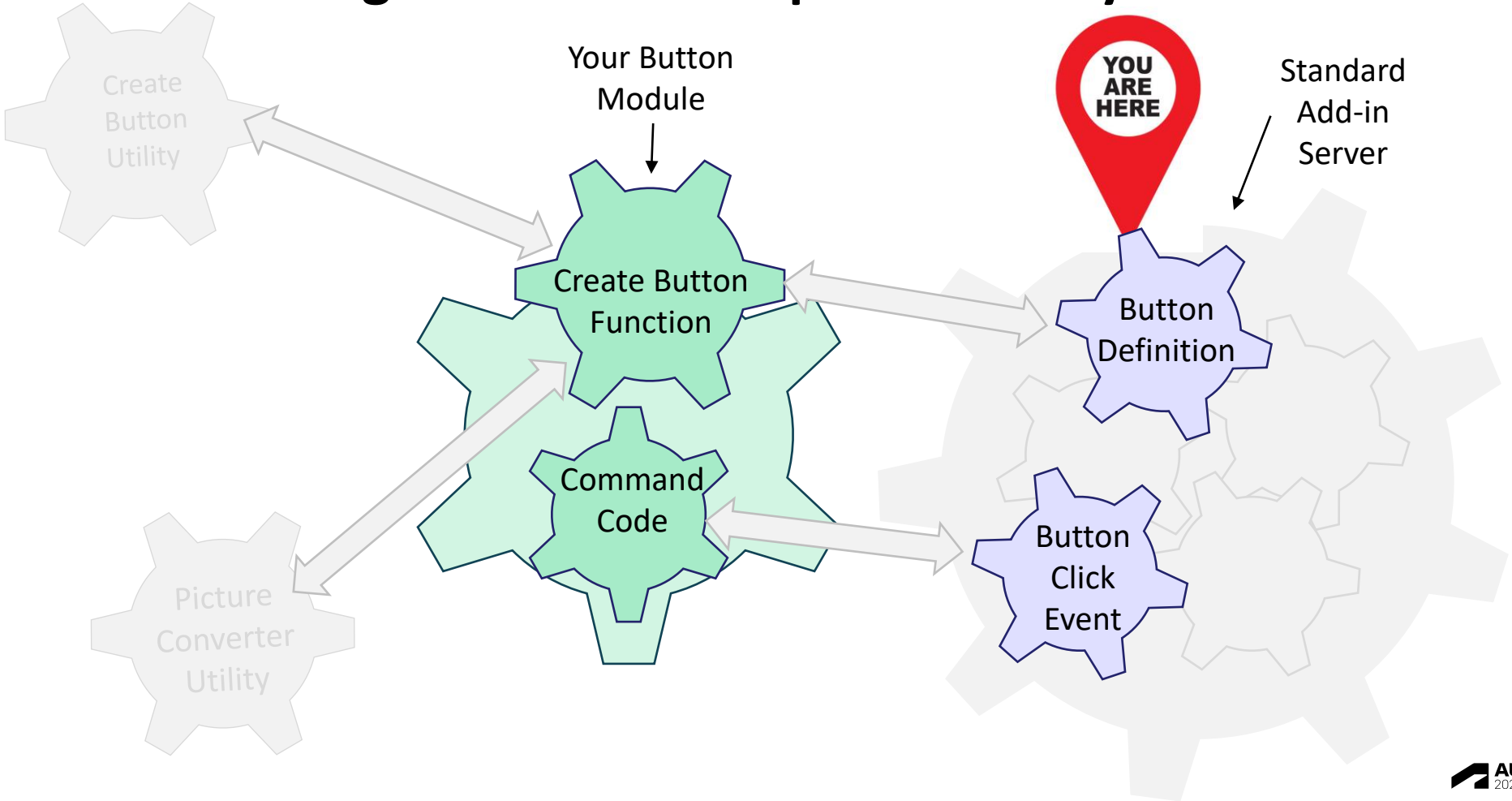

- Still in the **StandardAddinServer** file…
- Note the area at the top where the buttons are defined.
- We'll add a new button

# Modifying the Add-in

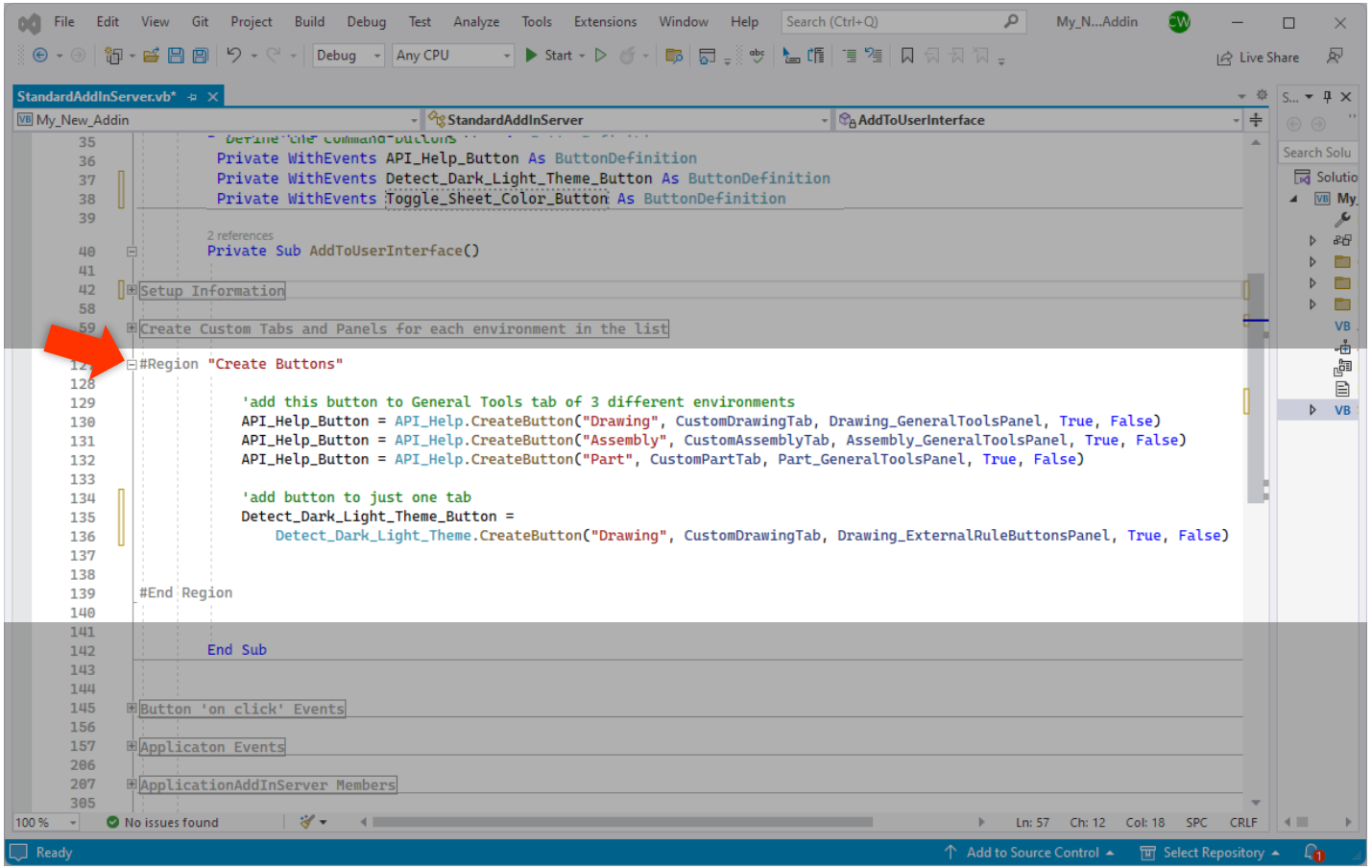



- We'll add a new button called:
  - Toggle_Sheet_Color_Button

# Understanding the Add-in Template Visually

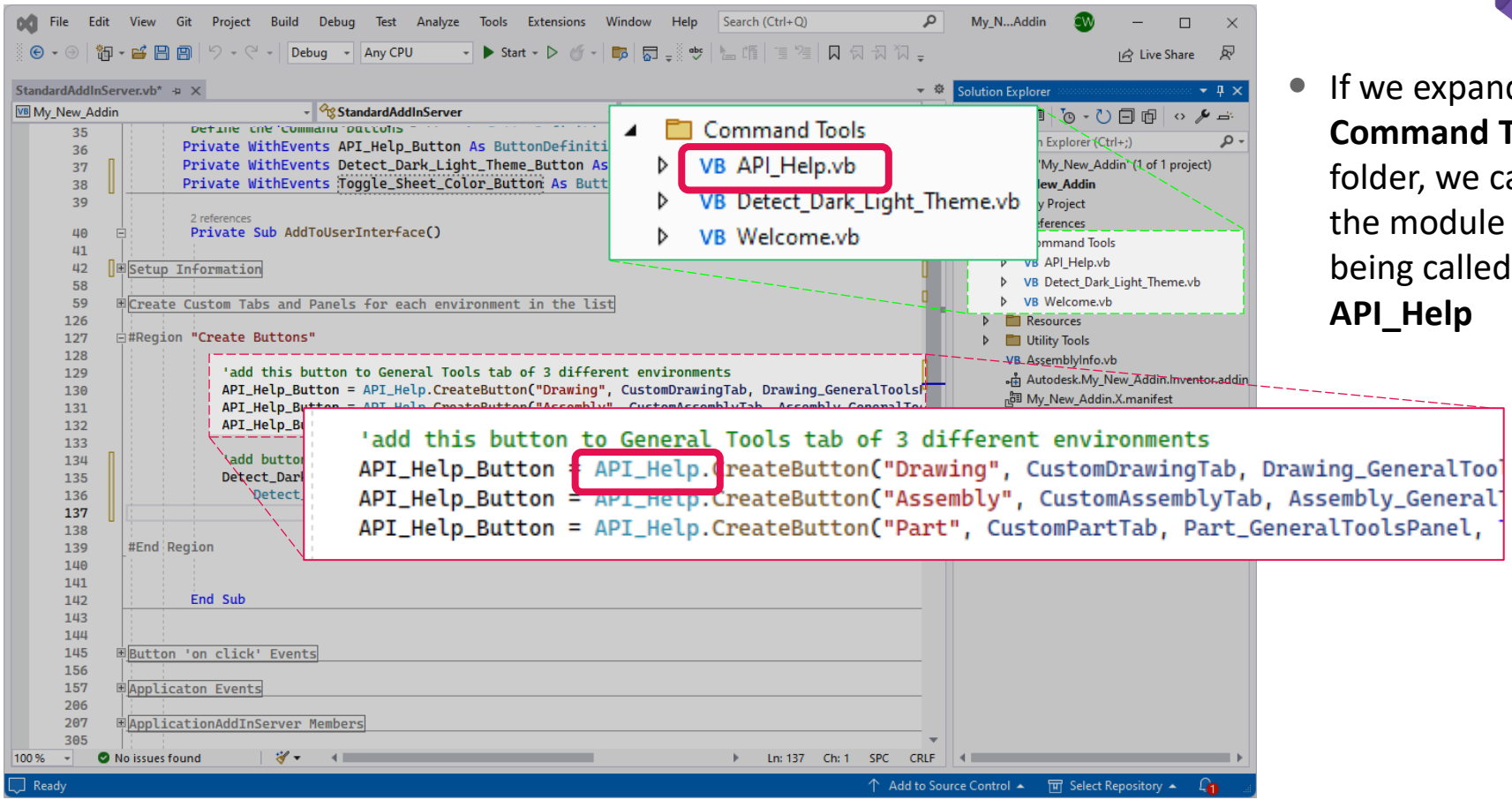


# Modifying the Add-in



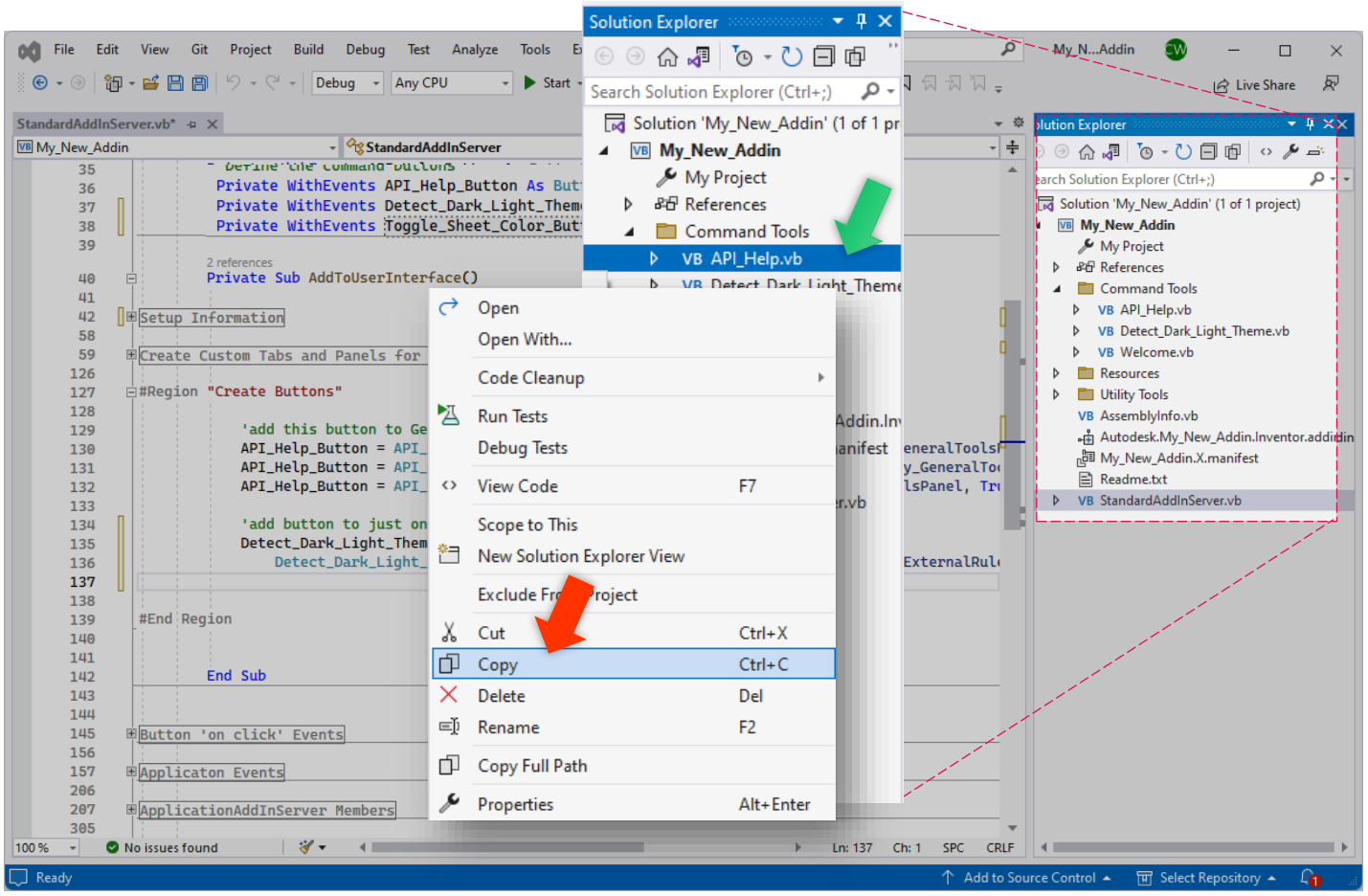

- If we expand the **Create Buttons** region, we can see where the buttons are created

# Modifying the Add-in



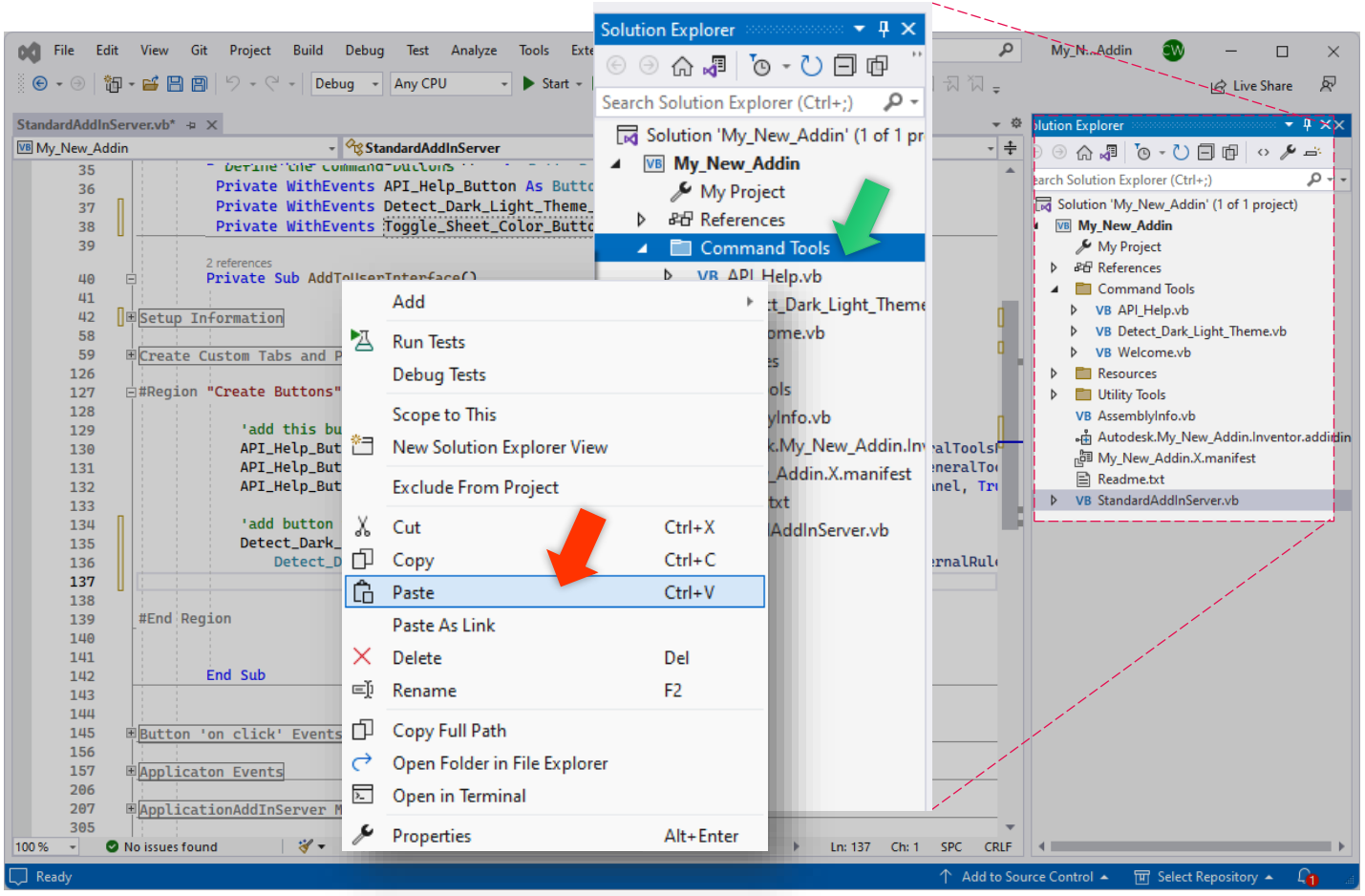

- If we expand the **Command Tools** folder, we can see the module that is being called **API_Help**

# Modifying the Add-in



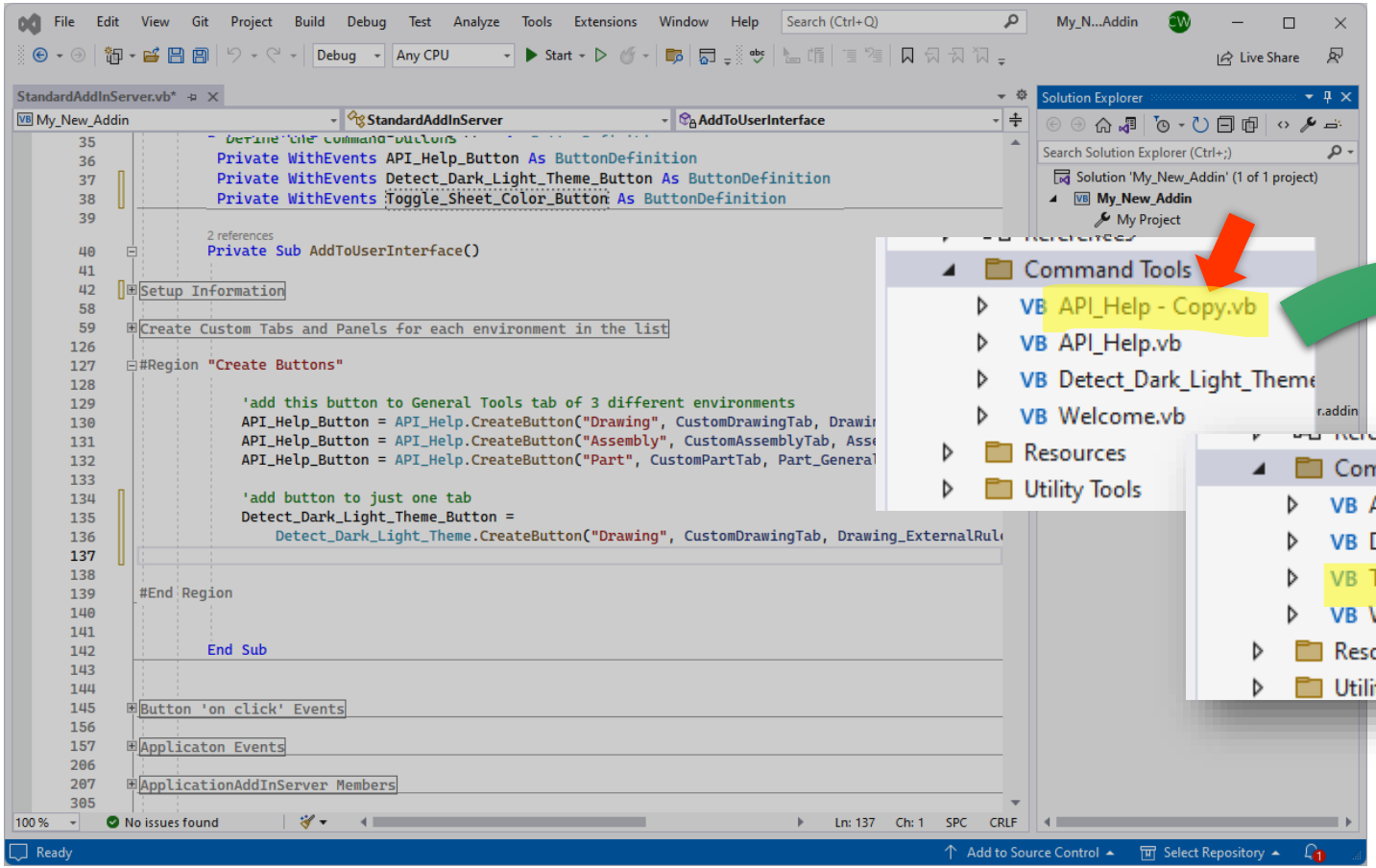

- To create a new button module we can simply copy an existing module
- Right click on the module to copy and choose Copy as shown

# Modifying the Add-in



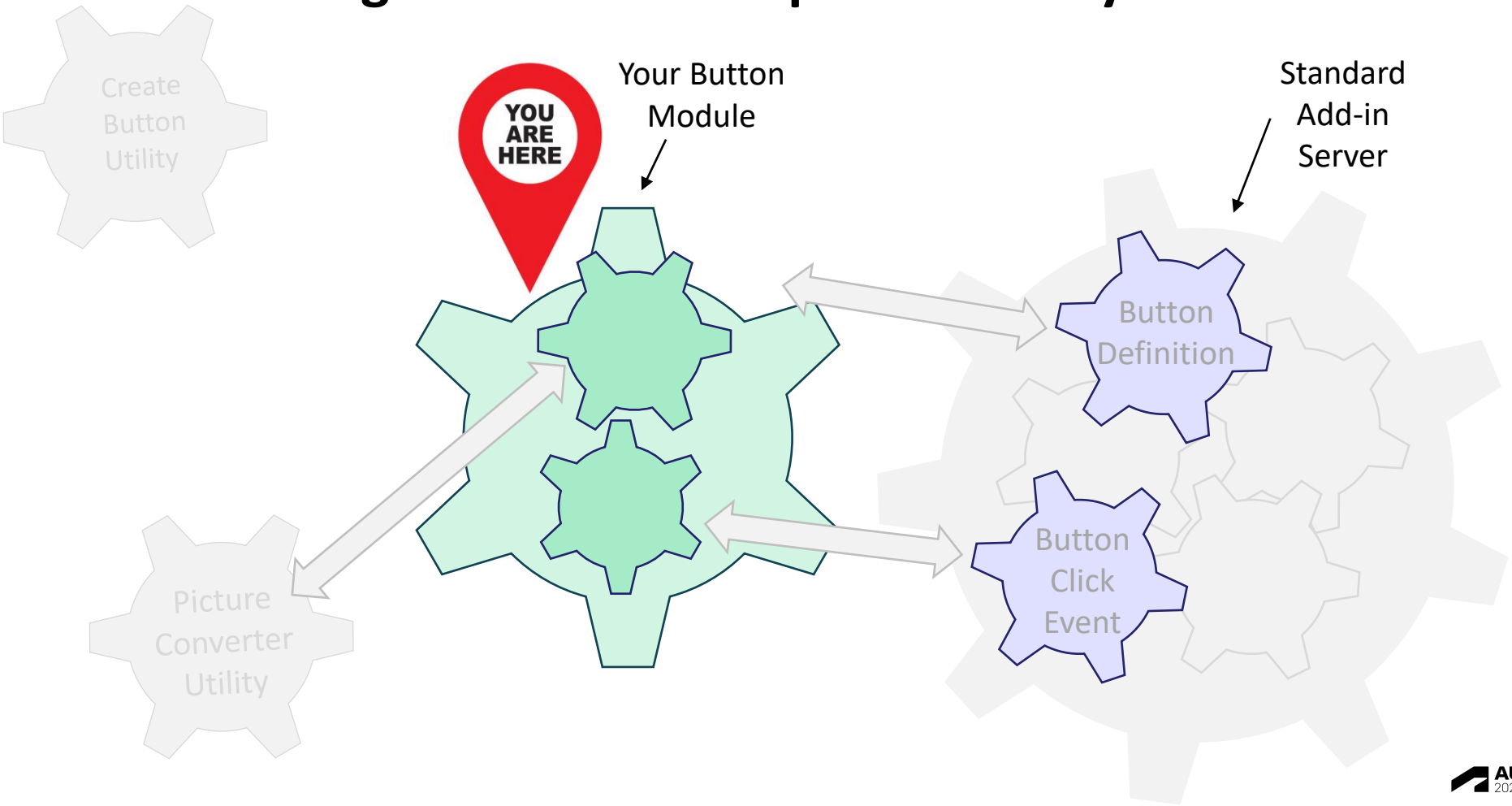

- Right click on the Command Tools folder and choose Paste

# Modifying the Add-in

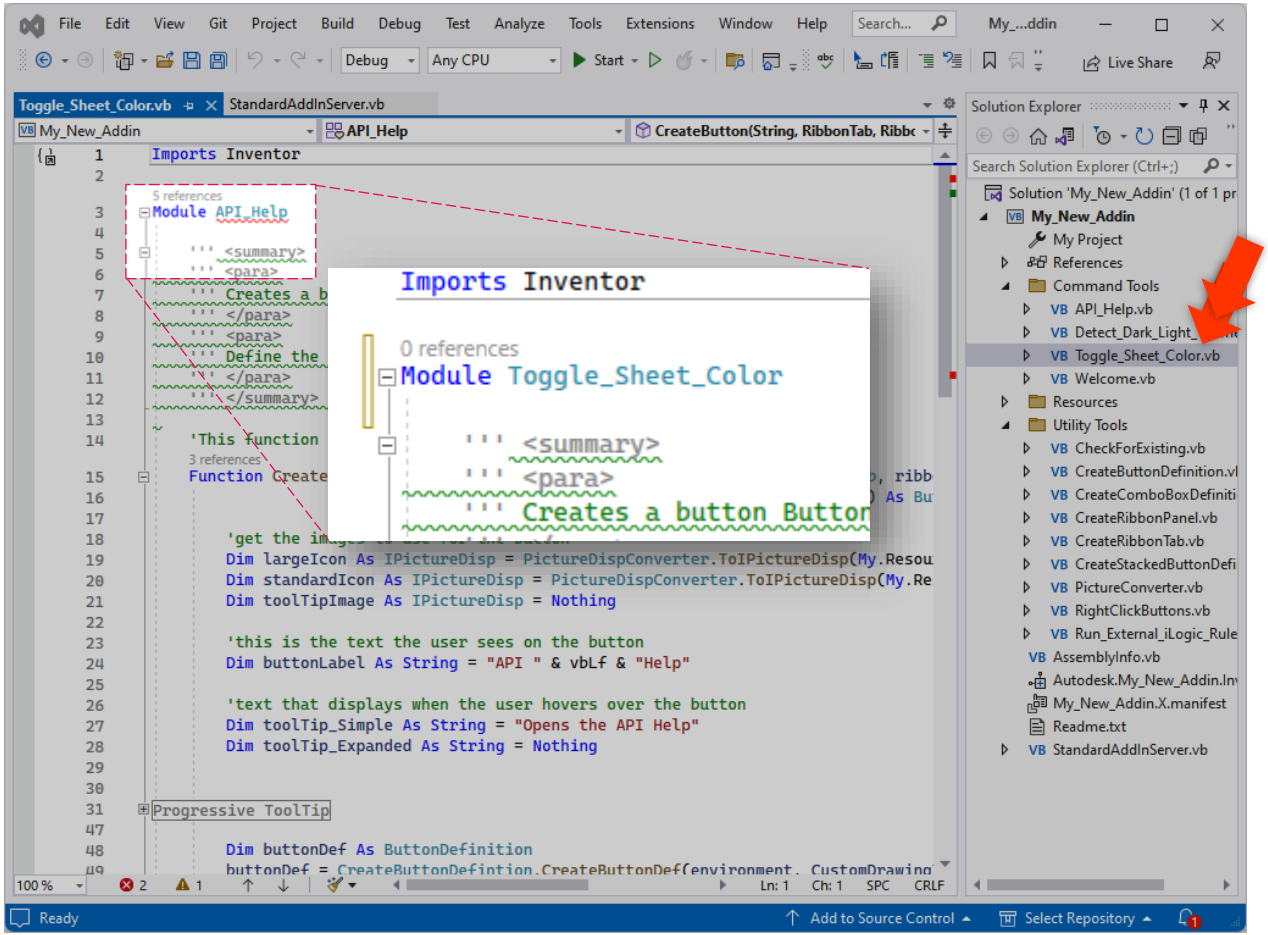



- Rename the copy

# Understanding the Add-in Template Visually

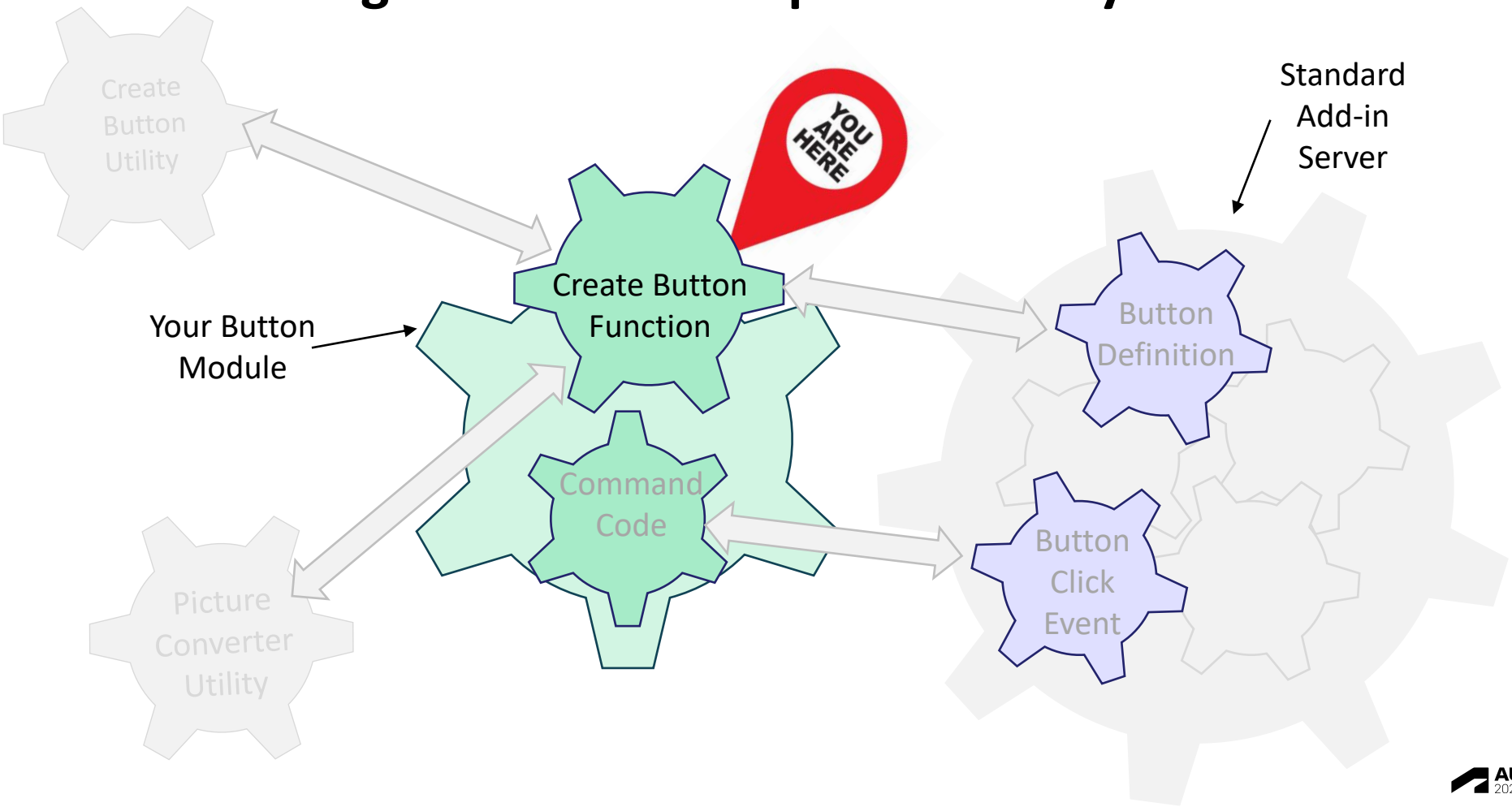


# Modifying the Add-in

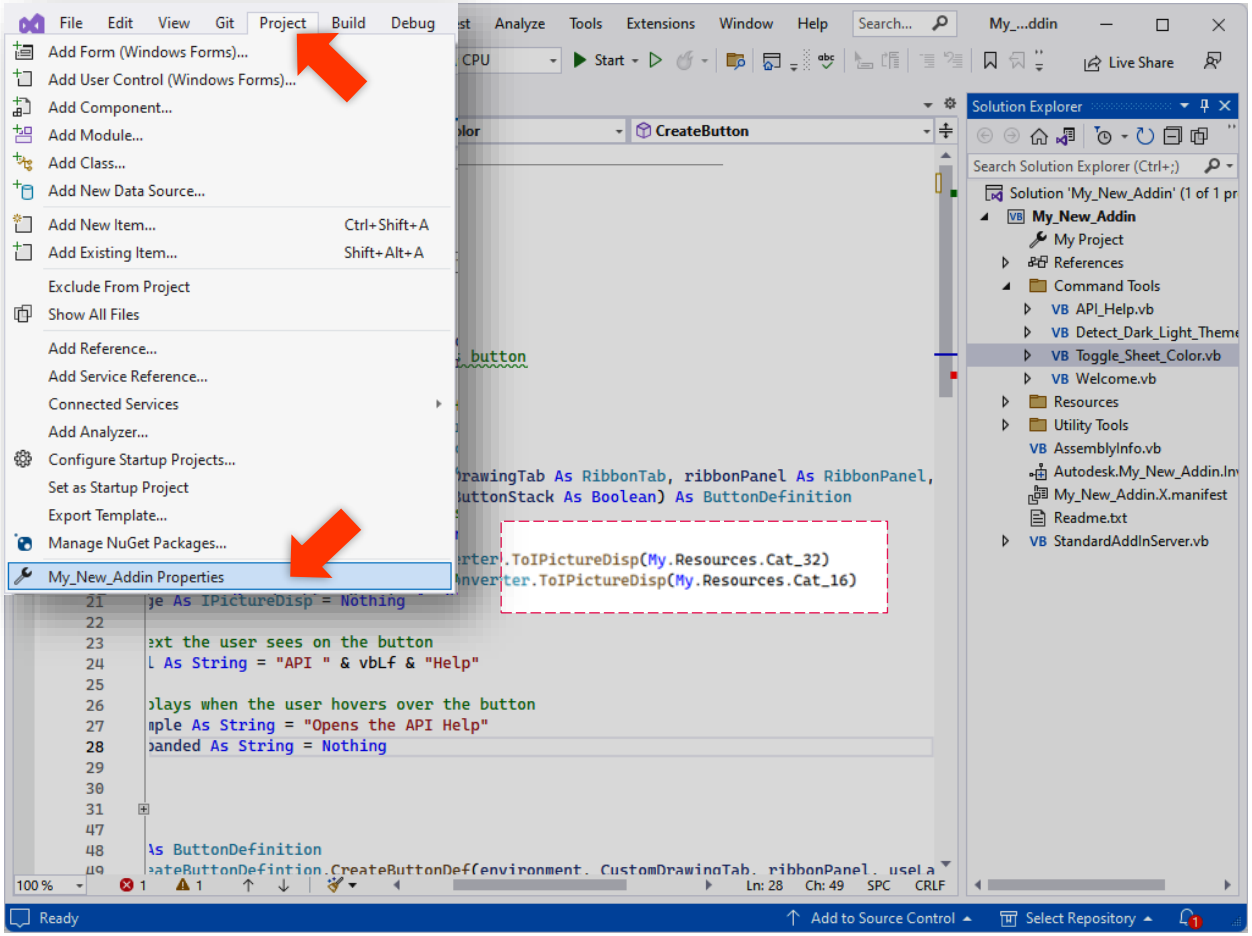



- Next we'll double click the Toggle_Sheet_Color module to open it
- And change the module name from **API_Help** to **Toggle_Sheet_Color**

# Understanding the Add-in Template Visually



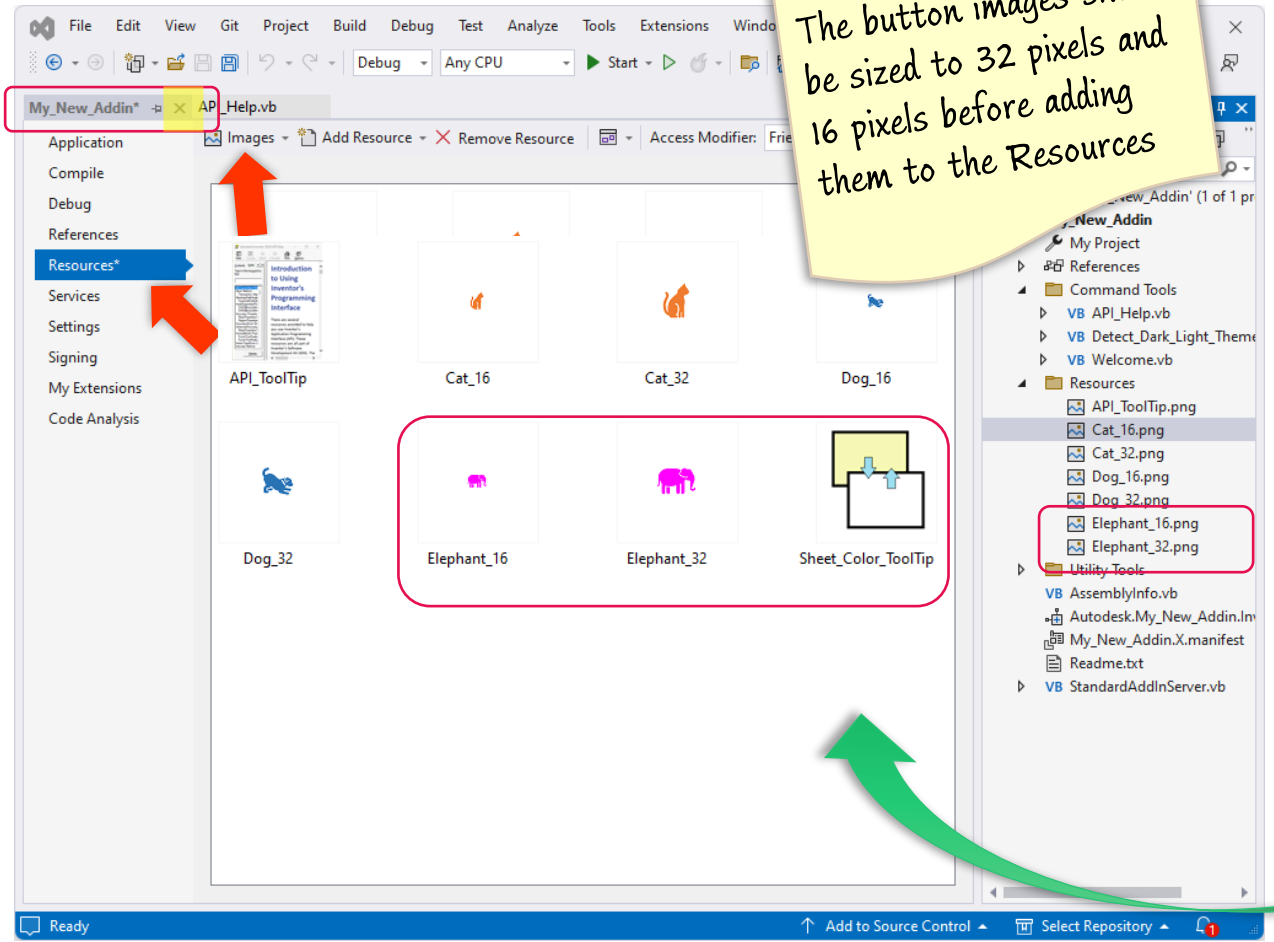
# Modifying the Add-in



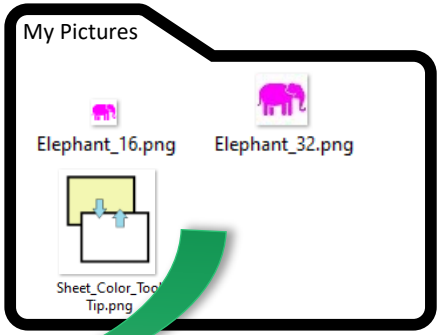

- Next we'll change the button image
- But first we need to add an image to our Resources.
- Go to **Project** > **Properties** as shown

# Modifying the Add-in

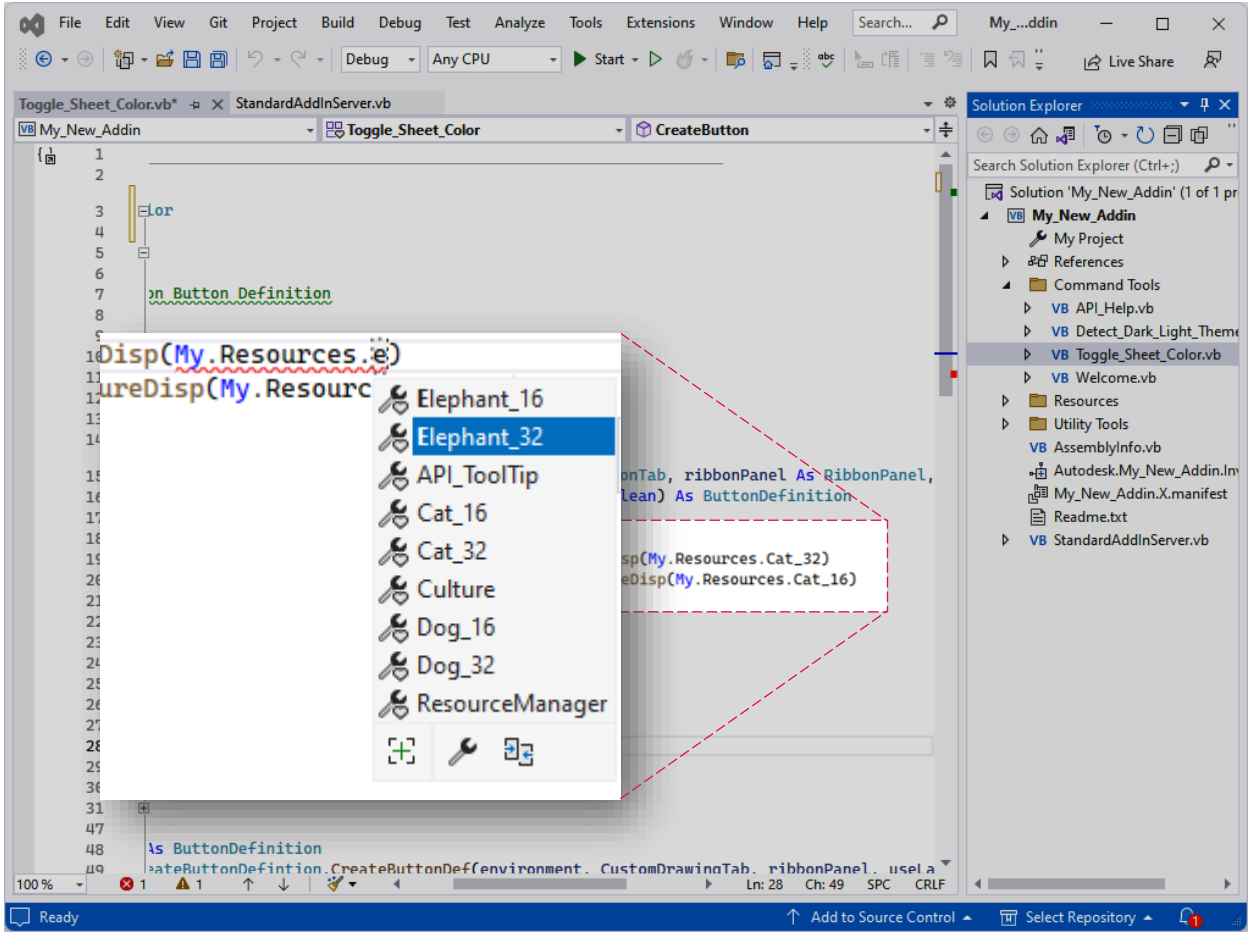
Note:
The button images should be sized to 32 pixels and 16 pixels before adding them to the Resources




- You can use your own images or use the ones provided with the download materials
- Simply drag and drop the image on the Images pane as shown
- Then click the X to close and save the **My_New_Addin** file

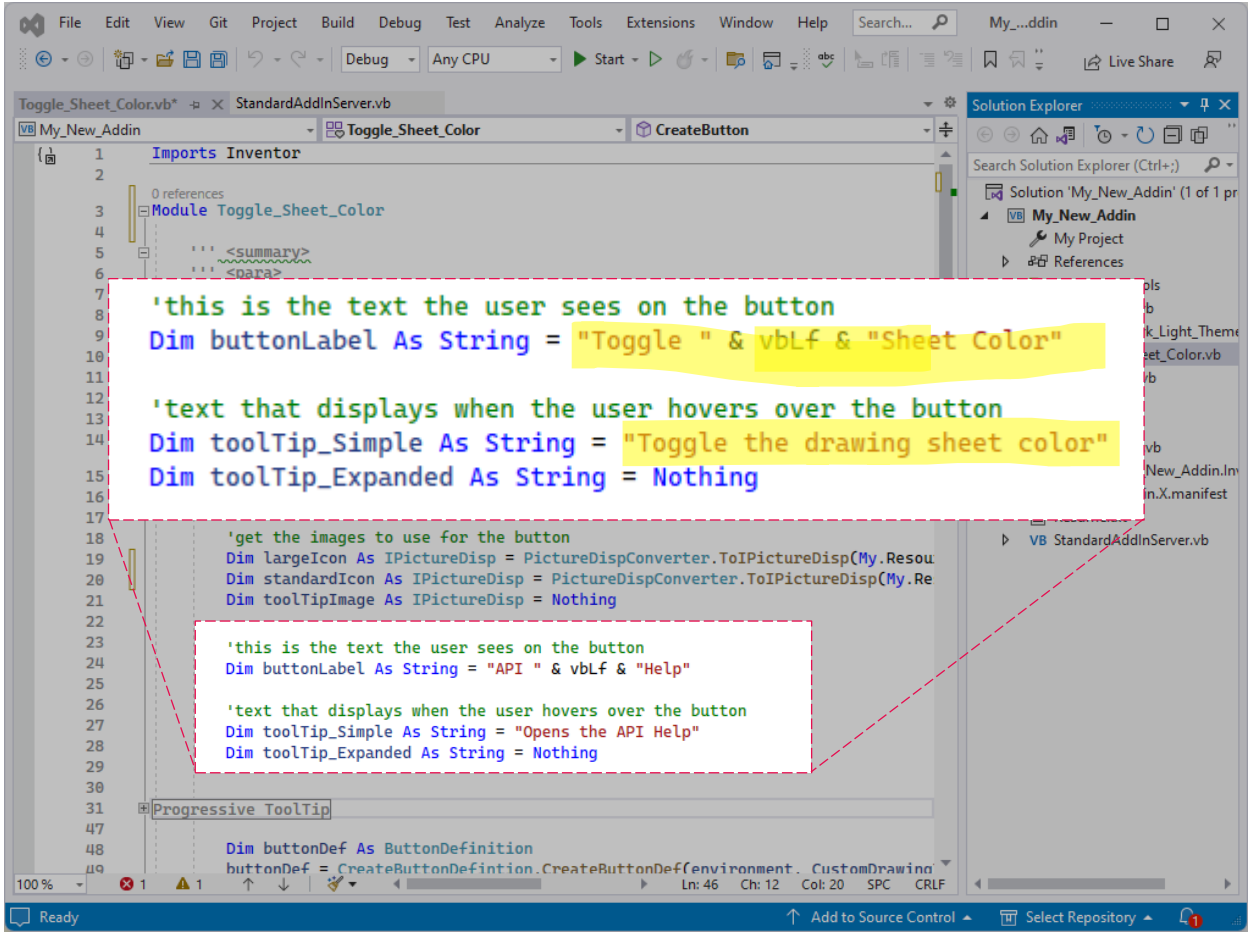


# Modifying the Add-in



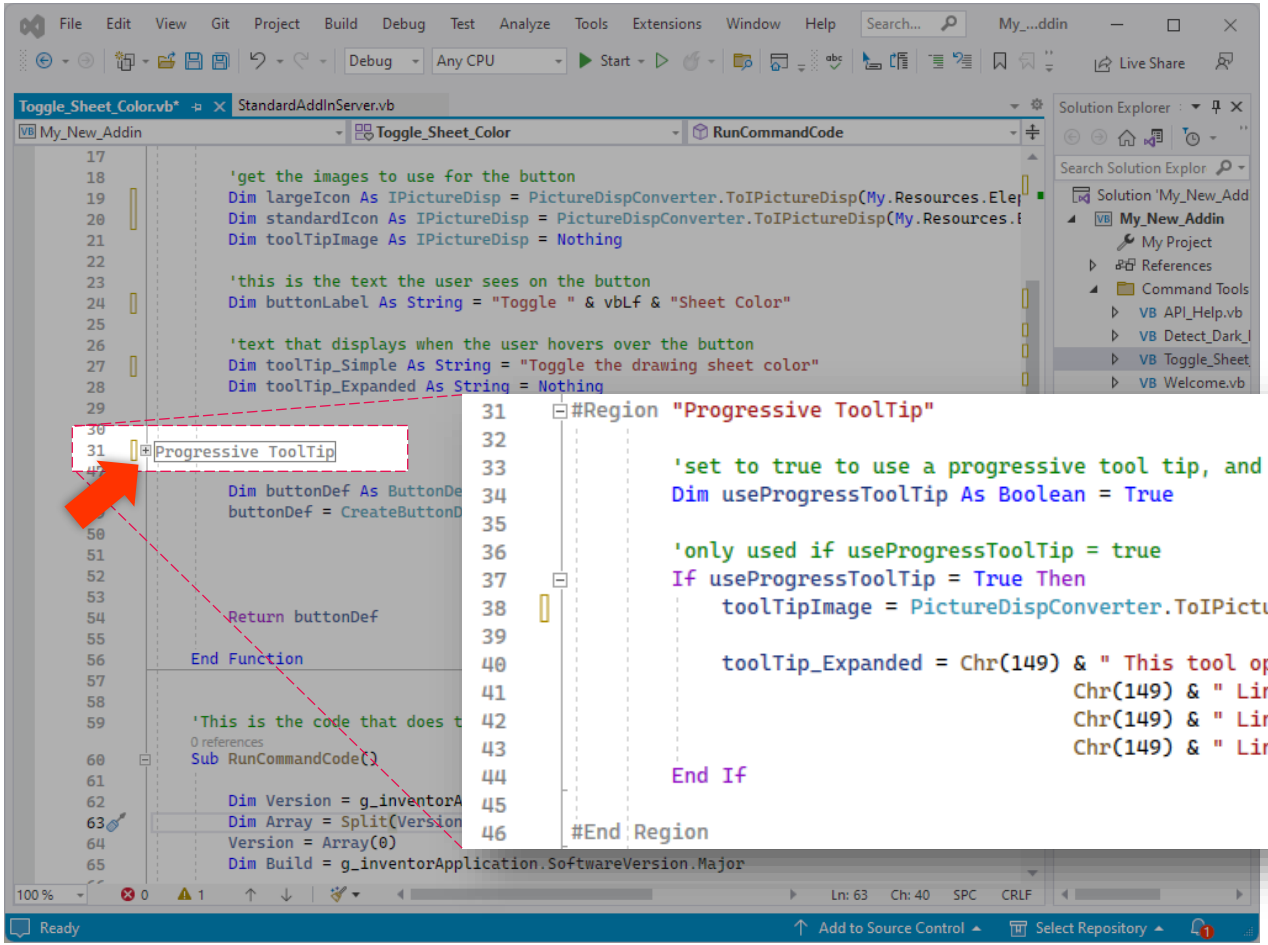

- Returning to our new command module file…
- Backspace over Cat_32 and start typing the name of your images to see them appear in the list
- You can then just select the image from the list and double-click it
- Do the same to the _16 image

# Modifying the Add-in



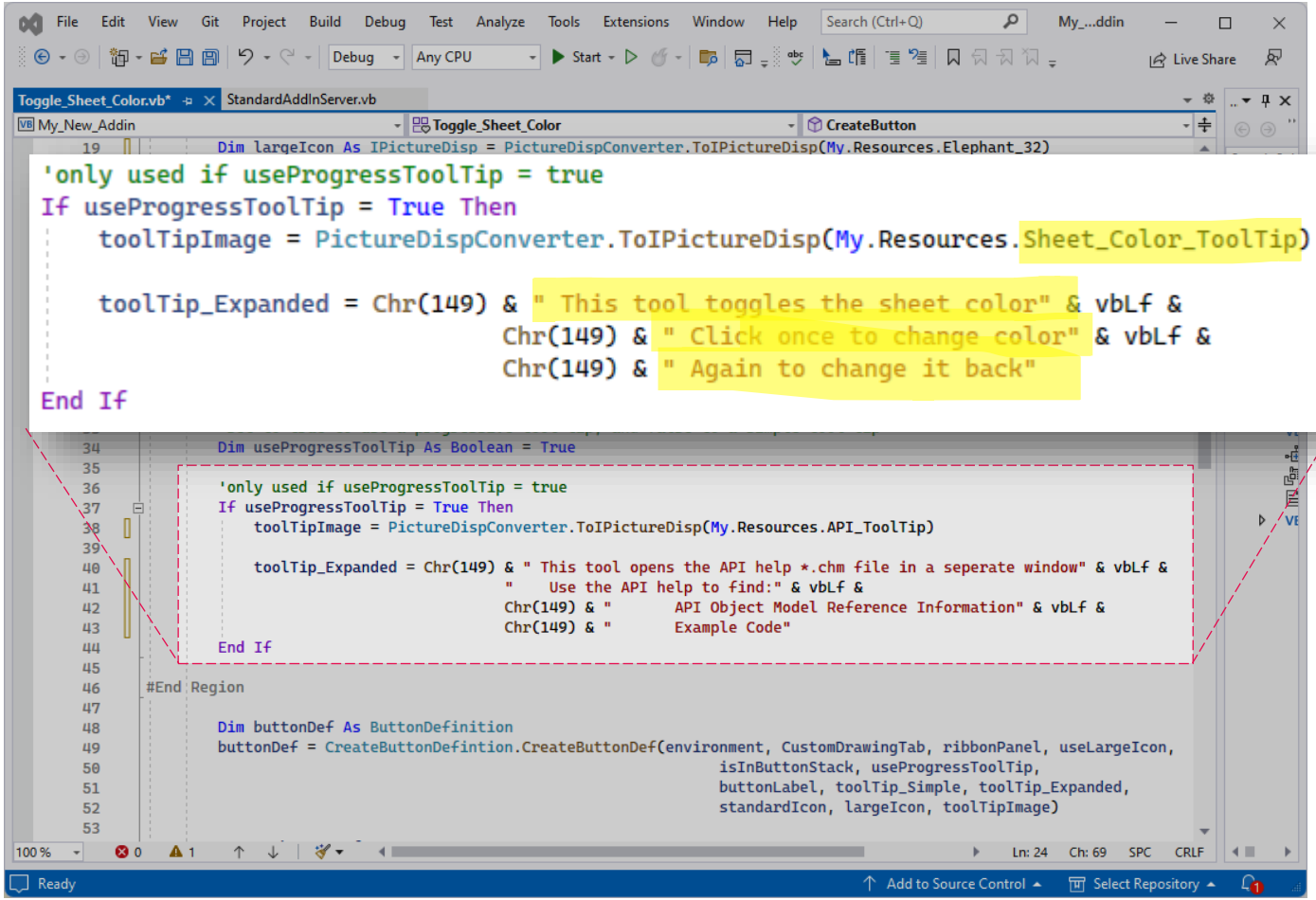

- Next we'll update the button label
- And the simple tool tip string

# Modifying the Add-in



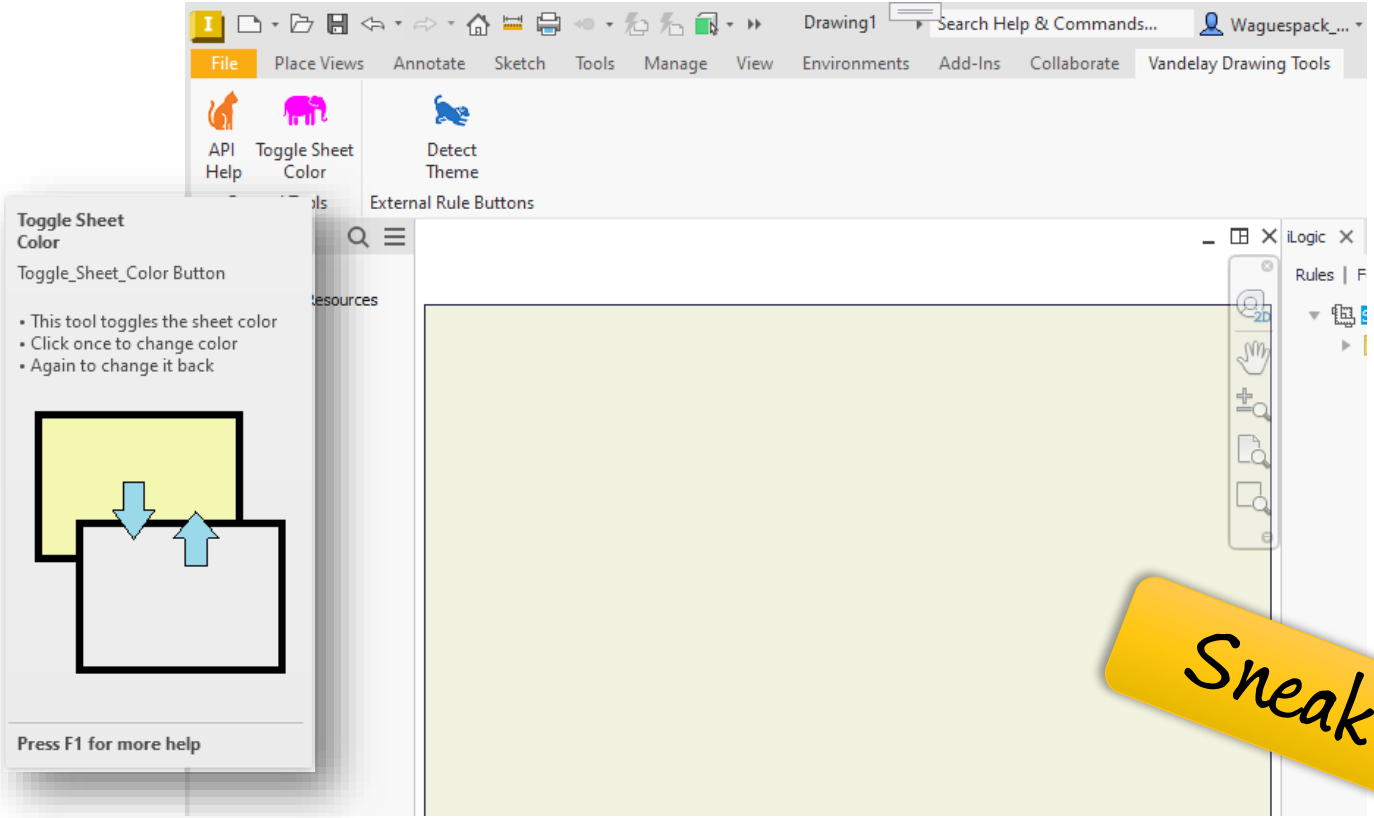

- Next we'll expand the Progressive ToolTip region by clicking the + sign

# Modifying the Add-in




- We'll change the tooltip image
- As well as the extended tooltip text

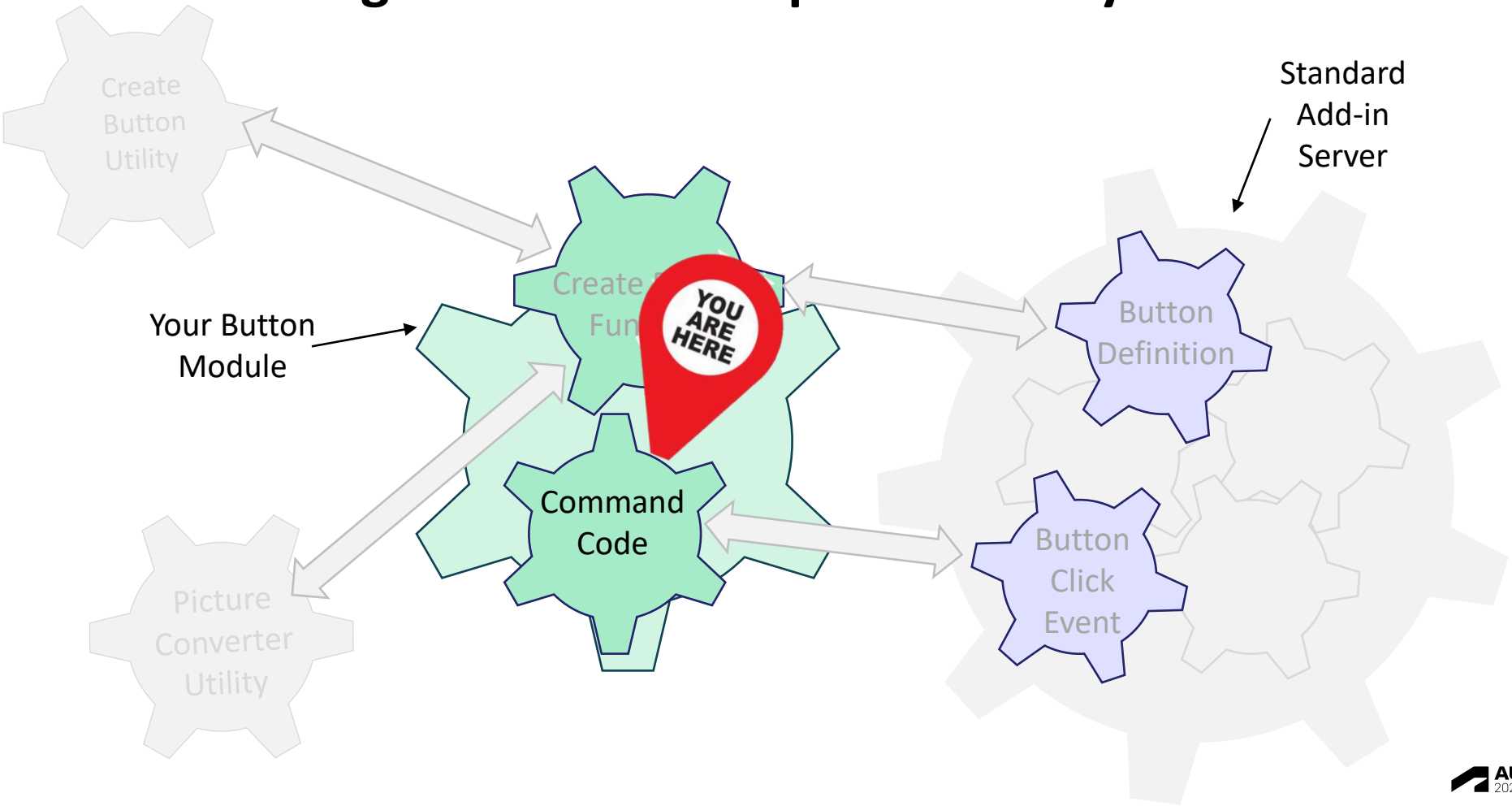
# Modifying the Add-in



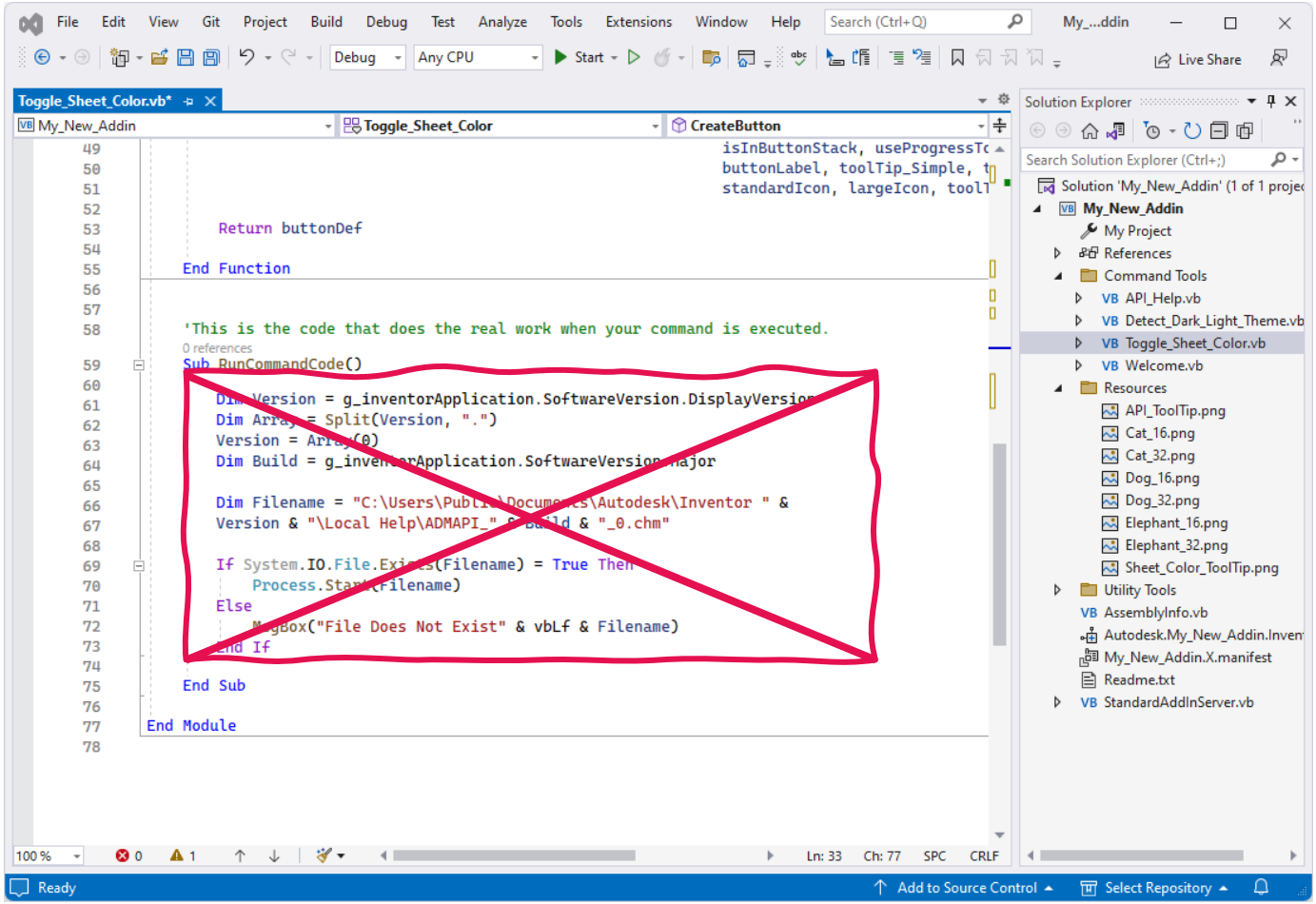

Sneak Peak

- This is what the extended tooltip that we are creating will look like

# Understanding the Add-in Template Visually

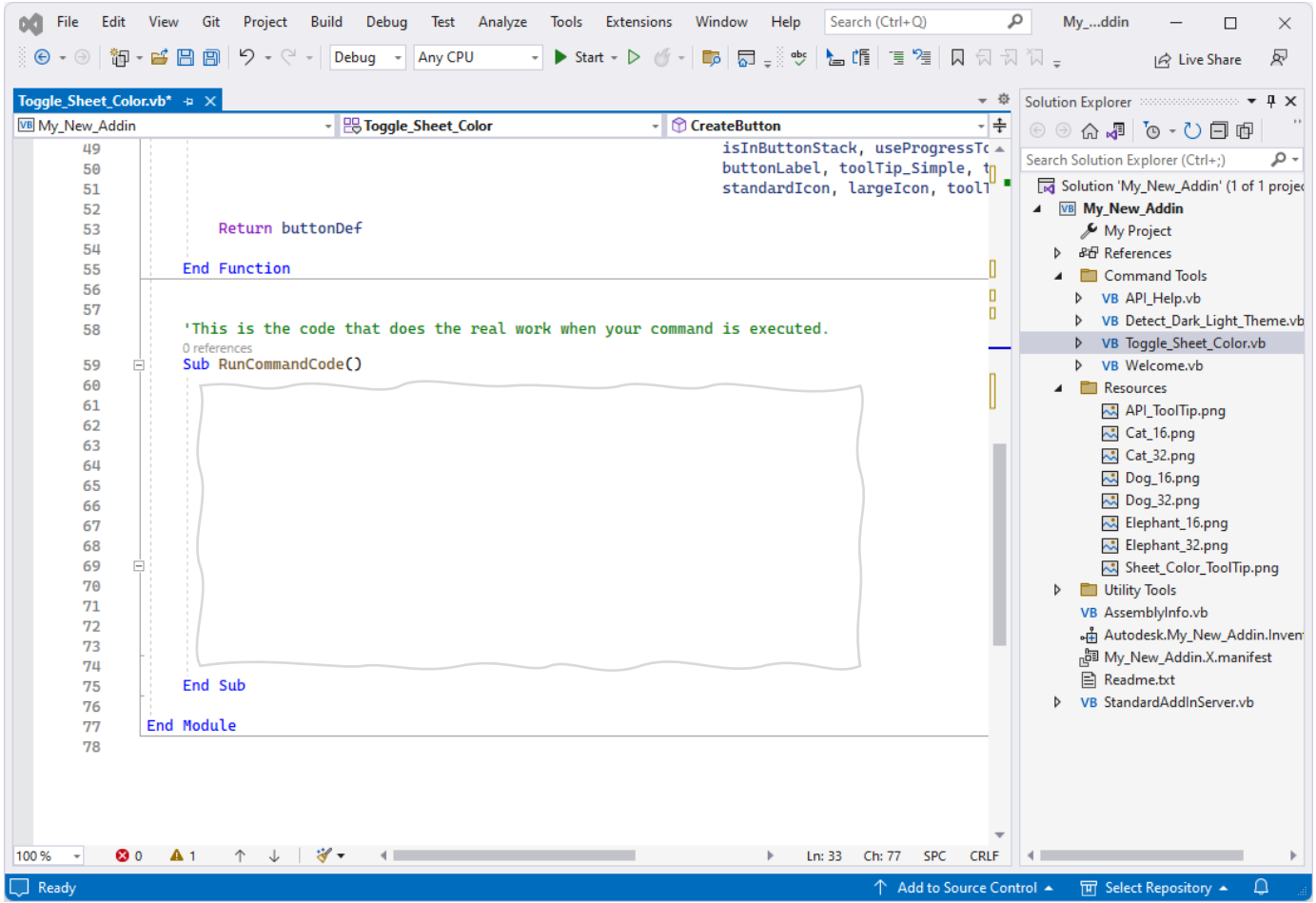


# Modifying the Add-in



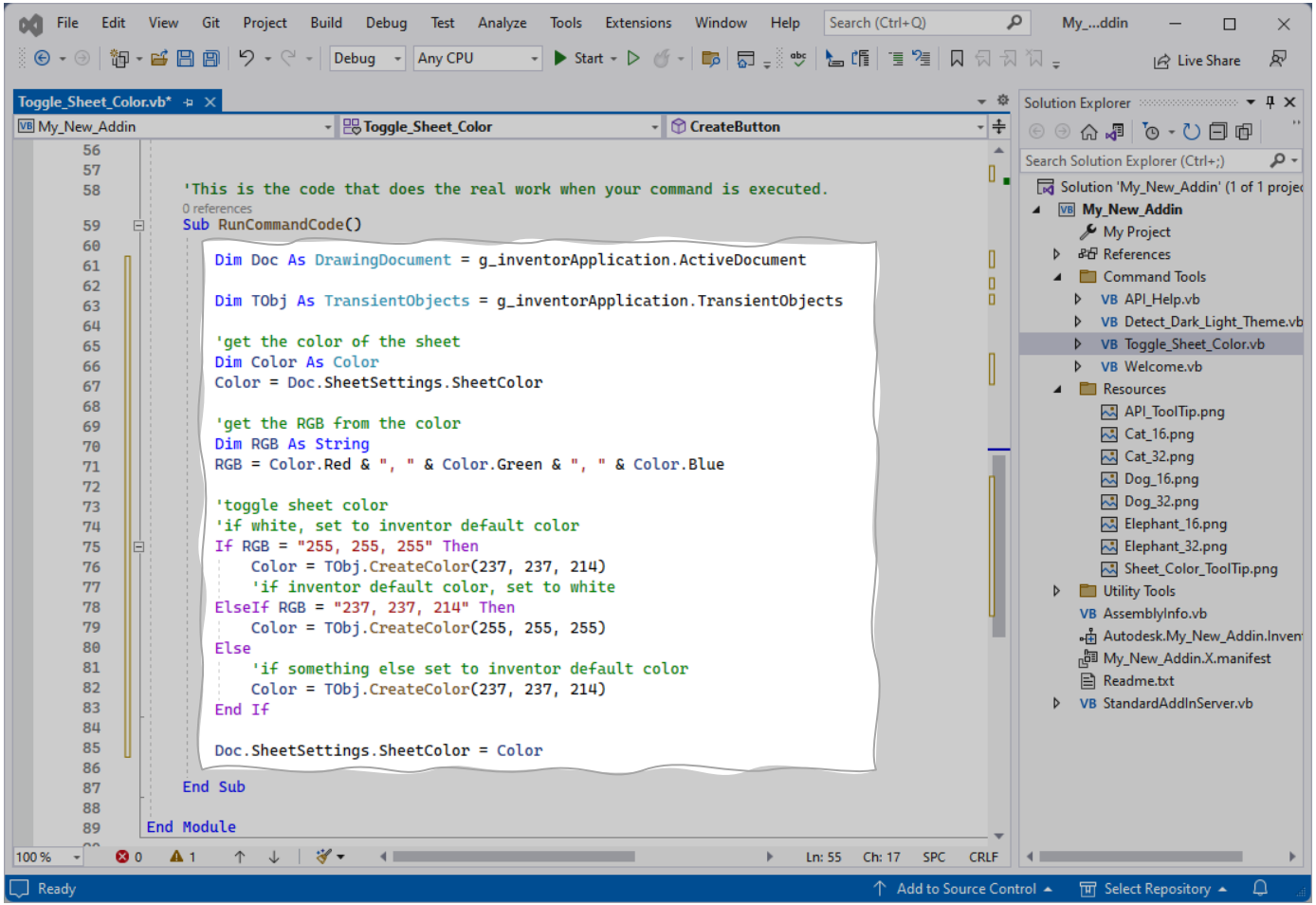

- Next we'll scroll down to the **RunCommandCode** sub
- Delete all of the code between the **Sub** and **End Sub** lines

# Modifying the Add-in



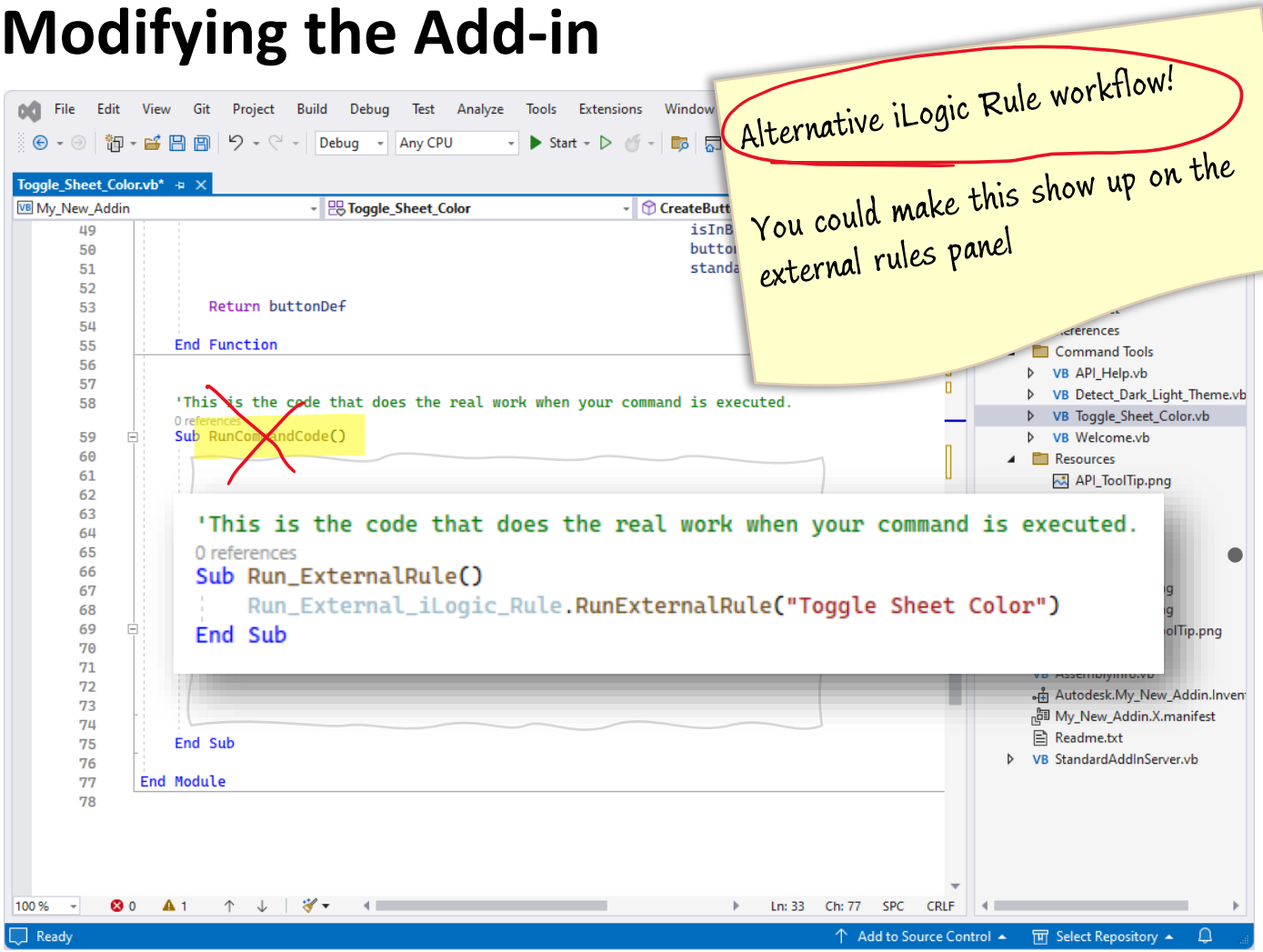

- Next we'll scroll down to the **RunCommandCode** sub
- Delete all of the code between the **Sub** and **End Sub** lines

# Modifying the Add-in




- Add the code that we want to run when the button is clicked.
- You can find this example code in the download materials for this class

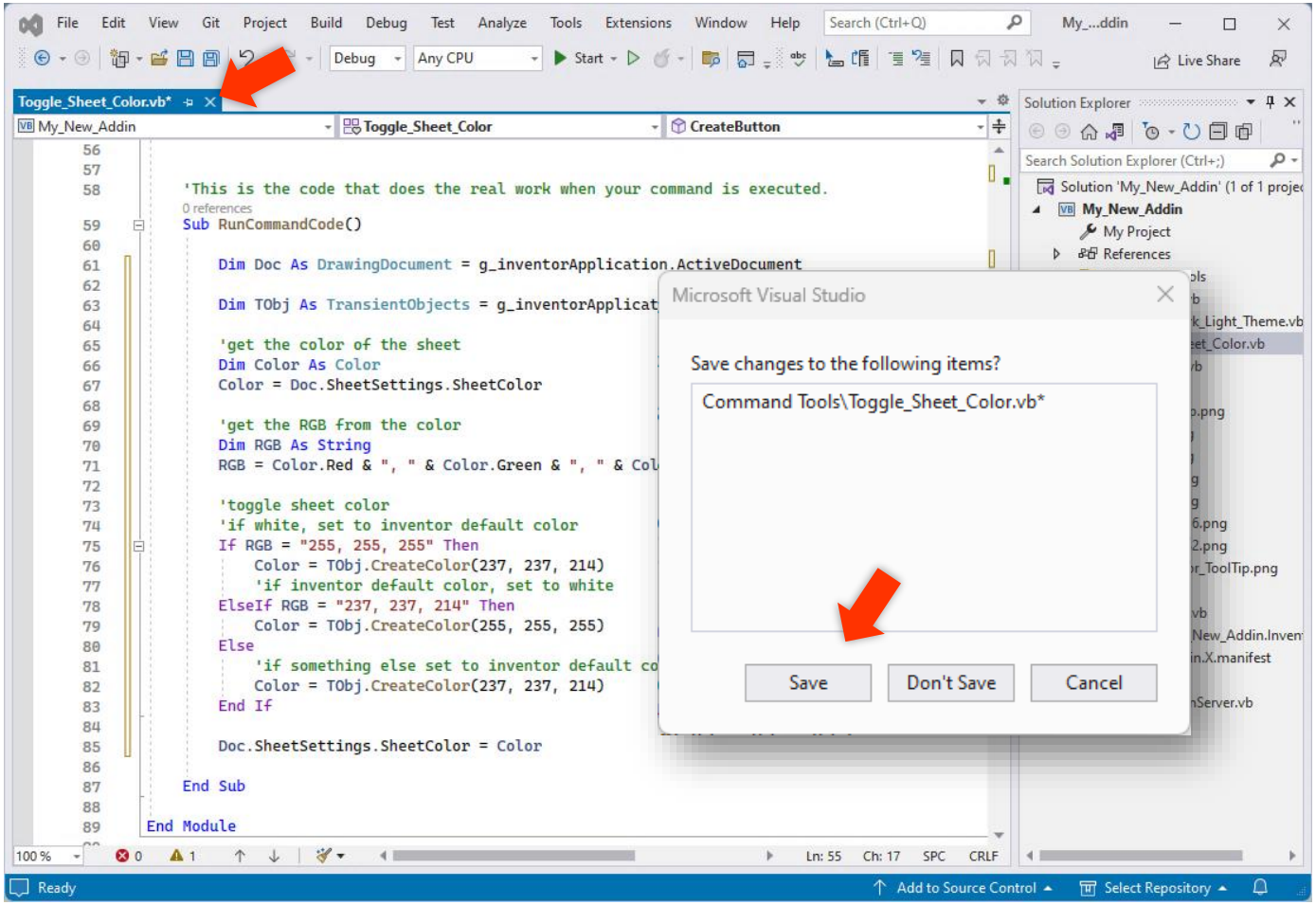
# Modifying the Add-in




you wanted this module to run an external iLogic rule, rather than executing code from the add-in, then you would change the sub as shown.

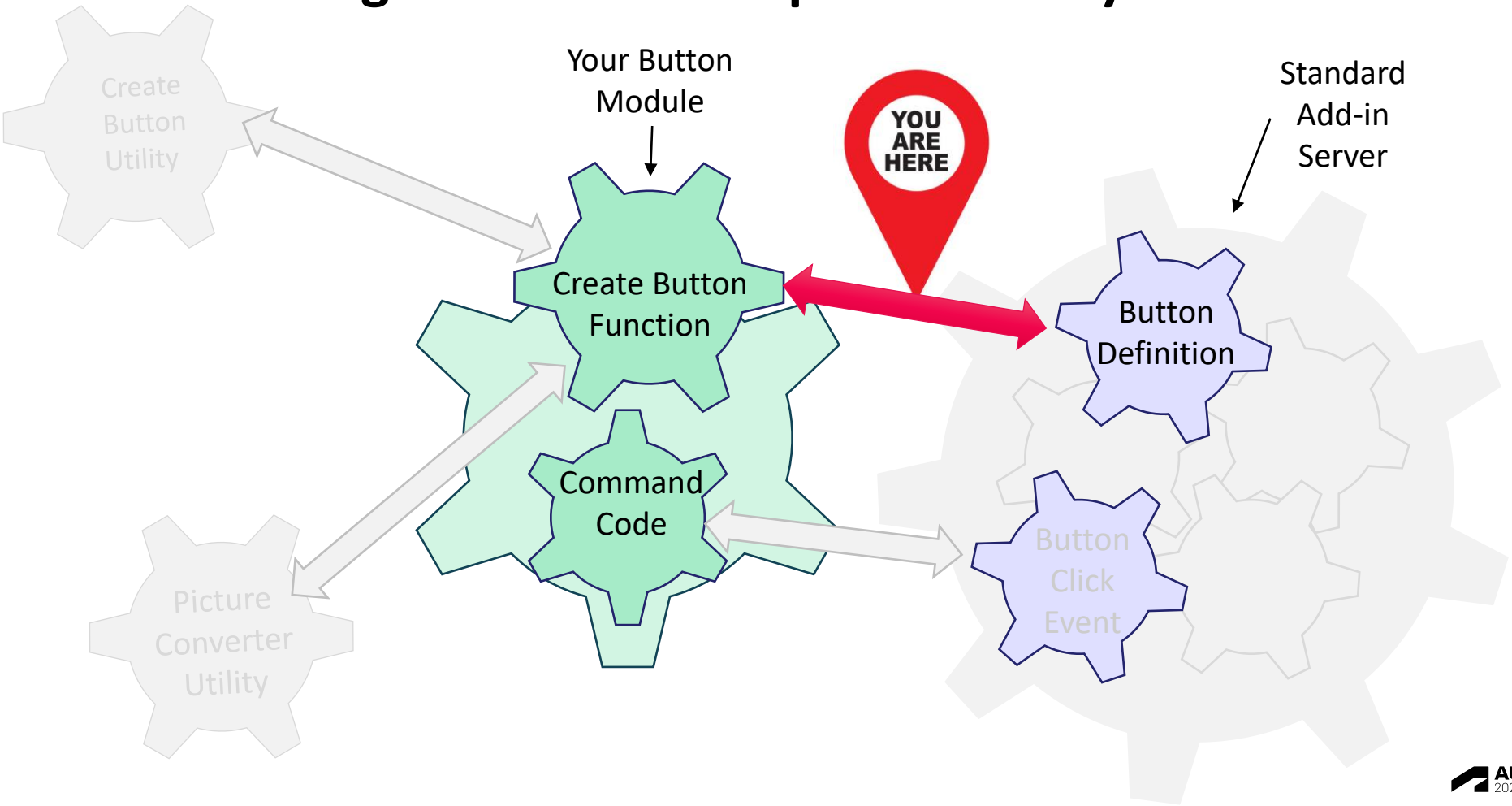
This will use a utility module called **Run_External_iLogic_Rule** as shown

# Modifying the Add-in

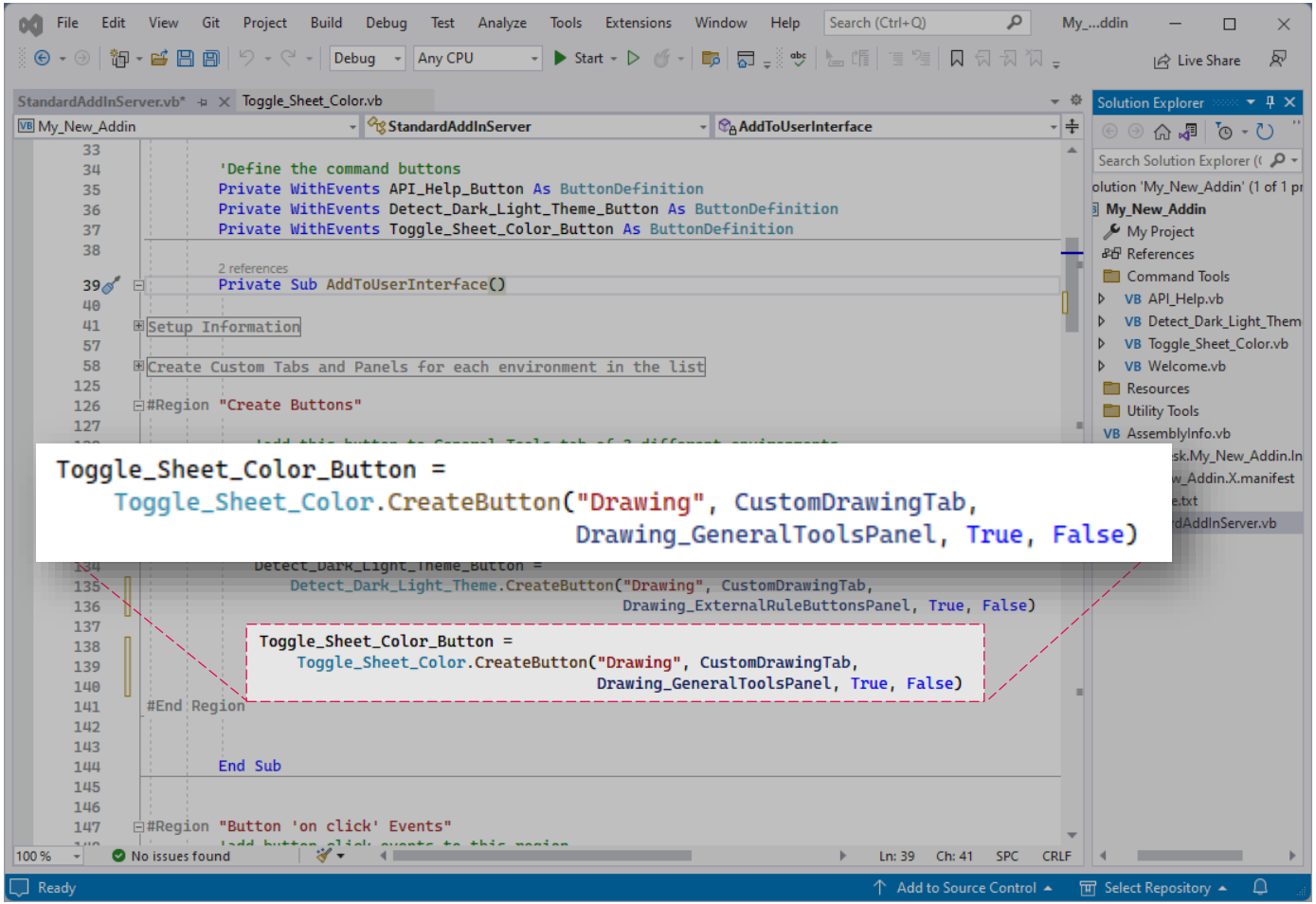



- We're done!
- We've just created a new button module.
- Next, we'll "wire it up" in the StandardAddInServer file.
- But fist click the close **X** and then click the **Save** button, as shown

# Understanding the Add-in Template Visually

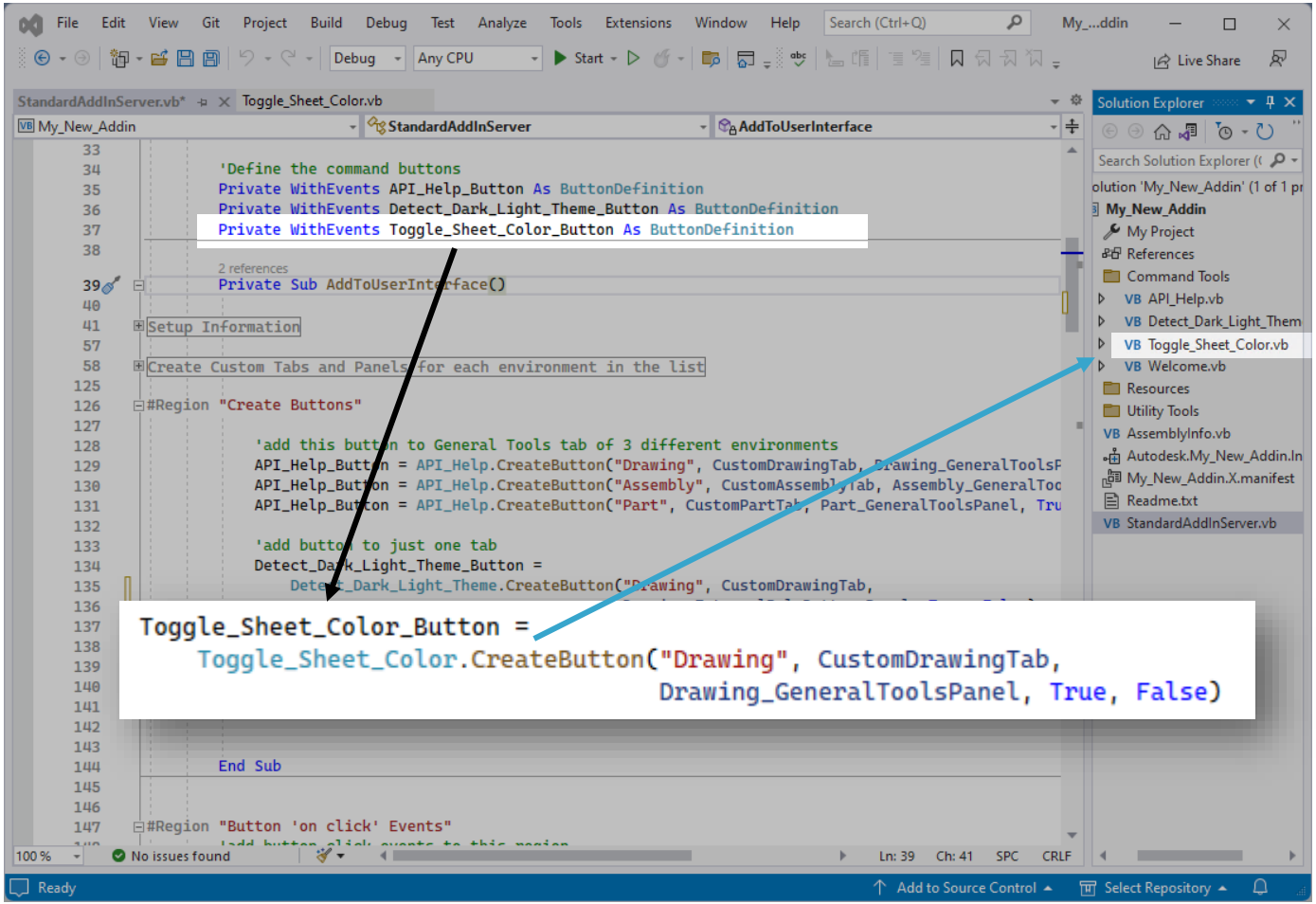


# Modifying the Add-in



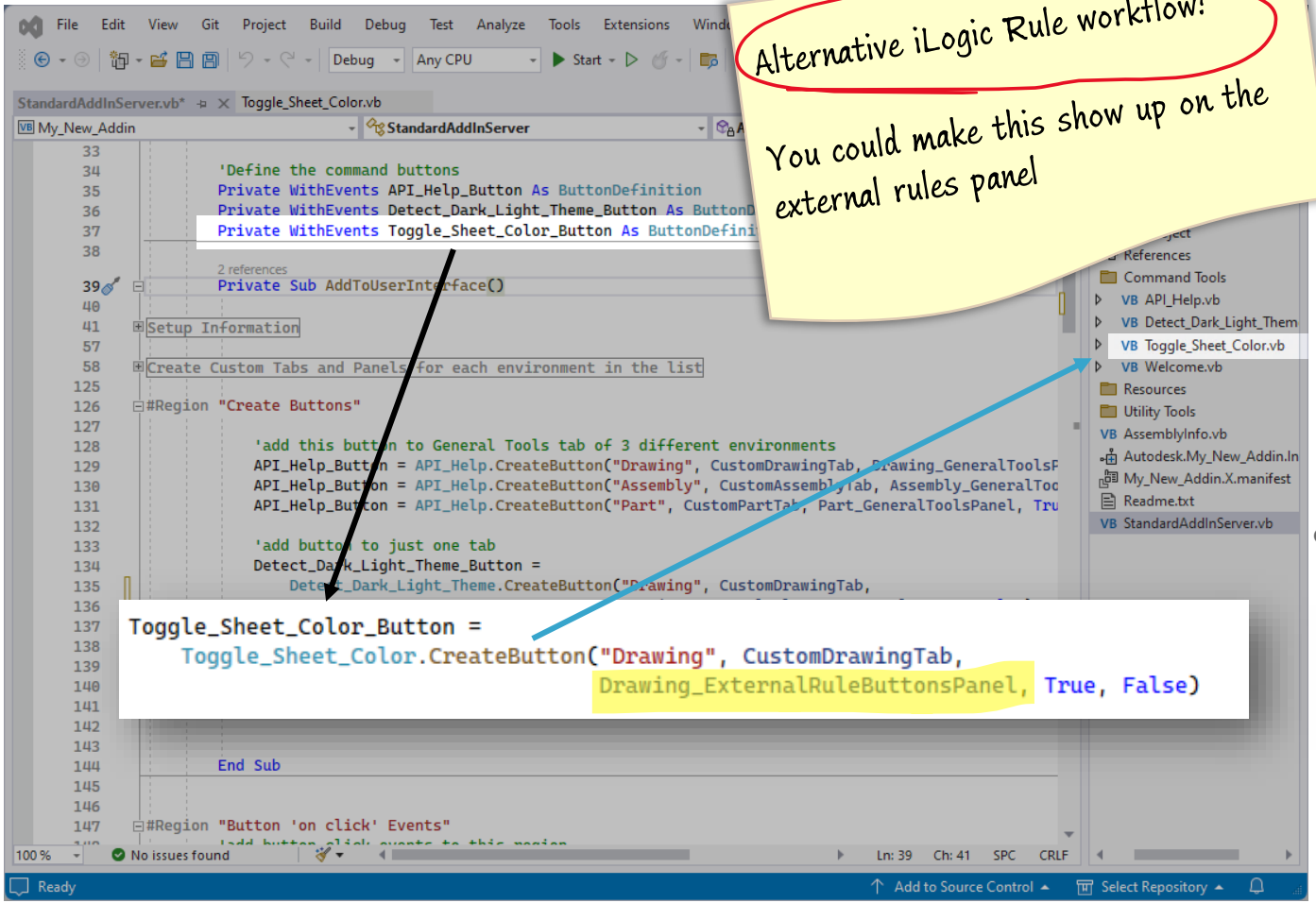

- In the Create Buttons region of the StandardAddinServer…
- We'll add a line to connect the button definition we created earlier, with the Function in the module we just created.
- This line of code is run when Inventor starts up and loads the add-in

# Modifying the Add-in



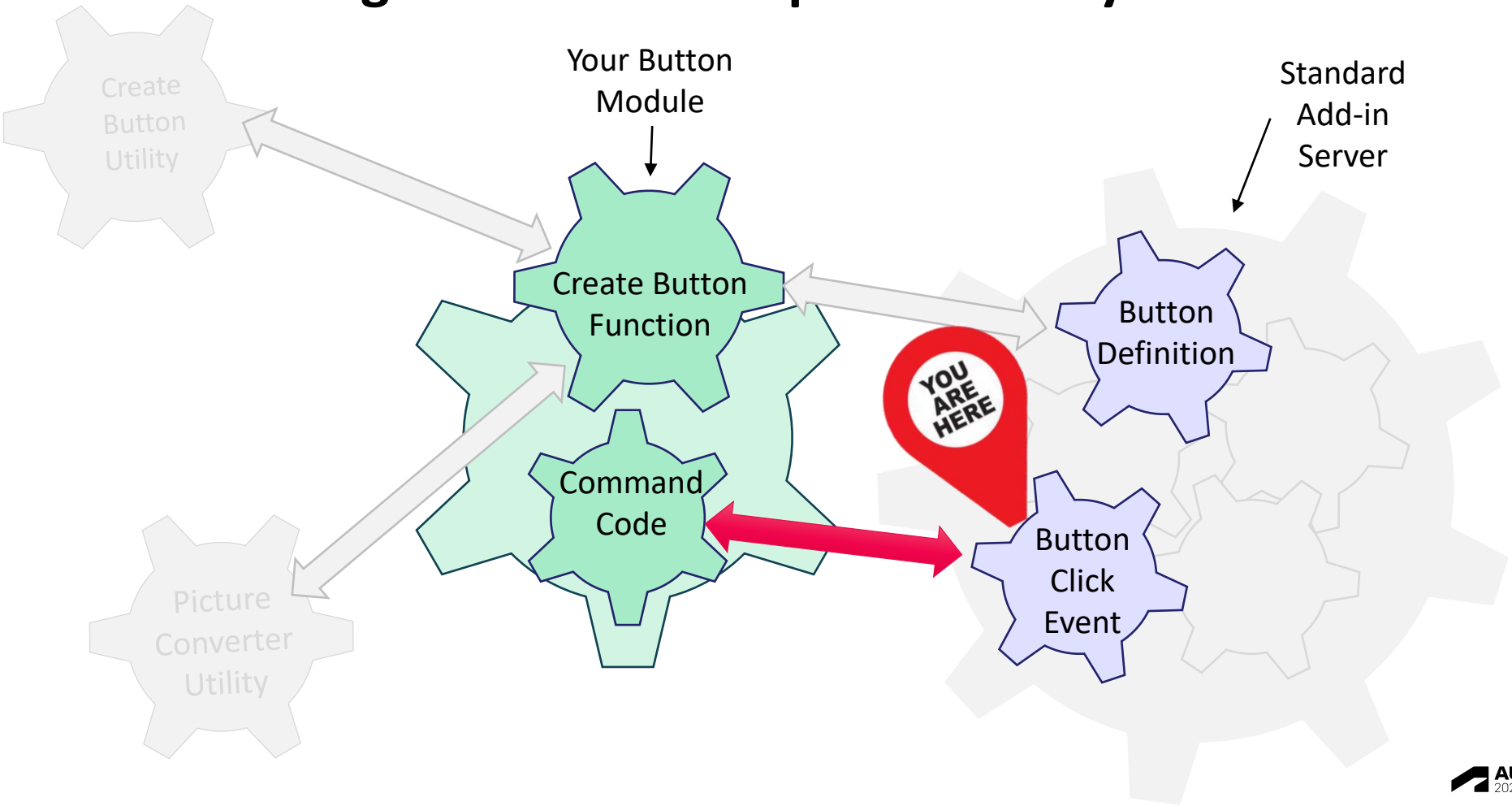

- We'll add a line to connect the button definition we created earlier, with the Function in the module we just created.
- This line of code is run when Inventor starts up and loads the add-in

# Modifying the Add-in

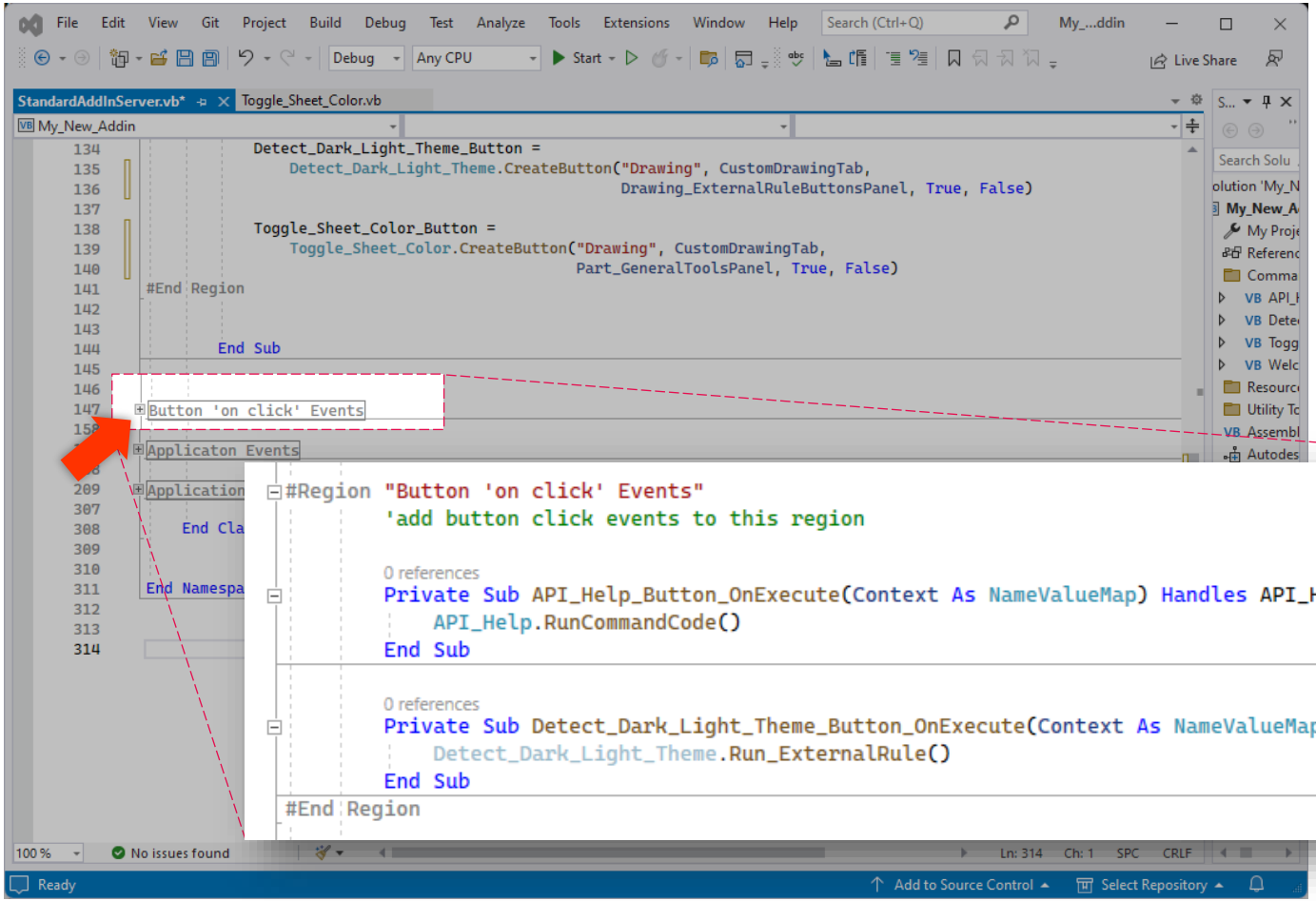



- We'll add a line to connect the button definition we created earlier, with the Function in the module we just created.
- This line of code is run when Inventor starts up and loads the add-in

# Understanding the Add-in Template Visually

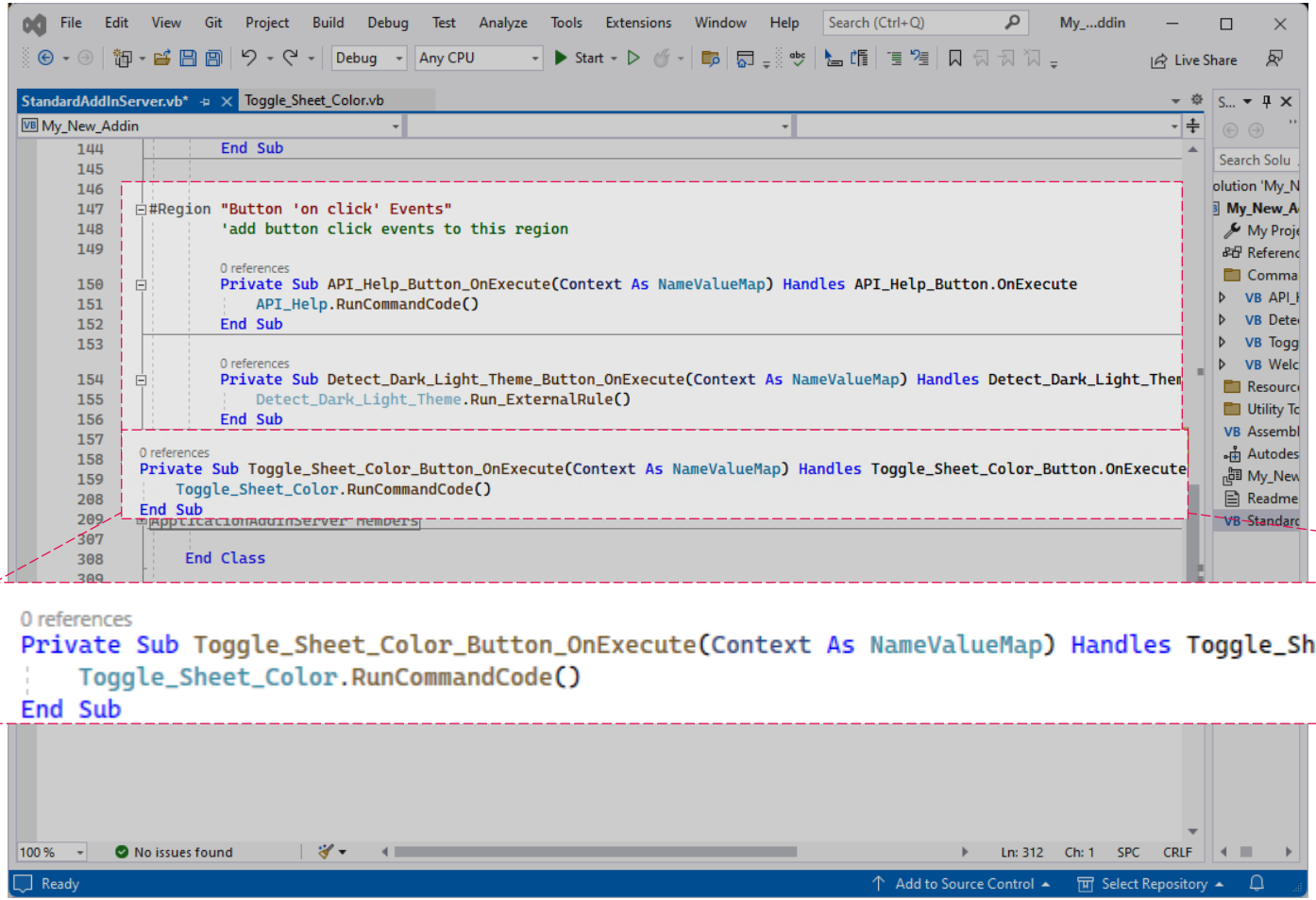


# Modifying the Add-in




- Next we'll expand the **Button 'on click'** region by clicking the **+** sign

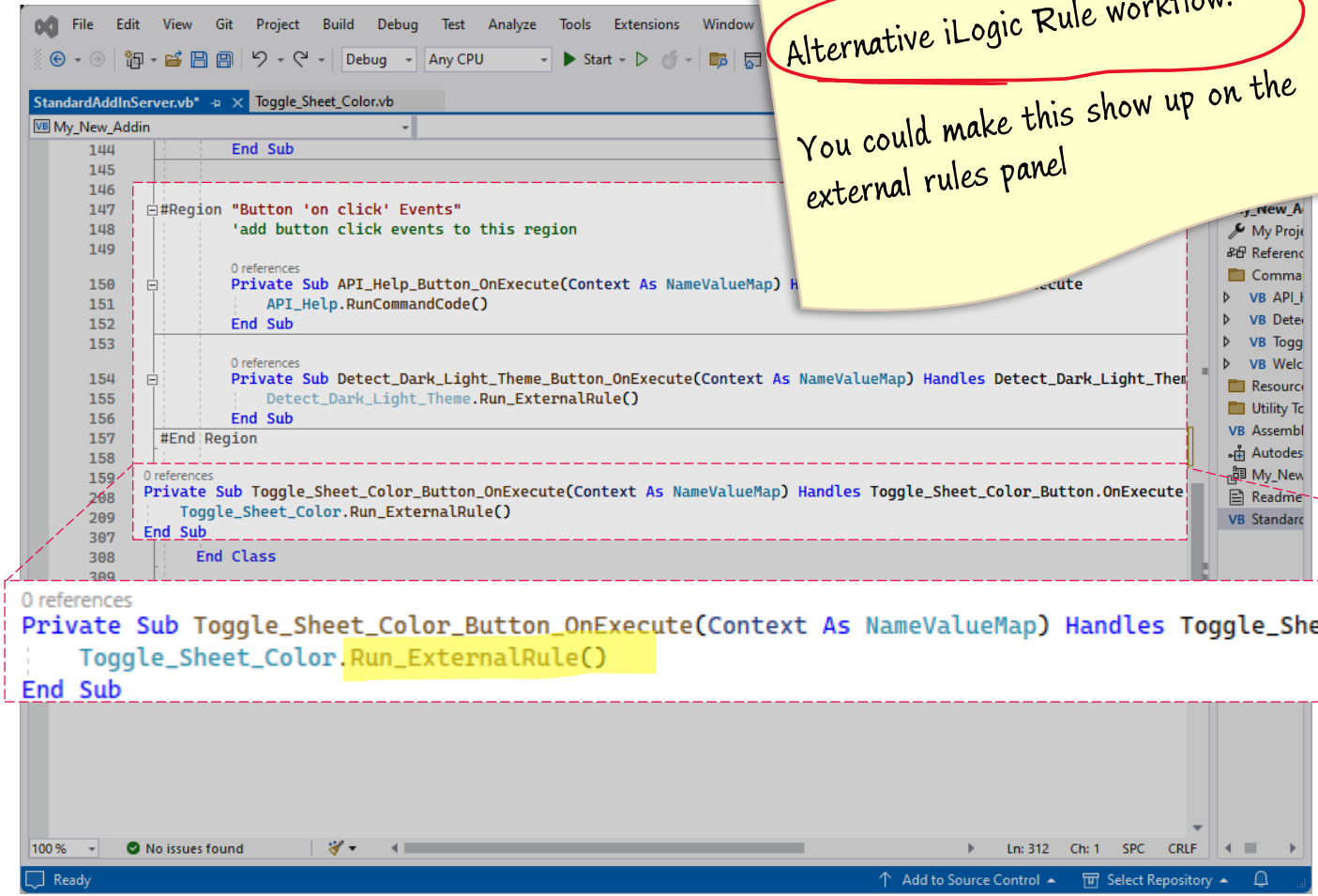
# Modifying the Add-in




```
0 references
Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
    Toggle_Sheet_Color.RunCommandCode()
End Sub
```

- Add a sub to run the Toggle_Sheet_Color Module command code wen the button is executed

# Modifying the Add-in



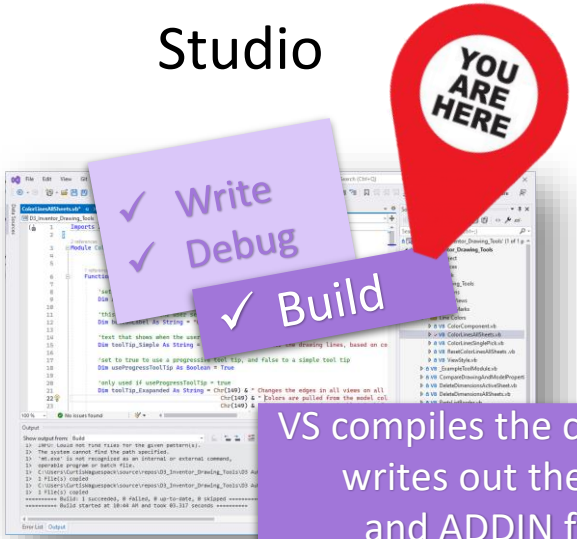

- If you wanted this to run an external ilogic rule, you could use this alternative

# Overall Process

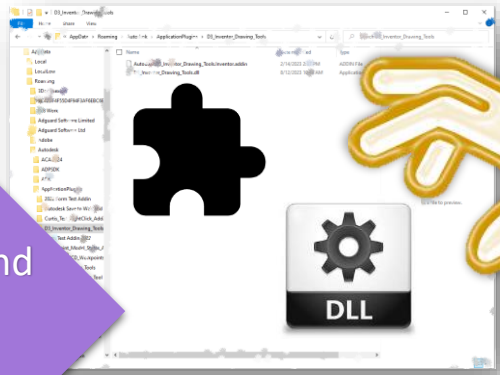
Visual Studio

Autodesk Application Plugins Folder

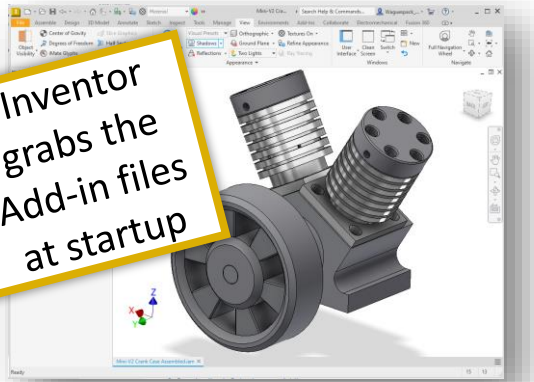
Autodesk Inventor



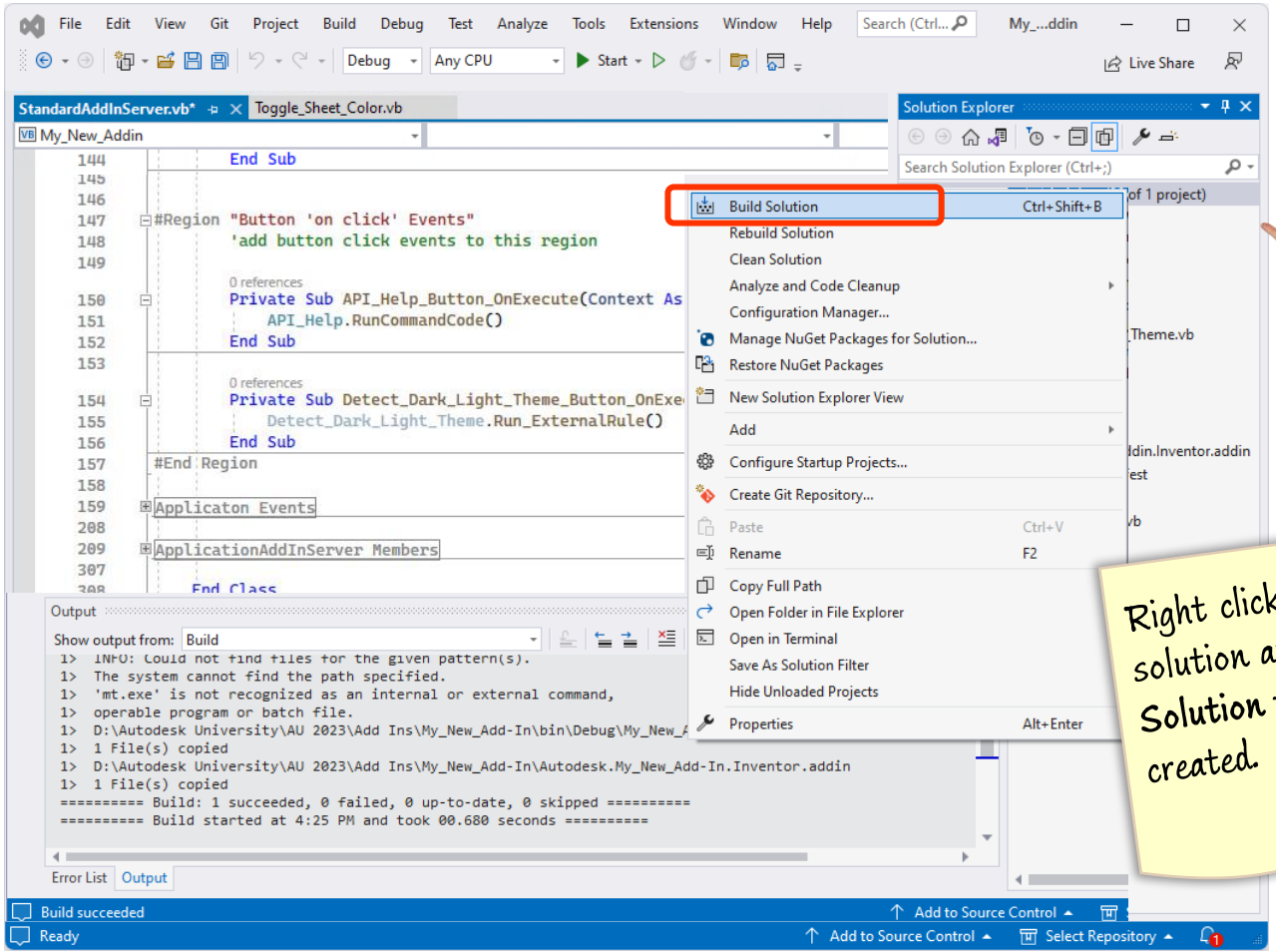

VS compiles the code and writes out the DLL and ADDIN files




Inventor grabs the Add-in files at startup



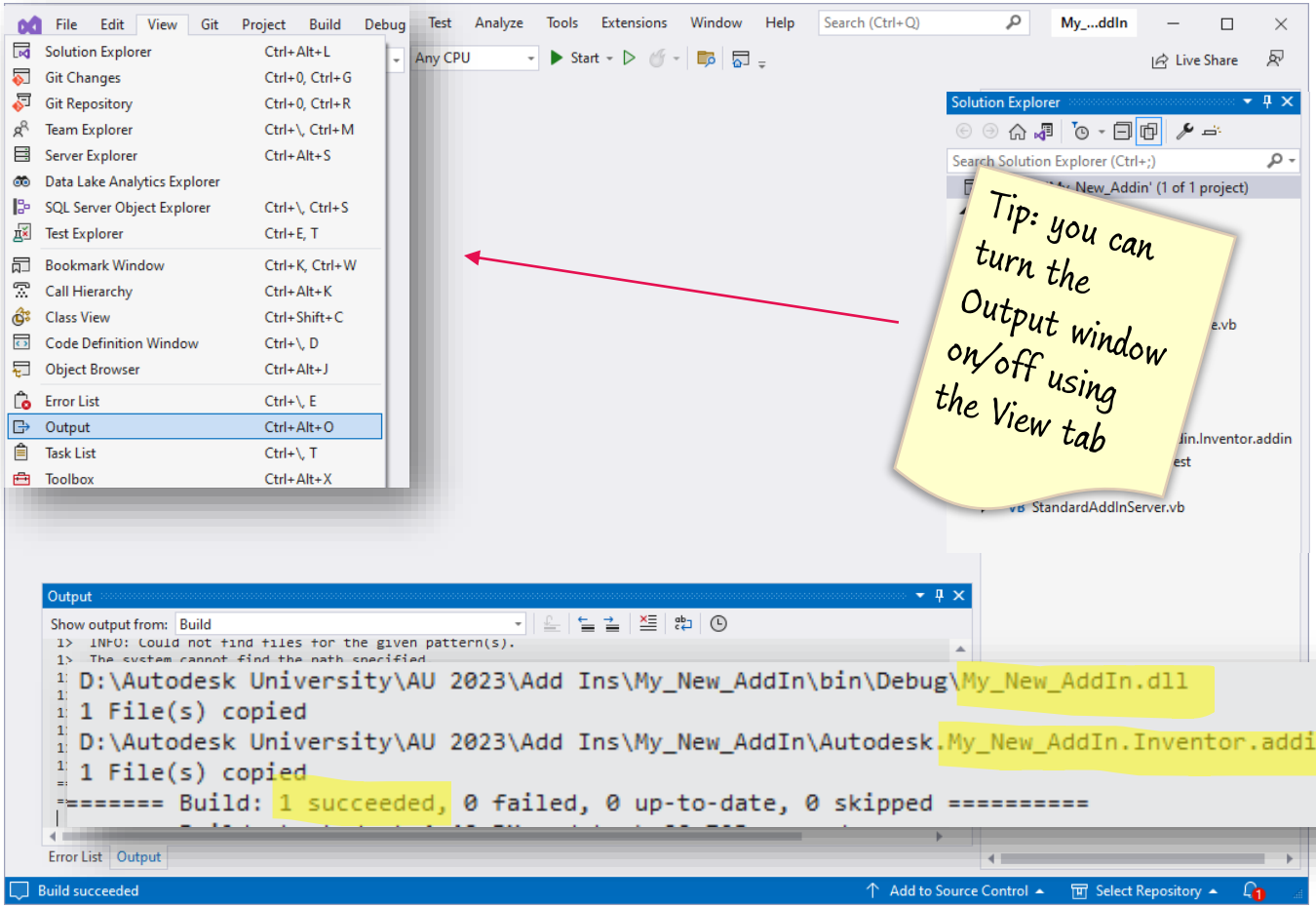
# Compiling the Add-in




- The solution is shown in the Solution Explorer

Right click on the solution and choose **Build Solution** to see the files created.

# Compiling the Add-in

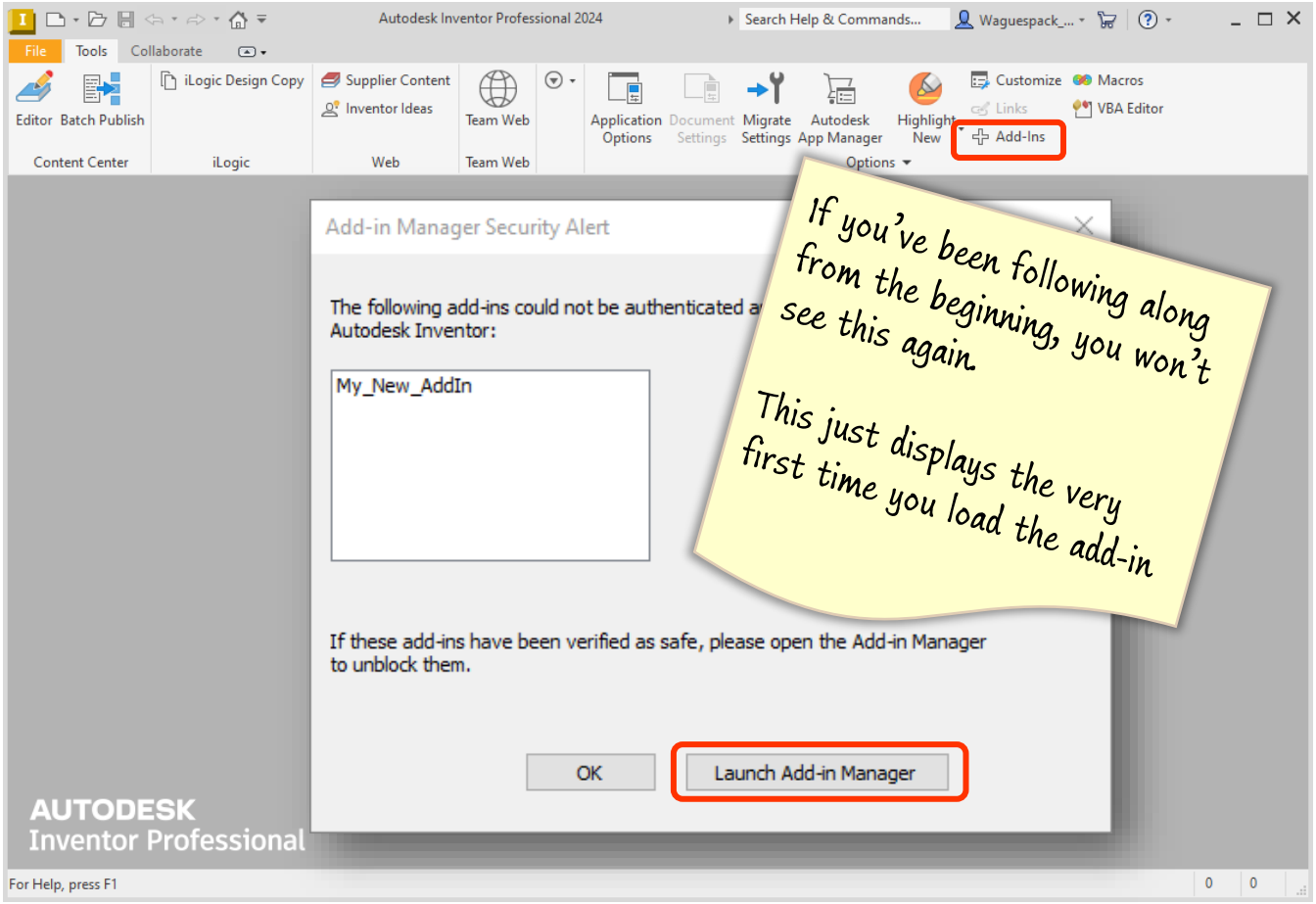



- The Build results are shown in the Output window

# Testing the add-in

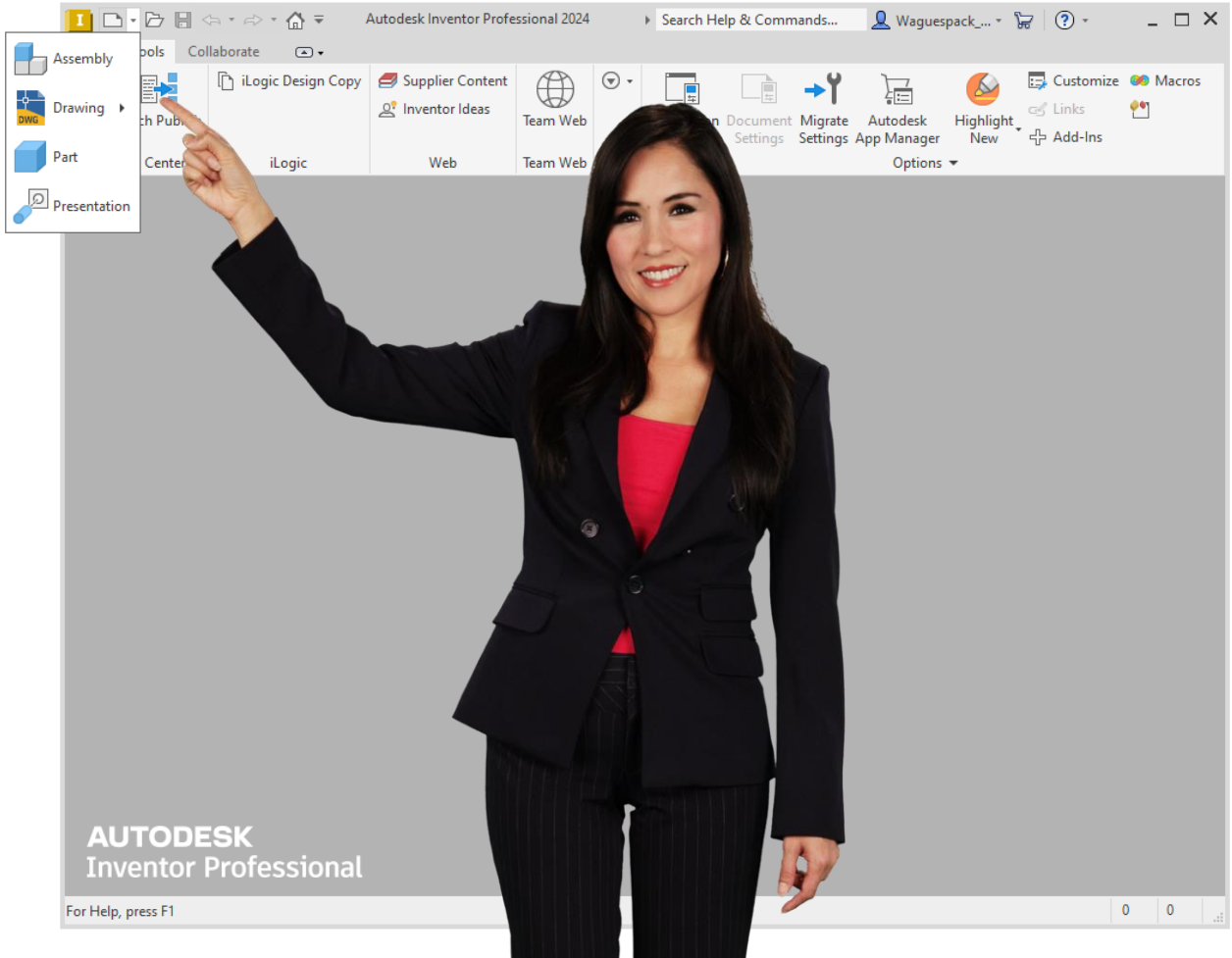
Working in Inventor

# Testing the Add-in



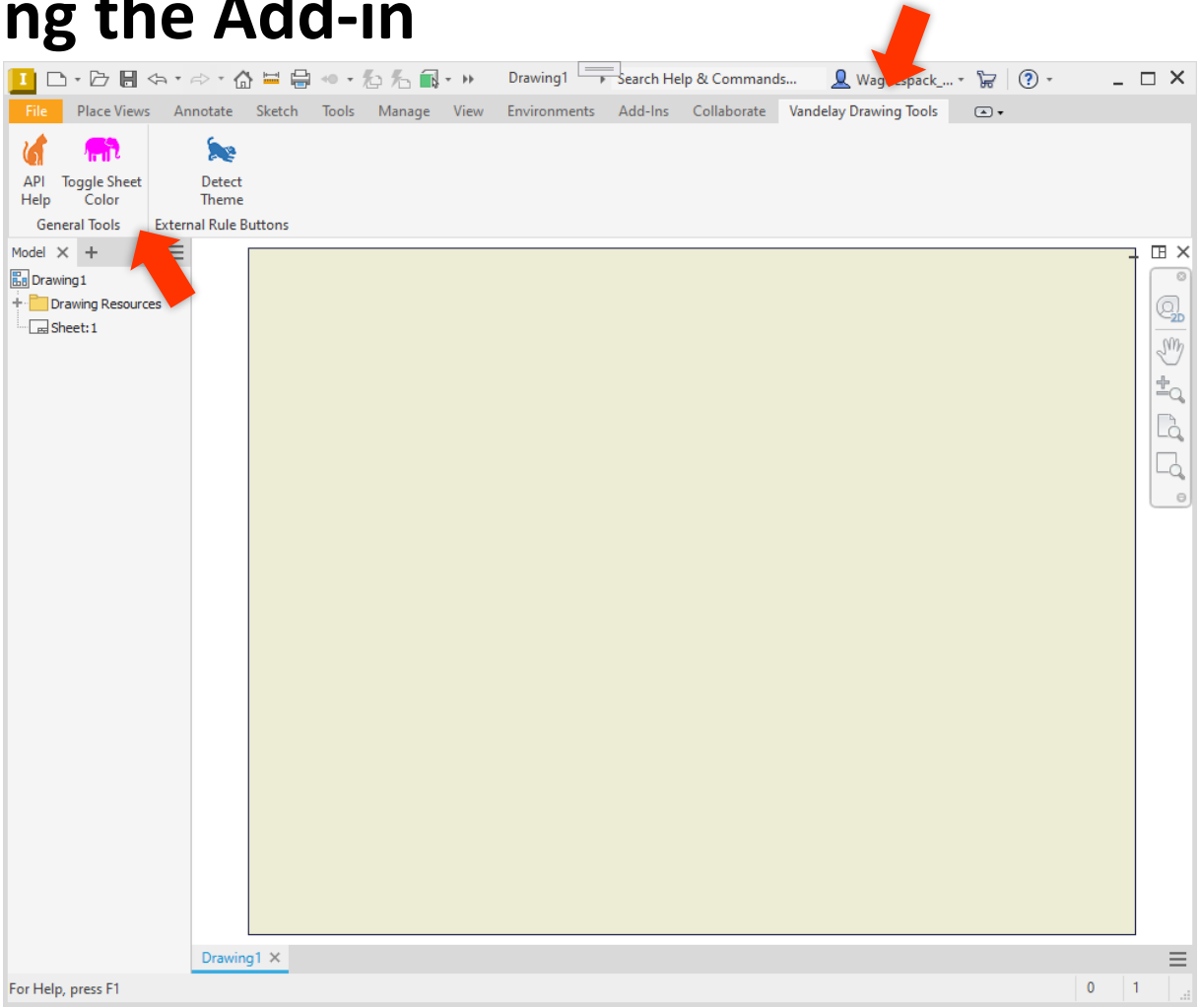

- Open Inventor.
- Note that you will be greeted by a message informing you that your add-in was blocked from loading
- Click the Launch Add-In Manager button or got to Tools > Add-ins

# Using the Add-in



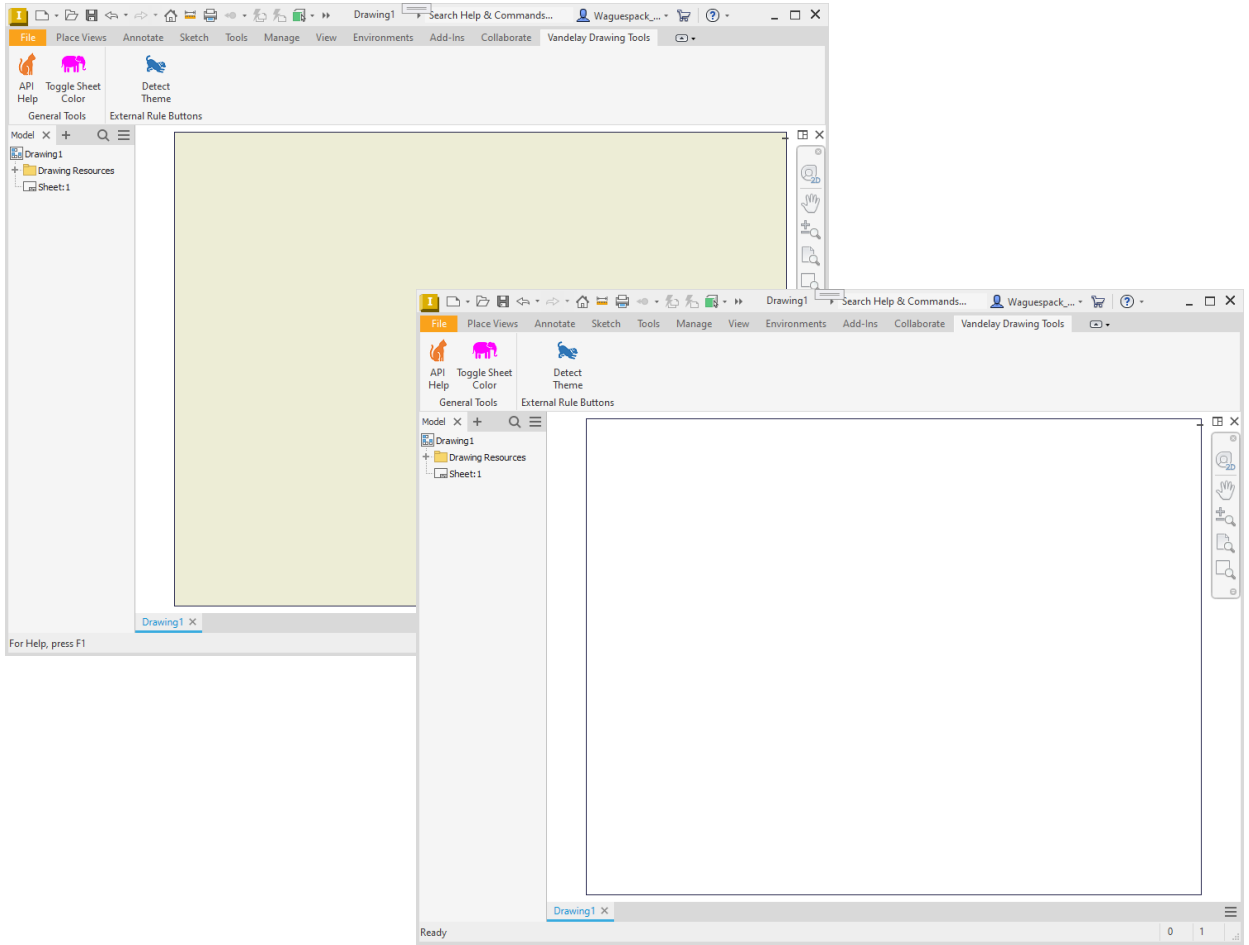

- Create a new drawing file from a template

# Using the Add-in



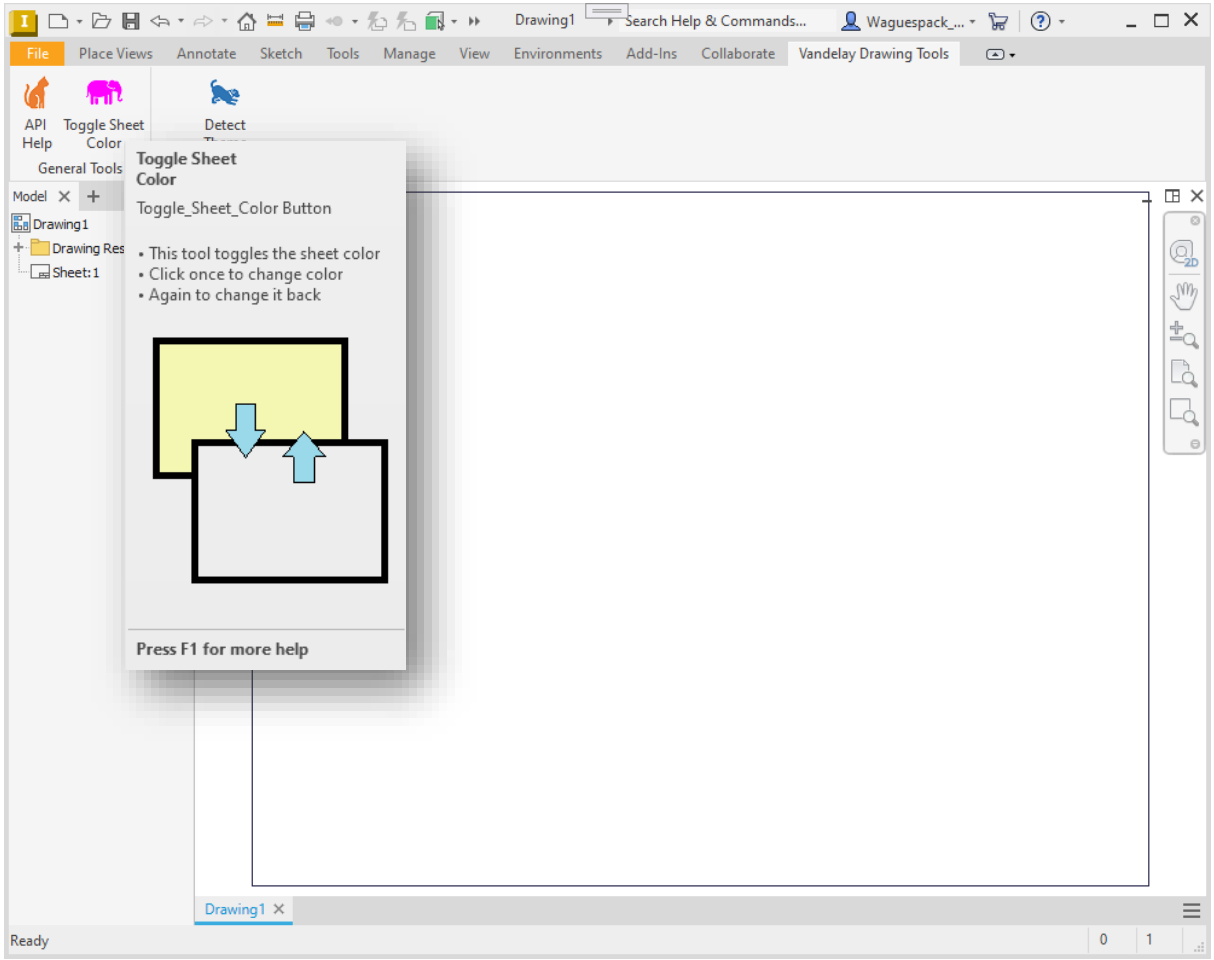

- Switch to the **Vandelay Drawing Tool** tab
- Note the new button on this tab

# Using the Add-in




- When you click the button the code in the Toggle_Sheet_Color module will execute
- And the sheet color will toggle from the default color to white

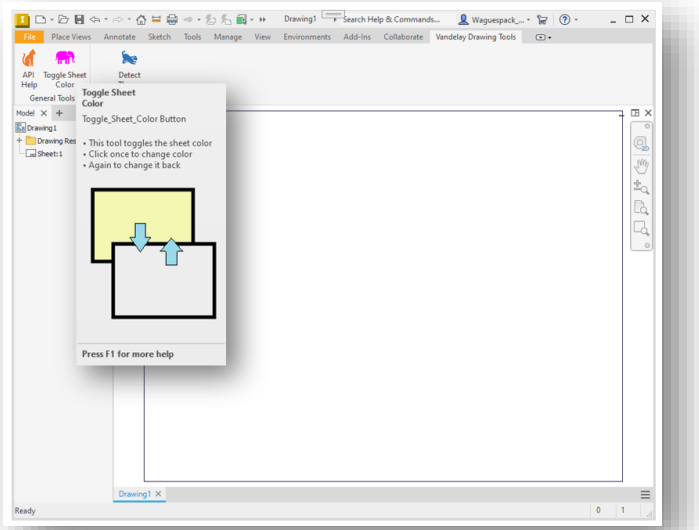
# Using the Add-in




- Finally hover over the new button to see the expanded tooltip

# Goal: be able to create new Inventor add-ins in just minutes!

## You did it!



# Quick Review

- Copy an existing Command Module to create a new one
- Open the module and change the module name
- In the CreateButton function:
  - Add button icons to the Resources
  - Set the button icons
  - Set the button label name
  - Set the simple tooltip
  - Set the progressive tooltip is desired
- In the Sub (where the code that does the real work resides )
  - Add the code
  - Alternatively add the line to run and external ilogic rule

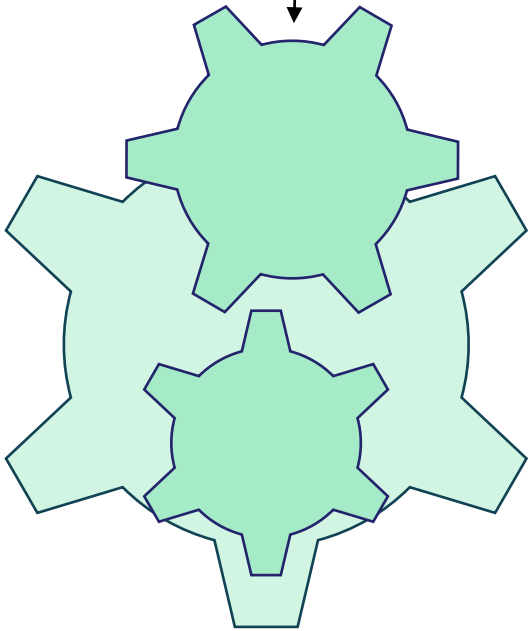
- In the StandardAddInServer
- Add the “WithEvents” line at the top
  - This creates the button definition name
- In the Create Buttons region:
  - Add a line to “wire up” the button definition name to the command module
- In the "Button 'on click' Events“ region:
  - Add an OnExecute sub to call the code in the module when the button is clicked

# Review

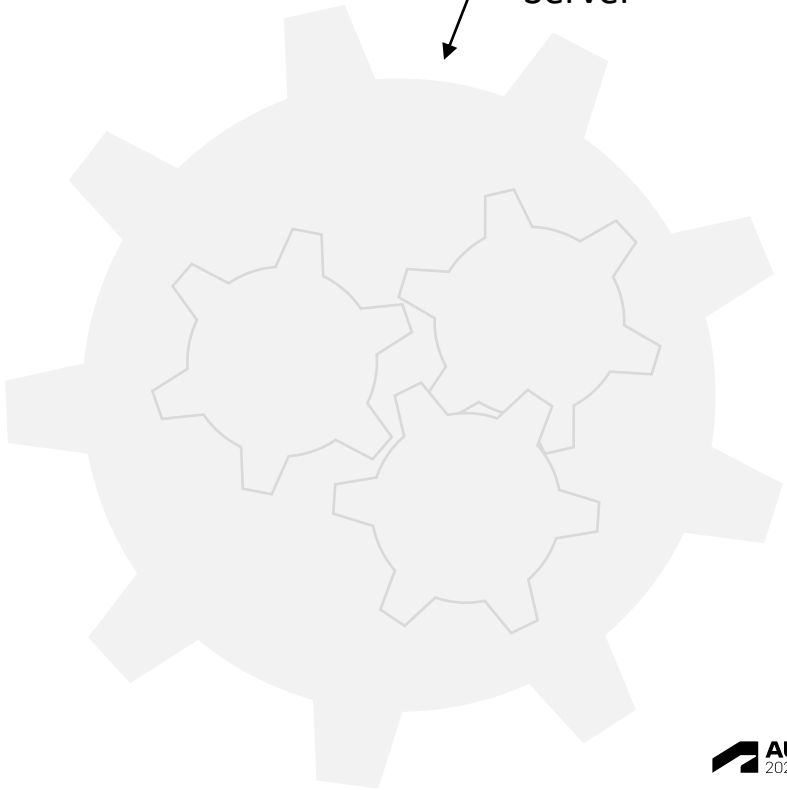
Create Button Utility

Picture Converter Utility

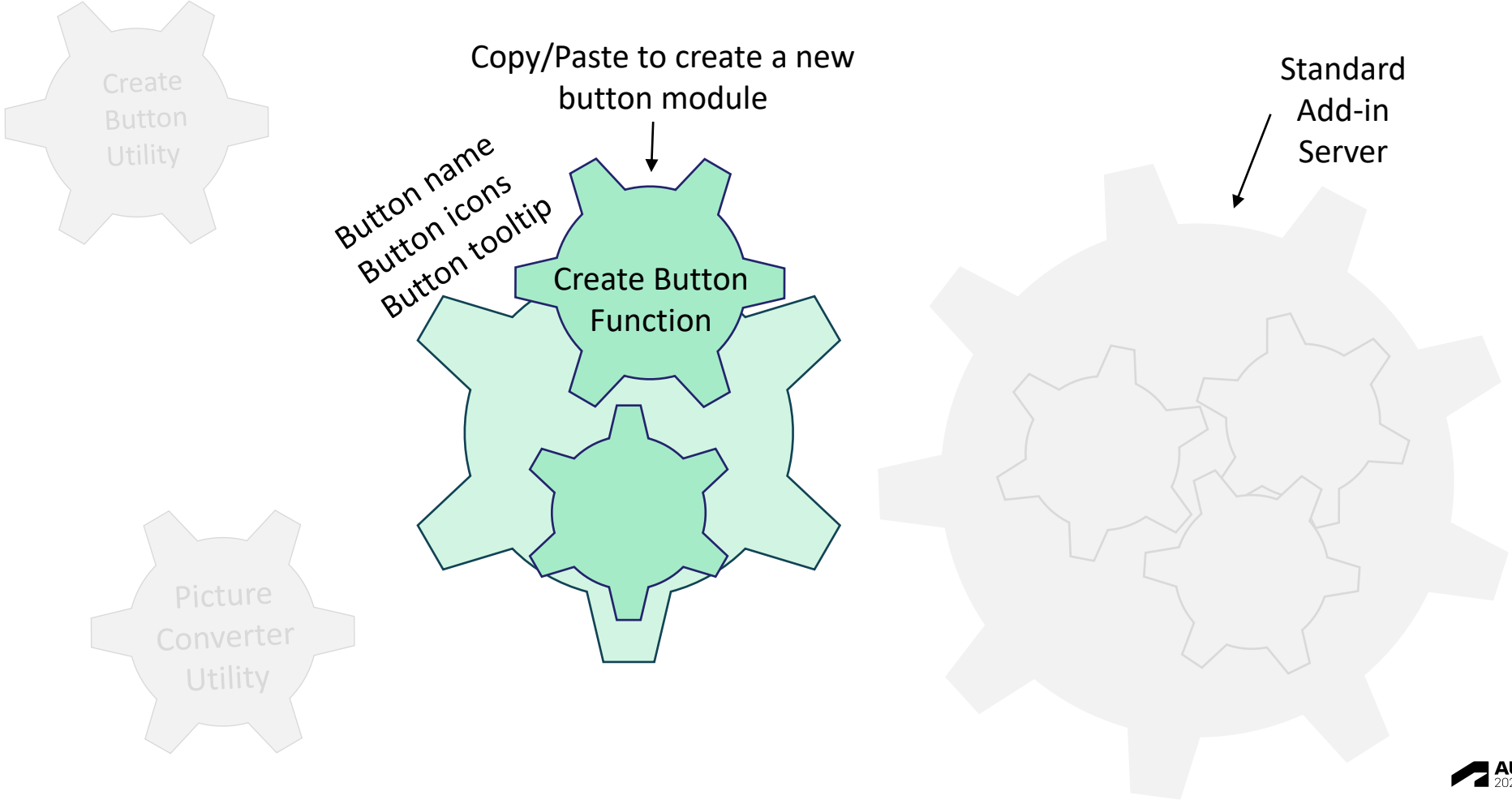
Copy/Paste to create a new button module



Standard Add-in Server



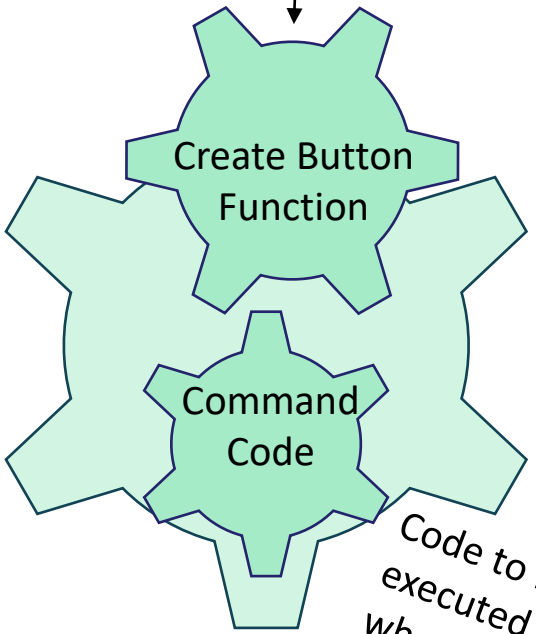
# Review


Create Button Utility
Copy/Paste to create a new button module
Button name
Button icons
Button tooltip
Create Button Function
Standard Add-in Server
Picture Converter Utility

# Review

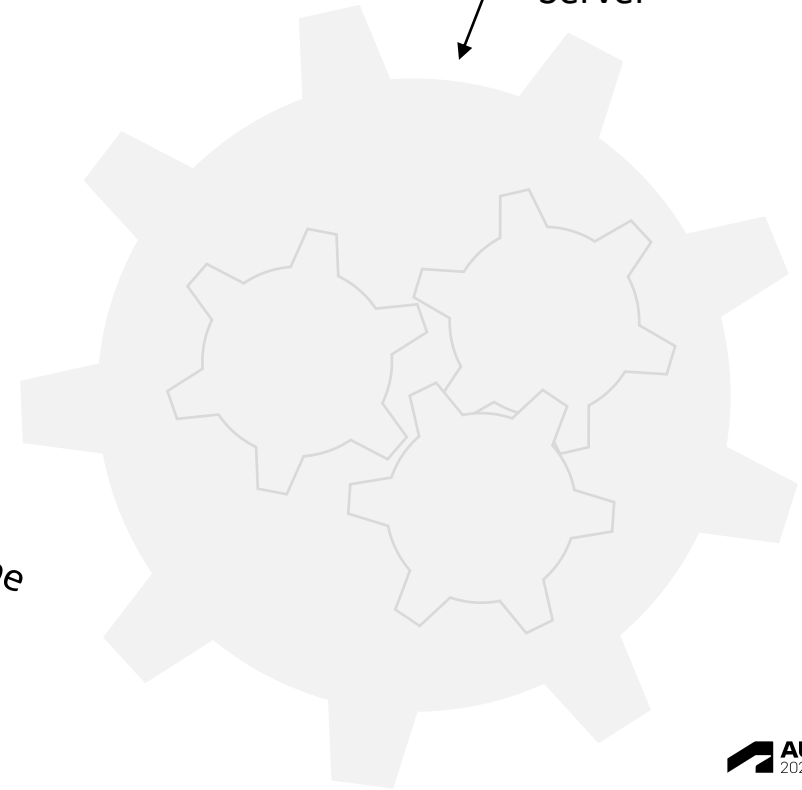



Copy/Paste to create a new button module

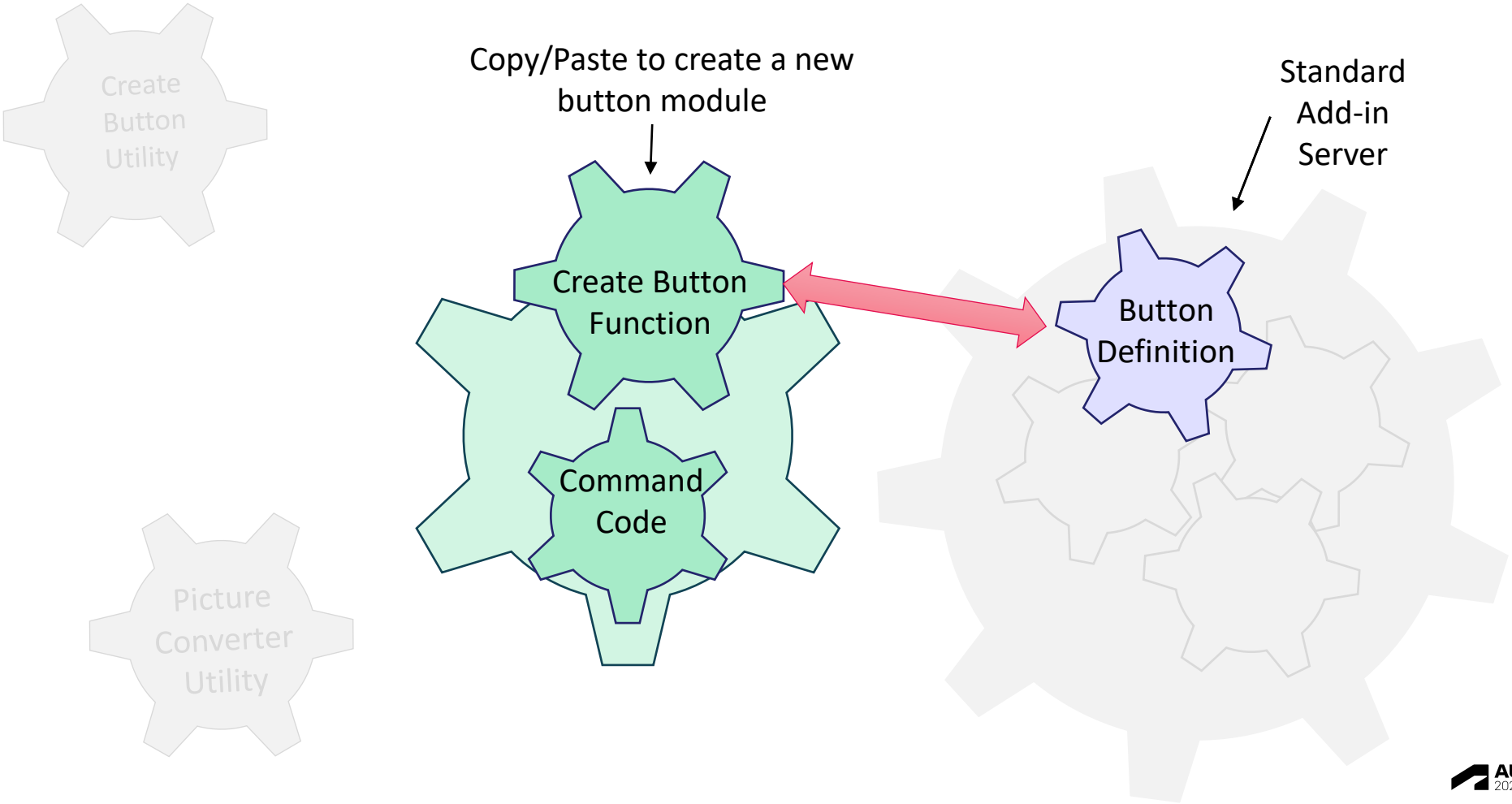



Code to be executed when the button is clicked

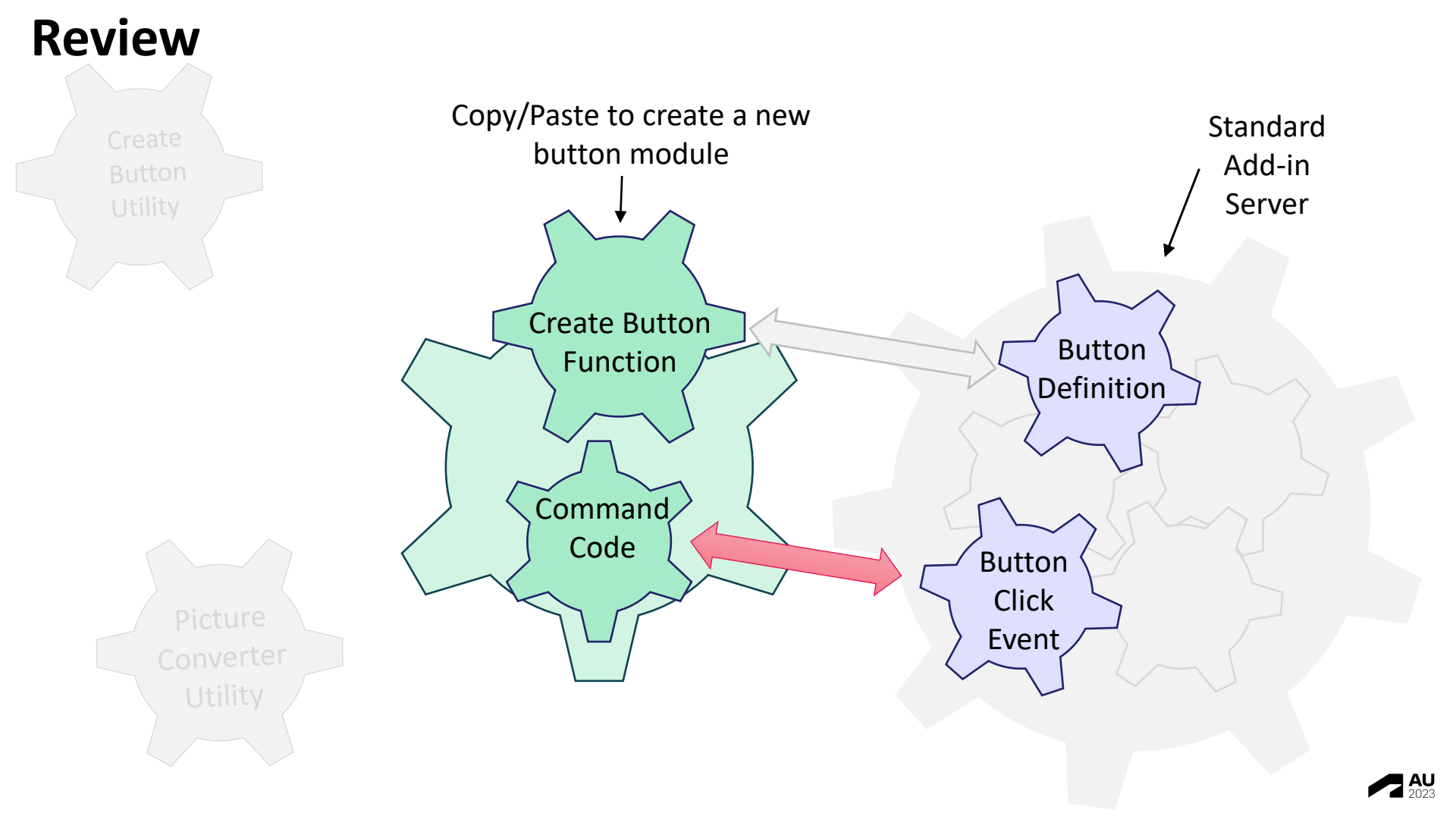
Standard Add-in Server



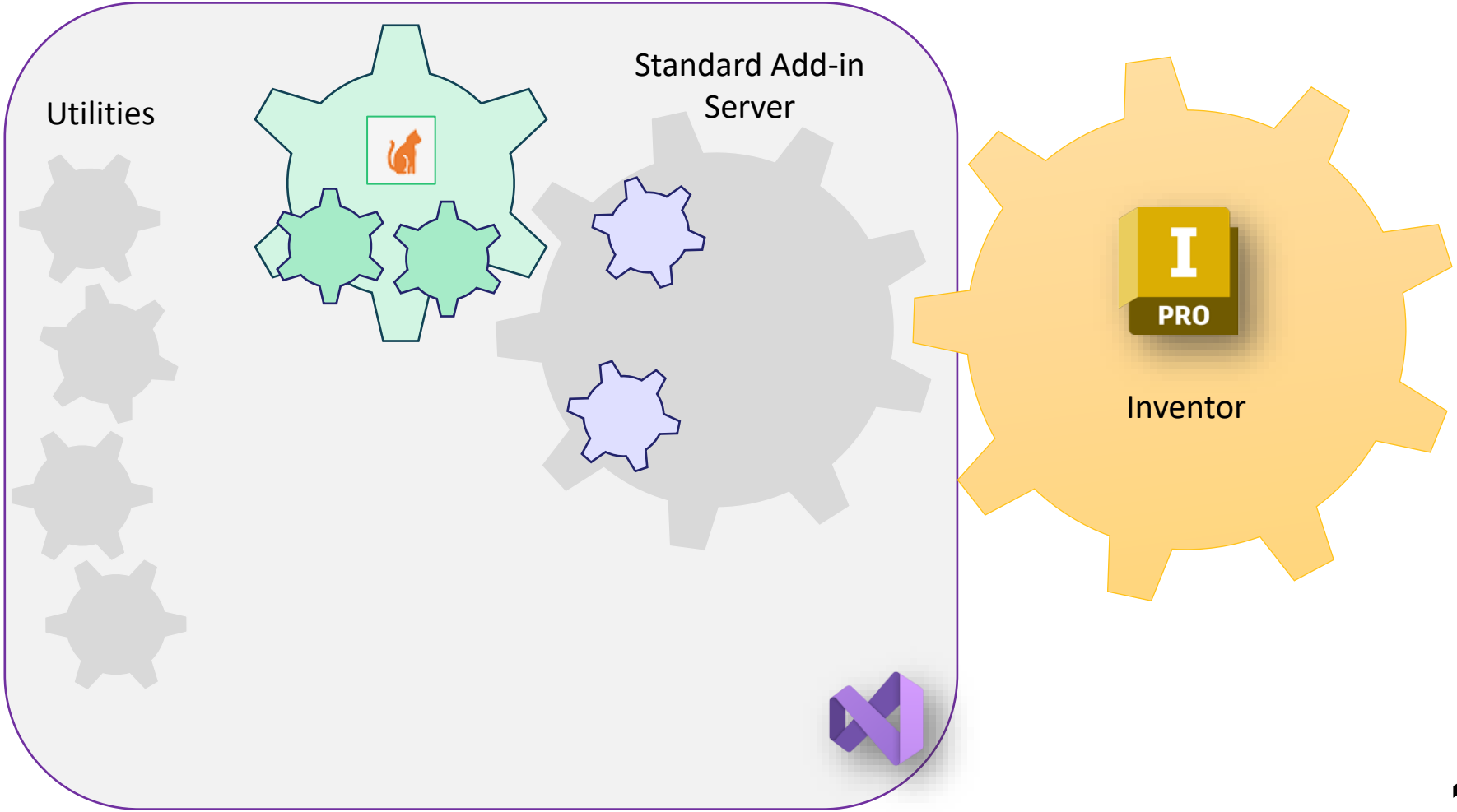
# Review


Create Button Utility
Copy/Paste to create a new button module
Standard Add-in Server
Create Button Function
Button Definition
Command Code
Picture Converter Utility

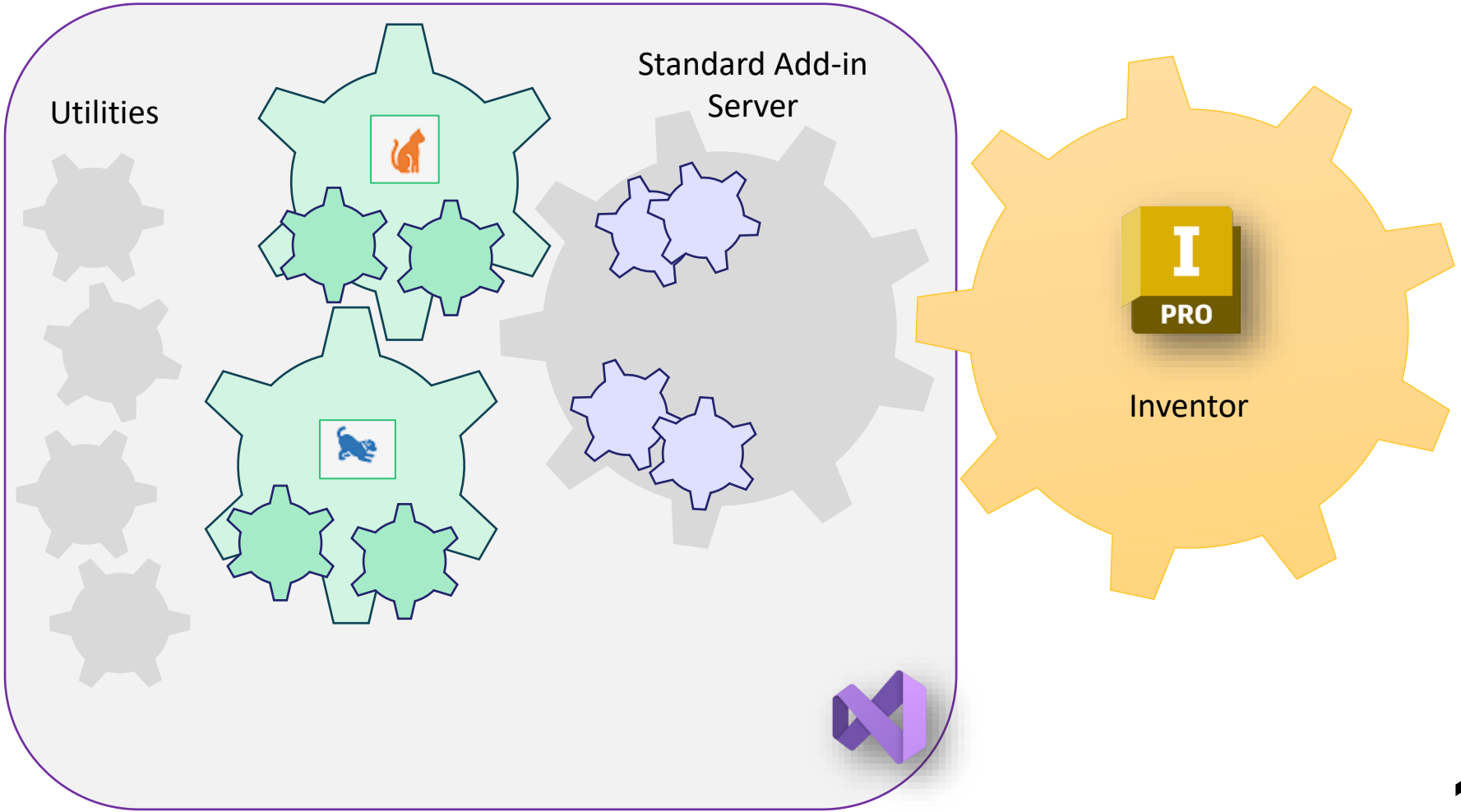
# Review


Create Button Utility
Copy/Paste to create a new button module
Standard Add-in Server
Create Button Function
Command Code
Button Definition
Button Click Event
Picture Converter Utility

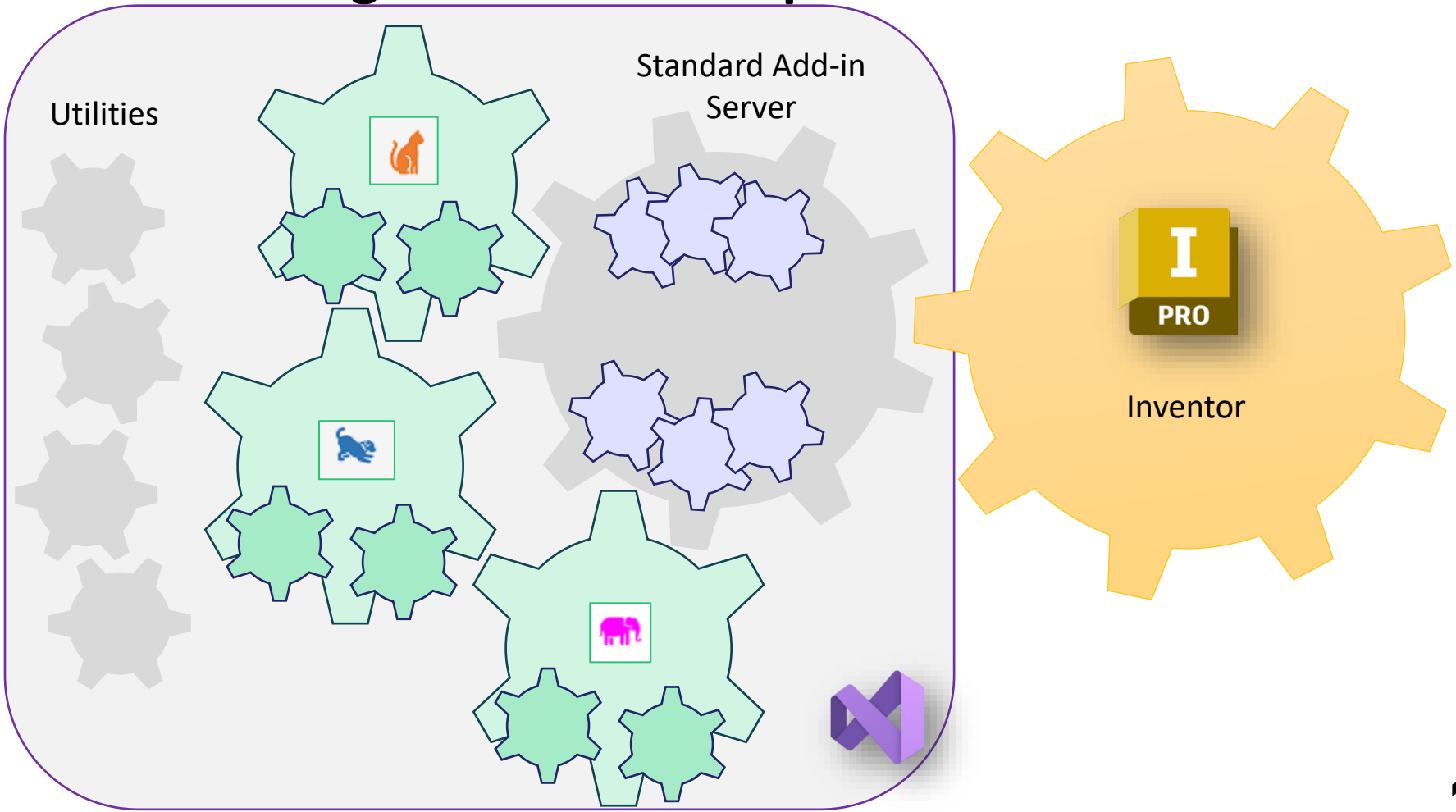
# Review



# Review



# Understanding the Add-in Template Structure



Thank you!


THE DESIGN & MAKE CONFERENCE