AUTODESK UNIVERSITY

ES123215

How I learnt to create AddIns for AutoCAD, Revit, Navisworks and Inventor in 3 months

Drew Jarvis
SolidCAD, a Cansel Company

Learning Objectives

- Create add-ins for multiple Autodesk Solutions
- Load up Dockable Windows in each product
- Connect to a SQL database and save data into a DataTable for later use
- Use the Event Handlers available in the different Autodesk Solutions

Description

The speaker started as a programmer with basic experience of Visual Basic for Applications (VBA) and LISP and self-learned how to create professional add-ins for multiple Autodesk, Inc. solutions in just a few months.
Now you will learn how and why this task was accomplished. This class will show you how to get started with each product's add-in with take away example code. You will see how the speaker created a suite of tools that are visually similar in each product and along the way learned many tips and tricks. Topics covered will include C#, PackageContents.xml files, add-in files, deployment bundles, Extensible Application Markup Language (XAML), Dockable Panels, SQL cloud connections, Windows Presentation Foundation (WPF) WebBrowsers, Event Handlers, and more.

Contents

AUTODESK UNIVERSITY

Drew Jarvis

18+ years experience with Autodesk Products, specializing in AutoCAD, Revit, Navisworks
Started with LISP and VBA in 2000
Moved into vb.Net in 2012
Tried to Create a Revit Addin in 2012… and failed
Tried again in 2013… and failed
Finally just jumped in with Addins for multiple products in 2014

Interface Options

In order to interact with an end user, you may need to create forms, dialog boxes or palettes.

I believe that the dockable palettes that first appeared in AutoCAD 2004 are the most modern way to interact.

Traditional Winforms can also be created, but I think they give applications a dated look.

Dockable palettes open up the ability to show content delivered with Windows Presentation Format which can be described with xaml or can be created on the fly with code.

The names of the dockable palettes differ between products, where AutoCAD names them PaletteSet's, Revit calls them DockablePane's. Navisworks and Inventor both go with DockableWindow's

Setting up Dockable Panes in Revit

Setting up a dockable pane in Revit is as simple as the code below

In OnStartUp

```
public static DockablePaneId dpid = null;
DockablePaneProviderData data = new DockablePaneProviderData();
DockableItem RevitDockableWindow = new DockableItem();
m_RevitDockableWindow = RevitDockableWindow;
data.FrameworkElement = RevitDockableWindow as System.Windows.FrameworkElement;
data.InitialState = new DockablePaneState();
data.InitialState.DockPosition = DockPosition.Tabbed;
data.InitialState.TabBehind = DockablePanes.BuiltInDockablePanes.PropertiesPalette;
dpid = new DockablePaneId(new Guid("{43456832-2780-42c9-88b1-905950c96757}"));
application.RegisterDockablePane(dpid, "AU2015", RevitDockableWindow as IDockablePaneProvider);
```

In addin assembly

```
DockablePane dp = commandData.Application.GetDockablePane(App.dpid);
dp.Show();
```

You then just need to add content to MainDockableWindow, note that DockablePane is a xaml userform in order to support beautiful modern interfaces ☺

AUTODESK UNIVERSITY

PaletteSets in AutoCAD

Setting up a PaletteSet in AutoCAD is as simple as the code below

```
using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.Windows;
using System;
namespace AU2017_AutoCAD
{
    public class App
    {
        public static DockablePalette AutoCADDockableWindow = new DockablePalette();
        public static PaletteSet ps = null;
        [CommandMethod("AU2017GROUP", "AU2017", CommandFlags.NoActionRecording)]
        public void AU2017()
        {
            if (ps == null)
            {
                ps = new PaletteSet("Autodesk University 2017", new System.Guid("ad2f9de8-a4ef-4c06-8bbb-5c17f5feaee8"));
                ps.Load += Ps_Load;
                ps.Save += Ps_Save;
                ps.Size = new System.Drawing.Size(400, 600);
                ps.DockEnabled = (DockSides)((int)DockSides.Left + (int)DockSides.Right);
                ps.Style = PaletteSetStyles.ShowCloseButton;
                ps.Style = PaletteSetStyles.ShowAutoHideButton;
                ps.AddVisual("AddVisual", AutoCADDockableWindow);
                ps.KeepFocus = true;
                ps.Visible = true;
            }
            ps.Visible = true;
        }
        private static void Ps_Save(object sender, PalettePersistEventArgs e)
        {
            string position = Convert.ToString(e.ConfigurationSection.ReadProperty("Position", "Default"));
            if (position == "Floating")
            {
                double x = Convert.ToDouble(e.ConfigurationSection.ReadProperty("Left", 22.3));
                double y = Convert.ToDouble(e.ConfigurationSection.ReadProperty("Top", 22.3));
                double width = Convert.ToDouble(e.ConfigurationSection.ReadProperty("Width", 22.3));
                double height = Convert.ToDouble(e.ConfigurationSection.ReadProperty("Height", 22.3));
                ps.InitializeFloatingPosition(new System.Windows.Rect(x, y, width, height));
            }
            else if (position == "Left")
            {
                ps.Dock = Autodesk.AutoCAD.Windows.DockSides.Left;
            }
            else if (position == "Right")
            {
                ps.Dock = Autodesk.AutoCAD.Windows.DockSides.Right;
            }
            else
            {
                ps.InitializeFloatingPosition(new System.Windows.Rect(100, 100, 100, 200));
            }
        }
        private static void Ps_Load(object sender, PalettePersistEventArgs e)
        {
            ps.Style = PaletteSetStyles.ShowCloseButton;
            if (ps.Visible == false)
            {
                e.ConfigurationSection.WriteProperty("Position", "Hidden");
            }
            else if (ps.Dock == Autodesk.AutoCAD.Windows.DockSides.None)
            {
                e.ConfigurationSection.WriteProperty("Position", "Floating");
                e.ConfigurationSection.WriteProperty("Left", ps.Location.X);
                e.ConfigurationSection.WriteProperty("Top", ps.Location.Y);
                e.ConfigurationSection.WriteProperty("Width", ps.Size.Width);
                e.ConfigurationSection.WriteProperty("Height", ps.Size.Height);
            }
            else if (ps.Dock == Autodesk.AutoCAD.Windows.DockSides.Left)
            {
                e.ConfigurationSection.WriteProperty("Position", "Left");
            }
            else if (ps.Dock == Autodesk.AutoCAD.Windows.DockSides.Right)
            {
                e.ConfigurationSection.WriteProperty("Position", "Right");
            }
            else
            {
                e.ConfigurationSection.WriteProperty("Position", "Default");
            }
        }
    }
}
```

You then just need to add content to a UserControl (WPF), note that DockablePalette is a xaml userform in order to support beautiful modern interfaces ☺

DockableWindows in Inventor

Setting up a DockableWindow in Inventor is as simple as the code below
DockableWindow docableWin;

```
uiMan = app.UserInterfaceManager;
docableWin = uiMan.DockableWindows.Add(Guid.NewGuid().ToString(), "INAW_UIMiscs_DockableWindow1",
“AU2015");
docableWin.AddChild(CreateChildDialog());
docableWin.DisabledDockingStates = DockingStateEnum.kDockLeft | DockingStateEnum.kDockTop;
docableWin.DockingState = DockingStateEnum.kDockRight;
docableWin.ShowVisibilityCheckBox = true;
docableWin.ShowTitleBar = true;
docableWin.SetMinimumSize(100, 100);
docableWin.Visible = true;
public static long CreateChildDialog()
{
System.Windows.Forms.Integration.ElementHost host = new
System.Windows.Forms.Integration.ElementHost();
host.Dock = DockStyle.Fill;
ButtonCtrl bc = new ButtonCtrl();
host.Child = bc;
bc.Controls.Add(host);
bc.FormBorderStyle = FormBorderStyle.None;
bc.HelpButton = bc.MinimizeBox = bc.MaximizeBox = false;
bc.ShowIcon = bc.ShowInTaskbar = false;
bc.TopMost = true;
bc.Height = 100;
bc.Width = 300;
bc.MinimumSize = new System.Drawing.Size(bc.Width, bc.Height);
bc.Show();
return dc.Handle.ToInt64();
}
```

You then just need to add content to bc, note that ButtonCtrl is a xaml userform in order to support beautiful modern interfaces ☺

PaletteSets in Navisworks

Setting up a DockPane in Navisworks is so easy as it is ‘baked’ into the Plugin Type

```
[Plugin(“AU2015PlugIn", “AU2015", DisplayName = “AU2015 DockPane", ToolTip = “Show a DockPane in
Navisworks")]
[DockPanePlugin(150, 200, FixedSize = false)]
ElementHost eh = new ElementHost();
DockablePane dp = new DockablePane();
eh.Child = dp;
PluginRecord pr =
Autodesk.Navisworks.Api.Application.Plugins.FindPlugin("NavisworksAU2015.PlugIn.NNAW");
DockPanePlugin dpp = pr.LoadedPlugin as DockPanePlugin;
if(dpp != null)
{
dpp.Visible = !dpp.Visible;
}
```

You then just need to add content to dp, note that DockablePane is a xaml userform in order to support beautiful modern interfaces ☺

Good Reference:
http://spiderinnet.typepad.com/blog/2013/11/navisworks-net-use-dockpaneplugindockpanepluginattribute-to-create-dock-panel.html

Event Handling

All of the software platforms implement event handlers in different ways, some require handlers for each command, some have a general event handler for all commands, and some don't really want to help you catch events at all ☺
AutoCAD has a general catch all, Revit requires specific event handlers for each command while Navisworks and Inventor don't seem to have Event Handlers for the internal commands, so I had to do something else with those products.

Handling Events in AutoCAD

Handling the firing of a command in AutoCAD is super easy

```
DocumentManager.MdiActiveDocument.CommandWillStart += new
CommandEventHandler(CommandBeginHandler);
public void CommandBeginHandler(object sender, CommandEventArgs e)
{
e.GlobalCommandName //This is the command name, use this to direct your addin
```

AutoCAD has a handler to capture the start of any command whether it starts from a Ribbon, the command line, a tool palette or a toolbar button (I guess even a Menu/Tile/other Legacy stuff… man there are a lot of ways to do the same thing in AutoCAD ☺)
So capture the command, check if you are interested in it, and if you are then you can do something to it.

Handling Events in Revit

Handling the firing of a command in Revit is easy, but you need to know the CommandID, for example ID_WINDOW_CLOSE_HIDDEN

```
AddInCommandBinding importBindingID_WINDOW_CLOSE_HIDDEN =
app.CreateAddInCommandBinding(RevitCommandId.LookupCommandId("ID_WINDOW_CLOSE_HIDDEN"));
importBindingID_WINDOW_CLOSE_HIDDEN.BeforeExecuted += new
EventHandler<Autodesk.Revit.UI.Events.BeforeExecutedEventArgs>((sender, arg) =>
DoSomething(sender, arg, "ID_WINDOW_CLOSE_HIDDEN"));
public void DoSomething(object sender, EventArgs e, String RevitInternalName)
{
RevitInternalName //This is the command name, use this to direct your addin
```

Excellent link on TheBuildingCoder blog that has all of the internal command names:
http://thebuildingcoder.typepad.com/blog/2012/06/replacing-built-in-commands-and-their-ids.html
http://thebuildingcoder.typepad.com/files/commandids.xlsx

Handling Events in Inventor/Navisworks

Handling the firing of a command in Inventor or Navisworks requires monitoring the Ribbon for which buttons get clicked, there are no specific command handlers unfortunately (That I know of)

```
Autodesk.Windows.ComponentManager.ItemExecuted += new
EventHandler<Autodesk.Internal.Windows.RibbonItemExecutedEventArgs>(ItemExecutedTest);
void ItemExecutedTest(object sender, Autodesk.Internal.Windows.RibbonItemExecutedEventArgs e)
{
    if (e.Item.Text != null)
    {
        string CommandNameValue = e.Item.Text.ToString(); //This is the command name
    }
}
```

Connecting to Online SQL Data

SQL Cloud connections

Connecting to a cloud database was a great way to provide up to date information to the client.
I selected Microsoft Azure as my base, AWS or similar would be just as good.

DataTable for local data storage

Initially I queried the database everytime I needed to get the data from it, however this was inefficient.
I realized that a local DataTable that is populated once per session would suffice, the downside was a lack of guaranteed “updatedness” – but I could live with this.

```
using System.Data;
using System.Data.SqlClient;
 //SQL Connection
DataTable dtAllColors = new DataTable();
dtAllColors.Clear();
SqlConnection connResources = new SqlConnection("ConnectionString");
SqlCommand cmdResources = new SqlCommand("SELECT DISTINCT Something FROM Somethingelse",
connResources);
connResources.Open();
dtAllColors.Load(cmdResources.ExecuteReader(CommandBehavior.CloseConnection));
connResources.Dispose();
```

Entitlement

Entitlement deals with ensuring that people who use your addins are allowed to use them, for example you may create an addin and want to charge people a fee to use the addin, so you need a way to check that people are entitled to use it.
Luckily you don’t need to create this framework, Autodesk has given you a service you can make use of if you sell you addin via the app store.

Entitlement API

Autodesk has provided an easy to use Entitlement Check that will talk to the exchange app web server and determine if a user is entitled to use the app.
The user will need to be logged in with their Autodesk ID account that they used to download the app. As all software is sold as a subscription product that requires you to have an Autodesk ID this is a very user friendly process.
The entitlement check will also work for subscription based AddIns where it will check that the user has an active subscription for the AddIn, ie the user has paid for the service that month.

```
public const string _AppId = @"< appstore.exchange.autodesk.com:*************>";
public static Result CheckForEntitlement(ExternalCommandData commandData)
{
    UIApplication uiApp = commandData.Application;
    Application rvtApp = uiApp.Application;
    if (!Application.IsLoggedIn)
    {
        TaskDialog.Show("Entitlement Check", "Please login to Autodesk 360 first\n");
        return Result.Failed;
    }
        string userId = rvtApp.LoginUserId;
        bool isValid = false;
    try
    {
        isValid = CheckEntitlement(_AppId, userId);
    }
    catch (Exception ex)
    {
        string message = ex.Message;
    }
    if (isValid)
    {
        return Result.Succeeded;
    }
    TaskDialog.Show("Entitlement Check", "You are not entitled to use this app,\nplease contact
*****@*****.com \n");
    return Result.Failed;
}
```

Autoloader

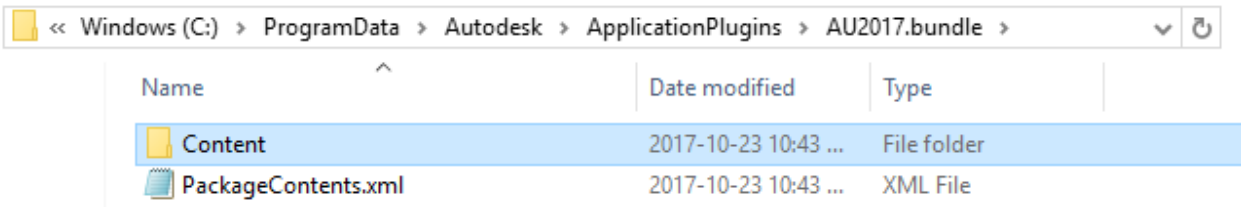
The Autoloader is designed to simplify 90% of all Autodesk application deployments. It allows you to deploy your plugins as a simple package format. Specifies a few select locations that you can place your files on your computer that the Autodesk product will check on startup.

Simplified the installation process for the end user and the developer.

File are placed in one of the following locations on the machine:

%PROGRAMDATA%\Autodesk\ApplicationPlugins\
%APPDATA%\Autodesk\ApplicationPlugins\
C:\Program Files\Autodesk\ApplicationPlugins\

A folder with a suffix of .bundle is placed into the location above and an xml file that describes the files to be loaded for the Addin is placed into the root of the .bundle folder, for example:



Great Whitepaper Here:

http://adndevblog.typepad.com/autocad/2013/01/autodesk-autoloader-white-paper.html

Simple Package Contents examples can be seen in the example code for each product at the end of this document.

Creating Addins, Step by Step Guides

The following pages will give detailed instructions on how to create an AddIn.
If you can get the welcome message to show up then from there you can do anything available in the API, these guides are a simple 'foolproof' guide to getting started, if you complete them you will have an addin that successfully loads when you startup your software.

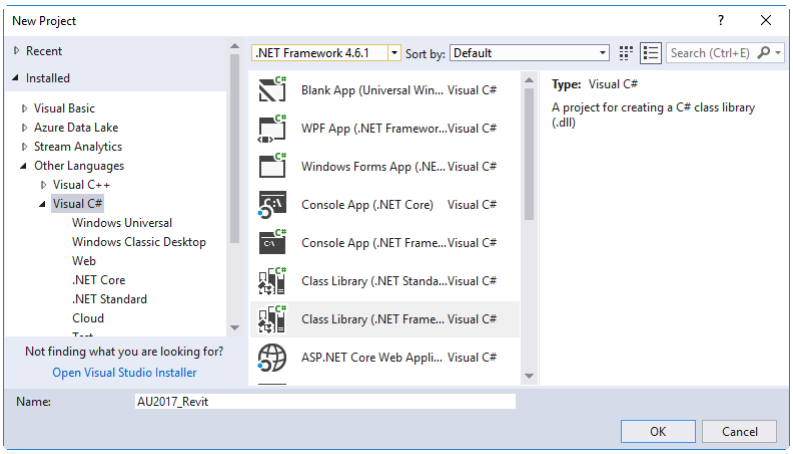
The completed Projects are available for download with this presentation, looks for:

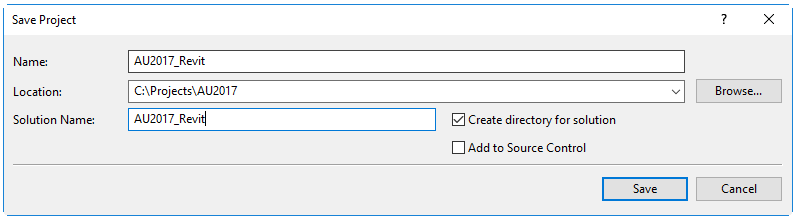
AU2017_Revit. zip
AU2017_AutoCAD.zip
AU2017_Navisworks.zip
AU2017_Inventor.zip

Revit

There is a Video here showing the process:
https://1drv.ms/v/s!AnQuwhbiBgLhg_8e2I4SWom48cw7Fg

Create a New Class Library Project





Right click on the Project in the Solution Explorer and select Add then Reference to add the following references:

C:\Program Files\Autodesk\Revit 2018\RevitAPI.dll
C:\Program Files\Autodesk\Revit 2018\RevitAPIUI.dll
Assemblies -> System.Windows.Forms

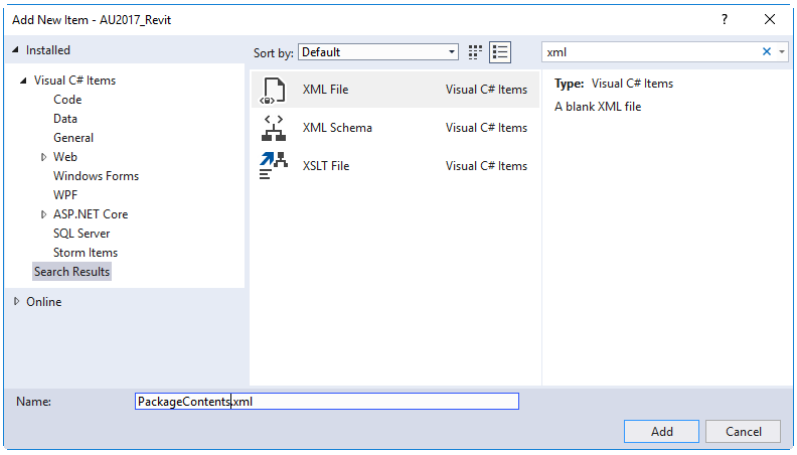
Set the Copy Local property to False for each added Reference.

Right click on Class1.cs in the Solution Explorer and rename Class1.cs to App.cs

Put the following code in a class named App.cs:

```
using System;
using Autodesk.Revit.UI;
using System.Windows.Forms;
namespace AU2017_Revit
{
    class App : IExternalApplication
    {
        public Result OnShutdown(UIControlledApplication application)
        {
            return Result.Succeeded;
        }
        public Result OnStartup(UIControlledApplication application)
        {
            MessageBox.Show("Hello Autodesk University 2017");
            return Result.Succeeded;
        }
    }
}
```

Right Click on the Project in the Solution Explorer and select Add then New Item… or press CRTL + SHIFT + A then select XML file and call the new xml file PackageContents.xml



Replace the contents of PackageContents.xml with:

```
<?xml version="1.0" encoding="utf-8" ?>
<ApplicationPackage SchemaVersion="1.0" AppVersion="15.12.02" ProductCode="b57c480f-24f5-4fbf-
804c-7df86c770305">
  <RuntimeRequirements OS="Win64" Platform="Revit" SeriesMin="R2018" SeriesMax="R2018" />
  <Components Description="2018">
    <RuntimeRequirements OS="Win64" Platform="Revit" SeriesMin="R2018" SeriesMax="R2018" />
    <ComponentEntry AppName="AU2017_Revit"
ModuleName="./Contents/2018/AU2017_Revit.addin"></ComponentEntry>
  </Components>
</ApplicationPackage>
```

Add another new XML file to the project named AU2017_Revit.addin
Replace the contents of the above file with:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<RevitAddIns>
  <AddIn Type="Application">
    <Name>AU2017</Name>

<Assembly>C:\ProgramData\Autodesk\ApplicationPlugins\AU2017.bundle\Contents\2018\AU2017_Revit.dll<
/Assembly>
    <AddInId>a694f098-20d5-42c2-a8a0-ac7ae6857c24</AddInId>
    <FullClassName>AU2017_Revit.App</FullClassName>
    <ClientId>0c37a558-6df2-44b7-b367-70b469a399a7</ClientId>
    <VendorId>272c22da-13ee-40ad-9b9f-d014a7fda46a</VendorId>
    <VendorDescription>CompanyName, www.CompanyName.com</VendorDescription>
  </AddIn>
</RevitAddIns>
```

Double click on Properties in the Solution Exporer and go to the Debug Tab.
Set the Debug Start Action to Start external program and pick C:\Program Files\Autodesk\Revit 2018\Revit.exe

Go to the Build Events Tab of the Properties -> Post Build Event Command Line to:

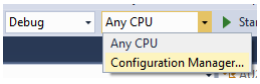
```
xcopy "$(TargetDir)AU2017_Revit.dll"
"C:\ProgramData\Autodesk\ApplicationPlugins\AU2017.bundle\Contents\2018\" /i /e /y /c
xcopy "$(SolutionDir)AU2017_Revit\PackageContents.xml"
"C:\ProgramData\Autodesk\ApplicationPlugins\AU2017.bundle\" /i /e /y /c
xcopy "$(SolutionDir)AU2017_Revit\AU2017_Revit.addin"
"C:\ProgramData\Autodesk\ApplicationPlugins\AU2017.bundle\Contents\2018\" /i /e /y /c
```

Using Notepad, navigate to your project and Open the file AU2017_Revit.csproj & add the 'DebugEngines' property within 'PropertyGroup' like the below

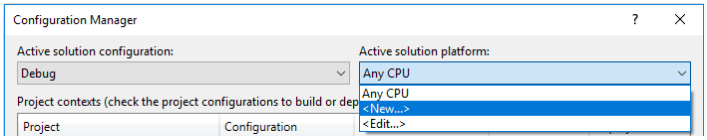
```
    <DebugEngines>{351668CC-8477-4fbf-BFE3-5F1006E4DB1F}</DebugEngines>
```

Close and Save the file RevitAU2015.csproj

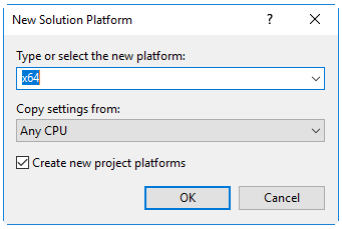
Start the Configuration Manager



Select New from the Any CPU Dropdown



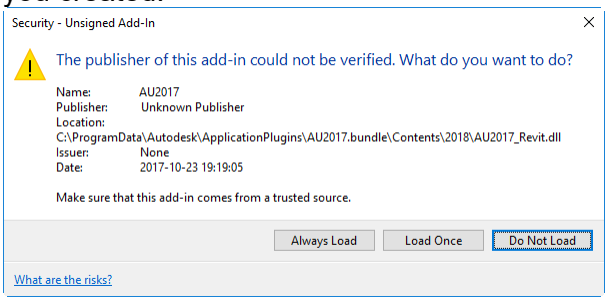
Verify x64 is the new platform copying settings from Any CPU and click OK



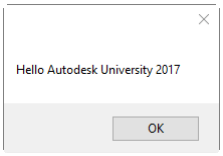
Close the Configuration Manager

Right click on the Project in the Solution Explorer and select Build

Start the Debugging, You should get 2 dialog boxes the first asking you to OK loading the addin you created:



Click Always Load on this one, then a second giving you a nice welcome message in Revit.



You now have a working Revit Addin.

AutoCAD

There is a Video here showing this process:
https://1drv.ms/v/s!AnQuwhbiBgLhg_8cOfInrT4kMmnT8g

Create a New Class Library Project Named AU2017_AutoCAD

Right click on the Project in the Solution Explorer and select Add then Reference to add the following references:

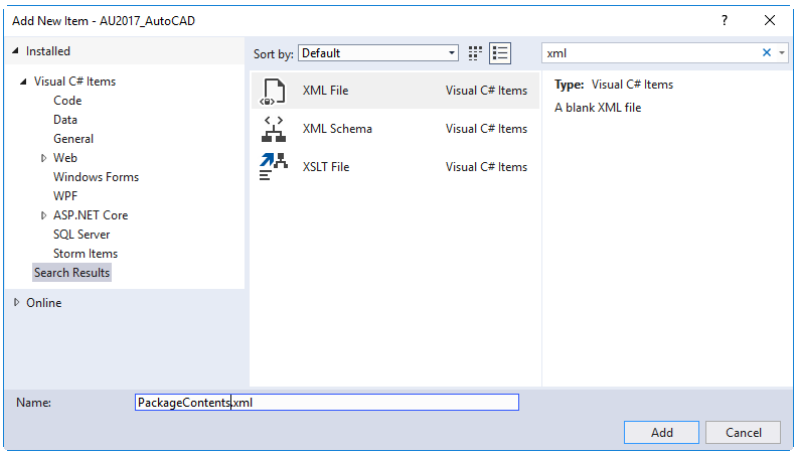
C:\Program Files\Autodesk\AutoCAD 2018\AcDbMgd.dll
C:\Program Files\Autodesk\AutoCAD 2018\AcMgd.dll
C:\Program Files\Autodesk\AutoCAD 2018\AcCoreMgd.dll
Assemblies -> System.Windows.Forms

Right click on Class1.cs in the Solution Explorer and rename Class1.cs to App.cs

Put the following code in a class named App:

```
using Autodesk.AutoCAD.Runtime;
using System.Windows.Forms;
namespace AU2017_AutoCAD
{
    public class App
    {
        [CommandMethod("AU2017GROUP", "AU2017", CommandFlags.NoActionRecording)]
        public void AU2017()
        {
            MessageBox.Show("Hello Autodesk University 2017");
        }
    }
}
```

Add a new XML file to the project named PackageContents.xml

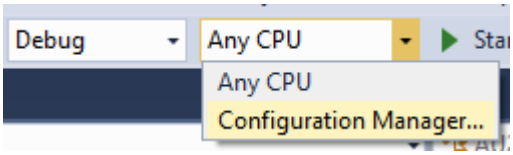


Replace the contents of the above file with:

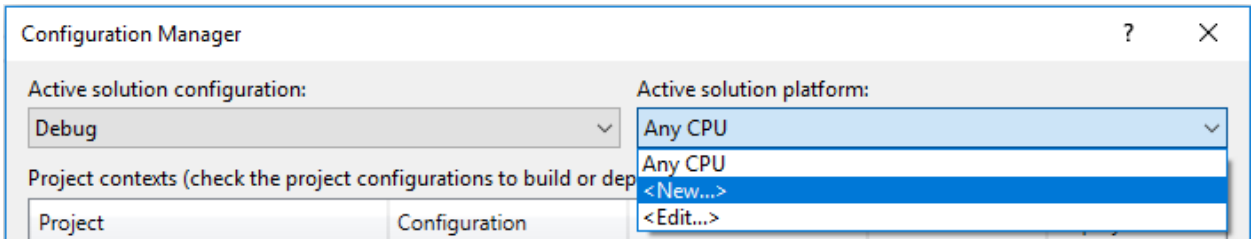
```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
```

```
<ApplicationPackage SchemaVersion="1.0" AppVersion="17.11.19" AutodeskProduct="AutoCAD"
Description="" Author="abc" HelpFile="./Contents/Help.htm" ProductCode="{d26e7bec-9960-4689-a699-
22aacfc6dbb6}" UpgradeCode="{b3328766-c9b2-43d1-abaf-b9a65e39a2f2}" Name="">
  <RuntimeRequirements OS="Win64" Platform="AutoCAD" SeriesMin="22.0" SeriesMax="22.0" />
  <Components Description="2018_64">
    <RuntimeRequirements OS="Win64" Platform="AutoCAD" SeriesMin="22.0" SeriesMax="22.0" />
    <ComponentEntry AppName="AU2017" Version="17.11.19"
ModuleName="./Contents/2018/AU2017_AutoCAD.dll" AppDescription="AU2017 Add-In"
LoadOnAutoCADStartup="True">
      <Commands GroupName="AU2017GROUP">
        <Command Local="AU2017" Global="AU2017" StartupCommand="True" ></Command>
      </Commands>
    </ComponentEntry>
  </Components>
</ApplicationPackage>
```

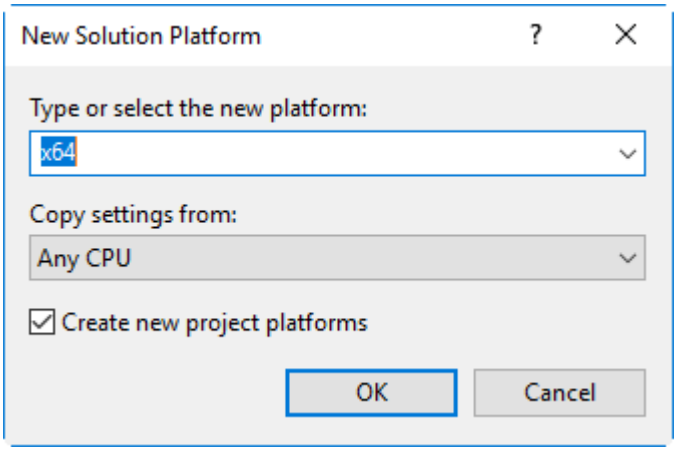
Start the Configuration Manager



Select New from the Any CPU Dropdown

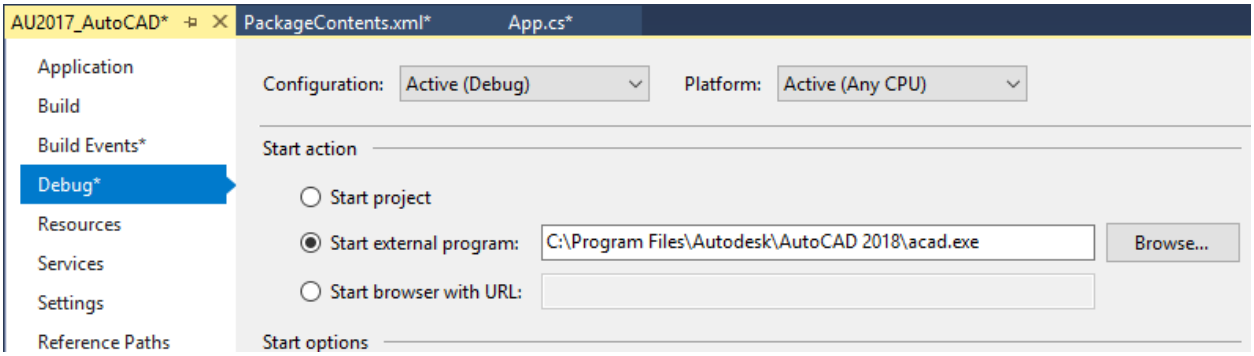


Verify x64 is the new platform copying settings from Any CPU and click OK



Close the Configuration Manager

In the Properties Set the Debug Start Action to Start external program and pick C:\Program Files\Autodesk\AutoCAD 2018\acad.exe

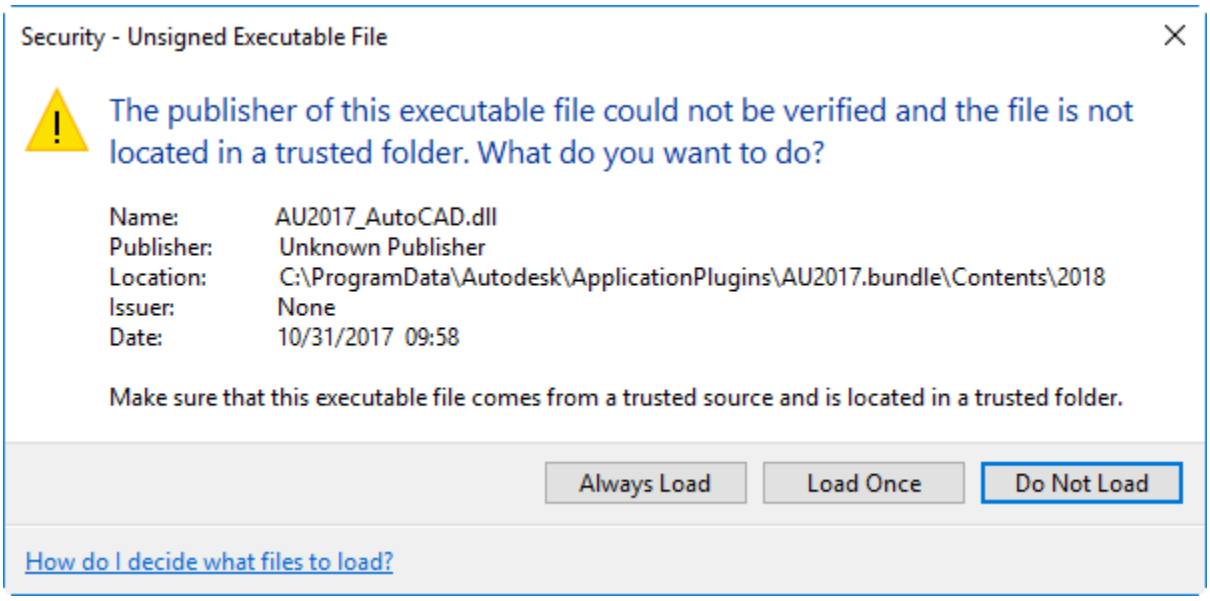


In the Properties Set the Build Events -> Post Build Event Command Line to:

```
xcopy "$(TargetPath)"
"C:\ProgramData\Autodesk\ApplicationPlugins\AU2017.bundle\Contents\2018\" /i /e /y /c
xcopy "$(SolutionDir)AU2017_AutoCAD\PackageContents.xml"
"C:\ProgramData\Autodesk\ApplicationPlugins\AU2017.bundle\" /i /e /y /c
```

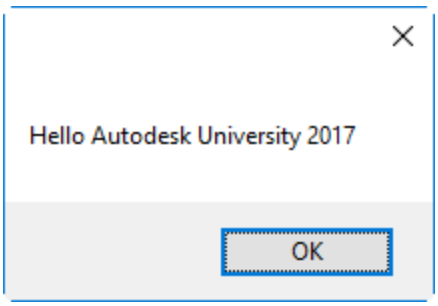
Right click on the Project in the Solution Explorer and select Build

Start the Debugging, You should get 2 dialog boxes the first asking you to OK loading the addin you created:



Click Always Load or Load Once

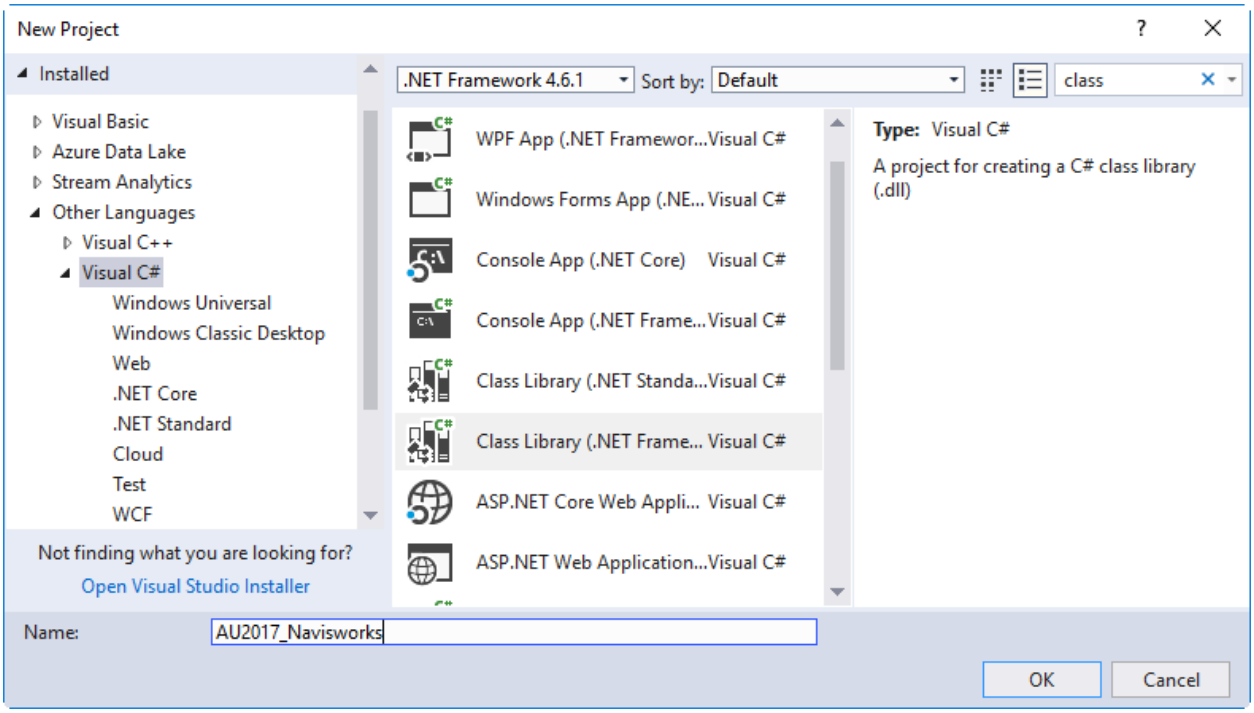
Followed by a welcome message:


Hello Autodesk University 2017
OK

Navisworks

There is a Video here showing the process:
https://1drv.ms/v/s!AnQuwhbiBgLhg_8dVKIERBglt4Tl0w

Create a New Class Library Project



Right click on the Project in the Solution Explorer and select Add then Reference to add the following references:

C:\Program Files\Autodesk\Navisworks Simulate 2018\Autodesk.Navisworks.Api.dll
OR
C:\Program Files\Autodesk\Navisworks Manage 2018\Autodesk.Navisworks.Api.dll
OR
C:\Program Files\Autodesk\Navisworks Freedom 2018\Autodesk.Navisworks.Api.dll

And Assemblies -> System.Windows.Forms

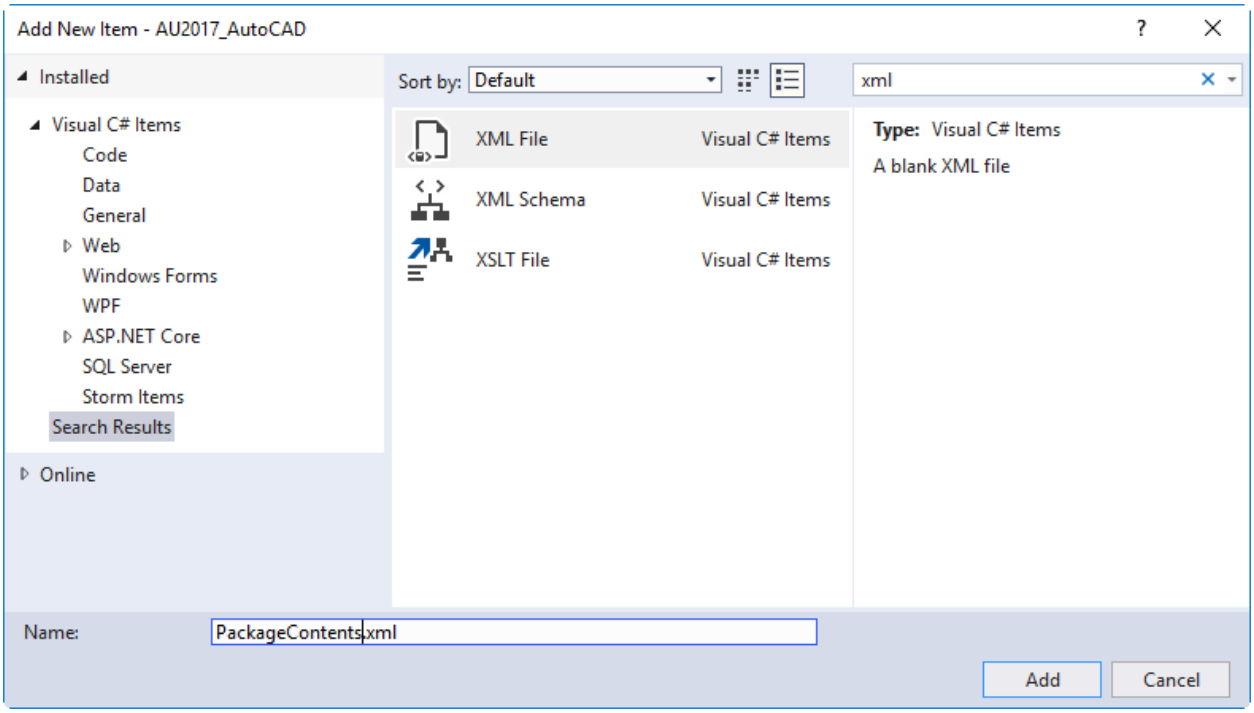
Right click on Class1.cs in the Solution Explorer and rename Class1.cs to App.cs

Put the following code in the class named App:

```
using Autodesk.Navisworks.Api.Plugins;
using System.Windows.Forms;
namespace AU2017_Navisworks
{
    [PluginAttribute("AU2017_Navisworks.App", "NNAW", ToolTip = "Plugin", DisplayName = "Plugin")]
    public class App : AddInPlugin
    {
```

```
        public override int Execute(params string[] parameters)
        {
            MessageBox.Show("Hello Autodesk University 2017");
            return 0;
        }
    }
}
```

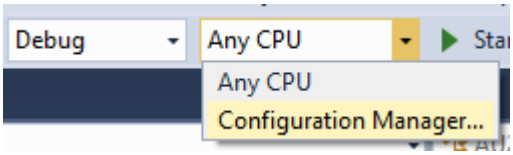
Add a new XML file to the project named PackageContents.xml



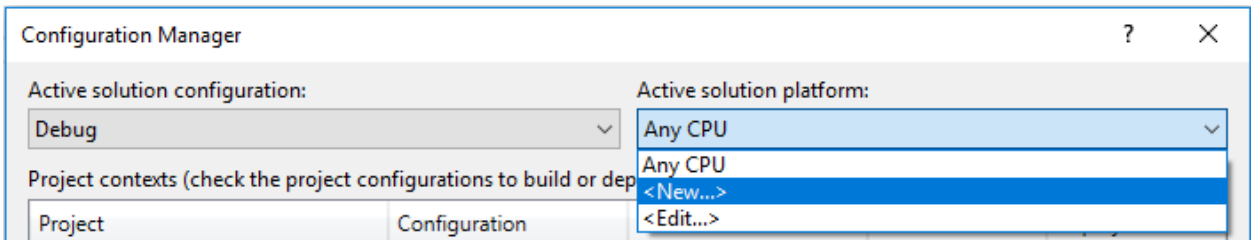
Replace the contents of the above file with:

```
<?xml version="1.0" encoding="utf-8"?>
<ApplicationPackage SchemaVersion="1.0" AppVersion="17.10.31" ProductCode="{ad7c8c48-32a9-4d66-
80ac-38e4cfbfefe9}" HelpFile="./Contents/Help.htm" Icon="./Contents/icon.bmp">
  <CompanyDetails Name="Company Name"/>
  <Components Description="2018">
    <RuntimeRequirements OS="Win64" Platform="NAVMAN|NAVSIM" SeriesMin="Nw15" SeriesMax="Nw15" />
    <ComponentEntry AppName="AU2017" AppType="ManagedPlugin" Version="17.10.31"
ModuleName="./Contents/2018/AU2017_Navisworks.dll" />
  </Components>
</ApplicationPackage>
```

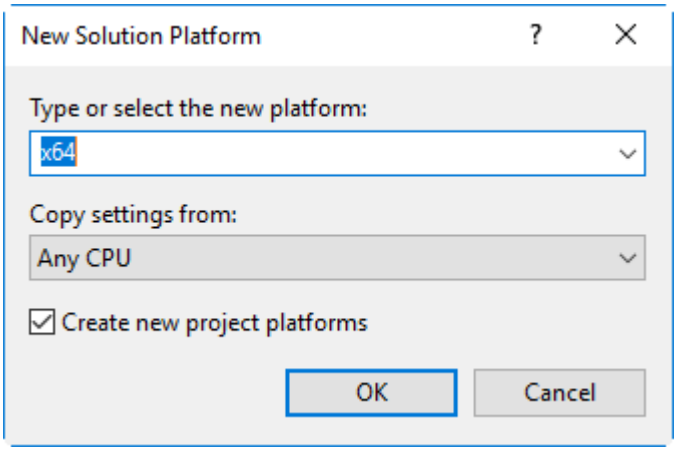
Start the Configuration Manager



Select New from the Any CPU Dropdown

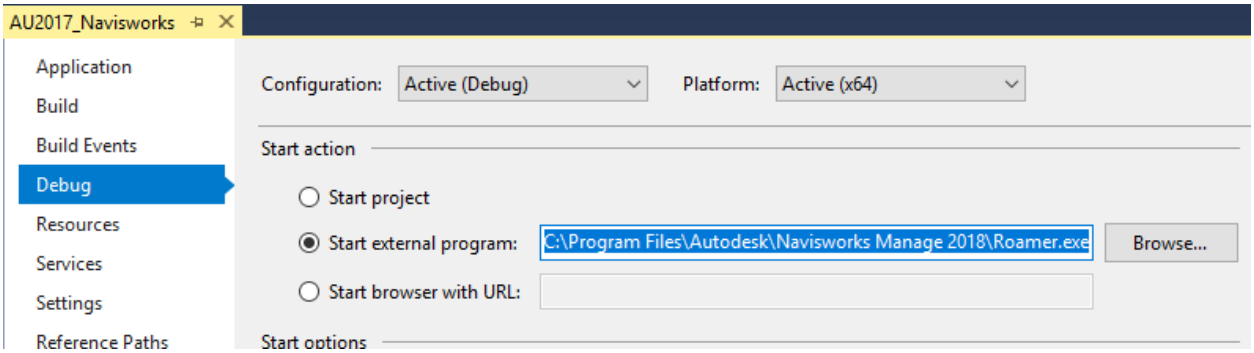


Verify x64 is the new platform copying settings from Any CPU and click OK



Close the Configuration Manager

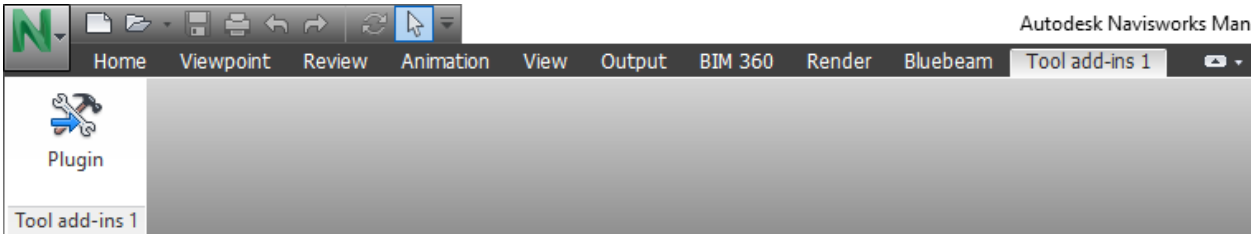
In the Properties Set the Debug Start Action to Start external program and pick C:\Program Files\Autodesk\Navisworks Manage 2018\Roamer.exe

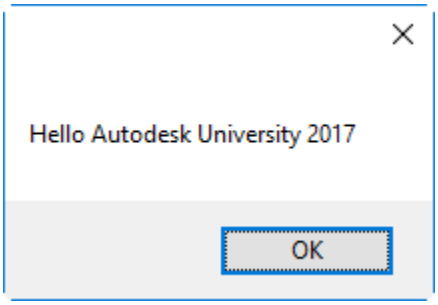


In the Properties Set the Build Events -> Post Build Event Command Line to:

```
xcopy "$(TargetDir)AU2017_Navisworks.dll"
"C:\ProgramData\Autodesk\ApplicationPlugins\AU2017.bundle\Contents\2018\" /i /e /y /c
xcopy "$(SolutionDir)AU2017_Navisworks\PackageContents.xml"
"C:\ProgramData\Autodesk\ApplicationPlugins\AU2017.bundle\" /i /e /y /c
```

Start the Debugging, you should get a welcome message when you click the Plugin command on the Tool add-ins Ribbon Palette.

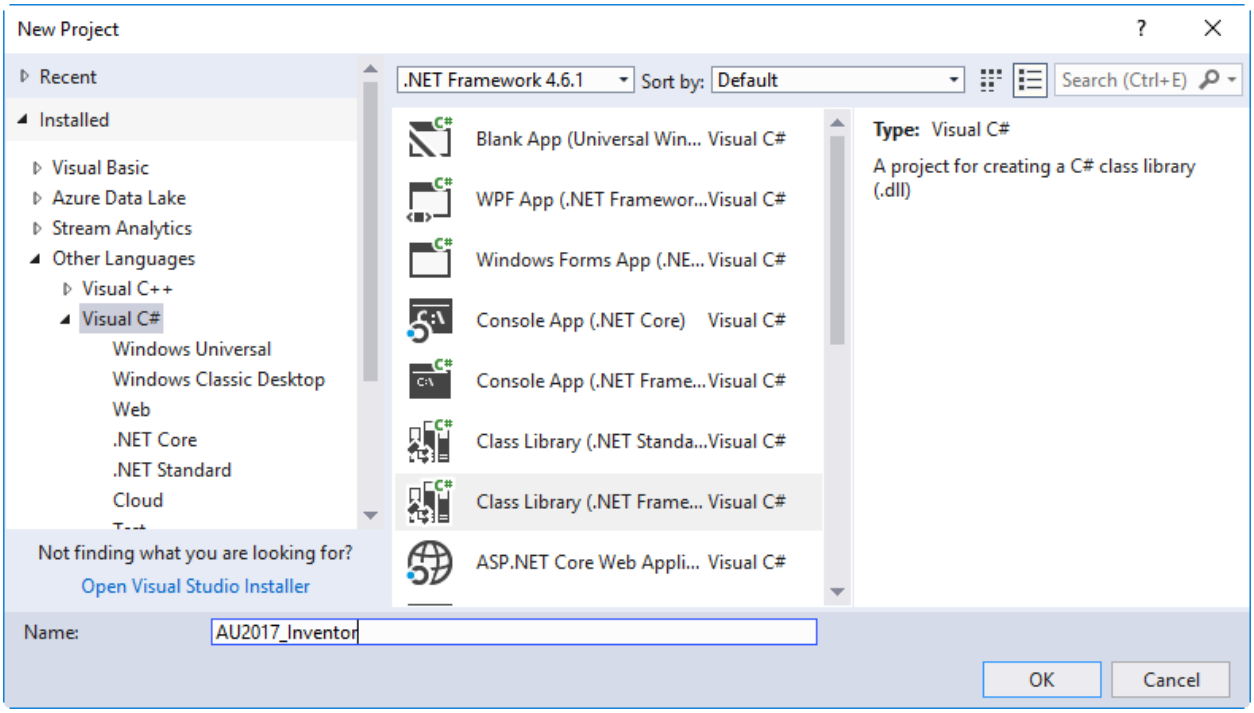




Inventor

There is a Video here showing the process:
https://1drv.ms/v/s!AnQuwhbiBgLhg_8bFa4IvSiMHbQpVw

Create a New Class Library Project



Right click on the Project in the Solution Explorer and select Add then Reference to add the following references:

C:\Program Files\Autodesk\Inventor 2018\Bin\Public Assemblies\Autodesk.Inventor.Interop.dll

And Assemblies -> System.Windows.Forms

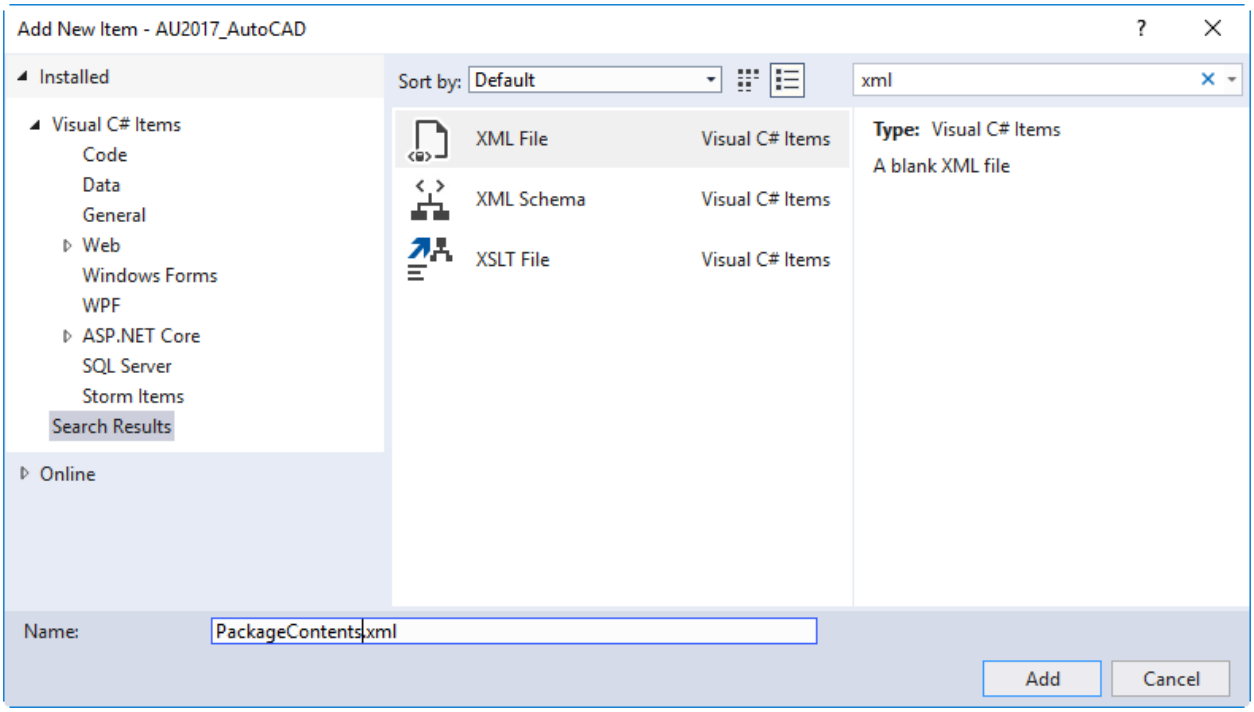
Right click on Class1.cs in the Solution Explorer and rename Class1.cs to StandardAddInServer.cs

Put the following code in the class named StandardAddInServer:

```
using System;
using System.Runtime.InteropServices;
using Inventor;
using System.Windows.Forms;
namespace AU2017_Inventor
{
    [Guid("e6d53d6c-442f-4979-a4ae-26916f1c59a0"), ComVisible(true)]
    public class StandardAddInServer : Inventor.ApplicationAddInServer
    {
```

```
        public StandardAddInServer()
        {
        }
        public void Activate(ApplicationAddInSite AddInSiteObject, bool FirstTime)
        {
            MessageBox.Show("Hello Autodesk University 2017");
        }
        public dynamic Automation
        {
            get
            {
                throw new NotImplementedException();
            }
        }
        public void Deactivate()
        {
            throw new NotImplementedException();
        }
        public void ExecuteCommand(int CommandID)
        {
            throw new NotImplementedException();
        }
    }
}
```

Add a new XML file to the project named PackageContents.xml



Replace the contents of the above file with:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
```

```
  <assemblyIdentity name="AU2017_Inventor" version="15.12.02" />
  <clrClass clsid="{e6d53d6c-442f-4978-a4ae-26916f1c59a0}"
            progid="AU2017_Inventor.StandardAddInServer"
            threadingModel="Both"
            name="AU2017_Inventor.StandardAddInServer"
            runtimeVersion="" />
  <file name="AU2017_Inventor.dll" hashalg="SHA1" />
</assembly>
```

Add a new XML file to the project named Autodesk.AU2017_Inventor.addin

Replace the contents of the above file with:

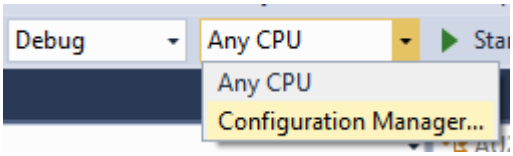
```
<?xml version="1.0" encoding="utf-16"?>
<Addin Type="Standard">
  <ClassId>{e6d53d6c-442f-4979-a4ae-26916f1c59a0}</ClassId>
  <ClientId>{e6d53d6c-442f-4979-a4ae-26916f1c59a0}</ClientId>
  <DisplayName>AU2017_Inventor</DisplayName>
  <Description>AU2017_Inventor Sample App</Description>

<Assembly>C:\ProgramData\Autodesk\ApplicationPlugins\AU2017.bundle\Contents\2018\AU2017_Inventor.dll</Asse
mbly>
  <LoadOnStartUp>1</LoadOnStartUp>
  <Hidden>0</Hidden>
</Addin>
```

Add a new XML file to the project named AU2017_Inventor.manifest

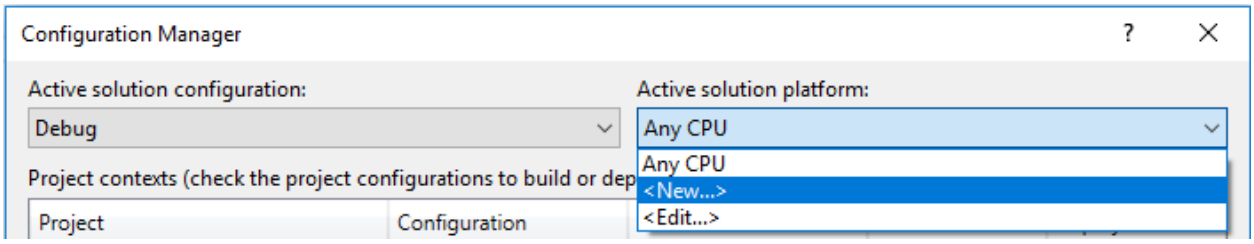
Replace the contents of the above file with:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
  <assemblyIdentity name="AU2017_Inventor" version="15.12.02" />
  <clrClass clsid="{e6d53d6c-442f-4979-a4ae-26916f1c59a0}"
            progid="AU2017_Inventor.StandardAddInServer"
            threadingModel="Both"
            name="AU2017_Inventor.StandardAddInServer"
            runtimeVersion="" />
  <file name="AU2017_Inventor.dll" hashalg="SHA1" />
</assembly>
```

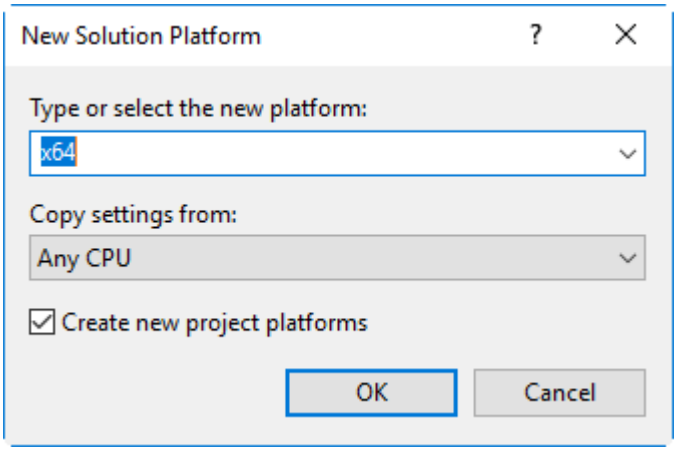
Start the Configuration Manager



Select New from the Any CPU Dropdown



Verify x64 is the new platform copying settings from Any CPU and click OK



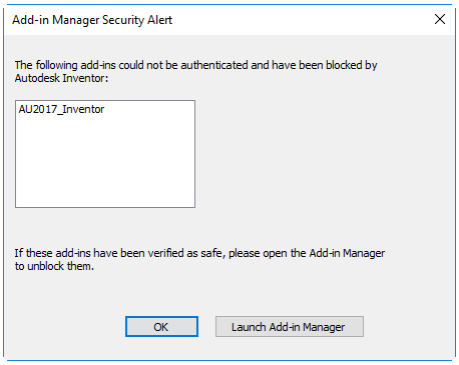
Close the Configuration Manager

In the Properties Set the Debug Start Action to Start external program and pick Inventor.exe

In the Properties Set the Build Events -> Post Build Event Command Line to:

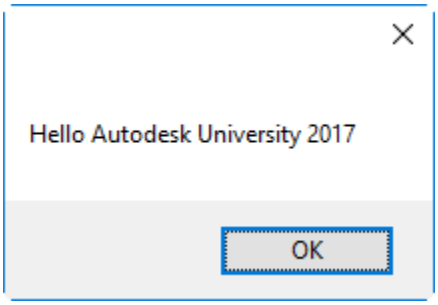
```
call "%VS150COMNTOOLS%vsvars32.bat"

mt.exe -manifest "$(ProjectDir)\AU2017_Inventor.manifest" -outputresource:"$(TargetPath)";#2

xcopy "$(TargetDir)AU2017_Inventor.dll"
"C:\ProgramData\Autodesk\ApplicationPlugins\AU2017.bundle\Contents\2018\" /i /e /y /c

xcopy "$(ProjectDir)Autodesk.AU2017_Inventor.addin" "C:\ProgramData\Autodesk\Inventor 2018\Addins\" /i /e
/y /c
```

Start the Debugging, you will get the following message:



Click OK to accept your addin as safe to run.

Create a New Part once Inventor loads then you should get a welcome message:



You may need to go to Add-Ins and Unblock the Addin inorder to get it to load.