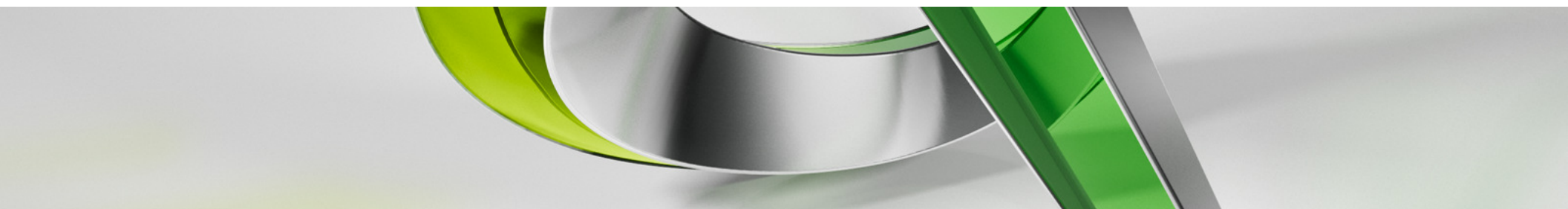




How I learnt to create AddIns for AutoCAD, Revit, Navisworks and Inventor in 3 months

Drew Jarvis

Applications Specialist



Class summary

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.

Drew Jarvis

16+ years experience with Autodesk Products, specializing in AutoCAD, Revit, Navisworks

Started with LISP and VBA in 2002

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

Wrote simple how too instructions on creating addins and presented them at the end of a presentation at AU2015

Drew Jarvis... the truth

Am I an expert programmer.....

No

Have I taken more than 36 hours of training in programming in my life?

No

Do I rely on google when I program?

Sure Do!

So If I can do it

You can do it!

Key learning objectives

At the end of this class, you will be able to:

- Learn how to create add-ins for multiple Autodesk Solutions
- Learn how to load up Dockable Windows in each product
- Learn how to connect to a SQL database and save data into a DataTable for later use
- Discover the types of Event Handlers available in the different Autodesk Solutions

What did I need to do?

The Challenge

Global eTraining (GeT) is an award-winning Canadian-based provider of interactive online training solutions. They are a strategic training partner to some of the world's largest and most diverse multinational design, construction and software organizations, government bodies and educational institutions, and the global leader in designing training for the 21st Century.

They asked me to produce Add-Ins for them, at the time I had successfully created an Add-In for Revit 2014...

The Challenge

Generate an addin that reacts to software use to provide training tools like PDF's and Videos

Initially for Revit and AutoCAD

2nd phase was for AutoCAD Verticals, Inventor and Navisworks

What I needed/wanted

Easy Installation

Easy loading

Modern appearance

Event Handlers

Database Connection

Entitlement

Easy Installation - Autoloader distribution

What is the Autoloader?

The Autoloader is designed to simplify 90% of all Autodesk application deployments.

Allows you to deploy your plugins as a simple package format.

Specifies a few select locations that you can place your files.

Simplified the installation process for the end user and the developer

Autoloader Files

[%PROGRAMDATA%\Autodesk\ApplicationPlugins\](#)

Or

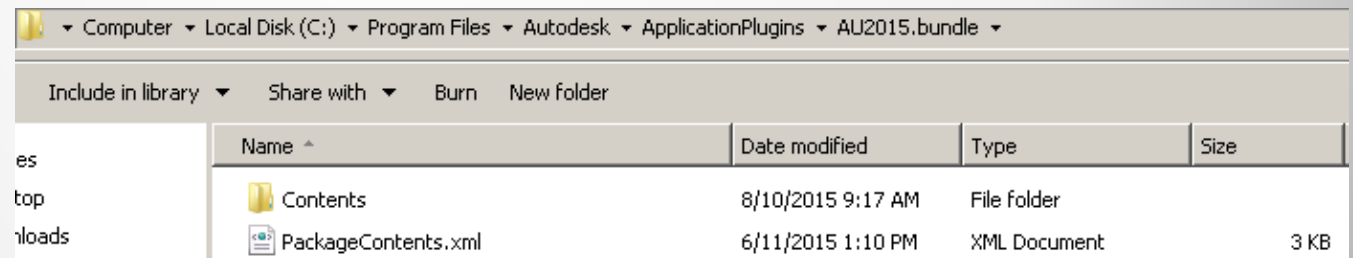
[%APPDATA%\Autodesk\ApplicationPlugins\](#)

Or

[C:\Program Files\Autodesk\ApplicationPlugins\](#)

.bundle folder contains an xml file that describes the files to be loaded for the PlugIn

PackageContents.xml



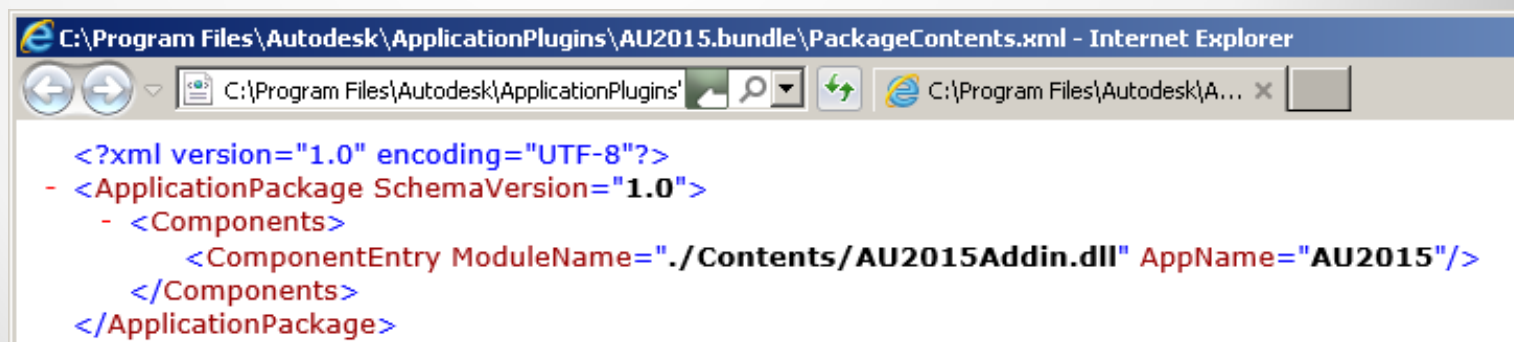
Name ^	Date modified	Type	Size
Contents	8/10/2015 9:17 AM	File folder	
PackageContents.xml	6/11/2015 1:10 PM	XML Document	3 KB



PackageContents.xml

Great Whitepaper Here:

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



```
<?xml version="1.0" encoding="UTF-8"?>
- <ApplicationPackage SchemaVersion="1.0">
  - <Components>
    <ComponentEntry ModuleName="./Contents/AU2015Addin.dll" AppName="AU2015"/>
  </Components>
</ApplicationPackage>
```

Easy loading – Run on StartUp

Loading Automatically

I found that the process for this was different for different platforms:

Revit requires an addin file that references the dll, this however still requires you to click a button to start the addin

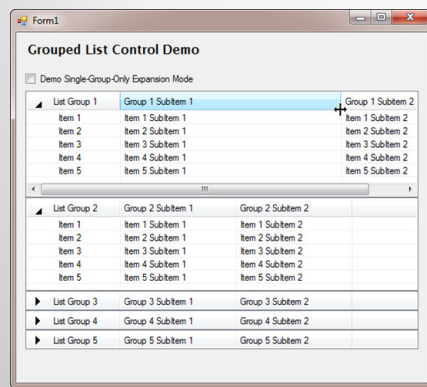
AutoCAD required placing a StartupCommand="Yes" item into the PackageContents.xml

Modern appearance – Dockable Palettes XAML/WPF

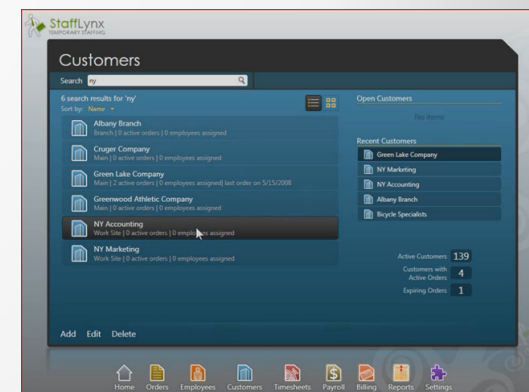
Modern Appearance

Dockable palettes are the name I give to interface items that I saw in AutoCAD 2004 (?) where you can have them Auto-hide and dock into the side of the screen, this is the appearance I wanted for the Add-In Content.

I also wanted to get away from traditional Winforms, which I think look dated, WPF was the solution that I found from my searches.



versus



All of the Software has Dockable Palettes

However they are loaded in different ways and have different names

AutoCAD – PaletteSet

Revit – DockablePane

Navisworks - DockableWindow

Inventor - DockableWindow

Dockable Panes in Revit

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

```
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 😊

PaletteSets in AutoCAD

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

```
public static DockablePane uc = new DockablePane();  
ps = new PaletteSet("AU2015");  
ps.Size = new System.Drawing.Size(400, 600);  
ps.DockEnabled = (DockSides)((int)DockSides.Left + (int)DockSides.Right);  
ps.Style = PaletteSetStyles.ShowAutoHideButton;  
ps.AddVisual("AddVisual", uc);  
ps.KeepFocus = true;  
ps.Visible = true;
```

You then just need to add content to MainDockableWindow, note that DockablePane 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.UIManager;  
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;
```

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

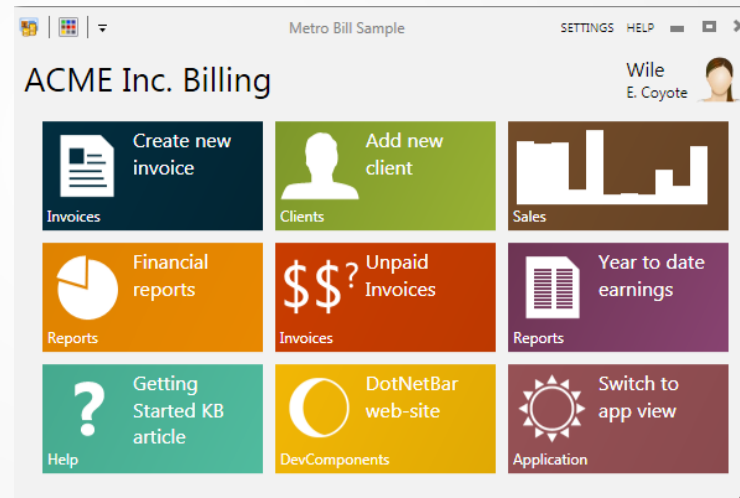
Good Reference:

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

Modern Look

Win Forms seemed old

I didn't want an interface that looked like this



I wanted an interface that looked like this

WPF for the GUI

So in order to have more control over the interface and to make it appear modern I figured WPF was the solution.

WinForms were old 😊

WinRT was too new (Windows 8+, didn't support Windows 7 or older)

WPF was my favoured solution

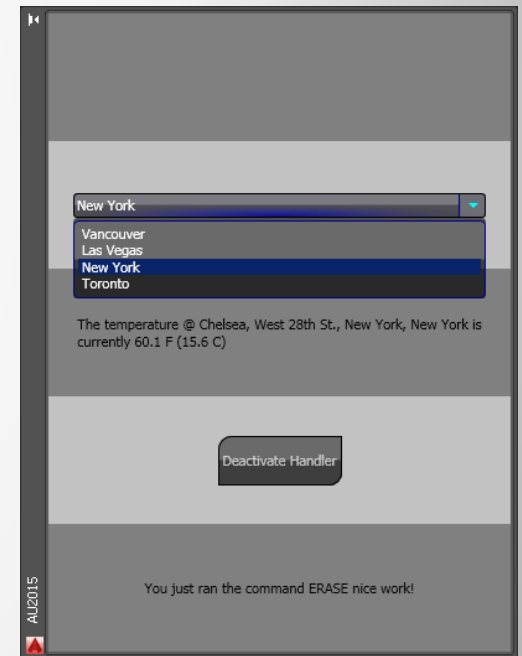
WPF?

WPF (Windows Presentation Foundation) uses XAML

Controls can contain other controls or media

Styles and Templates give greater control

```
<UserControl x:Class="AutoCADAU2015.DockableItem"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  mc:Ignorable="d"
  d:DesignHeight="300" d:DesignWidth="300">
  <UserControl.Resources>
    <ResourceDictionary>
      <ResourceDictionary.MergedDictionaries>
        <ResourceDictionary Source="/SharedAU2015;component/Styles.xaml"/></ResourceDictionary>
      </ResourceDictionary.MergedDictionaries>
    </ResourceDictionary>
  </UserControl.Resources>
```



Event Handler

Handling Events

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 😊

The Good

AutoCAD – general catch all

The Bad

Revit – Specific event handlers

The Ugly

Navisworks/Inventor... - Nothing... Nada... So what to do?

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, lets take a look.

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/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  
    }  
}
```

SQL Cloud connections

Connecting to a cloud database was a great way to provide upto date information to the client.

I selected Microsoft Azure as my base, this enabled easy connections and high level up time (and I am a bit of a MS Fanboy, I have a Lumia phone for example 😊)

```
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();
```


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 gaurenteed “updatedness” – but I could live with this.

Entitlement

Publishing to the Exchange App Store

The Exchange App is a great place to market / promote your AddIns, and the process of submitting to them is straight forward, however one of the more challenging parts was making use of the subscription payment method.

To do this you need to check that a user has purchased your product, and to do that you need to check that they are... entitled.

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 user account that they used to download the app.

The entitlement check will also work for subscription based AddIns where it will check that the user has an active subscription for the AddIn

Entitlement API

```
public const string _AppId = @"< appstore.exchange.autodesk.com:*****>";
public static Result Execute(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;
}
```



Creating Addins

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

Starting a Revit Addin 1 of 4

Create a New Class Library Project

Add References to RevitAPI.dll, RevitAPIUI.dll & System.Windows.Forms

Put the following code in a class named App:

```
using System;
using Autodesk.Revit.UI;
using System.Windows.Forms;

namespace RevitAU2015
{
    class App : IExternalApplication
    {
        public Result OnShutdown(UIControlledApplication application)
        {
            throw new NotImplementedException();
        }

        public Result OnStartup(UIControlledApplication application)
        {
            MessageBox.Show("Hello Autodesk University");
            throw new NotImplementedException();
        }
    }
}
```

Starting a Revit Addin 2 of 4

Add a new XML file to the project named PackageContents.xml
Replace the contents of the above file with:

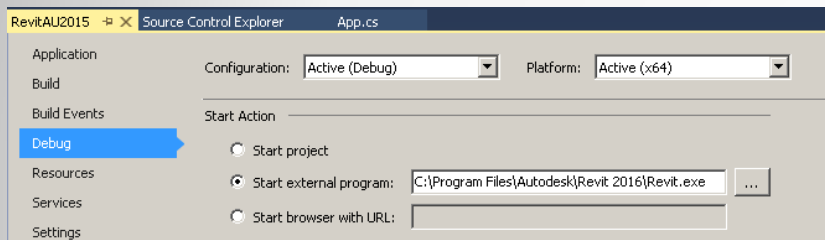
```
<?xml version="1.0" encoding="utf-8" ?>
<ApplicationPackage SchemaVersion="1.0" AppVersion="15.12.02" ProductCode="b57c480f-24f5-4fbf-804c-7df86c770305">
  <RuntimeRequirements OS="Win64" Platform="Revit" SeriesMin="R2016" SeriesMax="R2016" />
  <Components Description="2016">
    <RuntimeRequirements OS="Win64" Platform="Revit" SeriesMin="R2016" SeriesMax="R2016" />
    <ComponentEntry AppName="RevitAU2015" ModuleName="./Contents/2016/RevitAU2015.addin"></ComponentEntry>
  </Components>
</ApplicationPackage>
```

Add a new XML file to the project named RevitAU2015.addin
Replace the contents of the above file with:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<RevitAddIns>
  <AddIn Type="Application">
    <Name>AU2015</Name>
    <Assembly>C:\ProgramData\Autodesk\ApplicationPlugins\AU2015.bundle\Contents\2016\RevitAU2015.dll</Assembly>
    <AddInId>a694f098-20d5-42c2-a8a0-ac7ae6857c24</AddInId>
    <FullClassName>RevitAU2015.App</FullClassName>
    <ClientId>0c37a558-6df2-44b7-b367-70b469a399a7</ClientId>
    <VendorId>272c22da-13ee-40ad-9b9f-d014a7fda46a</VendorId>
    <VendorDescription>CompanyName, www.CompanyName.com</VendorDescription>
  </AddIn>
</RevitAddIns>
```


Starting a Revit Addin 3 of 4

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



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

```
xcopy "$(TargetDir)RevitAU2015.dll" "C:\ProgramData\Autodesk\ApplicationPlugins\AU2015.bundle\Contents\2016\" /i /e /y /c  
xcopy "$(SolutionDir)RevitAU2015\PackageContents.xml" "C:\ProgramData\Autodesk\ApplicationPlugins\AU2015.bundle\" /i /e /y /c  
xcopy "$(SolutionDir)RevitAU2015\RevitAU2015.addin" "C:\ProgramData\Autodesk\ApplicationPlugins\AU2015.bundle\Contents\2016\" /i /e /y /c
```

Open the file RevitAU2015.csproj & add the 'DebugEngines' property within 'PropertyGroup' like the below

```
<PropertyGroup>  
  <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>  
  <DebugEngines>{351668CC-8477-4fbf-BFE3-5F1006E4DB1F}</DebugEngines>
```

Close and Save the file RevitAU2015.csproj

Starting a Revit Addin 4 of 4

Start the Debugging, you should get a welcome message when Revit starts.

Starting an AutoCAD Addin 1 of 3

Create a New Class Library Project

Add References to AcDbMgd.dll , AcMgd.dll, AcCoreMgd.dll & System.Windows.Forms

Put the following code in a class named App:

```
using Autodesk.AutoCAD.Runtime;
using System.Windows.Forms;

namespace AutoCADAU2015
{
    public class App
    {
        [CommandMethod("AU2015GROUP", "AU2015", CommandFlags.NoActionRecording)]
        public void AU2015()
        {
            MessageBox.Show("Hello Autodesk University");
        }
    }
}
```

Starting an AutoCAD Addin 2 of 3

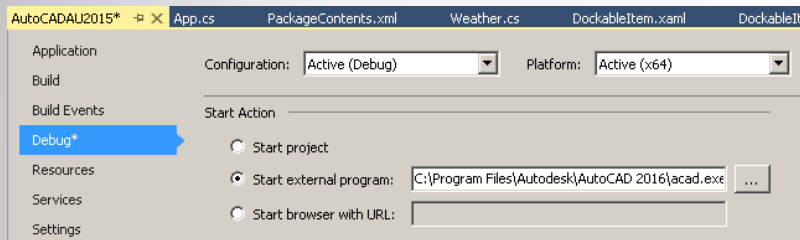
Add a new XML file to the project named PackageContents.xml
Replace the contents of the above file with:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<ApplicationPackage SchemaVersion="1.0" AppVersion="15.11.19" 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="20.0" SeriesMax="20.1" />
  <Components Description="2016_64">
    <RuntimeRequirements OS="Win64" Platform="AutoCAD" SeriesMin="20.1" SeriesMax="20.1" />
    <ComponentEntry AppName="AU2015" Version="15.11.19" ModuleName="./Contents/2016/AutoCADAU2015.dll" AppDescription="AU2015 Add-In"
LoadOnAutoCADStartup="True">
      <Commands GroupName="AU2015GROUP">
        <Command Local="AU2015" Global="AU2015" StartupCommand="True" ></Command>
      </Commands>
    </ComponentEntry>
  </Components>
</ApplicationPackage>
```



Starting an AutoCAD Addin 3 of 3

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



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

```
xcopy "$(TargetDir)AutoCADAU2015.dll" "C:\ProgramData\Autodesk\ApplicationPlugins\AU2015.bundle\Contents\2016\" /i /e /y /c  
xcopy "$(SolutionDir)AutoCADAU2015\PackageContents.xml" "C:\ProgramData\Autodesk\ApplicationPlugins\AU2015.bundle\" /i /e /y /c
```

Start the Debugging, you should get a welcome message when AutoCAD starts.

Starting a Navisworks Addin 1 of 3

Create a New Class Library Project

Add References to Autodesk.Navisworks.Api.dll & System.Windows.Forms

Put the following code in a class named App:

```
using Autodesk.Navisworks.Api.Plugins;
using System.Windows.Forms;

namespace NavisworksAU2015
{
    [PluginAttribute("NavisworksAU2015.App", "NNAW", ToolTip = "Plugin", DisplayName = "Plugin")]
    public class App : AddInPlugin
    {
        public override int Execute(params string[] parameters)
        {
            MessageBox.Show("Hello Autodesk University");
            return 0;
        }
    }
}
```

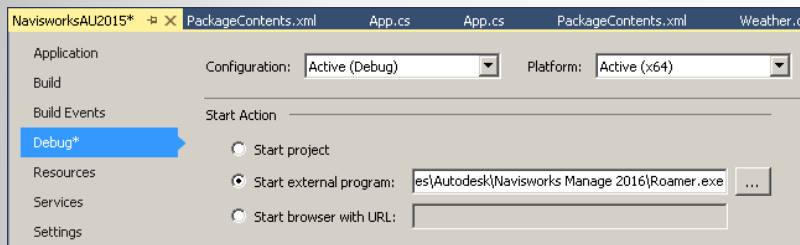
Starting a Navisworks Addin 2 of 3

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="15.12.02" ProductCode="{ad7c8c48-32a9-4d66-80ac-38e4cfbfefe9}" HelpFile="./Contents/Help.htm"
Icon="./Contents/icon.bmp">
  <CompanyDetails Name="Company Name"/>
  <Components Description="2016">
    <RuntimeRequirements OS="Win64" Platform="NAVMAN|NAVSIM" SeriesMin="Nw13" SeriesMax="Nw13" />
    <ComponentEntry AppName="AU2015" AppType="ManagedPlugin" Version="15.12.02" ModuleName="./Contents/2016/NavisworksAU2015.dll" />
  </Components>
</ApplicationPackage>
```

Starting a Navisworks Addin 3 of 3

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



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

```
xcopy "$(TargetDir)NavisworksAU2015.dll" "C:\ProgramData\Autodesk\ApplicationPlugins\AU2015.bundle\Contents\2016\" /i /e /y /c  
xcopy "$(SolutionDir)NavisworksAU2015\PackageContents.xml" "C:\ProgramData\Autodesk\ApplicationPlugins\AU2015.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.



Starting an Inventor Addin 1 of 3

Create a New Class Library Project

Add References to Autodesk.Inventor.Interop.dll & System.Windows.Forms

Put the following code in a class named StandardAddInServer:

```
using System;
using System.Runtime.InteropServices;
using Inventor;
using System.Windows.Forms;

namespace InventorAU2015
{
    [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");
        }
        public dynamic Automation
        {
            get
            {
                throw new NotImplementedException();
            }
        }
        public void Deactivate()
        {
            throw new NotImplementedException();
        }
        public void ExecuteCommand(int CommandID)
        {
            throw new NotImplementedException();
        }
    }
}
```

Starting an Inventor Addin 2 of 3

Add a new XML file to the project named InventorAU2015.manifest
Replace the contents of the above file with:

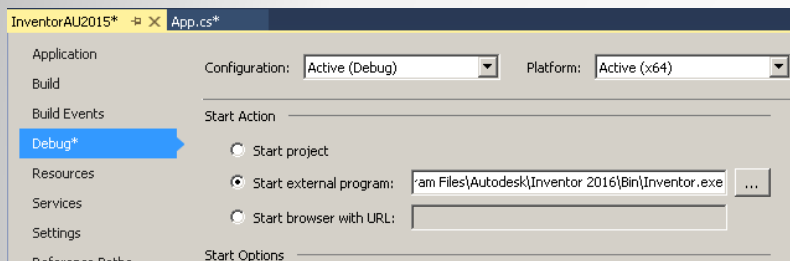
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
  <assemblyIdentity name="InventorAU2015" version="15.12.02" />
  <clrClass clsid="{e6d53d6c-442f-4979-a4ae-26916f1c59a0}"
    progid="InventorAU2015.StandardAddInServer"
    threadingModel="Both"
    name="InventorAU2015.StandardAddInServer"
    runtimeVersion="" />
  <file name="InventorAU2015.dll" hashalg="SHA1" />
</assembly>
```

Add a new XML file to the project named Autodesk.InventorAU2015.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>InventorAU2015</DisplayName>
  <Description>InventorAU2015 Sample App</Description>
  <Assembly>C:\ProgramData\Autodesk\ApplicationPlugins\AU2015.bundle\Contents\2016\InventorAU2015.dll</Assembly>
  <LoadOnStartup>1</LoadOnStartup>
  <Hidden>0</Hidden>
</Addin>
```

Starting an Inventor Addin 3 of 3

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 "%VS140COMNTOOLS%vsvars32.bat"  
mt.exe -manifest "$(\ProjectDir)\InventorAU2015.manifest" -outputresource:"$(TargetPath)";#2  
xcopy "$(\TargetDir)InventorAU2015.dll" "C:\ProgramData\Autodesk\ApplicationPlugins\AU2015.bundle\Contents\2016\" /i /e /y /c  
xcopy "$(\ProjectDir)Autodesk.InventorAU2015.Inventor.addin" "C:\ProgramData\Autodesk\Inventor 2016\Addins\" /i /e /y /c
```

Start the Debugging, you should get a welcome message

