



Hands-on Introduction to VB.NET Add-Ins for AutoCAD®

Jerry Winters – VB CAD, Inc.

SD10673-L Creating an add-in for AutoCAD software using Microsoft Visual Basic .NET is a simple and straightforward task when you know the steps to follow. This hands-on class will teach you exactly what you need to know and only what you need to know to begin writing your own add-ins for AutoCAD software. We will begin by discussing and following the steps to creating an add-in. After taking some time to practice that skill, we will see how easy it is to take some existing code and modify it to meet specific requirements. You won't believe how much you will learn in 90 minutes. No prior programming experience is required.

Learning Objectives

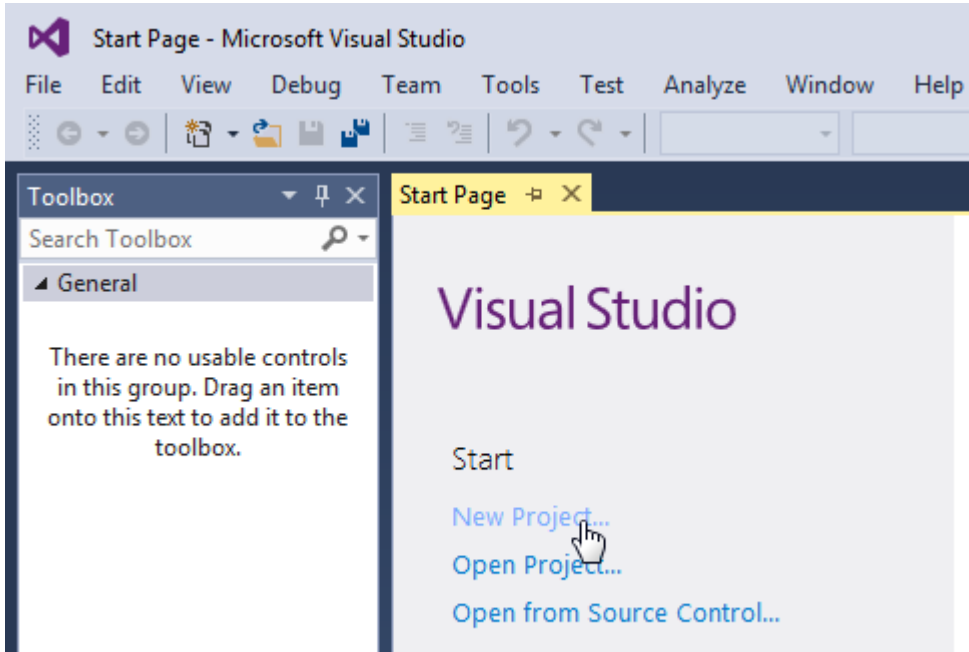
- Learn how to create a new Class Library Project in Visual Studio
- Learn how to reference the Essential Autodesk DLL files to create an add-in
- Learn how to compile and load an add-in into AutoCAD
- Learn how to modify existing code to meet specific requirements

About the Speaker

Jerry Winters has taught thousands at Autodesk University, mixing real-world practical examples with engaging humor. His "I'm just a drafter" approach to API topics make his classes as enjoyable as they are applicable.
jerryw@vbcad.com

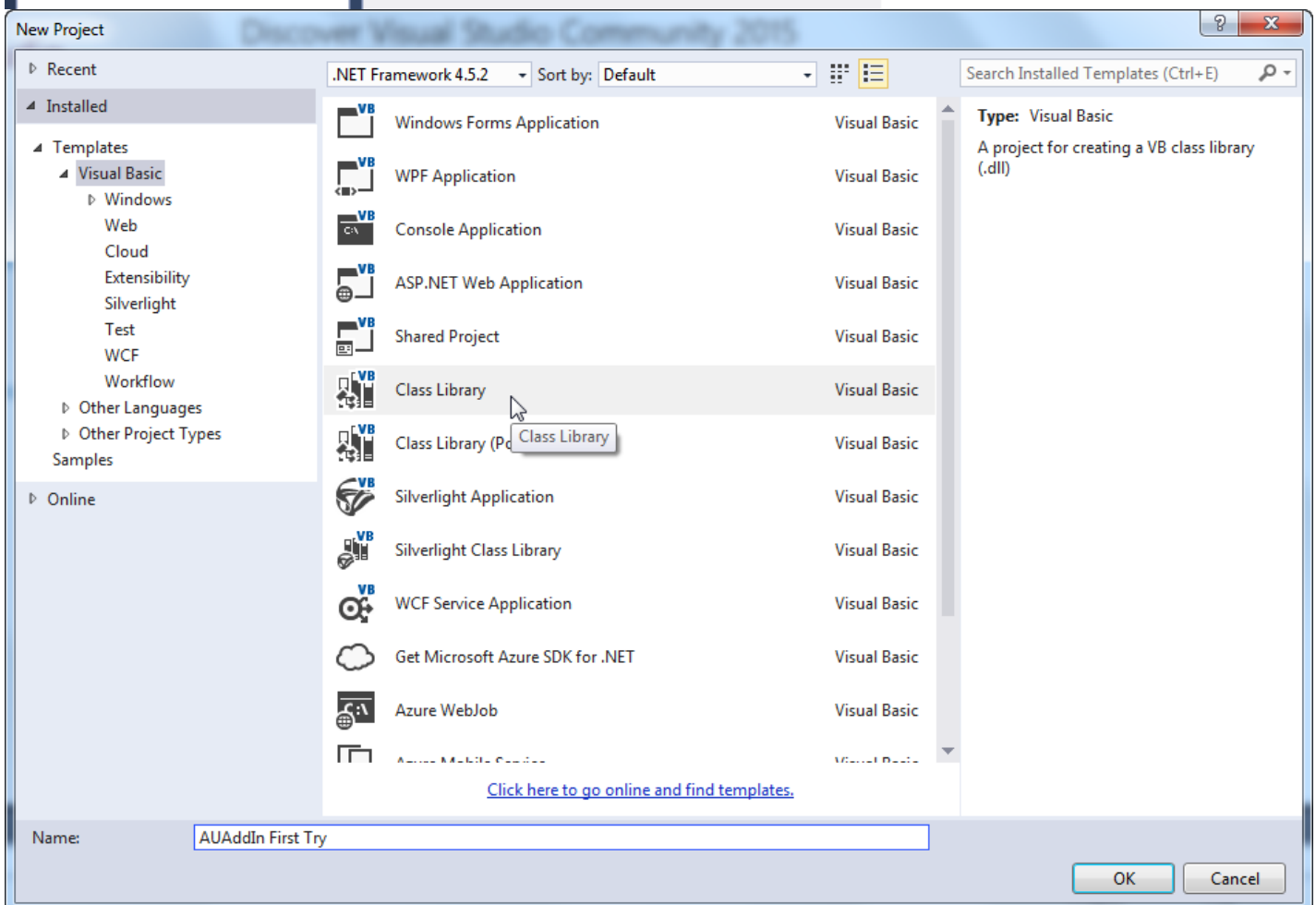


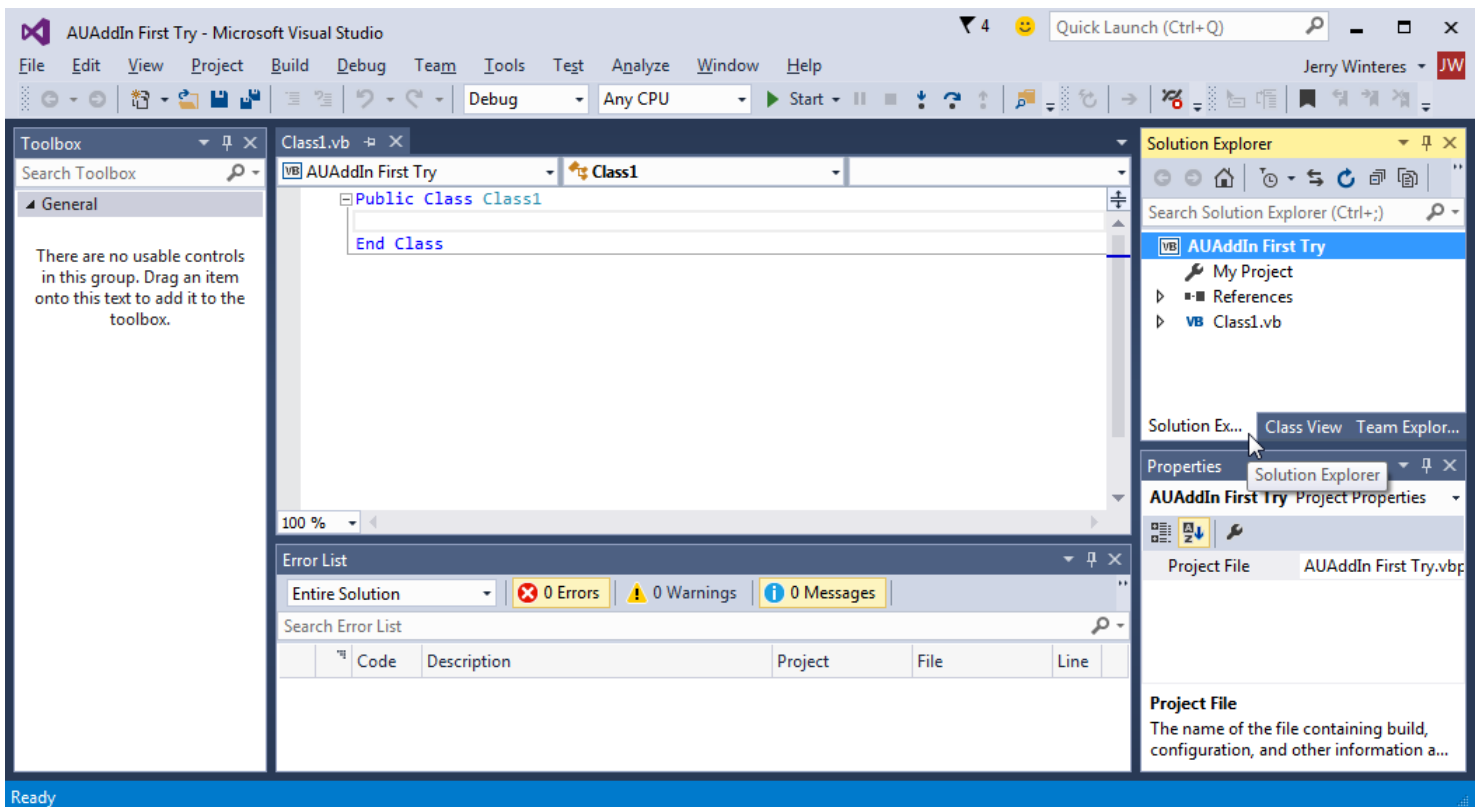
Create a New Class Library Project in Visual Studio



Creating a new VB.NET Project from scratch is easy to do. It requires a few basic steps.

1. Create a New Project by clicking "New Project" on the Visual Studio Start Page.
2. Select "Class Library" under the Visual Basic Templates list.
3. Give your project a name. In this case the name should be "AUAddIn First Try".
4. Click the OK button.

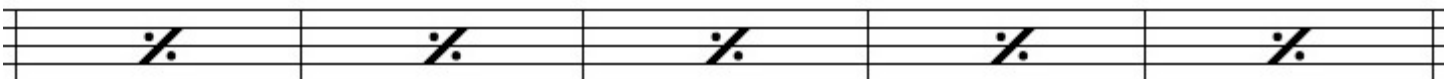




Congratulations. You now have a new, completely blank Class Library Project. Let's take a look at the interface for a moment.

The Visual Studio interface is divided into a number of different areas.

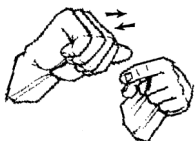
- Toolbox—Typically shows controls to be added to a Form but can also be used to store code to be easily and quickly inserted into our projects.
- Class / Module / Form (Design) Tab area—We create and modify our code and Forms in this area.
- Error List—Your best friend, your worst enema (spelling error intentional).
- Solution Explorer—Shows all of the files utilized in your project.
- Properties—Properties to show and/or edit of Forms, Controls, Classes, and other files in the project.



For those who are musically inclined, the symbols above may mean something to you. If these symbols do not mean anything to you, it simply means "REPEAT". Why "REPEAT"? Because if we cover all of the material in this class and we go back to the office next week and are unable to create a new blank Add-In for AutoCAD, our time will have been wasted.

For more information on Repeat, please watch "Monk and the Airplane", Season 1 Episode 12.

So, please repeat the process of creating a new Project several times. 4 or 5 would be good. With this short practice session, a new project should be able to be created within one minute.

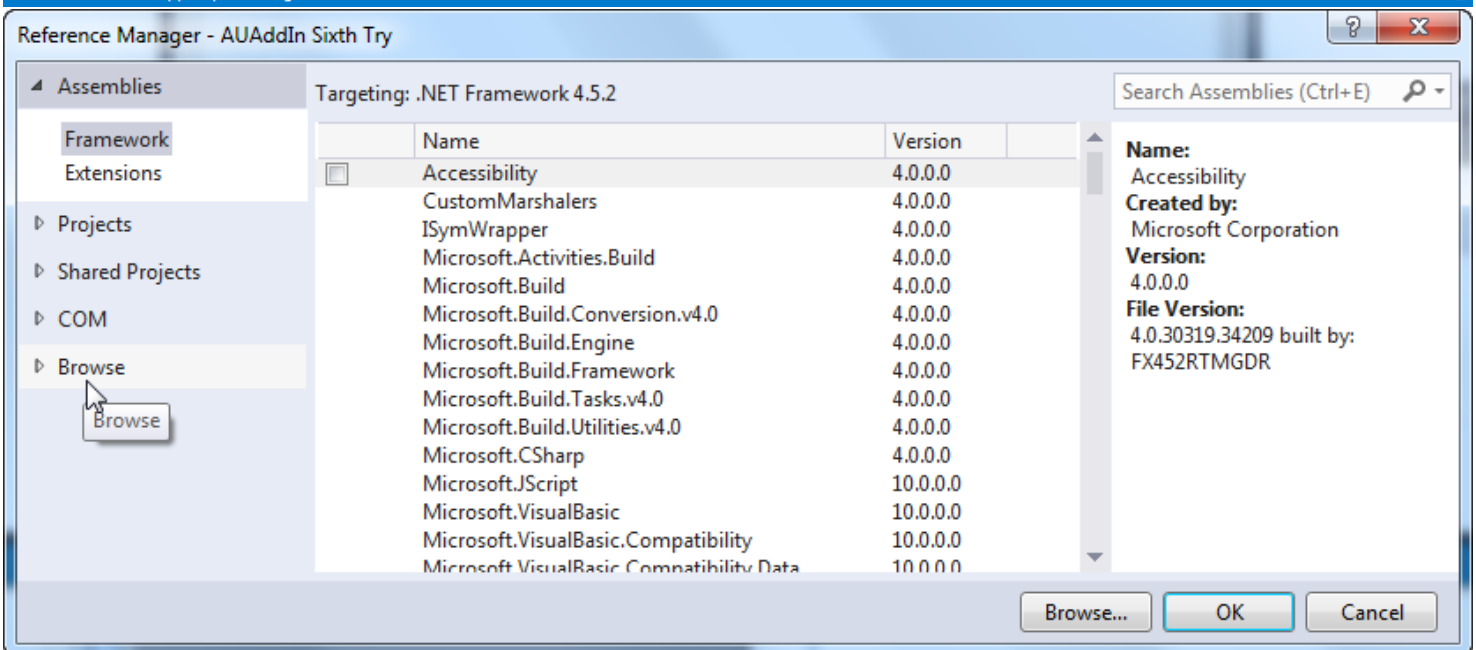
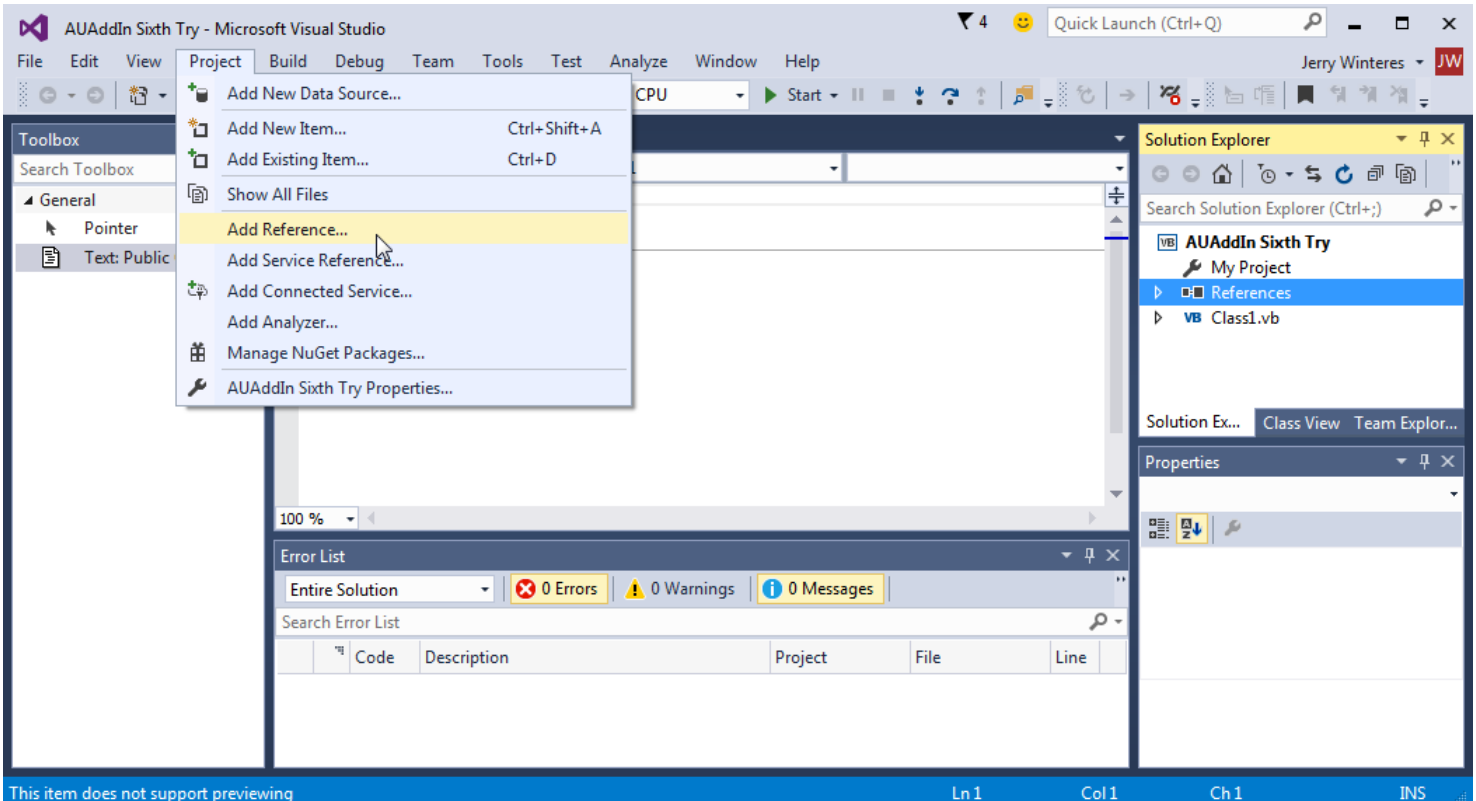


PRACTICE: Create a new Class Library Project 5 times.

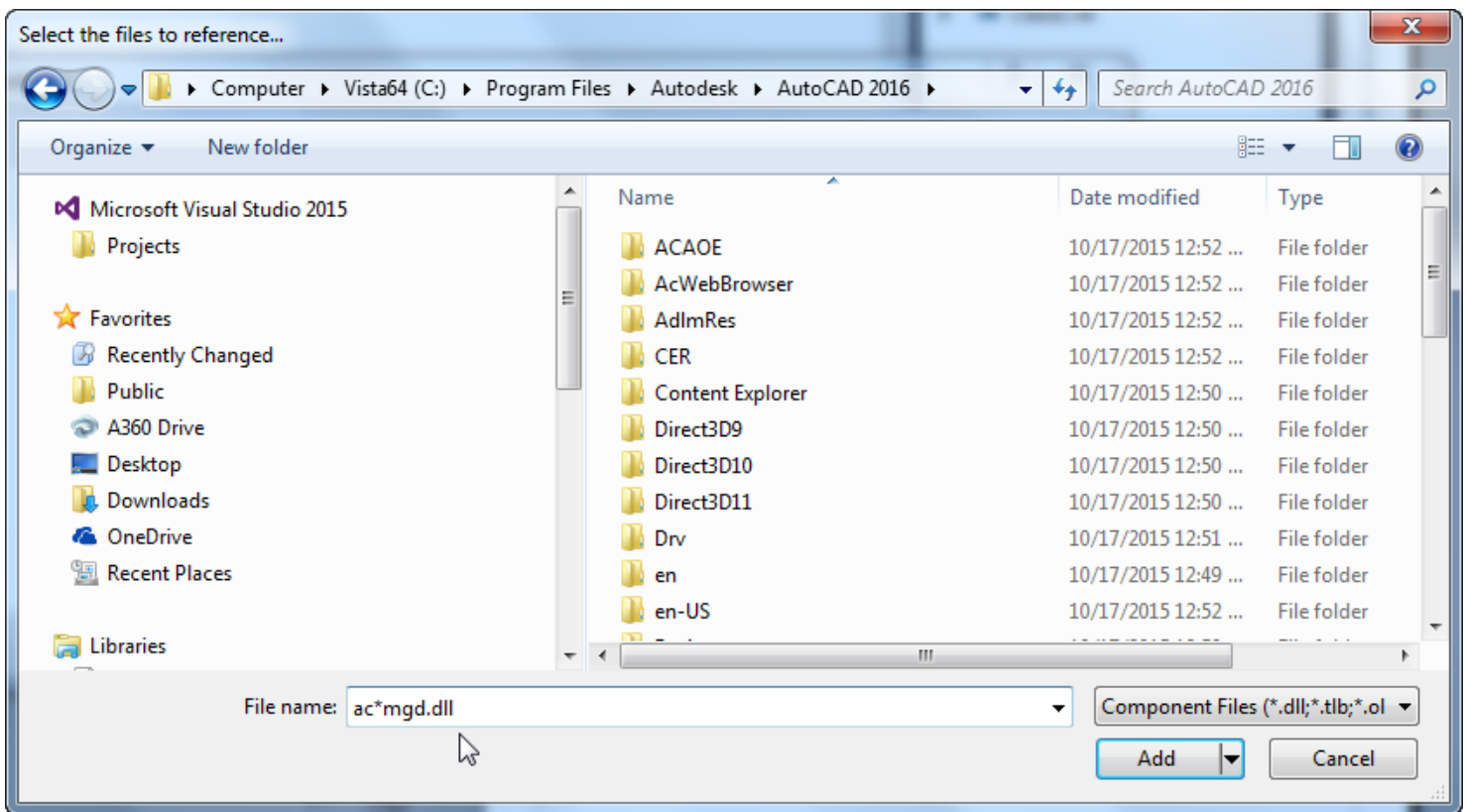
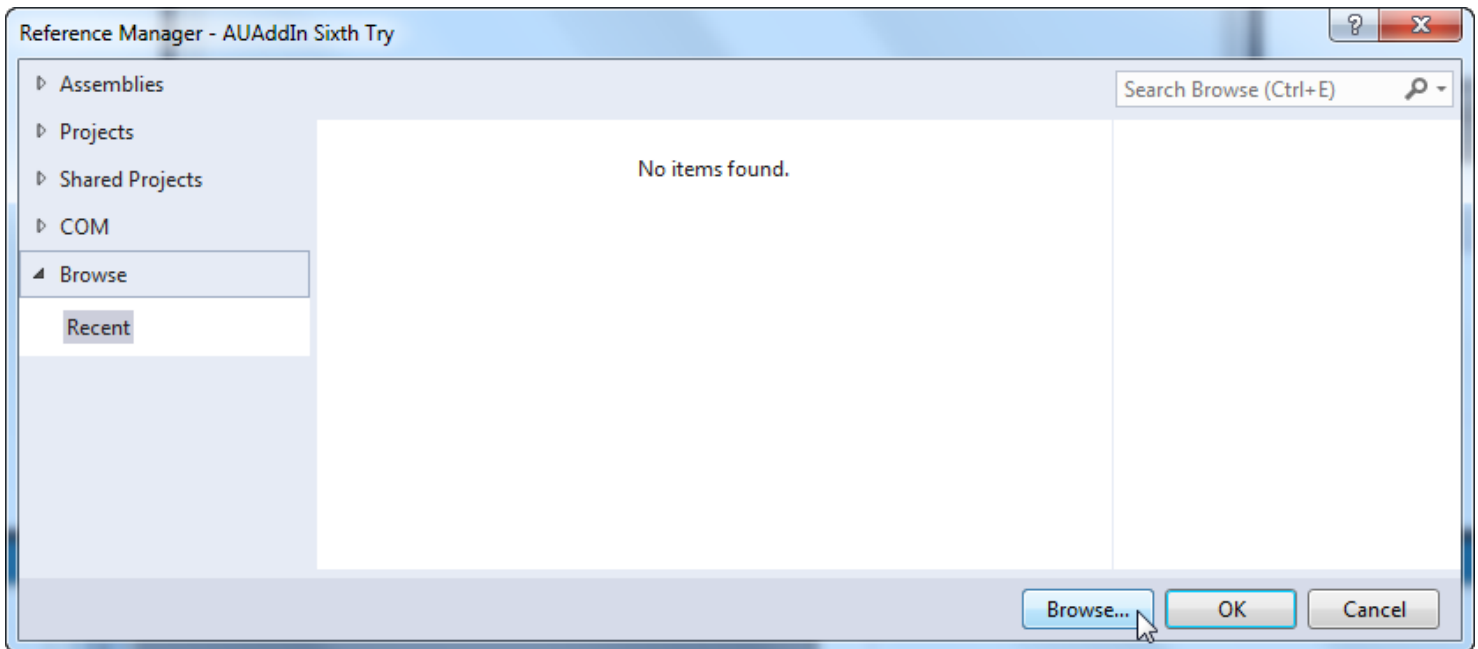


Add the Essential References

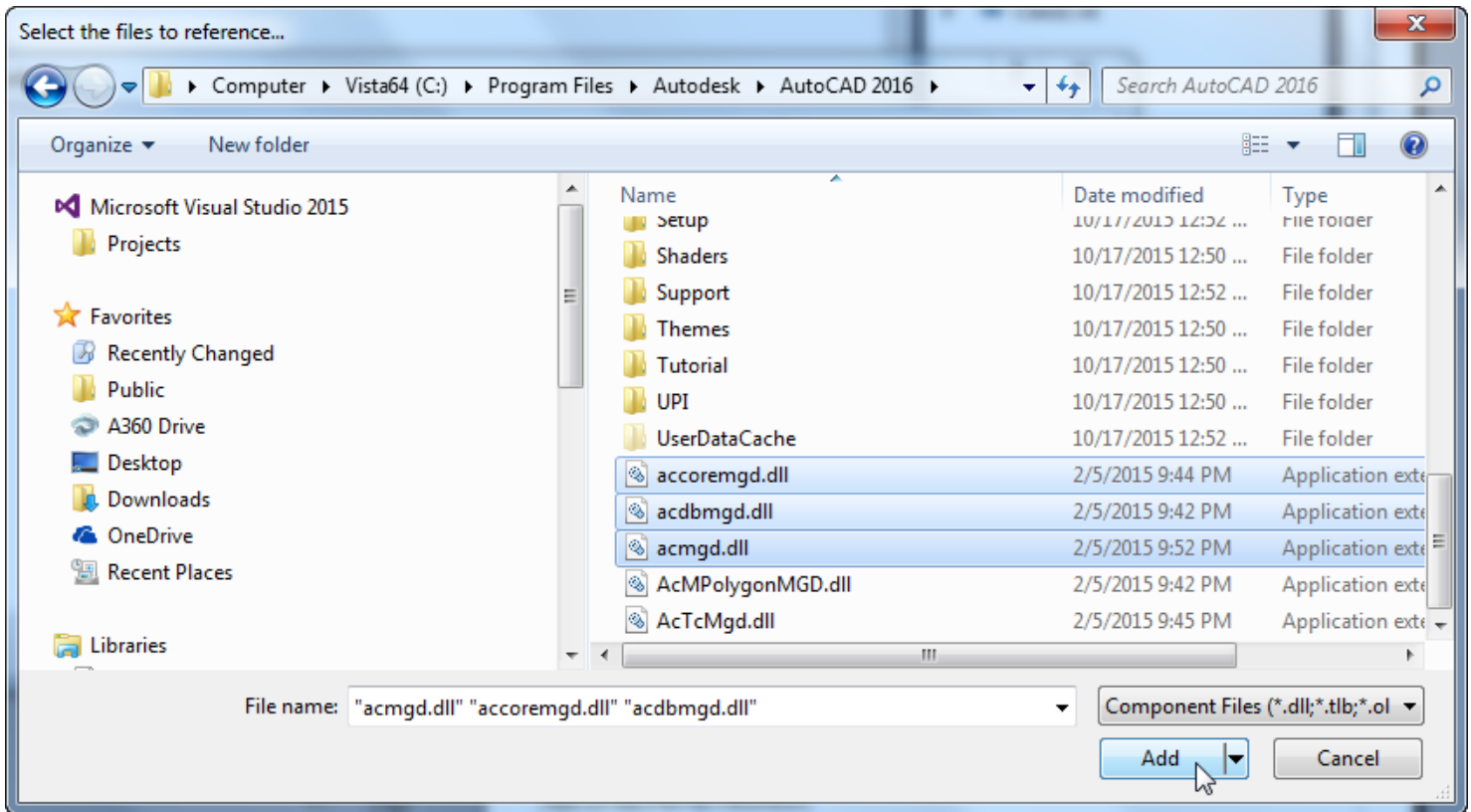
Creating a .DLL in .NET is easy as we have just discovered. But that DLL is not an AutoCAD Add-In until the essential AutoCAD DLL References are added. This is done by clicking “Project” then “Add Reference” in the Visual Studio menu.



There are thousands of potential References that can be added to our Project. The .NET Framework itself contains many useful references. The 3 references we want is not part of Microsoft’s Framework. It’s an Autodesk element. We need to Browse for it.



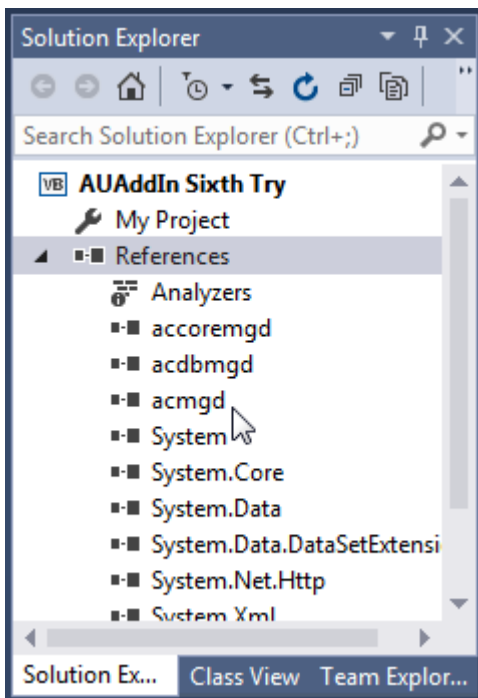
Browsing to the AutoCAD 2016 Directory as shown above and entering "ac*mgd.dll" in the File name textbox and hitting ENTER filters out most of the files in this directory.



Filtering out the .dll files by filename restricts the number of DLLs displayed and we can easily select the 3 essential DLLs we need to reference to turn our project into an AutoCAD Add-In.

As we can see here, we need the "accoremgd.dll", "acdbmgd.dll", and "acmgd.dll" files referenced.

Click the "Add" button now, then the OK button in the "References" dialog box.



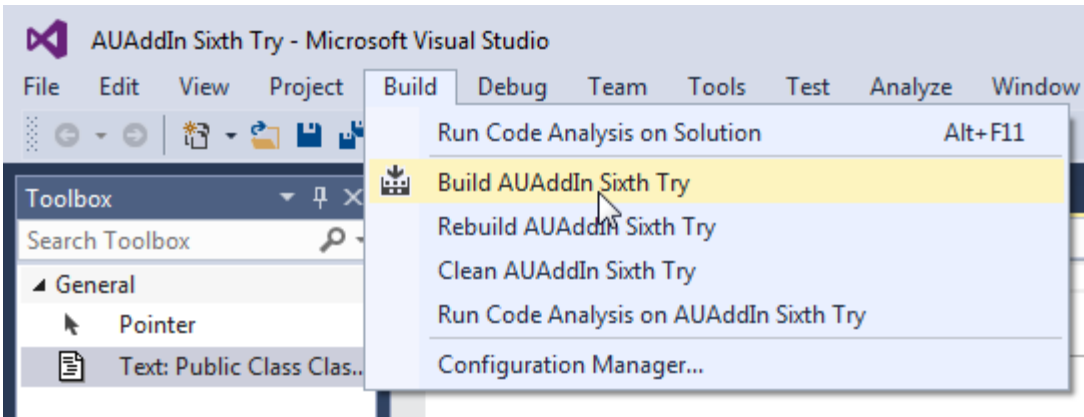
Now if we look at the References node in the Solution Explorer, we should see the three references we just added along with other added by Visual Studio when we created our project.



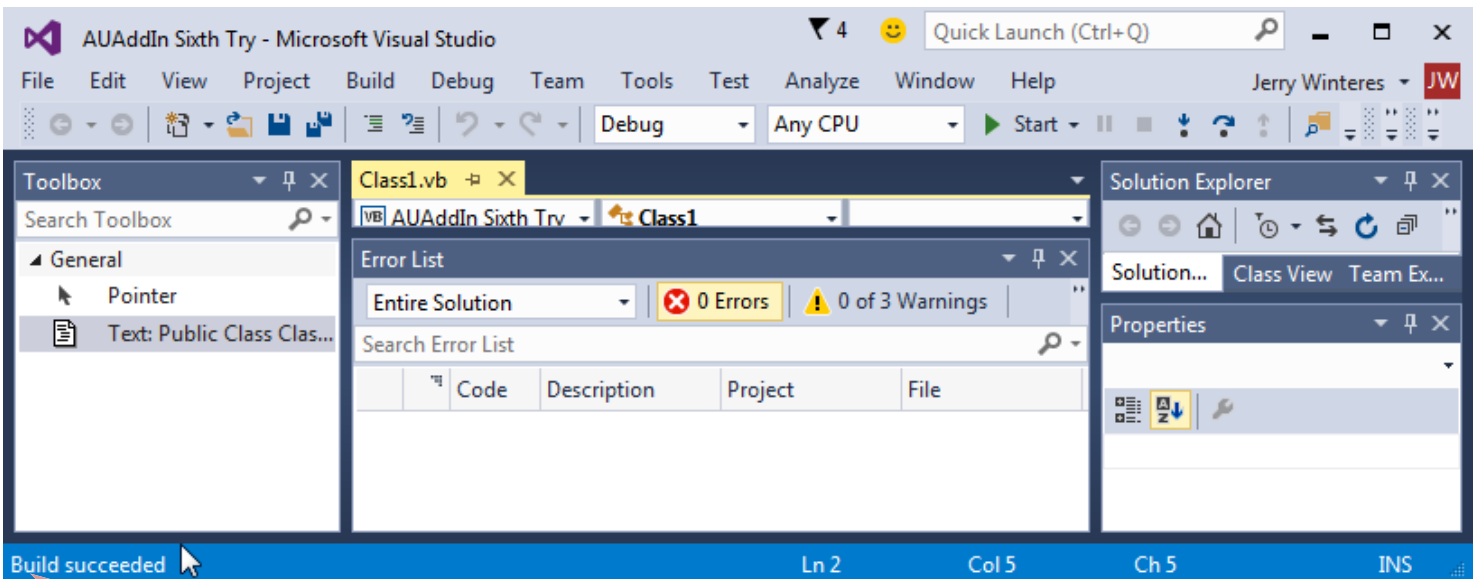
Compile and Load an Add-In

Projects, Code, and References are worthless unless these elements are combined by being Compiled.

SAVE THE PROJECT BEFORE COMPILING

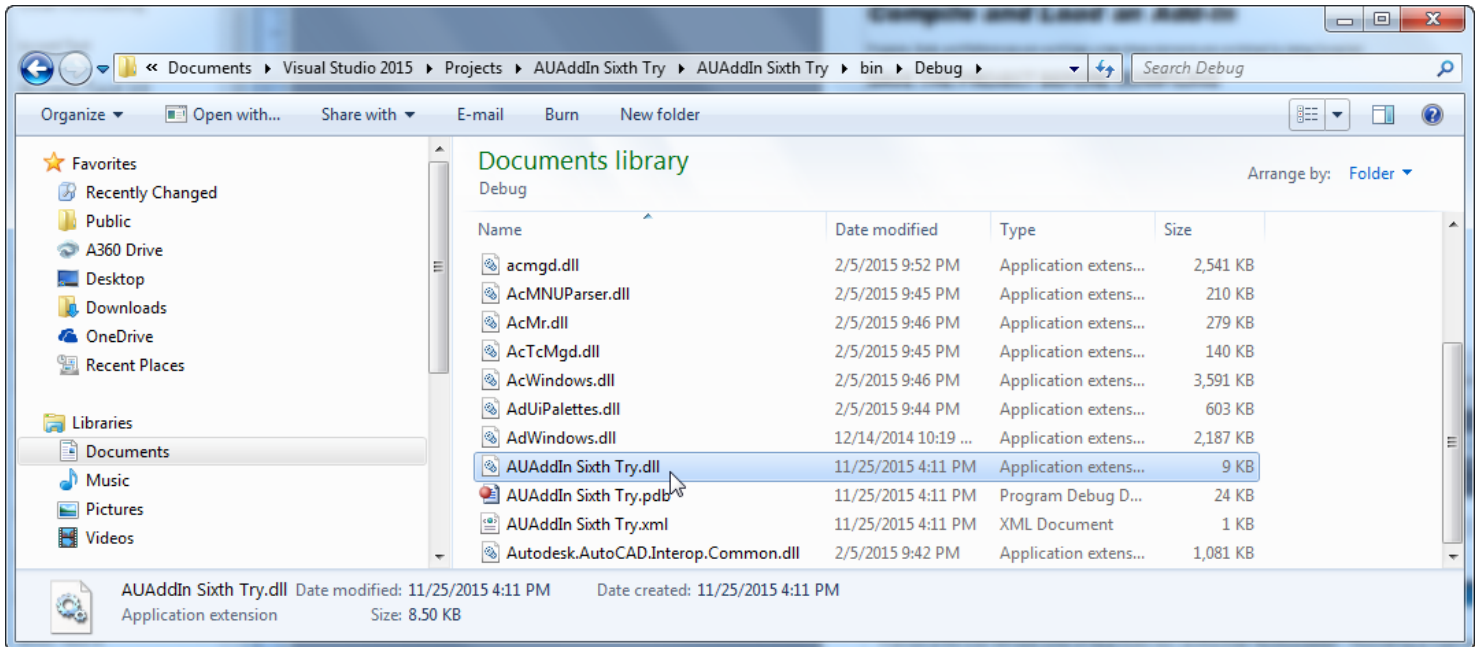


Let's compile or "Build" our project by going to the Visual Studio menu: "Build", "Build xxxxxxxxxx".



If we look at the lower-left hand corner of Visual Studio now, we should see "Build succeeded". This is telling us in part that our project was compiled into a .DLL file

Let's browse to the "Documents\Visual Studio 2015\Projects\AUAddIn Sixth Try\AUAddIn Sixth Try\bin\Debug" directory.



Here we can see our “AUAddIn Sixth Try.dll” dll listed along with a number of other DLL files. We are going to ignore the other files for now. We can change the AutoCAD references’ Copy Local property to False which would eliminate most of these files but they aren’t hurting us now so we will leave them alone.

Knowing where these References are located is important because the next step we are going to perform will require us to browse to and select the DLL.

Let’s Start AutoCAD now.

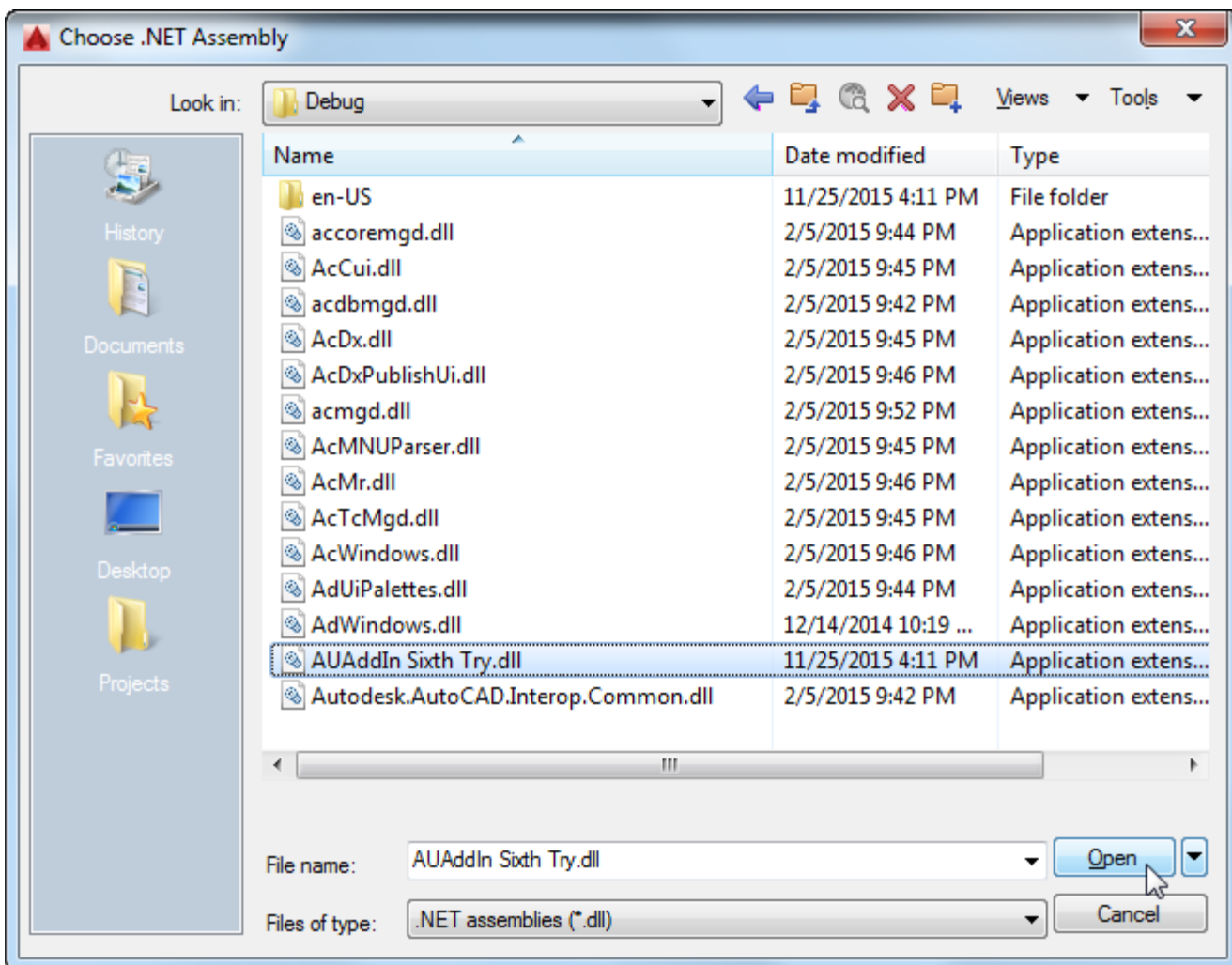
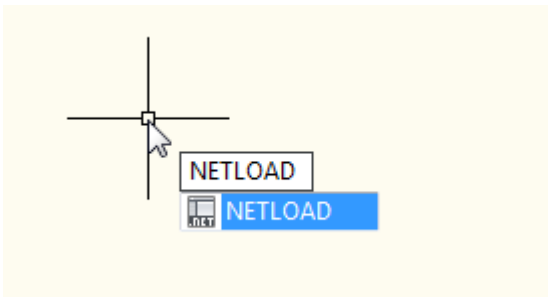
Notice on the lower left hand corner of the AutoCAD startup window the “Loading plug-ins . . .” notifications. One of them (as we can see here) is acmgd.dll, one of the same DLL we referenced to turn our project into an AutoCAD Add-In.



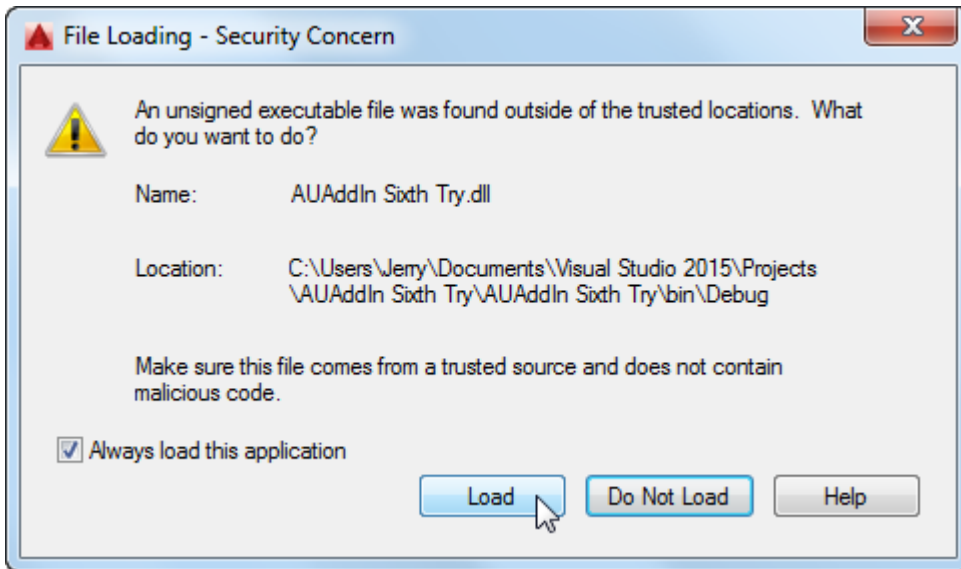


NETLOAD is your friend.

Use the “NETLOAD” command and browse to the new DLL created when we compiled (Built) our project.



Make sure the correct DLL is selected then click “Open”.



If you see this dialog box, go ahead and click the “Always load this application” and click the “Load” button.

Now What?

If after running the NETLOAD command, nothing else happened, this means our new DLL has been successfully loaded.

QUICK REVIEW

At this point in this class, we have learned how to create a new Class Library, Add the Essential AutoCAD References, and Compile and Load our Program. But what does our program do? Let’s take a look at the code.

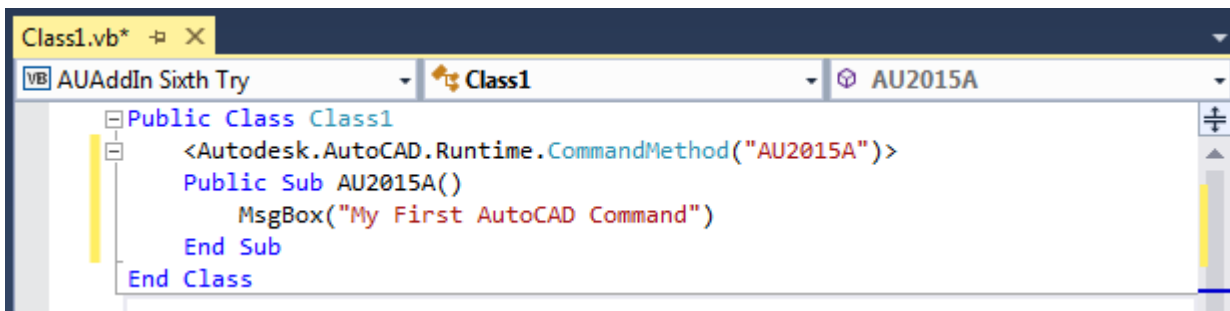


According to our code, our program does nothing.

Let’s try adding some code.

CLOSE AUTOCAD

TYPE THE FOLLWING CODE IN “Class1”





BUILD the Project

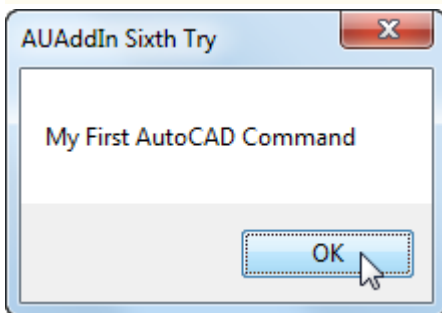
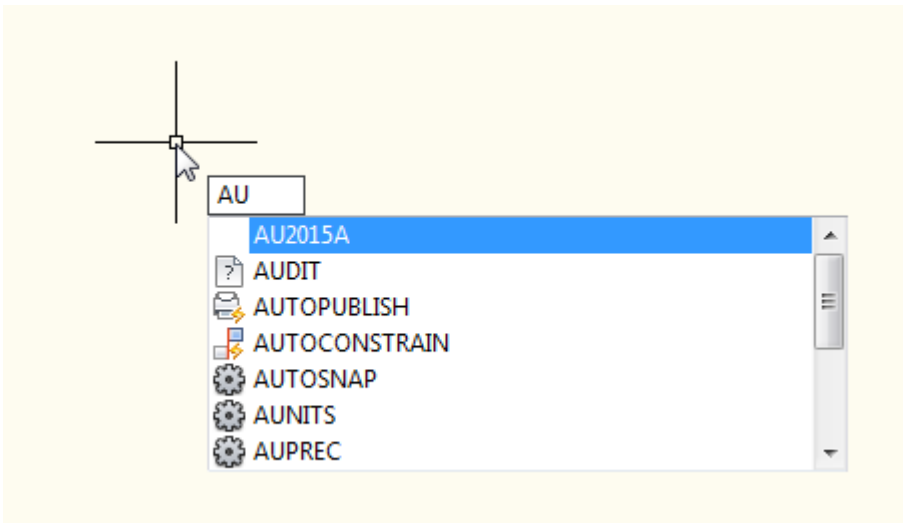
Visual Studio Menu : Build -> Build the project name here

START AutoCAD

NETLOAD the project

Note, AutoCAD should remember the most recent Netload directory which simplifies things a little bit.

Run the new "AU2015A" command.



Here is the MsgBox we put into our Command.

Let's take a quick look at the code we typed again.

```
<Autodesk.AutoCAD.Runtime.CommandMethod("AU2015A")>  
Public Sub AU2015A()  
    MsgBox("My First AutoCAD Command")  
End Sub
```

The text "AU2015A" appears twice in the code. The first time is inside the CommandMethod. This is where we define the name of our Command. The second is the name of the Public Sub (Procedure). These do not have to be the same but many developers keep them the same to keep things uniform.

MsgBox is used to display text. In this case, "My First AutoCAD Command".



Modifying Existing Code

The bulk of our time in this class will be spent modifying existing code. Before we do this we need to understand a few programming fundamentals.

Procedures—Pieces of code that begin with “Sub” and end with “End Sub”.

```
Sub ShowMessageBox()  
    MsgBox("Here we are.")  
End Sub
```

Functions—Pieces of code that begin with “Function” and end with “End Function” and return an Object.

```
Function GetPi() As Double  
    Return Math.PI  
End Function
```

```
Function GetArea(Width As Double, Height As Double) As Double  
    Return Width * Height  
End Function
```

Parameters—Values passed into Procedures and Functions. In the GetArea Function shown above, Width and Height are parameters.

Variable—A name that represents an Object or Value. In the following example, X and Y are variables of the type “Double”. myDoc is a variable of type “Document” that represents an AutoCAD DWG file open in AutoCAD. Z is also declared as Double.

```
Sub GetValues()  
    Dim X As Double  
    Dim Y As Double  
    Dim myDoc As Autodesk.AutoCAD.ApplicationServices.Document  
    myDoc = Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocument  
    X = myDoc.Editor.GetDouble("Enter X").Value  
    Y = myDoc.Editor.GetDouble("Enter Y").Value  
    Dim Z As Double = X * Y  
    Z = X / Y  
    Z = X + Y  
End Sub
```

Variables are declared with the words “Dim” from within Functions and Procedures and , Dim, “Public”, “Private”, and “Const” in side Classes and Modules.

```
Public ABC As Double  
Private DEF As Double  
Const GHI As Double = 1.234
```



Class—A group of code that defines an Object.

```
Public Class Computer
    Private pRAM As Integer
    Private pClockSpeed As Double
    Private pDrives As New List(Of Drive)
    Property RAM As Integer
        Get
            Return pRAM
        End Get
        Set(value As Integer)
            pRAM = value
        End Set
    End Property
    Property ClockSpeed As Double
        Get
            Return pClockSpeed
        End Get
        Set(value As Double)
            pClockSpeed = value
        End Set
    End Property
    ReadOnly Property Drives As List(Of Drive)
        Get
            Return pDrives
        End Get
    End Property
    Function AddDrive(DriveLetter As String, DriveCapacity As Long) As Drive
        Dim myDrive As New Drive("C", 4000000000000)
        Drives.Add(myDrive)
        Return myDrive
    End Function
End Class

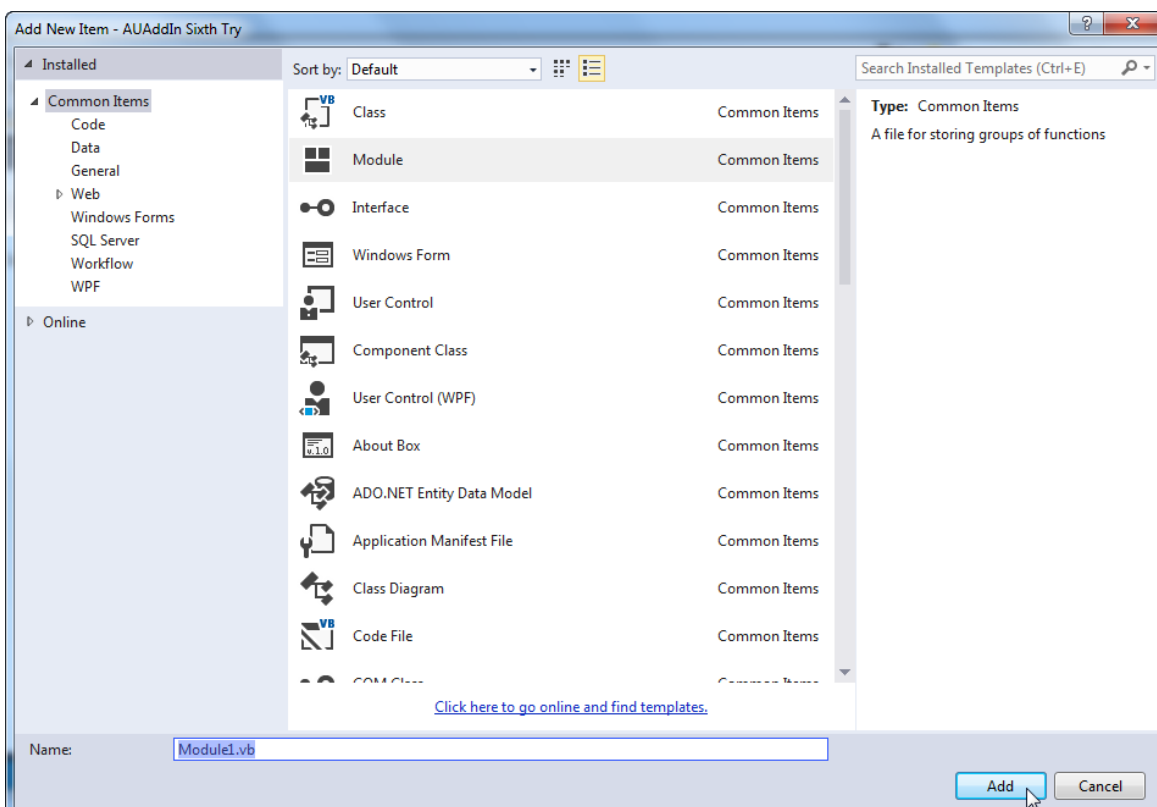
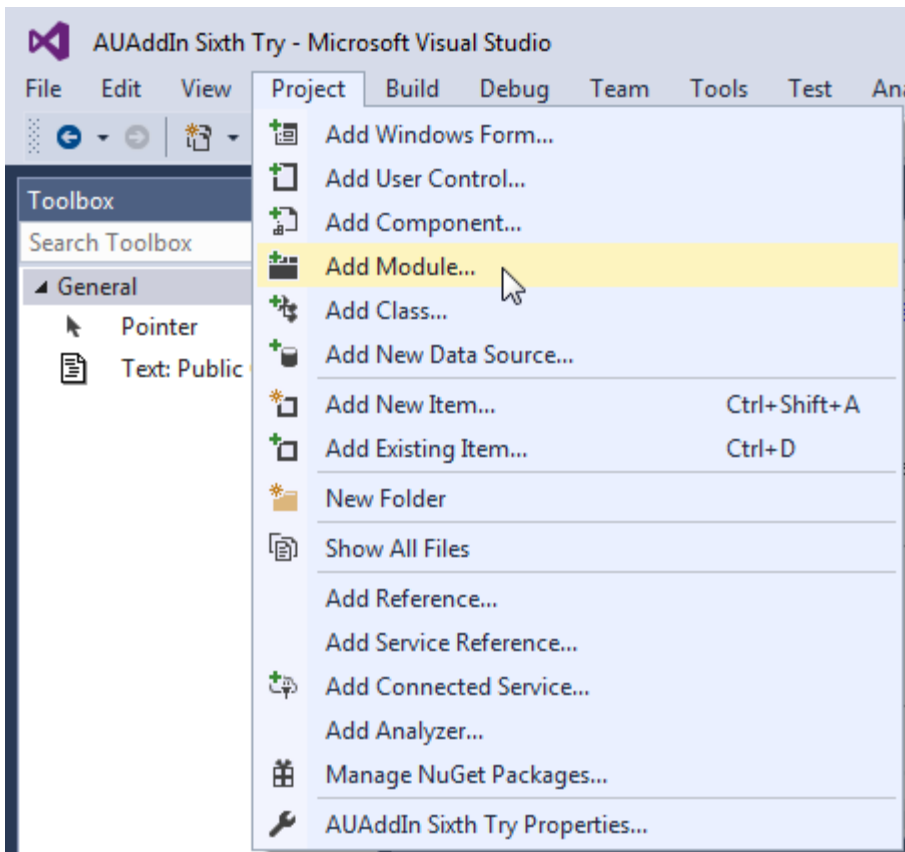
Public Class Drive
    Private pLetter As String
    Private pCapacity As Long
    Property Letter As String
        Get
            Return pLetter
        End Get
        Set(value As String)
            pLetter = value
        End Set
    End Property
    Property Capacity As Long
        Get
            Return pCapacity
        End Get
        Set(value As Long)
            pCapacity = value
        End Set
    End Property
    Public Sub New(Letter As String, Capacity As Long)
        pLetter = Letter
        pCapacity = Capacity
    End Sub
End Class
```

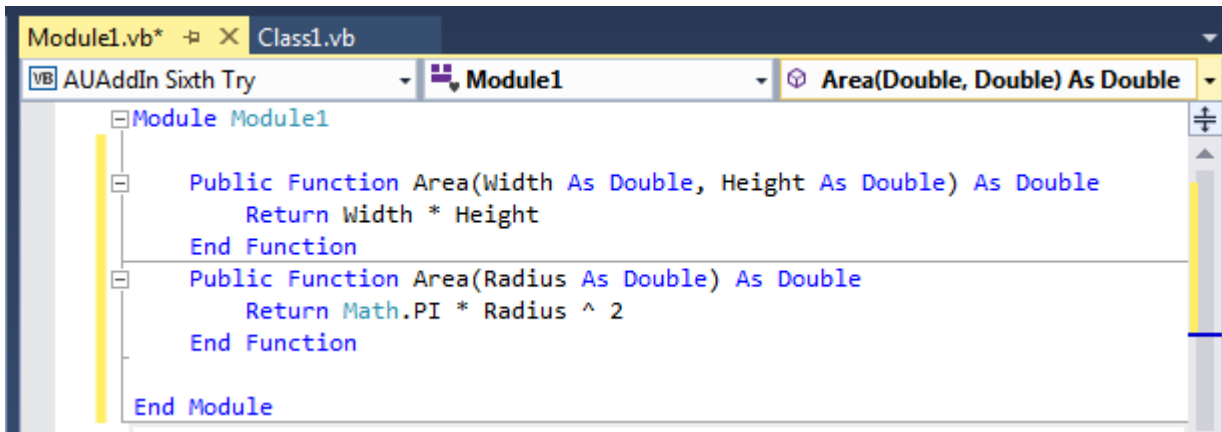


AUTODESK UNIVERSITY 2015

Module—A group of code that is accessible to other parts of a project inside and outside the project.

We have been working in the file “Class1.vb” which was created when the new project was started. If we want to use a Module, we need to add it to our Project.





```
Module1.vb* X Class1.vb
VB AUAddIn Sixth Try Module1 Area(Double, Double) As Double
Module Module1
    Public Function Area(Width As Double, Height As Double) As Double
        Return Width * Height
    End Function
    Public Function Area(Radius As Double) As Double
        Return Math.PI * Radius ^ 2
    End Function
End Module
```

Here is a Module with two Functions. These Functions are declared as Public and are accessible to other Procedures and Functions within the Project.

Please note that these two Functions have the same name but different parameter signatures. This is called an Overloaded Function.

Create a new Module and type in the two Functions shown above.

COPY AND PASTE IS YOUR FRIEND

Now it's time to do some code modifying. Copy and Paste the Area Function with two parameters and rename the Function to "Volume".

```
Public Function Volume(Width As Double, Height As Double) As Double
    Return Width * Height
End Function
```

Now, something's wrong with this Function. The Volume of a 'box' needs an additional parameter. Let's add it now.

```
Public Function Volume(Width As Double, Height As Double, Depth As Double) As Double
    Return Width * Height * Depth
End Function
```

That's all it takes to modify existing code to accomplish a different task. Let's take a look at a few more examples.

Enter the following URL in a web browser, then copy and paste all of the code into the Class Module:

<http://www.vbcad.com/au2015vbcode.txt>

We are also going to need one of Autodesk's sample dwg files:

http://download.autodesk.com/us/samplefiles/acad/blocks_and_tables_-_imperial.dwg



```
Function PickPoint(Msg As String) As Autodesk.AutoCAD.Geometry.Point3d
    Dim myDoc As Autodesk.AutoCAD.ApplicationServices.Document
    myDoc = Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocument
    Dim myEditor As Autodesk.AutoCAD.EditorInput.Editor = myDoc.Editor
    Dim myPPR As Autodesk.AutoCAD.EditorInput.PromptPointResult
    myPPR = myEditor.GetPoint(Msg)
    If myPPR.Status = Autodesk.AutoCAD.EditorInput.PromptStatus.OK Then
        Return myPPR.Value
    Else
        Return Nothing
    End If
End Function

Function PickPoint(Msg As String, BasePoint As Autodesk.AutoCAD.Geometry.Point3d) _
    As Autodesk.AutoCAD.Geometry.Point3d
    Dim myDoc As Autodesk.AutoCAD.ApplicationServices.Document
    myDoc = Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocument
    Dim myEditor As Autodesk.AutoCAD.EditorInput.Editor = myDoc.Editor
    Dim myPPO As New Autodesk.AutoCAD.EditorInput.PromptPointOptions(Msg)
    myPPO.BasePoint = BasePoint
    myPPO.UseBasePoint = True
    Dim myPPR As Autodesk.AutoCAD.EditorInput.PromptPointResult
    myPPR = myEditor.GetPoint(myPPO)
    If myPPR.Status = Autodesk.AutoCAD.EditorInput.PromptStatus.OK Then
        Return myPPR.Value
    Else
        Return Nothing
    End If
End Function

Function DrawLine(dbIn As Autodesk.AutoCAD.DatabaseServices.Database,
    X1 As Double, Y1 As Double, Z1 As Double, X2 As Double, Y2 As Double, Z2 As Double) _
    As Autodesk.AutoCAD.DatabaseServices.ObjectId

    Using myTrans As Autodesk.AutoCAD.DatabaseServices.Transaction = _
        dbIn.TransactionManager.StartTransaction
        Dim myBTR As Autodesk.AutoCAD.DatabaseServices.BlockTableRecord = _
            dbIn.CurrentSpaceId.GetObject(Autodesk.AutoCAD.DatabaseServices.OpenMode.ForWrite)
        Dim stPt As New Autodesk.AutoCAD.Geometry.Point3d(X1, Y1, Z1)
        Dim enPt As New Autodesk.AutoCAD.Geometry.Point3d(X2, Y2, Z2)
        Dim newLine As New Autodesk.AutoCAD.DatabaseServices.Line(stPt, enPt)
        myBTR.AppendEntity(newLine)
        myTrans.AddNewlyCreatedDBObject(newLine, True)
        myTrans.Commit()
        Return newLine.ObjectId
    End Using
End Function

Function PickEntity(Msg As String) As Autodesk.AutoCAD.DatabaseServices.ObjectId
    Dim myDoc As Autodesk.AutoCAD.ApplicationServices.Document
    myDoc = Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocument
    Dim myEditor As Autodesk.AutoCAD.EditorInput.Editor = myDoc.Editor
    Dim myPEO As New Autodesk.AutoCAD.EditorInput.PromptEntityOptions(Msg)
    Dim myPER As Autodesk.AutoCAD.EditorInput.PromptEntityResult
    myPER = myEditor.GetEntity(myPEO)
    If myPER.Status = Autodesk.AutoCAD.EditorInput.PromptStatus.OK Then
        Return myPER.ObjectId
    Else
        Return Nothing
    End If
End Function
```



```
Function SelectBlocksOnScreen() As Autodesk.AutoCAD.DatabaseServices.ObjectId()
    Dim myDoc As Autodesk.AutoCAD.ApplicationServices.Document
    myDoc = Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocument
    Dim myEditor As Autodesk.AutoCAD.EditorInput.Editor = myDoc.Editor
    Dim myTVs(0) As Autodesk.AutoCAD.DatabaseServices.TypedValue
    myTVs(0) = New Autodesk.AutoCAD.DatabaseServices.TypedValue(0, "INSERT")
    Dim myFilter As New Autodesk.AutoCAD.EditorInput.SelectionFilter(myTVs)
    Dim myPSR As Autodesk.AutoCAD.EditorInput.PromptSelectionResult
    myPSR = myEditor.GetSelection(myFilter)
    If myPSR.Status = Autodesk.AutoCAD.EditorInput.PromptStatus.OK Then
        Return myPSR.Value.GetObjectIds
    Else
        Return Nothing
    End If
End Function

Function SelectAllBlocks() As Autodesk.AutoCAD.DatabaseServices.ObjectId()
    Dim myDoc As Autodesk.AutoCAD.ApplicationServices.Document
    myDoc = Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocument
    Dim myEditor As Autodesk.AutoCAD.EditorInput.Editor = myDoc.Editor
    Dim myTVs(0) As Autodesk.AutoCAD.DatabaseServices.TypedValue
    myTVs(0) = New Autodesk.AutoCAD.DatabaseServices.TypedValue(0, "INSERT")
    Dim myFilter As New Autodesk.AutoCAD.EditorInput.SelectionFilter(myTVs)
    Dim myPSR As Autodesk.AutoCAD.EditorInput.PromptSelectionResult
    myPSR = myEditor.SelectAll(myFilter)
    If myPSR.Status = Autodesk.AutoCAD.EditorInput.PromptStatus.OK Then
        Return myPSR.Value.GetObjectIds
    Else
        Return Nothing
    End If
End Function

Function GetBlockAttributes(BlockID As Autodesk.AutoCAD.DatabaseServices.ObjectId) _
    As Dictionary(Of String, String)
    Dim myD As New Dictionary(Of String, String)
    Using myTrans As Autodesk.AutoCAD.DatabaseServices.Transaction =
        BlockID.Database.TransactionManager.StartTransaction()
        Dim myBlockRef As Autodesk.AutoCAD.DatabaseServices.BlockReference =
            BlockID.GetObject(Autodesk.AutoCAD.DatabaseServices.OpenMode.ForRead)
        If myBlockRef.AttributeCollection.Count > 0 Then
            For Each myAttRefID As _
                Autodesk.AutoCAD.DatabaseServices.ObjectId In myBlockRef.AttributeCollection
                Dim myAttRef As Autodesk.AutoCAD.DatabaseServices.AttributeReference = _
                    myAttRefID.GetObject(Autodesk.AutoCAD.DatabaseServices.OpenMode.ForRead)
                If myD.ContainsKey(myAttRef.Tag) = False Then
                    myD.Add(myAttRef.Tag, myAttRef.TextString)
                End If
            Next
        End If
        myTrans.Dispose()
    End Using
    Return myD
End Function
```



```

Function GetBlockProperties(ObjID As Autodesk.AutoCAD.DatabaseServices.ObjectId) _
    As Dictionary(Of String, Object)
    Dim myD As New Dictionary(Of String, Object)
    Using myTrans As Autodesk.AutoCAD.DatabaseServices.Transaction = _
        ObjID.Database.TransactionManager.StartTransaction
        Dim myEnt As Autodesk.AutoCAD.DatabaseServices.BlockReference = _
            ObjID.GetObject(Autodesk.AutoCAD.DatabaseServices.OpenMode.ForRead)
        For Each myPInfo As System.Reflection.PropertyInfo In _
            GetType(Autodesk.AutoCAD.DatabaseServices.BlockReference).GetProperties
            myD.Add(myPInfo.Name, myPInfo.GetValue(myEnt))
        Next
    End Using
    Return myD
End Function

Function GetBTRProperties(ObjID As Autodesk.AutoCAD.DatabaseServices.ObjectId) _
    As Dictionary(Of String, Object)
    Dim myD As New Dictionary(Of String, Object)
    Using myTrans As Autodesk.AutoCAD.DatabaseServices.Transaction = _
        ObjID.Database.TransactionManager.StartTransaction
        Dim myEnt As Autodesk.AutoCAD.DatabaseServices.BlockTableRecord = _
            ObjID.GetObject(Autodesk.AutoCAD.DatabaseServices.OpenMode.ForRead)
        For Each myPInfo As System.Reflection.PropertyInfo In _
            GetType(Autodesk.AutoCAD.DatabaseServices.BlockTableRecord).GetProperties
            myD.Add(myPInfo.Name, myPInfo.GetValue(myEnt))
        Next
    End Using
    Return myD
End Function

Sub WriteToFile(TextToWrite As String, FilePath As String)
    Dim mySW As New IO.StreamWriter(FilePath)
    mySW.WriteLine(TextToWrite)
    mySW.Close()
    mySW.Dispose()
End Sub

Sub AppendToFile(TextToWrite As String, FilePath As String)
    Dim mySW As New IO.StreamWriter(FilePath, True)
    mySW.WriteLine(TextToWrite)
    mySW.Close()
    mySW.Dispose()
End Sub

```

That's a lot of code that does a lot of different things. We are going to be using them to accomplish a number of different tasks.

```

<Autodesk.AutoCAD.Runtime.CommandMethod("AU2015B")>
Public Sub AU2015B()
    Dim PtA As Autodesk.AutoCAD.Geometry.Point3d = PickPoint("Pick")
    Dim PtB As Autodesk.AutoCAD.Geometry.Point3d = PickPoint("Pick 2", PtA)
    MsgBox(PtA.ToString & vbTab & PtB.ToString)
End Sub

```



AUTODESK UNIVERSITY 2015

```
<Autodesk.AutoCAD.Runtime.CommandMethod("AU2015C")>
Public Sub AU2015C()
    DrawLine
    (Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocument.Database, 0, 0, 0,
    -2, 2, 0)
    DrawLine
    (Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocument.Database, 0, 0, 0,
    2, 2, 0)
    DrawLine
    (Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocument.Database, 0, 0, 0,
    0, 2, 0)
End Sub
```

```
<Autodesk.AutoCAD.Runtime.CommandMethod("AU2015D")>
Public Sub AU2015D()
    Dim myLineID As Autodesk.AutoCAD.DatabaseServices.ObjectId = PickEntity("Select a line:")
    If IsNothing(myLineID) = False Then
        Using myTrans As Autodesk.AutoCAD.DatabaseServices.Transaction =
            myLineID.Database.TransactionManager.StartTransaction
            Dim myLine As Autodesk.AutoCAD.DatabaseServices.Line =
                myLineID.GetObject(Autodesk.AutoCAD.DatabaseServices.OpenMode.ForRead)
            MsgBox(myLine.Layer)
        End Using
    End If
End Sub
```

```
<Autodesk.AutoCAD.Runtime.CommandMethod("AU2015E")>
Public Sub AU2015E()
    Dim myIDs() As Autodesk.AutoCAD.DatabaseServices.ObjectId = SelectBlocksOnScreen()
End Sub
```

```
<Autodesk.AutoCAD.Runtime.CommandMethod("AU2015F")>
Public Sub AU2015F()
    Dim myIDs() As Autodesk.AutoCAD.DatabaseServices.ObjectId = SelectAllBlocks()
    If myIDs Is Nothing = False Then
        For Each myID As Autodesk.AutoCAD.DatabaseServices.ObjectId In myIDs
            Dim myAtts As Dictionary(Of String, String) = GetBlockAttributes(myID)
            For Each myVal As KeyValuePair(Of String, String) In myAtts
                MsgBox(myVal.Key & vbTab & myVal.Value)
            Next
        Next
    End If
End Sub
```



```
<Autodesk.AutoCAD.Runtime.CommandMethod("AU2015G")>
Public Sub AU2015G()
    Dim myIDs2() As Autodesk.AutoCAD.DatabaseServices.ObjectId = SelectAllBlocks()
    If myIDs2 Is Nothing = False Then
        For Each myID As Autodesk.AutoCAD.DatabaseServices.ObjectId In myIDs2
            Dim myProps As Dictionary(Of String, Object) = GetBlockProperties(myID)
            For Each myAtt As KeyValuePair(Of String, String) In GetBlockAttributes(myID)
                If myProps("DynamicBlockTableRecord") = myProps("BlockTableRecord") Then
                    MsgBox(myProps("Name").ToString & vbTab & myProps("Layer").ToString & _
                        vbTab & myAtt.Key & vbTab & myAtt.Value)
                Else
                    Dim myProps2 As Dictionary(Of String, Object) = _
                        GetBTRProperties(myProps("DynamicBlockTableRecord"))
                    MsgBox(myProps2("Name").ToString & "(" & myProps("Name").ToString & ")" & _
                        vbTab & myProps("Layer").ToString & vbTab & _
                        myAtt.Key & vbTab & myAtt.Value)
                End If
            Next
        Next
    End If
End Sub

<Autodesk.AutoCAD.Runtime.CommandMethod("AU2015H")>
Public Sub AU2015H()
    WriteToFile("this is a test.", "C:\temp\output.txt")
End Sub

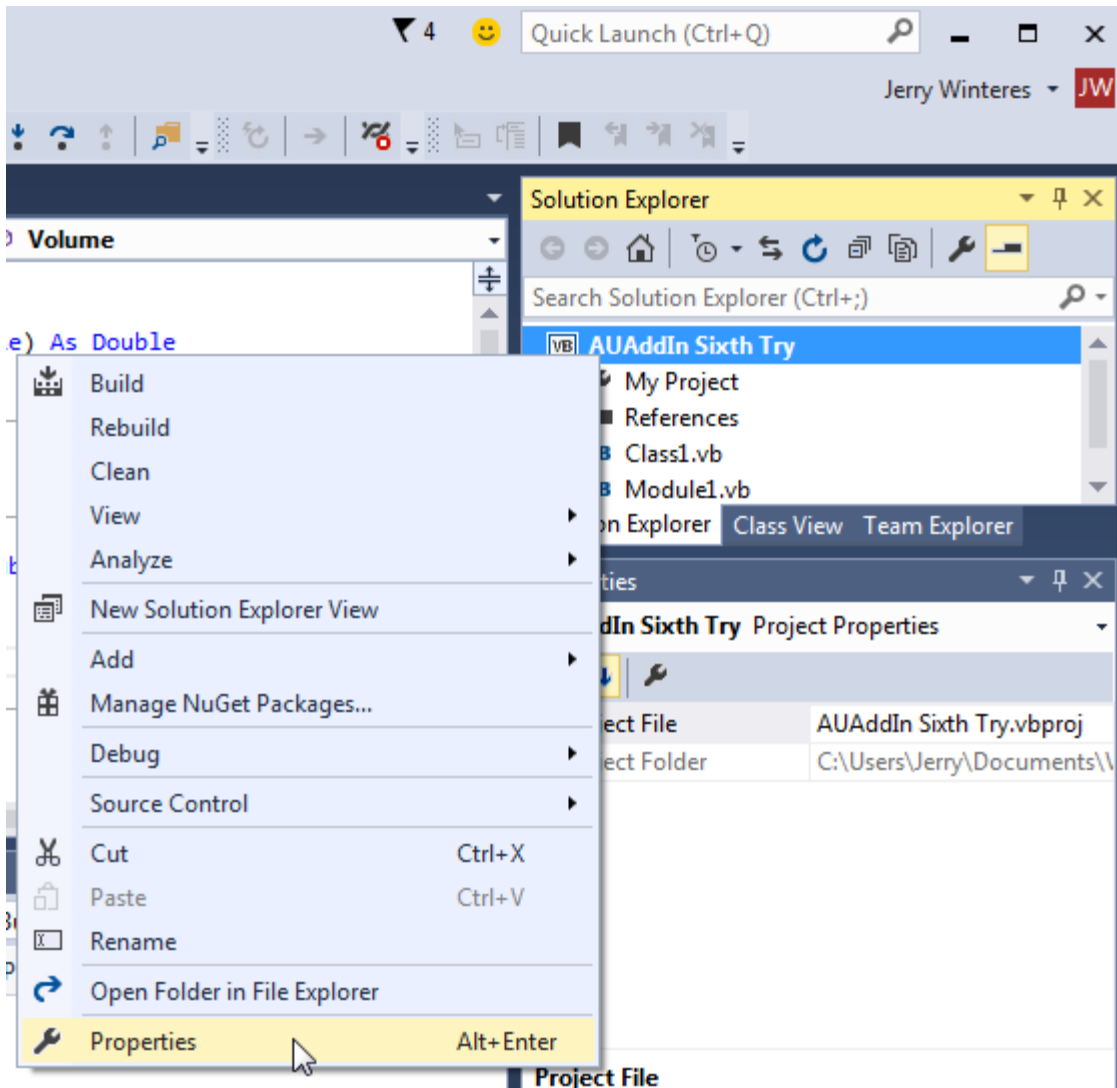
<Autodesk.AutoCAD.Runtime.CommandMethod("AU2015J")>
Public Sub AU2015J()
    AppendToFile("this is another test.", "C:\temp\output.txt")
End Sub
```



Debugging and Real-Time Development

Compiling, starting AutoCAD, NetLoading, Running, closing AutoCAD, changing code, starting AutoCAD, NetLoading, it works but it's not the most efficient way to do things. Let's get Visual Studio and AutoCAD working together to make things happen.

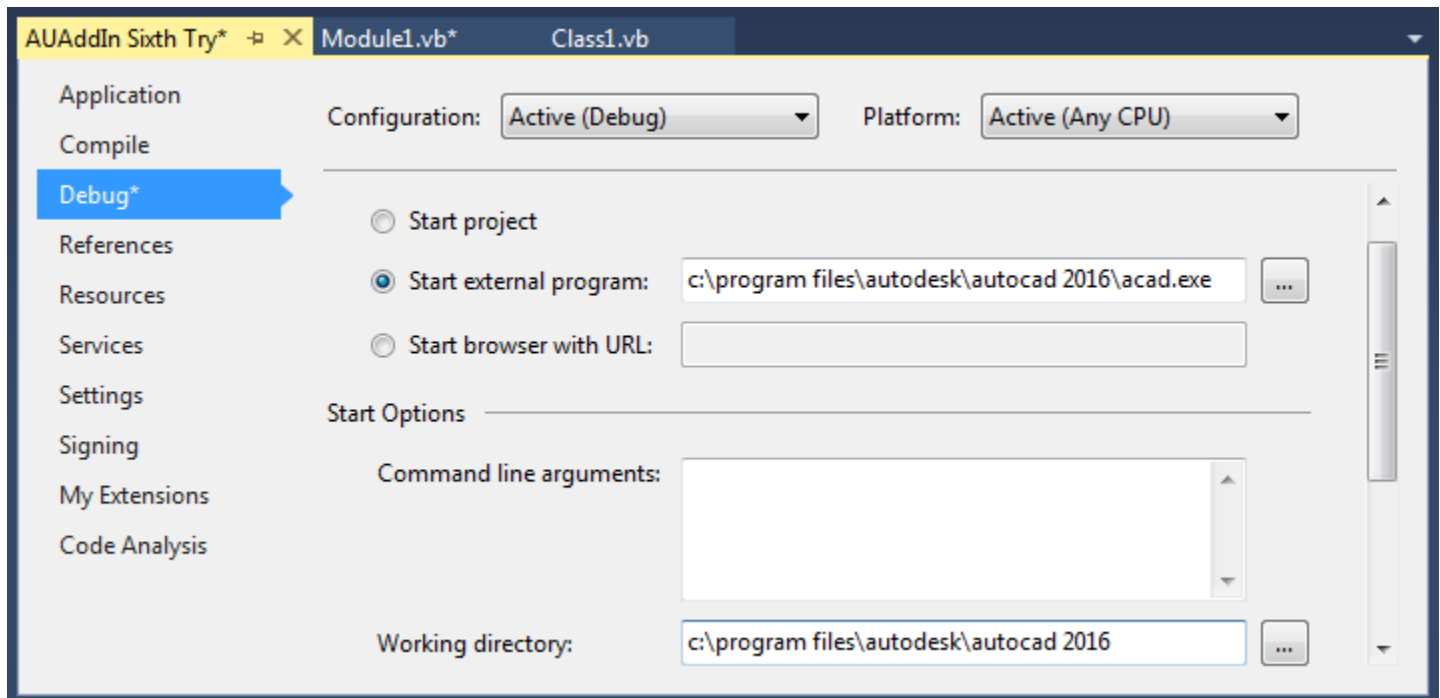
Right-Click on the Project Name in the Solution Explorer and click on Properties



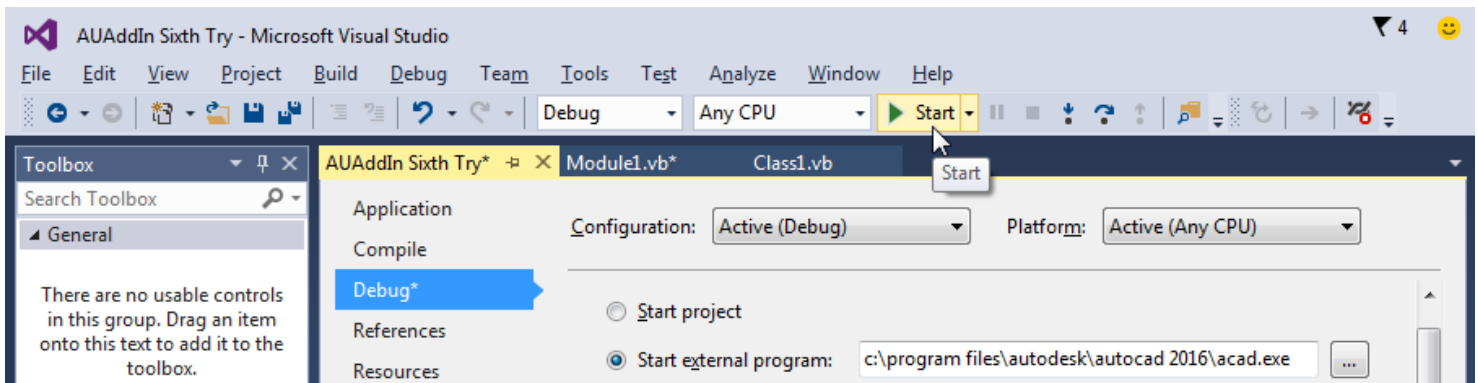


AUTODESK UNIVERSITY 2015

Now, click on the Debug tab, click on the “Start external program” option button and select or type the path to the AutoCAD Executable in the text box next to it. Also, copy and paste the path where the executable is located into the “Working directory”.

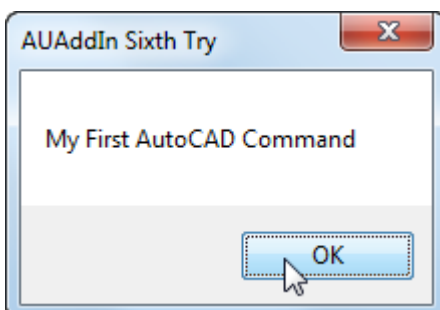


Now, click on the “Start” button.



When we begin debugging AutoCAD now starts and a link between AutoCAD and Visual Studio is created.

Netload the application and run the command “AU2015A”.



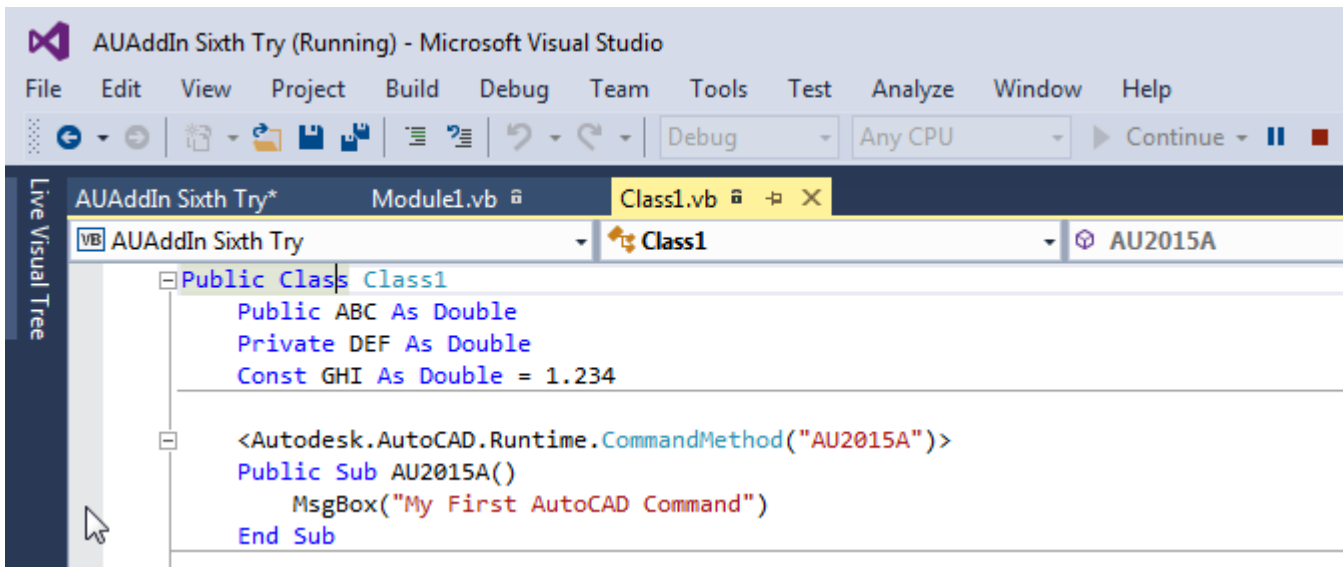
Here’s the MessageBox we have seen before. Let’s finish the command by clicking the “OK” button.

Thus far, nothing is different. Or at least it doesn’t look different. But there is a link between Visual Studio and AutoCAD. We will see this happen when we add a breakpoint in Visual Studio.

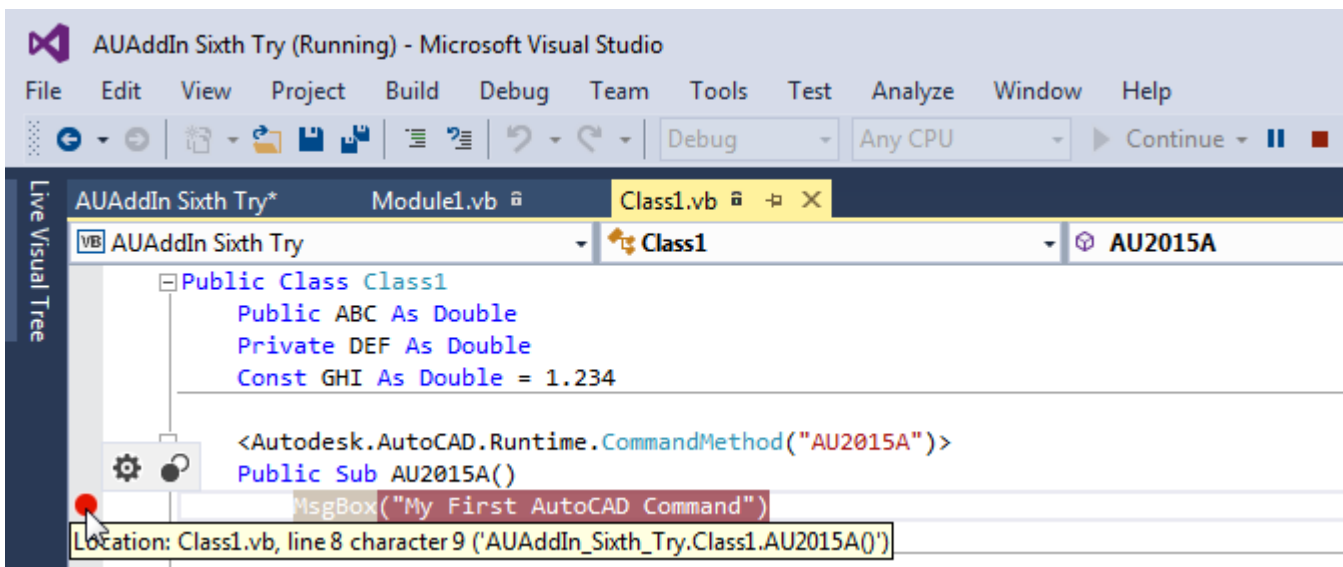


AUTODESK UNIVERSITY 2015

There is a gray vertical bar in Visual Studio that, when clicked on, adds a breakpoint.



Click in this gray bar to add a breakpoint to the project.

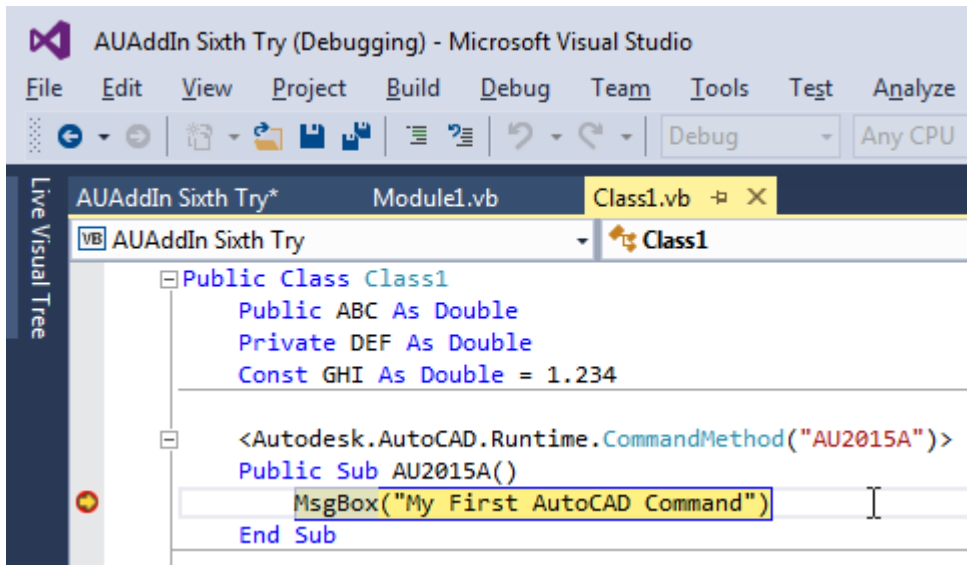


Now, run the “AU2015A” command again.

This time when the command is run, we don’t see the MessageBox immediately. We are taken back into Visual Studio.

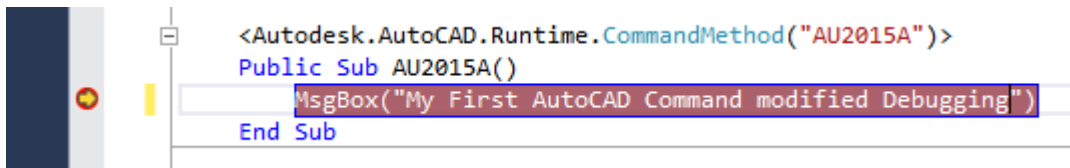


AUTODESK UNIVERSITY 2015

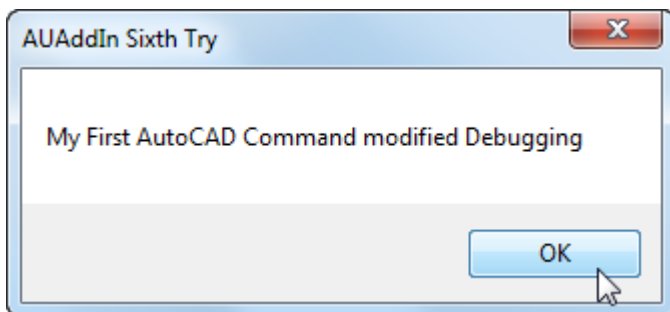


The line of code shown in Yellow is the next line of code that will be run.

While we are debugging, we can modify our code.



After modifying the code, hit the F5 key on the keyboard. This will run the code.



Now we can modify our code without stopping and starting AutoCAD repeatedly. This is called "Edit and Continue".



REVIEW

Creating AutoCAD Add-ins using VB.NET is easy to do. We have learned how to create a new Add-in from scratch, create new commands, and step through and modify our code. I hope this class has been helpful and that you will continue your learning by attending Autodesk University classes in the future.

Thank you for your time.

Jerry Winters (jerryw@vbcad.com)