AUTODESK UNIVERSITY 2013

VB.NET: A Hands-on Introduction to Creating Autodesk® AutoCAD® Add-Ins

Jerry Winters – VB CAD, Inc.

DV3328-L The most difficult part of learning something new is getting a good initial introduction. If you want to learn how to customize Autodesk AutoCAD software with VB.NET, this is the class you need. In this hands-on lab, we walk through the entire process of creating a VB.NET add-in, including creating a new project in VB.NET Express Edition, creating custom commands, debugging and testing your code, and creating an autoload package. Learn from an industry leader and begin writing your own add-ins now. No prior programming experience is required.

Learning Objectives

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

- Create a new VB.NET project
- Add references to create an AutoCAD add-in
- Create new custom commands
- Package your add-in so it will autoload

Wednesday, Dec 4, 8:00 AM - 9:30 AM – San Polo 3505

About the Speaker

Jerry Winters has educated thousands as he has brought his simple and humorous approach to software development topics to Autodesk University. After taking a year off last year, he is anxious to jump back in this year! With .NET's continued adoption and the reincarnation of VBA in AutoCAD 2014, the number of topics to be discussed are huge!!!

Jerry started learning AutoCAD in 1988 and has enjoyed the fast-paced action and great variety found in the wonderful world of CAD since that time. After entering the workforce as a drafter, he found he could write a program to perform his job duties in a fraction of the time it took him to do them. Since making this discovery, he hasn't looked back as he has written AutoCAD add-ins and taught others to do the same.

As much as Jerry loves customizing AutoCAD, he is devoted to his wife and 9 (not a typo) children and loves spending high quality and quantity time with them in the marvelous rural community of Lake Point, Utah.

jerryw@vbcad.com

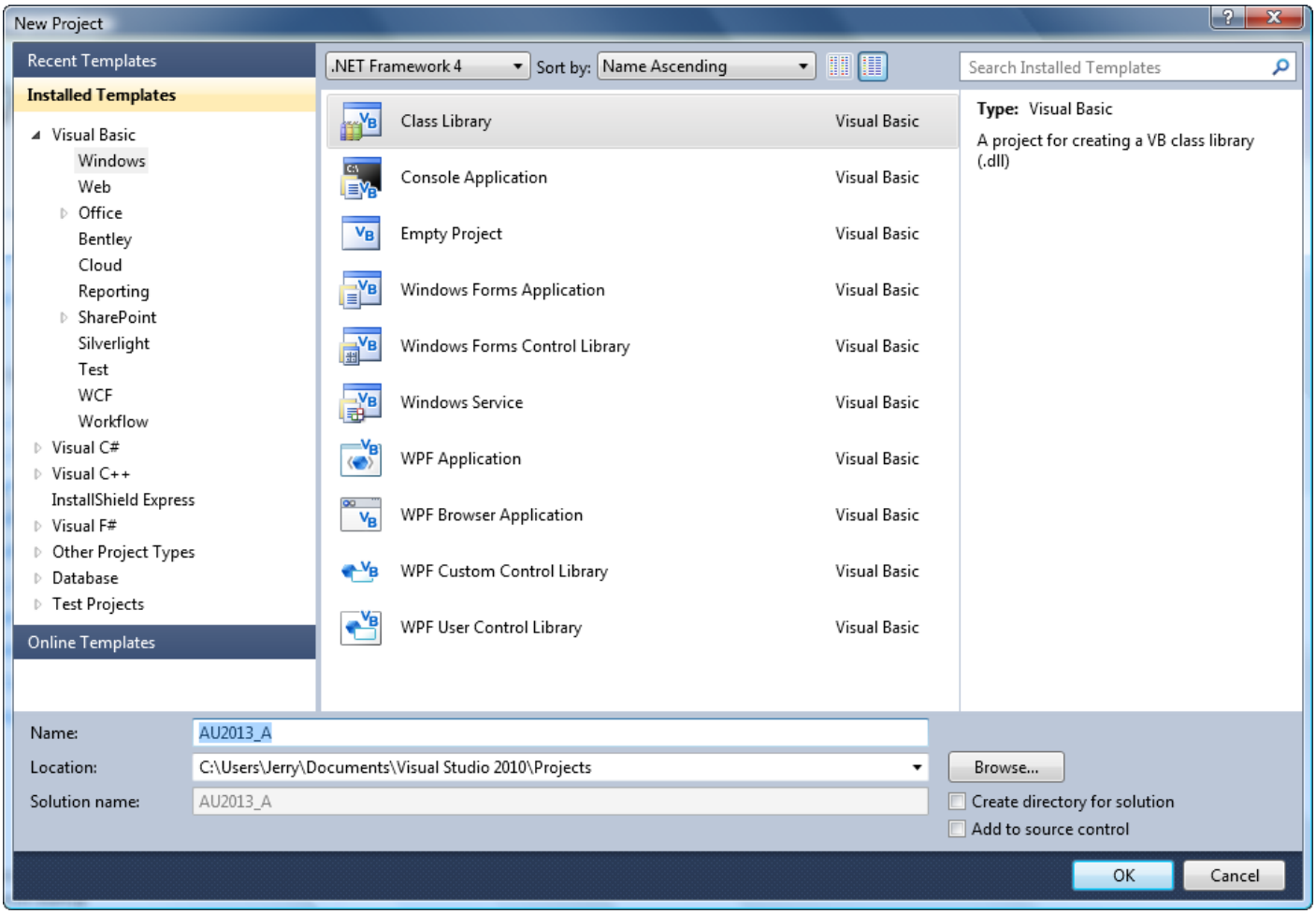
Create a new VB.NET Project, Add References, Create New Commands

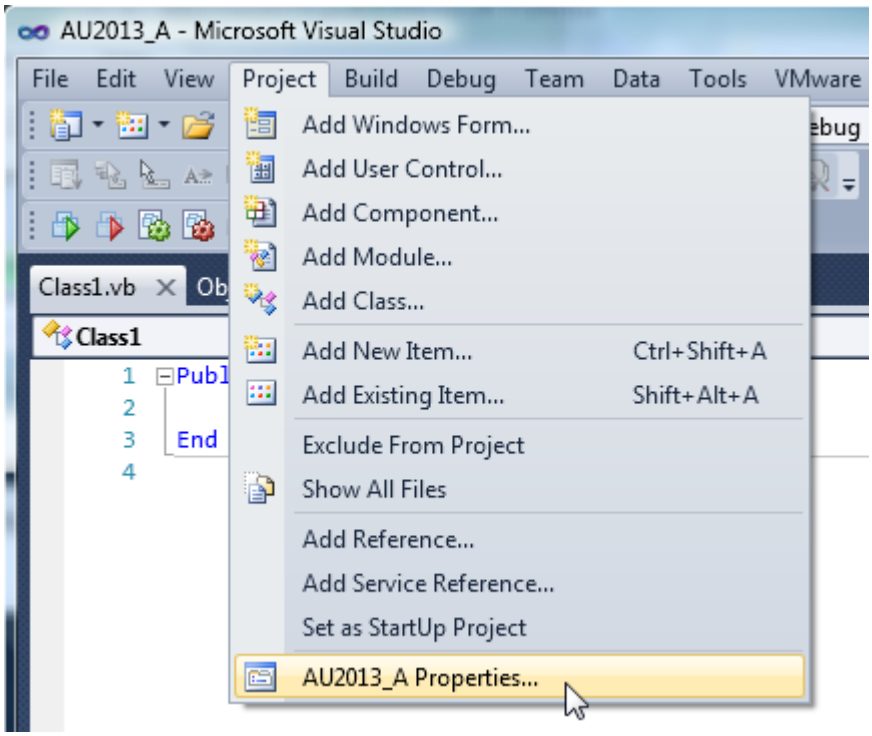
VB.NET is one of the languages that can be used to create AutoCAD Add-ins. Visual Studio's paid versions (including Visual Sudio Professional) can be used or Microsoft's Express Editions (free versions) can be used to create these Add-Ins. .NET Add-ins have been in use since AutoCAD 2007.

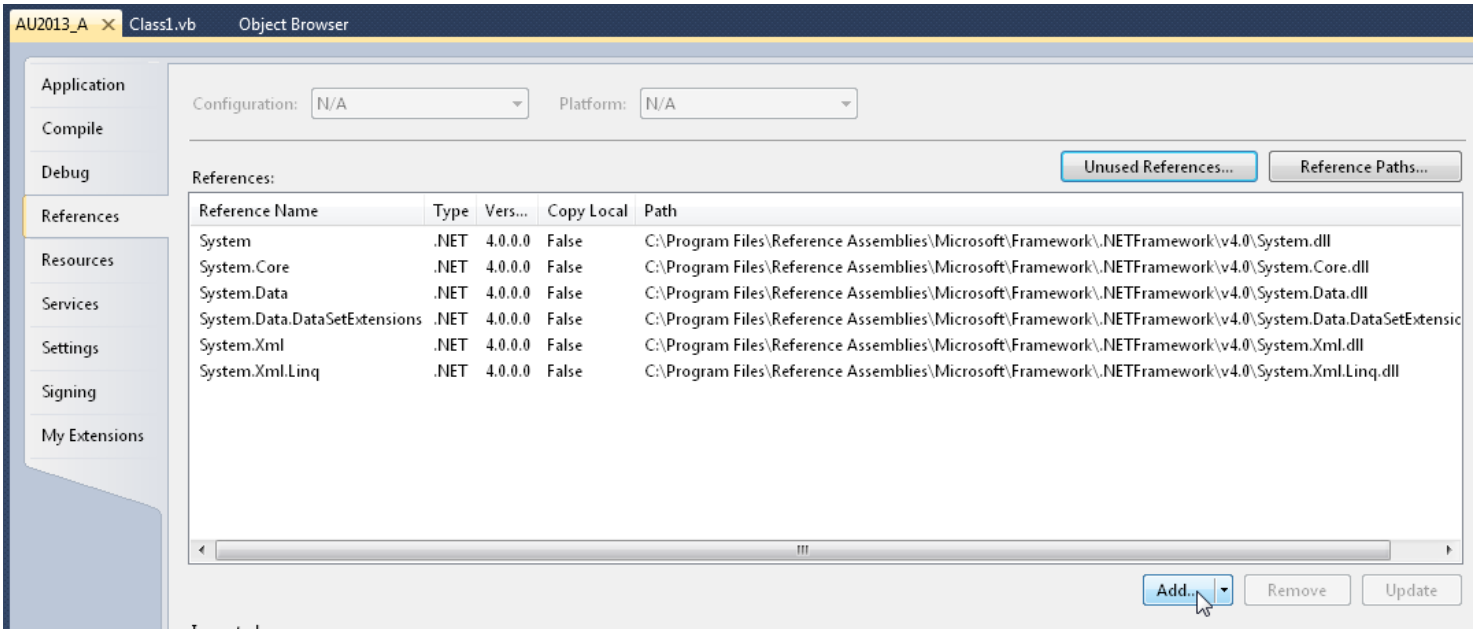
The process of creating a .NET Add-In for AutoCAD has been nearly the same since AutoCAD 2007's release. Here are the steps:

1. Create a new Class Library Project in Visual Studio.
2. Add References to the following:
   acmgd.dll
   acdbmgd.dll
   accoremgd.dll (For AutoCAD 2013 and later)
3. Set the .NET Project's Working Directory property to the same path as AutoCAD's acad.exe file.
4. Create a new Command
5. Compile the .dll

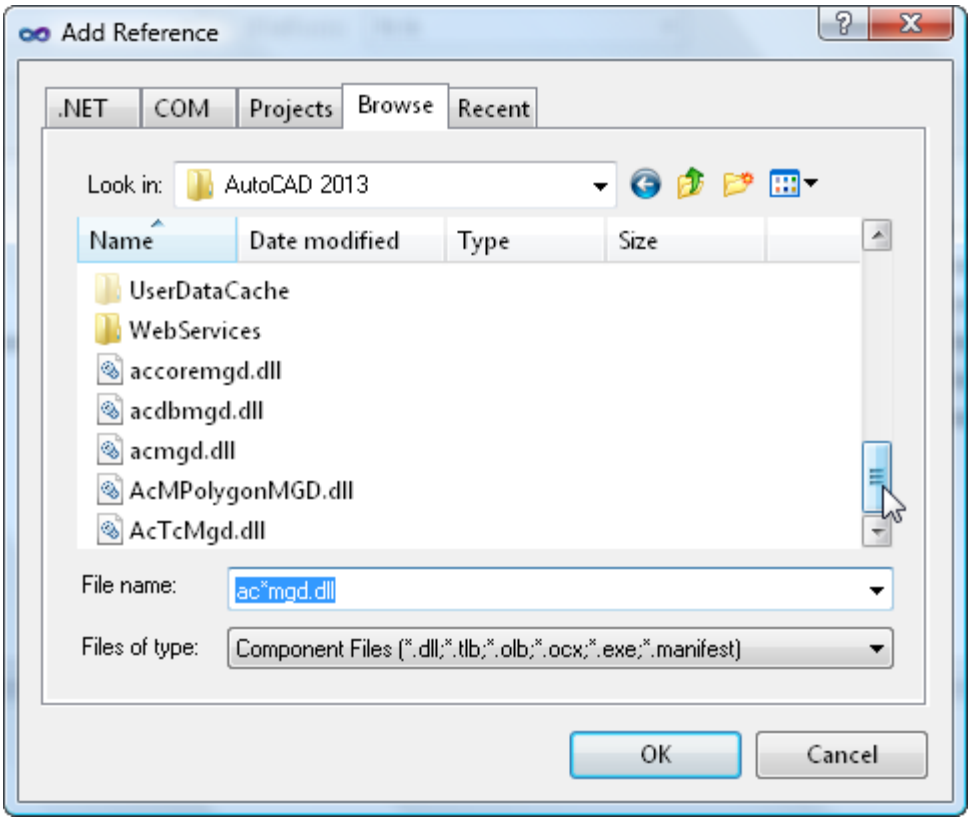
Let's see these one more time.

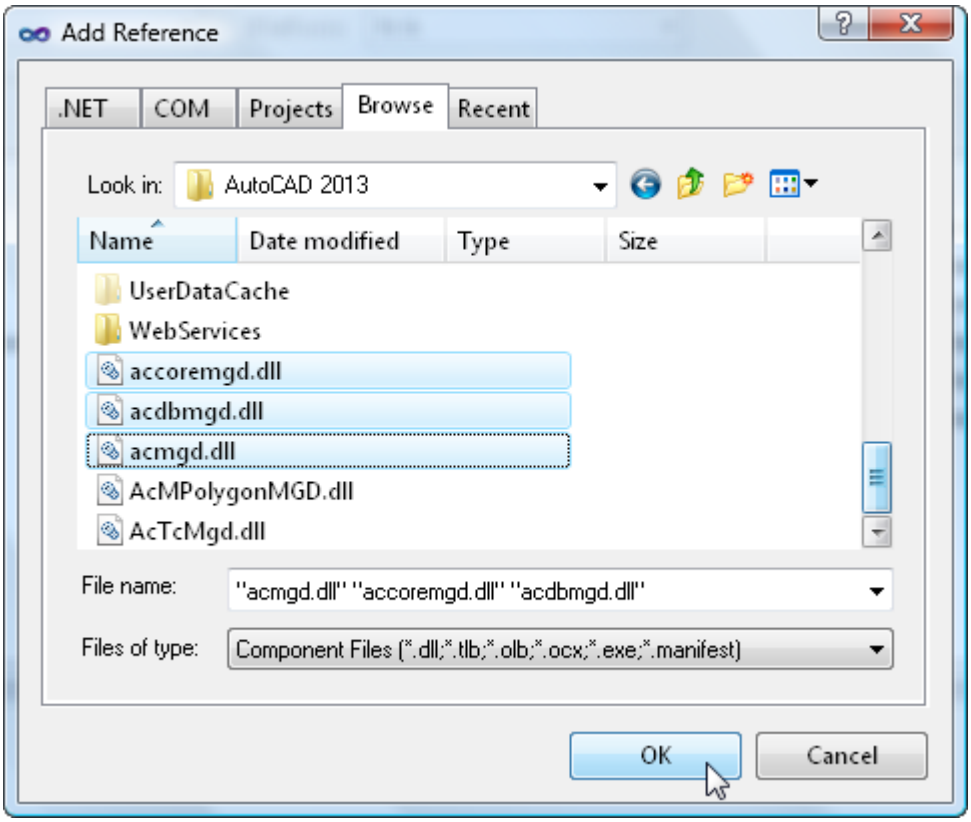


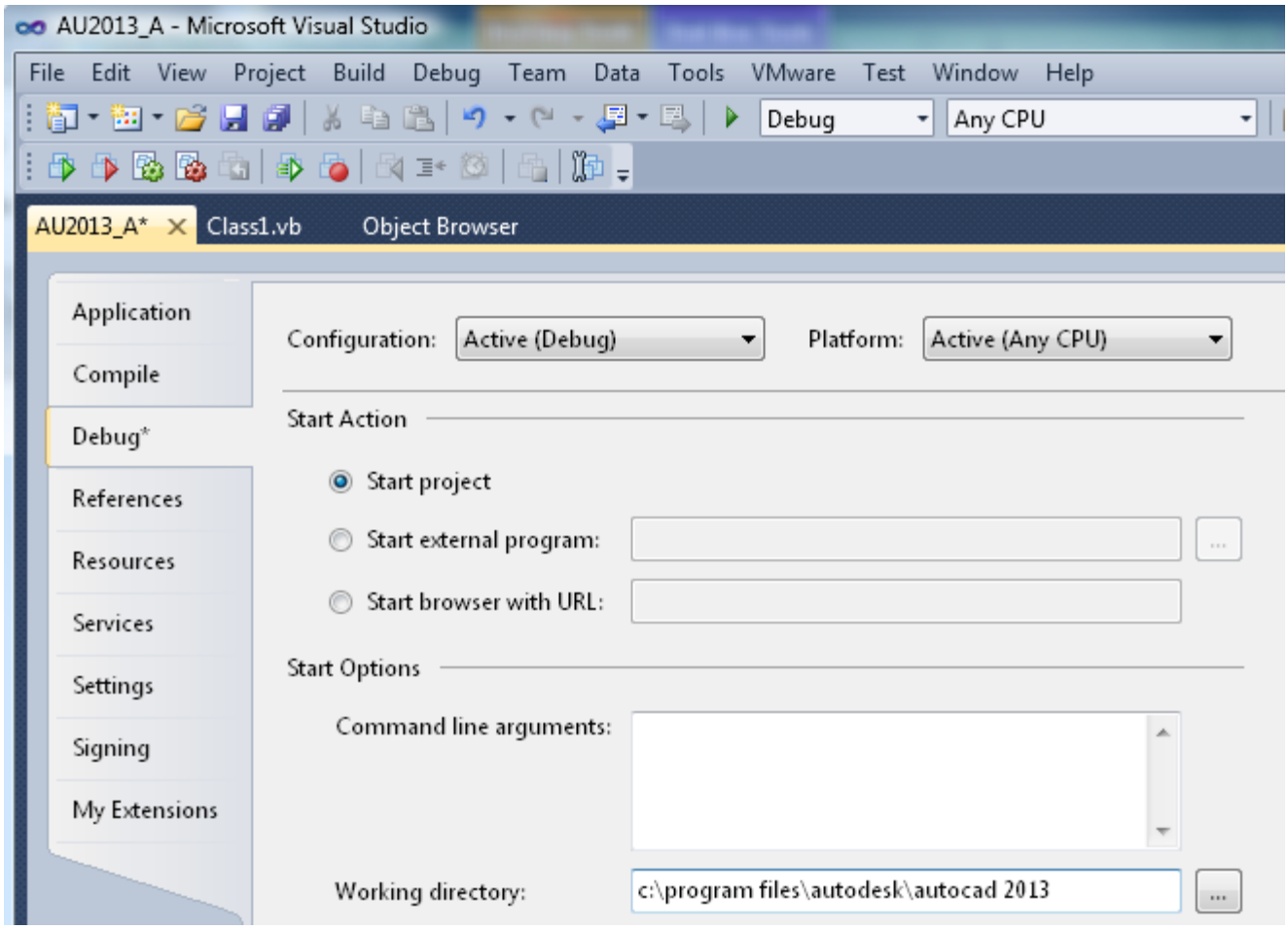

AU2013_A - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Tools VMware
Add Windows Form...
Add User Control...
Add Component...
Add Module...
Add Class...
Add New Item... Ctrl+Shift+A
Add Existing Item... Shift+Alt+A
Exclude From Project
Show All Files
Add Reference...
Add Service Reference...
Set as StartUp Project
AU2013_A Properties...
Class1.vb
Class1
1 Publ
2
3 End
4

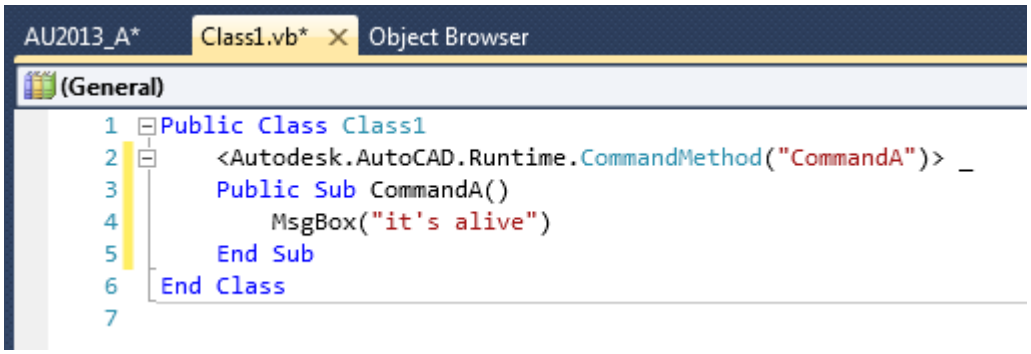

AU2013_A
Class1.vb
Object Browser
Application
Compile
Debug
References
Resources
Services
Settings
Signing
My Extensions
Configuration: N/A
Platform: N/A
References:
Unused References...
Reference Paths...
Reference Name Type Vers... Copy Local Path
System .NET 4.0.0.0 False C:\Program Files\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.0\System.dll
System.Core .NET 4.0.0.0 False C:\Program Files\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.0\System.Core.dll
System.Data .NET 4.0.0.0 False C:\Program Files\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.0\System.Data.dll
System.Data.DataSetExtensions .NET 4.0.0.0 False C:\Program Files\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.0\System.Data.DataSetExtensic
System.Xml .NET 4.0.0.0 False C:\Program Files\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.0\System.Xml.dll
System.Xml.Linq .NET 4.0.0.0 False C:\Program Files\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.0\System.Xml.Linq.dll
Add...
Remove
Update

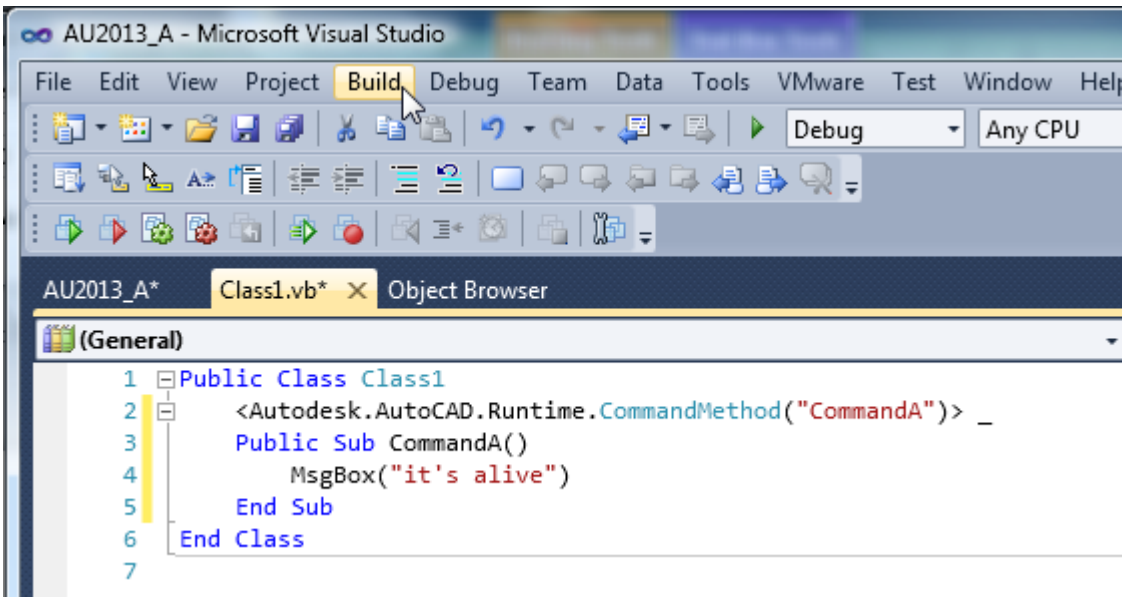
HINT: When adding these references, I filter on **ac*mgd.dll** to isolate the 3 dlls needed.

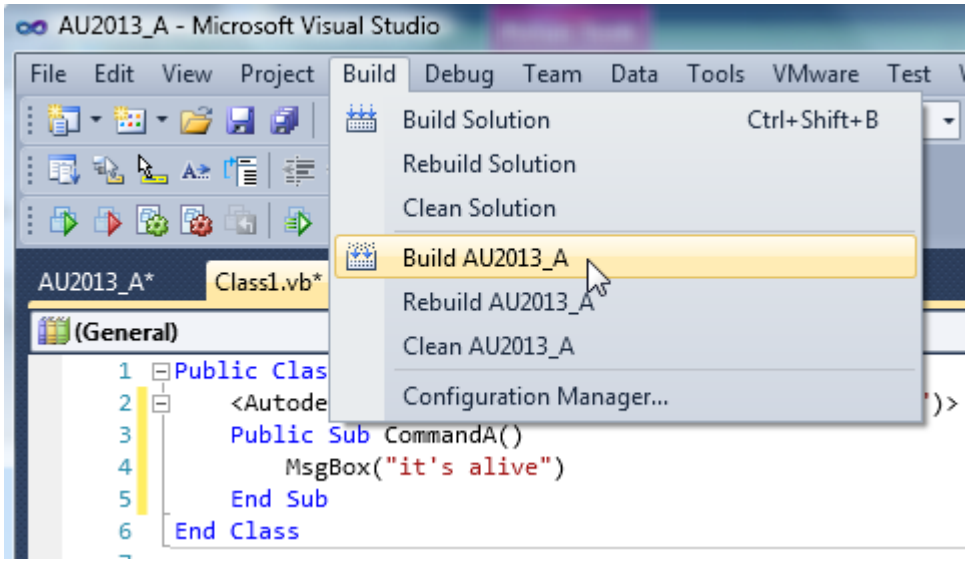





AU2013_A - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Tools VMware Test Window Help
Debug
Any CPU
AU2013_A*
Class1.vb
Object Browser
Application
Compile
Debug*
References
Resources
Services
Settings
Signing
My Extensions
Configuration: Active (Debug)
Platform: Active (Any CPU)
Start Action
Start project
Start external program:
Start browser with URL:
Start Options
Command line arguments:
Working directory: c:\program files\autodesk\autocad 2013

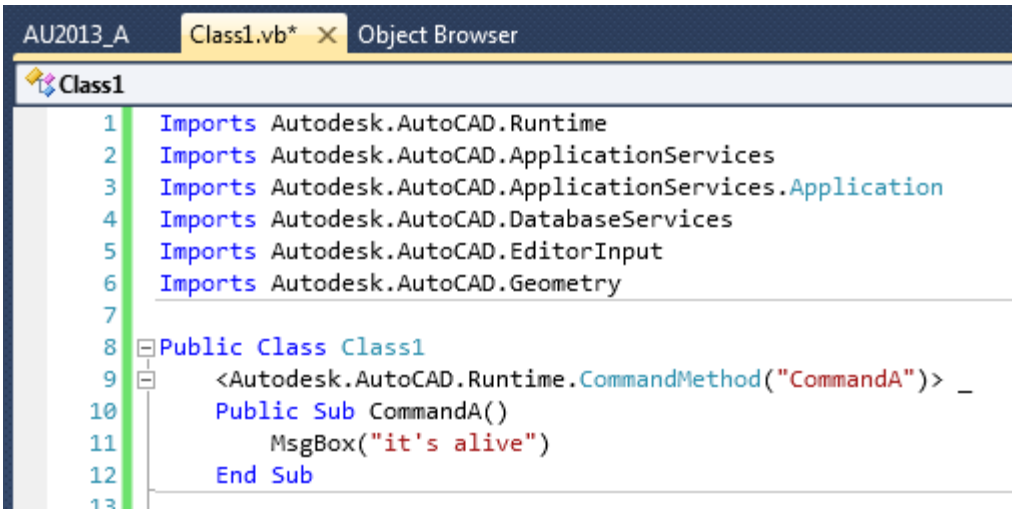

AU2013_A*
Class1.vb*
Object Browser
(General)
1 Public Class Class1
2 <Autodesk.AutoCAD.Runtime.CommandMethod("CommandA")> _
3 Public Sub CommandA()
4 MsgBox("it's alive")
5 End Sub
6 End Class
7





Once we hit the “Built” menu item, our project is compiled into a .dll file. This .dll file can then be netloaded into AutoCAD by using the **Netload** command.

Before we user Netload, let’s add some **Imports** statements in our project along with a few more commands.



***Notice how the Imports Statements appear above the Class code.

Here are a few more Commands to consider adding to your code. You may choose one that looks interesting or you may start at the top of the list and work your way down.

```
'Select Entity, display Layer in MessageBox
<CommandMethod("CommandB")> _
Public Sub CommandB()
    Dim myPER As PromptEntityResult = DocumentManager.MdiActiveDocument.Editor.GetEntity("Select:")
    If myPER.Status = PromptStatus.OK Then
        Using myTrans As Transaction = _
                DocumentManager.MdiActiveDocument.TransactionManager.StartTransaction
            Dim myEntity As Entity = myPER.ObjectId.GetObject(OpenMode.ForRead)
            MsgBox(myEntity.Layer)
            myTrans.Abort()
        End Using
    End If
End Sub

'Draw Line in ModelSpace
<CommandMethod("CommandC")> _
Public Sub CommandC()
    Dim myDB As Database = HostApplicationServices.WorkingDatabase
    Using myTrans As Transaction = myDB.TransactionManager.StartTransaction
        Dim myLine As New Line(Point3d.Origin, New Point3d(4, 5, 6))
        Dim mySpace As BlockTableRecord = myDB.CurrentSpaceId.GetObject(OpenMode.ForWrite)
        mySpace.AppendEntity(myLine)
        myTrans.AddNewlyCreatedDBObject(myLine, True)
        myTrans.Commit()
    End Using
End Sub
```

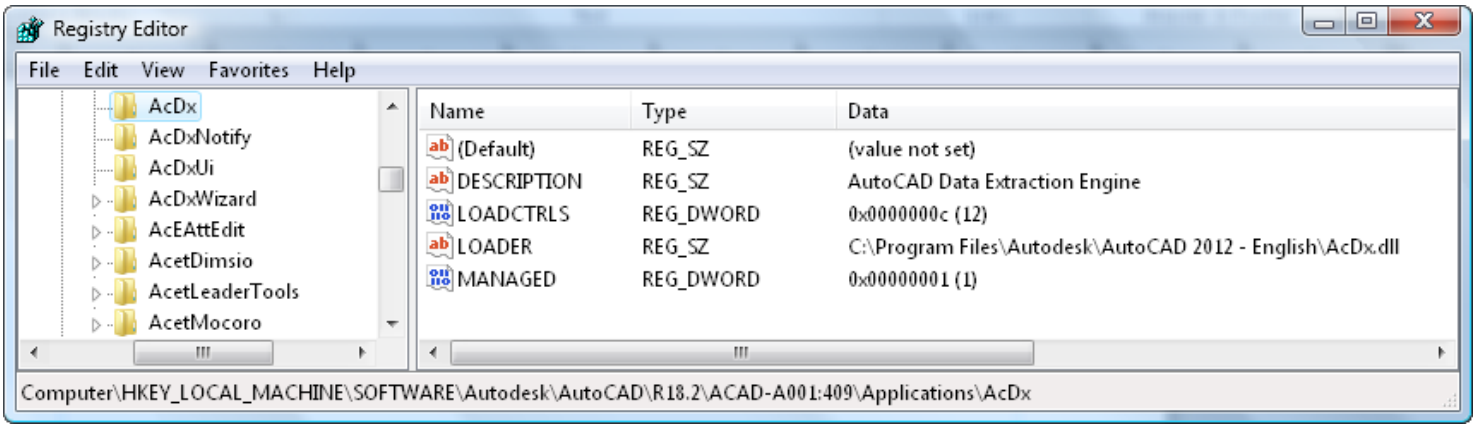
```
'Export Layers to Text File
<CommandMethod("CommandD")> _
Public Sub CommandD()
    Dim myDB As Database = HostApplicationServices.WorkingDatabase
    Dim myStreamWriter As New IO.StreamWriter("C:\temp\layers.txt")
    Using myTrans As Transaction = myDB.TransactionManager.StartTransaction
        Dim myLayerTable As LayerTable = myDB.LayerTableId.GetObject(OpenMode.ForRead)
        For Each myLayerID As ObjectId In myLayerTable
            Dim myLayer As LayerTableRecord = myLayerID.GetObject(OpenMode.ForRead)
            myStreamWriter.WriteLine(myLayer.Name)
        Next
        myTrans.Abort()
    End Using
    myStreamWriter.Close()
    myStreamWriter.Dispose()
End Sub

'Export BlockReferences to Text File
<CommandMethod("CommandE")> _
Public Sub CommandE()
    Dim myDB As Database = HostApplicationServices.WorkingDatabase
    Dim myStreamWriter As New IO.StreamWriter("C:\temp\blocks.txt")
    Using myTrans As Transaction = myDB.TransactionManager.StartTransaction
        Dim mySpace As BlockTableRecord = myDB.CurrentSpaceId.GetObject(OpenMode.ForRead)
        For Each myEntityID As ObjectId In mySpace
            If myEntityID.ObjectClass.DxfName = "INSERT" Then
                Dim myBlockRef As BlockReference = myEntityID.GetObject(OpenMode.ForRead)
                myStreamWriter.WriteLine(myBlockRef.Name & vbTab & myBlockRef.Position.ToString)
            End If
        Next
        myTrans.Abort()
    End Using
    myStreamWriter.Close()
    myStreamWriter.Dispose()
End Sub
```

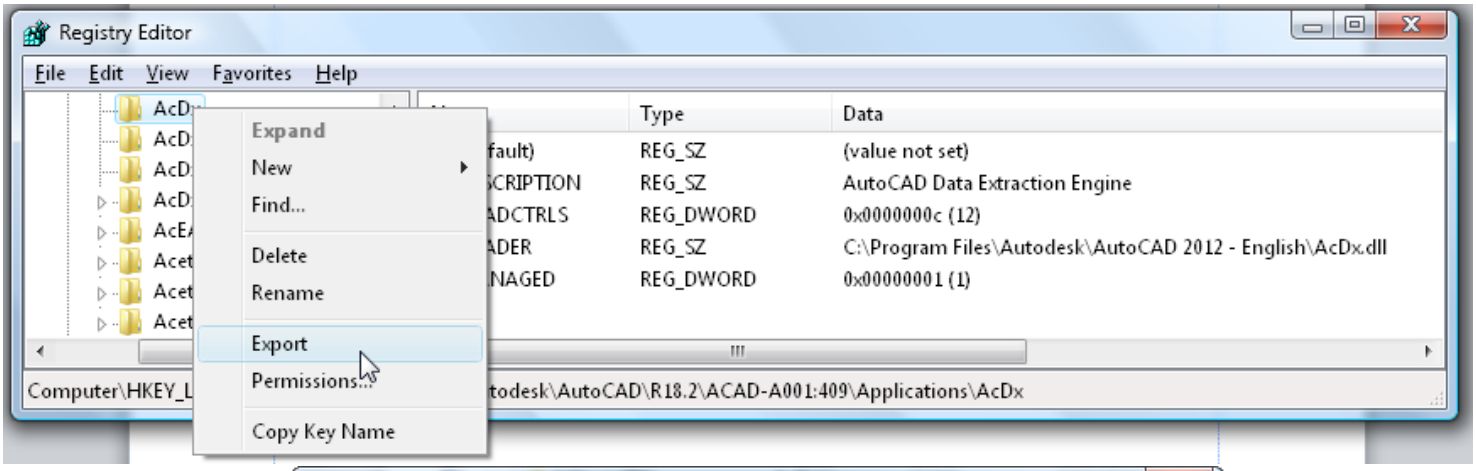
Package your Add-In so it will autoload

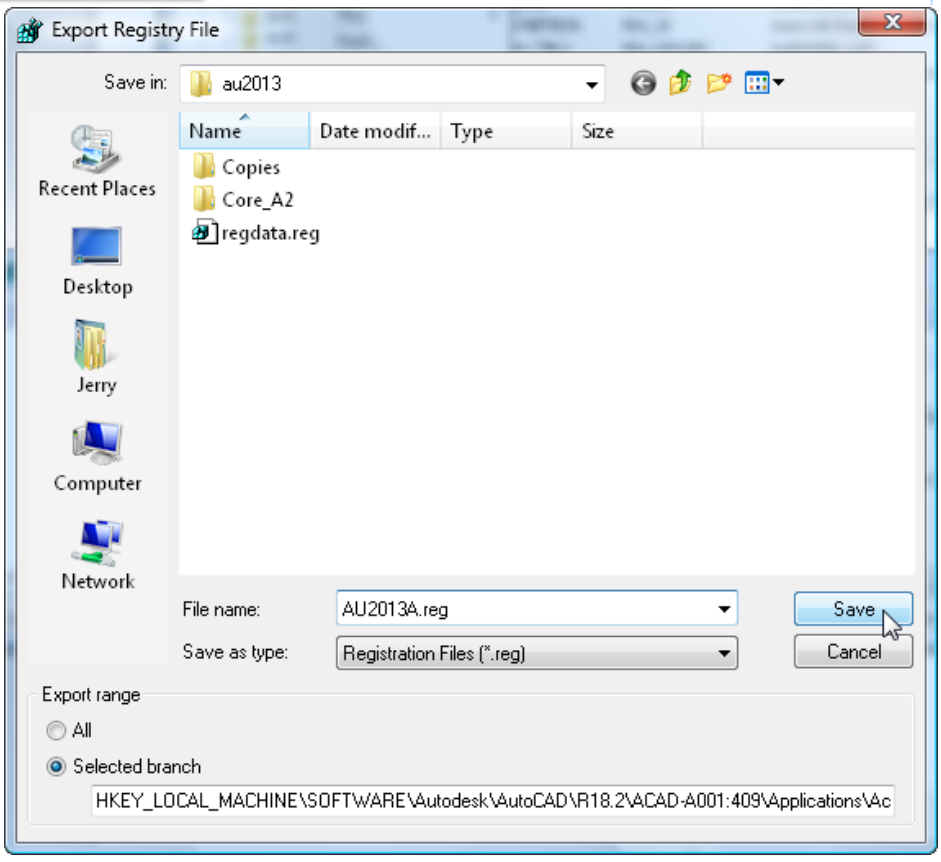
A new Autoload mechanism was introduced recently (within the last couple of years) that is, from the user's point of view, far more elegant than using the registry to autoload our add-ins. However, it would take more time than we currently have here at Autodesk University to explain it and give you a hands-on opportunity to put it into action. So we will use the Registry method for now.

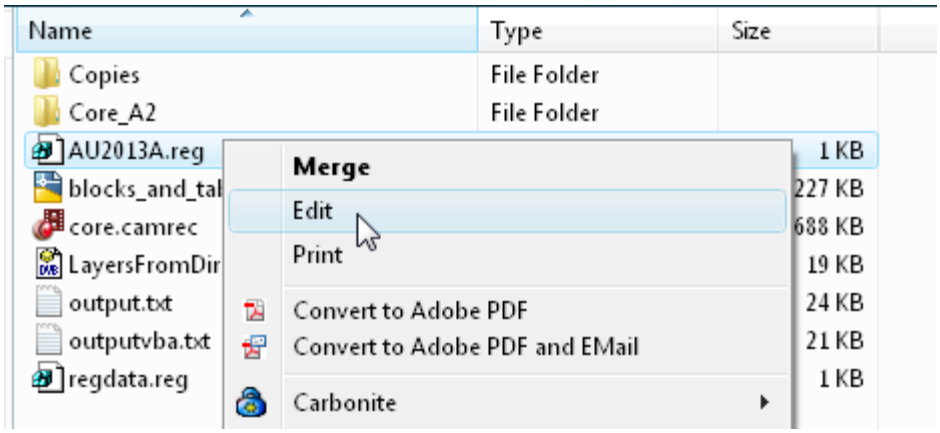
1. Go to the Applications entry under the current version of AutoCAD.
2. Find an existing entry where "Managed" is available.
3. Export the entry to a .reg file.
4. Edit and save the .reg file in Notepad to point to our .dll file.
5. Double-click on the .reg file in Windows Explorer to add the registry entries to the Registry.
6. Start AutoCAD and our Add-In should autoload.



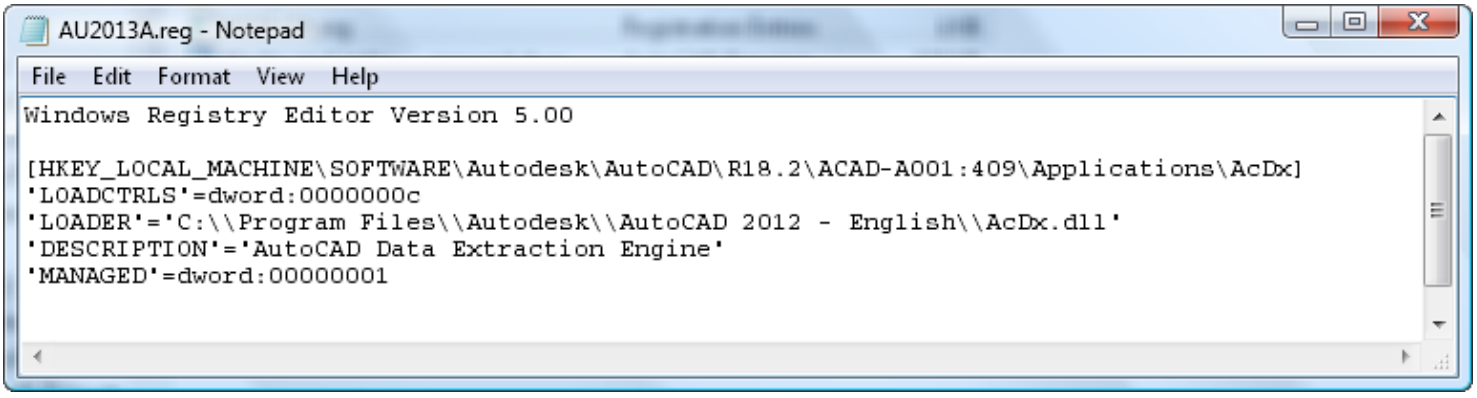
Right-click on the registry entry and select “Export”.



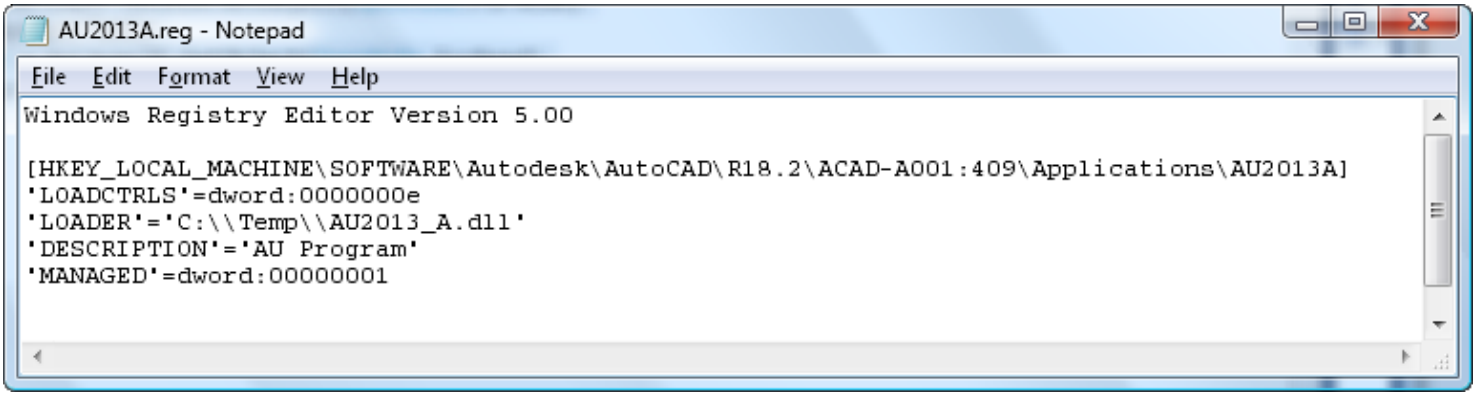




BEFORE



AFTER



Now, save the .reg file and browse for it in Windows Explorer. When you find it in Windows Explorer, double-click on it and accept the prompt that the data will be written to the Registry

Once a .dll is loaded using Netload or the Registry entry we just finished looking at, the commands defined in our Add-In can be run in AutoCAD.

REVIEW

Creating a new AutoCAD Add-In in VB.NET is simple and straight forward. It is, of course, necessary to follow each step precisely. However, once mastered, the Add-In creation process can be followed in a minute or two. The 'difficult' part is writing the code inside our newly defined commands. That's a lesson for another day. Actually, that's a lesson that can be learned throughout the rest of your life. So, start learning and enjoy.

Take care, my friends.

Jerry Winters
jerryw@vbcad.com