AUTODESK UNIVERSITY

SD22321-L

Hands-on Introduction to VB.NET Add-ins for AutoCAD

Jerry Winters
VB CAD, Inc.

Learning Objectives

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

Description

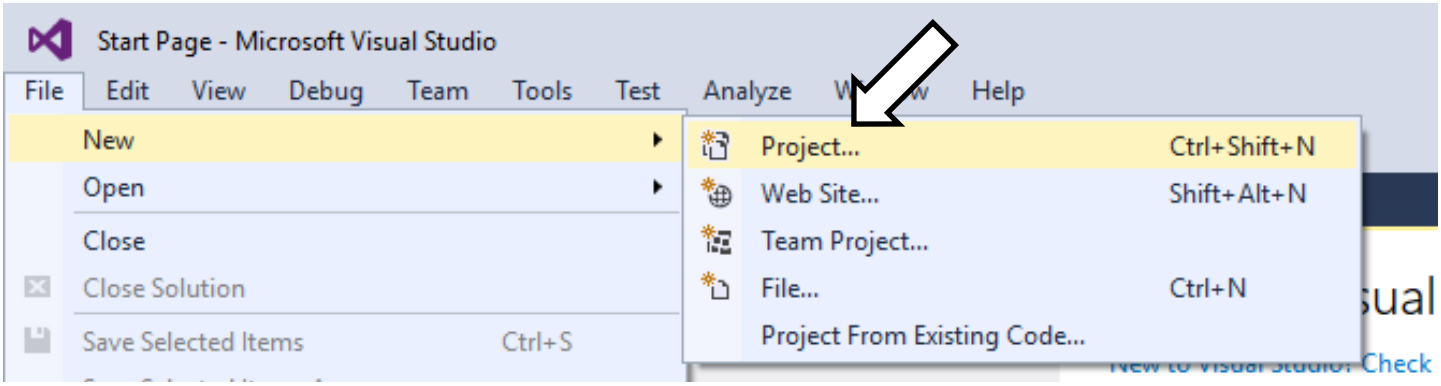
Knowing how to write AutoCAD software add-ins enables us to create new AutoCAD software commands that automate routine tasks, perform complex calculations, or integrate multiple systems to help us be more productive, more precise, and more effective. The use of Microsoft's community edition of Visual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

Your AU Expert

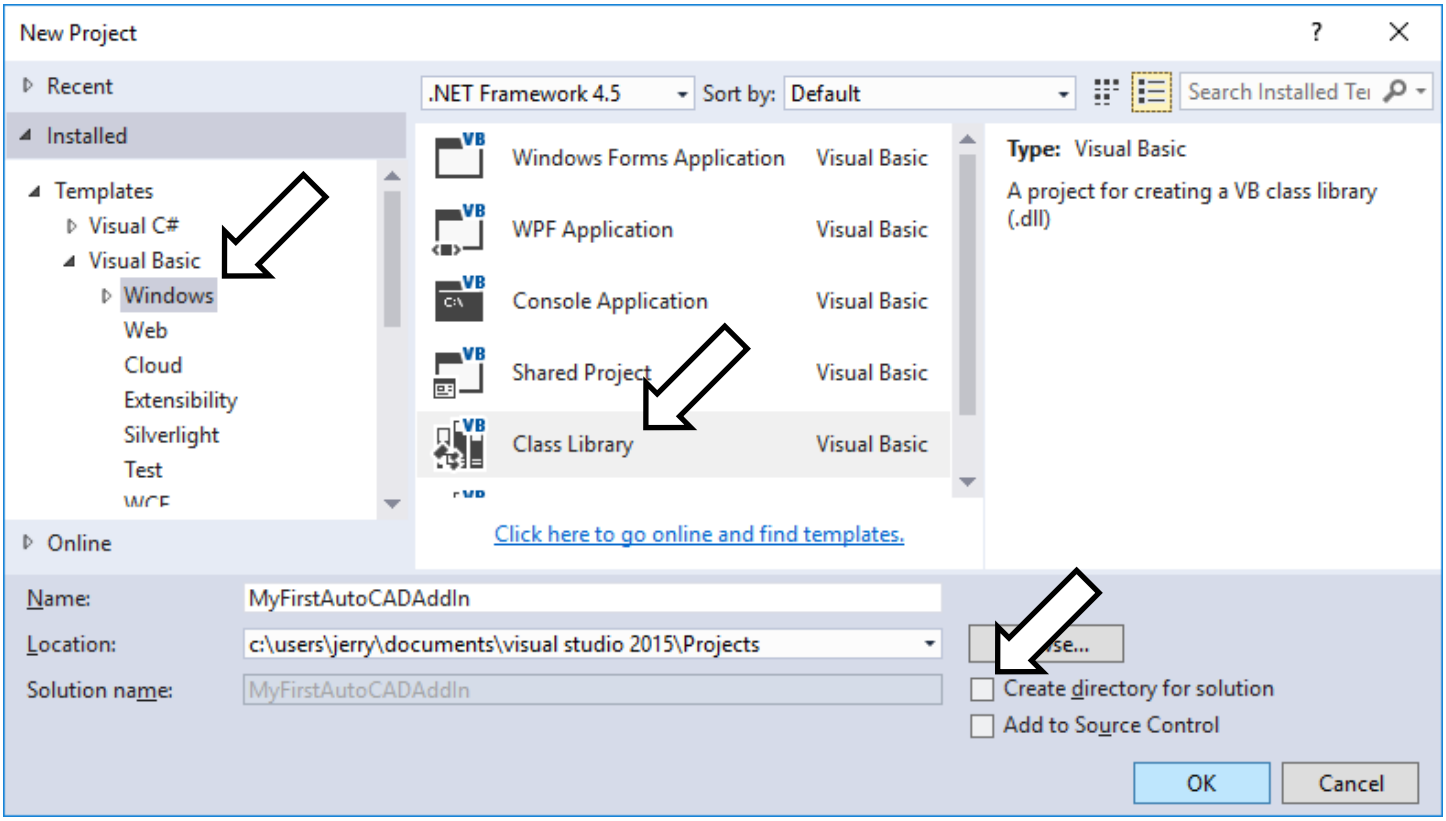
Jerry Winters has been writing add-ins for AutoCAD software since 1992, and he lends his experience to developing powerful, useful add-ins to improve accuracy, increase efficiency, and eliminate user error. Although technology continues to advance at a rapid pace, Winters strives to be aware of and proficient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

Creating an Add-In using Visual Studio Community 2015 Visual Basic

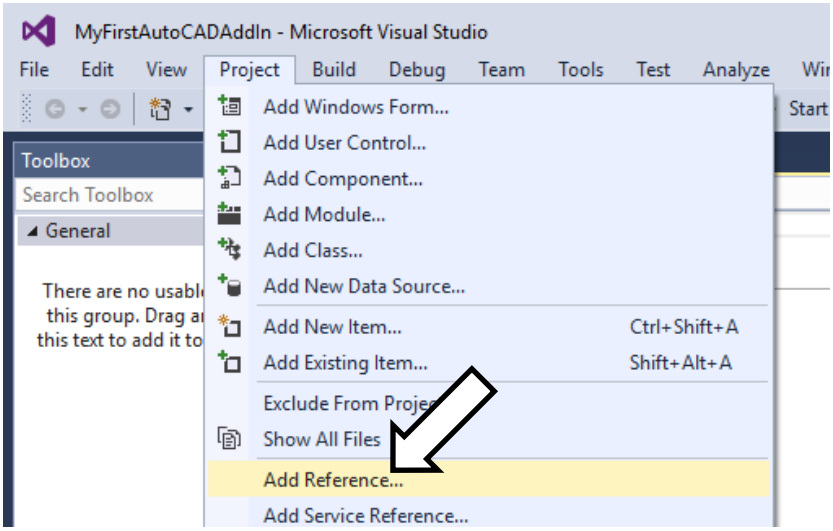
Step 1: Using the Visual Studio Menu, go to ***File -> New -> Project***.

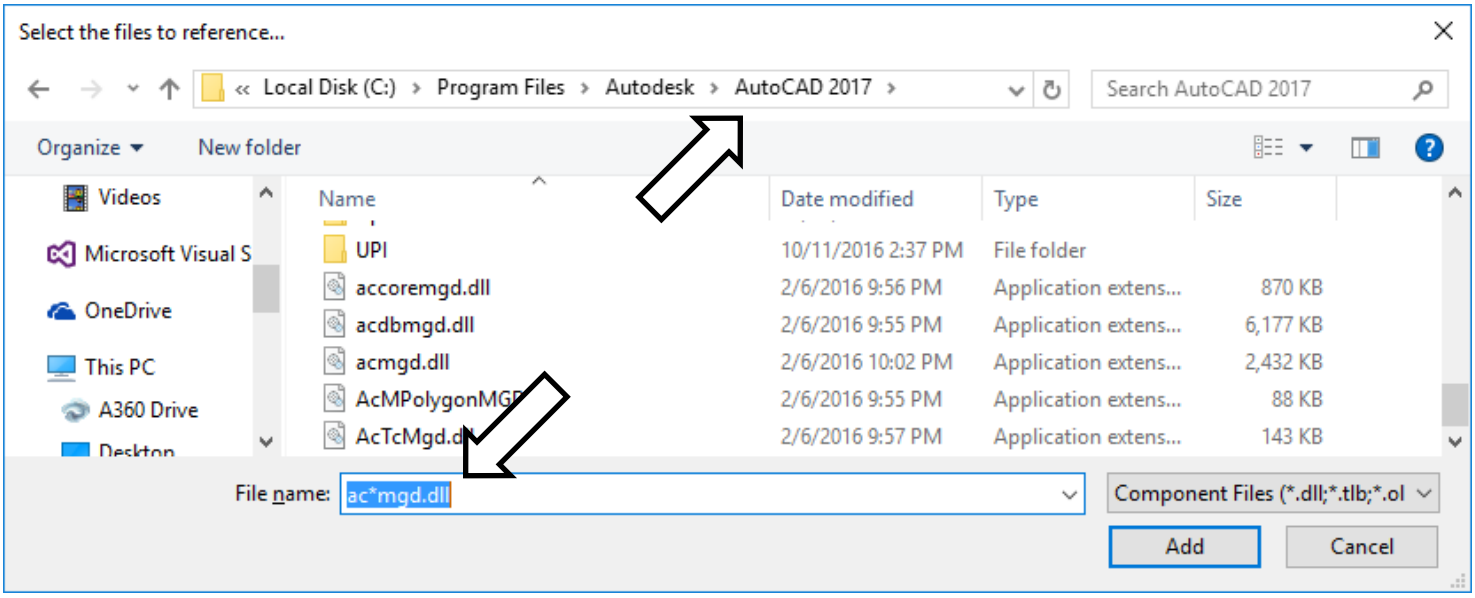


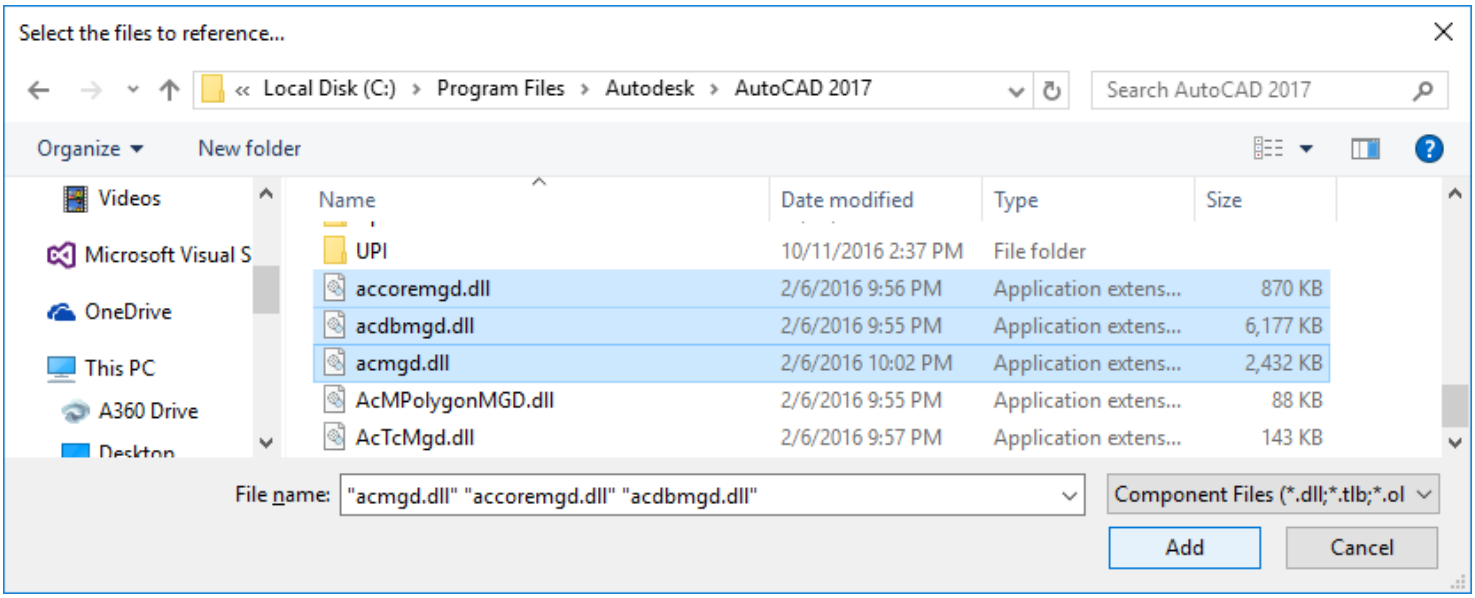
Step 2: Select the "Class Library" Template under the Visual Basic\Windows Template Type and give your Project a Name. In this example, the name is "MyFirstAutoCADAddIn". Before you click the "OK" button, make sure the "Create directory for solution" checkbox is NOT checked.



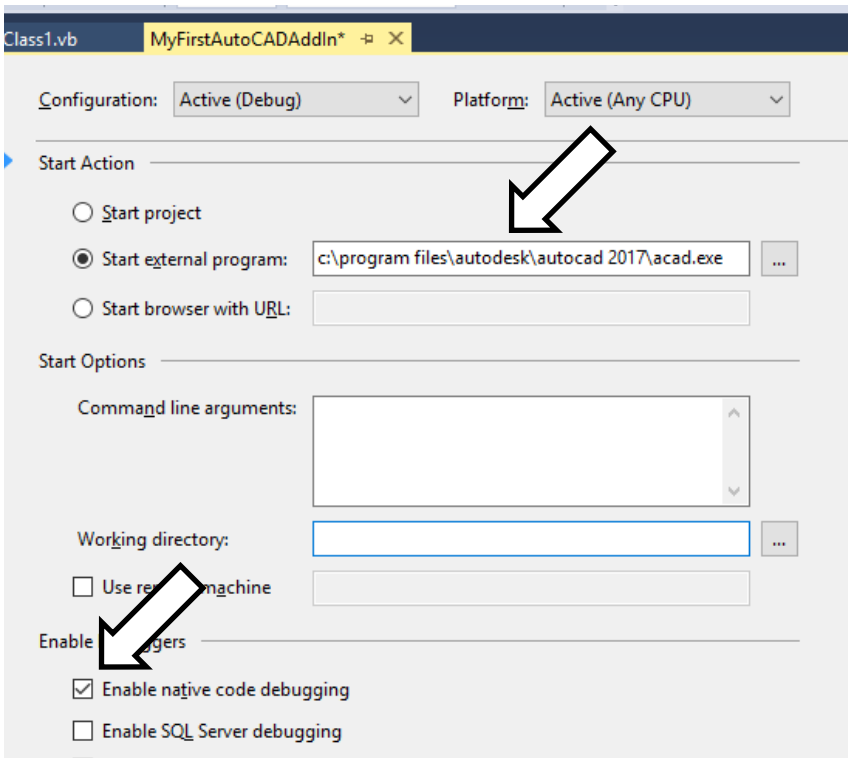
Step 3: Add References to the files “accoremgd.dll”, “acdbmgd.dll”, and “acmgd.dll” by going to the Visual Studio Menu: **Project -> Add Reference**.



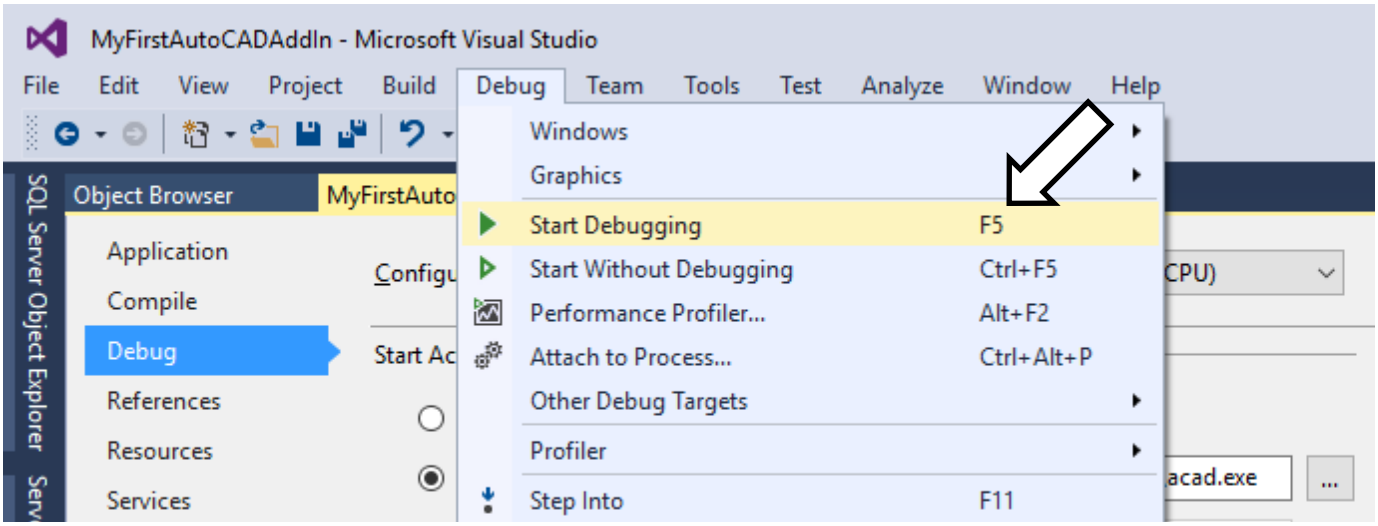




Step 4: Tell Visual Studio to start AutoCAD for debugging purposes by going to the Debug Tab in the Project Properties. While you are here, check the box labeled "Enable native code debugging".

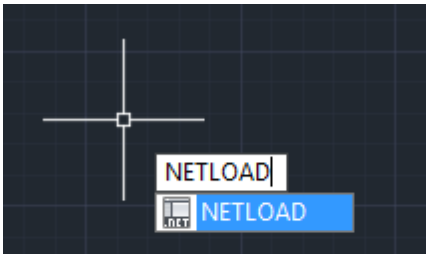


Step 5: Begin Debugging by going to the Menu:
Debug -> Start Debugging



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

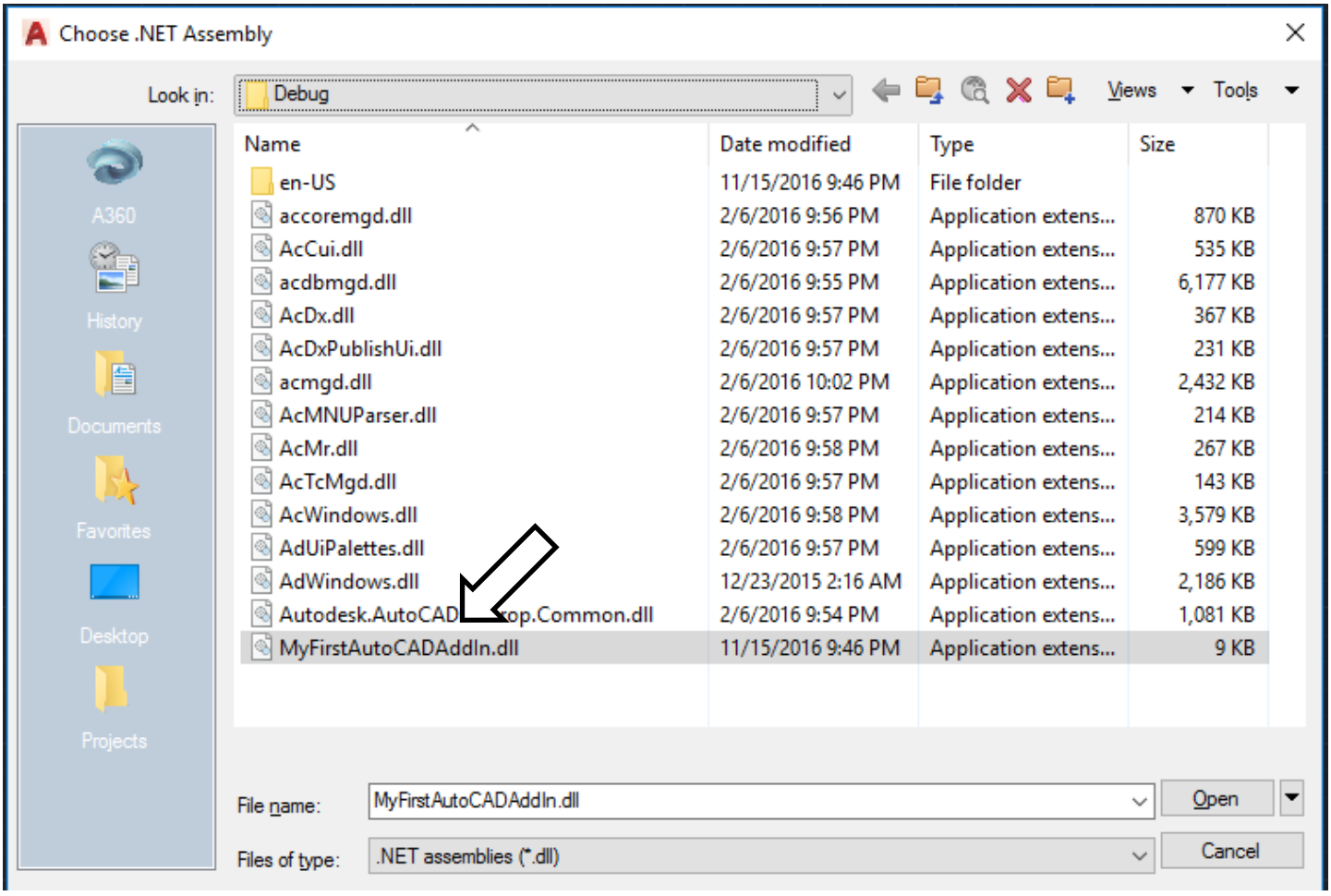
Step 7: "Netload" the DLL

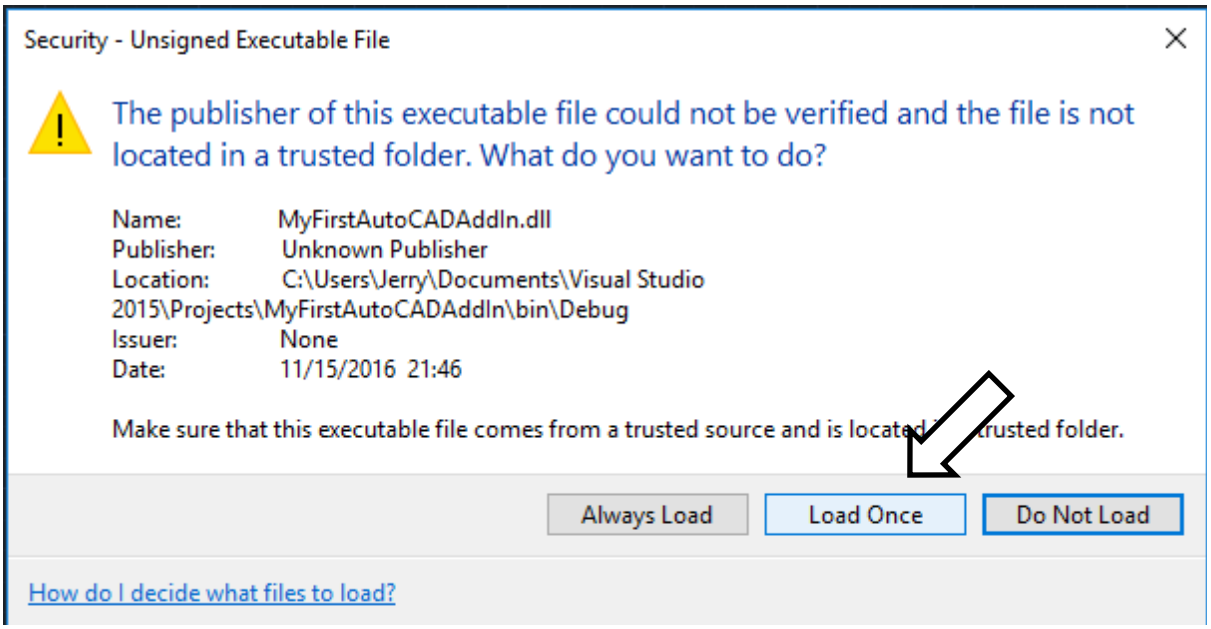


The DLL should be in the path:

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

Click on “Load Once” in the subsequent dialog box.





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 VB.NET.

Create a new AutoCAD Command in VB.NET

Creating a Command in VB.NET is as simple as this:

```
Public Class Class1
    <Autodesk.AutoCAD.Runtime.CommandMethod("Command1")>
    Public Sub Command1()
        MsgBox("Whatever you do, don't type Hello World.")
    End Sub
End Class
```

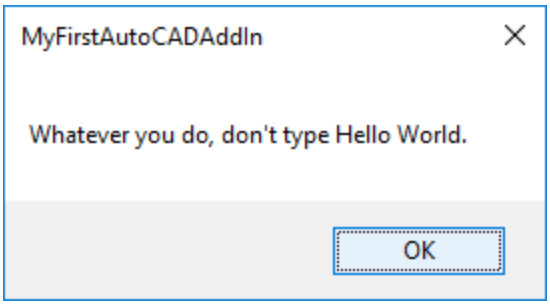
There you have it. The name of the command is "Command1". Let's take a look at another one. Please note this must be placed inside the Class1 Class.

```
    <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
```

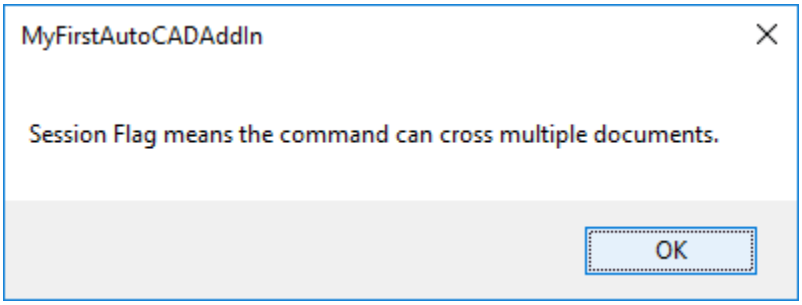
Command2 uses the Session flag which tells AutoCAD the command can cross multiple documents if this is needed.

```
    <Autodesk.AutoCAD.Runtime.CommandMethod("Command3",
                  Autodesk.AutoCAD.Runtime.CommandFlags.UsePickSet)>
    Public Sub Command3()
        Dim myDC As Autodesk.AutoCAD.ApplicationServices.DocumentCollection
        myDC = Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager
        Dim myDoc As Autodesk.AutoCAD.ApplicationServices.Document
        myDoc = myDC.CurrentDocument
        Dim myEd As Autodesk.AutoCAD.EditorInput.Editor = myDoc.Editor
        Dim myPSR As Autodesk.AutoCAD.EditorInput.PromptSelectionResult
        myPSR = myEd.SelectImplied
        If myPSR.Status = Autodesk.AutoCAD.EditorInput.PromptStatus.OK Then
            MsgBox(myPSR.Value.Count & " selected.")
        Else
            MsgBox("0 selected.")
        End If
    End Sub
```

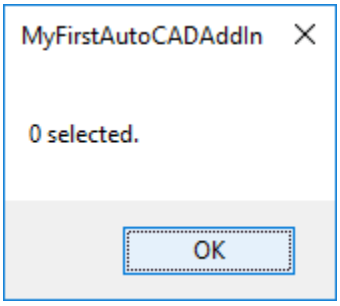
Command3 uses the UsePickSet flag to indicate that entities selected prior to the command being started can be accessed using the "SelectImplied" call.



Here is the Message Box shown from Command1.



Here is the Message Box shown from Command2.



Here is the Message Box shown from Command3.

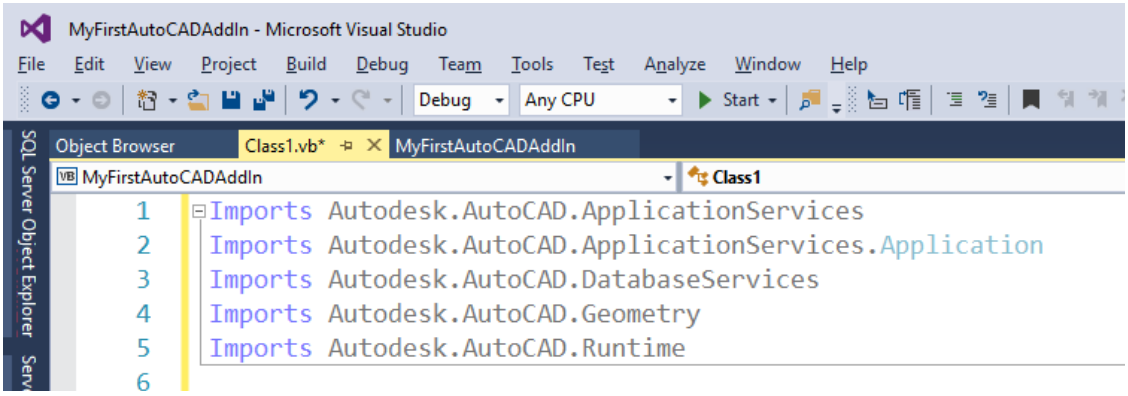
The basics of Drawing in AutoCAD through the .NET API

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 Sub Command4()
    Dim myDB As Autodesk.AutoCAD.DatabaseServices.Database
    myDB = Autodesk.AutoCAD.DatabaseServices.HostApplicationServices.WorkingDatabase
    Using myTrans As Autodesk.AutoCAD.DatabaseServices.Transaction =
                myDB.TransactionManager.StartTransaction
        Dim startPoint As New Autodesk.AutoCAD.Geometry.Point3d(1, 2, 3)
        Dim endPoint As New Autodesk.AutoCAD.Geometry.Point3d(4, 5, 6)
        Dim myLine As New Autodesk.AutoCAD.DatabaseServices.Line(startPoint, endPoint)
        Dim myBTR As Autodesk.AutoCAD.DatabaseServices.BlockTableRecord
        myBTR = myDB.CurrentSpaceId.GetObject(Autodesk.AutoCAD.DatabaseServices.OpenMode.ForWrite)
        myBTR.AppendEntity(myLine)
        myTrans.AddNewlyCreatedDBObject(myLine, True)
        myTrans.Commit()
    End Using
End Sub
```

This code creates a new Line entity in the current space (ModelSpace or PaperSpace Layout) from (0, 0, 0) 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 Sub Command5()
    Dim myDB As Database
    myDB = HostApplicationServices.WorkingDatabase
    Using myTrans As Transaction =
                    myDB.TransactionManager.StartTransaction
        Dim startPoint As New Point3d(1, 2, 3)
        Dim endPoint As New Point3d(4, 5, 6)
        Dim myLine As New Line(startPoint, endPoint)
        Dim myBTR As BlockTableRecord
        myBTR = myDB.CurrentSpaceId.GetObject(OpenMode.ForWrite)
        myBTR.AppendEntity(myLine)
        myTrans.AddNewlyCreatedDBObject(myLine, True)
        myTrans.Commit()
    End Using
End Sub
```

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 Sub Command6()
    Dim myDB As Database
    myDB = HostApplicationServices.WorkingDatabase
    Dim myDoc As Document = DocumentManager.MdiActiveDocument
    Dim myEd As Autodesk.AutoCAD.EditorInput.Editor = myDoc.Editor
    Using myTrans As Transaction =
                    myDB.TransactionManager.StartTransaction
        Dim startPoint As Point3d = myEd.GetPoint("Point 1:").Value
        Dim endPoint As Point3d = myEd.GetPoint("Point 2:").Value
        Dim myLine As New Line(startPoint, endPoint)
        Dim myBTR As BlockTableRecord
        myBTR = myDB.CurrentSpaceId.GetObject(OpenMode.ForWrite)
        myBTR.AppendEntity(myLine)
        myTrans.AddNewlyCreatedDBObject(myLine, True)
        myTrans.Commit()
    End Using
End Sub
```

In this example we are adding 4 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.

```
Function GetBlockNames(DBIn As Database) As List(Of String)
    Dim retList As New List(Of String)
    Using myTrans As Transaction = DBIn.TransactionManager.StartTransaction
        Dim myBT As BlockTable = DBIn.BlockTableId.GetObject(OpenMode.ForRead)
        For Each myOID As ObjectId In myBT
            Dim myBTR As BlockTableRecord = myOID.GetObject(OpenMode.ForRead)
            If myBTR.IsLayout = False And myBTR.IsAnonymous = False Then
                retList.Add(myBTR.Name)
            End If
        Next
        Return retList
    End Using
End Function
```

```
Function GetBlockIDs(DBIn As Database, BlockName As String) As ObjectIdCollection
    Dim retCollection As New ObjectIdCollection
    Using myTrans As Transaction = DBIn.TransactionManager.StartTransaction
        Dim myBT As BlockTable = DBIn.BlockTableId.GetObject(OpenMode.ForRead)
        If myBT.Has(BlockName) Then
            Dim myBTR As BlockTableRecord = myBT(BlockName).GetObject(OpenMode.ForRead)
            retCollection = myBTR.GetBlockReferenceIds(True, True)
            myTrans.Commit()
            Return retCollection
        Else
            myTrans.Commit()
            Return retCollection
        End If
    End Using
End Function
```

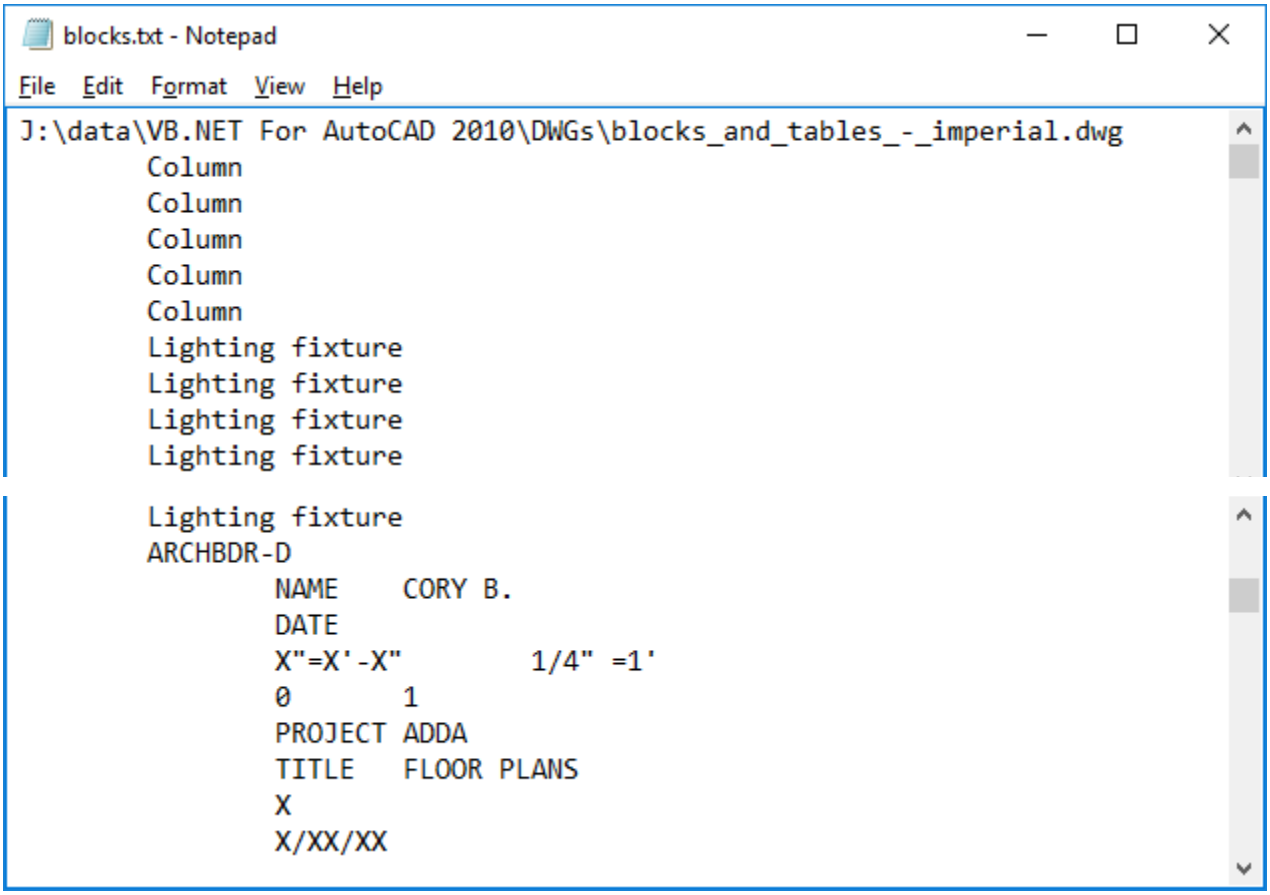
GetBlockIDs gives us ObjectIDs of all Blocks of a given name in a given database.

```
Function GetAttributes(BlockRefID As ObjectId) As Dictionary(Of String, String)
    Dim retDictionary As New Dictionary(Of String, String)
    Using myTrans As Transaction = BlockRefID.Database.TransactionManager.StartTransaction
        Dim myBref As BlockReference = BlockRefID.GetObject(OpenMode.ForRead)
        If myBref.AttributeCollection.Count = 0 Then
            Return retDictionary
        Else
            For Each myBRefID As ObjectId In myBref.AttributeCollection
                Dim myAttRef As AttributeReference = myBRefID.GetObject(OpenMode.ForRead)
                If retDictionary.ContainsKey(myAttRef.Tag) = False Then
                    retDictionary.Add(myAttRef.Tag, myAttRef.TextString)
                End If
            Next
            Return retDictionary
        End If
    End Using
End Function
```

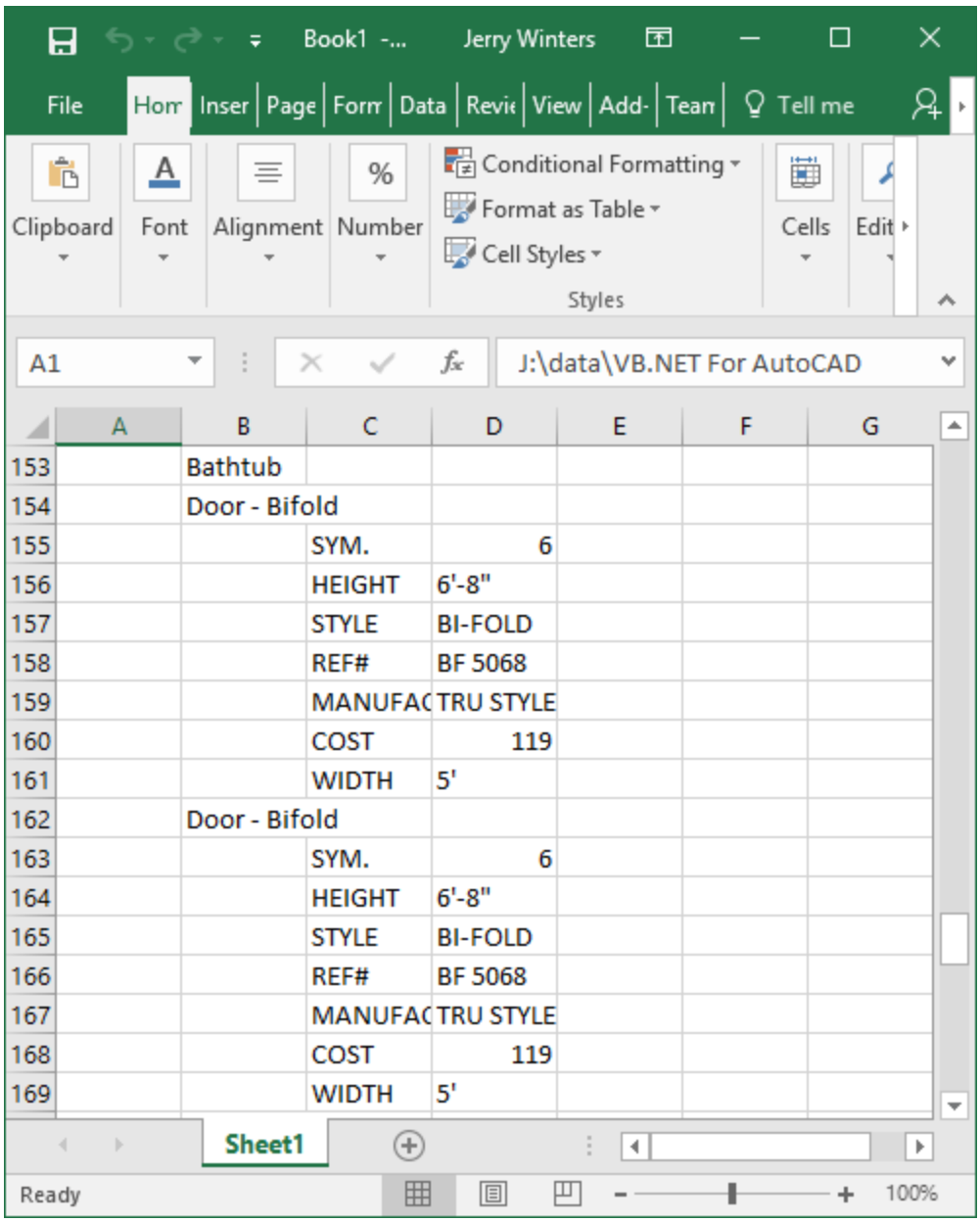
GetAttribtues gives us the attribute Tags and Values of a given Block Reference based on its ObjectID.

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 Sub Command7()
    Dim myFIO As New IO.FileInfo("C:\temp\blocks.txt")
    If myFIO.Directory.Exists = False Then
        myFIO.Create()
    End If
    Dim dbToUse As Database = HostApplicationServices.WorkingDatabase
    Dim mySW As New IO.StreamWriter(myFIO.FullName)
    mySW.WriteLine(HostApplicationServices.WorkingDatabase.Filename)
    For Each myName As String In GetBlockNames(dbToUse)
        For Each myBrefID As ObjectId In GetBlockIDs(dbToUse, myName)
            mySW.WriteLine(vbTab & myName)
            For Each myKVP As KeyValuePair(Of String, String) In GetAttributes(myBrefID)
                mySW.WriteLine(vbTab & vbTab & myKVP.Key & vbTab & myKVP.Value)
            Next
        Next
    Next
    mySW.Close()
    mySW.Dispose()
End Sub
```



```
<CommandMethod("Command8")>
Public Sub Command8()
    Dim myExcel As Object = CreateObject("Excel.Application")
    myExcel.Visible = True
    Dim myWB As Object = myExcel.Workbooks.Add()
    Dim myWS As Object = myExcel.ActiveSheet
    Dim dbToUse As Database = HostApplicationServices.WorkingDatabase
    Dim curRow As Integer = 1
    myWS.cells(curRow, "A").value = HostApplicationServices.WorkingDatabase.Filename
    curRow += 1
    For Each myName As String In GetBlockNames(dbToUse)
        For Each myBrefID As ObjectId In GetBlockIDs(dbToUse, myName)
            myWS.cells(curRow, "B").value = myName
            curRow += 1
            For Each myKVP As KeyValuePair(Of String, String) In GetAttributes(myBrefID)
                myWS.cells(curRow, "C").value = myKVP.Key
                myWS.cells(curRow, "D").value = myKVP.Value
                curRow += 1
            Next
        Next
    Next
    myWS = Nothing
    myWB = Nothing
    myExcel = Nothing
End Sub
```



REVIEW

Creating an AutoCAD Add-In using VB.NET? You can do that.

Creating new AutoCAD Commands in VB.NET? 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