AUTODESK UNIVERSITY

SD125743

Writing Better, Stronger, More Stable .NET AutoCAD Add-Ins

Jerry Winters
VB CAD, Inc.

Learning Objectives

- Learn how to simplify, inherit, encapsulate, and expand
- Learn about when and how to use dictionaries, lists, and arrays
- Learn how to use all of the .NET Framework and be creative once in a while
- Learn how to identify and consolidate your code into these tidy little units

Description

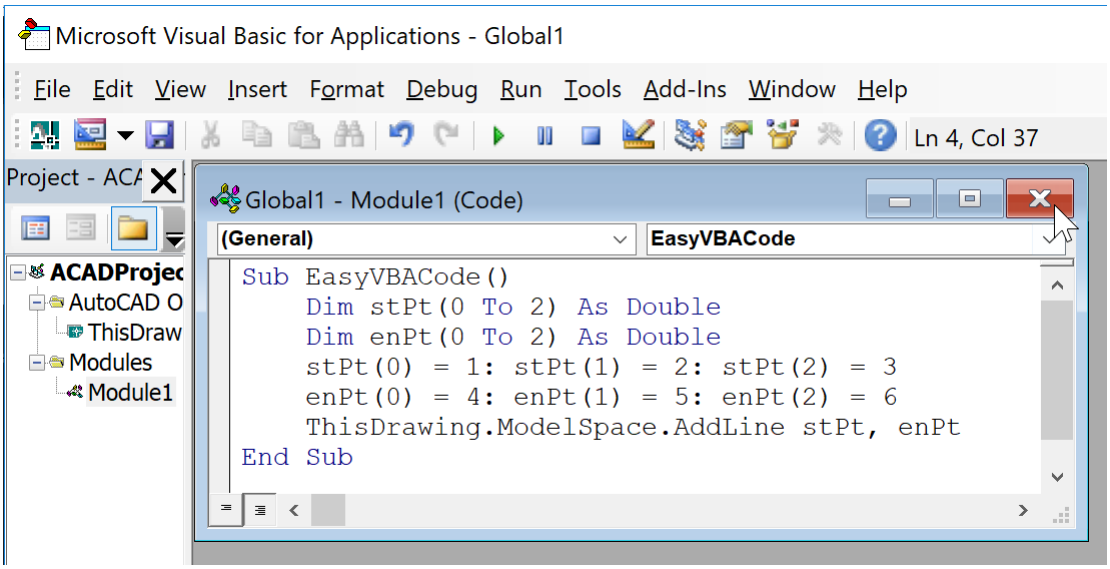
When we look back 10 years, we can easily identify how much we've learned and how much growth we've experienced. If only we knew 10 years ago what we know now, how much better would we be? This session hopes to provide the top lessons learned over the last 10 years of .NET development for AutoCAD software. Add your class description.

Speaker(s)

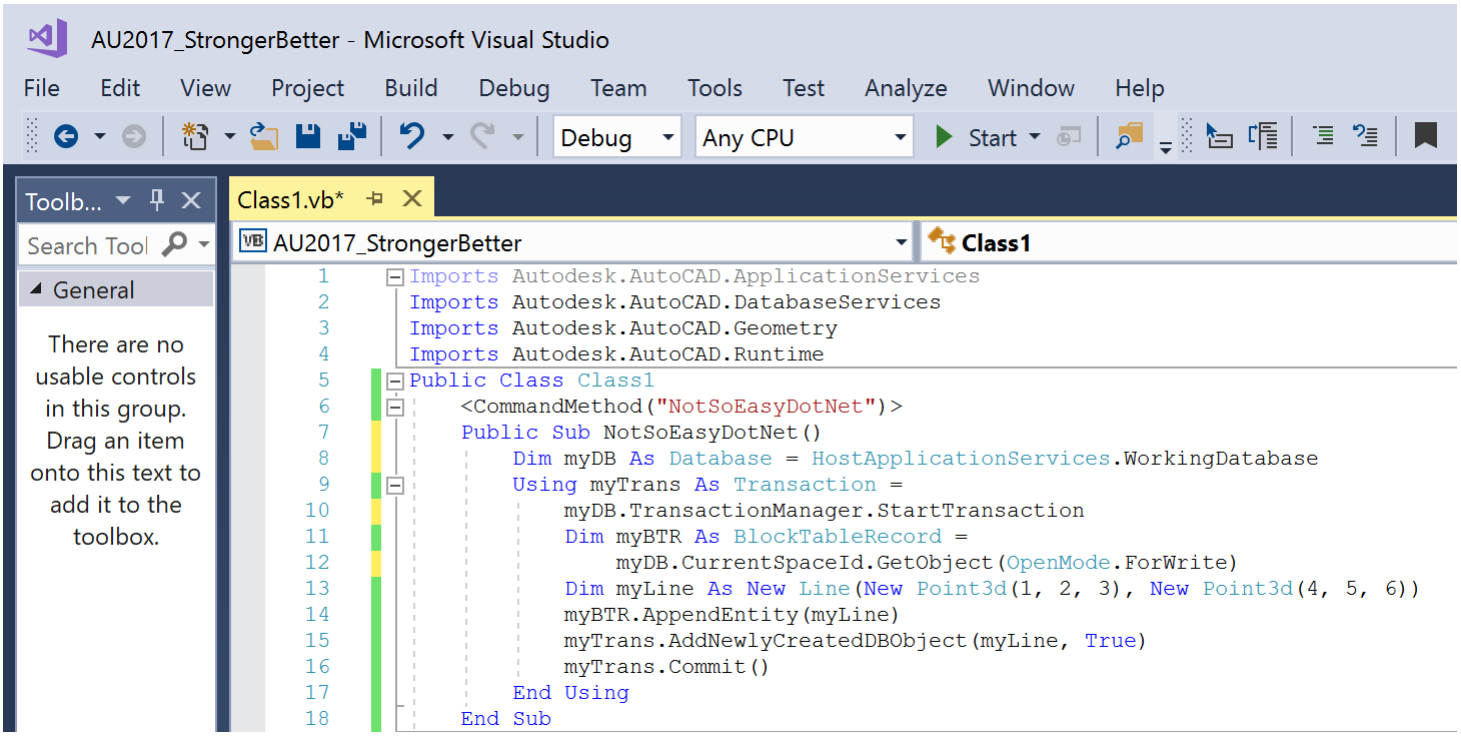
Jerry Winters has taught at nearly every U.S.-based Autodesk University over the past 17 years primarily on topics of AutoCAD software customization using Microsoft Visual Basic for Applications (VBA) and VB.NET. He brings a unique blend of education and entertainment as he tackles what would otherwise be a dry topic. And while he reserves the week before Thanksgiving for meeting with and enjoying the company of his Autodesk friends, the rest of the year is spent with his beautiful wife of 26 years and his 9 (yes, that's 9) children in rural Lake Point, Utah, where they are occasionally found singing at community events and senior centers.

Learn how to simplify, inherit, encapsulate, and expand

I loved VBA. One of the primary reasons was that it was simple.



Do you want to draw a line in ModelSpace? Use the AddLine method of the ModelSpace Class. Simple. What does VB.NET look like?



Typing that much code every time I want to draw a simple line in AutoCAD is far from ideal. I'd much rather use something like this:

```
<CommandMethod("AU2017_A")>
Public Sub AU2017_A()
    Dim myOID As ObjectId = DrawLine(HostApplicationServices.WorkingDatabase, 1, 2, 3, 4, 5, 6, "Electrical")
End Sub
```

How do we get there? By writing code, of course.

```
Function DrawLine(DBIn As Database, X1 As Double, Y1 As Double, Z1 As Double,
        X2 As Double, Y2 As Double, Z2 As Double, Optional layoutName As String = "") As ObjectId
    Dim activeTransaction As Transaction = Nothing
    Dim newTransaction As Transaction = Nothing
    If DBIn.TransactionManager.NumberOfActiveTransactions > 0 Then
        activeTransaction = DBIn.TransactionManager.TopTransaction
    Else
        newTransaction = DBIn.TransactionManager.StartTransaction
    End If
    Dim myBT As BlockTable = DBIn.BlockTableId.GetObject(OpenMode.ForRead)
    Dim drawToID As ObjectId = Nothing
    If layoutName = "" Then
        drawToID = myBT(BlockTableRecord.ModelSpace) 'Default
    Else
        drawToID = LayoutNameToBTRid(DBIn, layoutName)
        If drawToID.IsValid = False Then
            drawToID = myBT(BlockTableRecord.ModelSpace)
        End If
    End If
    Dim myBTR As BlockTableRecord = drawToID.GetObject(OpenMode.ForWrite)
    Dim myLine As New Line(New Point3d(X1, Y1, Z1), New Point3d(X2, Y2, Z2))
    myBTR.AppendEntity(myLine)
    If newTransaction IsNot Nothing Then
        newTransaction.AddNewlyCreatedDBObject(myLine, True)
        newTransaction.Commit()
        newTransaction.Dispose()
    Else
        activeTransaction.AddNewlyCreatedDBObject(myLine, True)
    End If
    Return myLine.ObjectId
End Function
```

That's a lot of code. I get it. But let's take a look at why it is designed the way it is.

```
Function DrawLine(DBIn As Database, X1 As Double, Y1 As Double, Z1 As Double,
        X2 As Double, Y2 As Double, Z2 As Double, Optional layoutName As String = "") As ObjectId
```

First Parameter: DBIn as Database—this allows us to add a line to any database we can supply. This means it can be used against the current database or against a DWG file that was read into a Database but without opening in AutoCAD. This can be used with Core Console applications because the Database is a parameter.

Params 2-7: Point parameters are individualized instead of having an array (as in VBA) which makes it easier to work with in many circumstances.

Param 8: LayoutName allows us to specify either "Model" for ModelSpace or the name of the PaperSpace Layout we want to draw into.

```
Dim activeTransaction As Transaction = Nothing
Dim newTransaction As Transaction = Nothing
If DBIn.TransactionManager.NumberOfActiveTransactions > 0 Then
    activeTransaction = DBIn.TransactionManager.TopTransaction
Else
    newTransaction = DBIn.TransactionManager.StartTransaction
End If
```

This code is not a recommendation for all situations. However, there are times when rather than always creating a new Transaction, using an existing Transaction makes more sense. So, in this example, we are using the TopTransaction if it exists. Otherwise we start a new Transaction.

```
Dim myBT As BlockTable = DBIn.BlockTableId.GetObject(OpenMode.ForRead)
Dim drawToID As ObjectId = Nothing
If layoutName = "" Then
    drawToID = myBT(BlockTableRecord.ModelSpace) 'Default
Else
    drawToID = LayoutNameToBTRid(DBIn, layoutName)
    If drawToID.IsValid = False Then
        drawToID = myBT(BlockTableRecord.ModelSpace)
    End If
End If
Dim myBTR As BlockTableRecord = drawToID.GetObject(OpenMode.ForWrite)
```

We want to be able to specify the Layout to draw in. If a specific layoutName is not provided, we draw in ModelSpace. Otherwise, we attempt to draw to the specified Layout. If the layout does not exist, we draw in ModelSpace. A Function named "LayoutNameToBTRid" was created to simplify determining the BlockTableRecord to draw to if a LayoutName is specified. Here's that code:

```
Public Function LayoutNameToBTRid(DBIn As Database, LayoutNameIn As String) As ObjectId
    Try
        Using myTrans As Transaction = DBIn.TransactionManager.StartTransaction
            Dim myLD As DBDictionary = DBIn.LayoutDictionaryId.GetObject(OpenMode.ForRead)
            For Each myEntry As DBDictionaryEntry In myLD
                If myEntry.Key.Equals(LayoutNameIn, StringComparison.CurrentCultureIgnoreCase) Then
                    Dim myLayout As Layout = myEntry.Value.GetObject(OpenMode.ForRead)
                    Return myLayout.BlockTableRecordId
                End If
            Next
            Return New ObjectId
        End Using
    Catch ex As System.Exception
        Return New ObjectId
    End Try
End Function
```

There are many ways to accomplish the same task. This is just one of them. Once again, we pass in the Layout's Database so this Function can be used in many ways.

OK. Back to our code:

```
    Dim myLine As New Line(New Point3d(X1, Y1, Z1), New Point3d(X2, Y2, Z2))
    myBTR.AppendEntity(myLine)
    If newTransaction IsNot Nothing Then
        newTransaction.AddNewlyCreatedDBObject(myLine, True)
        newTransaction.Commit()
        newTransaction.Dispose()
    Else
        activeTransaction.AddNewlyCreatedDBObject(myLine, True)
    End If
    Return myLine.ObjectId
End Function
```

We create the new Line Entity, append it to the BlockTableRecord, add it to the Transaction, and commit the Transaction ONLY if we created a new Transaction. If, however, we used a Transaction that was already open, we don't want to commit it because that could cause other problems where that open Transaction is being used.

We return the ObjectID of the Line because we can do a lot with an ObjectID.

- An ObjectID can be passed around without needing an active Transaction.
- The ObjectID has a Database property which allows us to, from the ObjectID, start a new Transaction to read and/or write to other elements in the ObjectID's database.

Let's not forget why we have all of that code. It's because we want to be able to use one line of code to add a line to a DWG file.

```
Dim myOID As ObjectId = DrawLine(HostApplicationServices.WorkingDatabase, 1, 2, 3, 4, 5, 6, "Electrical")
```

Simplifying our code means writing 'big code' once and using this simplified code any time we want to draw a line in all future applications. Think about it. 10 minutes or so to write a Function that allows us to write 1-liners for the next 10 years.

Encapsulation (computer programming)

From Wikipedia, the free encyclopedia

In programming languages, **encapsulation** is used to refer to one of two related but distinct notions, and sometimes to the combination[1][2] thereof:

- A language mechanism for restricting direct access to some of the object's components.[3][4]
- A language construct that facilitates the bundling of data with the methods (or other functions) operating on that data.[5][6]

Some programming language researchers and academics use the first meaning alone or in combination with the second as a distinguishing feature of object-oriented programming, while some programming languages which provide lexical closures view encapsulation as a feature of the language orthogonal to object orientation.

The second definition is motivated by the fact that in many of the OOP languages hiding of components is not automatic or can be overridden; thus, information hiding is defined as a separate notion by those who prefer the second definition.

The features of encapsulation are supported using classes in most object-oriented programming languages, although other alternatives also exist.

As this Wikipedia entry describes, Encapsulation is defined differently by different people. The first definition talks about 'restricting direct access'. That sounds like a security matter but encapsulation is also a way to 'hide' complexity. It goes hand in hand with simplifying our code.

Inheritance

Implementing Inheritance does not necessarily make our applications more powerful. It can, however, help us as we create more powerful applications. It takes some planning as we will see.

We want to create our own Classes to be used anytime we want to work with Lines, Circles, Arcs, etc. We do this by identifying properties all of these entities have in common.

All entities have an ObjectID.

All entities have a Layer.

All entities have a Color.

All entities have a Handle.

There are other properties shared by all entities but we will stick with these

vbcEntity.vb* | AU2017_StrongerBetter | vbcEntity | pLayer

```vb
Imports Autodesk.AutoCAD.DatabaseServices
Public Class vbcEntity
    'Private Variables for Properties
    Private pLayer As String = "0"
    Private pColor As Autodesk.AutoCAD.Colors.Color
    Public Property eObjectID As ObjectId = Nothing
    Public Property eHandle As String = ""

    Public Property Layer As String
        Get
            Return pLayer
        End Get
        Set(value As String)
            SetLayer(eObjectID, value)
        End Set
    End Property
    Public Property Color As Autodesk.AutoCAD.Colors.Color
        Get
            Return pColor
        End Get
        Set(value As Autodesk.AutoCAD.Colors.Color)
            SetColor(eObjectID, value)
        End Set
    End Property
End Class
```

The name of the Class is “vbcEntity”. We have implemented the ObjectID, Handle, Layer, and Color properties. Two of these (ObjectID and Handle) can be used by simply declaring them as Properties. The Color and Layer, however, need to be implemented with custom ‘Set’s.

```vb
Public Function SetLayer(ObjectIdIn As ObjectId, LayerName As String) As Boolean
    Dim retVal As Boolean = False
    Try
        Using myTrans As Transaction = ObjectIdIn.Database.TransactionManager.StartTransaction
            Dim myEnt As Entity = ObjectIdIn.GetObject(OpenMode.ForWrite)
            Dim myLT As LayerTable = ObjectIdIn.Database.LayerTableId.GetObject(OpenMode.ForWrite)
            If myLT.Has(LayerName) = False Then
                Dim newLTR As New LayerTableRecord()
                newLTR.Name = LayerName
                myLT.Add(newLTR)
                myTrans.AddNewlyCreatedDBObject(newLTR, True)
                retVal = True
            End If
            myEnt.Layer = LayerName
            myTrans.Commit()
        End Using
    Catch ex As System.Exception
    End Try
    Return retVal
End Function
```

The SetLayer Function is used to set an Entity’s Layer. If the specified Layer does not exist, we want to create it. This allows us to create Entities and specify their Layer as a Property while Encapsulating all of the code required into a simple call that requires an ObjectID and a LayerName.

```
Public Sub SetColor(ObjectIdIn As ObjectId, ColorIn As Autodesk.AutoCAD.Colors.Color)
    Dim retVal As Boolean = False
    Try
        Using myTrans As Transaction = ObjectIdIn.Database.TransactionManager.StartTransaction
            Dim myEnt As Entity = ObjectIdIn.GetObject(OpenMode.ForWrite)
            myEnt.Color = ColorIn
            myTrans.Commit()
        End Using
    Catch ex As System.Exception
    End Try
End Sub
```

SetColor is used to set an Entity's Color. It's fairly straight forward. Though it is not as 'complex' as SetLayer, it still allows us to set an Entity's Color with one line of code when we use it.

Let's take another look at the Full vbcEntity Class:

```
Imports Autodesk.AutoCAD.DatabaseServices
Public Class vbcEntity
    'Private Variables for Properties
    Private pLayer As String = "0"
    Private pColor As Autodesk.AutoCAD.Colors.Color
    Public Property eObjectID As ObjectId = Nothing
    Public Property eHandle As String = ""

    Public Property Layer As String
        Get
            Return pLayer
        End Get
        Set(value As String)
            SetLayer(eObjectID, value)
        End Set
    End Property
    Public Property Color As Autodesk.AutoCAD.Colors.Color
        Get
            Return pColor
        End Get
        Set(value As Autodesk.AutoCAD.Colors.Color)
            SetColor(eObjectID, value)
        End Set
    End Property
End Class
```

Now let's take a look at the vbcLine Class:

```
 1  Imports Autodesk.AutoCAD.DatabaseServices
 2  Imports Autodesk.AutoCAD.Geometry
 3  Public Class vbcLine
 4      Inherits vbcEntity
 5
 6      Public Sub New(DBIn As Database, X1 As Double, Y1 As Double, Z1 As Double,
11
12      Public Sub New(dbin As Database, BlockName As String, X1 As Double, Y1 As
17
18      Public Property StartPoint As Point3d ...
33
34      Public Property EndPoint As Point3d ...
49
50      Public Property MidPoint As Point3d ...
67  End Class
```

Notice the "Inherits vbcEntity" line of code. This means that not only does the vbcLine have a StartPoint, EndPoint, and MidPoint property, but it also has the 4 properties of the vbcEntity.

Let's take a look at the full code of the vbcLine Class:

```
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry
Public Class vbcLine
    Inherits vbcEntity

    Public Sub New(DBIn As Database, X1 As Double, Y1 As Double, Z1 As Double,
                   X2 As Double, Y2 As Double, Z2 As Double, Optional layoutName As String = "")
        eObjectID = DrawLine(DBIn, X1, Y1, Z1, X2, Y2, Z2, layoutName)
        eHandle = eObjectID.Handle.ToString
    End Sub

    Public Sub New(dbin As Database, BlockName As String, X1 As Double, Y1 As Double, Z1 As Double,
                   X2 As Double, Y2 As Double, Z2 As Double)
        eObjectID = DrawLine(dbin, BlockName, X1, Y1, Z1, X2, Y2, Z2)
        eHandle = eObjectID.Handle.ToString
    End Sub

    Public Property StartPoint As Point3d
        Get
            Using myTrans As Transaction = eObjectID.Database.TransactionManager.StartTransaction
                Dim myLine As Line = eObjectID.GetObject(OpenMode.ForRead)
                Return (myLine.StartPoint)
            End Using
        End Get
        Set(value As Point3d)
            Using myTrans As Transaction = eObjectID.Database.TransactionManager.StartTransaction
                Dim myLine As Line = eObjectID.GetObject(OpenMode.ForWrite)
                myLine.StartPoint = value
                myTrans.Commit()
            End Using
        End Set
    End Property

    Public Property EndPoint As Point3d
        Get
            Using myTrans As Transaction = eObjectID.Database.TransactionManager.StartTransaction
                Dim myLine As Line = eObjectID.GetObject(OpenMode.ForRead)
                Return (myLine.EndPoint)
            End Using
        End Get
        Set(value As Point3d)
            Using myTrans As Transaction = eObjectID.Database.TransactionManager.StartTransaction
                Dim myLine As Line = eObjectID.GetObject(OpenMode.ForWrite)
                myLine.EndPoint = value
                myTrans.Commit()
            End Using
        End Set
    End Property

    Public Property MidPoint As Point3d
        Get
            Using myTrans As Transaction = eObjectID.Database.TransactionManager.StartTransaction
                Dim myLine As Line = eObjectID.GetObject(OpenMode.ForRead)
                Dim myLS As New LineSegment3d(myLine.StartPoint, myLine.EndPoint)
                Return (myLS.MidPoint)
            End Using
        End Get
        Set(value As Point3d)
            Using myTrans As Transaction = eObjectID.Database.TransactionManager.StartTransaction
                Dim myLine As Line = eObjectID.GetObject(OpenMode.ForWrite)
                Dim myLS As New LineSegment3d(myLine.StartPoint, myLine.EndPoint)
                myLine.TransformBy(Matrix3d.Displacement(myLS.MidPoint.GetVectorTo(value)))
                myTrans.Commit()
            End Using
        End Set
    End Property
End Class
```

You may have noticed there are 2 ways to create our vbcLine.

```
Public Sub New(DBIn As Database, X1 As Double, Y1 As Double, Z1 As Double,
               X2 As Double, Y2 As Double, Z2 As Double, Optional layoutName As String = "")
    eObjectID = DrawLine(DBIn, X1, Y1, Z1, X2, Y2, Z2, layoutName)
    eHandle = eObjectID.Handle.ToString
End Sub
```

```
Public Sub New(dbin As Database, BlockName As String, X1 As Double, Y1 As Double, Z1 As Double,
               X2 As Double, Y2 As Double, Z2 As Double)
    eObjectID = DrawLine(dbin, BlockName, X1, Y1, Z1, X2, Y2, Z2)
    eHandle = eObjectID.Handle.ToString
End Sub
```

The first 'New' takes the Database, XYZ Coordinates, and allows us to optionally specify a Layout Name.

The second 'New' takes the Database, BlockName, and XYZ Coordinates. Why the difference?

If we want to draw a Line into a Block Definition that can be inserted as a Block Reference, it needs to be created differently than if we are drawing in ModelSpace. In this example, if a BlockName is given and that Block does not currently exist in the Database specified, we need to create the Block Definition (BlockTableRecord).

The first 'New' uses the DrawLine Function we already looked at. Let's take a look at the second 'New's DrawLine Function:

```
Function DrawLine(dbin As Database, BlockName As String, X1 As Double, Y1 As Double, Z1 As Double,
                  X2 As Double, Y2 As Double, Z2 As Double) As ObjectId
    Using myTrans As Transaction = dbin.TransactionManager.StartTransaction
        Dim myBT As BlockTable = dbin.BlockTableId.GetObject(OpenMode.ForRead)
        Dim myBTR As BlockTableRecord = BlockNameToBTRid(dbin, BlockName).GetObject(OpenMode.ForWrite)
        Dim myLine As New Line(New Point3d(X1, Y1, Z1), New Point3d(X2, Y2, Z2))
        myBTR.AppendEntity(myLine)
        If myTrans IsNot Nothing Then
            myTrans.AddNewlyCreatedDBObject(myLine, True)
            myTrans.Commit()
        Else
            myTrans.AddNewlyCreatedDBObject(myLine, True)
        End If
        Return myLine.ObjectId
    End Using
End Function
```

```
Public Function BlockNameToBTRid(dbin As Database, BlockName As String) As ObjectId
    Using myTrans As Transaction = dbin.TransactionManager.StartTransaction
        Dim myBT As BlockTable = dbin.BlockTableId.GetObject(OpenMode.ForRead)
        Dim myBTR As BlockTableRecord
        If myBT.Has(BlockName) Then
            Return myBT(BlockName)
        Else
            myBT.UpgradeOpen()
            myBTR = New BlockTableRecord()
            myBTR.Name = BlockName
            myBT.Add(myBTR)
            myTrans.AddNewlyCreatedDBObject(myBTR, True)
            myTrans.Commit()
            Return myBTR.ObjectId
        End If

    End Using
End Function
```

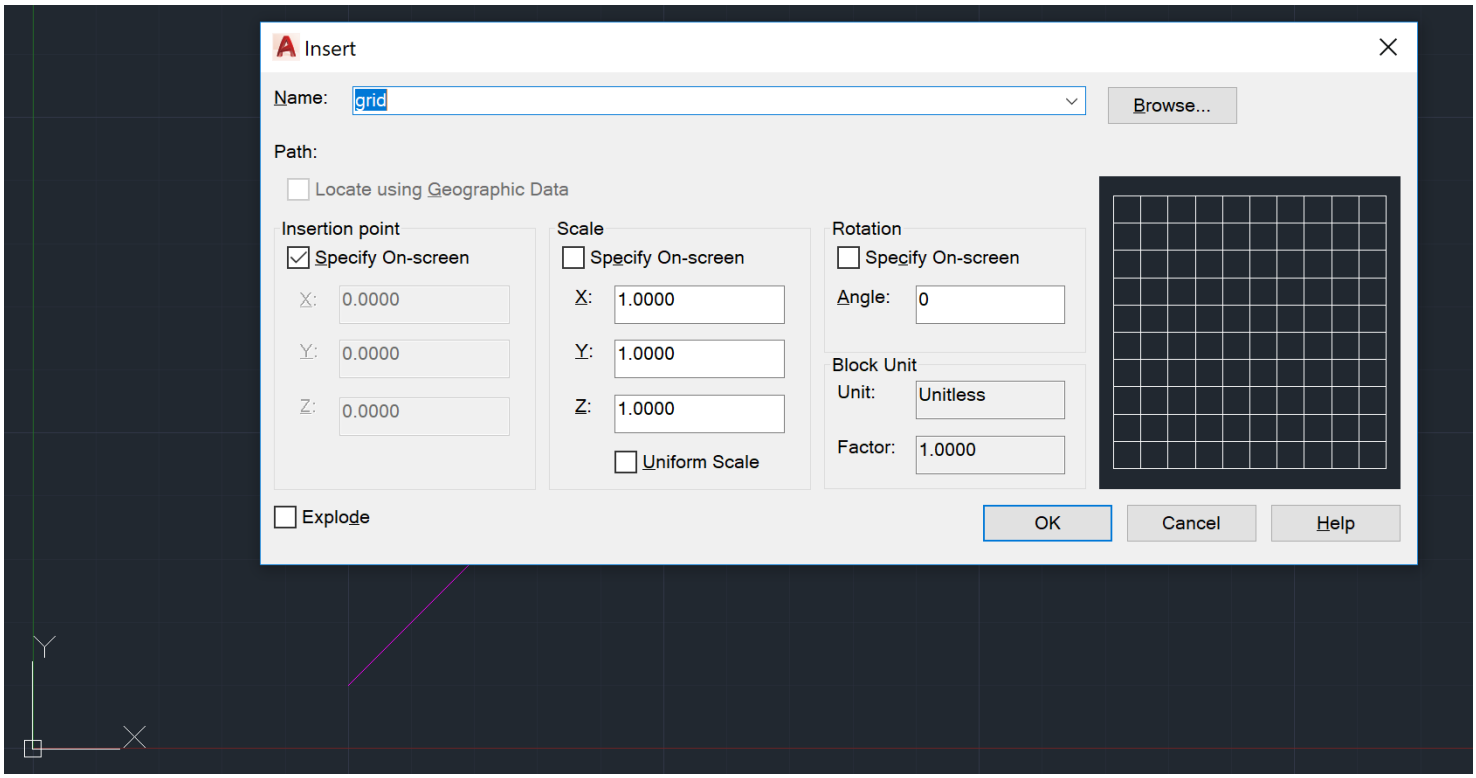
The BlockNameToBTRid Function is used to return the ObjectID of a BlockTableRecord if it exists. Otherwise, it creates a new BlockTableRecord and returns its ObjectID.

Let's take a look at a little code that makes use of the vbcLine Class now.

```
<CommandMethod("AU2017_B")>
Public Sub AU2017_B()
    Dim myLine As New vbcLine(HostApplicationServices.WorkingDatabase, 1, 2, 3, 4, 5, 6, "Electrical")
    myLine.Layer = "Ground"
    myLine.MidPoint = New Point3d(4, 2, 0)
    myLine.Color = Autodesk.AutoCAD.Colors.Color.FromRgb(255, 0, 255)
    For X As Double = 0 To 10
        For Y As Double = 0 To 10
            myLine = New vbcLine(HostApplicationServices.WorkingDatabase, "grid", X, 0, 0, X, 10, 0)
            myLine = New vbcLine(HostApplicationServices.WorkingDatabase, "grid", 0, Y, 0, 10, Y, 0)
        Next
    Next
End Sub
```

Here we can see the vbcLine Class being created both ways we just discussed. In the first use, the Line is placed on the "Electrical" Layout if that Layout exists. If the Layout does not exist, it is placed in ModelSpace. We also set the Layer Property to "Ground". Please ignore the MidPoint property for now. We will get to that in a minute.

Inside the For I . . . Next loop, we can see that each Line created is placed in a Block named "grid".



We can see the Line created in ModelSpace and the "grid" block drawn with the For I . . . Next loops.

Let's talk about the MidPoint Property of the vbcLine Object.

In the AutoCAD .NET API, a Line does not have a MidPoint Property. We want to Extend/Expand our vbcLine Class to include this property. To add to that, the MidPoint property of most objects is a read-only property. We want to be able to 'move' our vbcLine to a specified MidPoint.

Let's take a look at the StartPoint, EndPoint, and MidPoint properties we have implemented:

```
Public Property StartPoint As Point3d
    Get
        Using myTrans As Transaction = eObjectID.Database.TransactionManager.StartTransaction
            Dim myLine As Line = eObjectID.GetObject(OpenMode.ForRead)
            Return (myLine.StartPoint)
        End Using
    End Get
    Set(value As Point3d)
        Using myTrans As Transaction = eObjectID.Database.TransactionManager.StartTransaction
            Dim myLine As Line = eObjectID.GetObject(OpenMode.ForWrite)
            myLine.StartPoint = value
            myTrans.Commit()
        End Using
    End Set
End Property
```

```
Public Property EndPoint As Point3d
    Get
        Using myTrans As Transaction = eObjectID.Database.TransactionManager.StartTransaction
            Dim myLine As Line = eObjectID.GetObject(OpenMode.ForRead)
            Return (myLine.EndPoint)
        End Using
    End Get
    Set(value As Point3d)
        Using myTrans As Transaction = eObjectID.Database.TransactionManager.StartTransaction
            Dim myLine As Line = eObjectID.GetObject(OpenMode.ForWrite)
            myLine.EndPoint = value
            myTrans.Commit()
        End Using
    End Set
End Property
```

```
Public Property MidPoint As Point3d
    Get
        Using myTrans As Transaction = eObjectID.Database.TransactionManager.StartTransaction
            Dim myLine As Line = eObjectID.GetObject(OpenMode.ForRead)
            Dim myLS As New LineSegment3d(myLine.StartPoint, myLine.EndPoint)
            Return (myLS.MidPoint)
        End Using
    End Get
    Set(value As Point3d)
        Using myTrans As Transaction = eObjectID.Database.TransactionManager.StartTransaction
            Dim myLine As Line = eObjectID.GetObject(OpenMode.ForWrite)
            Dim myLS As New LineSegment3d(myLine.StartPoint, myLine.EndPoint)
            myLine.TransformBy(Matrix3d.Displacement(myLS.MidPoint.GetVectorTo(value)))
            myTrans.Commit()
        End Using
    End Set
End Property
```

Extend / Expand

StartPoint and MidPoint are easy. The MidPoint is a little different. We use the LineSegment3d Class (build-into AutoCAD's .NET API) to get a MidPoint property. This is in the Geometry Namespace. But the Line Entity (DatabaseServices Namespace) does not have a MidPoint Property. The Get is easy. The Set is also fairly easy but we Transform (move) the Line to the specified MidPoint by getting a Vector from the existing LinePoint's MidPoint to the specified new MidPoint.

The MidPoint Property we have implemented is just the beginning. We can overload the 'New' Constructor. Here are a couple of thoughts:

- Specify Database, MidPoint, Length, Rotation
- Specify Database, StartPoint, Length, Rotation

Additional Properties and Methods can also be added to make it easy to work with Lines in our programming.

AUTODESK UNIVERSITY

Learning about when and how to use Dictionaries, Lists, and Arrays

Dictionaries, Lists, and Arrays allow us to 'hold' multiple items in a single 'container'. They are usually used to 'hold' multiple objects of the same type.

```
<CommandMethod("AU2017_D")>
Public Sub AU2017_D()
    Dim myDStrings As New Dictionary(Of String, String)
    myDStrings.Add("Child 1", "Brandon")
    myDStrings.Add("Child 2", "Kyle")
    myDStrings.Add("Child 3", "Benjamin")
    myDStrings.Add("Child 4", "Jacob")
    myDStrings.Add("Child 5", "Nathan")
    myDStrings.Add("Child 6", "Olivia")
    myDStrings.Add("Child 7", "Emily")
    myDStrings.Add("Child 8", "Sophia")
    myDStrings.Add("Child 9", "David")
    Dim myLStrings As New List(Of String)
    myLStrings.Add("Brandon")
    myLStrings.Add("Kyle")
    myLStrings.Add("Benjamin")
    myLStrings.Add("Jacob")
    myLStrings.Add("Nathan")
    myLStrings.Add("Olivia")
    myLStrings.Add("Emily")
    myLStrings.Add("Sophia")
    myLStrings.Add("David")
    Dim myAStrings(8) As String
    myAStrings(0) = "Brandon"
    myAStrings(1) = "Kyle"
    myAStrings(2) = "Benjamin"
    myAStrings(3) = "Jacob"
    myAStrings(4) = "Nathan"
    myAStrings(5) = "Olivia"
    myAStrings(6) = "Emily"
    myAStrings(7) = "Sophia"
    myAStrings(8) = "David"
    '    OR
    myAStrings = {"Brandon", "Kyle", "Benjamin",
                  "Jacob", "Nathan", "Olivia",
                  "Emily", "Sophia", "David"}

    MsgBox(myDStrings("Child 1"))
    MsgBox(myDStrings.Keys(0))
    MsgBox(myDStrings.Values(0))
    MsgBox(myDStrings.ContainsKey("Child 10"))
    MsgBox(myDStrings.ContainsValue("Candice"))
    MsgBox(myDStrings.ElementAt(1).Value)
    MsgBox(myDStrings.ElementAt(2).Key)
    MsgBox(myDStrings.Count)
    MsgBox(myDStrings.First.Value)
    MsgBox(myDStrings.Last.Value)
    myDStrings.Reverse
    MsgBox(myDStrings.First.Value)
    MsgBox(myLStrings(3))
    For Each myVal As String In myLStrings
        MsgBox(myVal & " - " & myLStrings.IndexOf(myVal))
    Next
    For Each myVal As String In myAStrings
        MsgBox(myVal)
    Next
End Sub
```

Dictionaries are not just for String, String pairs. Take a look:

```
<CommandMethod("AU2017_E")>
Public Sub AU2017_E()

    Dim myFiles As New Dictionary(Of String, IO.FileInfo)

    Dim myCells As New Dictionary(Of System.Drawing.Point, Microsoft.Office.Interop.Excel.Range)

    Dim myBlockReferences As New Dictionary(Of String, BlockReference)

    Dim myAttributes As New Dictionary(Of String, Dictionary(Of String, String))
    'Key is BlockReference Handle, Value (Dictionary(of String, String)) is Attribute Tag / Value pairs.

    Dim myLayouts As New Dictionary(Of String, Layout)

    Dim myLayers As New Dictionary(Of ObjectId, LayerTableRecord)

End Sub
```

Keys can be any Type and Values can be any Type (or at least any type I have tried). The above code shows a variety of Keys and Values.

Once in a Dictionary, the Key/Value Pairs can be stepped through with code such as:

```
<CommandMethod("AU2017_F")>
Public Sub AU2017_F()
    Dim myDStrings As New Dictionary(Of String, String)
    myDStrings.Add("Child 1", "Brandon")
    myDStrings.Add("Child 2", "Kyle")
    myDStrings.Add("Child 3", "Benjamin")
    myDStrings.Add("Child 4", "Jacob")
    myDStrings.Add("Child 5", "Nathan")
    myDStrings.Add("Child 6", "Olivia")
    myDStrings.Add("Child 7", "Emily")
    myDStrings.Add("Child 8", "Sophia")
    myDStrings.Add("Child 9", "David")
    For Each myKVP As KeyValuePair(Of String, String) In myDStrings
        MsgBox(myKVP.Key & vbTab & myKVP.Value)
    Next
End Sub
```

SortedDictionaries and SortedLists

SortedDictionaries and SortedLists automatically sort the elements based on the keys. The SortedList has a sort Key and a Value just like the Dictionary. There is a difference in the way the Dictionary and List is sorted so it may be good to do a little testing to see which works best in each given situation.

```
<CommandMethod("AU2017_G")>
Public Sub AU2017_G()
    Dim myDStrings As New Dictionary(Of String, String)
    myDStrings.Add("Brandon", "Brandon")
    myDStrings.Add("Kyle", "Kyle")
    myDStrings.Add("Benjamin", "Benjamin")
    myDStrings.Add("Jacob", "Jacob")
    myDStrings.Add("Nathan", "Nathan")
    myDStrings.Add("Olivia", "Olivia")
    myDStrings.Add("Emily", "Emily")
    myDStrings.Add("Sophia", "Sophia")
    myDStrings.Add("David", "David")
    For Each myKVP As KeyValuePair(Of String, String) In myDStrings
        MsgBox(myKVP.Key & vbTab & myKVP.Value)
    Next
    Dim mySStrings As New Dictionary(Of String, String)
    mySStrings.Add("Brandon", "Brandon")
    mySStrings.Add("Kyle", "Kyle")
    mySStrings.Add("Benjamin", "Benjamin")
    mySStrings.Add("Jacob", "Jacob")
    mySStrings.Add("Nathan", "Nathan")
    mySStrings.Add("Olivia", "Olivia")
    mySStrings.Add("Emily", "Emily")
    mySStrings.Add("Sophia", "Sophia")
    mySStrings.Add("David", "David")
    For Each myKVP As KeyValuePair(Of String, String) In mySStrings
        MsgBox(myKVP.Key & vbTab & myKVP.Value)
    Next
End Sub
```

So, when do we use a Dictionary, List, or Array? If something needs to be sorted, either a SortedDictionary or SortedList is used, of course. If we just only need items, not 'named' items, the List is best. If we need 'named' items, the Dictionary is what should be used. Where does the Array come in? Arrays of various objects are used as return values of Functions from a variety of sources. The great thing is that Lists can be 'converted' to an Array by using the ToArray method if an Array is needed. The same can be said of Values and Keys of Dictionaries.

Learn how to use all of the .NET Framework and be creative once in a while

The .NET Framework is massive. There are a lot of things that can be done. If we take a look at the .NET Framework References we can add to our projects, we may become overwhelmed.

Name	Version
Accessibility	4.0.0.0
CustomMarshalers	4.0.0.0
ISymWrapper	4.0.0.0
Microsoft.Activities.Build	4.0.0.0
Microsoft.Build	4.0.0.0
Microsoft.Build.Conversion.v4.0	4.0.0.0
Microsoft.Build.Engine	4.0.0.0
Microsoft.Build.Framework	4.0.0.0
Microsoft.Build.Tasks.v4.0	4.0.0.0
Microsoft.Build.Utilities.v4.0	4.0.0.0
Microsoft.CSharp	4.0.0.0
Microsoft.JScript	10.0.0.0
Microsoft.VisualBasic	10.0.0.0
Microsoft.VisualBasic.Compatibility	10.0.0.0
Microsoft.VisualBasic.Compatibility.Data	10.0.0.0
Microsoft.VisualC	10.0.0.0
Microsoft.VisualC.STLCLR	2.0.0.0
mscorlib	4.0.0.0
PresentationBuildTasks	4.0.0.0
PresentationCore	4.0.0.0
PresentationFramework	4.0.0.0
PresentationFramework.Aero	4.0.0.0
PresentationFramework.Aero2	4.0.0.0
PresentationFramework.AeroLite	4.0.0.0
PresentationFramework.Classic	4.0.0.0
PresentationFramework.Luna	4.0.0.0
PresentationFramework.Royale	4.0.0.0
ReachFramework	4.0.0.0
sysglobl	4.0.0.0
System	4.0.0.0
System.Activities	4.0.0.0
System.Activities.Core.Presentation	4.0.0.0
System.Activities.DurableInstancing	4.0.0.0
System.Activities.Presentation	4.0.0.0
System.AddIn	4.0.0.0
System.AddIn.Contract	4.0.0.0
System.ComponentModel.Composition	4.0.0.0
System.ComponentModel.Composition.Regist...	4.0.0.0
System.ComponentModel.DataAnnotations	4.0.0.0
System.Configuration	4.0.0.0
System.Configuration.Install	4.0.0.0
System.Core	4.0.0.0
System.Data	4.0.0.0
System.Data.DataSetExtensions	4.0.0.0
System.Data.Entity	4.0.0.0

Name	Version
System.Data.Entity.Design	4.0.0.0
System.Data.Linq	4.0.0.0
System.Data.OracleClient	4.0.0.0
System.Data.Services	4.0.0.0
System.Data.Services.Client	4.0.0.0
System.Data.Services.Design	4.0.0.0
System.Data.SqlXml	4.0.0.0
System.Deployment	4.0.0.0
System.Design	4.0.0.0
System.Device	4.0.0.0
System.DirectoryServices	4.0.0.0
System.DirectoryServices.AccountManagement	4.0.0.0
System.DirectoryServices.Protocols	4.0.0.0
System.Drawing	4.0.0.0
System.Drawing.Design	4.0.0.0
System.Dynamic	4.0.0.0
System.EnterpriseServices	4.0.0.0
System.IdentityModel	4.0.0.0
System.IdentityModel.Selectors	4.0.0.0
System.IdentityModel.Services	4.0.0.0
System.IO.Compression	4.0.0.0
System.IO.Compression.FileSystem	4.0.0.0
System.IO.Log	4.0.0.0
System.Management	4.0.0.0
System.Management.Instrumentation	4.0.0.0
System.Messaging	4.0.0.0
System.Net	4.0.0.0
System.Net.Http	4.0.0.0
System.Net.Http.WebRequest	4.0.0.0
System.Numerics	4.0.0.0
System.Numerics.Vectors	4.0.0.0
System.Printing	4.0.0.0
System.Reflection.Context	4.0.0.0
System.Runtime.Caching	4.0.0.0
System.Runtime.DurableInstancing	4.0.0.0
System.Runtime.Remoting	4.0.0.0
System.Runtime.Serialization	4.0.0.0
System.Runtime.Serialization.Formatters.Soap	4.0.0.0
System.Security	4.0.0.0
System.ServiceModel	4.0.0.0
System.ServiceModel.Activation	4.0.0.0
System.ServiceModel.Activities	4.0.0.0
System.ServiceModel.Channels	4.0.0.0
System.ServiceModel.Discovery	4.0.0.0
System.ServiceModel.Routing	4.0.0.0

Name	Version
System.ServiceModel.Web	4.0.0.0
System.ServiceProcess	4.0.0.0
System.Speech	4.0.0.0
System.Transactions	4.0.0.0
System.Web	4.0.0.0
System.Web.Abstractions	4.0.0.0
System.Web.ApplicationServices	4.0.0.0
System.Web.DataVisualization	4.0.0.0
System.Web.DataVisualization.Design	4.0.0.0
System.Web.DynamicData	4.0.0.0
System.Web.DynamicData.Design	4.0.0.0
System.Web.Entity	4.0.0.0
System.Web.Entity.Design	4.0.0.0
System.Web.Extensions	4.0.0.0
System.Web.Extensions.Design	4.0.0.0
System.Web.Mobile	4.0.0.0
System.Web.RegularExpressions	4.0.0.0
System.Web.Routing	4.0.0.0
System.Web.Services	4.0.0.0
System.Windows	4.0.0.0
System.Windows.Controls.Ribbon	4.0.0.0
System.Windows.Forms	4.0.0.0
System.Windows.Forms.DataVisualization	4.0.0.0
System.Windows.Forms.DataVisualization.Desi...	4.0.0.0
System.Windows.Input.Manipulations	4.0.0.0
System.Windows.Presentation	4.0.0.0
System.Workflow.Activities	4.0.0.0
System.Workflow.ComponentModel	4.0.0.0
System.Workflow.Runtime	4.0.0.0
System.WorkflowServices	4.0.0.0
System.Xaml	4.0.0.0
System.Xml	4.0.0.0
System.Xml.Linq	4.0.0.0
System.Xml.Serialization	4.0.0.0
UIAutomationClient	4.0.0.0
UIAutomationClientsideProviders	4.0.0.0
UIAutomationProvider	4.0.0.0
UIAutomationTypes	4.0.0.0
WindowsBase	4.0.0.0
WindowsFormsIntegration	4.0.0.0
XamlBuildTask	4.0.0.0

That’s a lot to work with. Let’s try using a couple of them.

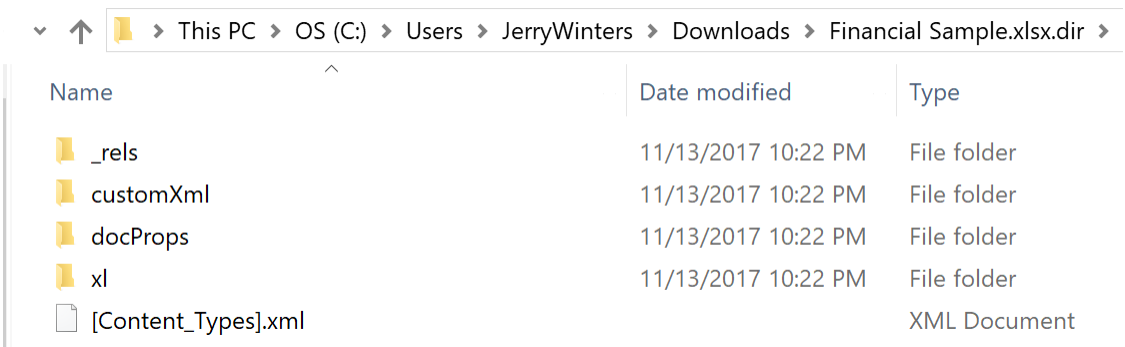
```
<CommandMethod("AU2017_H")>
Public Sub AU2017_H()
    Dim sourceBMP As System.Drawing.Bitmap = GetPreviewBMP(HostApplicationServices.WorkingDatabase)
    Dim outputBMP As New System.Drawing.Bitmap(sourceBMP.Width, sourceBMP.Height)
    Dim myG As System.Drawing.Graphics = System.Drawing.Graphics.FromImage(outputBMP)
    myG.DrawImage(sourceBMP, 0, 0)
    Dim myFont As New System.Drawing.Font("Arial", 8)
    Dim myBrush As System.Drawing.Brush = System.Drawing.Brushes.Blue
    Dim fileText As String = System.IO.Path.GetFileNameWithoutExtension(HostApplicationServices.WorkingDatabase.Filename)
    myG.DrawString(fileText,
                   myFont, myBrush, New Drawing.PointF(10, outputBMP.Height - 20))
    Dim myFIO As New IO.FileInfo(IO.Path.Combine(My.Computer.FileSystem.SpecialDirectories.Desktop,
                                 IO.Path.GetFileNameWithoutExtension(HostApplicationServices.WorkingDatabase.Filename)))
    If myFIO.Exists Then
        Try
            myFIO.Delete()
            outputBMP.Save(myFIO.FullName & ".jpg")
            outputBMP.Dispose()
        Catch ex As Exception
            MsgBox("Error: " & ex.Message)
            outputBMP.Dispose()
        End Try
    Else
        outputBMP.Save(myFIO.FullName & ".jpg")
        outputBMP.Dispose()
    End If
End Sub

Public Function GetPreviewBMP(dbIn As Database) As System.Drawing.Bitmap
    If dbIn.ThumbnailBitmap IsNot Nothing Then
        Return dbIn.ThumbnailBitmap
    Else
        Dim myBMP As New System.Drawing.Bitmap(200, 200)
        Dim myBrush As System.Drawing.Brush = System.Drawing.Brushes.Red
        Dim myFont As New System.Drawing.Font("Arial", 14)
        Dim myG As System.Drawing.Graphics = System.Drawing.Graphics.FromImage(myBMP)
        myG.DrawString("No Preview.", myFont, myBrush, New Drawing.PointF(10, 10))
        Return myBMP
    End If
End Function
```

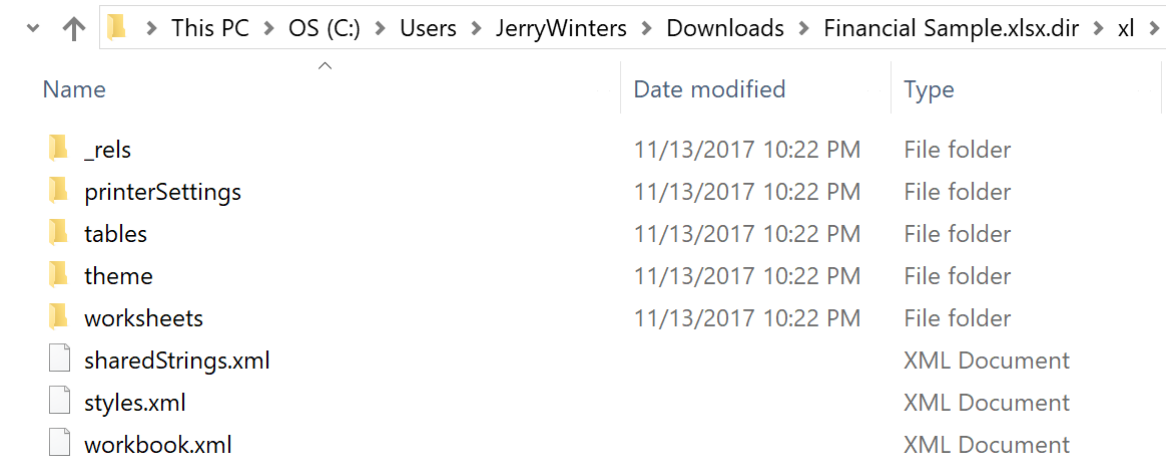
This example uses the System.Drawing namespace to get the Thumbnail Bitmap from the supplied Database, then writes the name of the .dwg file at the bottom of the preview bitmap.

System.IO.Compression.FileSystem

```
<CommandMethod("AU2017_J")>
Public Sub AU2017_J()
    Dim myFIO As New IO.FileInfo("C:\Users\JerryWinters\Downloads\Financial Sample.xlsx")
    System.IO.Compression.ZipFile.ExtractToDirectory(myFIO.FullName, myFIO.FullName & ".dir")
End Sub
```

If you have ever wanted to read an Excel file without having Excel load the file, the ExtractToDirectory call can be used to 'extract' the Excel file components into directories and XML files.





workbook.xml*

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<workbook xmlns="http://schemas.openxmlformats.org/spreadsheetml/2006/main"
          xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships"
          xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
          mc:Ignorable="x15"
          xmlns:x15="http://schemas.microsoft.com/office/spreadsheetml/2010/11/main">
  <fileVersion appName="xl" lastEdited="6" lowestEdited="6" rupBuild="14420"/>
  <workbookPr defaultThemeVersion="153222"/>
  <mc:AlternateContent xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006">
    <mc:Choice Requires="x15">
      <x15ac:absPath url="C:\Users\amac\Desktop\Workbooks\" xmlns:x15ac="http://schemas.microsoft.com/office/spreadsheetml/2010/11/ac"/>
    </mc:Choice>
  </mc:AlternateContent>
  <bookViews>
    <workbookView xWindow="0" yWindow="0" windowWidth="21570" windowHeight="8145"/>
  </bookViews>
  <sheets>
    <sheet name="Sheet1" sheetId="6" r:id="rId1"/>
  </sheets>
  <calcPr calcId="152511"/>
  <extLst>
    <ext uri="{140A7094-0E35-4892-8432-C4D2E57EDEB5}" xmlns:x15="http://schemas.microsoft.com/office/spreadsheetml/2010/11/main">
      <x15:workbookPr chartTrackingRefBase="1"/>
    </ext>
  </extLst>
</workbook>
```

sheet1.xml*

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<worksheet xmlns="http://schemas.openxmlformats.org/spreadsheetml/2006/main" xmlns:r="h
  <dimension ref="A1:P701"/>
  <sheetViews>
    <sheetView tabSelected="1" zoomScale="85" zoomScaleNormal="85" workbookViewId="0">
      <selection activeCell="N6" sqref="N6"/>
    </sheetView>
  </sheetViews>
  <sheetFormatPr defaultRowHeight="15" x14ac:dyDescent="0.25"/>
  <cols>
    <col min="1" max="1" width="16.28515625" bestFit="1" customWidth="1"/>
    <col min="2" max="2" width="26.5703125" customWidth="1"/>
    <col min="3" max="3" width="14.140625" style="3" bestFit="1" customWidth="1"/>
    <col min="4" max="4" width="17.42578125" bestFit="1" customWidth="1"/>
    <col min="5" max="5" width="14.28515625" style="1" customWidth="1"/>
    <col min="6" max="6" width="14.28515625" style="1" bestFit="1" customWidth="1"/>
    <col min="7" max="7" width="12.5703125" style="1" bestFit="1" customWidth="1"/>
    <col min="8" max="8" width="14.28515625" style="1" bestFit="1" customWidth="1"/>
    <col min="9" max="9" width="12.5703125" style="1" bestFit="1" customWidth="1"/>
    <col min="10" max="10" width="17.7109375" style="1" customWidth="1"/>
    <col min="11" max="11" width="11.5703125" bestFit="1" customWidth="1"/>
    <col min="12" max="12" width="18.42578125" bestFit="1" customWidth="1"/>
    <col min="13" max="13" width="11.5703125" style="4" bestFit="1" customWidth="1"/>
    <col min="14" max="14" width="17.140625" style="9" bestFit="1" customWidth="1"/>
    <col min="15" max="15" width="16.5703125" bestFit="1" customWidth="1"/>
    <col min="16" max="16" width="7.5703125" style="2" bestFit="1" customWidth="1"/>
  </cols>
  <sheetData>
    <row r="1" spans="1:16" x14ac:dyDescent="0.25">
      <c r="A1" t="s">
        <v>6</v>
      </c>
      <c r="B1" t="s">
        <v>36</v>
      </c>
      <c r="C1" s="3" t="s">
        <v>37</v>
      </c>
```

sharedStrings.xml*

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<sst xmlns="http://schemas.openxmlformats.org/spreadshe
  <si>
    <t>Year</t>
  </si>
  <si>
    <t>Gross Sales</t>
  </si>
  <si>
    <t>Discounts</t>
  </si>
  <si>
    <t>COGS</t>
  </si>
  <si>
    <t>Units Sold</t>
  </si>
  <si>
    <t>Manufacturing Price</t>
  </si>
  <si>
    <t>Segment</t>
  </si>
  <si>
    <t>Small Business</t>
  </si>
  <si>
    <t>Midmarket</t>
  </si>
  <si>
    <t>Enterprise</t>
  </si>
  <si>
    <t>Government</t>
  </si>
  <si>
    <t>Channel Partners</t>
  </si>
```

If you are planning on using this code, keep in mind that most of the values found in Excel files are in the sharedStrings.xml file.

AUTODESK. UNIVERSITY

Learn how to identify and consolidate your code into these tidy little units

While writing 10,000 lines of code in one CommandMethod will run and get things done, there are usually better ways to do things. Here are some things to consider in identifying what to 'break out' and how to break them out.

1. Is this code a one-off or will it be useful in other projects?
2. Can a group of code be considered a 'Step'? Could you call it 'Step 1', 'Step 2', 'Step 3', etc.?
3. What can be 'parameterized' versus hard-coded values?
4. Will adding additional parameters make the Method or Function more useful?
5. ByVal vs ByRef—what should the Parameters be declared as?
6. ParamArray—Not used greatly but is perfect once in a while.
7. Can/Should a Call be Overloaded?

The next question is, how is this code 'stored' for future use?

A. Store it in a Class?
B. Store it in a Module?
C. Store it in a Code Snippet?
D. Store it in other Code Storage Mechanism?

One more thought—What is the code worth?

If 10 years are spent creating a library of Functions and Methods, Classes, Collections, etc. and we can now write a fully functional program with a few lines of code, is it worth 10 minutes of programming?

Review

Our AutoCAD Add-Ins can be Better, Stronger, and More Stable if we take time to consider how to best write our code. Taking time during each project to write code that will be useful in future projects also results in code that is easier to debug and maintain. Making use of the .NET Framework can give our applications powerful features.

I hope your time here at Autodesk University is helpful you.

Thank you for your time.

Jerry Winters
VB CAD, Inc.
Jerryw@vbcad.com