

MFG601910

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

Curtis Waguespack

Team D3

### Learning Objectives

- Create simple add-ins using your existing iLogic rules.
- Demystify the “how to get started” process of Inventor automation add-ins.
- Learn to create a richer user interface for your automation tools.
- Better understand the Inventor add-in tools, templates, and techniques.

### Description

You've seen how iLogic automation can tame tedious and error-prone tasks and make you and your fellow Inventor users more efficient.

But have you also run up against the limitations present within the automation tools offered with iLogic? Have you struggled to take things to the next level and create Inventor add-ins? Are you concerned about having to start over?

You're not alone. Making the transition from iLogic is a difficult transition for many busy professionals who want to take their automation further but find themselves sticking with what is known, what is familiar, and what is already in place.

Let's have a look at what you need to get started on a path forward that allows you to leverage the familiarity of your existing automation while learning the landscape of the Inventor add-in.

### Author

Curtis has used Autodesk Inventor and AutoCAD for over two decades to design a wide range of manufactured products. In addition to his design experience, Curtis has authored and co-authored multiple editions of the Mastering Autodesk Inventor book and has taught Inventor to professionals in the classroom. Currently, Curtis works as an Automation Solutions Consultant, where his focus is on automation within Autodesk Inventor and helping others in the Inventor community understand and utilize iLogic and Inventor API automation. Curtis creates Inventor automation tools that help companies become more efficient with their design efforts, and he teaches iLogic classes as well.

## Getting Started Steps

<b>Installing Visual Studio.....</b>	4
Determine the version to use .....	4
<b>Unpack the Inventor SDK files.....</b>	5
Examine the DevelopersTools folder.....	6
Examine the Visual Studio Templates .....	7
<b>Ensure the Inventor Add-In Templates show up in Visual Studio.....</b>	8
Ensure the Inventor Add-In Templates show up in Visual Studio .....	8
Move the *.zip files into the standard folders .....	11
Ensure the templates show up .....	12
<b>Downloading the custom add-in template.....</b>	14

These are the three general steps to getting started Visual Studio and the Inventor add-ins templates. This document will take you through these steps.

**1**



**Install  
Visual  
Studio**

**2**



**Unpack  
the  
Inventor  
SDK**

**3**



**Ensure the  
Inventor  
Add-in  
Templates  
are Loaded**

## Installing Visual Studio

Go to the link below and download and install Microsoft Visual Studio.

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

### Determine the version to use

You will need to determine the version of Visual Studio to download and use.

- If your intention is to use it for educational purposes, you can use the Community version.
- If you are intending to use it for production or profit you will need to use the Professional version.

#### 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

## Unpack the Inventor SDK files

The Inventor SDK ( Software Developer Kit ) installs with Inventor automatically. However, you will need to “unpack” the contents of the **developertools.msi** file to use the SDK contents.

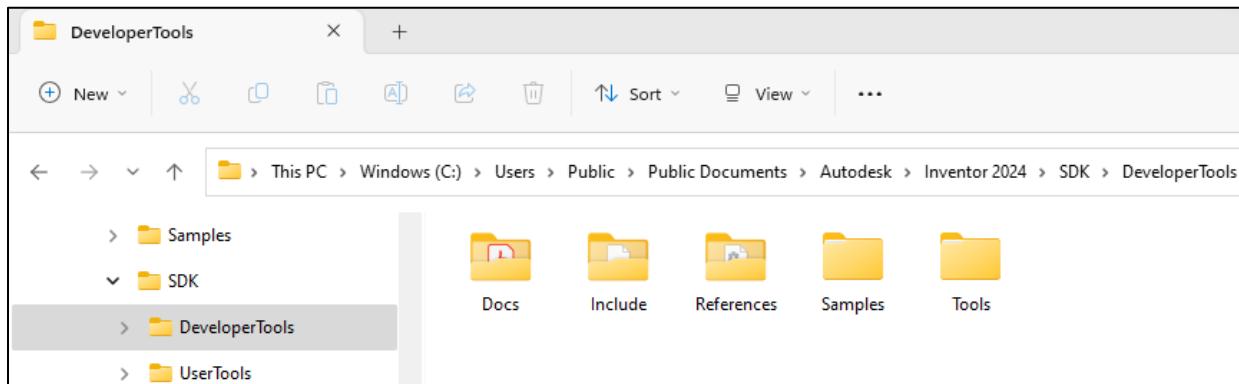
- In Windows Explorer go to:
  - C:\Users\Public\Documents\Autodesk\Inventor 2024\SDK
- Double-click the **developertools.msi** file
- Click the **Next** button
- Check the **Accept Terms** box
- Click the **Next** button
- Choose the path
- Click the **Next** button
- Click the **Install** button
- Click the **Finish** button

*Note that you need to have Visual Studio Installed before you unpack the developertools.msi file.*

## Examine the DevelopersTools folder

You will now have a DevelopersTools folder containing the subfolders shown.

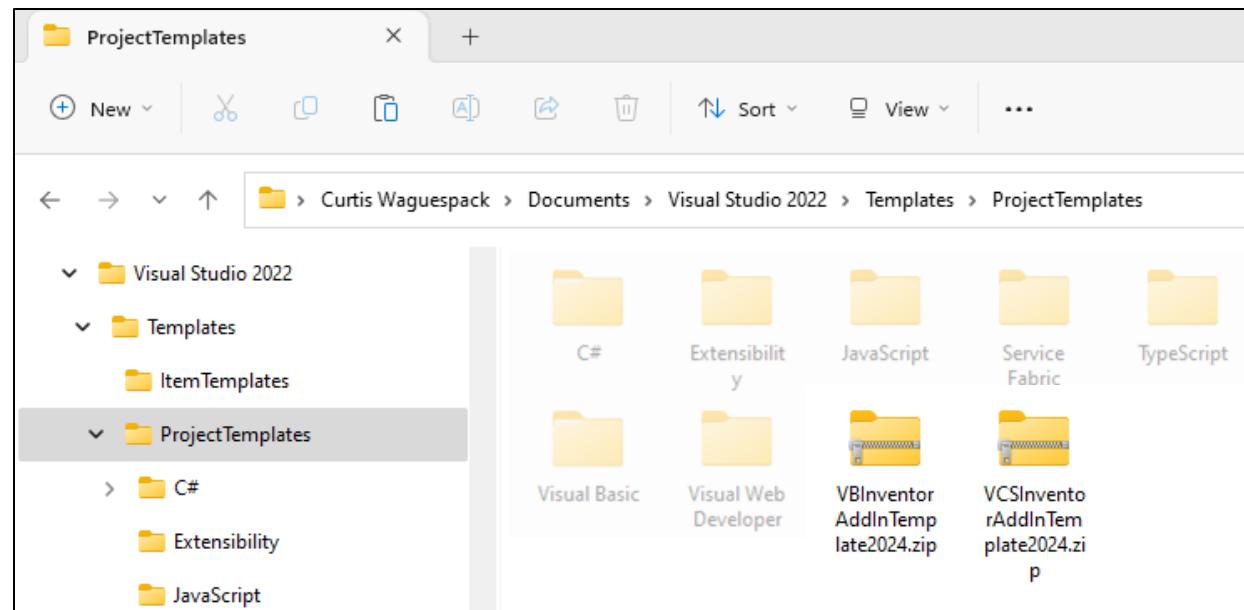
**C:\Users\Public\Documents\Autodesk\Inventor 2024\SDK\DevelopersTools**



## Examine the Visual Studio Templates

In addition to the DevelopersTools folder the developertools.msi will “unpack” some Visual Studio template files. Typically, the location of these templates something like this:

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



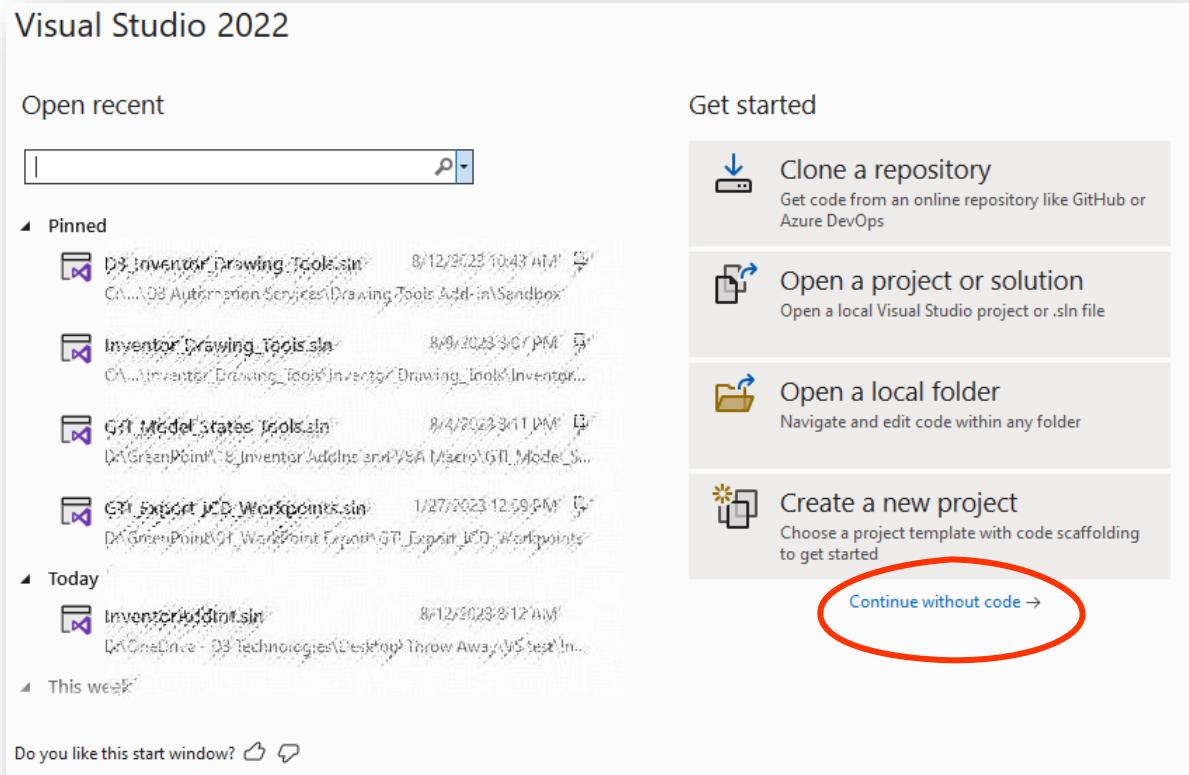
## Ensure the Inventor Add-In Templates show up in Visual Studio

In the following section you will ensure the Inventor Add-in templates are loaded in Visual Studio

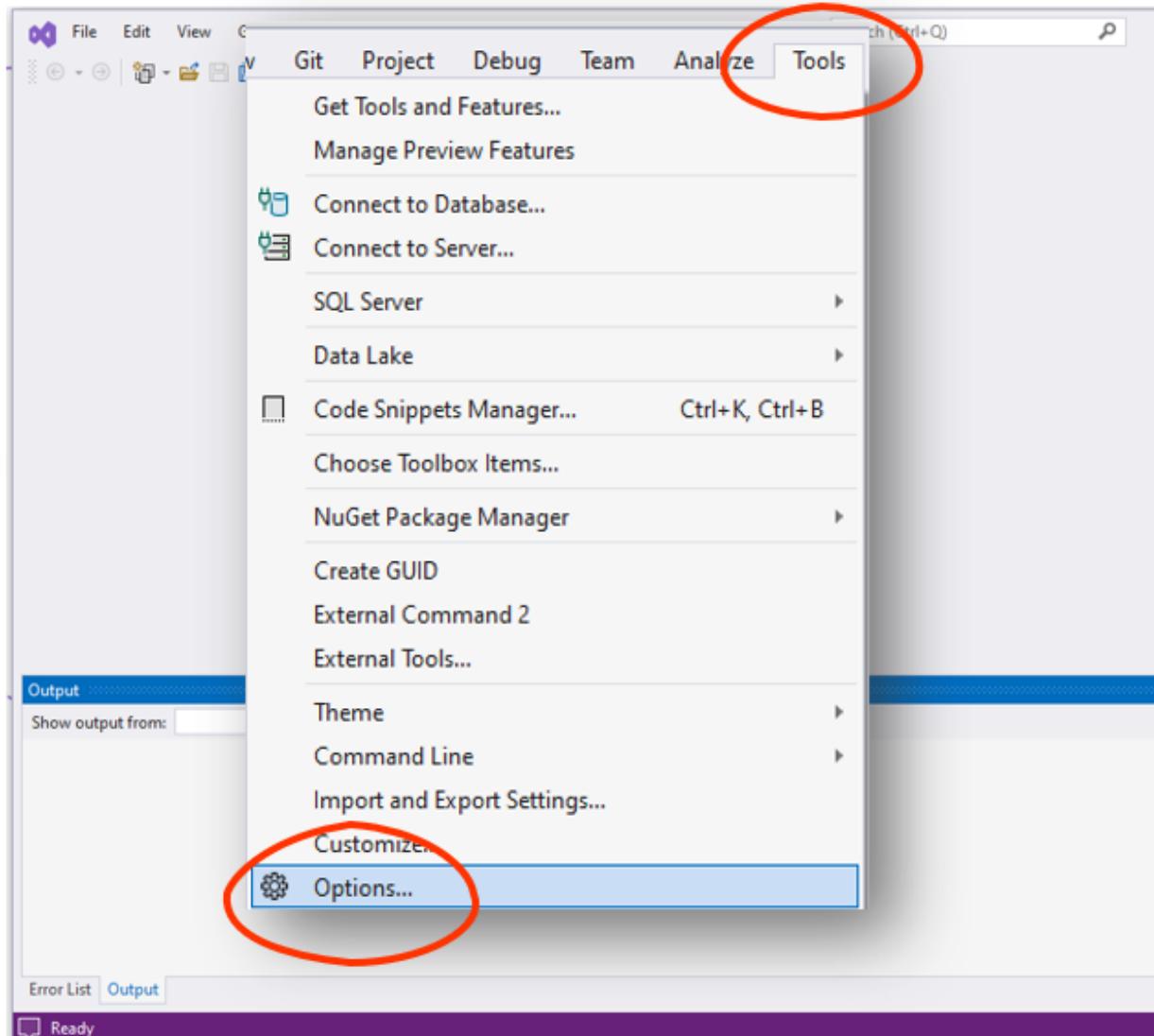
### Ensure the Inventor Add-In Templates show up in Visual Studio

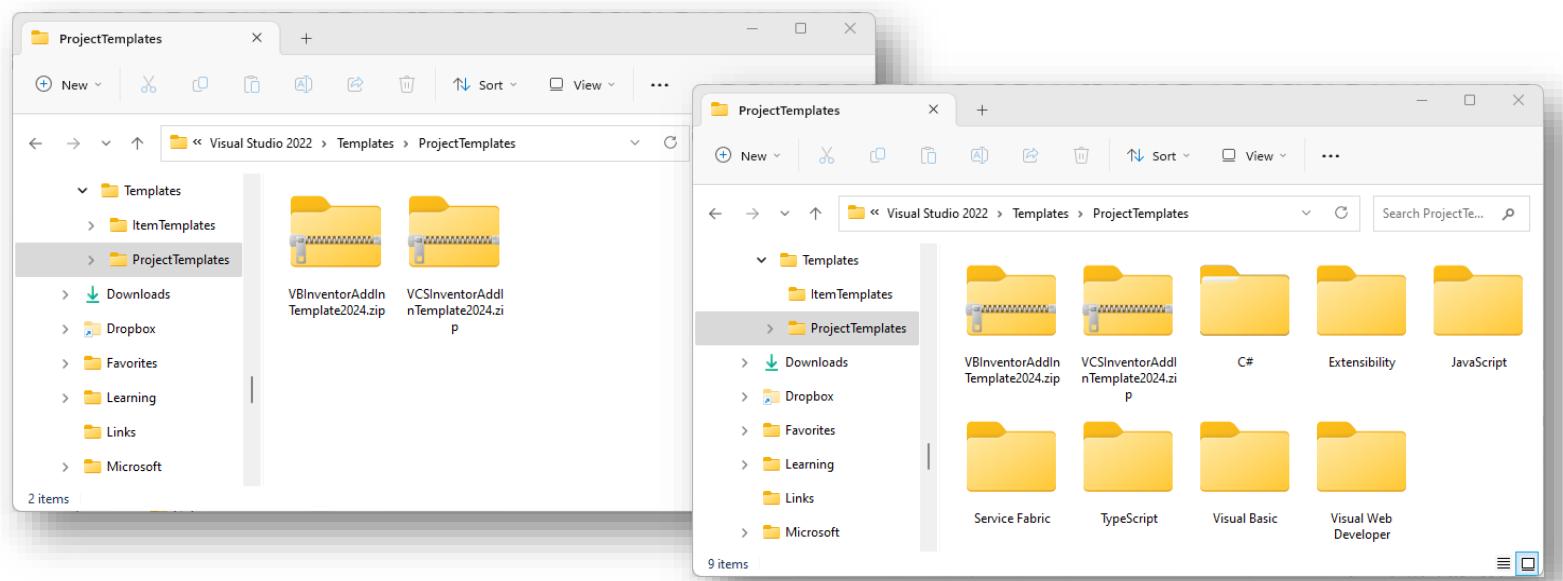
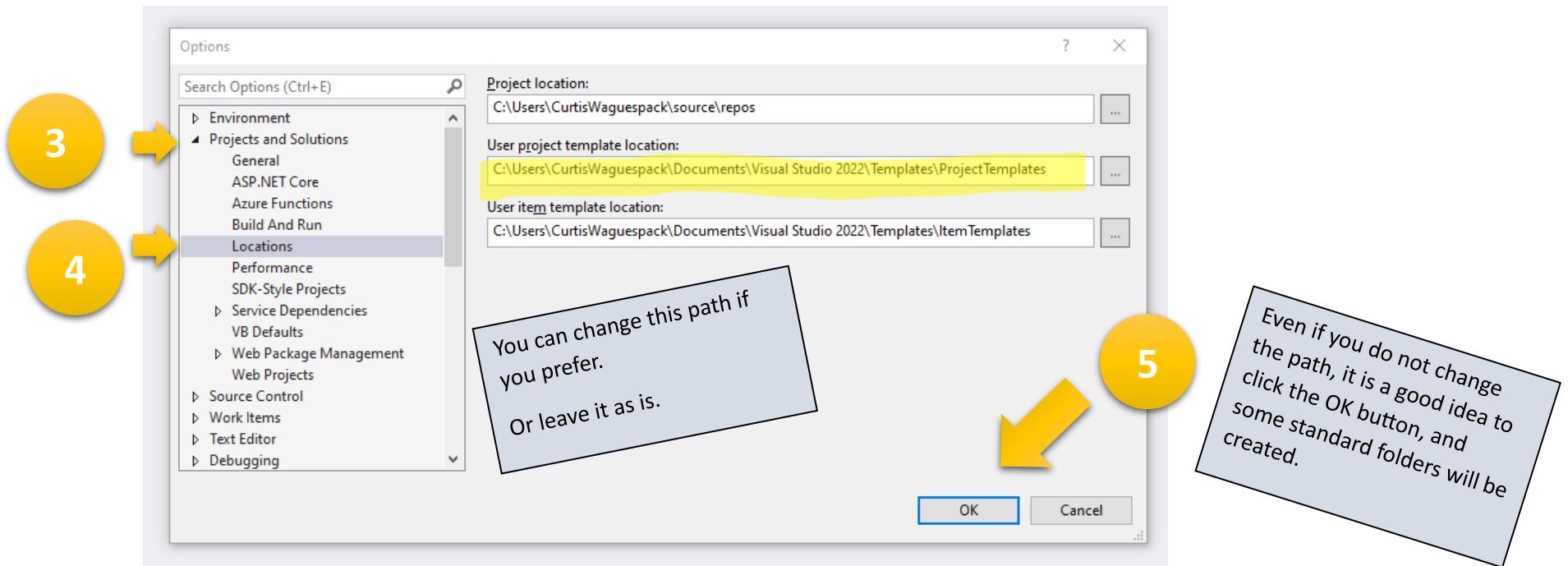
Next you will open Visual Studio and check the template location.

1. Open Visual Studio > click **Continue without code**
2. In Visual Studio click the **Tools** tab > select **Options** from the menu
3. Select **Projects and Solutions** in the left pane
4. Click **Locations**
5. Click the **OK** button to create the standard folders



2

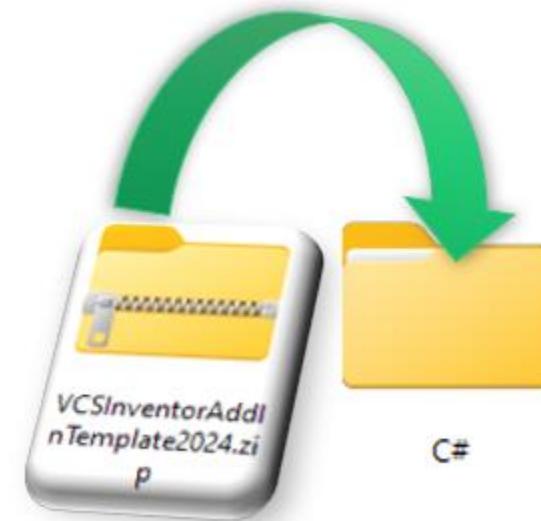
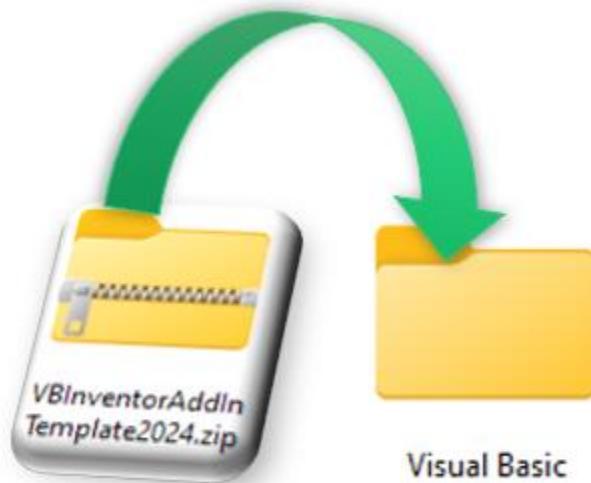




Move the \*.zip files into the standard folders

You can browser to he template folder and move the \*.zip file to their respective folders. This isn't required, but is best practice.

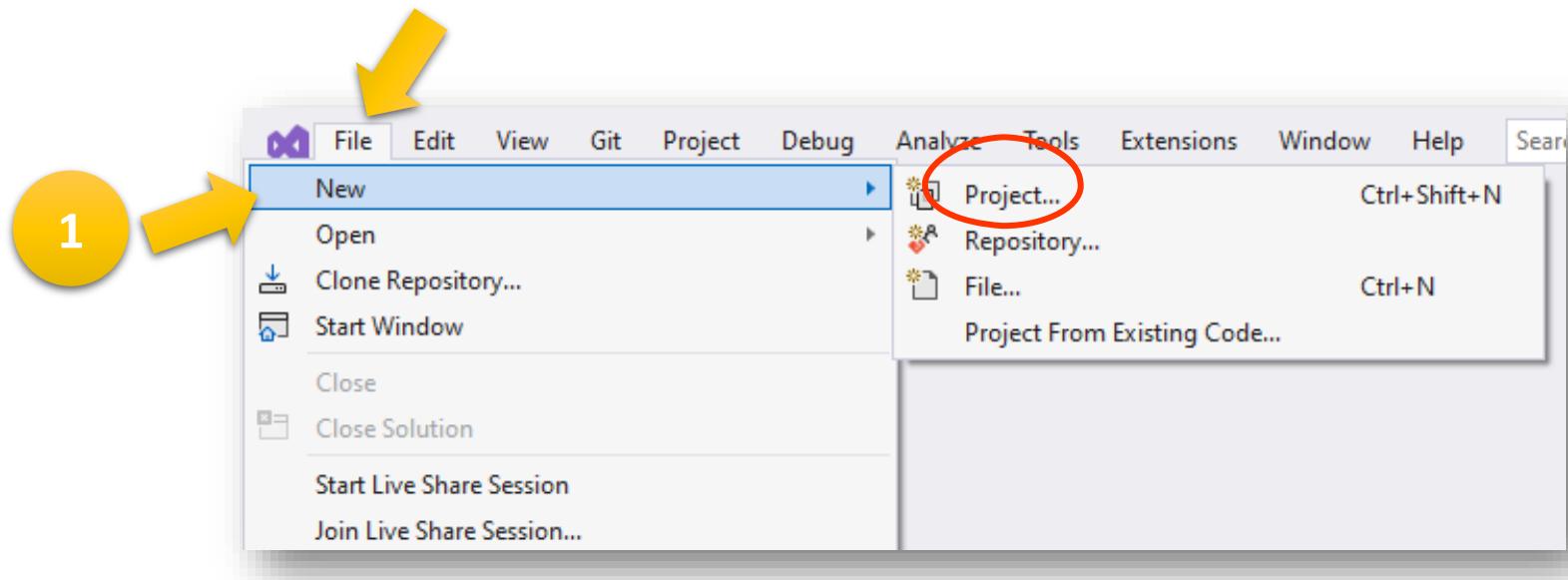
- Move the **VBInventorAddinTemplate** zip file to the **Visual Basic** folder
  - You don't need to unzip it
- Move the **VCSInventorAddinTemplate** zip file to the **C#** folder
  - You don't need to unzip it



## Ensure the templates show up

Next you will go through the steps to create a new Visual Studio project in order to check that the \*.zip file templates show up in the Create New Project dialog.

1. Still in Visual Studio click the **File** tab > select **New** from the menu.
2. Select **Create New Project** in the resulting dialog.
3. Type **Autodesk** or **Inventor** int the search bar
4. Note the templates in the list
5. Click the X in the top right corner to exit out of the Create New Project dialog.
  - o This step was just to ensure the add-ins loaded, you will create a new project using a custom add-in in the steps to come.



## Visual Studio 2022

### Open recent

- Pinned
  - D3\_Inventor\_Drawing\_Tools.sln 8/12/2023 10:43 AM
  - CA\_03\_Authorization\_Services\_Drawing\_Tools.AddIn\Sandbox\
  - Inventor\_Drawing\_Tools.sln 8/9/2023 3:01 PM
  - CA\_Inventor\_Drawing\_Tools\inventor\Drawing\_Tools\Inventor...
  - GPI\_Model\_Status\_Tools.sln 8/3/2023 3:11 PM
  - (X:\GreenPoint\Autodesk\Inventor Addins and VBA Macro\GPI\_Model\_S...
  - GPI\_Export\_KCD\_Workpoints.sln 1/27/2023 12:59 PM
  - DX\GreenPoint\Autodesk\WorkPoint\Export\GPI\_Export\_KCD\_Workpoints\
- Today
  - InventorAddin.sln 8/12/2023 8:12 AM
  - (X:\OneDrive - 03 Technologies\Desktop\Throw Away\VS test\In...
- This week

Do you like this start window?

### Get started

- Clone a repository
- Open a project or solution
- Open a local folder
- Create a new project

[Continue without code →](#)

2

### Create a new project

All languages

All platforms

[Clear all](#)

3

Autodesk Inventor 2024 C# AddIn  
A project for creating an Inventor AddIn

Autodesk Inventor 2024 VB AddIn  
A project for creating an Inventor AddIn

5

4

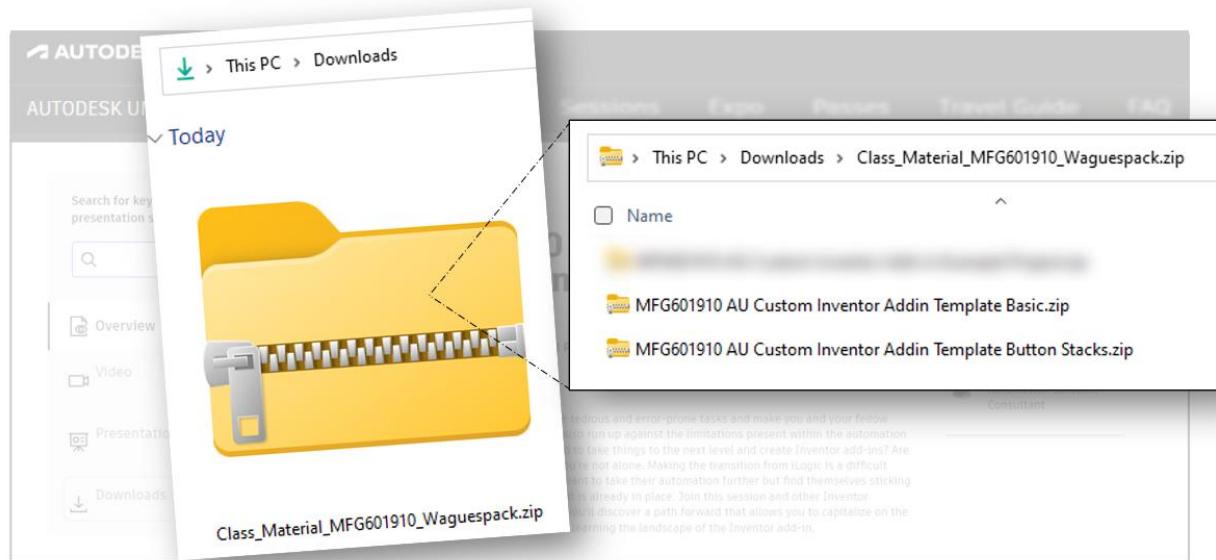
## Downloading the custom add-in template

This AU class provides some custom add-in templates to get you going with Inventor add-ins. You will download these from the AU web site for this class (*MFG601910 - Bridging the Gap Between iLogic Automation and Inventor Add-Ins*).

1. Find this AU class online, and download the class materials.
2. Find the templates within the class materials.
  - o The exact file count and names might vary from the images in this guide by the time the download materials are released.
3. Move the template zip files to the Visual Basic folder.
  - o C:\Users\CurtisWaguespack\Documents\Visual Studio 2022\Templates\ProjectTemplates\Visual Basic
  - o You don't need to unzip them.

The screenshot shows the Autodesk University website interface. At the top, there's a navigation bar with links for 'Agenda & Experience', 'Sessions', 'Expo', and 'Past'. Below the navigation, a search bar is present with the placeholder text 'Search for keywords in videos, presentation slides and handouts:'. To the right of the search bar, there's a large callout box containing the text: 'The AU page might not look exactly like this, but it should be close.' A yellow circle with the number '1' is overlaid on the left side of the screen. On the right, the session details for 'MFG601910 | Bridging the Gap Between iLogic Automation and Inventor Add-Ins' are displayed. The session is listed under 'Agenda & Experience' and is scheduled for 'Tuesday, Nov 14 | 10:30 AM - 12:00 PM PST'. The 'Downloads' section is highlighted with a black circle, and a line connects it to a similar 'Downloads' button on a separate circular interface on the left. This circular interface also contains 'Overview', 'Video', 'Presentation', and 'Downloads' buttons, all enclosed within a larger black circle.

2



3



# Creating an Add-in from an improved template

Getting Started





# Creating an Add-in

Click Create a New Project

Visual Studio 2022

Open recent

- Pinned
  - D3\_Inventor\_Drawing\_Tools.sln 8/12/2023 10:43 AM C:\...\3D Automation Services\Drawing Tools Add-in\Sandbox
  - Inventor\_Drawing\_Tools.sln 8/9/2023 3:07 PM C:\...\Inventor\_Drawing\_Tools\Inventor\_Drawing\_Tools\Inventor...
  - GPI\_Model\_Status\_Tools.sln 8/4/2023 3:11 PM C:\...\GreenPoint\...\Inventor Addins and VBA Macro\GPI\_Model\_S...
  - GPI\_Export\_XCD\_Workpoints.sln 1/27/2023 12:59 PM C:\...\GreenPoint\GPI\_Workpoint Export\GPI\_Export\_XCD\_Workpoints
- Today
  - InventorAddin.sln 8/12/2023 8:12 AM C:\...\OneDrive - 93 Technologies\Desktop\Throw Away\VS test\In...
- This week

Get started

- Clone a repository
- Open a project or solution
- Open a local folder
- Create a new project

Do you like this start window?

[Continue without code →](#)

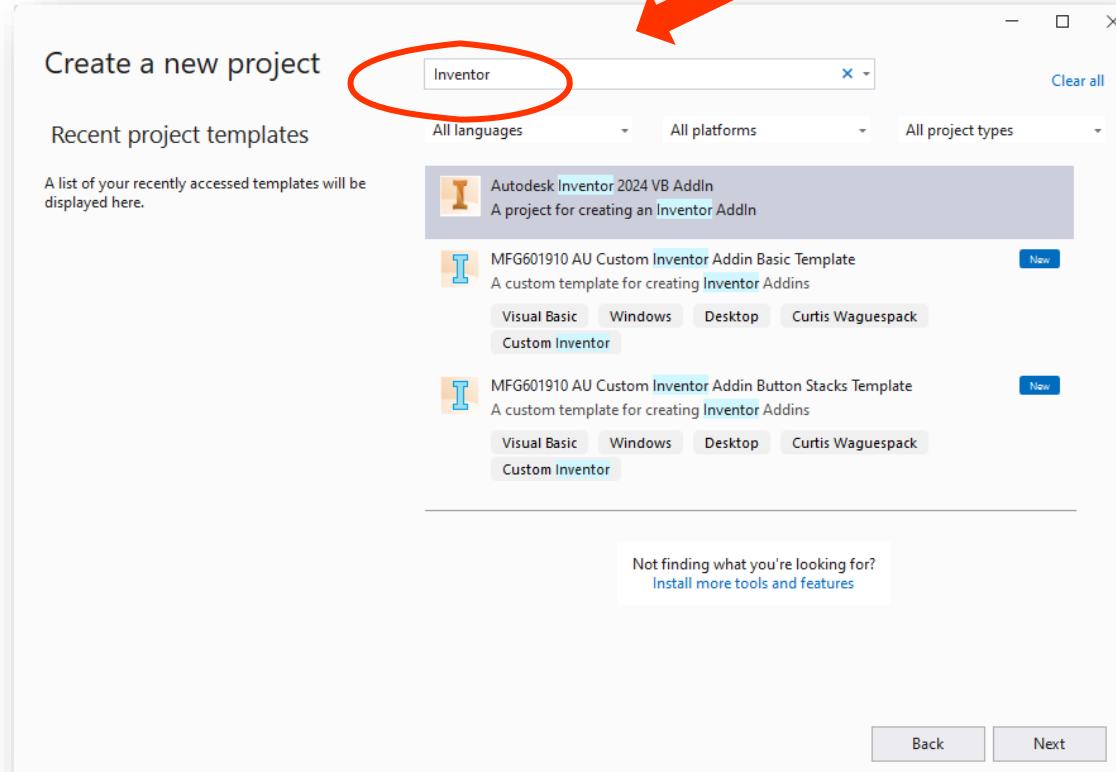




# Creating an Add-in

## 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

Create a new project

Recent project templates

A list of your recently accessed templates will be displayed here.

Search for templates (Alt+S)

All languages All platforms

Curtis Waguespack

MFG601910 AU Custom Inventor Addin Button Stacks Template  
A custom template for creating Inventor Addins  
Visual Basic Windows Desktop Curtis Waguespack  
Custom Inventor

MFG601910 AU Custom Inventor Addin Basic Template  
A custom template for creating Inventor Addins  
Visual Basic Windows Desktop Curtis Waguespack  
Custom Inventor

Not finding what you're looking for?  
[Install more tools and features](#)

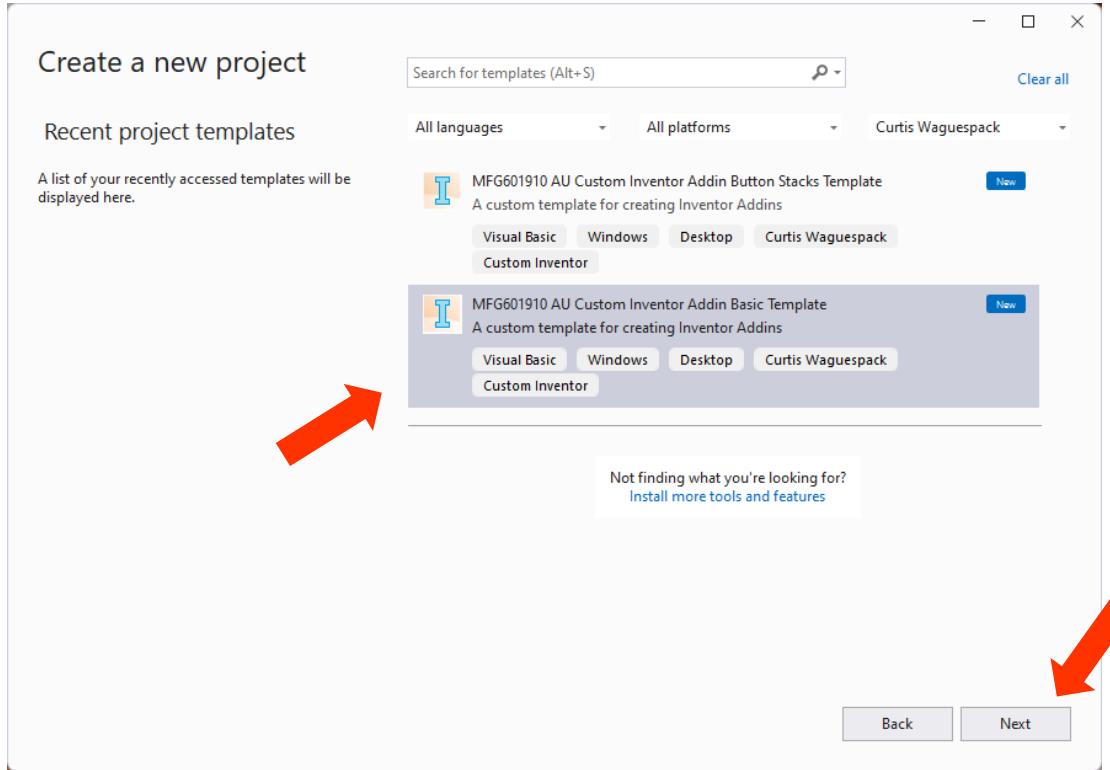
Back Next



# 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

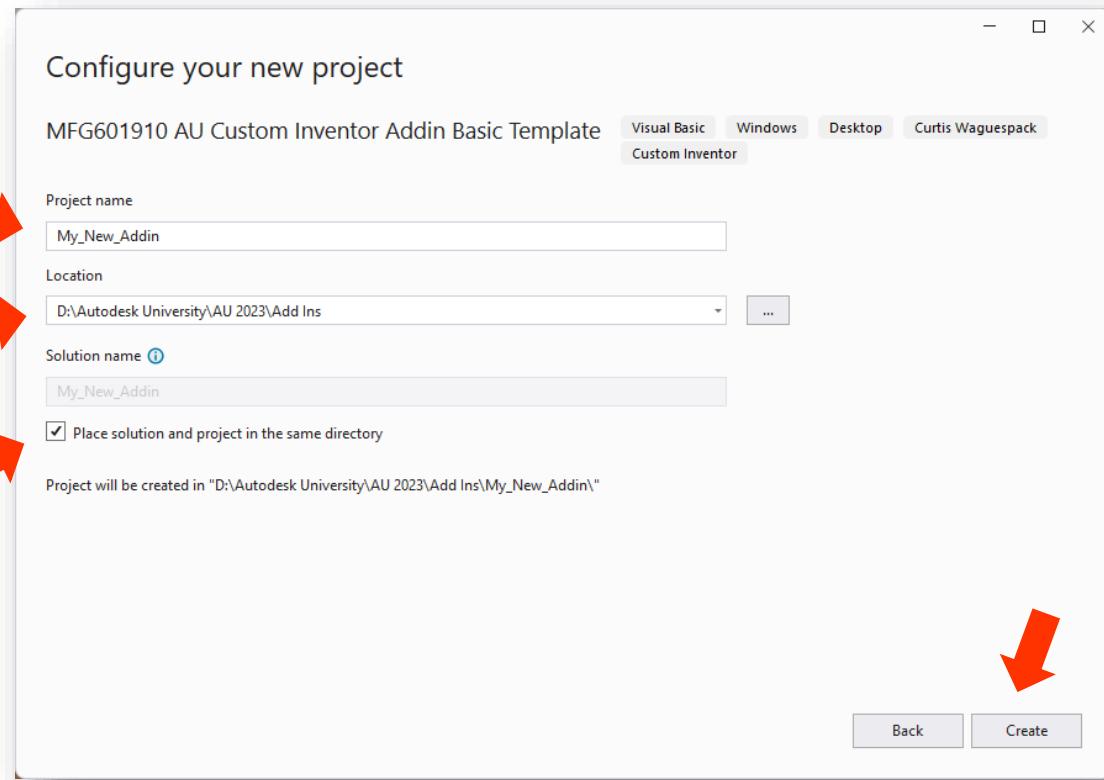
- Enter the project name

- 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



A woman in a black blazer and pinstripe pants is pointing her right index finger towards the 'View' menu in the top navigation bar of the Visual Studio interface. A yellow sticky note is overlaid on the bottom left, containing the following text:

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

The Visual Studio interface shows the 'Solution Explorer' window on the right side, which is currently expanded to show the contents of a solution named 'My\_New\_Addin'. The contents include:

- My\_New\_Addin (My Project)
- References
- Command Tools
  - API\_Help.vb
  - Detect\_Dark\_Light\_Theme.vb
  - Welcome.vb
- Resources
- Utility Tools
- AssemblyInfo.vb
  - Autodesk.My\_New\_Addin.Inventor.addin
  - My\_New\_Addin.manifest
  - Readme.txt
- StandardAddInServer.vb

- The solution is shown in the Solution Explorer

# Creating an Add-in



The screenshot shows the Microsoft Visual Studio interface. In the top navigation bar, the 'File' tab is selected. The 'Solution Explorer' window is open, displaying a solution named 'My\_New\_Addin' containing several files and folders. A context menu is open over the solution node, with the 'Open Folder in File Explorer' option highlighted by a red rectangle. The 'Output' window at the bottom shows the status 'Ready'.

- 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 screenshot shows the Microsoft Visual Studio interface with the following details:

- Solution Explorer:** Displays the project structure for "My\_New\_Addin".
- Items in Solution:**
  - Folder: .vs (File folder)
  - Folder: bin (File folder)
  - Folder: Command Tools (File folder)
  - Folder: My Project (File folder)
  - Folder: obj (File folder)
  - Folder: Resources (File folder)
  - Folder: Utility Tools (File folder)
  - File: AssemblyInfo.vb (Visual Basic Source File)
  - File: Autodesk.My\_New\_Addin.Inventor.ad... (ADDIN File)
  - File: My\_New\_Addin.manifest (MANIFEST File)
  - File: My\_New\_Addin.sln (Visual Studio Solution)
  - File: My\_New\_Addin.vbproj (Visual Basic Project File)
  - File: My\_New\_Addin.vbproj.user (Per-User Project Options File)
  - File: Readme.txt (Text Document)
  - File: StandardAddInServer.vb (Visual Basic Source File)
- Error List:** Shows 0 errors.
- Status Bar:** Shows "Ready".

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

# Creating an Add-in



The screenshot shows the Microsoft Visual Studio interface. The top menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and a Search bar. The toolbar below has icons for Undo, Redo, Cut, Copy, Paste, Find, Replace, and others. The status bar at the bottom shows "Build succeeded" and "Ready".

The main area displays the Solution Explorer, which lists one project named "My\_New\_Add-In". A context menu is open over the project node, with the "Build Solution" option highlighted by a red box and a cursor. Other options in the menu include Rebuild Solution, Clean Solution, Analyze and Code Cleanup, Configuration Manager..., Manage NuGet Packages for Solution..., Restore NuGet Packages, New Solution Explorer View, Add, Configure Startup Projects..., Create Git Repository..., Paste, Rename, Copy Full Path, Open Folder in File Explorer, Open in Terminal, Save As Solution Filter, Hide Unloaded Projects, and Properties.

The Output window below shows the build logs:

```
Show output from: Build
1> INFO: Could not find files for the given pattern(s).
1> The system cannot find the path specified.
1> 'mt.exe' is not recognized as an internal or external command,
1> operable program or batch file.
1> D:\Autodesk University\AU 2023\Add Ins\My_New_Add-In\bin\Debug\My_New_
1> 1 File(s) copied
1> D:\Autodesk University\AU 2023\Add Ins\My_New_Add-In\Autodesk.My_New_Add-In.Inventor.addin
1> 1 File(s) copied
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
===== Build started at 4:25 PM and took 00.680 seconds =====
```

The Error List tab is selected in the Output window.

- The solution is shown in the Solution Explorer



# Creating an Add-in



The screenshot shows the Microsoft Visual Studio interface. A yellow sticky note on the left says: "Tip: you can turn the Output window on/off using the View tab". The Output window at the bottom shows build logs:

```
Show output from: Build
1> INFO: Could not find files for the given pattern(s).
1> The system cannot find the path specified.
1> 'mt.exe' is not recognized as an internal or external command,
1> operable program or batch file.
1> D:\Autodesk University\AU 2023\Add Ins\My_New_AddIn\bin\Debug\My_New_AddIn.dll
1> 1 File(s) copied
1> D:\Autodesk University\AU 2023\Add Ins\My_New_AddIn\Autodesk.My_New_AddIn.Inventor.addin
1> 1 File(s) copied
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
===== Total 1 item(s) and took 00.723 seconds =====
|
```

A yellow sticky note on the right says: "Note the output logging indicates the solution build status." The Output tab is selected in the bottom navigation bar.

- The Build results are shown in the Output window

# Overall Process



Visual  
Studio

```
    *Text that shows when the case  
    Dm_toolTip_Simple As String  
    = "The system cannot find the path specified."  
    *Set to true to use a progressive tool tip, and false to a simple tool tip.  
    Dm_useProgressiveToolTip As Boolean = True  
  
    *only used if userProgressiveToolTip = true  
    Dm_toolTip_Exceeded As String = "Changes the edges in all views on all  
    Ch(140) & * Edges are pulled from the model cat.  
    Ch(140) & *"  
  
    Show output from: build  
    Output  
    1>-----  
    1> C:\Program Files\Autodesk\Inventor 2023\bin>C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe /t:library /out:D3I.dll /nologo /warn:4 /high /noconfig /optimize+ /subsystem:windows /target:library D3I_code.cs  
    1>-----  
    1> The system cannot find the path specified.  
    1> An error occurred while reading from the assembly manifest or internet comment.  
    1> Generating program or native file.  
    1> C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe /t:library /out:D3I.dll /nologo /warn:4 /high /noconfig /optimize+ /subsystem:windows /target:library D3I_code.cs  
    1> 1 File(s) copied  
    1> C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe /t:library /out:D3I.dll /nologo /warn:4 /high /noconfig /optimize+ /subsystem:windows /target:library D3I_code.cs  
    1> 1 File(s) copied  
    1>-----  
    1> Build started at 10:44 AM, and took 00:337 seconds  
    1>-----  
    1>-----  
    1>-----
```

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



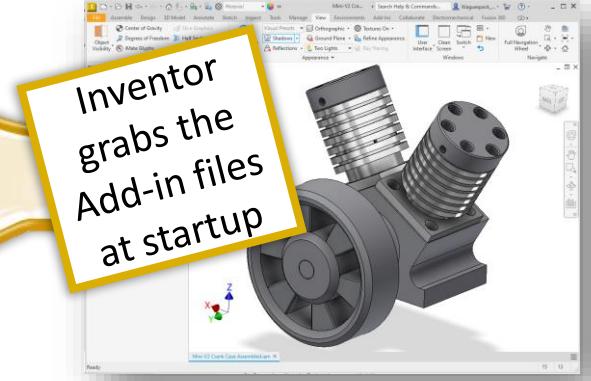
Autodesk  
Application Plugins  
Folder



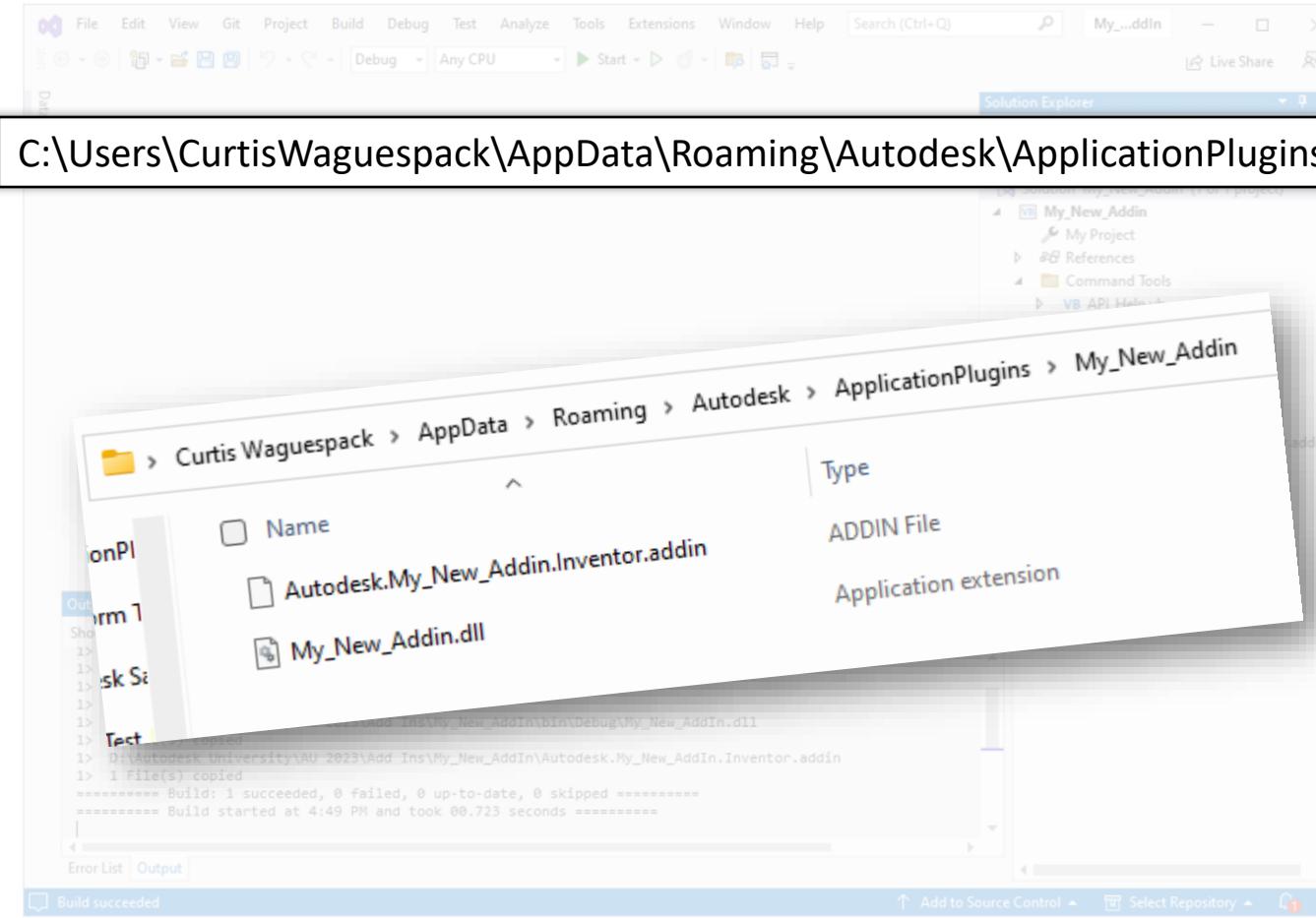
# Review



Autodesk  
Inventor



# Creating an Add-in



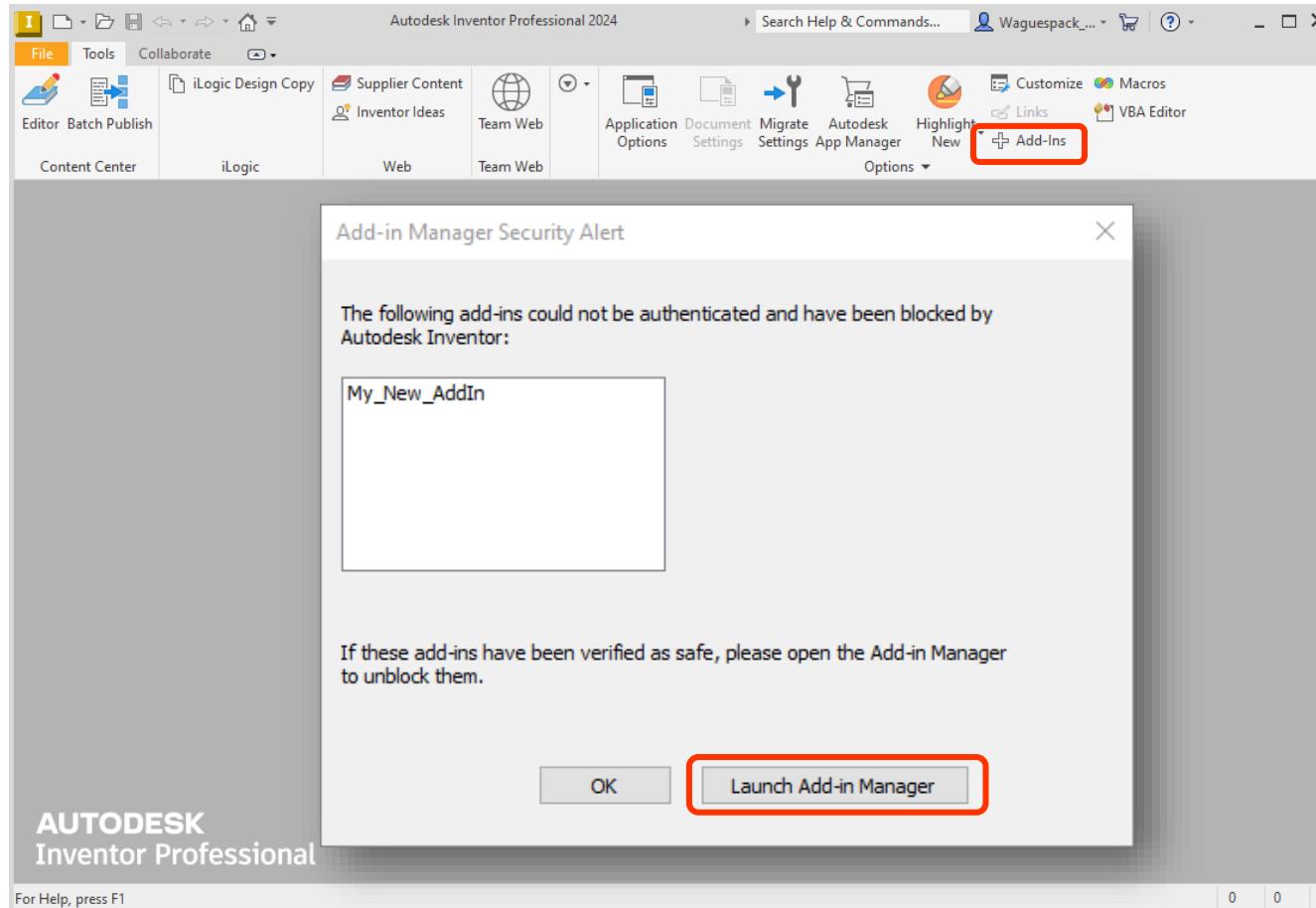
- Browse to the ApplicationPluggins folder and observe the files created during the Build

# Loading the add-in

Working in Inventor



# Loading an Add-in

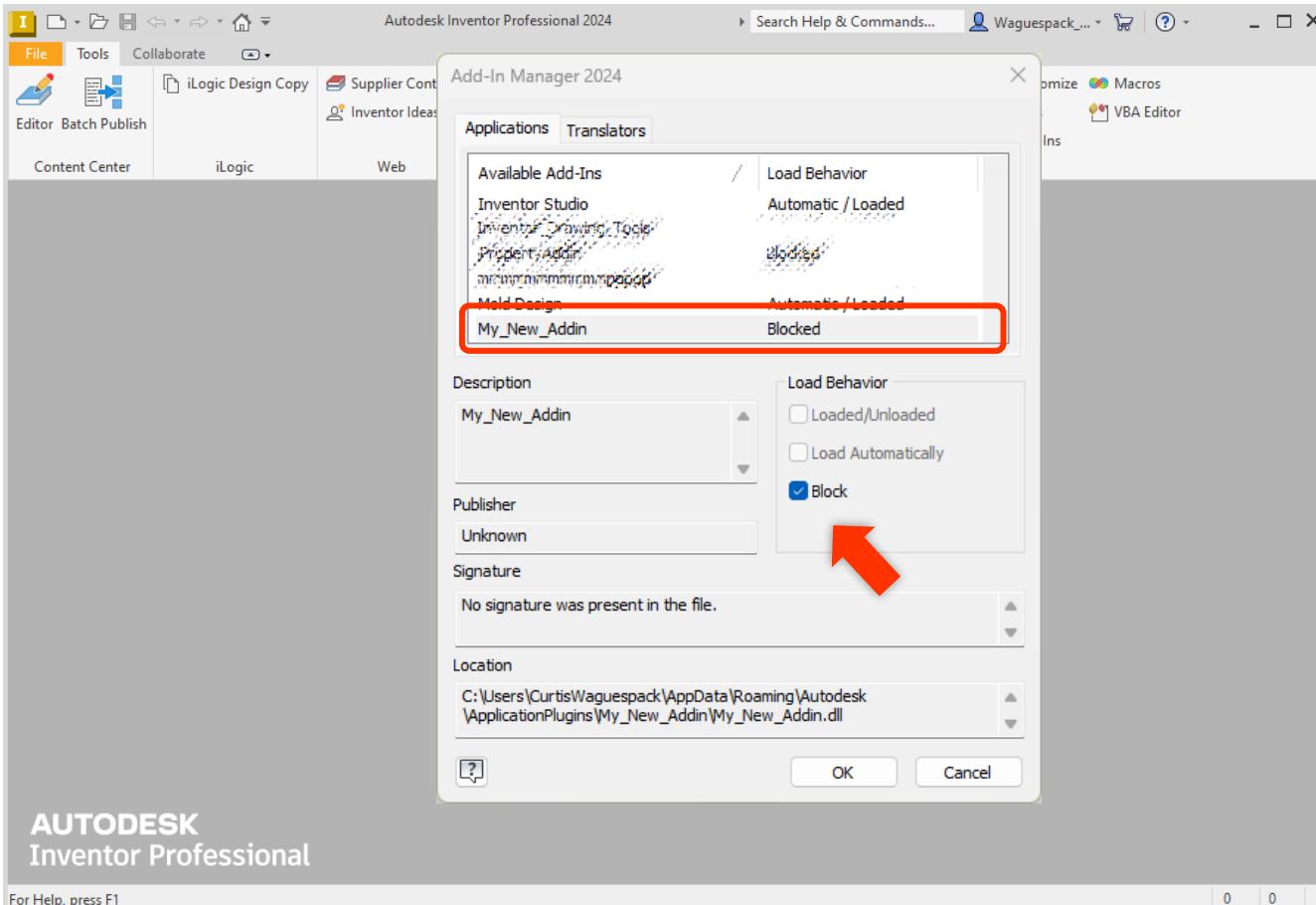


AUTODESK  
Inventor Professional

For Help, press F1

- 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 go to Tools > Add-ins

# Loading an Add-in



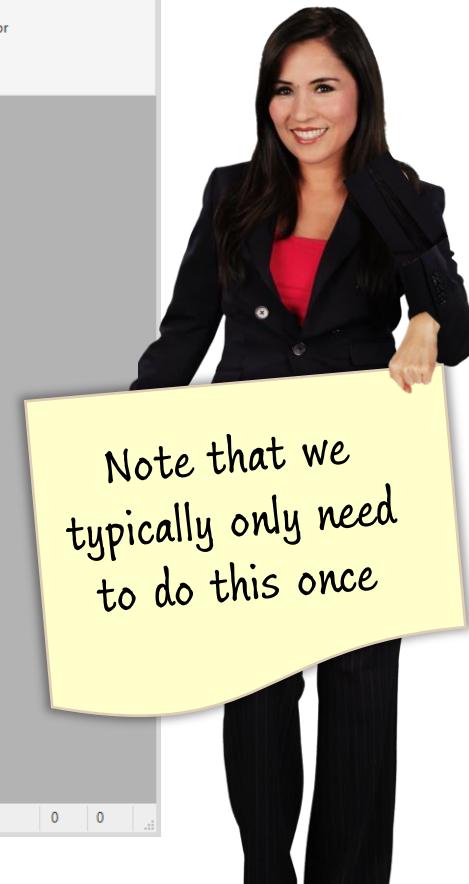
- With the add-in selected, un-check the **Block** checkbox
- And then check the **Loaded/Unloaded**
- Ad the **Load Automatically** check box

# Loading an Add-in

A screenshot of the Autodesk Inventor Professional 2024 interface. The main window title is "Autodesk Inventor Professional 2024". The top menu bar includes "File", "Tools", "Collaborate", "Editor Batch Publish", "iLogic Design Copy", "Supplier Connect", and "Inventor Ideas". A user profile "Waguespack..." is visible. The central area shows the "Add-In Manager 2024" dialog box. The "Applications" tab is selected, displaying a list of available add-ins: "Inventor Studio", "Inventor Drawing Tools", "Property Manager", "Mold Design", and "My\_New\_Addin". The "Load Behavior" column for "My\_New\_Addin" shows "Automatic / Loaded" and "Loaded". Below this, the "Description" section shows "My\_New\_Addin" and the "Publisher" is listed as "Unknown". The "Signature" section states "No signature was present in the file." The "Location" section shows the path "C:\Users\CurtsWaguespack\AppData\Roaming\Autodesk\ApplicationPlugins\My\_New\_Addin\My\_New\_Addin.dll". On the right side of the dialog, there is a "Load Behavior" group box with three checkboxes: "Loaded/Unloaded" (checked), "Load Automatically" (checked), and "Block" (unchecked). A red oval highlights this group box. At the bottom of the dialog are "OK" and "Cancel" buttons. The status bar at the bottom of the Inventor window shows "0 0".

AUTODESK  
Inventor Professional

For Help, press F1

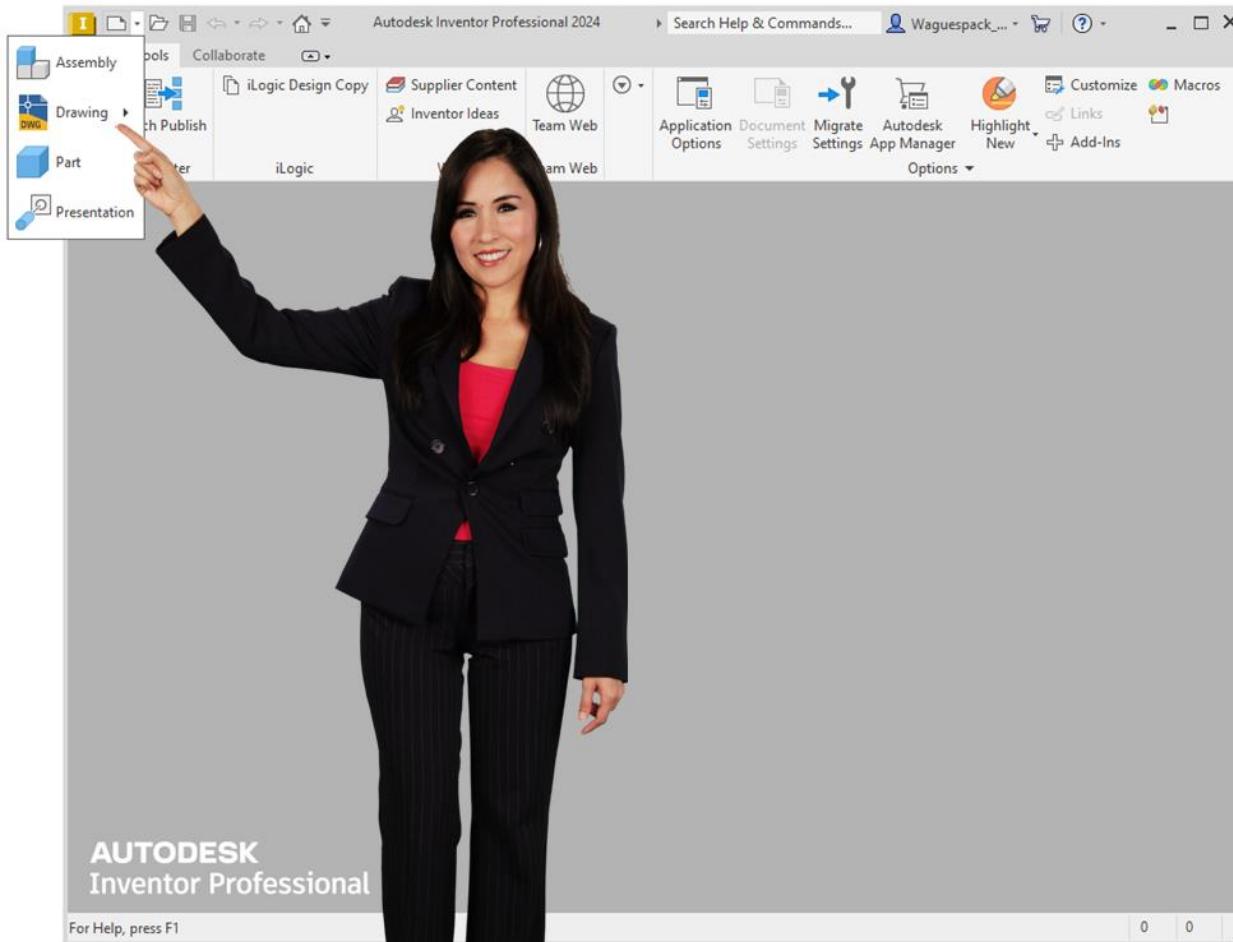


# Using the add-in

Working in Inventor

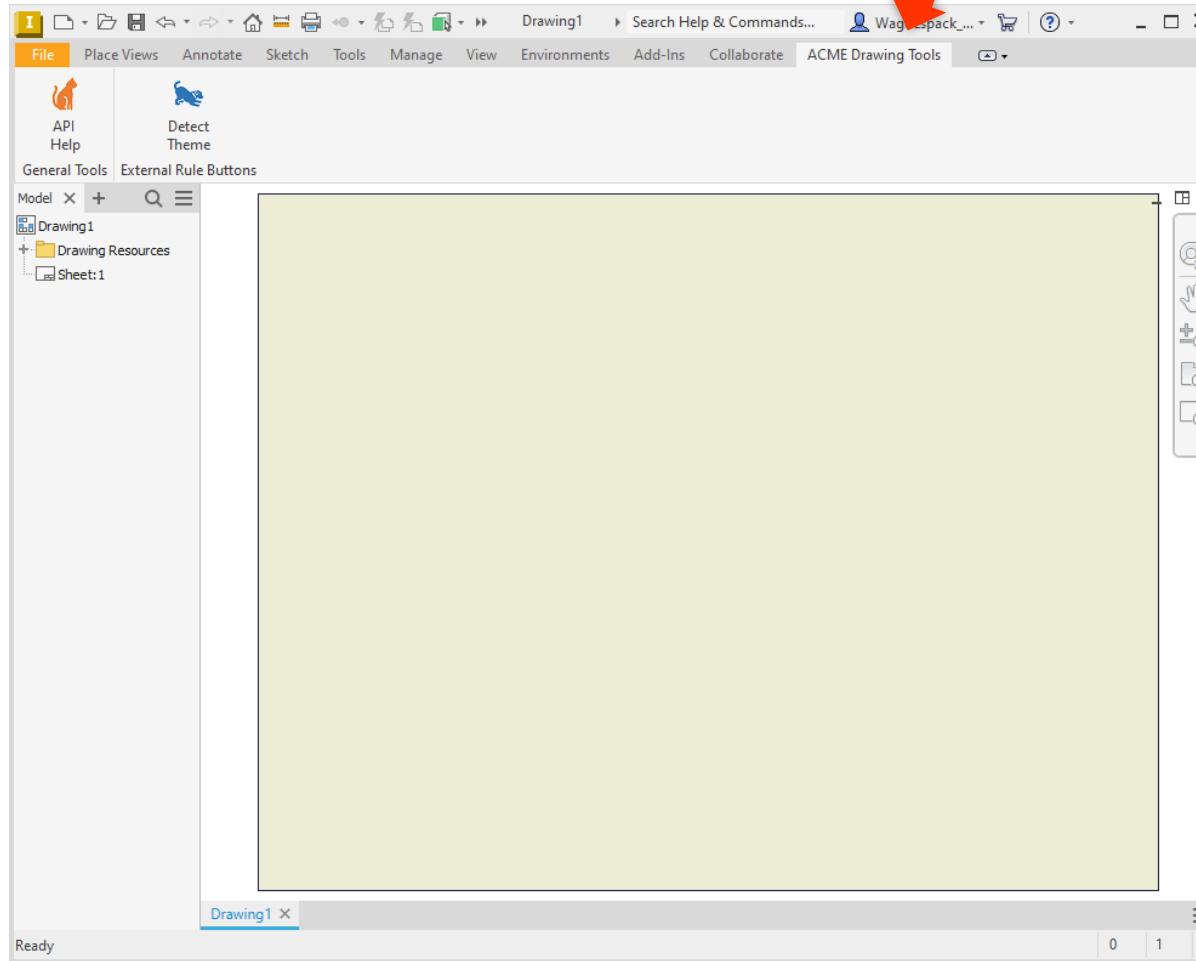


# Using the Add-in



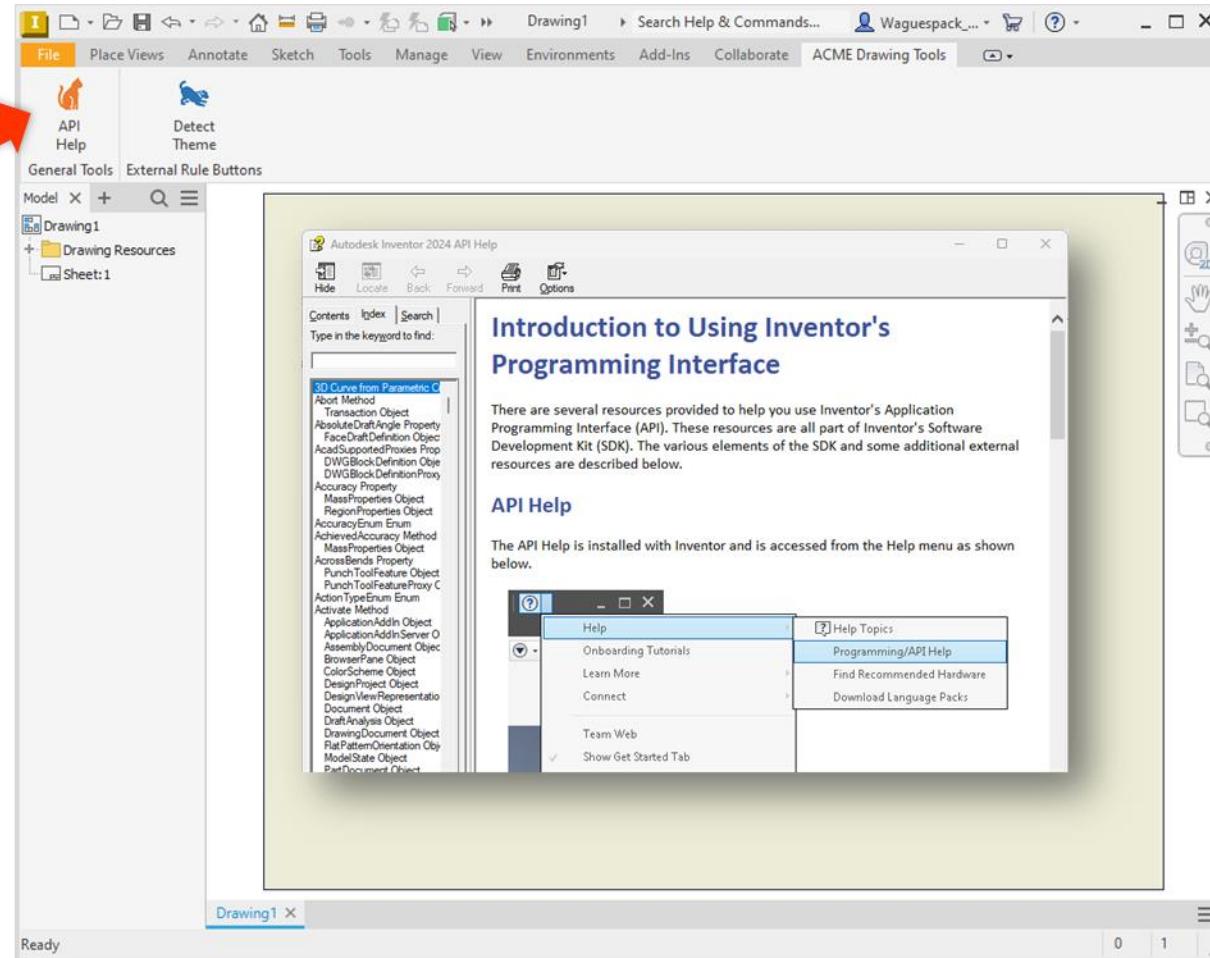
- Create a new drawing file from a template

# Using the Add-in



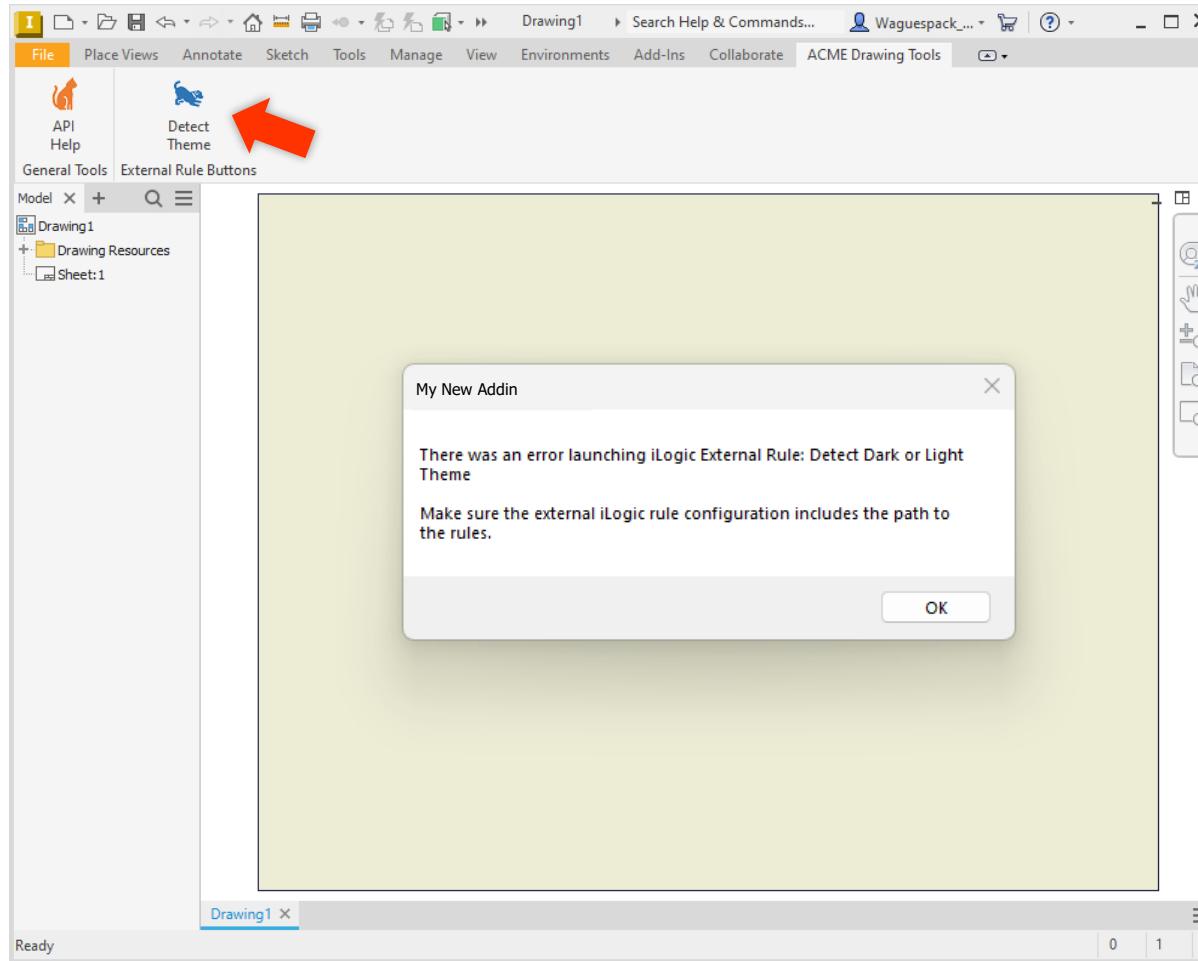
- Switch to the **ACME Drawing Tool** tab
- Note the 2 buttons found on this tab

# Using the Add-in



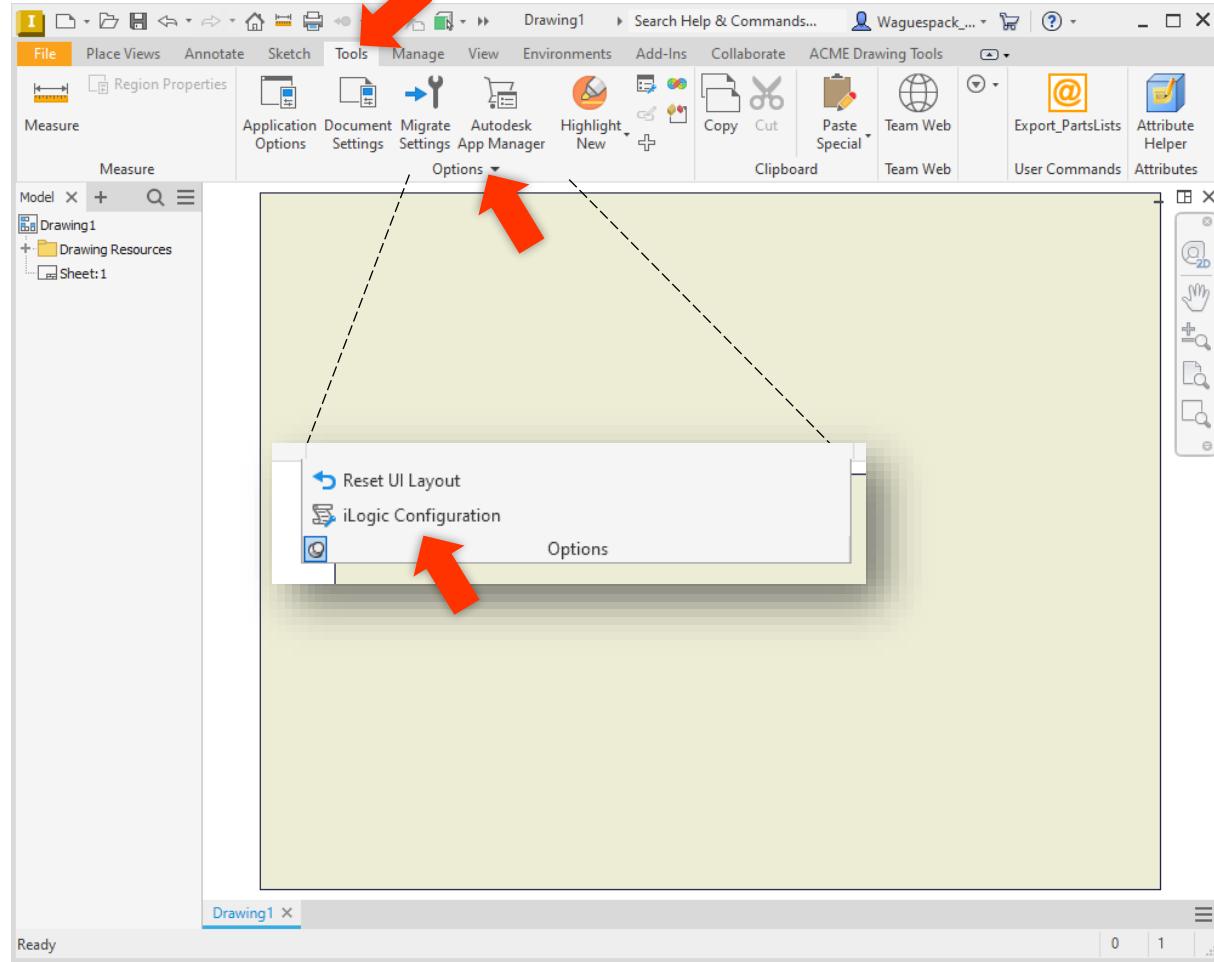
- Click the button called **API Help**
- This will open the Inventor API Help in a separate window
- This was done with code written in the add-in
- You can close the window if you like

# Using the Add-in



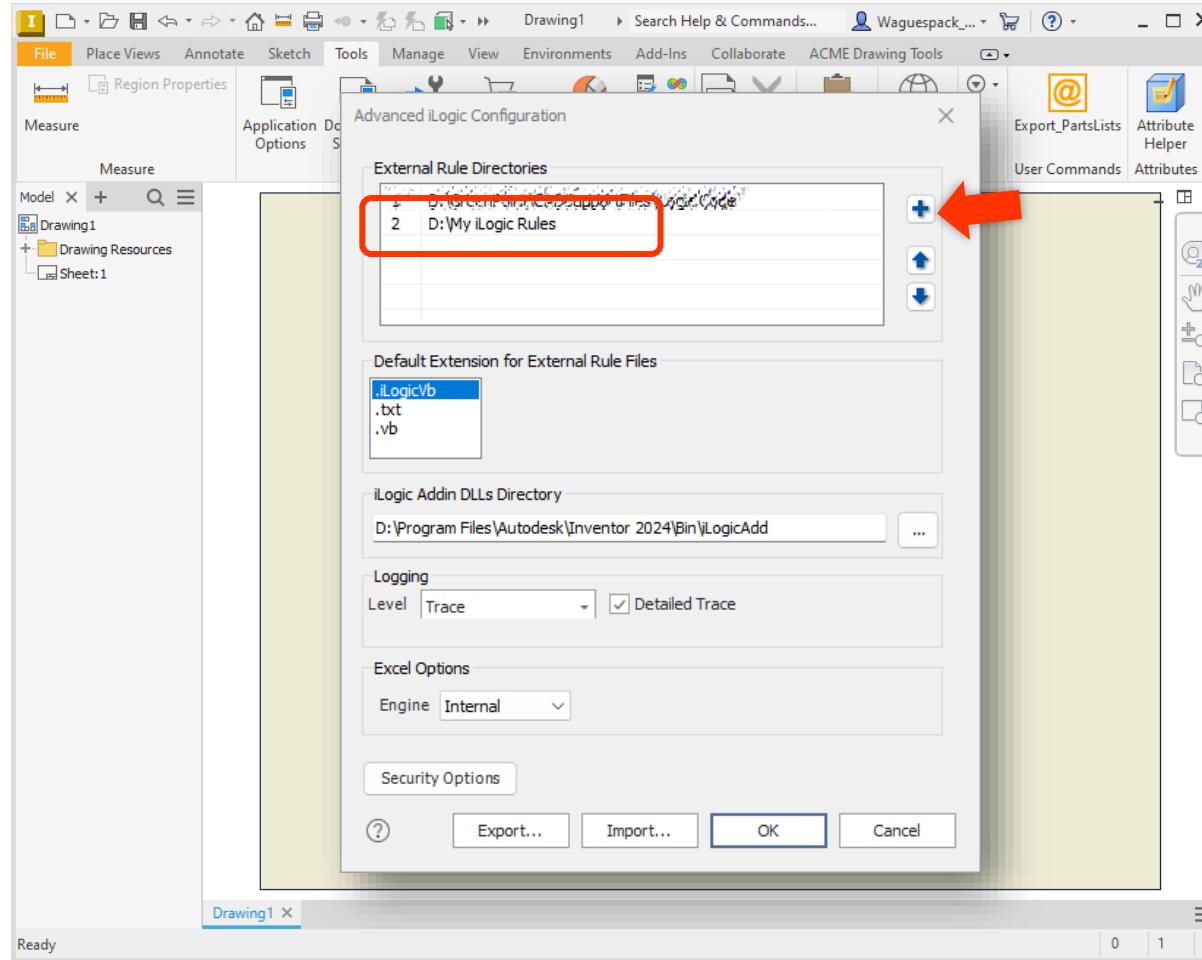
- Click the button called **API Help**
- This will open the Inventor API Help in a separate window
- This was done with code written in the add-in
- You can close the window if you like

# Using the Add-in



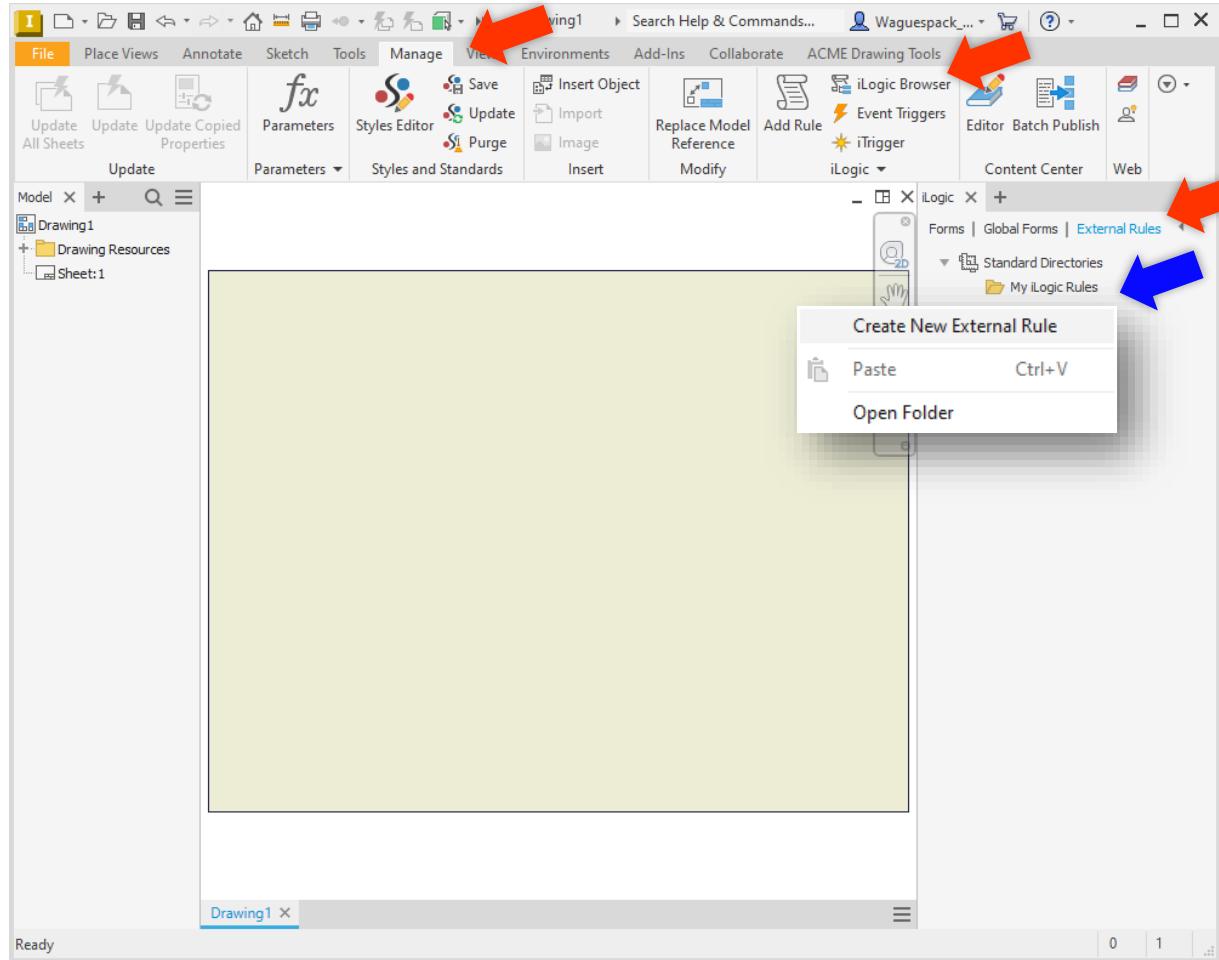
- Switch to the **Tools** tab and click the **Options** button
- Click the **iLogic Configuration** button as shown

# Using the Add-in



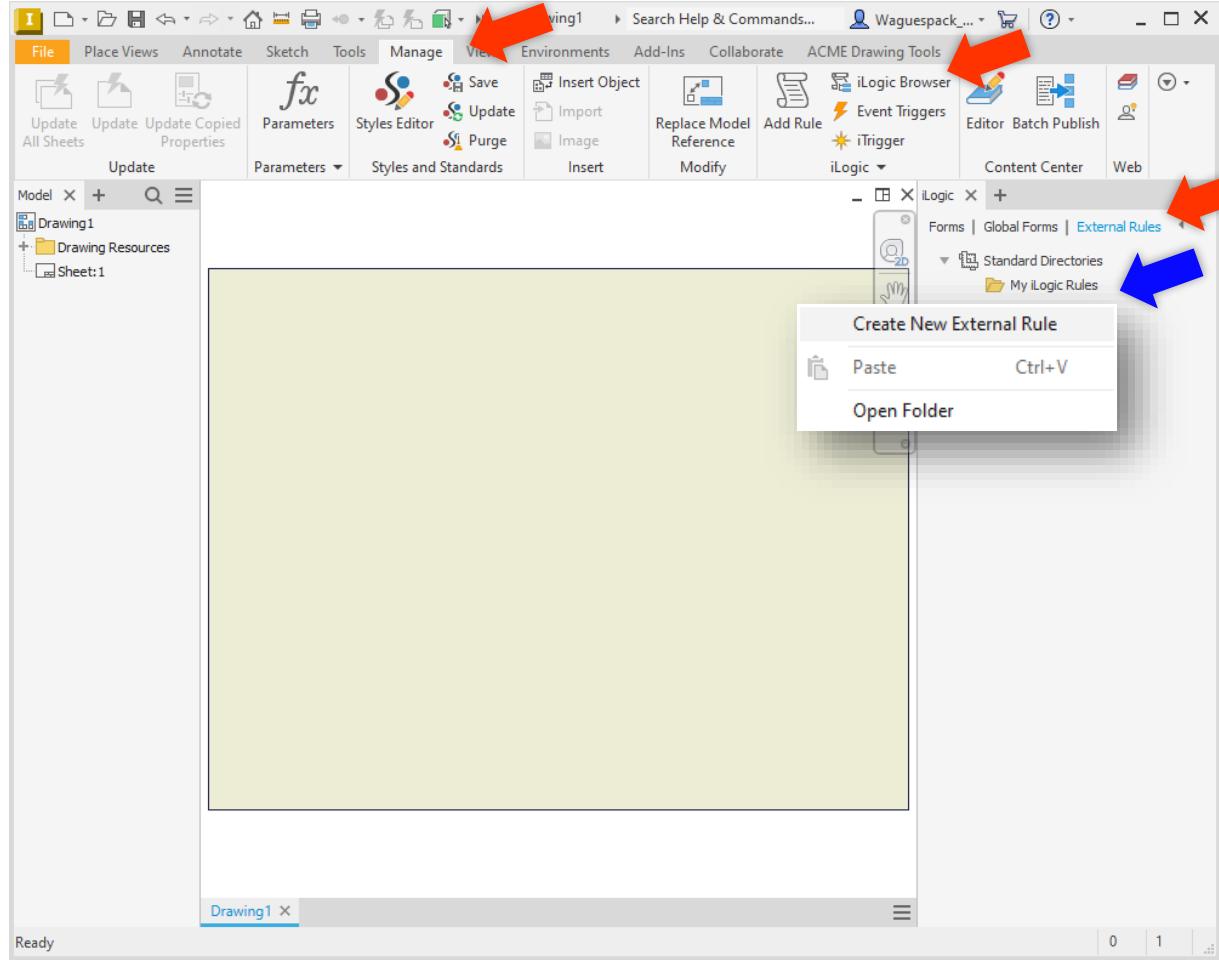
- If you have an External Rule directory already specified you can just use it
- If not use the + button to add a new path of your choice
- Click the **OK** button when done

# 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

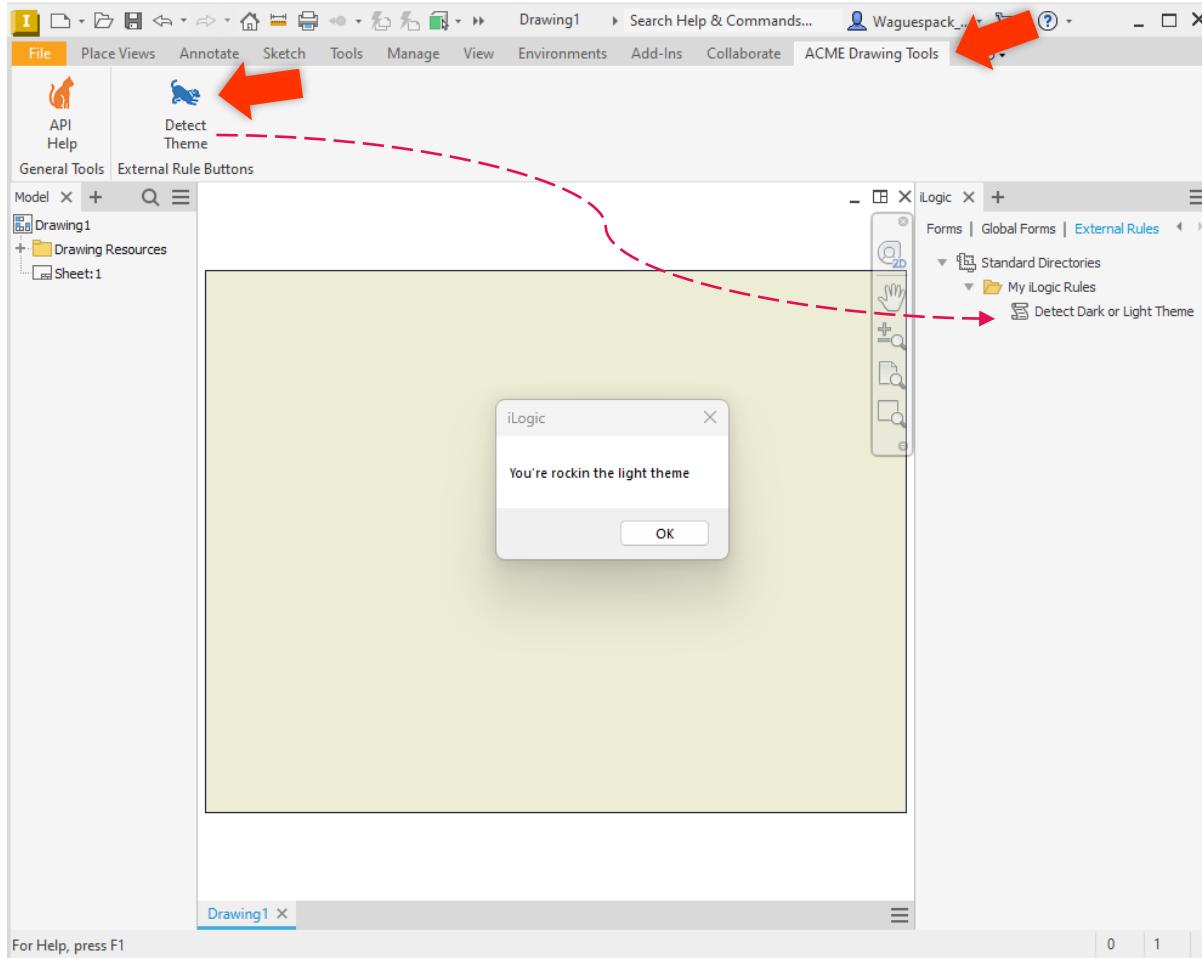
A screenshot of the ACME Drawing Tools software interface. The top menu bar includes File, Place Views, Annotate, Sketch, Tools, Manage, View, Environments, Add-Ins, Collaborate, and ACME Drawing Tools. The Manage tab is selected. Below the menu is a toolbar with icons for Update All Sheets, Update Copied Properties, Parameters, Styles Editor, Save, Import, Replace Model Reference, Add Rule, Event Triggers, iTrigger, Editor, Batch Publish, and Web. A central window titled "Edit Rule: Detect Dark or Light Theme" shows the iLogic Editor. The left sidebar lists categories like System, Custom, Parameters, Features, Components (classic), etc. The main editor area has tabs for Model, Options, Search and Replace, and Wizards. Under Model, "Drawing1" is selected, showing "Sheet:1". The iLogic code editor contains the following VBA-like script:

```
1 Dim activeTheme As String
2 activeTheme = ThisApplication.ThemeManager.ActiveTheme.Name
3
4 If activeTheme = "DarkTheme" Then
5     MsgBox("You're rockin the dark theme", "iLogic")
6 ElseIf activeTheme = "LightTheme" Then
7     MsgBox("You're rockin the light theme", "iLogic")
8 End If
9
```

The bottom of the editor shows Log Level (Trace) and Detailed Trace checkboxes, and buttons for Save, Save & Run, and Cancel.

- 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



A screenshot of the SolidWorks interface demonstrating the use of the ACME Add-in across different environments. The top navigation bar shows tabs for File, Assemble, Design, 3D Model, Annotate, Sketch, and Inspect, along with a custom tab labeled 'ACME Assembly Tools'. A red arrow points to the user profile icon in the top right of the main window. Below, three separate windows are shown: 1) Assembly environment with tabs for File, 3D Model, Sketch, Annotate, Inspect, Tools, Fusion 360, and ACME Part Tools. 2) Part environment with tabs for File, Place Views, Annotate, Sketch, Tools, Manage, View, Environments, Add-Ins, Collaborate, and ACME Drawing Tools. 3) Drawing environment with tabs for File, Place Views, Annotate, Sketch, Tools, Manage, View, Environments, Add-Ins, Collaborate, and ACME Drawing Tools. Red arrows point to the user profile icons in the top right of each of these three windows. The bottom of the interface shows General Tools and External Rule Buttons sections.

- 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

# 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

The screenshot shows the Visual Studio interface with the 'File' menu open. A yellow sticky note is overlaid on the left side of the screen, containing the 'TIP' text. The 'Solution Explorer' window is visible on the right, displaying the file structure of the 'My\_New\_Addin' project. The 'StandardAddInServer.vb' file is selected at the bottom of the list. The status bar at the bottom shows 'Ready' and some repository-related icons.

- File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
- Solution Explorer Ctrl+Alt+L
- Git Changes Ctrl+0, Ctrl+G
- Git Repository Ctrl+0, Ctrl+R
- Team Explorer Ctrl+\, Ctrl+M
- Server Explorer Ctrl+Alt+S
- Data Lake Analytics Explorer
- SQL Server Object Explorer Ctrl+\, Ctrl+S
- Test Explorer Ctrl+E, T

Any CPU Start Live Share

Solution Explorer

Search Solution Explorer (Ctrl+.)

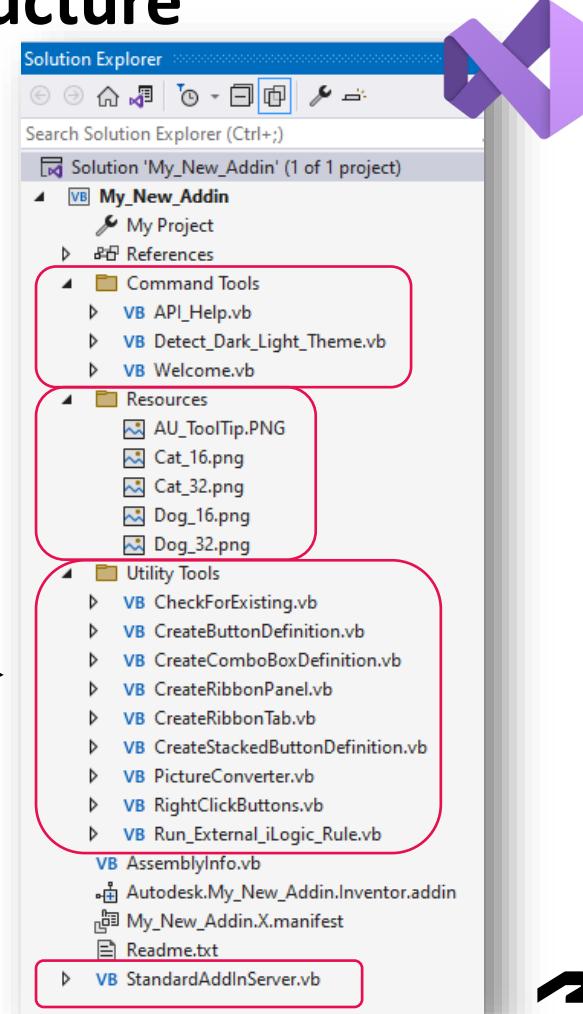
Solution 'My\_New\_Addin' (1 of 1 project)

- My\_New\_Addin
- References
- Command Tools
  - VB API.Help.vb
  - VB Detect\_Dark\_Light\_Theme.vb
  - VB Welcome.vb
- Resources
  - AU\_ToolTip.PNG
  - Cat\_16.png
  - Cat\_32.png
  - Dog\_16.png
  - Dog\_32.png
- Utility Tools
  - VB CheckForExisting.vb
  - VB CreateButtonDefinition.vb
  - VB CreateComboBoxDefinition.vb
  - VB CreateRibbonPanel.vb
  - VB CreateRibbonTab.vb
  - VB CreateStackedButtonDefinition.vb
  - VB PictureConverter.vb
  - VB RightClickButtons.vb
  - VB Run\_External\_iLogic\_Rule.vb
- VB AssemblyInfo.vb
- Autodesk.My\_New\_Addin.Inventor.addin
- My\_New\_Addin.X.manifest
- Readme.txt
- VB StandardAddInServer.vb

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



# Understanding the Add-in Template Structure



Command Tool Modules →

Resources →

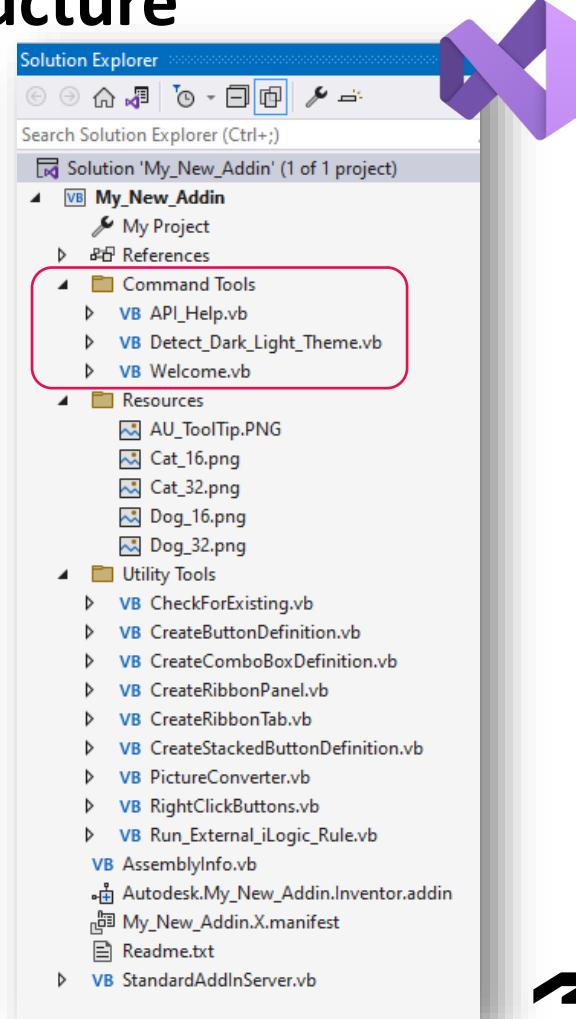
Utility Modules →

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* ✘ X
VB My_New_Addin
Imports Inventor
Module API_Help
    ''' <summary> Creates a button Button Definition Define the label, icons, tool t ...
    'This function is where the button is defined
    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, useL:
        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* ✘ X
VB My_New_Addin
Imports Inventor
Module API_Help
    'summary> Creates a button Button Definition Define the label, icons, tool t ...
    'This function is where the button is defined
    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 " & vbCrLf & "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

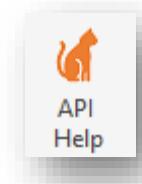
# Understanding the Add-in Template Structure



```
API_Help.vb* ✘ X
VB My_New_Addin API_Help CreateButton
Imports Inventor
Module API_Help
    ''' <summary> Creates a button Button Definition Define the label, icons, tool t ...
    'This function is where the button is defined
    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 " & vbCrLf & "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
```



Specify the icons



# Understanding the Add-in Template Structure



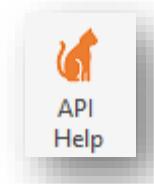
```
API_Help.vb* ✘ X
VB My_New_Addin API_Help CreateButton
Imports Inventor
Module API_Help
    'summary> Creates a button Button Definition Define the label, icons, tool t ...
    'This function is where the button is defined
    Function CreateButton(environment As String, CustomDrawingTab As RibbonTab, 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 " & vbCrLf & "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
```

Specify the Button  
Display Name



# Understanding the Add-in Template Structure



API\_Help.vb\* ✘ X

VB My\_New\_Addin API\_Help CreateButton

```
Imports Inventor

Module API_Help
    'summary> Creates a button Button Definition Define the label, icons, tool t ...
    'This function is where the button is defined
    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 " & vbCrLf & "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
```

Define the Tool Tip

The screenshot shows the Autodesk Inventor 2024 API Help window. The title bar says "Autodesk Inventor 2024 API Help". The main content area displays the "Introduction to Using Inventor's Programming Interface" page. The page includes a table of contents on the left and a detailed description of the API interface on the right. A red box highlights the "Press F1 for more help" link at the bottom of the page.

# Understanding the Add-in Template Structure



API\_Help.vb\* ✘ X

VB My\_New\_Addin API\_Help CreateButton

```
Imports Inventor
Module API_Help
    ''' <summary> Creates a button Button Definition Define the label, icons, tool t ...
    'This function is where the button is defined
    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 " & vbCrLf & "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
    End Function

    'Pass the information to the utility module
    Sub 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 Sub
End Module
```

# Understanding the Add-in Template Structure



```
API_Help.vb* ✘ X
VB My_New_Addin
Imports Inventor
Module API_Help
    'summary> Creates a button Button Definition Define the label, icons, tool t ...
    'This function is where the button is defined
    Function CreateButton(environment As String, CustomDrawingTab As RibbonTab, ribbonP
        useLargeIcon As Boolean, isInButtonStack As Boolean) As Button
        'get the images to use for the button
        Dim largeIcon As IPictureDisp = PictureDispConverter.ToIPictureDisp(My.Resource
        Dim standardIcon As IPictureDisp = PictureDispConverter.ToIPictureDisp(My.Resou
        Dim toolTipImage As IPictureDisp = Nothing
    End Function

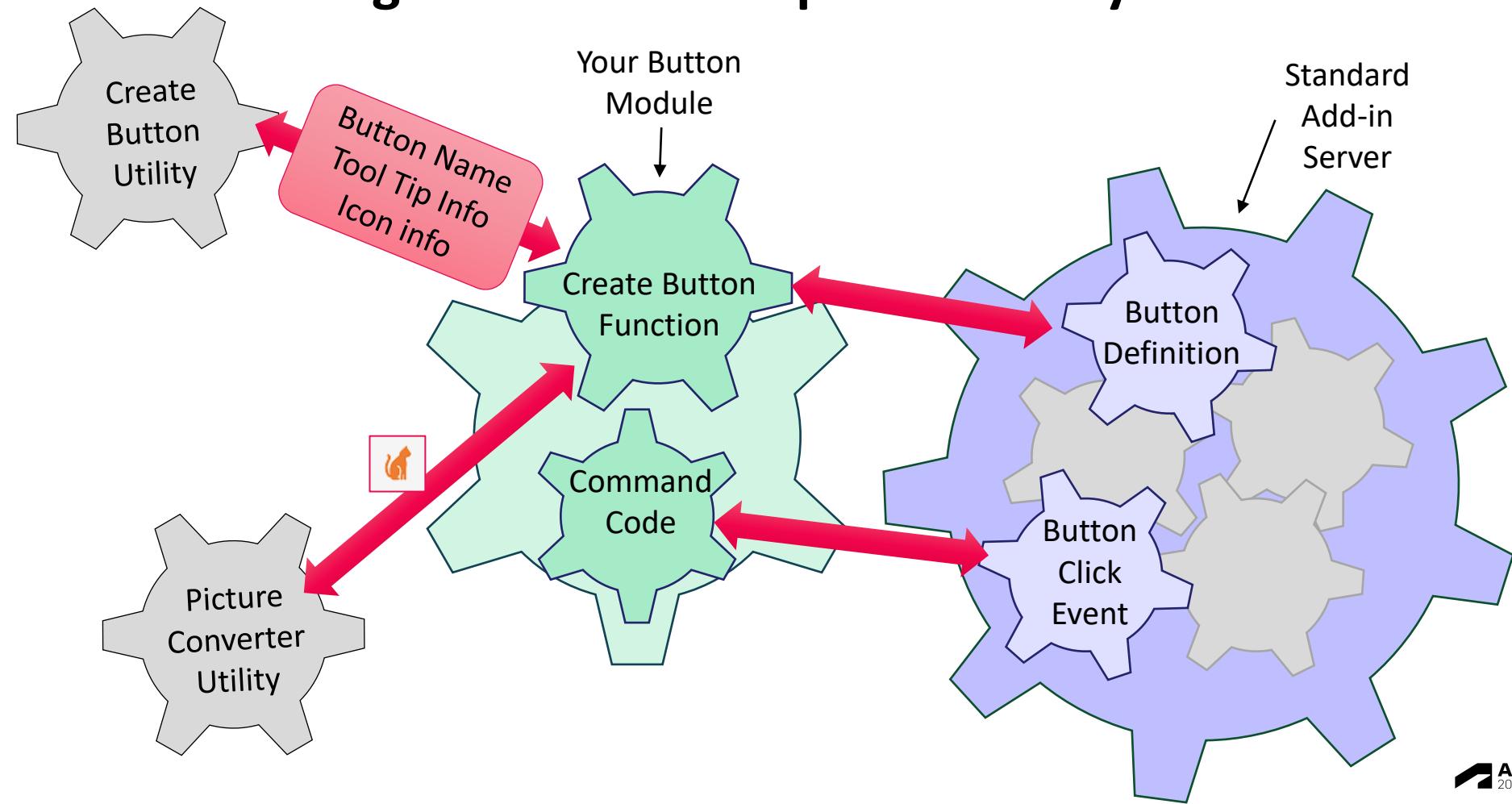
    'This is the code that does the real work when your command is executed.
    Sub RunCommandCode()
        Dim Version = g_inventorApplication.SoftwareVersion.DisplayVersion
        Dim Array = Split(Version, ".")
        Version = Array(0)
        Dim Build = g_inventorApplication.SoftwareVersion.Major

        Dim Filename = "C:\Users\Public\Documents\Autodesk\Inventor " &
        Version & "\Local Help\ADMAPI_" & Build & "_0.chm"

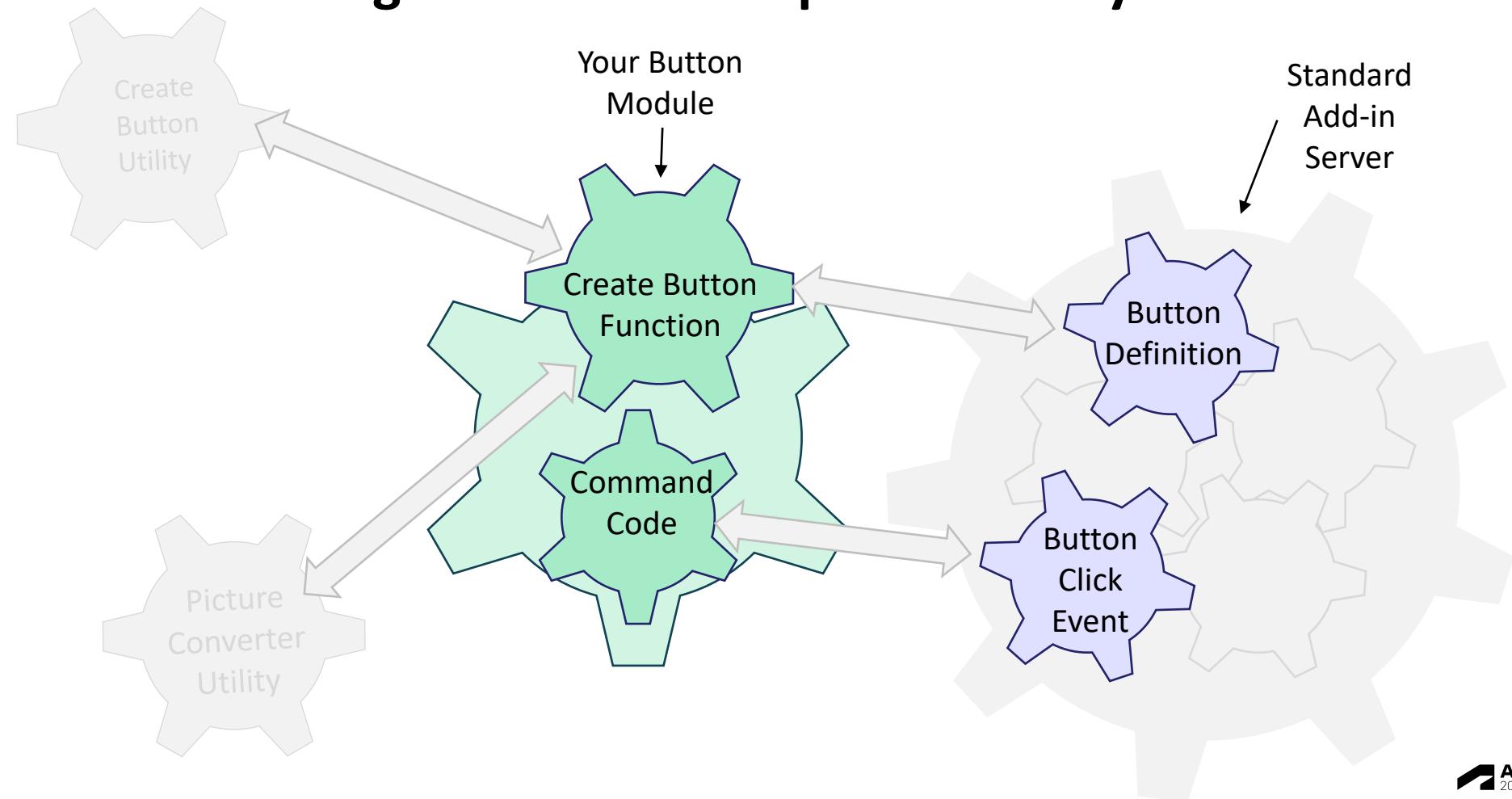
        If System.IO.File.Exists(Filename) = True Then
            Process.Start(Filename)
        Else
            MsgBox("File Does Not Exist" & vbCrLf & Filename)
        End If
    End Sub
End Module
```

- 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 Visually

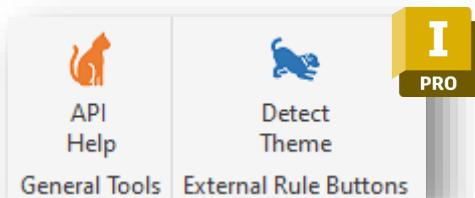
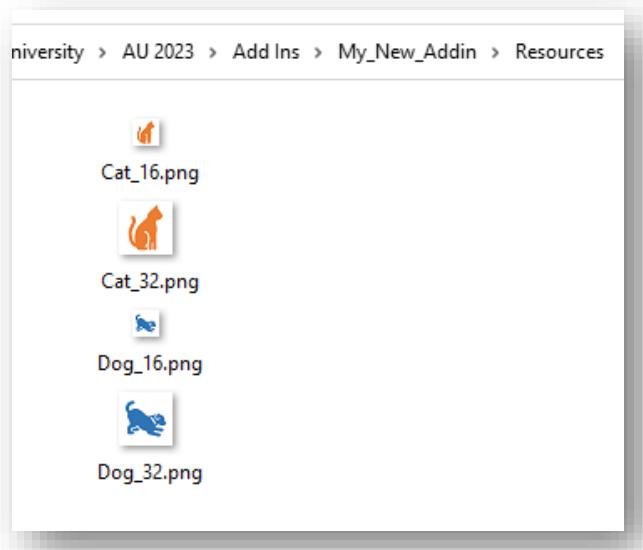


# Understanding the Add-in Template Visually

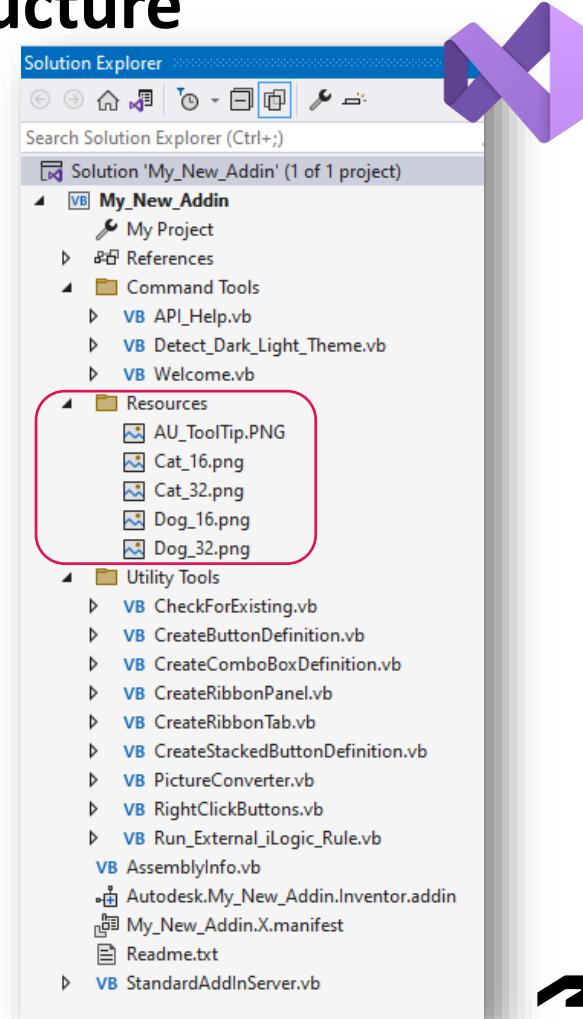


# Understanding the Add-in Template Structure

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



Resources →



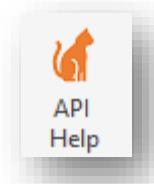
# Adding a button image



```
API_Help.vb* ✘ X
VB My_New_Addin API_Help CreateButton
Imports Inventor
Module API_Help
    ''' <summary> Creates a button Button Definition Define the label, icons, tool t ...
    'This function is where the button is defined
    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 " & vbCrLf & "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
```



Specify the icons



# Adding a button image



Screenshot of the Visual Studio IDE showing the Project menu open. A red arrow points to the 'Project' option in the menu bar. Another red arrow points to the 'My\_New\_Addin Properties' item in the bottom-left corner of the menu.

```
API File Edit View Git Project Build Debug
VB M
+ Add Form (Windows Forms)...
+ Add User Control (Windows Forms)...
+ Add Component...
+ Add Module...
+ Add Class...
+ Add New Data Source...
□ Add New Item... Ctrl+Shift+A
□ Add Existing Item... Shift+Alt+A
Exclude From Project
□ Show All Files
Add Reference...
Add Service Reference...
Connected Services
Add Analyzer...
⚙ Configure Startup Projects...
Set as Startup Project
Export Template...
🔧 My_New_Addin Properties
End Function
77 End Module
78
79
```

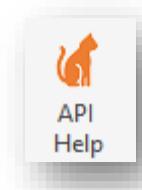
The code editor shows a snippet of VBA-like code:

```
    'omDrawingTab As RibbonTab, ribbonPanel As RibbonPanel,
    'InButtonStack As Boolean) As ButtonDefinition
    Inverter.ToIPictureDisp(My.Resources.Cat_32)
    -pConverter.ToIPictureDisp(My.Resources.Cat_16)

    'Help"
    'the button
    'Help"

    'nDef(environment, CustomDrawingTab, ribbonPanel, useLargeIcon,
    'isInButtonStack, useProgressToolTip,
    'buttonLabel, toolTip_Simple, toolTip_Expanded,
    'standardIcon, largeIcon, toolTipImage)
```

Specify the icons



# Understanding the Add-in Template Structure



The screenshot shows the Microsoft Visual Studio interface. On the left, the 'Resources' pane is open, displaying various image files: Cat\_16, Cat\_32, Dog\_16, Dog\_32, API\_ToolTip, Elephant\_16, and Elephant\_32. A red arrow points from the 'Resources' tab in the left sidebar to the pane itself. Another red arrow points from the 'Images' button in the top toolbar to the pane. A large green curved arrow at the bottom points from the 'Resources' pane towards the 'Solution Explorer' on the right.

My\_New\_Addin\* API\_Help.vb

File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Search... Live Share

My\_New\_Addin

Application Compile Debug References Services Settings Signing My Extensions Code Analysis

Resources\*

Images Add Resource Remove Resource Access Modifier: Friend

Cat\_16 Cat\_32 Dog\_16

Dog\_32 API\_ToolTip Elephant\_16

Elephant\_32

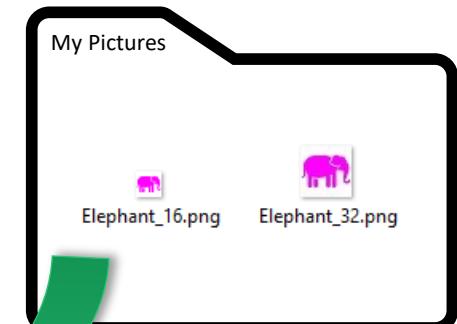
Solution Explorer

Search Solution Explorer (Ctrl+)

Solution 'My\_New\_Addin' (1 of 1 pr

- My\_New\_Addin
  - My Project
  - References
  - Command Tools
    - VB API\_Help.vb
    - VB Detect\_Dark\_Light\_Theme
    - VB Welcome.vb
  - Resources
    - API\_ToolTip.png
    - Cat\_16.png
    - Cat\_32.png
    - Dog\_16.png
    - Dog\_32.png
    - Elephant\_16.png
    - Elephant\_32.png
  - Utility Tools
    - VB AssemblyInfo.vb
    - Autodesk.My\_New\_Addin.manifest
    - My\_New\_Addin.X.manifest
    - Readme.txt
  - VB StandardAddInServer.vb

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



# Understanding the Add-in Template Structure

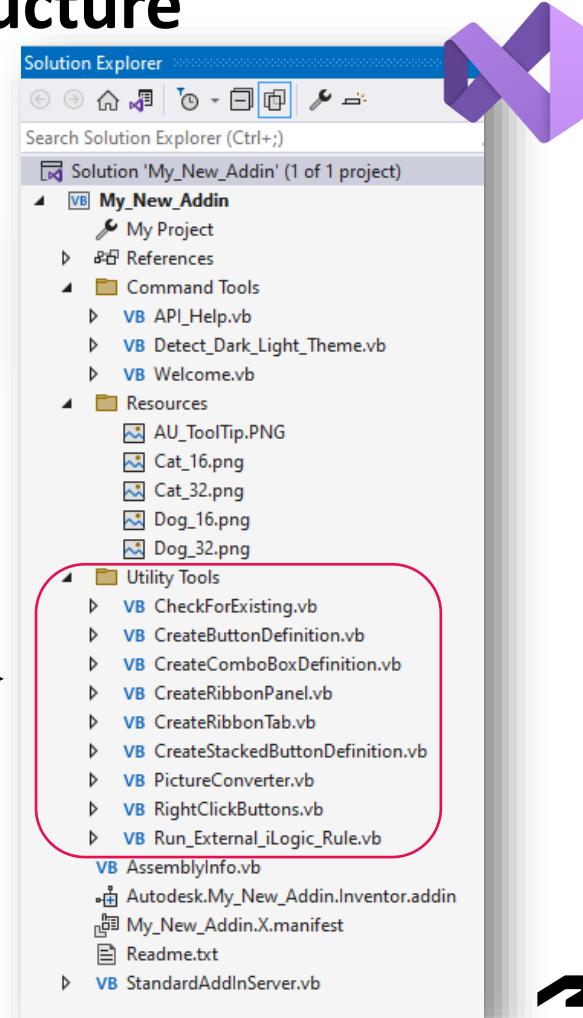
- Modules that are called over and over from the Command Tools and the StandardAddInServer

```
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")
```

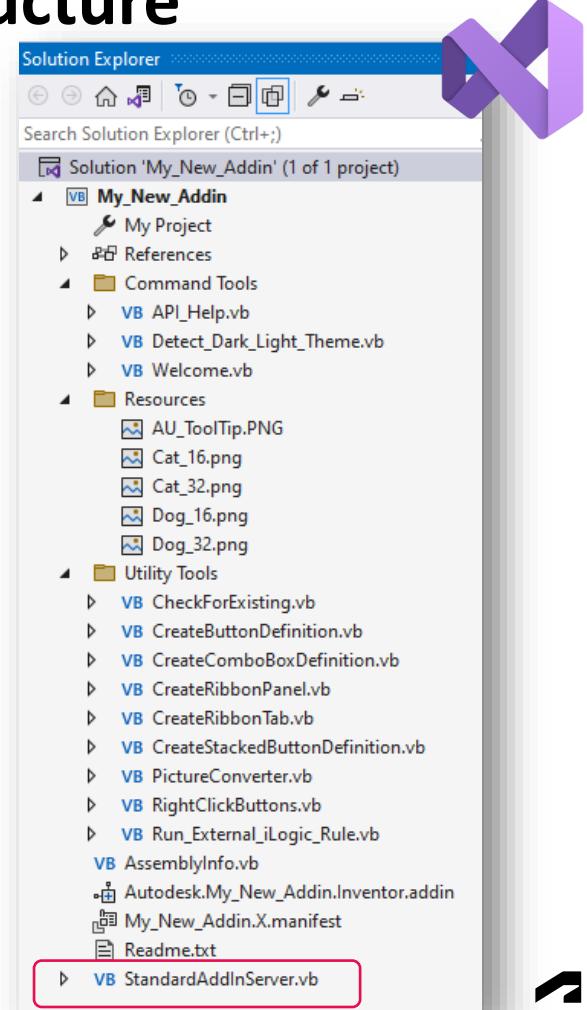
Utility Modules →



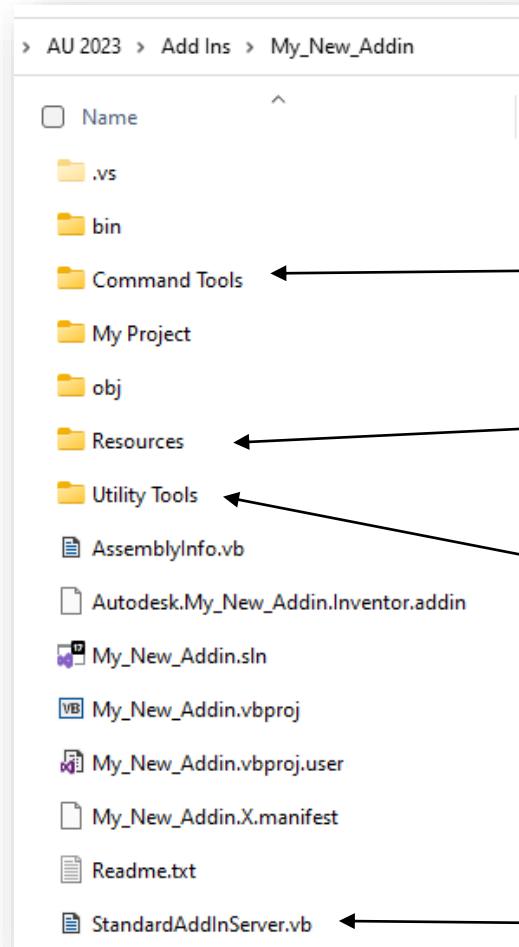
# Understanding the Add-in Template Structure

- 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 →



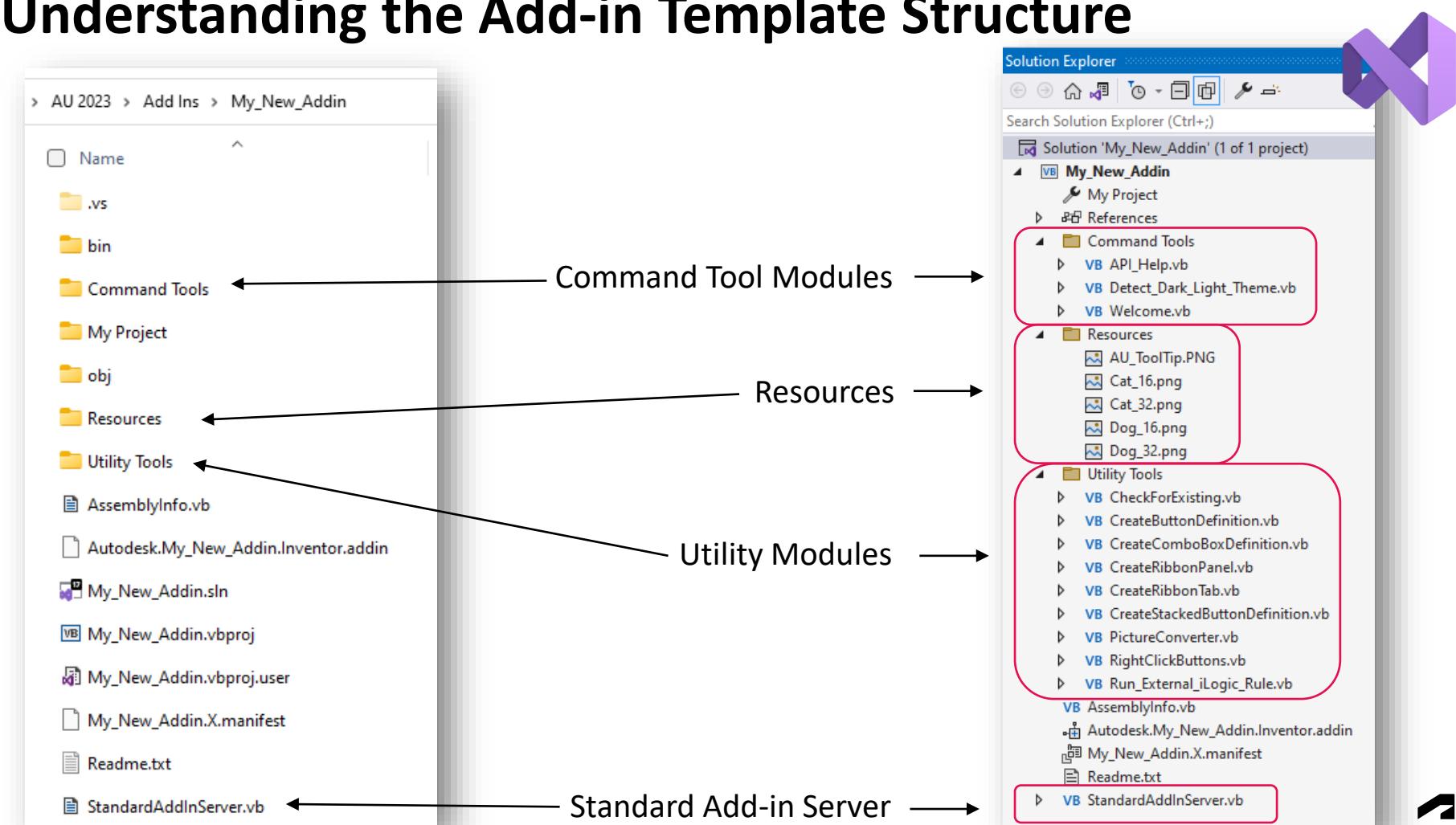
# Understanding the Add-in Template Structure



Command Tool Modules

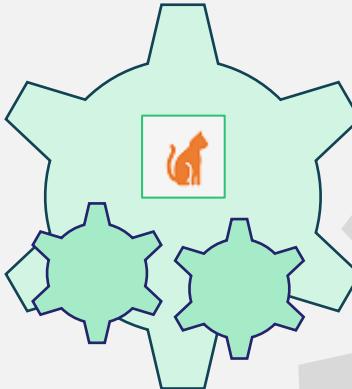
Resources

Utility Modules



# Understanding the Add-in Template Structure

Utilities



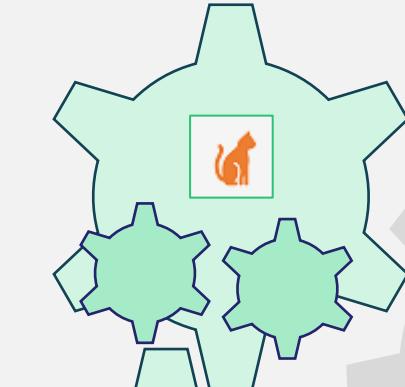
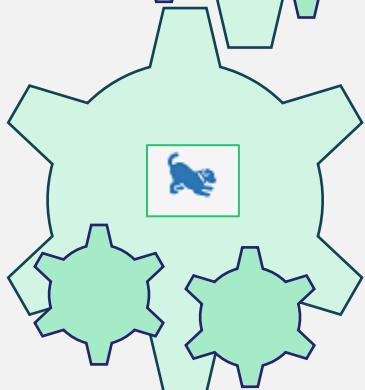
Standard Add-in  
Server



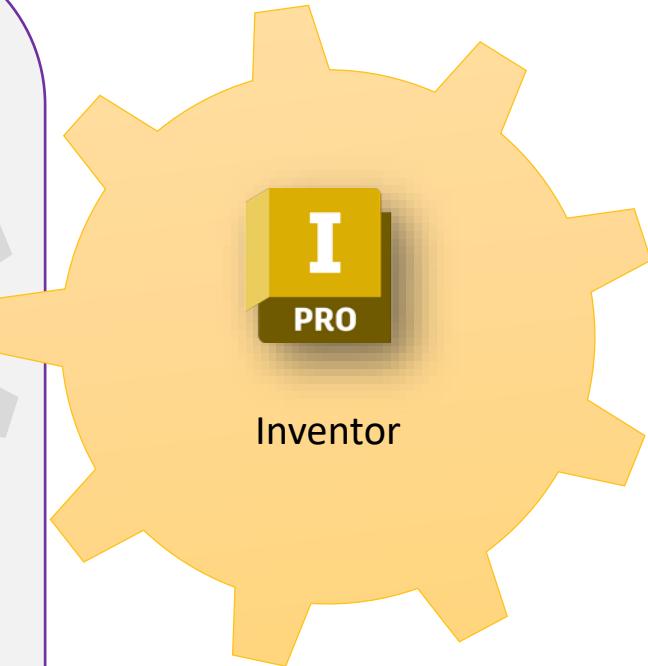
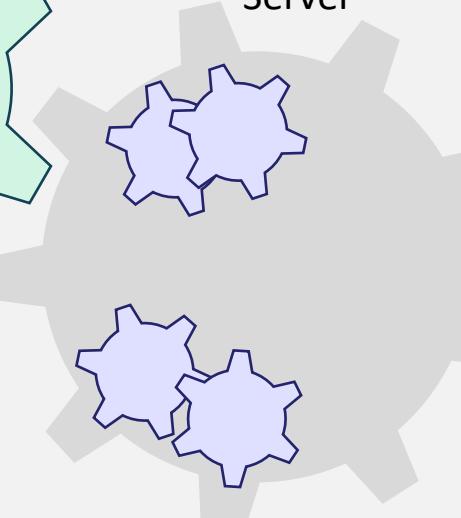
Inventor

# Understanding the Add-in Template Structure

Utilities



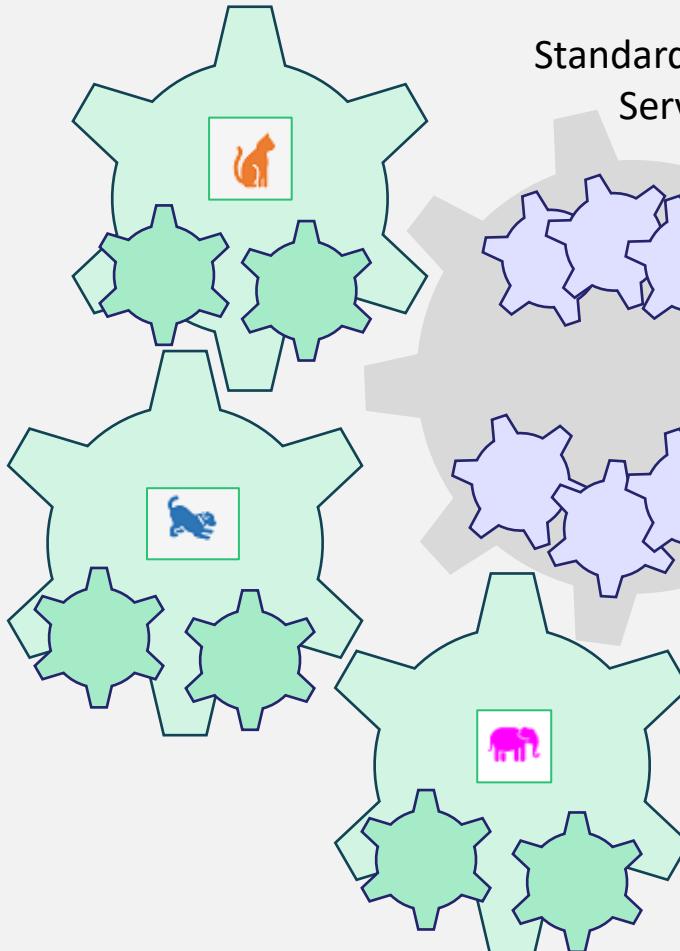
Standard Add-in  
Server



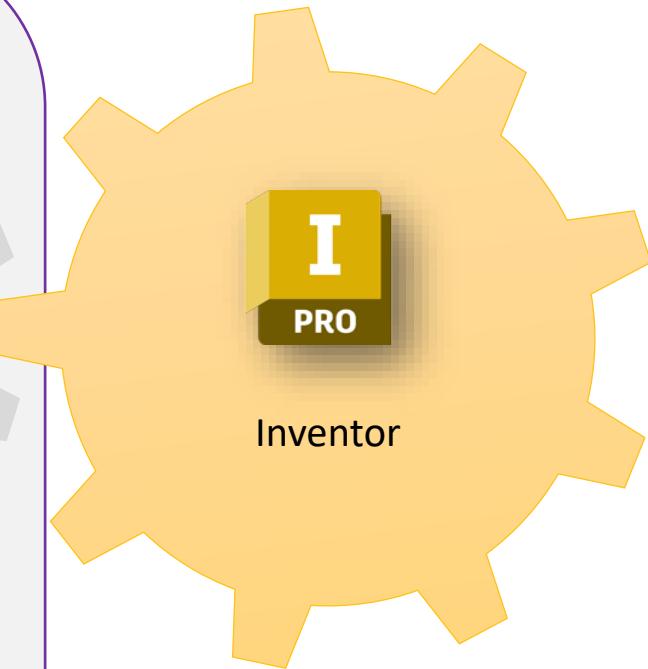
Inventor

# Understanding the Add-in Template Structure

Utilities



Standard Add-in  
Server



Inventor

# Modifying the Add-in

Working in Visual Studio

# Setup Information



A screenshot of Microsoft Visual Studio showing the code for the StandardAddinServer file. The code defines several buttons and includes a region labeled "Setup Information". A red arrow points to the start of this region. A dashed box highlights the line of code that creates a list of environments:

```
43     'create list of environments to cycle through
44     Dim EnvironmentList As New List(Of String)({"Drawing", "Assembly", "Part"})
45
46     'define custom ribbon tab prefix and suffix
47     'this will be combined with the EnvironmentList to create ribbon tabs like "ACME Draw
48     Dim RibbonTabName_Prefix As String = "ACME"
49     Dim RibbonTabName_Suffix As String = "Tools"
50
51     'create list of environments to cycle through
52     Dim EnvironmentList As New List(Of String)({"Drawing", "Assembly", "Part"})
```

- We double click the **StandardAddinServer** file in the Solution Explorer 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



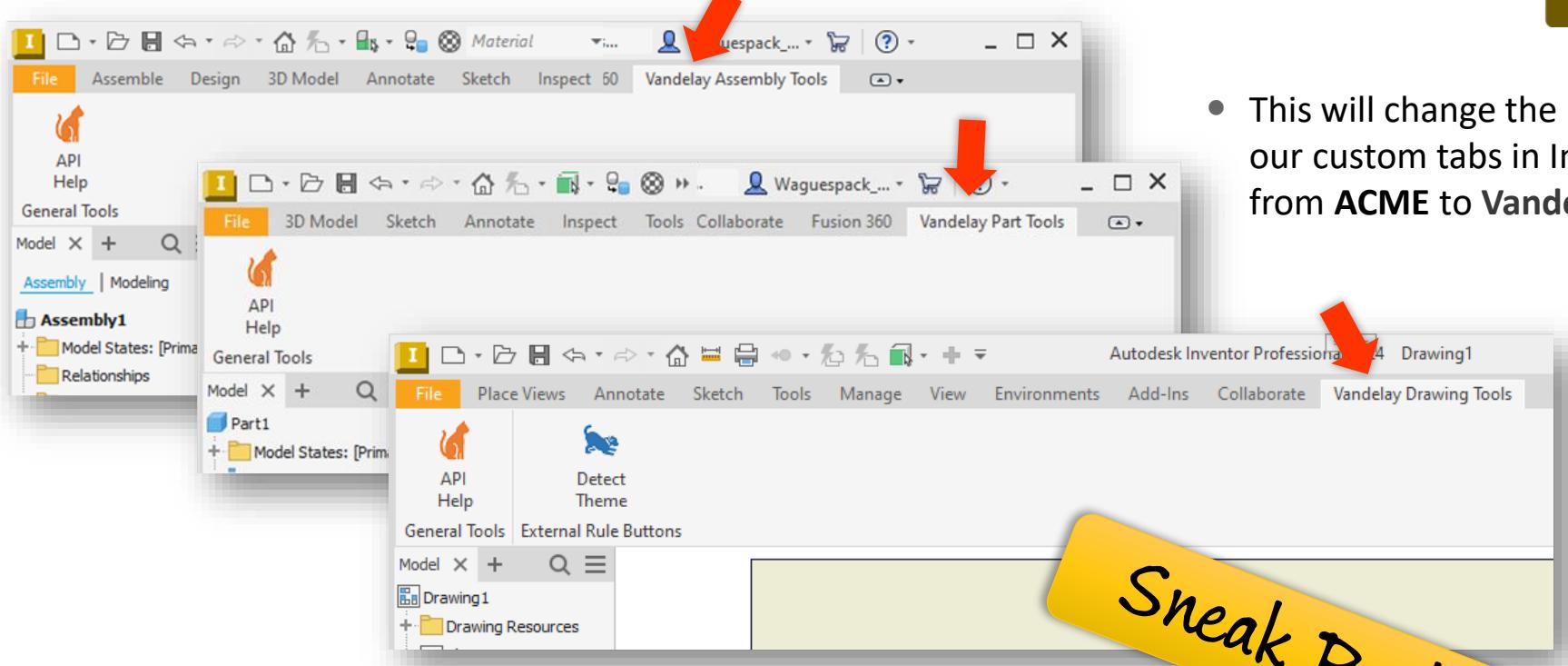
The screenshot shows the Microsoft Visual Studio IDE with the StandardAddInServer.vb file open. The code is written in VB.NET and defines a ribbon tab setup. A callout box highlights the modification of the 'RibbonTabName\_Prefix' variable from 'ACME' to 'Vandelay'. The code also includes logic for creating custom tabs and panels based on environments.

```
StandardAddInServer.vb*  StandardAddInServer  AddToUserInterface

34     'Define the command buttons
35     Private WithEvents API_Help_Button As ButtonDefinition
36     Private WithEvents Detect_Dark_Light_Theme_Button As ButtonDefinition
37     Private WithEvents Toggle_Sheet_Color As ButtonDefinition
38     Private WithEvents Balloon_Notify_Button As ButtonDefinition
39
40     2 references
41     Private Sub AddToUserInterface()
42         #Region "Setup Information"
43             'create a object collection and Control Definition array to be used with button stack
44             Dim buttonObjectCollection As ObjectCollection = g_inventorApplication.TransientObject
45             Dim ctrlDef(0 To 99) As ControlDefinition
46
47             'create list of environments to cycle through
48             Dim EnvironmentList As New List(Of String)( {"Drawing", "Assembly", "Part"})
49
50             'define custom ribbon tab prefix and suffix
51             'this will be combined with the EnvironmentList to create ribbon tabs like "ACME Draw"
52             Dim RibbonTabName_Prefix As String = "ACME"
53             Dim RibbonTabName_Suffix As String = "Tools"
54
55             'create list of panels to create on the custom
56             Dim CustomPanelList As New List(Of String)( {"Ge
57         #End Region
58
59         'Create Custom Tabs and Panels for each environment in the
60
61         'Create Buttons
62
63             End Sub
64
65             'Button 'on click' Events
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100 %  No issues found  Add to Source Control  Select Repository  Ready
```

- 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

The screenshot shows the Autodesk Inventor development environment. On the left, the Solution Explorer window displays the project structure for 'My\_New\_Addin'. The 'StandardAddInServer.vb' file is selected. On the right, the ribbon bar is visible with two tabs: 'General Tools' and 'External Rule Buttons'. Under 'General Tools', there are two buttons: 'API Help' (represented by a fox icon) and 'Detect Theme' (represented by a cat icon). A callout box points from the 'Create Buttons' section of the list to the 'External Rule Buttons' tab in the ribbon, indicating that the buttons shown are defined in external manifest files.

Solution Explorer

Solution 'My\_New\_Addin' (1 of 1 project)

- My\_New\_Addin
  - My Project
  - References
  - Command Tools
    - API\_Help.vb
    - Detect\_Dark\_Light\_Theme.vb
    - Welcome.vb
  - Resources
  - Utility Tools
  - AssemblyInfo.vb
  - Autodesk.My\_New\_Addin.Inventor.ad
  - My\_New\_Addin.X.manifest
  - Readme.txt
- StandardAddInServer.vb

Solution Explorer

Solution 'My\_New\_Addin' (1 of 1 project)

- My\_New\_Addin
  - My Project
  - References
  - Command Tools
    - API\_Help.vb
    - Detect\_Dark\_Light\_Theme.vb
    - Welcome.vb
  - Resources
  - Utility Tools
  - AssemblyInfo.vb
  - Autodesk.My\_New\_Addin.Inventor.ad
  - My\_New\_Addin.X.manifest
  - Readme.txt
- StandardAddInServer.vb

General Tools External Rule Buttons

API Help Detect Theme

I PRO



# Modifying the Add-in

The screenshot shows the Microsoft Visual Studio IDE interface. On the left, the code editor displays the `StandardAddinServer.vb` file under the `My_New_Addin` namespace. The code defines a class `StandardAddInServer` that implements `Inventor.ApplicationAddInServer`. It also declares several event handlers for application events like `WithEvents InventorApplicationEvents As ApplicationEvents`. In the middle section, there are comments for defining command buttons. On the right, the Solution Explorer pane shows the project structure for `'My_New_Addin'`, including files like `My_New_Addin.csproj`, `VB API_Help.vb`, `VB Detect_Dark_Light_Theme_Button.cs`, `VB Welcome.vb`, `Resources`, `Utility Tools`, and `Autodesk.My_New_Addin.manifest`.

```
Imports ...  
'Define global variables  
Public Module Globals ...  
  
Namespace My_New_Addin  
    <ProgIdAttribute("My_New_Addin.StandardAddInServer"),  
     GuidAttribute(g_simpleg_addInClientID)>  
    Public Class StandardAddInServer  
        Implements Inventor.ApplicationAddInServer  
  
        'Application events  
        Private WithEvents InventorApplicationEvents As ApplicationEvents  
        Private WithEvents UIEvents As UserInterfaceEvents  
        Private WithEvents UserInputEvents As UserInputEvents  
        Private WithEvents InvTransactionEvents As TransactionEvents  
  
        'Define the command buttons  
        Private WithEvents API_Help_Button As ButtonDefinition  
        Private WithEvents Detect_Dark_Light_Theme_Button As ButtonDefinition  
  
        2 references  
        Private Sub AddToUserInterface()  
  
        #Setup Information  
  
        #Create Custom Tabs and Panels for each environment in the list  
  
        #Create Single Buttons  
  
        End Sub
```

- 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



The screenshot shows the Microsoft Visual Studio interface. The code editor window displays the file `StandardAddInServer.vb`. A new line of code has been added:

```
Private WithEvents Toggle_Sheet_Color_Button As ButtonDefinition
```

The Solution Explorer window on the right shows the project structure for "My\_New\_Addin".

- File
- Edit
- View
- Git
- Project
- Build
- Debug
- Test
- Analyze
- Tools
- Extensions
- Window
- Help
- Search (Ctrl+F)

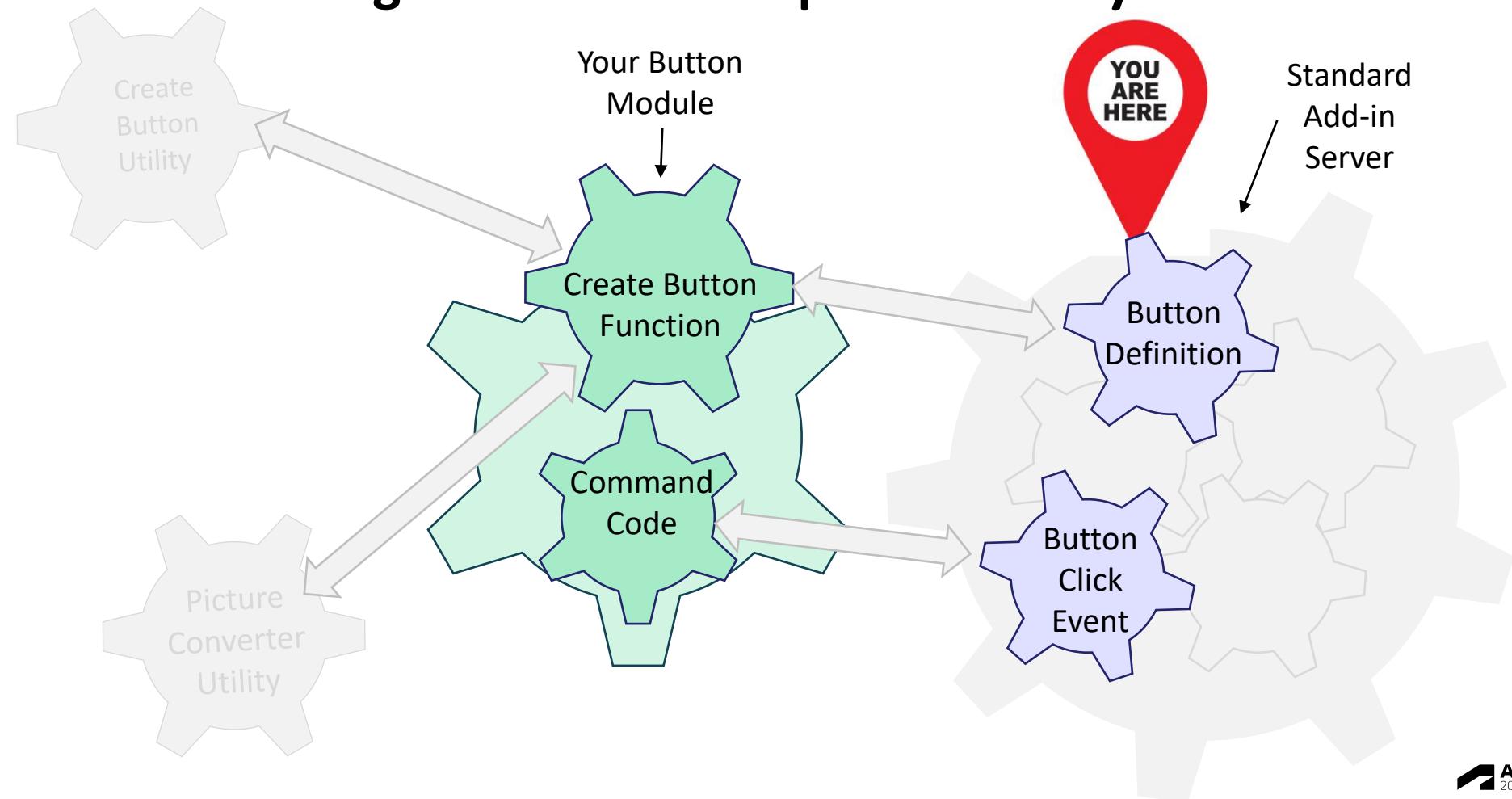
Debug Any CPU Start Live Share

Solution Explorer

- My\_New\_Addin (1 of 1)
  - My Project
  - References
    - Command Tools
      - VB API\_Help.vb
      - VB Detect\_Dark\_Light\_Theme.vb
      - VB Welcome.vb
    - Resources
    - Utility Tools
      - VB AssemblyInfo.vb
      - Autodesk.My\_New\_Addin.manifest
      - My\_New\_Addin.Xmanifest
      - Readme.txt
    - VB StandardAddInServer.vb

- 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



```
VB My_New_Addin
 35  Private WithEvents API_Help_Button As ButtonDefinition
 36  Private WithEvents Detect_Dark_Light_Theme_Button As ButtonDefinition
 37  Private WithEvents Toggle_Sheet_Color_Button As ButtonDefinition
 38
 39
 40  Private Sub AddToUserInterface()
 41
 42  #Setup Information
 43
 44  #Create Custom Tabs and Panels for each environment in the list
 45
 46  #Region "Create Buttons"
 47
 48    'add this button to General Tools tab of 3 different environments
 49    API_Help_Button = API_Help.CreateButton("Drawing", CustomDrawingTab, Drawing_GeneralToolsPanel, True, False)
 50    API_Help_Button = API_Help.CreateButton("Assembly", CustomAssemblyTab, Assembly_GeneralToolsPanel, True, False)
 51    API_Help_Button = API_Help.CreateButton("Part", CustomPartTab, Part_GeneralToolsPanel, True, False)
 52
 53    'add button to just one tab
 54    Detect_Dark_Light_Theme_Button =
 55      Detect_Dark_Light_Theme.CreateButton("Drawing", CustomDrawingTab, Drawing_ExternalRuleButtonsPanel, True, False)
 56
 57  #End Region
 58
 59
 60  End Sub
 61
 62
 63
 64
 65  #Button 'on click' Events
 66
 67  #Application Events
 68
 69  #ApplicationAddInServer Members
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
 100
```

The screenshot shows the Microsoft Visual Studio IDE interface with the code editor open. The code is written in VB.NET and defines methods for adding user interface elements to a Microsoft Office add-in. A red arrow points to the line '#Region "Create Buttons"' at line 126. The code within this region creates buttons for the 'Drawing', 'Assembly', and 'Part' tabs in the 'General Tools' panel. Below this region, there are sections for 'Button 'on click' Events', 'Application Events', and 'ApplicationAddInServer Members'. The status bar at the bottom indicates 'Ready'.

# Modifying the Add-in



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

```
StandardAddInServer.vb*  StandardAddInServer
My_New_Addin
35  Define the 'Command Tools'
36  Private WithEvents API_Help_Button As ButtonDefinition
37  Private WithEvents Detect_Dark_Light_Theme_Button As ButtonDefinition
38  Private WithEvents Toggle_Sheet_Color_Button As ButtonDefinition
39
40  2 references
41  Private Sub AddToUserInterface()
42
43  #Setup Information
44
45  #Create Custom Tabs and Panels for each environment in the list
46
47  #Region "Create Buttons"
48
49  'add this button to General Tools tab of 3 different environments
50  API_Help_Button = API_Help.CreateButton("Drawing", CustomDrawingTab, Drawing_GeneralToolsPanel)
51  API_Help_Button = API_Help.CreateButton("Assembly", CustomAssemblyTab, Assembly_GeneralToolsPanel)
52  API_Help_Button = API_Help.CreateButton("Part", CustomPartTab, Part_GeneralToolsPanel)
53
54  #End Region
55
56  End Sub
57
58
59  #Button 'on click' Events
60
61  #Application Events
62
63  #ApplicationAddInServer Members
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
```

No issues found | Add to Source Control | Select Repository | Ready

# 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

The screenshot shows the Microsoft Visual Studio interface with the following details:

- Solution Explorer:** Displays the solution 'My\_New\_Addin' with one project 'My\_New\_Addin'. Inside the project, there are several files: 'My Project', 'References', 'Command Tools' (containing 'VB API\_Help.vb'), 'VB Detect\_Dark\_Light\_Theme.vb', 'VB Welcome.vb', 'Resources', 'Utilities', 'AssemblyInfo.vb', 'Autodesk.My\_New\_Addin.Inventor.addin', 'My\_New\_Addin.X.manifest', 'Readme.txt', and 'VB StandardAddInServer.vb'.
- Code Editor:** Shows the file 'StandardAddInServer.vb'. The code includes sections for 'Setup Information', 'Create Custom Tabs and Panels for', and 'Create Buttons'. It defines a class 'StandardAddInServer' with methods like 'AddToUserInterface()' and 'Detect\_Dark\_Light\_Theme()'. A red arrow points to the 'Copy' option in the context menu for 'VB API\_Help.vb'.
- Context Menu:** A context menu is open over 'VB API\_Help.vb' in the Solution Explorer. The 'Copy' option is highlighted with a blue selection bar and a green arrow pointing to it from the top right.
- Status Bar:** At the bottom, the status bar shows '100 %', 'No issues found', 'Ln: 137 Ch: 1 SPC CRLF', and navigation icons.
- Bottom Bar:** Includes 'Ready', 'Add to Source Control', 'Select Repository', and a small icon with the number '1'.

# Modifying the Add-in



- Right click on the Command Tools folder and choose Paste

The screenshot shows the Microsoft Visual Studio interface. On the left is the StandardAddInServer.vb code editor, displaying VB.NET code for an Inventor add-in. In the center is the Solution Explorer window, which lists the project 'My\_New\_Addin' with its files: My\_New\_Addin.vb, References, Command Tools, and API.Help.vb. A context menu is open over the 'Command Tools' folder, with the 'Paste' option highlighted by a red arrow. To the right of the menu, another Solution Explorer window is shown, also listing the 'Command Tools' folder. A green arrow points from this second Solution Explorer window towards the 'Command Tools' folder in the main window's context menu.

```
VB My_New_Addin
35  Private WithEvents API_Help_Button As Button
36  Private WithEvents Detect_Dark_Light_Theme_
37  Private WithEvents Toggle_Sheet_Color_Button
38
39
40  Private Sub AddToInventorInterface()
41
42  #Region "Setup Information"
43
44  #Region "Create Custom Tabs and P
45
46  #Region "Create Buttons"
47
48  'add this bu
49  API_Help_But
50  API_Help_But
51  API_Help_But
52
53  'add button
54  Detect_Dark_
55  Detect_D
56
57  #End Region
58
59  End Sub
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
```

File Edit View Git Project Build Debug Test Analyze Tools Extended Help StandardAddInServer.vb\* > X Debug Any CPU Start Solution Explorer Search Solution Explorer (Ctrl+;) Solution 'My\_New\_Addin' (1 of 1 project) My\_New\_Addin My Project References Command Tools API.Help.vb Setup Information Create Custom Tabs and P Create Buttons Add Run Tests Debug Tests Scope to This New Solution Explorer View Exclude From Project Cut Ctrl+X Copy Ctrl+C Paste Ctrl+V Paste As Link Delete Del Rename F2 Copy Full Path Open Folder in File Explorer Open in Terminal Properties Alt+Enter

Ln: 137 Ch: 1 SPC CRLF Add to Source Control Select Repository 1

# Modifying the Add-in



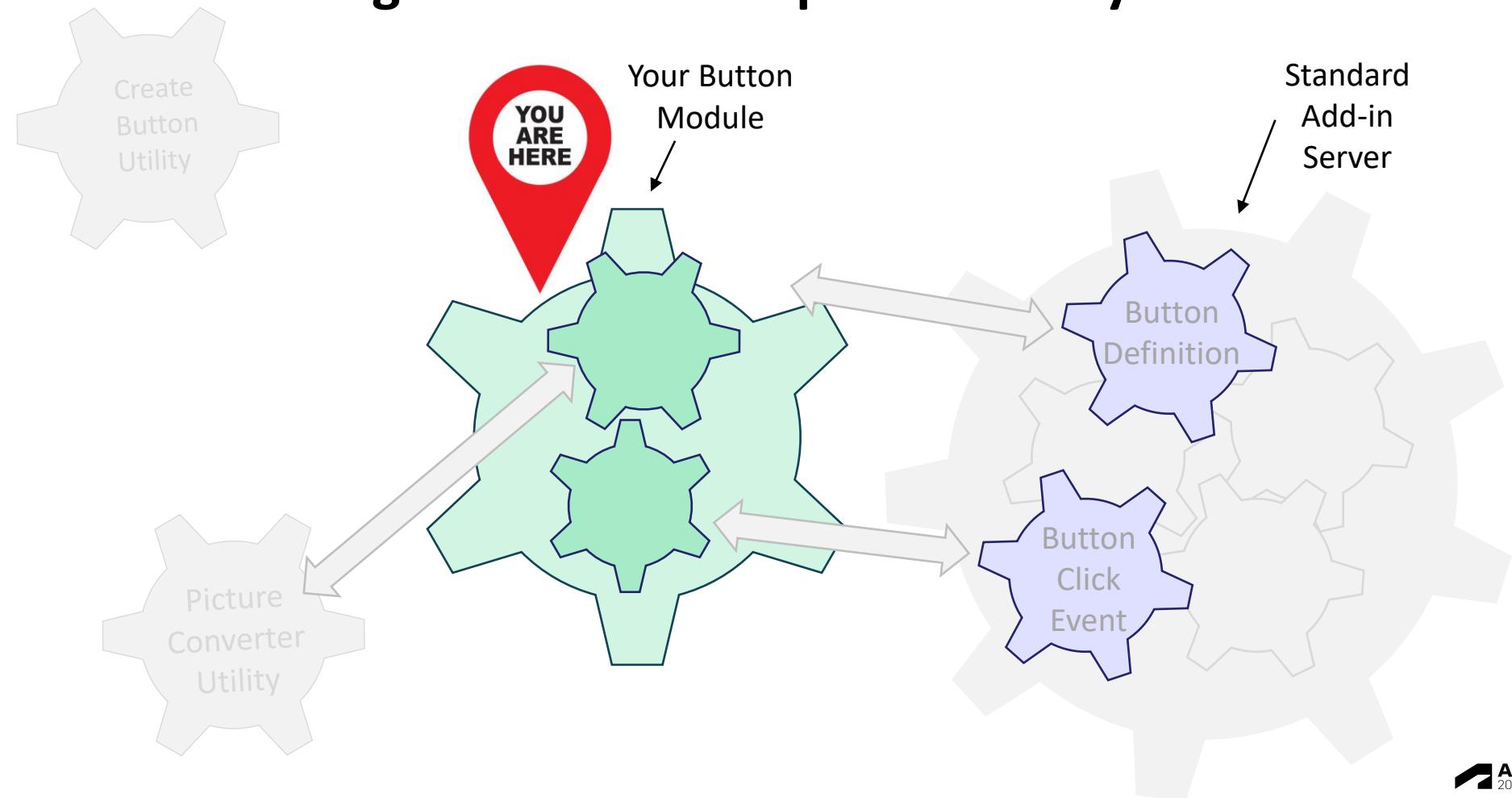
- Rename the copy

The screenshot shows the Microsoft Visual Studio interface with the following details:

- Solution Explorer:** Shows a single project named "My\_New\_Addin" which contains a folder "Command Tools" and files "VB API\_Help - Copy.vb", "VB API\_Help.vb", "VB Detect\_Dark\_Light\_Theme.vb", and "VB Welcome.vb".
- Code Editor:** The file "StandardAddInServer.vb" is open, showing VB.NET code for an add-in. It includes sections for defining command buttons, adding them to user interfaces, and handling application events.
- Toolbars and Menus:** Standard VS menus like File, Edit, View, etc., are visible at the top. A toolbar with icons for various operations is also present.
- Status Bar:** Shows "100 %", "No issues found", "Ln: 137 Ch: 1 SPC CRLF", and other status indicators.
- Visual Elements:** A red arrow points from the "VB API\_Help - Copy.vb" file in the Solution Explorer to its corresponding entry in the code editor's list. A green curved arrow indicates the movement of the file from the original location to its new location.

```
VB My_New_Addin
35     ' Define "the_CommandButtons" ...
36     Private WithEvents API_Help_Button As ButtonDefinition
37     Private WithEvents Detect_Dark_Light_Theme_Button As ButtonDefinition
38     Private WithEvents Toggle_Sheet_Color_Button As ButtonDefinition
39
40     Private Sub AddToUserInterface()
41
42     ' Setup Information
43
44     ' Create Custom Tabs and Panels for each environment in the list
45
46     '#Region "Create Buttons"
47
48         'add this button to General Tools tab of 3 different environments
49         API_Help_Button = API_Help.CreateButton("Drawing", CustomDrawingTab, Drawing_ExternalRule)
50         API_Help_Button = API_Help.CreateButton("Assembly", CustomAssemblyTab, Assembly_ExternalRule)
51         API_Help_Button = API_Help.CreateButton("Part", CustomPartTab, Part_General)
52
53         'add button to just one tab
54         Detect_Dark_Light_Theme_Button =
55             Detect_Dark_Light_Theme.CreateButton("Drawing", CustomDrawingTab, Drawing_ExternalRule)
56
57     '#End Region
58
59     End Sub
60
61     ' Button 'on click' Events
62
63     ' Application Events
64
65     ' ApplicationAddInServer Members
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

# Understanding the Add-in Template Visually



# Modifying the Add-in



The screenshot shows the Microsoft Visual Studio IDE interface. The Solution Explorer on the right lists the project 'My\_New\_Addin' with its files: My\_New\_Addin.vb, References, Command Tools, Utility Tools, Resources, and StandardAddInServer.vb. A red arrow points from the Solution Explorer to the module name 'API\_Help' in the code editor. The code editor displays the file 'Toggle\_Sheet\_Color.vb' with the following content:

```
Imports Inventor

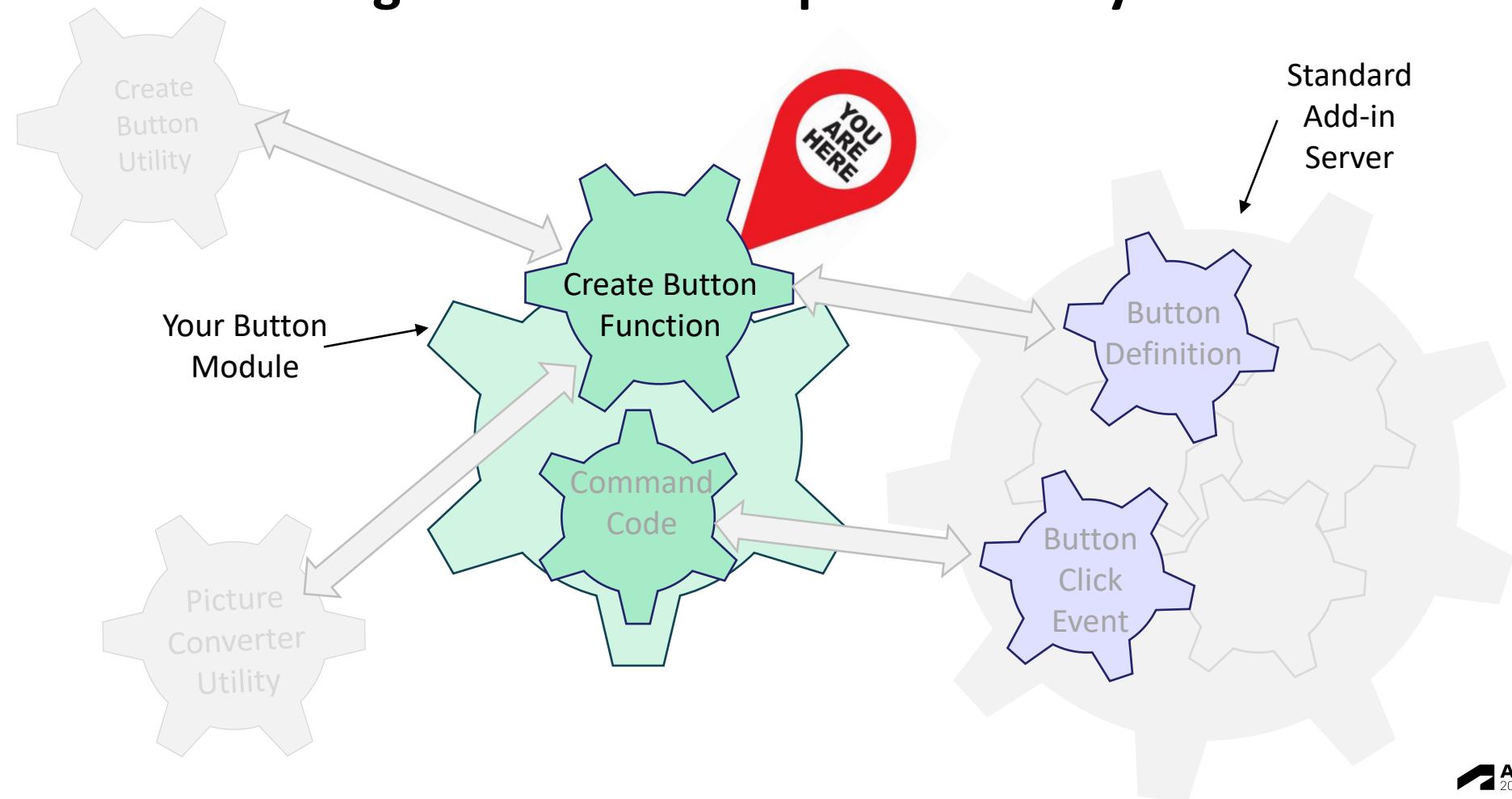
Module API_Help
    'Creates a button
    'Define the
    'Creates a button Button
    'This Function
    Function Create
        'get the information
        Dim largeIcon As IPictureDisp = PictureDispConverter.ToIPictureDisp(My.Resources)
        Dim standardIcon As IPictureDisp = PictureDispConverter.ToIPictureDisp(My.Resources)
        Dim toolTipImage As IPictureDisp = Nothing

        'this is the text the user sees on the button
        Dim buttonLabel As String = "API " & vbCrLf & "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
    End Function
End Module
```

- 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



The screenshot shows the Visual Studio IDE interface. A red arrow points from the top-left towards the 'Project' menu item in the top navigation bar. Another red arrow points from the bottom-left towards the 'My\_New\_Addin Properties' entry in the 'Project' menu dropdown. The main code editor window displays VBA code for a ribbon button definition:

```
    drawingTab As RibbonTab, ribbonPanel As RibbonPanel,
    buttonStack As Boolean) As ButtonDefinition
    ...
    Interator.ToIPictureDisp(My.Resources.Cat_32)
    Interator.ToIPictureDisp(My.Resources.Cat_16)
```

The code editor shows several comments explaining the properties of the button:

```
    'je As IPictureDisp = Nothing
    'ext the user sees on the button
    L As String = "API " & vbCrLf & "Help"
    'lays when the user hovers over the button
    mple As String = "Opens the API Help"
    banded As String = Nothing
```

The Solution Explorer on the right shows the project structure for 'My\_New\_Addin'.

- 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



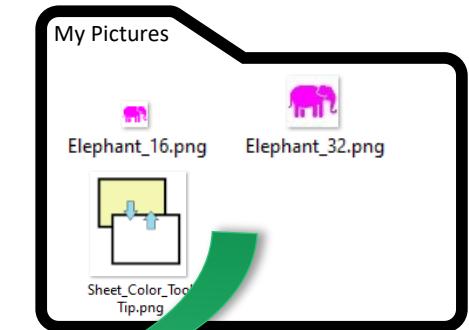
The screenshot shows the Microsoft Visual Studio interface with the 'My\_New\_Addin' project open. The 'Resources' tab is selected in the left sidebar. In the center, the 'Images' pane displays several icons: 'API\_ToolTip' (a help icon), 'Cat\_16' (orange cat), 'Cat\_32' (orange cat), 'Dog\_16' (blue dog), 'Dog\_32' (blue dog), 'Elephant\_16' (pink elephant), 'Elephant\_32' (pink elephant), and 'Sheet\_Color\_ToolTip' (a sheet of paper with arrows). A red box highlights the 'API\_ToolTip' icon, and a red arrow points from it to the 'Resources' tab in the sidebar. A yellow sticky note in the top right corner contains the following text:

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

The Solution Explorer pane on the right shows the project structure:

- My Project
- References
- VB API\_Help.vb
- VB Detect\_Dark\_Light\_Theme.vb
- VB Welcome.vb
- Resources
  - API\_ToolTip.png
  - Cat\_16.png
  - Cat\_32.png
  - Dog\_16.png
  - Dog\_32.png
  - Elephant\_16.png
  - Elephant\_32.png
- Utility Tools
- VB AssemblyInfo.vb
- Autodesk.My\_New\_Addin.manifest
- My\_New\_Addin.X.manifest
- Readme.txt
- VB StandardAddInServer.vb

- 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



A screenshot of the Microsoft Visual Studio IDE. The code editor window shows a portion of the file 'Toggle\_Sheet\_Color.vb' with the following code:

```
1
2
3    Else
4        On Button Definition
5
6        Disp(My.Resources.e)
7        ureDisp(My.Resources.e)
8
9        Disp(My.Resources.e)
10       ureDisp(My.Resources.e)
11      Elephan_16
12      Elephan_32
13      API.ToolTip
14      Cat_16
15      Cat_32
16      Culture
17      Dog_16
18      Dog_32
19      ResourceManager
20
21      onTab, ribbonPanel As RibbonPanel,
22      lean) As ButtonDefinition
23
24      sp(My.Resources.Cat_32)
25      eDisp(My.Resources.Cat_16)
```

The 'Solution Explorer' window on the right lists the project structure:

- My\_New\_Addin (My Project)
- References
- Command Tools
  - API\_Help.vb
  - Detect\_Dark\_Light\_Theme.vb
  - Toggle\_Sheet\_Color.vb
  - Welcome.vb
- Resources
- Utility Tools
- AssemblyInfo.vb
- Autodesk.My\_New\_Addin.lnk
- My\_New\_Addin.X.manifest
- Readme.txt
- StandardAddinServer.vb

- 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



The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and a search bar. The toolbar below has icons for file operations like Open, Save, and Print. The main code editor window displays VB.NET code for a Microsoft Office add-in. The code defines two buttons: one for toggling sheet colors and another for opening API help. The code uses string variables for button labels and tool tips, which are highlighted with yellow boxes. The Solution Explorer on the right shows the project structure, including files like StandardAddinServer.vb and CreateButton.vb. The status bar at the bottom shows the zoom level (100%), line number (47), character position (Ch: 12), column position (Col: 20), and file type (SPC CRLF).

```
Imports Inventor

Module Toggle_Sheet_Color
    'this is the text the user sees on the button
    Dim buttonLabel As String = "Toggle " & vbCrLf & "Sheet Color"

    'text that displays when the user hovers over the button
    Dim toolTip_Simple As String = "Toggle the drawing sheet color"
    Dim toolTip_Expanded As String = Nothing

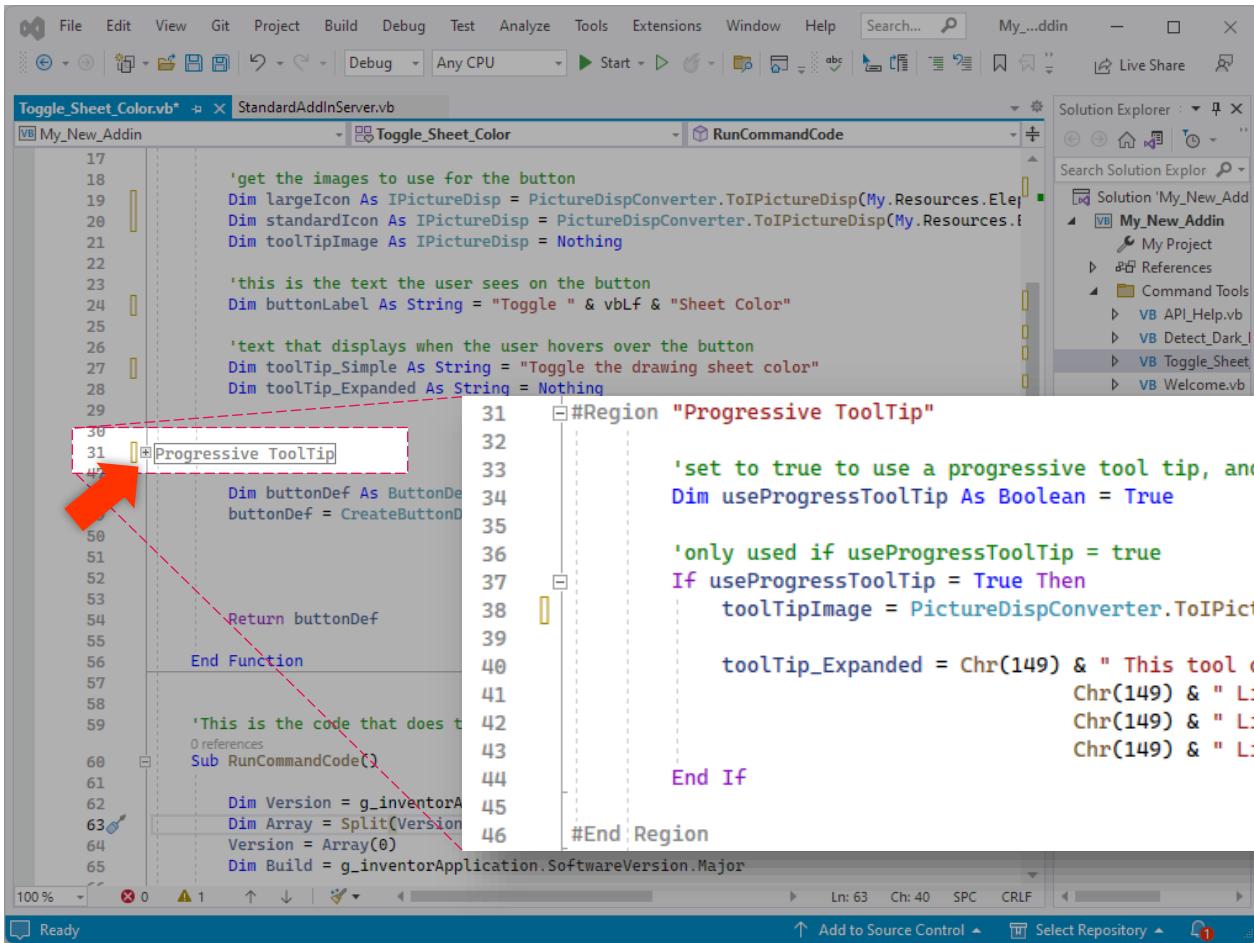
    'get the images to use for the button
    Dim largeIcon As IPictureDisp = PictureDispConverter.ToIPictureDisp(My.Resources.Tog
    Dim standardIcon As IPictureDisp = PictureDispConverter.ToIPictureDisp(My.Resources.
    Dim toolTipImage As IPictureDisp = Nothing

    'this is the text the user sees on the button
    Dim buttonLabel As String = "API " & vbCrLf & "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
End Module
```

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

# Modifying the Add-in



```
17
18
19     'get the images to use for the button
20     Dim largeIcon As IPictureDisp = PictureDispConverter.ToIPictureDisp(My.Resources.Ele...
21     Dim standardIcon As IPictureDisp = PictureDispConverter.ToIPictureDisp(My.Resources.I...
22     Dim tooltipImage As IPictureDisp = Nothing
23
24     'this is the text the user sees on the button
25     Dim buttonLabel As String = "Toggle " & vbCrLf & "Sheet Color"
26
27     'text that displays when the user hovers over the button
28     Dim toolTip_Simple As String = "Toggle the drawing sheet color"
29     Dim toolTip_Expanded As String = Nothing
30
31     #Region "Progressive ToolTip"
32
33         'set to true to use a progressive tool tip, and false to a simple tool tip
34         Dim useProgressToolTip As Boolean = True
35
36         'only used if useProgressToolTip = true
37         If useProgressToolTip = True Then
38             tooltipImage = PictureDispConverter.ToIPictureDisp(My.Resources.API_ToolTip)
39
40             toolTip_Expanded = Chr(149) & " This tool opens the API help *.chm file in a se...
41             Chr(149) & " Line2" & vbCrLf &
42             Chr(149) & " Line3" & vbCrLf &
43             Chr(149) & " Line4"
44
45         End If
46     #End Region
47
48     'This is the code that does t
49     0 references
50     Sub RunCommandCode()
51
52         Dim Version = g_inventorA...
53         Dim Array = Split(Version)
54         Version = Array(0)
55         Dim Build = g_inventorApplication.SoftwareVersion.Major
56
57
58
59
60
61
62
63
64
65
```

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

# Modifying the Add-in



The screenshot shows the Microsoft Visual Studio IDE interface. The title bar reads "My\_New\_Addin". The menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and a search bar "Search (Ctrl+Q)". The toolbar has various icons for file operations like Open, Save, and Print. The code editor displays VB.NET code for a ribbon button creation. A yellow callout box highlights the tooltip text being modified:

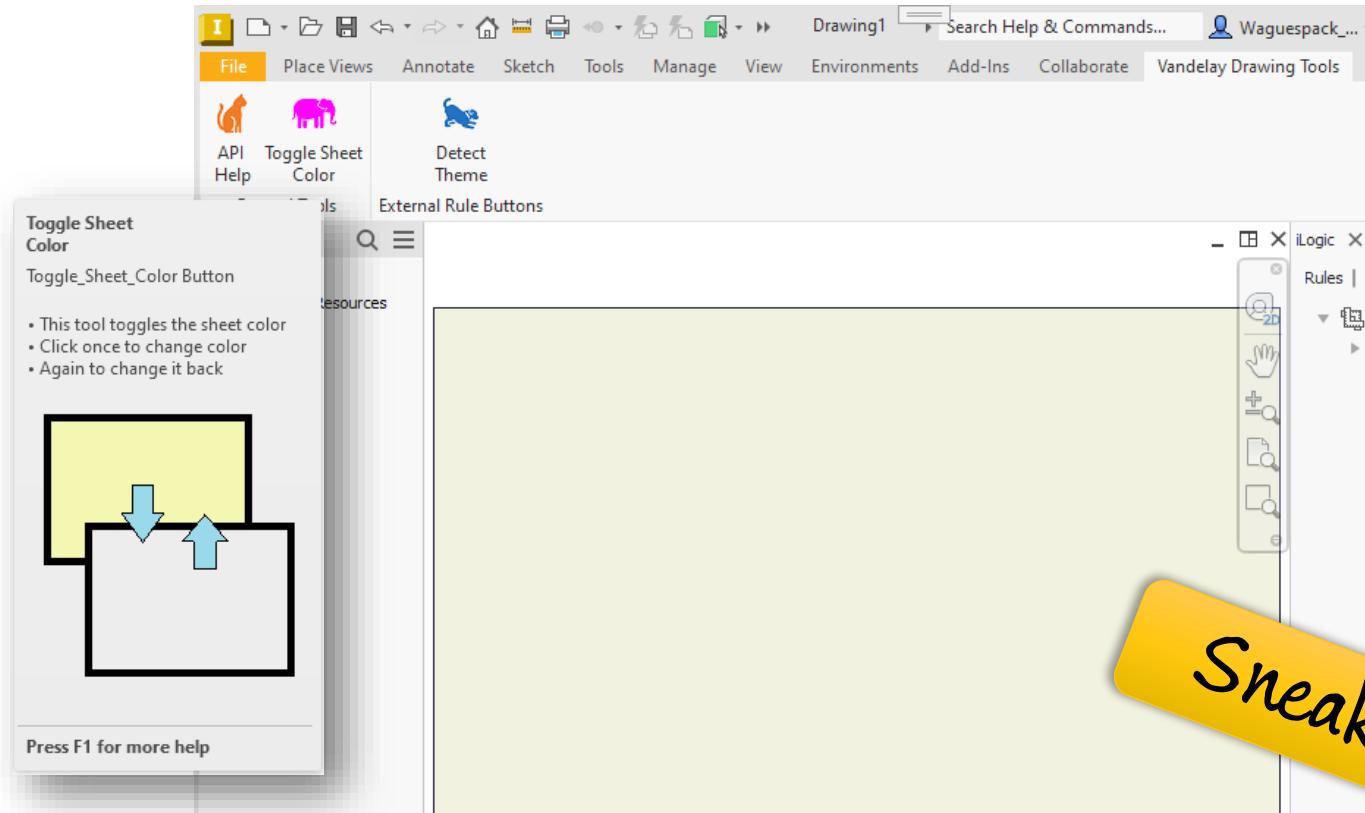
```
'only used if useProgressToolTip = true
If useProgressToolTip = True Then
    toolTipImage = PictureDispConverter.ToIPictureDisp(My.Resources.Sheet_Color_ToolTip)

    toolTip_Expanded = Chr(149) & " This tool toggles the sheet color" & vbCrLf &
                      Chr(149) & " Click once to change color" & vbCrLf &
                      Chr(149) & " Again to change it back"
End If
```

A red dashed box encloses the entire tooltip definition block, which includes the original API help text and the new sheet color toggle instructions. The status bar at the bottom shows "Ln: 24 Ch: 69 SPC CRLF".

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

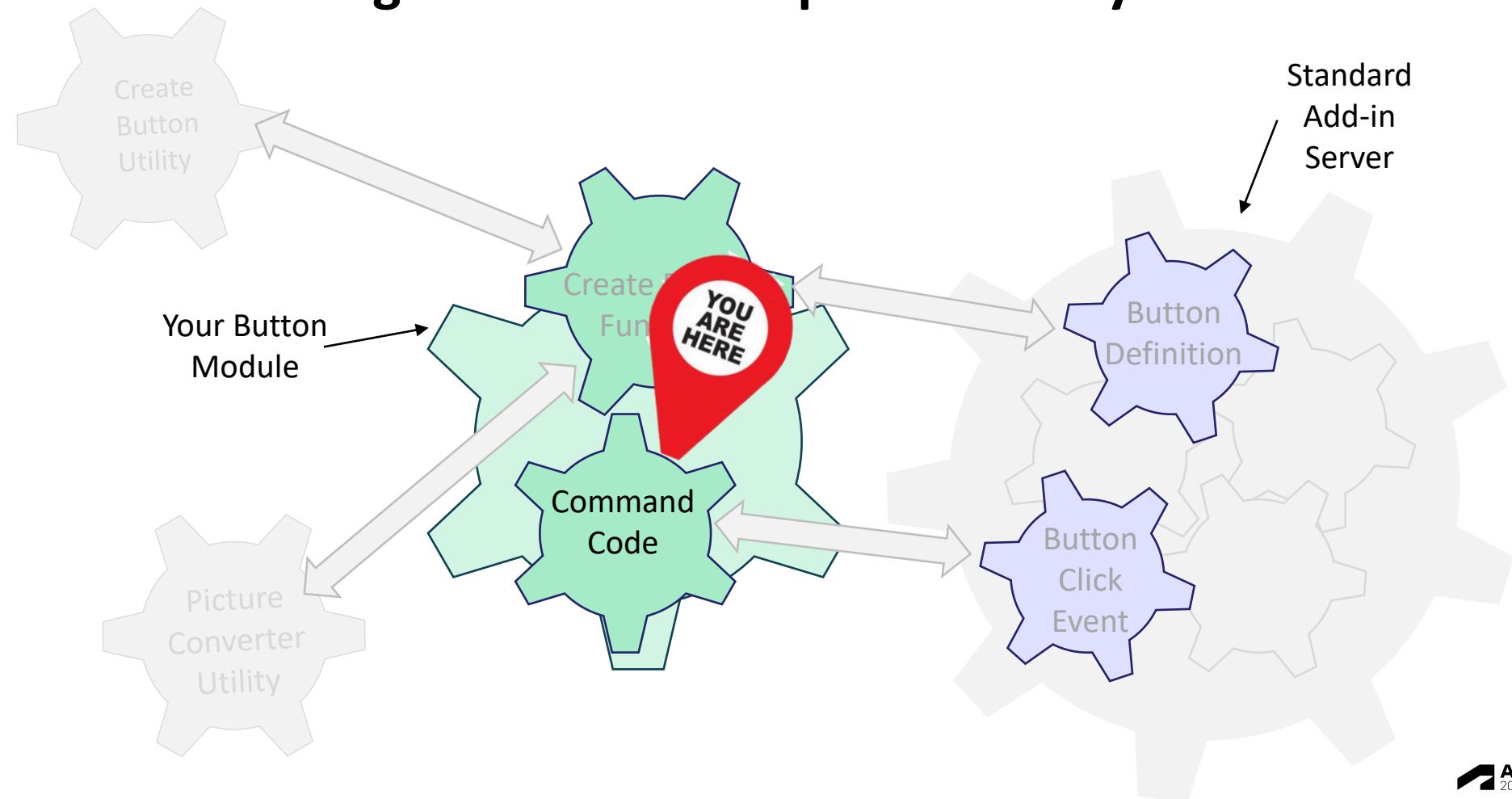
# Modifying the Add-in



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

Sneak Peak

# Understanding the Add-in Template Visually



# Modifying the Add-in



The screenshot shows the Microsoft Visual Studio IDE interface. The title bar says "My\_New\_Addin". The menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and a search bar. The toolbar has various icons for file operations. The main window shows the code editor with the file "Toggle\_Sheet\_Color.vb" open. The code defines a function "CreateButton" and a subroutine "RunCommandCode". The "RunCommandCode" subroutine contains logic to check if a specific help file exists and process it if found. A large red X is drawn over the entire "RunCommandCode" block. To the right is the Solution Explorer pane, which lists the solution "My\_New\_Addin" with one project "My\_New\_Addin" containing files like "API\_Help.vb", "Detect\_Dark\_Light\_Theme.vb", "Toggle\_Sheet\_Color.vb", "Welcome.vb", and various resource files. The bottom status bar shows "100% 0 1 Ln: 33 Ch: 77 SPC CRLF".

```
49
50
51
52
53     Return buttonDef
54
55 End Function
56
57
58 'This is the code that does the real work when your command is executed.
59 Sub RunCommandCode()
60
61     Dim Version = g_inventorApplication.SoftwareVersion.DisplayVersion
62     Dim Array = Split(Version, ".")
63     Version = Array(0)
64     Dim Build = g_inventorApplication.SoftwareVersion.Major
65
66     Dim Filename = "C:\Users\Public\Documents\Autodesk\Inventor " &
67     Version & "\Local Help\ADMAPI_" & Build & ".chm"
68
69     If System.IO.File.Exists(Filename) = True Then
70         Process.Start(Filename)
71     Else
72         MsgBox("File Does Not Exist" & vbCrLf & Filename)
73     End If
74
75 End Sub
76
77 End Module
```

- 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



The screenshot shows the Microsoft Visual Studio interface with a VBA project named "My\_New\_Addin". The code editor window displays the file "Toggle\_Sheet\_Color.vb" containing the following code:

```
49
50
51
52
53     Return buttonDef
54
55 End Function
56
57
58 'This is the code that does the real work when your command is executed.
59 Sub RunCommandCode()
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74     End Sub
75
76 End Module
77
78
```

The Solution Explorer window on the right lists the project structure:

- My\_New\_Addin (My Project)
- References
- Command Tools
  - VB API\_Help.vb
  - VB Detect\_Dark\_Light\_Theme.vb
  - VB Toggle\_Sheet\_Color.vb
  - VB Welcome.vb
- Resources
  - API\_ToolTip.png
  - Cat\_16.png
  - Cat\_32.png
  - Dog\_16.png
  - Dog\_32.png
  - Elephant\_16.png
  - Elephant\_32.png
  - Sheet\_Color\_ToolTip.png
- Utility Tools
  - VB AssemblyInfo.vb
  - Autodesk.My\_New\_Addin.Inventor
  - My\_New\_Addin.X.manifest
  - Readme.txt
  - VB StandardAddInServer.vb

- 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

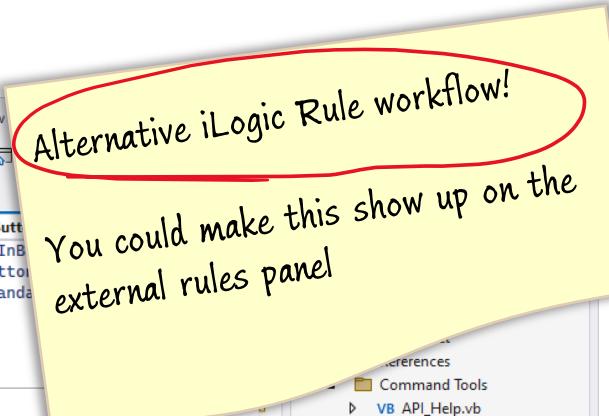


The screenshot shows the Microsoft Visual Studio interface. The code editor window displays a VB.NET module named 'Toggle\_Sheet\_Color.vb'. The module contains a Subroutine 'RunCommandCode()' which performs several tasks: it gets the active document, retrieves transient objects, and then the sheet color. It then extracts the RGB values from the color. Depending on the current color (white or black), it creates a new color object with either white or black as the background color. Finally, it sets the sheet color to this newly created color. The Solution Explorer window on the right shows the project structure for 'My\_New\_Addin'.

```
56
57
58     'This is the code that does the real work when your command is executed.
59     0 references
60     Sub RunCommandCode()
61
62         Dim Doc As DrawingDocument = g_inventorApplication.ActiveDocument
63
64         Dim TObj As TransientObjects = g_inventorApplication.TransientObjects
65
66         'get the color of the sheet
67         Dim Color As Color
68         Color = Doc.SheetSettings.SheetColor
69
70         'get the RGB from the color
71         Dim RGB As String
72         RGB = Color.Red & ", " & Color.Green & ", " & Color.Blue
73
74         'toggle sheet color
75         'if white, set to inventor default color
76         If RGB = "255, 255, 255" Then
77             Color = TObj.CreateColor(237, 237, 214)
78             'if inventor default color, set to white
79         ElseIf RGB = "237, 237, 214" Then
80             Color = TObj.CreateColor(255, 255, 255)
81         Else
82             'if something else set to inventor default color
83             Color = TObj.CreateColor(237, 237, 214)
84         End If
85
86         Doc.SheetSettings.SheetColor = Color
87
88     End Sub
89
End Module
```

- 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



```
File Edit View Git Project Build Debug Test Analyze Tools Extensions Window
Toggle_Sheet_Color.vb* | Debug Any CPU Start ...
VB My_New_Addin Toggle_Sheet_Color CreateButton
49
50
51
52
53
54
55
56
57
58     Return buttonDef
End Function

' This is the code that does the real work when your command is executed.
0 references
Sub RunCommandCode()
    ' This is the code that does the real work when your command is executed.
    0 references
    Sub Run_ExternalRule()
        Run_External_iLogic_Rule.RunExternalRule("Toggle Sheet Color")
    End Sub
End Sub

End Module
```

The screenshot shows the Microsoft Visual Studio IDE interface. A yellow sticky note is overlaid on the code editor with the handwritten text: "Alternative iLogic Rule workflow! You could make this show up on the external rules panel". The code in the editor is written in VB.NET. It contains two subroutines: "RunCommandCode" and "Run\_ExternalRule". The "RunCommandCode" subroutine is annotated with a large red X over its entire body, indicating it is no longer used. The "Run\_ExternalRule" subroutine calls the "RunExternalRule" method from the "Run\_External\_iLogic\_Rule" class with the argument "Toggle Sheet Color". The solution explorer on the right shows files like "VB API\_Help.vb", "VB Detect\_Dark\_Light\_Theme.vb", "VB Toggle\_Sheet\_Color.vb", and "VB Welcome.vb". The task list at the bottom shows items like "Add to Source Control" and "Select Repository".

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

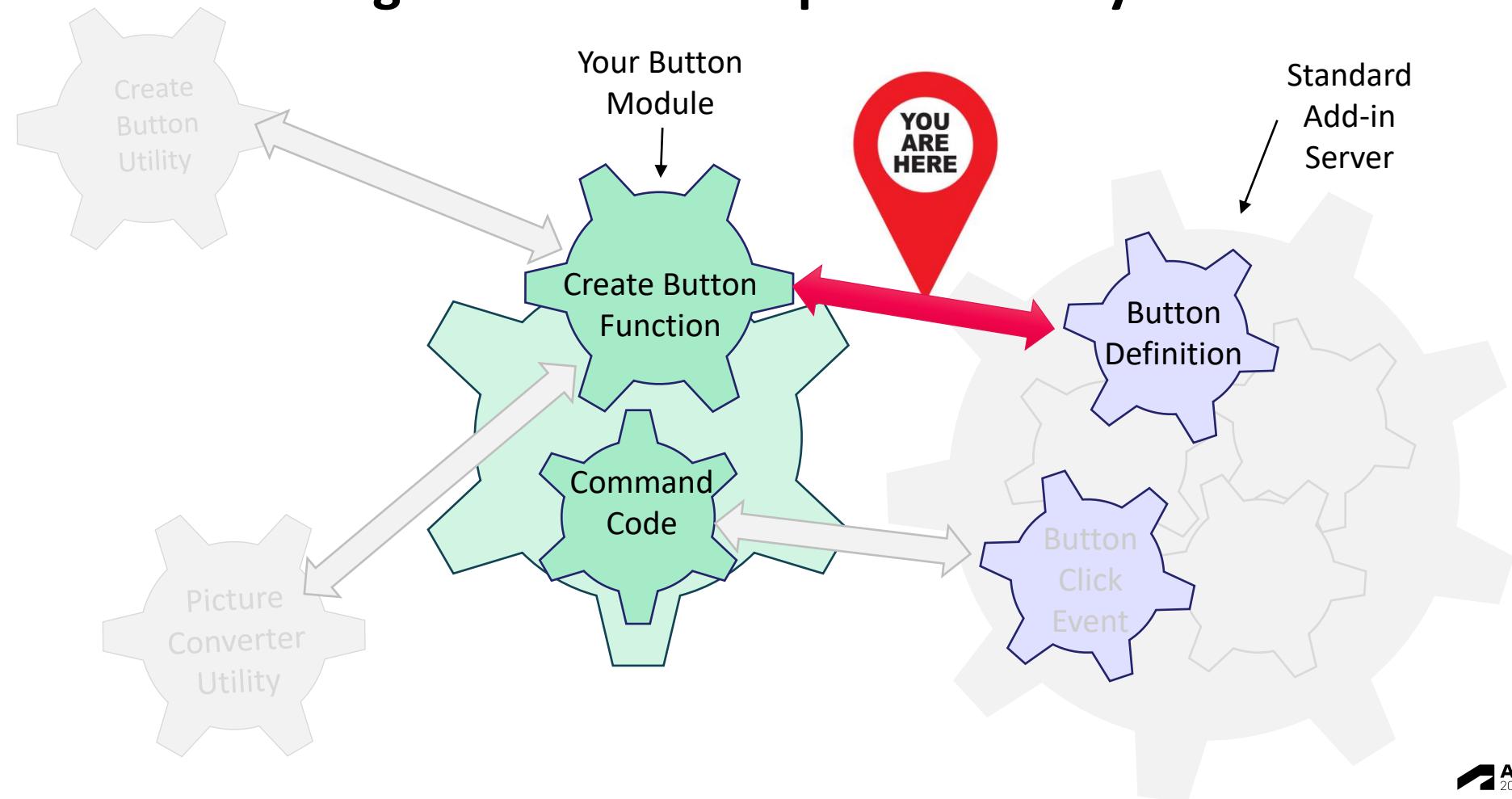


The screenshot shows the Microsoft Visual Studio interface. In the center, there is a code editor window displaying VB.NET code for a module named `Toggle_Sheet_Color`. The code implements a command to toggle the sheet color in a drawing document. A red arrow points to the close button (X) in the top-left corner of the code editor window. To the right of the code editor is a solution explorer window showing a project named `My_New_Addin` with files like `My_New_Addin.manifest` and `StandardAddInServer.vb`. In the foreground, a modal dialog box from Microsoft Visual Studio is displayed, asking "Save changes to the following items?". It lists the file `Command Tools\Toggle_Sheet_Color.vb*`. A large red arrow points to the "Save" button at the bottom of this dialog.

```
56
57
58     'This is the code that does the real work when your command is executed.
59     0 references
60     Sub RunCommandCode()
61
62         Dim Doc As DrawingDocument = g_inventorApplication.ActiveDocument
63
64         Dim TObj As TransientObjects = g_inventorApplicat
65
66         'get the color of the sheet
67         Dim Color As Color
68         Color = Doc.SheetSettings.SheetColor
69
70         'get the RGB from the color
71         Dim RGB As String
72         RGB = Color.Red & ", " & Color.Green & ", " & Col
73
74         'toggle sheet color
75         'if white, set to inventor default color
76         If RGB = "255, 255, 255" Then
77             Color = TObj.CreateColor(237, 237, 214)
78             'if inventor default color, set to white
79         ElseIf RGB = "237, 237, 214" Then
80             Color = TObj.CreateColor(255, 255, 255)
81             Else
82                 'if something else set to inventor default co
83                 Color = TObj.CreateColor(237, 237, 214)
84             End If
85
86             Doc.SheetSettings.SheetColor = Color
87
88         End Sub
89
90     End Module
```

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

# Understanding the Add-in Template Visually



# Modifying the Add-in



```
StandardAddInServer.vb  Toggle_Sheet_Color.vb
My_New_Addin
Private WithEvents API_Help_Button As ButtonDefinition
Private WithEvents Detect_Dark_Light_Theme_Button As ButtonDefinition
Private WithEvents Toggle_Sheet_Color_Button As ButtonDefinition

Private Sub AddToUserInterface()
    'Define the command buttons
    Private WithEvents API_Help_Button As ButtonDefinition
    Private WithEvents Detect_Dark_Light_Theme_Button As ButtonDefinition
    Private WithEvents Toggle_Sheet_Color_Button As ButtonDefinition

    'Setup Information
    'Create Custom Tabs and Panels for each environment in the list
    '#Region "Create Buttons"
        'Add code here to Create Custom Tabs and Panels
    '#End Region
    'End Sub

    'Region "Button 'on click' Events"
        'Add button 'click' events to this region
    '#End Region
End Sub

#Region "Create Buttons"
    'Add code here to Create Buttons
#End Region

#Region "Button 'on click' Events"
    'Add button 'click' events to this region
#End Region

Private Sub Toggle_Sheet_Color_Button =
    Toggle_Sheet_Color.CreateButton("Drawing", CustomDrawingTab,
                                    Drawing_GeneralToolsPanel, True, False)

    Detect_Dark_Light_Theme_Button =
        Detect_Dark_Light_Theme.CreateButton("Drawing", CustomDrawingTab,
                                            Drawing_ExternalRuleButtonsPanel, True, False)
#End Region
End Sub

#Region "Button 'on click' Events"
    'Add button 'click' events to this region
#End Region
```

- 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



The screenshot shows the Microsoft Visual Studio interface with the following details:

- File Menu:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Search Bar:** Search (Ctrl+Q).
- Solution Explorer:** Shows the solution 'My\_New\_Addin' with files like My\_New\_Addin.vb, VB API\_Help.vb, VB Detect\_Dark\_Light\_Theme.vb, VB Toggle\_Sheet\_Color.vb, VB Welcome.vb, Resources, Utility Tools, VB AssemblyInfo.vb, Autodesk.My\_New\_Addin.Ini, My\_New\_Addin.X.manifest, Readme.txt, and VB StandardAddInServer.vb.
- Code Editor:** The file 'StandardAddInServer.vb' is open, showing VBA code. A specific line of code is highlighted:

```
Toggle_Sheet_Color_Button =  
    Toggle_Sheet_Color.CreateButton("Drawing", CustomDrawingTab,  
        Drawing_GeneralToolsPanel, True, False)
```

A black arrow points from the word 'Toggle\_Sheet\_Color' in this line to the corresponding declaration at the top of the code block.

```
Private WithEvents Toggle_Sheet_Color_Button As ButtonDefinition
```

A blue arrow points from the word 'CreateButton' in the highlighted line to its definition in the code block below.

```
API_Help_Button = API_Help.CreateButton("Drawing", CustomDrawingTab, Drawing_GeneralToolsPanel, True, False)  
API_Help_Button = API_Help.CreateButton("Assembly", CustomAssemblyTab, Assembly_GeneralToolsPanel, True, False)  
API_Help_Button = API_Help.CreateButton("Part", CustomPartTab, Part_GeneralToolsPanel, True, False)
```
- Status Bar:** Shows 'Ln: 39 Ch: 41 SPC CRLF'.
- Bottom Bar:** Ready, Add to Source Control, Select Repository, and a bell icon.

- 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



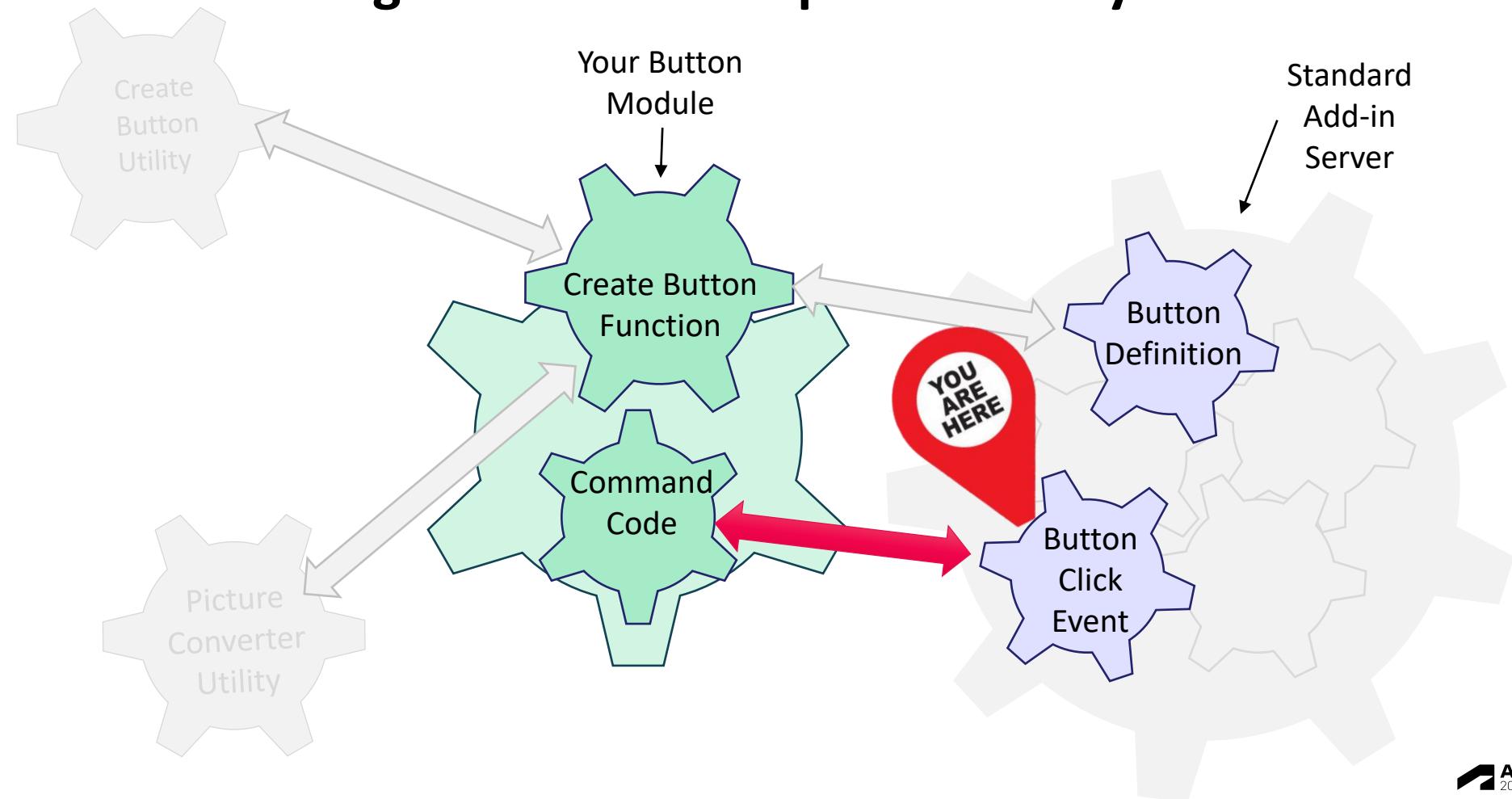
```
StandardAddInServer.vb  Toggle_Sheet_Color.vb
VB My_New_Addin  StandardAddInServer

33
34
35
36
37
38
39  Private Sub AddToUserInterface()
40
41  #Region "Setup Information"
42
43  #Region "Create Custom Tabs and Panels for each environment in the list"
44
45  #Region "Create Buttons"
46
47      'add this button to General Tools tab of 3 different environments
48      API_Help_Button = API_Help.CreateButton("Drawing", CustomDrawingTab, Drawing_GeneralToolsPanel)
49      API_Help_Button = API_Help.CreateButton("Assembly", CustomAssemblyTab, Assembly_GeneralToolsPanel)
50      API_Help_Button = API_Help.CreateButton("Part", CustomPartTab, Part_GeneralToolsPanel, True)
51
52      'add button to just one tab
53      Detect_Dark_Light_Theme_Button =
54          Detect_Dark_Light_Theme.CreateButton("Drawing", CustomDrawingTab,
55
56              Toggle_Sheet_Color_Button =
57                  Toggle_Sheet_Color.CreateButton("Drawing", CustomDrawingTab,
58
59                      Drawing_ExternalRuleButtonsPanel, True, False)
60
61
62  End Sub
63
64
65  #Region "Button 'on click' Events"
66  #End Region
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
```

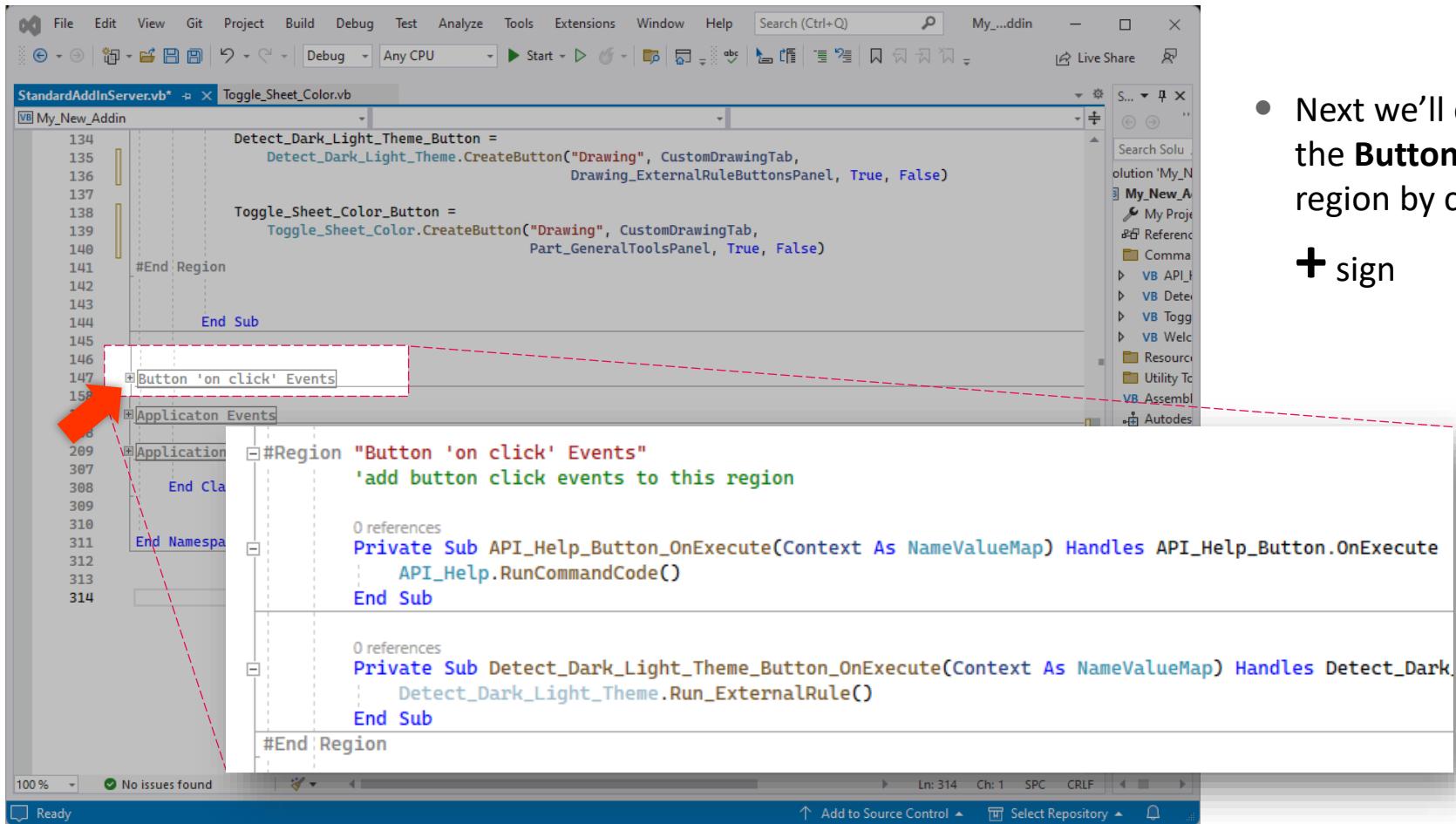
We'll add a line to connect the button definition we created earlier, with the Function in the module we just created.

- 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



```
File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Search (Ctrl+Q) My...ddin - X
StandardAddinServer.vb* Toggle_Sheet_Color.vb
VB My_New_Addin
134 Detect_Dark_Light_Theme_Button =
135     Detect_Dark_Light_Theme.CreateButton("Drawing", CustomDrawingTab,
136                                         Drawing_ExternalRuleButtonsPanel, True, False)
137
138 Toggle_Sheet_Color_Button =
139     Toggle_Sheet_Color.CreateButton("Drawing", CustomDrawingTab,
140                                     Part_GeneralToolsPanel, True, False)
141 #End Region
142
143 End Sub
144
145
146 #Region "Button 'on click' Events"
147
148 ApplicationEvents
149 Application
150 #Region "Button 'on click' Events"
151     'add button click events to this region
152
153     0 references
154     Private Sub API_Help_Button_OnExecute(Context As NameValueMap) Handles API_Help_Button.OnExecute
155         API_Help.RunCommandCode()
156     End Sub
157
158     0 references
159     Private Sub Detect_Dark_Light_Theme_Button_OnExecute(Context As NameValueMap) Handles Detect_Dark_Light_Theme_Button.OnExecute
160         Detect_Dark_Light_Theme.Run_ExternalRule()
161     End Sub
162 #End Region
163
164 End Class
165
166 End Namespace
167
168 End Class
169
170 End Module
171
172 End Class
173
174 End Class
175
176 End Class
177
178 End Class
179
180 End Class
181
182 End Class
183
184 End Class
185
186 End Class
187
188 End Class
189
190 End Class
191
192 End Class
193
194 End Class
195
196 End Class
197
198 End Class
199
200 End Class
201
202 End Class
203
204 End Class
205
206 End Class
207
208 End Class
209
210 End Class
211
212 End Class
213
214 End Class
215
216 End Class
217
218 End Class
219
220 End Class
221
222 End Class
223
224 End Class
225
226 End Class
227
228 End Class
229
230 End Class
231
232 End Class
233
234 End Class
235
236 End Class
237
238 End Class
239
240 End Class
241
242 End Class
243
244 End Class
245
246 End Class
247
248 End Class
249
250 End Class
251
252 End Class
253
254 End Class
255
256 End Class
257
258 End Class
259
260 End Class
261
262 End Class
263
264 End Class
265
266 End Class
267
268 End Class
269
270 End Class
271
272 End Class
273
274 End Class
275
276 End Class
277
278 End Class
279
280 End Class
281
282 End Class
283
284 End Class
285
286 End Class
287
288 End Class
289
290 End Class
291
292 End Class
293
294 End Class
295
296 End Class
297
298 End Class
299
299 End Class
300
300 End Class
301
301 End Class
302
302 End Class
303
303 End Class
304
304 End Class
305
305 End Class
306
306 End Class
307
307 End Class
308
308 End Class
309
309 End Class
310
310 End Class
311
311 End Class
312
312 End Class
313
313 End Class
314
314 End Class
100% No issues found Add to Source Control Select Repository Ready
```

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

# Modifying the Add-in



```
VB My_New_Addin
144     End Sub
145
146 #Region "Button 'on click' Events"
147     'add button click events to this region
148
149     0 references
150     Private Sub API_Help_Button_OnExecute(Context As NameValueMap) Handles API_Help_Button.OnExecute
151         API_Help.RunCommandCode()
152     End Sub
153
154     0 references
155     Private Sub Detect_Dark_Light_Theme_Button_OnExecute(Context As NameValueMap) Handles Detect_Dark_Light_Theme_
156         Detect_Dark_Light_Theme.Run_ExternalRule()
157     End Sub
158
159     0 references
160     Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
161         Toggle_Sheet_Color.RunCommandCode()
162     End Sub
163     #End Region
164     End Class
165
166     0 references
167     Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
168         Toggle_Sheet_Color.RunCommandCode()
169     End Sub
170
171     0 references
172     Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
173         Toggle_Sheet_Color.RunCommandCode()
174     End Sub
175
176     0 references
177     Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
178         Toggle_Sheet_Color.RunCommandCode()
179     End Sub
180
181     0 references
182     Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
183         Toggle_Sheet_Color.RunCommandCode()
184     End Sub
185
186     0 references
187     Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
188         Toggle_Sheet_Color.RunCommandCode()
189     End Sub
190
191     0 references
192     Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
193         Toggle_Sheet_Color.RunCommandCode()
194     End Sub
195
196     0 references
197     Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
198         Toggle_Sheet_Color.RunCommandCode()
199     End Sub
200
201     0 references
202     Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
203         Toggle_Sheet_Color.RunCommandCode()
204     End Sub
205
206     0 references
207     Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
208         Toggle_Sheet_Color.RunCommandCode()
209     End Sub
210
211     0 references
212     Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
213         Toggle_Sheet_Color.RunCommandCode()
214     End Sub
215
216     0 references
217     Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
218         Toggle_Sheet_Color.RunCommandCode()
219     End Sub
220
221     0 references
222     Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
223         Toggle_Sheet_Color.RunCommandCode()
224     End Sub
225
226     0 references
227     Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
228         Toggle_Sheet_Color.RunCommandCode()
229     End Sub
230
231     0 references
232     Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
233         Toggle_Sheet_Color.RunCommandCode()
234     End Sub
235
236     0 references
237     Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
238         Toggle_Sheet_Color.RunCommandCode()
239     End Sub
240
241     0 references
242     Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
243         Toggle_Sheet_Color.RunCommandCode()
244     End Sub
245
246     0 references
247     Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
248         Toggle_Sheet_Color.RunCommandCode()
249     End Sub
250
251     0 references
252     Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
253         Toggle_Sheet_Color.RunCommandCode()
254     End Sub
255
256     0 references
257     Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
258         Toggle_Sheet_Color.RunCommandCode()
259     End Sub
260
261     0 references
262     Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
263         Toggle_Sheet_Color.RunCommandCode()
264     End Sub
265
266     0 references
267     Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
268         Toggle_Sheet_Color.RunCommandCode()
269     End Sub
270
271     0 references
272     Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
273         Toggle_Sheet_Color.RunCommandCode()
274     End Sub
275
276     0 references
277     Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
278         Toggle_Sheet_Color.RunCommandCode()
279     End Sub
280
281     0 references
282     Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
283         Toggle_Sheet_Color.RunCommandCode()
284     End Sub
285
286     0 references
287     Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
288         Toggle_Sheet_Color.RunCommandCode()
289     End Sub
290
291     0 references
292     Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
293         Toggle_Sheet_Color.RunCommandCode()
294     End Sub
295
296     0 references
297     Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
298         Toggle_Sheet_Color.RunCommandCode()
299     End Sub
299
```

- Add a sub to run the Toggle\_Sheet\_Color Module command code when the button is executed

# Modifying the Add-in



StandardAddInServer.vb\* Toggle\_Sheet\_Color.vb

```
VB My_New_Addin
144
145
146
147 End Sub
148 #Region "Button 'on click' Events"
149 'add button click events to this region
150
151 Private Sub API_Help_Button_OnExecute(Context As NameValueMap) Handles API_Help.Button.OnExecute
152     API_Help.RunCommandCode()
153 End Sub
154
155 #Region "Detect Dark/Light Theme Button Events"
156 Private Sub Detect_Dark_Light_Theme_Button_OnExecute(Context As NameValueMap) Handles Detect_Dark_Light_Theme.Button.OnExecute
157     Detect_Dark_Light_Theme.Run_ExternalRule()
158 End Sub
159 #End Region
160
161 Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
162     Toggle_Sheet_Color.Run_ExternalRule()
163 End Sub
164
165 End Class
166
167 0 references
168 Private Sub Toggle_Sheet_Color_Button_OnExecute(Context As NameValueMap) Handles Toggle_Sheet_Color_Button.OnExecute
169     Toggle_Sheet_Color.Run_ExternalRule()
170 End Sub
```

0 references

Private Sub Toggle\_Sheet\_Color\_Button\_OnExecute(Context As NameValueMap) Handles Toggle\_Sheet\_Color\_Button.OnExecute  
    Toggle\_Sheet\_Color.Run\_ExternalRule()  
End Sub

100% No issues found | ↻ Ln: 312 Ch: 1 SPC CRLF | ↑ Add to Source Control | Select Repository | Ready

Alternative iLogic Rule workflow!  
You could make this show up on the external rules panel

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

# Overall Process



Visual  
Studio



- ✓ Write
- ✓ Debug
- ✓ Build

```
1 // Text that shows when the user
2 // has selected a single line or a group of lines, based on ca
3 // set to true to use a progressive tool tip, and false to a simple tool tip
4 // Set to true if userRequestToolTip = true
5 // Only used if userRequestToolTip = true
6 // User request tool tip is true if the user has right-clicked on
7 // a view and selected "Tool Tip" from the context menu
8 // Changes the edges in all views to a simple tool tip
9 // If false, the edges are pulled from the model col.
10 // Only used if userRequestToolTip = true
11 // User request tool tip is true if the user has right-clicked on
12 // a view and selected "Tool Tip" from the context menu
13 // Changes the edges in all views to a simple tool tip
14 // If false, the edges are pulled from the model col.
15 // Only used if userRequestToolTip = true
16 // User request tool tip is true if the user has right-clicked on
17 // a view and selected "Tool Tip" from the context menu
18 // Changes the edges in all views to a simple tool tip
19 // If false, the edges are pulled from the model col.
20 // Only used if userRequestToolTip = true
21 // User request tool tip is true if the user has right-clicked on
22 // a view and selected "Tool Tip" from the context menu
23 // Changes the edges in all views to a simple tool tip
24 // If false, the edges are pulled from the model col.
```

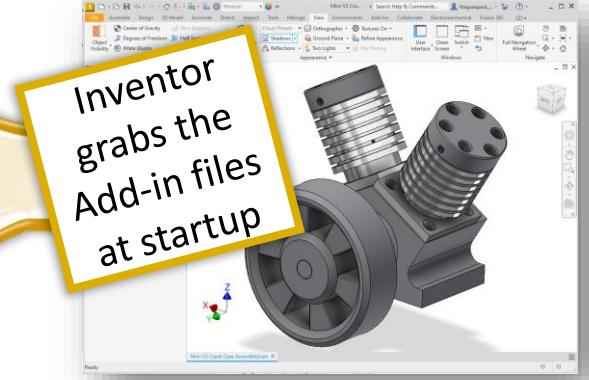
VS compiles the code and writes out the DLL and ADDIN files



Autodesk  
Application Plugins  
Folder



Autodesk  
Inventor



# Compiling the Add-in



The screenshot shows the Microsoft Visual Studio interface. In the center is the Solution Explorer window, which lists a single project named "My\_New\_Addin". A context menu is open over the project node, with the "Build Solution" option highlighted with a red box. To the left is the code editor showing VBA code for "StandardAddInServer.vb". The code includes regions for button click events and application events. At the bottom is the Output window displaying build logs.

```
StandardAddInServer.vb* Toggle_Sheet_Color.vb
VB My_New_Addin
144 End Sub
145
146 #Region "Button 'on click' Events"
147 'add button click events to this region
148
149 0 references
150 Private Sub API_Help_Button_OnExecute(Context As
151     API_Help.RunCommandCode()
152 End Sub
153
154 0 references
155 Private Sub Detect_Dark_Light_Theme_Button_OnExe
156     Detect_Dark_Light_Theme.Run_ExternalRule()
157 End Sub
158 #End Region
159 #Application Events
160 #ApplicationAddInServer Members
161
162 End Class
Output
Show output from: Build
1> INFO: Could not find files for the given pattern(s).
1> The system cannot find the path specified.
1> 'mt.exe' is not recognized as an internal or external command,
1> operable program or batch file.
1> D:\Autodesk University\AU 2023\Add Ins\My_New_Add-In\bin\Debug\My_New_A
1> 1 File(s) copied
1> D:\Autodesk University\AU 2023\Add Ins\My_New_Add-In\Autodesk.My_New_Ad
1> 1 File(s) copied
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
===== Build started at 4:25 PM and took 00.680 seconds =====
Error List Output
Build succeeded
Ready
```

- The solution is shown in the Solution Explorer



# Compiling the Add-in



The screenshot shows the Microsoft Visual Studio interface. The menu bar is visible at the top, with the 'Build' menu highlighted. A red arrow points from the text in the yellow sticky note to the 'Output' option in the 'Build' menu. The 'Output' tab is currently selected in the bottom-left corner. In the bottom status bar, there is a message 'Build succeeded'. A yellow sticky note is overlaid on the right side of the screen, containing handwritten text: 'Tip: you can turn the Output window on/off using the View tab'.

File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help

Any CPU Start Live Share

Solution Explorer

Git Changes

Git Repository

Team Explorer

Server Explorer

Data Lake Analytics Explorer

SQL Server Object Explorer

Test Explorer

Bookmark Window

Call Hierarchy

Class View

Code Definition Window

Object Browser

Error List

**Output**

Task List

Toolbox

Search (Ctrl+Q) My\_New\_AddIn

Output

Show output from: Build

1> INFO: Could not find files for the given pattern(s).

1> The system cannot find the path specified.

1> D:\Autodesk University\AU 2023\Add Ins\My\_New\_AddIn\bin\Debug\My\_New\_AddIn.dll

1> 1 File(s) copied

1> D:\Autodesk University\AU 2023\Add Ins\My\_New\_AddIn\Autodesk.My\_New\_AddIn.Inventor.addin

1> 1 File(s) copied

===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====

Add to Source Control Select Repository

Build succeeded

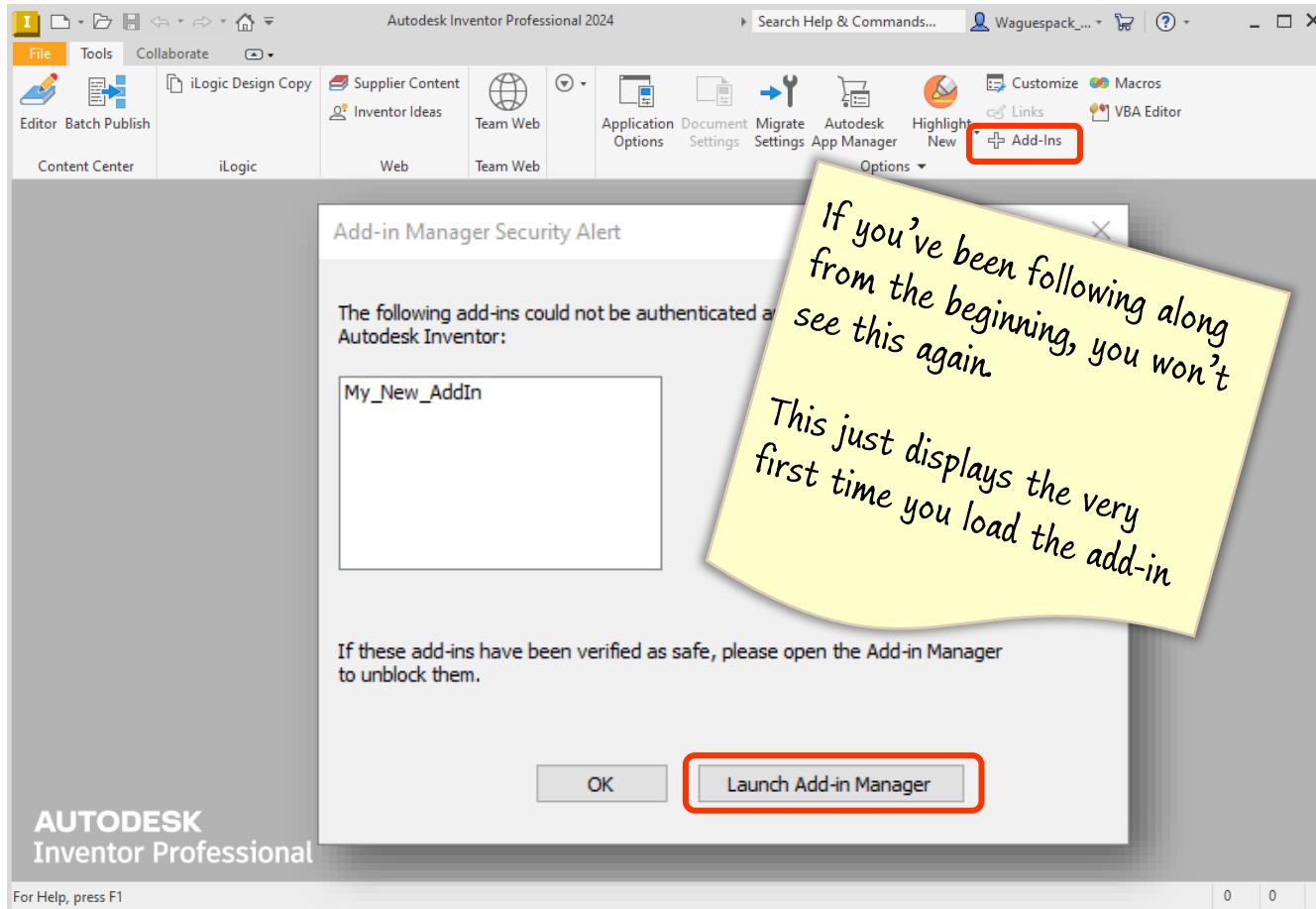
- The Build results are shown in the Output window

# Testing the add-in

Working in Inventor

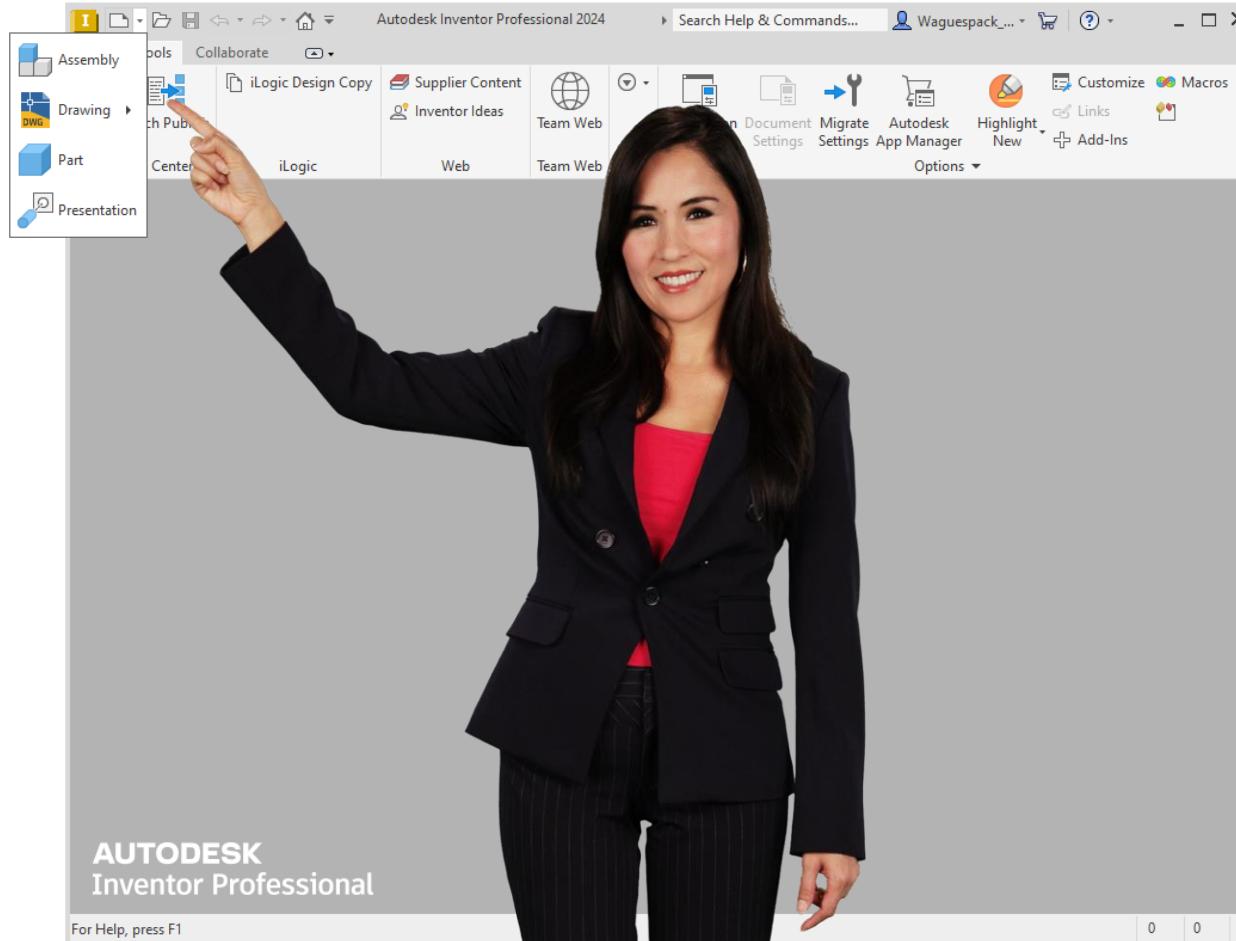


# Testing the Add-in



- 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 go to Tools > Add-ins

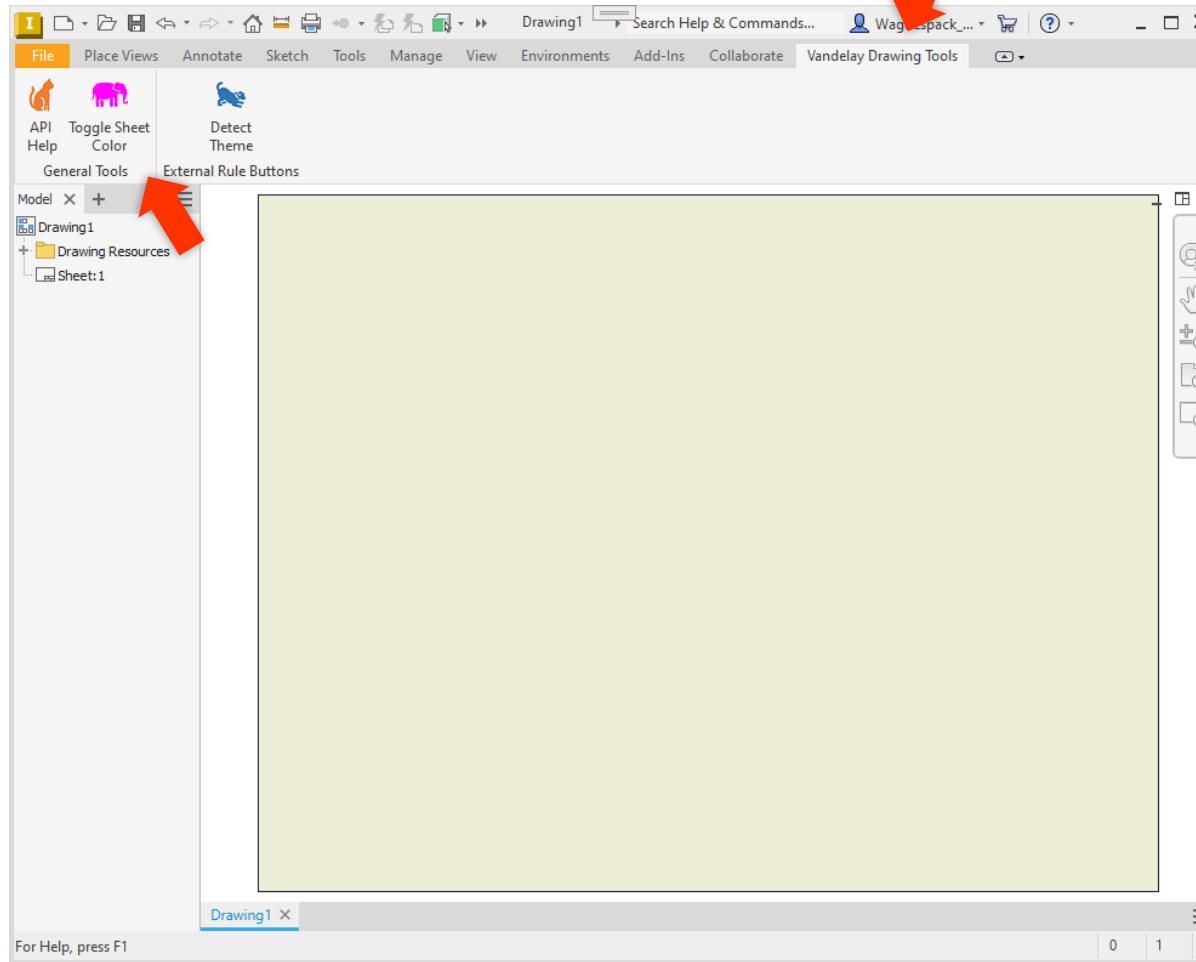
# Using the Add-in



- Create a new drawing file from a template

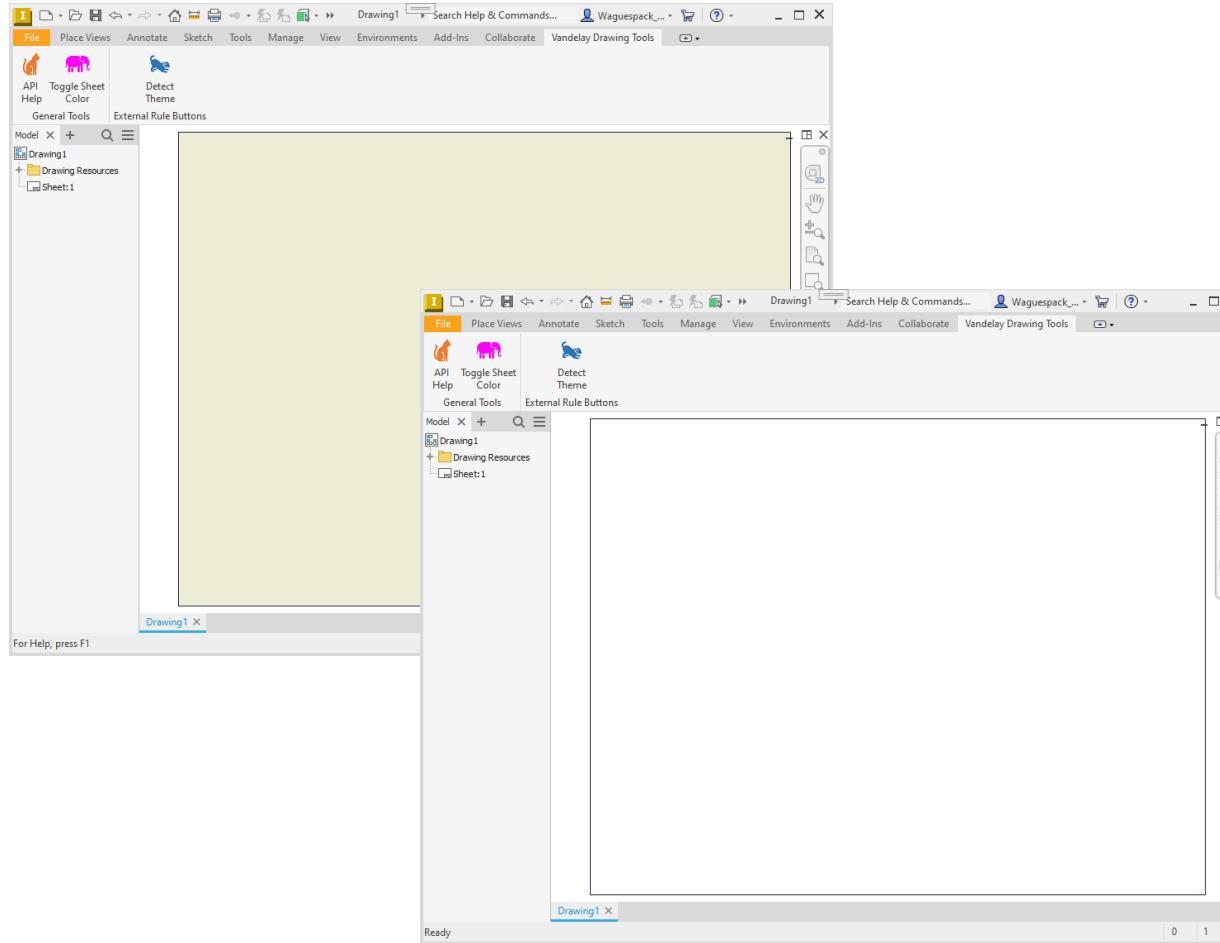
AUTODESK  
Inventor Professional

# Using the Add-in



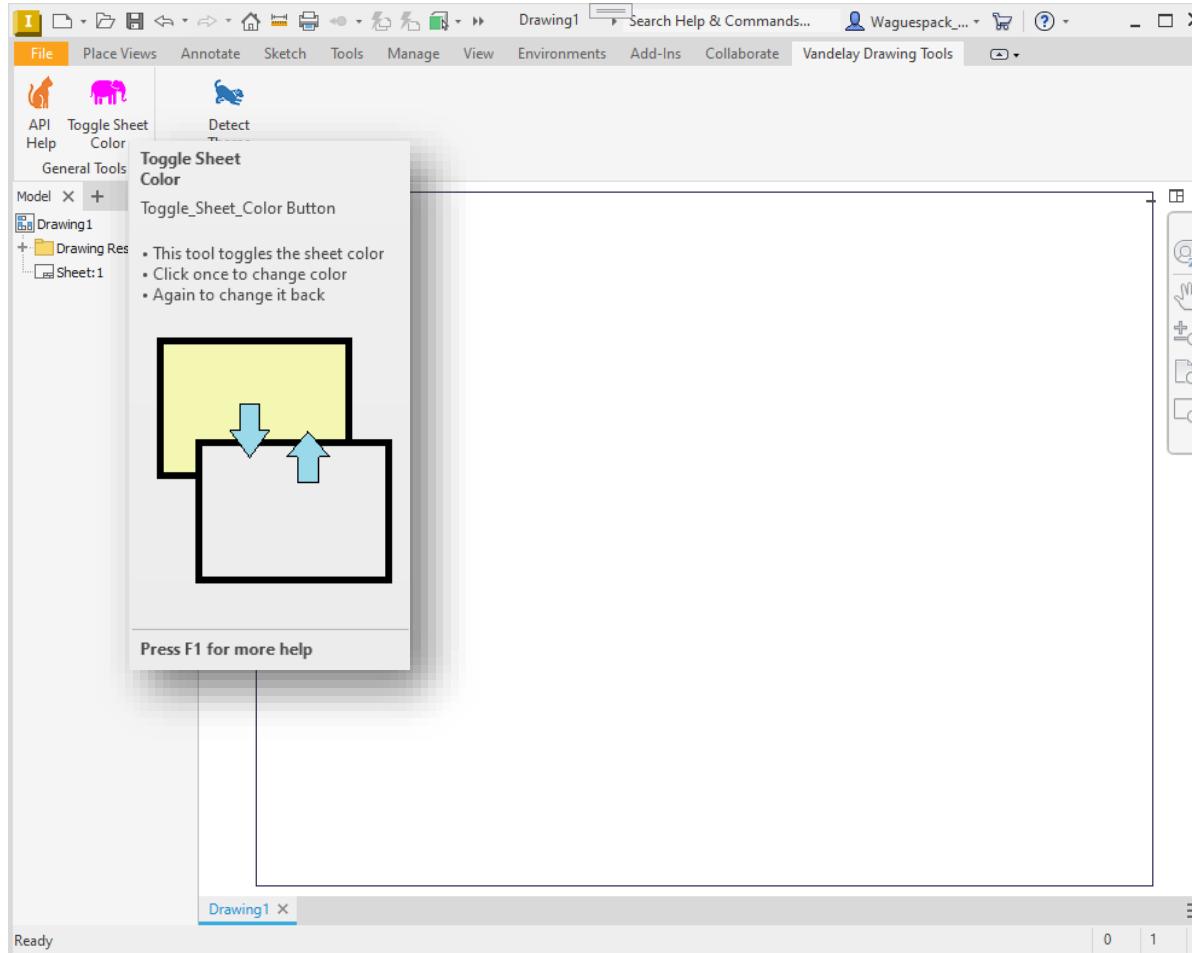
- Switch to the **Vandelay Drawing Tool** tab
- Note the new button on this tab

# 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

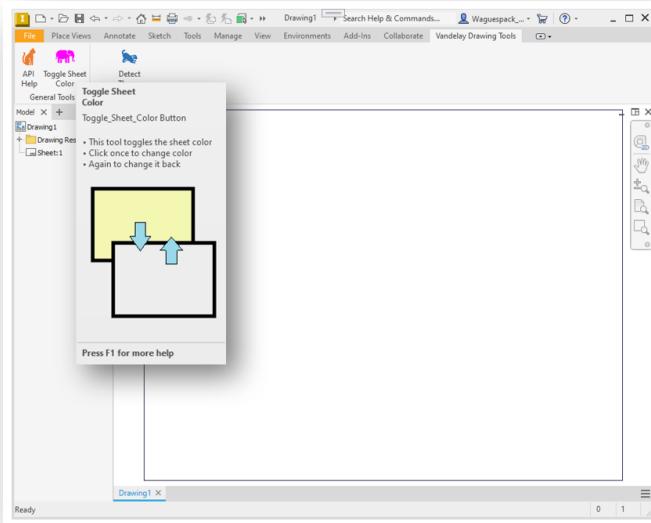
# Using the Add-in



- 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!

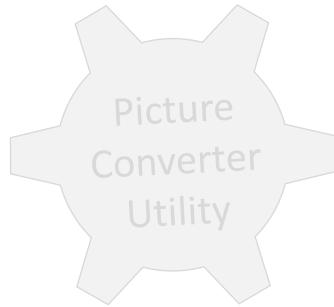
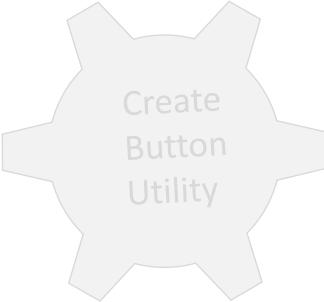


# Quick Review

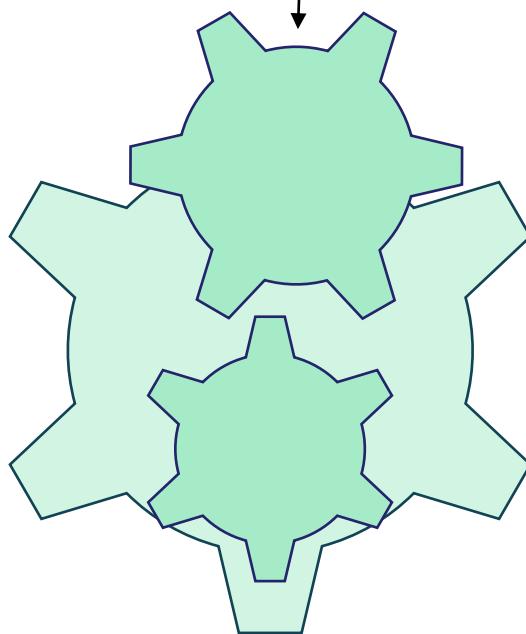
- Copy an existing Command Module to create a new one
- Open the module and change the module name
- In the CreateButton function:
  - Add button icons to the Resources
  - Set the button icons
  - Set the button label name
  - Set the simple tooltip
  - Set the progressive tooltip if desired
- In the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

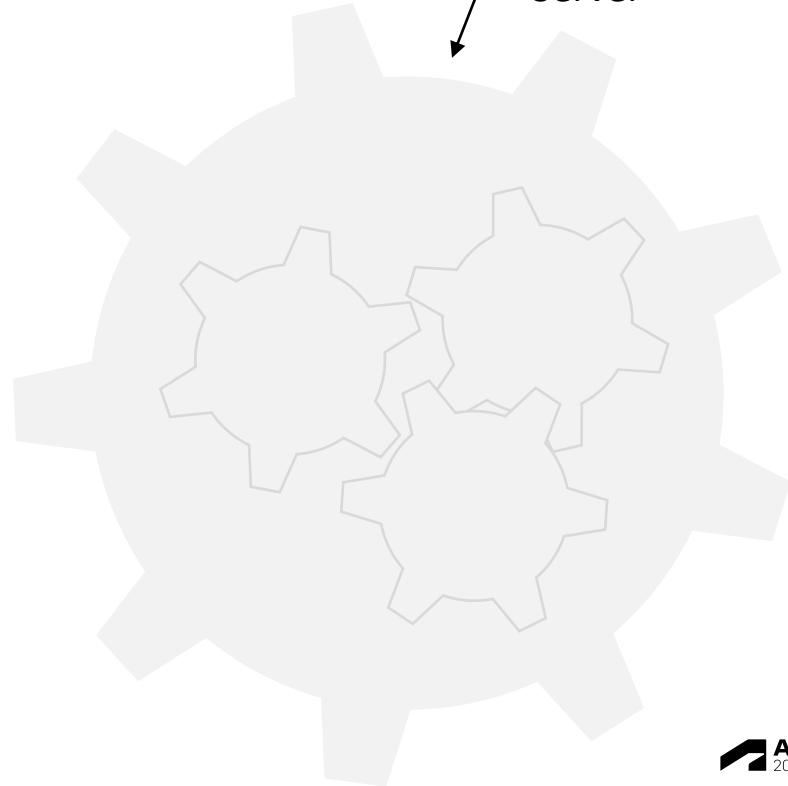
# Review



Copy/Paste to create a new button module



Standard Add-in Server



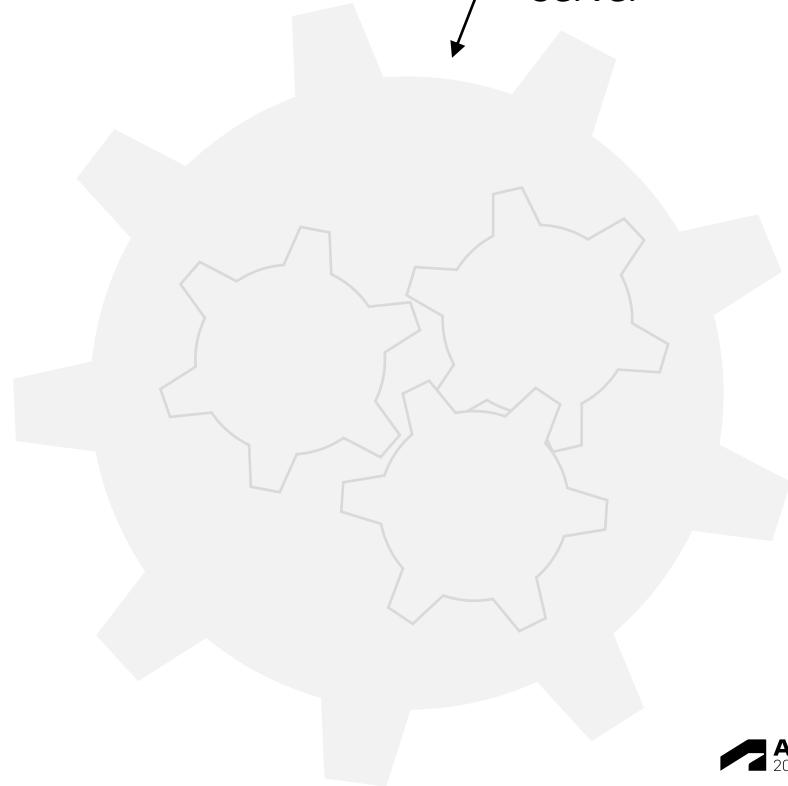
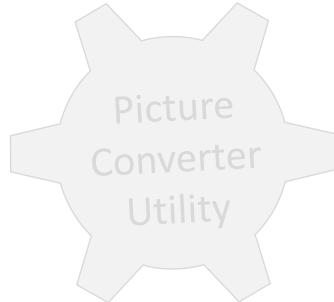
# Review

Copy/Paste to create a new button module

Button name  
Button icons  
Button tooltip

Create Button Function

Standard Add-in Server



# Review

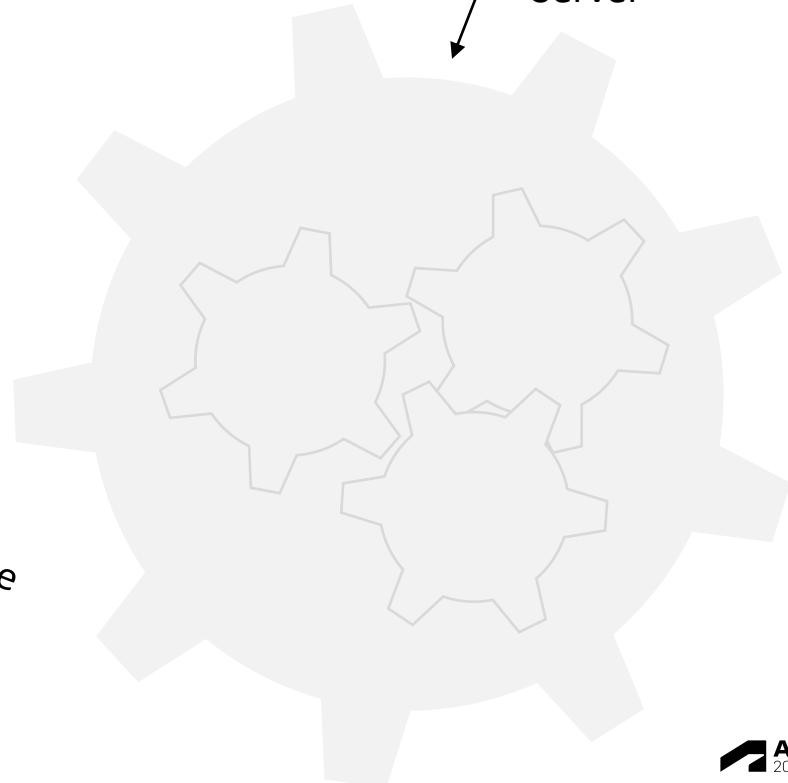
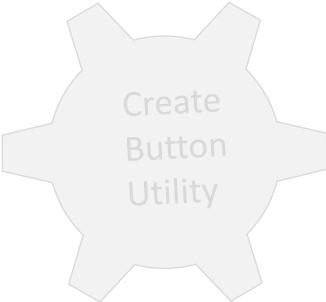
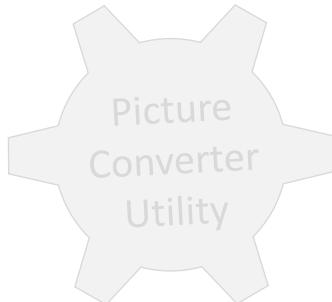
Copy/Paste to create a new button module

Create Button Function

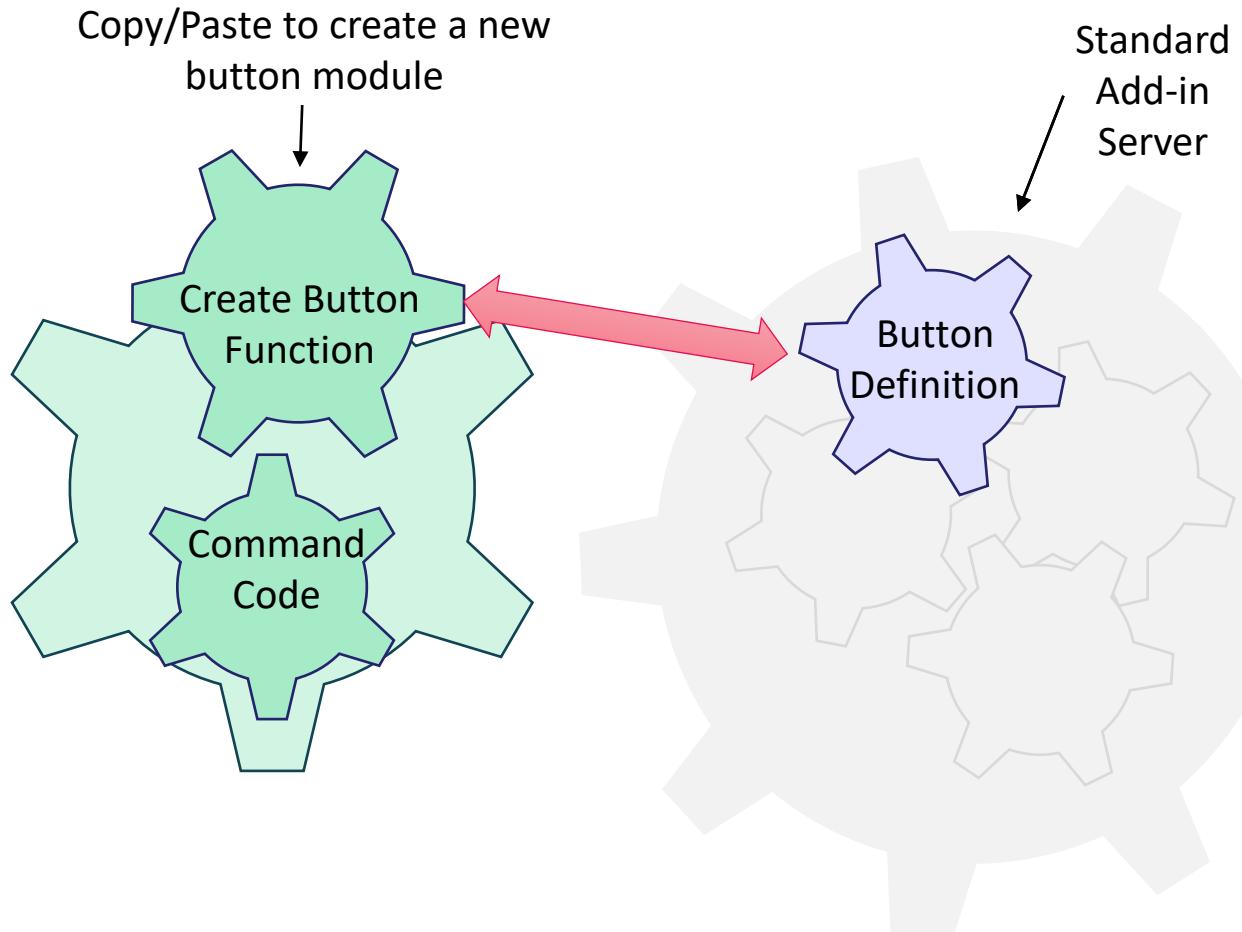
Command Code

Code to be  
executed  
when the  
button is  
clicked

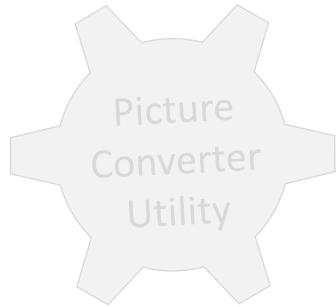
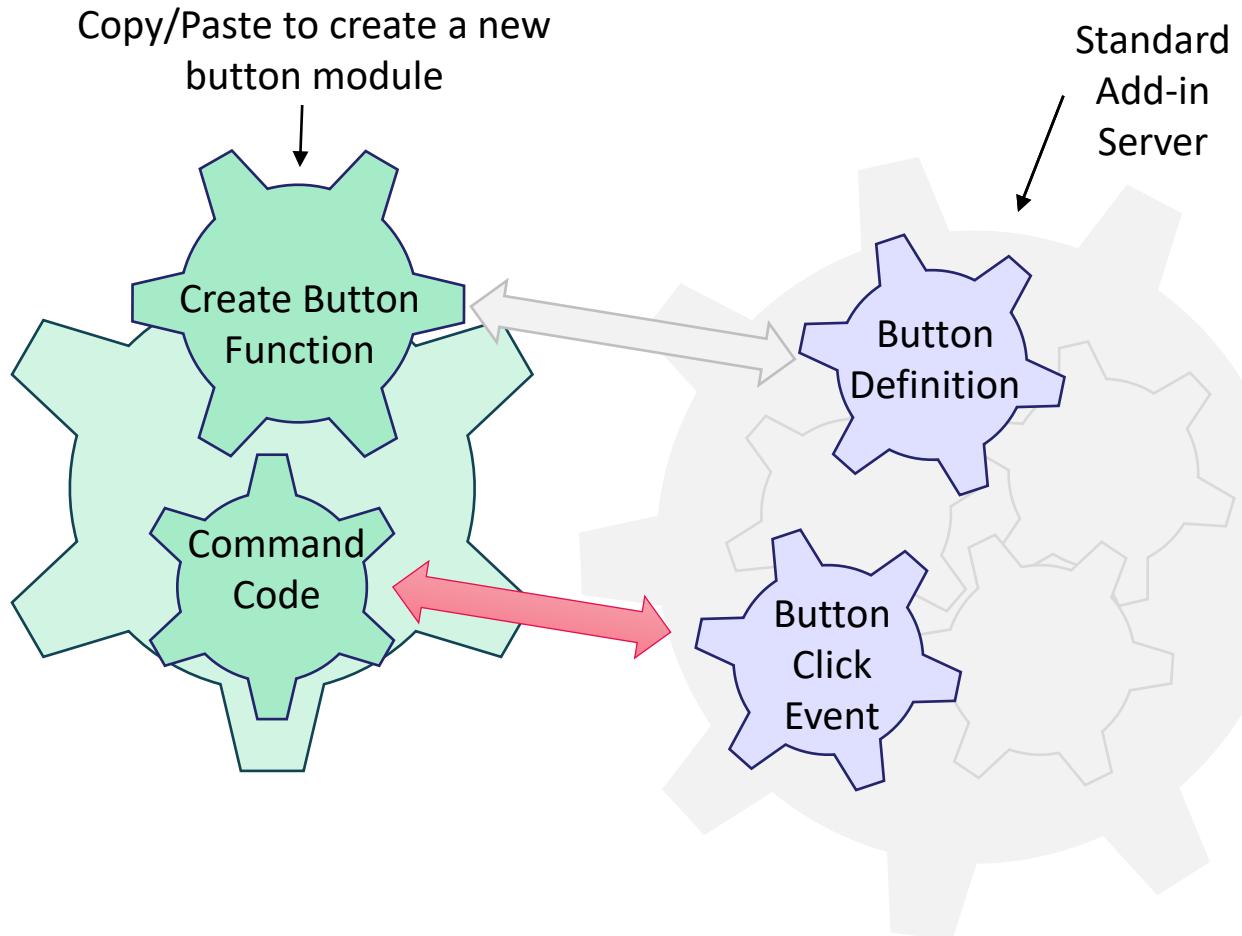
Standard  
Add-in  
Server



# Review

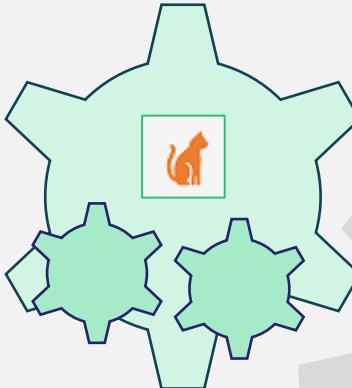


# Review

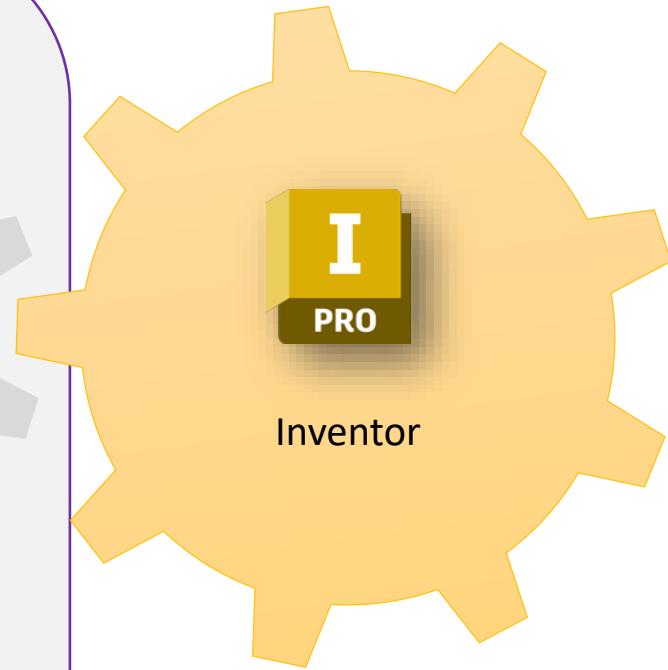


# Review

Utilities

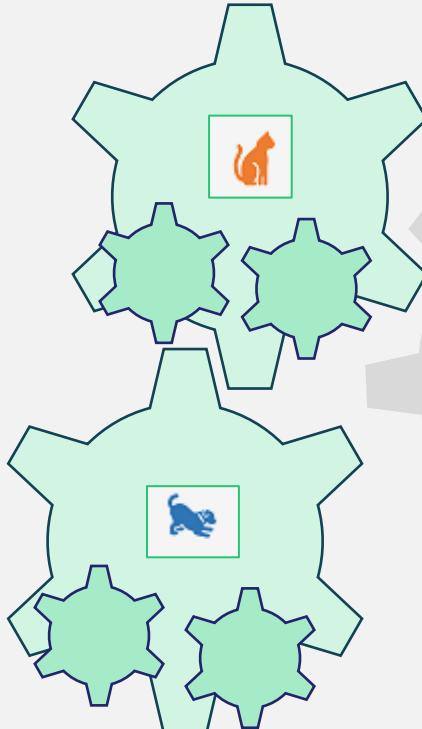


Standard Add-in  
Server

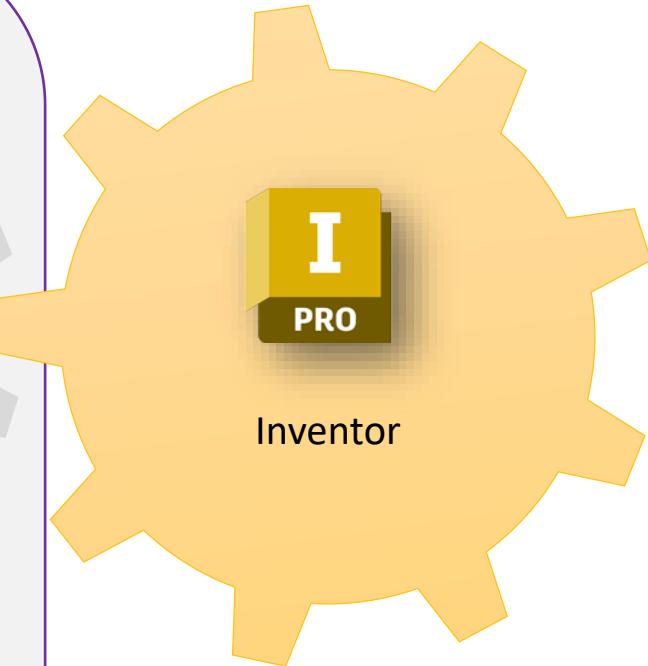
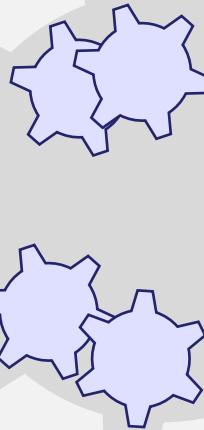


# Review

Utilities

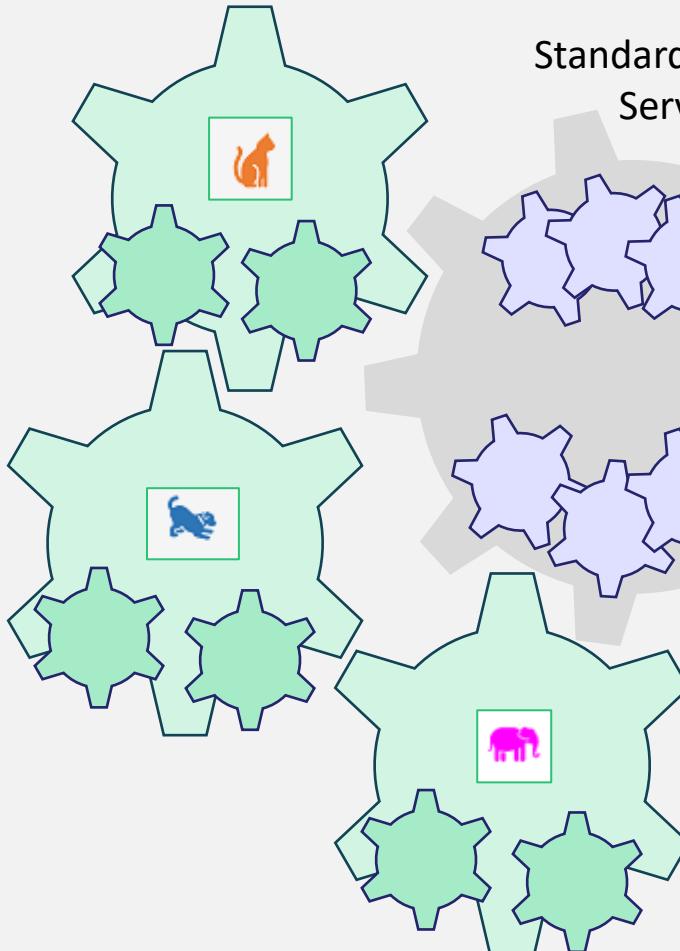


Standard Add-in  
Server

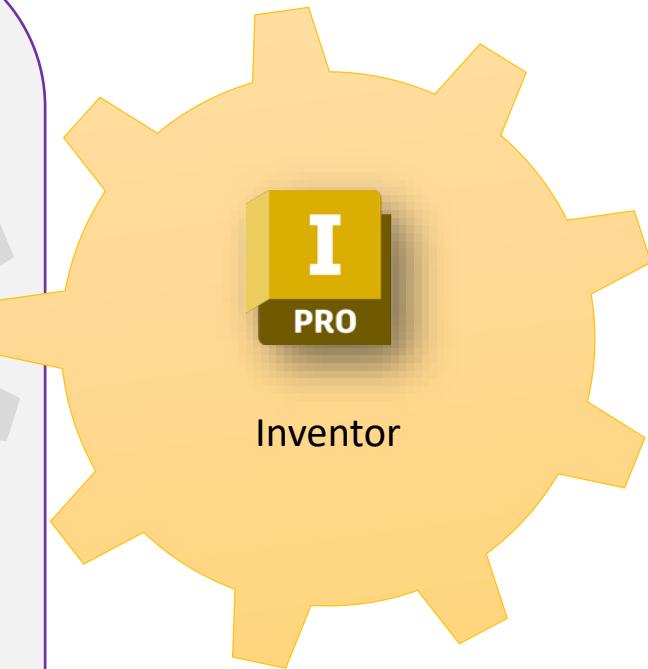


# Understanding the Add-in Template Structure

Utilities



Standard Add-in  
Server



Inventor