AUTODESK UNIVERSITY

SD22323-L

Hands-on Introduction to C# Add-ins for AutoCAD

Jerry Winters
VB CAD, Inc.

Learning Objectives

- Learn how to create a new AutoCAD add-in in C#
- Learn how to create a new AutoCAD Command in C#
- Learn the basics of Drawing in AutoCAD through the .NET API
- Learn how to extract Block and Attribute information through the .NET API

Description

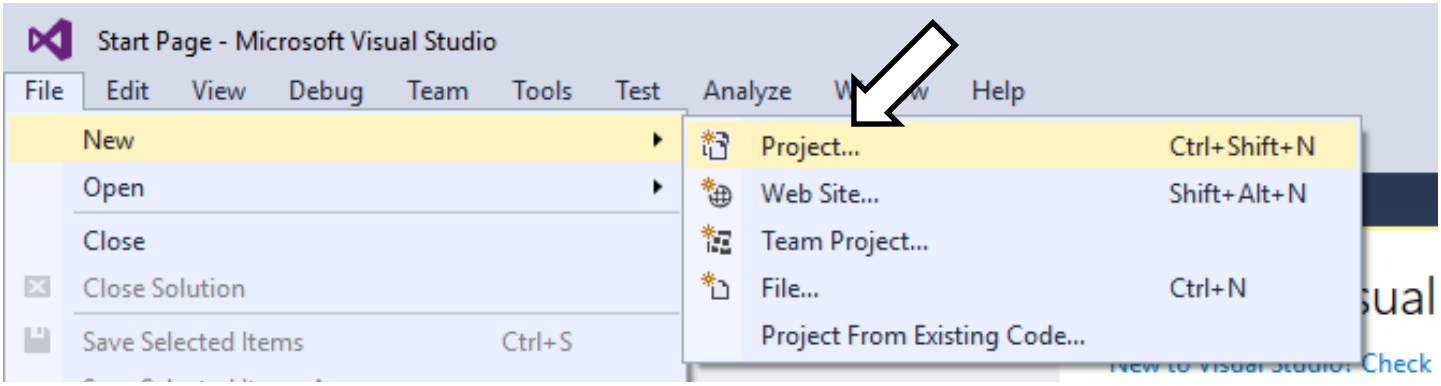
Knowing how to write AutoCAD software add-ins enables us to create new AutoCAD software commands that automate routine tasks, perform complex calculations, or integrate multiple systems to help us be more productive, more precise, and more effective. The use of Microsoft's community edition of Visual Studio, together with the information presented in this class, will help anyone begin creating AutoCAD software add-ins with C# programming language immediately. This session features AutoCAD, AutoCAD MEP, and AutoCAD Map 3D. AIA Approved

Your AU Expert

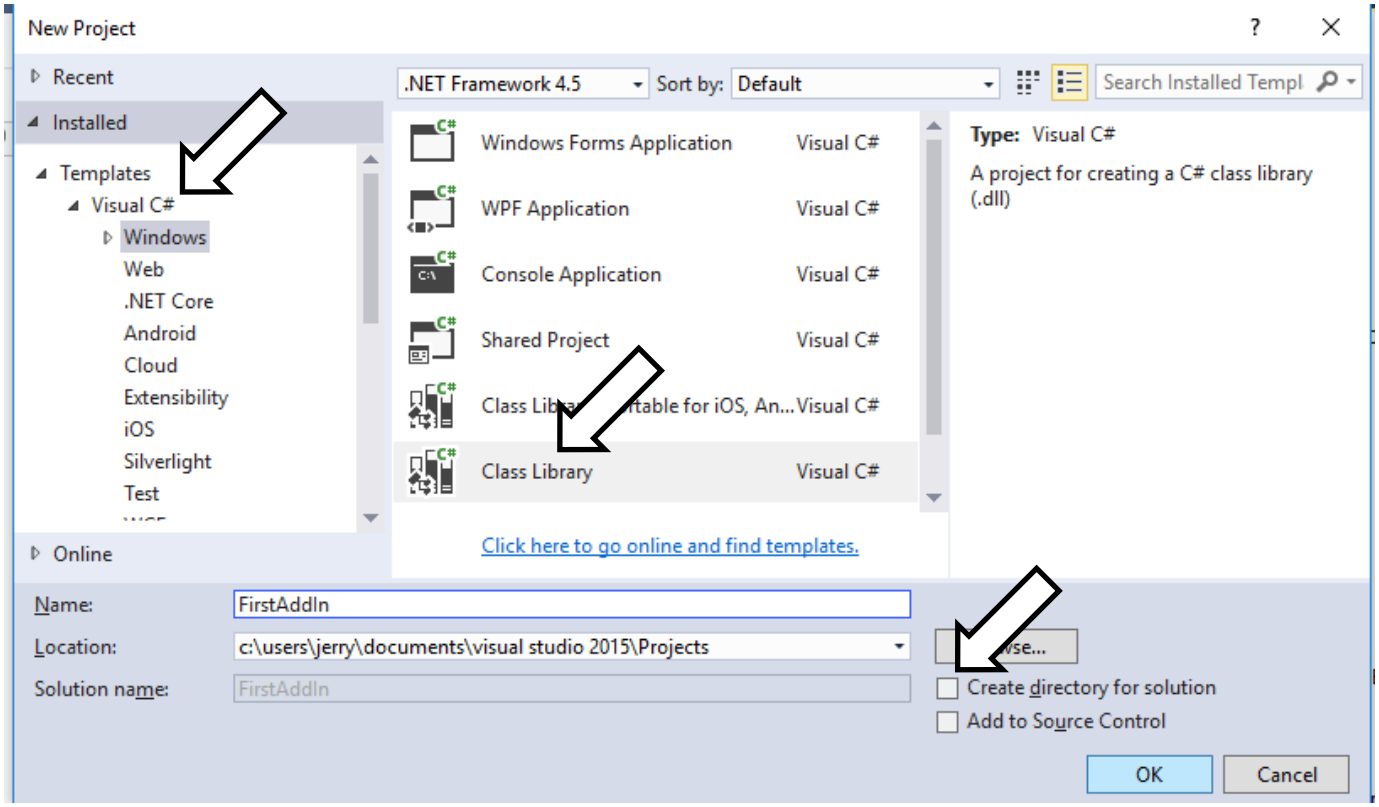
Jerry Winters has been writing add-ins for AutoCAD software since 1992, and he lends his experience to developing powerful, useful add-ins to improve accuracy, increase efficiency, and eliminate user error. Although technology continues to advance at a rapid pace, Winters strives to be aware of and proficient with emerging technologies.

Creating an Add-In using Visual Studio Community 2015 C#

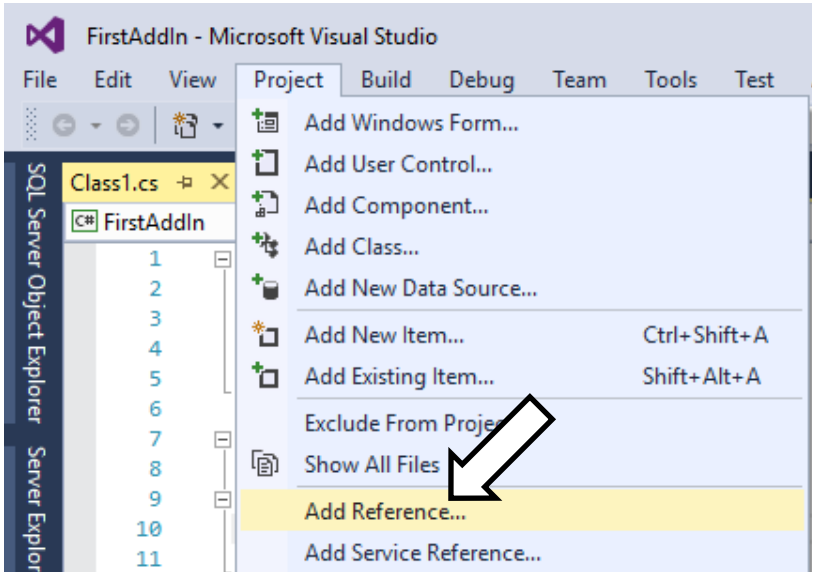
Step 1: Using the Visual Studio Menu, go to ***File -> New -> Project***.

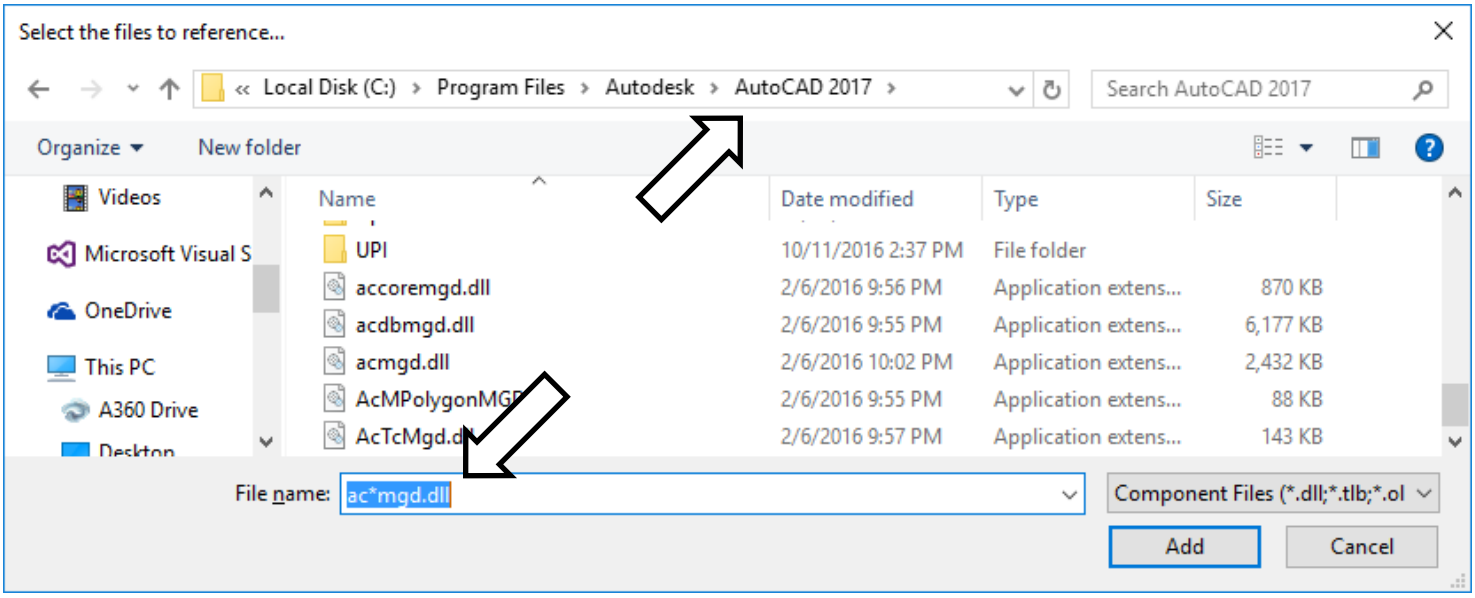


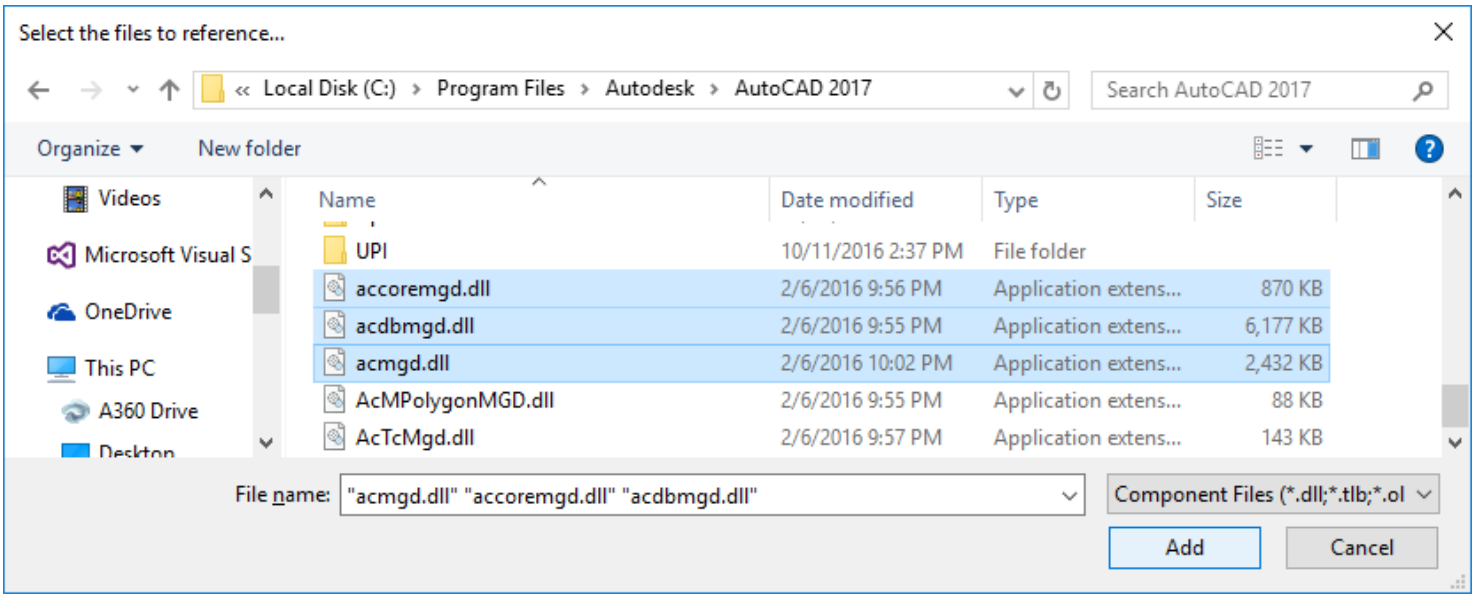
Step 2: Select the "Class Library" Template under the C#\Windows Template Type and give your Project a Name. In this example, the name is "FirstAddIn". Before you click the "OK" button, make sure the "Create directory for solution" checkbox is NOT checked.



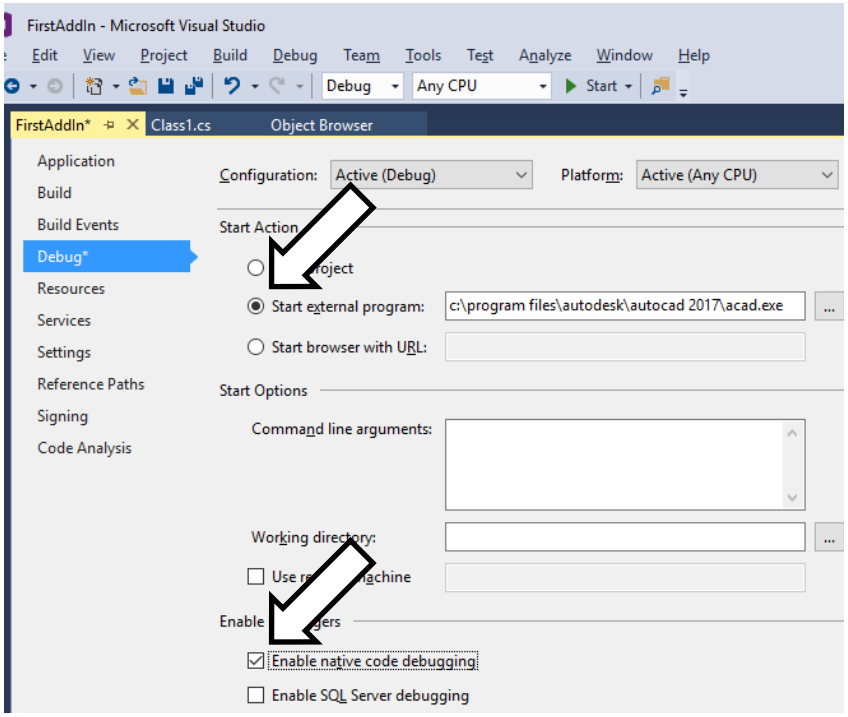
Step 3: Add References to the files “accoremgd.dll”, “acdbmgd.dll”, and “acmgd.dll” by going to the Visual Studio Menu: **Project -> Add Reference**.



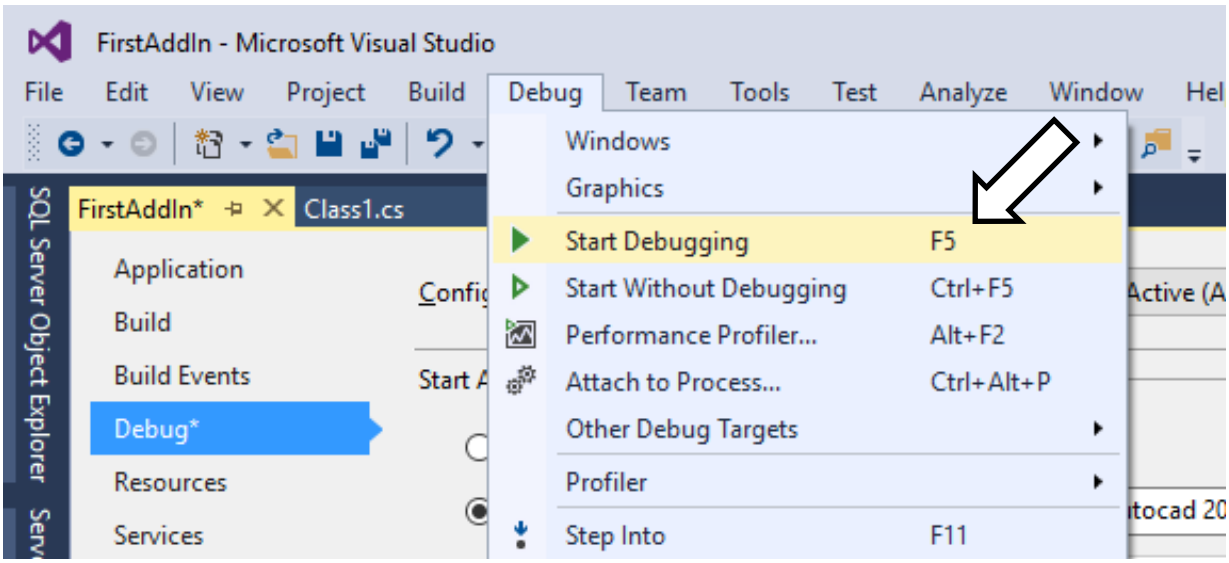




Step 4: Tell Visual Studio to start AutoCAD for debugging purposes by going to the Debug Tab in the Project Properties. **Project -> FirstAddIn Properties**. While you are here, check the box labeled “Enable native code debugging”.

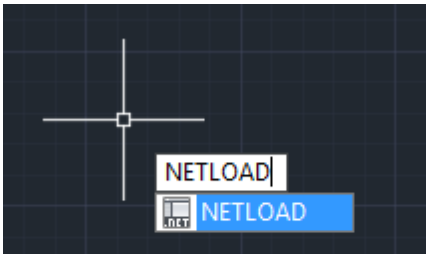


Step 5: Begin Debugging by going to the Menu: **Debug -> Start Debugging**



Step 6: In AutoCAD, set the SECURELOAD value to 0 in AutoCAD. This only need to happen once.

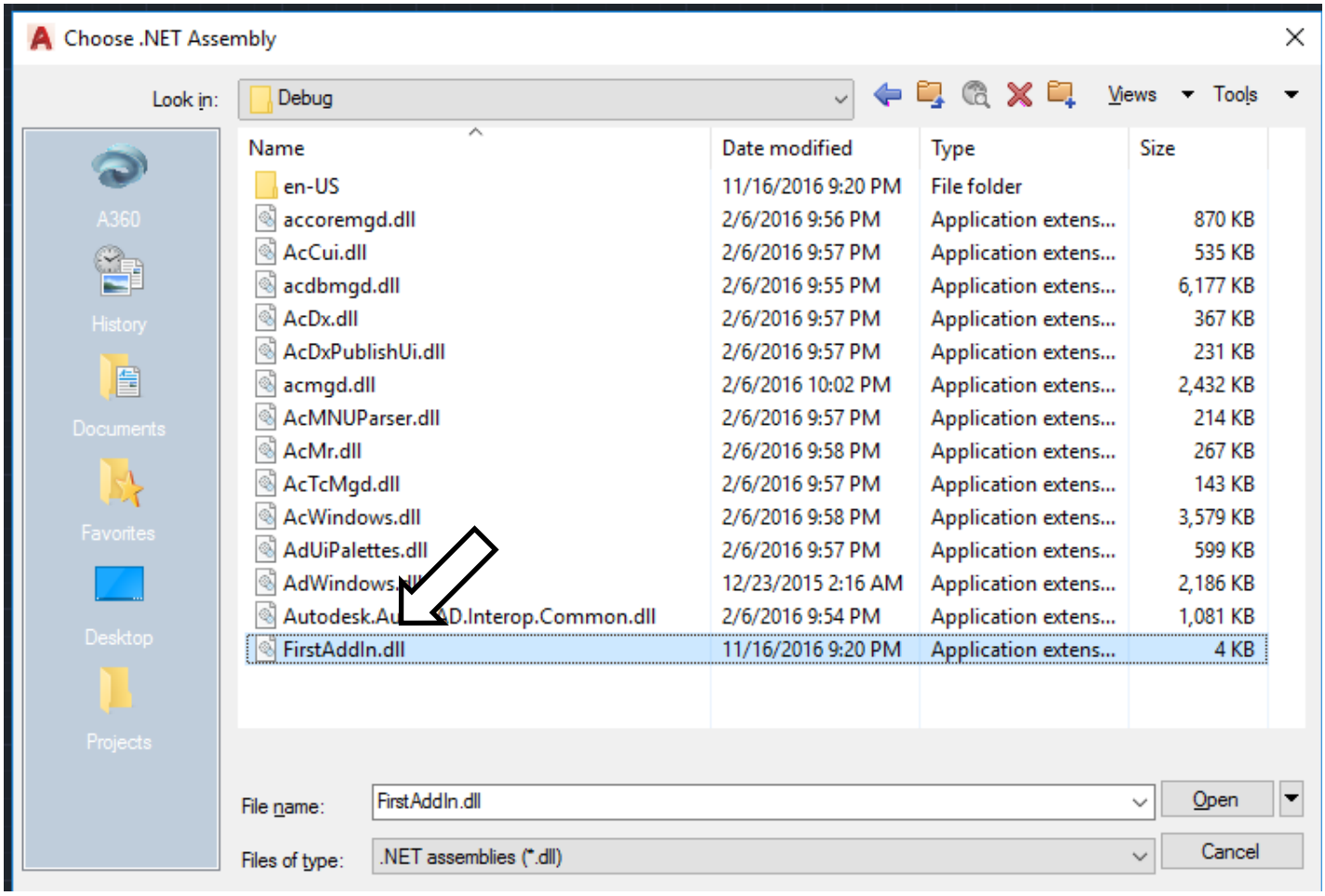
Step 7: “Netload” the DLL

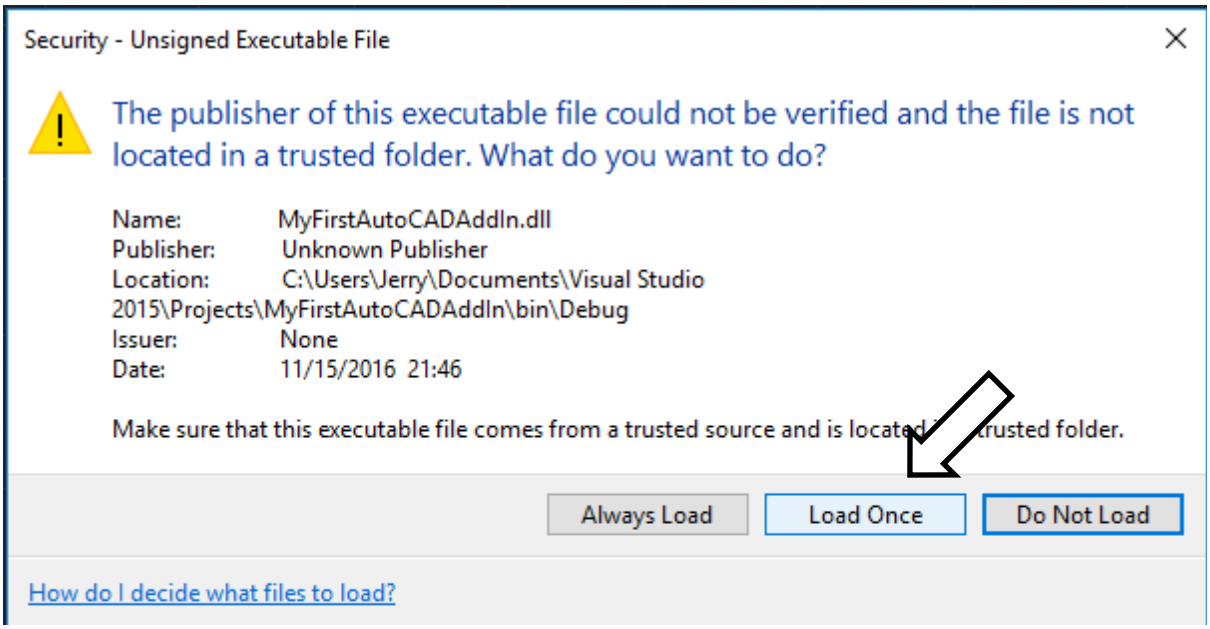


The DLL should be in the path:

C:\Users\Jerry\Documents\Visual Studio 2015\Projects\FirstAddIn\bin\Debug

Click on “Load Once” in the subsequent dialog box if it shows up.





Step 8: That's all.

Congratulations. You just created an AutoCAD Add-In using VB.NET. What does it do? Take a look at the next page to see.

Before we go any further, we need to close AutoCAD so we can change our code.

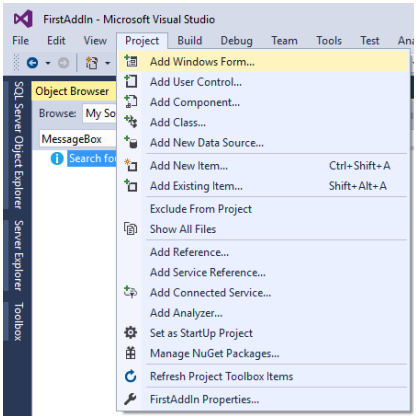
Nothing. It does nothing.

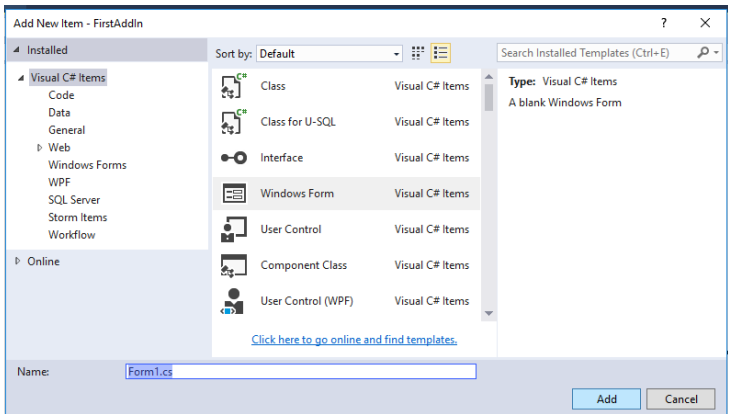
Why?

We didn't ask it do to anything.

The fact that it was created and netloaded without error says something. It tells us that we are well on our way to creating an AutoCAD Add-In using C#.

Before we continue, we are going to add a new Windows Form to our project. When we do this, a new Reference to the System.Windows.Forms will be added automatically which we need to show our MessageBoxes.





Create a new AutoCAD Command in C#

Creating a Command in C# is as simple as this:

```
[Autodesk.AutoCAD.Runtime.CommandMethod("Command1")]
public void Command1()
{
    System.Windows.Forms.MessageBox.Show("Whatever you do, don't type Hello World!!!");
}
```

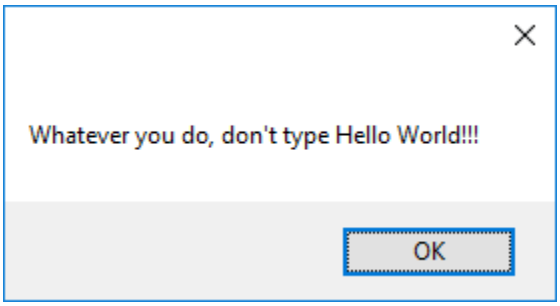
There you have it. The name of the command is “Command1”. Let’s take a look at another one. Please note this must be placed inside the Class1 Class.

```
<Autodesk.AutoCAD.Runtime.CommandMethod("Command2",
                Autodesk.AutoCAD.Runtime.CommandFlags.Session)>
Public Sub Command2()
    MsgBox("Session Flag means the command can cross multiple documents.")
End Sub
```

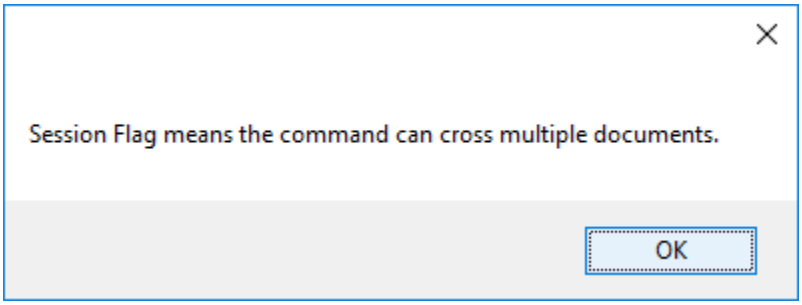
Command2 uses the Session flag which tells AutoCAD the command can cross multiple documents if this is needed.

```
[Autodesk.AutoCAD.Runtime.CommandMethod("Command3",
       Autodesk.AutoCAD.Runtime.CommandFlags.UsePickSet)]
public void Command3()
{
    Autodesk.AutoCAD.ApplicationServices.Document myDoc =
        Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocument;
    Autodesk.AutoCAD.EditorInput.Editor myEd = myDoc.Editor;
    Autodesk.AutoCAD.EditorInput.PromptSelectionResult myPSR = myEd.SelectImplied();
    if (myPSR.Status == Autodesk.AutoCAD.EditorInput.PromptStatus.OK)
    {
        System.Windows.Forms.MessageBox.Show(myPSR.Value.Count.ToString() + " selected.");
    }
    else
    {
        System.Windows.Forms.MessageBox.Show("0 selected.");
    }
}
```

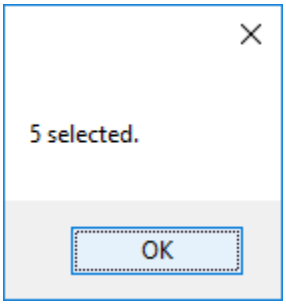
Command3 uses the UsePickSet flag to indicate that entities selected prior to the command being started can be accessed using the “SelectImplied” call.



Here is the Message Box shown from Command1.



Here is the Message Box shown from Command2.



Here is the Message Box shown from Command3. Of course, this is showing that 5 entities were selected prior to running this command.

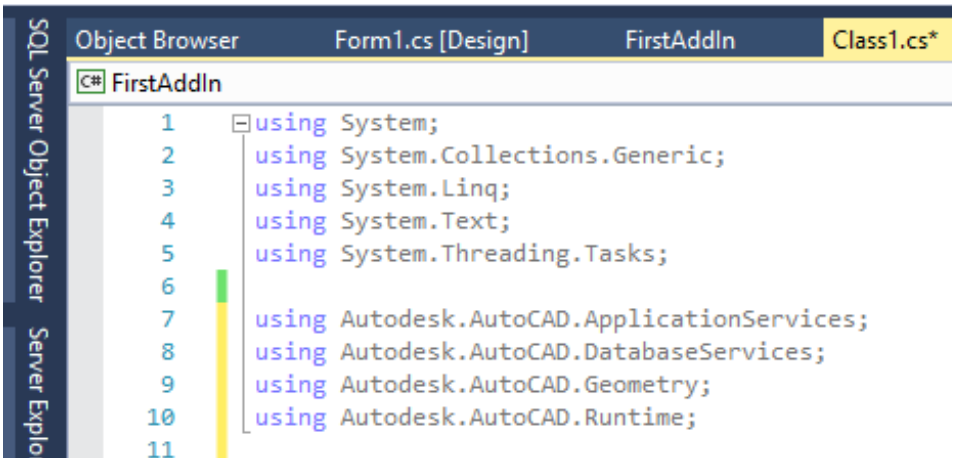
The basics of Drawing in AutoCAD through the .NET API

Drawing entities in AutoCAD using the .NET API is fairly straight forward. Let's take a look at an example.

```
[Autodesk.AutoCAD.Runtime.CommandMethod("Command4")]
public void Command4()
{
    Autodesk.AutoCAD.DatabaseServices.Database myDB;
    myDB = Autodesk.AutoCAD.DatabaseServices.HostApplicationServices.WorkingDatabase;
    using (Autodesk.AutoCAD.DatabaseServices.Transaction myTrans =
        myDB.TransactionManager.StartTransaction())
    {
        Autodesk.AutoCAD.Geometry.Point3d startPoint =
            new Autodesk.AutoCAD.Geometry.Point3d(1, 2, 3);
        Autodesk.AutoCAD.Geometry.Point3d endPoint =
            new Autodesk.AutoCAD.Geometry.Point3d(4, 5, 6);
        Autodesk.AutoCAD.DatabaseServices.Line myLine =
            new Autodesk.AutoCAD.DatabaseServices.Line(startPoint, endPoint);
        Autodesk.AutoCAD.DatabaseServices.BlockTableRecord myBTR =
            (Autodesk.AutoCAD.DatabaseServices.BlockTableRecord)
            myDB.CurrentSpaceId.GetObject
            (Autodesk.AutoCAD.DatabaseServices.OpenMode.ForWrite);
        myBTR.AppendEntity(myLine);
        myTrans.AddNewlyCreatedDBObject(myLine, true);
        myTrans.Commit();
    }
}
```

This code creates a new Line entity in the current space (ModelSpace or PaperSpace Layout) from (1, 2, 3) to (4, 5, 6).

The next thing we want to do is add a few lines of code at the very top of our "Class1.vb" file.



These Imports statements help us write more concise code. Let's copy and paste Command4 and rename it as Command5 and remove the portions of the code covered by the Imports statements.

```
[CommandMethod("Command5")]
public void Command5()
{
    Database myDB;
    myDB = HostApplicationServices.WorkingDatabase;
    using (Transaction myTrans = myDB.TransactionManager.StartTransaction())
    {
        Point3d startPoint = new Point3d(1, 2, 3);
        Point3d endPoint = new Point3d(4, 5, 6);
        Line myLine = new Line(startPoint, endPoint);
        BlockTableRecord myBTR = (BlockTableRecord)myDB.CurrentSpaceId.GetObject(OpenMode.ForWrite);
        myBTR.AppendEntity(myLine);
        myTrans.AddNewlyCreatedDBObject(myLine, true);
        myTrans.Commit();
    }
}
```

That looks much better. So much shorter and more concise.

Command4 and Command5 both do the same thing, draw a line from (1, 2, 3) to (4, 5, 6). How often do we need to do that? Let's Copy and Paste Command5 and rename it to Command6.

```
[CommandMethod("Command6")]
public void Command6()
{
    Database myDB;
    myDB = HostApplicationServices.WorkingDatabase;
    using (Transaction myTrans = myDB.TransactionManager.StartTransaction())
    {
        Autodesk.AutoCAD.EditorInput.Editor myEd =
            Application.DocumentManager.MdiActiveDocument.Editor;
        Point3d startPoint = myEd.GetPoint("First Point:").Value;
        Point3d endPoint = myEd.GetPoint("Second Point:").Value;
        Line myLine = new Line(startPoint, endPoint);
        BlockTableRecord myBTR = (BlockTableRecord)myDB.CurrentSpaceId.GetObject(OpenMode.ForWrite);
        myBTR.AppendEntity(myLine);
        myTrans.AddNewlyCreatedDBObject(myLine, true);
        myTrans.Commit();
    }
}
```

In this example we are adding 2 lines of code and modifying 2 lines of code, emphasized in Bold letting. Now instead of having hard -coded points, the user can select the points.

How to extract Block and Attribute information through the .NET API

We are going to create a couple of Functions that we will use in Command7 to get Block Information.

The first function is named "GetBlockNames". If you give it a Database (AutoCAD DWG Database), it will give you a list of Block Names.

```
List<string> GetBlockNames(Database DBIn)
{
    List<string> retList = new List<string>();
    using (Transaction myTrans = DBIn.TransactionManager.StartTransaction())
    {
        BlockTable myBT = (BlockTable)DBIn.BlockTableId.GetObject(OpenMode.ForRead);
        foreach (ObjectId myOID in myBT)
        {
            BlockTableRecord myBTR = (BlockTableRecord)myOID.GetObject(OpenMode.ForRead);
            if (myBTR.IsLayout == false | myBTR.IsAnonymous == false)
            {
                retList.Add(myBTR.Name);
            }
        }
    }
    return (retList);
}
```

```
ObjectIdCollection GetBlockIDs(Database DBIn, string BlockName)
{
    ObjectIdCollection retCollection = new ObjectIdCollection();
    using (Transaction myTrans = DBIn.TransactionManager.StartTransaction())
    {
        BlockTable myBT = (BlockTable)DBIn.BlockTableId.GetObject(OpenMode.ForRead);
        if (myBT.Has(BlockName))
        {
            BlockTableRecord myBTR = (BlockTableRecord)myBT[BlockName].GetObject(OpenMode.ForRead);
            retCollection = (ObjectIdCollection)myBTR.GetBlockReferenceIds(true, true);
            myTrans.Commit();
            return(retCollection);
        }
        else
        {
            myTrans.Commit();
            return(retCollection);
        }
    }
}
```

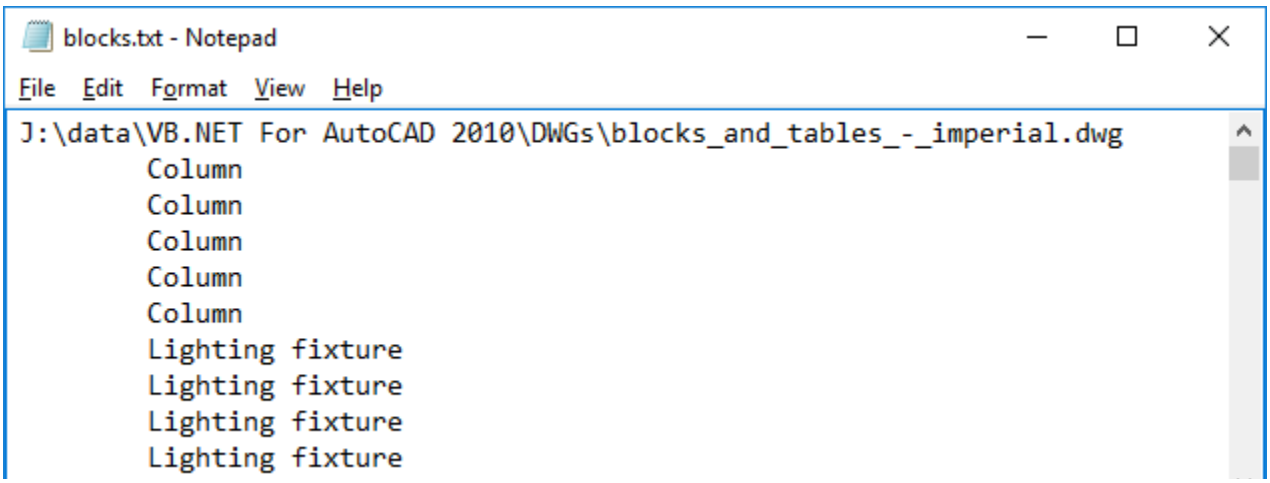
GetBlockIDs gives us ObjectIDs of all Blocks of a given name in a given database.

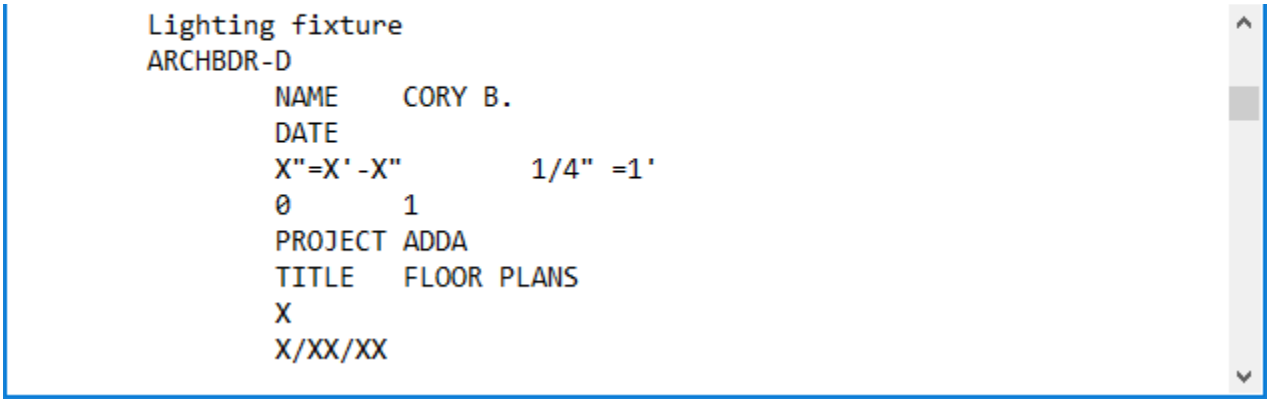
```
Dictionary<string, string> GetAttributes(ObjectId BlockRefID)
{
    Dictionary<string, string> retDictionary = new Dictionary<string, string>();
    using (Transaction myTrans = BlockRefID.Database.TransactionManager.StartTransaction())
    {
        BlockReference myBref = (BlockReference)BlockRefID.GetObject(OpenMode.ForRead);
        if (myBref.AttributeCollection.Count == 0)
        {
            return (retDictionary);
        }
        else
        {
            foreach (ObjectId myBRefID in myBref.AttributeCollection)
            {
               AttributeReference myAttRef = (AttributeReference)myBRefID.GetObject(OpenMode.ForRead);
               if (retDictionary.ContainsKey(myAttRef.Tag) == false)
               {
                   retDictionary.Add(myAttRef.Tag, myAttRef.TextString);
               }
            }
            return (retDictionary);
        }
    }
}
```

GetAttribtues gives us the attribute Tags and Values of a given Block Reference based on its ObjectID.

Now, these Functions do not run by themselves. They need to be called by another piece of code. In our example, we will do this from a new AutoCAD Command named “Command7”.

```
[CommandMethod("Command7")]
public void Command7()
{
    System.IO.FileInfo myFIO = new System.IO.FileInfo("C:\\temp\\blocks.txt");
    if (myFIO.Directory.Exists == false)
    {
        myFIO.Directory.Create();
    }
    Database dbToUse = HostApplicationServices.WorkingDatabase;
    System.IO.StreamWriter mySW = new System.IO.StreamWriter(myFIO.FullName);
    mySW.WriteLine(HostApplicationServices.WorkingDatabase.Filename);
    foreach (string myName in GetBlockNames(dbToUse))
    {
        foreach (ObjectId myBrefID in GetBlockIDs(dbToUse, myName))
        {
            mySW.WriteLine(" " + myName);
            foreach (KeyValuePair<string, string> myKVP in GetAttributes(myBrefID))
            {
                mySW.WriteLine("    " + myKVP.Key + "  " + myKVP.Value);
            }
        }
    }
    mySW.Close();
    mySW.Dispose();
}
```





REVIEW

Creating an AutoCAD Add-In using C#? You can do that.

Creating new AutoCAD Commands in C#? You can do that.

Drawing in AutoCAD? You can do that.

Extracting Block information? You can do that.

Of course there's much more to learn, much more to do, but hopefully this will give you a good start on creating your own Add-Ins to meet your own specific needs.

I hope your AU experience has been a good one.

Take care, my friend.

Jerry Winters

jerryw@vbcad.com