What Is an AutoCAD® ACA/MEP AutoLISP® Guy Supposed to Do?

R. Robert Bell – Sparling

CP5862 Let us face it. Every year, less and less of the data in AutoCAD ACA/MEP is exposed to AutoLISP. Even if some data may have been available in the last release of AutoCAD ACA/MEP, there is no guarantee that it will be exposed in the next version, especially when the DWG™ format changes. You may not feel comfortable replacing all your AutoLISP code with .NET code until there is a time where your company can afford a full rewrite of your code. So how do you bridge the gap between the data stored in AutoCAD ACA/MEP and the code you have written in AutoLISP? This class will take show you how you can write just enough .NET code to be dangerous, er, to be effective in exposing data which your AutoLISP applications need. We will use three different examples, based on the speaker's own need to access the data for AutoLISP applications. Programming skills are required for this class, although .NET experience is not a prerequisite.

Learning Objectives

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

- Use simple error-handling techniques
- Hunt through the .NET object model
- Create functions to read or write MEP data
- Use .NET to write a new AutoLISP function

About the Speaker

Robert is the design technology manager for Sparling, a specialty electrical engineering and technology consulting firm in the United States, headquartered in Seattle, Washington. He provides strategic direction, technical oversight, and high-level support for the Sparling enterprise design and production technology systems. He is instrumental in positioning Sparling as an industry and client leader in leveraging technology in virtual building and design. Robert has been writing code for customizing AutoCAD® since the release of AutoCAD v2.5. rbell@sparling.com

Introduction

Many of us still use AutoLISP to write quick functions to handle tasks. Why? I'm sure Autodesk continues to ask that question but the answer may not be simple to determine. In my case, it is because I spent decades writing code in AutoLISP for customizing AutoCAD and only a few years using .NET languages. Therefore, when a request comes in for some customization, and I determine that it is a simple bit of code, I tend to use AutoLISP because I know accurately how long it will take to write the application.

In years past, you could still get a great deal of information from even ACA/MEP drawings and in many cases modify ACA/MEP data. But those days are fading. Every year it seems as if less of the ACA/MEP data is exposed to AutoLISP and in some cases what worked the year before won't work in the latest release.

A perfect example of this is the Annotation Plot Size variable exposed under the Units command. This is a variable that used to be exposed in a dictionary and could be examined *and modified* by normal AutoLISP functions. However, that changed with the release of AutoCAD ACA/MEP 2009. Although the value associated with the variable could still be read by AutoLISP it could no longer be altered with AutoLISP.

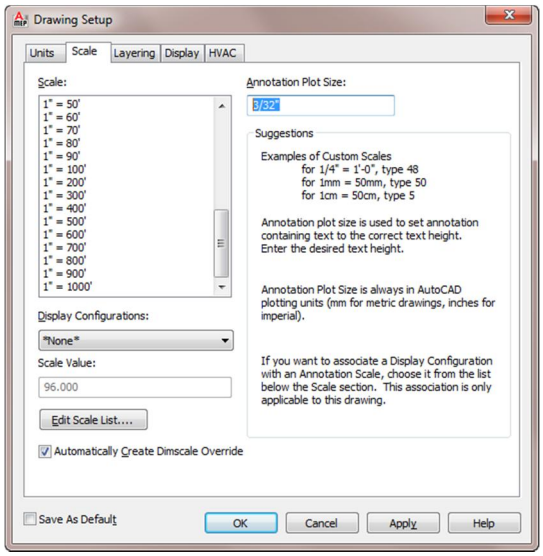

Figure 1, Annotation Plot Size

What is an AutoCAD ACA/MEP guy (or gal) supposed to do?

We need to drag ourselves, kicking and screaming, into the .NET world. However, we do not need to give up all of our AutoLISP code at once. Where functions that used to work with ACA/MEP data no longer work, we can write *new* AutoLISP functions using .NET code.

Yes, we will need to take more time to write the code, but it may be far quicker to write a new AutoLISP function using .NET than it would be to translate an entire AutoLISP application into a .NET language.

This class will present how a long-time AutoLISP guy was able to use some .NET code to create new AutoLISP functions to keep the code working in new releases. The class is *not* intended to teach you how to start a .NET application from scratch. Nor is it presented by a .NET programming guru. However, it is hoped that this class will provide you with the reference material you need in order to write a new AutoLISP function for working with ACA/MEP data.

The code presented here is written using VB.NET although I personally prefer C# if only because many samples written by Autodesk are in C#. (There are other reasons, but I won't take your time with this side topic.)

Use Simple Error-Handling Techniques

The first thing an AutoLISP programmer is going to need to learn is how to write .NET code to handled potential errors. When you write applications for AutoCAD using .NET languages you are expected to use *transactions* when examining or modifying the drawings or application settings.

Transactions are a new thing for AutoLISP programmers but they don't need to be scary. It is easiest to think of transactions as if they are a *vl-catch-all-apply* function or a (very!) robust traditional AutoLISP error handler.

VB.NET Syntax

Before diving into some actual code, it may be helpful to have an understanding of some VB.NET syntax in comparison to AutoLISP syntax

(setq)

Setting a variable in VB.NET is similar to AutoLISP except that you need to specify the type of data associated with the variable. Consider this VB.NET statement:

```
Dim annoPlotSize As Double = 0.125
```

It would be similar to this AutoLISP statement (if you were required to specify the data type, acDouble):

```
(setq annoPlotSize acDouble 0.125)
```

Code line breaks

VB.NET statements need to remain on the same line, similar to AutoLISP, unless an underscore (_) is used before splitting the statement into multiple lines. Some of the code samples in this handout use the underscore to split a single statement into multiple lines to avoid confusion due to word wrap.

Why do some things end in empty parentheses and some don't?

Anything that is referring to a method that does not have arguments is going to need empty parentheses at the end of the method name whereas if you are referring to a property there are no parentheses at the end of the property name.

The Transaction Framework

The transaction framework starts with the Using statement. You then Try something, Catch any errors, and Finally finish the transaction. It's as simple as that. Below is the basic framework, without full code, to make it easy to see how it needs to be constructed.

```
Using trans <a transaction>
  Try
    <something>
    trans.Commit()
  Catch myErr <an error>
    trans.Abort()
  Finally
    trans.Dispose()
  End Try
End Using
```

See, nothing scary there. ☺

Okay, let's flesh it out a bit more. The transaction is going to need to work within a database. What?! A database?! What kind of class is this?

Why did it have to be a Database?

A *database* to us in AutoCAD-land is simply the data contained within a drawing. It does not have to be an open drawing, it can be a drawing that has been saved, or even a drawing that hasn't been started in the AutoCAD editor (think of an invisible new drawing and you have the idea.) Because we are writing AutoLISP functions, we are going to be working with the current drawing. (Many AutoLISP functions only work in the context of the current drawing.)

Below is how you get the current drawing in VB.NET.

```
Dim thisDoc As Document = Application.DocumentManager.MdiActiveDocument
```

Now that we have the current document we can get its database. We need to get a database to feed to the transaction statement.

```
Dim thisDb As Database = thisDoc.Database
```

Once we have the database, we can use it in the Using statement.

```
Using trans As Transaction = thisDb.TransactionManager.StartTransaction()
```

See how the code defines a variable (trans) as a transaction and then assigns it to the database's TransactionManager, using its StartTransaction method?

Catching the Error

Since this code is ultimately going to be used to create an AutoLISP function there is not much need to report the actual error. It is only important to catch (trap) the error. This is the code needed to do catch the error.

```
Catch myErr As Autodesk.AutoCAD.Runtime.Exception
```

A Fleshed-Out Transaction Framework (Without Error Reporting)

Below is a transaction framework with some of the added code from above.

```
Dim thisDoc As Document = Application.DocumentManager.MdiActiveDocument
Dim thisDb As Database = thisDoc.Database

Using trans As Transaction = thisDb.TransactionManager.StartTransaction()
  Try
    <something>
    trans.Commit()
  Catch myErr As Autodesk.AutoCAD.Runtime.Exception
    trans.Abort()
  Finally
    trans.Dispose()
  End Try
End Using
```

Next we will discuss how to figure out where to find the information needed to affect ACA/MEP data.

Hunt Through the .NET Object Model

Trying to figure out what you are looking for when it comes to objects can be a challenge. The names of the .NET objects don't necessarily match the names of the AutoCAD objects. The descriptions in the .NET document never mention the actual AutoCAD object name so it becomes troublesome at times to locate the needed information. To make matters worse, for MEP folks, sometimes the object you are looking for is in the ACA documentation because MEP is based on ACA and utilizes a lot of the ACA .NET object model.

The documentation describe below is located in the Help folder for AutoCAD ACA/MEP which is typically:
C:\Program Files\Autodesk\<version>\Help

The ACA documentation is in acamgd.chm (think ACA Managed.)

The MEP documentation is in AecbBldSrvMgd.chm (think AECB Building Services Managed.)

AutoCAD .NET documentation, should you need it, is in arxmgd.chm (think ARX Managed.)

If you do any amount of .NET programming for ACA/MEP it is advantageous to create shortcuts to those files on your desktop.

The Easter egg Hunt

Let's start with an example from this class. We want to change the variable "Annotation Plot Size". Since this variable is in both ACA and MEP, we know we need to start in the ACA documentation.

It would seem natural to search for "Annotation Plot Size" since that is the only name we know for the system variable given the Units command. Or, as the Figure 2 shows, that was optimistic at best. (For the keen-eyed, you may be noticing that this help file is referring to 2011. But this really *is* the 2012 help file. Try it, and you will see.)

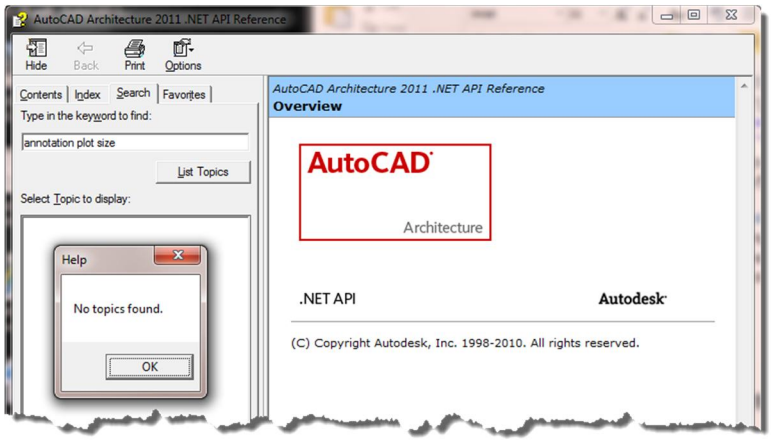

Figure 2, Search Denied

(groan) This is where it gets fun. Okay, "Annotation Plot Size" didn't work, let's try "variable". (fail) How about "size"? We got some results, but they have nothing to do with annotation plot size. "Plot"? (fail) "Annotation"? (over 100 results)

Are you beginning see how much fun you are going to have? Hang in there, it's worth it, to keep you code moving forward.

Scanning thru the overwhelming list of results as least rewards us with a promising item: DBVariables (database variables? That's what I needed to search on?...) (fail)

Double-click on the DBVariables topic in the list. Scan thru the results on the right. Have you seen anything that is promising? I haven't. (groan)

Back to the topic list… wait a minute! What about "DrawingSetupVariables Members"? Double-click on that topic. (This does look more promising!) Search for "Annotation". (groan)

Don't give up! There, nearly at the bottom of the list, is TextHeight. (high fives around the office) However, click on the link. Does it actually mention "Annotation Plot Size"? (no) How do you know this is the correct member? You don't, until you write code and test it, or you are lucky to find someone else that went thru this on Google.

Will it get easier with practice? I would love to tell you "yes", but I would be lying.

Now that we have an idea of where to get the data, we can write code to read and write to the ACA/MEP variable.

Create Functions to Read or Write MEP Data

One of the key differences between AutoLISP and .NET programming is that in .NET you really need to work with object IDs far more often. In fact, you almost *never* want to "remember" anything but an object ID. For example, in AutoLISP we are used to storing the actual object data and then changing it. Then same thing is usually frowned on in AutoCAD's .NET object model, due to the modeless operation that is becoming more common. This modeless operation, and potentially multithreading, is what requires the use of transactions.

Reading Data

Reading data requires at least two lines of code, one to get the object ID and the other to get the object itself for reading the data.

```
Dim objID As Autodesk.AutoCAD.DatabaseServices.ObjectId = _
DrawingSetupVariables.GetInstance(thisDb, False)
```

In order to examine the data exposed by the members of this object we need to get an instance of the object for the current drawing. Hence we use the GetInstance method. The documentation tells us we need two parameters, one for the database and one for if we need to create the database if needed. We are working with the current drawing so we won't need to create the database. We grab the object ID of the DrawingSetupVariables.

```
Dim aecDwgSetupVars As DrawingSetupVariables = _
trans.GetObject(objID, OpenMode.ForRead)
```

Using the transaction, we now attempt to open the DrawingSetupVariables for a read operation. We are only interested in reading the data at the moment.

Writing Data

Writing data requires at least three lines of code. Similar to reading data you need the object ID and get the object itself (albeit for writing instead of reading). However, you will want to write data to the object, which is the 3rd line of code.

```
Dim objID As Autodesk.AutoCAD.DatabaseServices.ObjectId = _
DrawingSetupVariables.GetInstance(thisDb, False)
```

The above line is exactly like the like the line for reading data because it is doing the same thing: getting the object ID.

```
Dim aecDwgSetupVars As DrawingSetupVariables = _
trans.GetObject(objID, OpenMode.ForWrite)
```

This line is similar to the one for reading data except that opens the object for writing.

```
aecDwgSetupVars.TextHeight = 0.125
```

This line actually sets the TextHeight property to the size provided.

Read Sample in the Transaction Framework

This is the code shown above within the framework of the transaction code.

```
Using trans As Transaction = thisDb.TransactionManager.StartTransaction()
  Try
    Dim objID As Autodesk.AutoCAD.DatabaseServices.ObjectId = _
    DrawingSetupVariables.GetInstance(thisDb, False)
    Dim aecDwgSetupVars As DrawingSetupVariables = _
    trans.GetObject(objID, OpenMode.ForRead)
    trans.Commit()
  Catch myErr As Autodesk.AutoCAD.Runtime.Exception
    trans.Abort()
  Finally
    trans.Dispose()
  End Try
End Using
```

Write Sample in the Transaction Framework

This is the code shown above within the framework of the transaction code.

```
Using trans As Transaction = thisDb.TransactionManager.StartTransaction()
  Try
    Dim objID As Autodesk.AutoCAD.DatabaseServices.ObjectId = _
    DrawingSetupVariables.GetInstance(thisDb, False)
    Dim aecDwgSetupVars As DrawingSetupVariables = _
    trans.GetObject(objID, OpenMode.ForWrite)
    aecDwgSetupVars.TextHeight = 0.125
    trans.Commit()
  Catch myErr As Autodesk.AutoCAD.Runtime.Exception
    trans.Abort()
  Finally
    trans.Dispose()
  End Try
End Using
```

The final step is to get this .NET code exposed as new AutoLISP functions.

Use .NET to write a New AutoLISP Function

Creating an AutoLISP function to read data takes less code that a function to write data, obviously.

Making a .NET Procedure into an AutoLISP Function

Once you have a full .NET procedure it is easy to expose it as an AutoLISP function. The following statement is almost all that's needed.

```
<LispFunction("<AutoLISP function name>")> _
```

Note that the trailing underscore (_) in the statement is required because there is no closing statement for the LispFunction statement.

Okay, the above is a bit simplistic. In reality you have to accommodate data provided to AutoLISP functions or coming out of AutoLISP functions. Date can be passed between AutoLISP functions and .NET procedures using a *ResultBuffer*. ResultBuffers are basically unstructured arrays of data.

Reading Data

Even for AutoLISP functions that are only going to read data it is a good idea to make sure that no arguments are provided accidently. The following code does that.

```
<LispFunction("sinc:GetAecTextHeight")> _
Public Function GetAnnotationPlotSize(ByVal rb As ResultBuffer)
  Dim thisDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim thisDb As Database = thisDoc.Database
  Dim theResult As TypedValue = New TypedValue(LispDataType.Nil)

  If rb = Nothing Then
    Using trans As Transaction = _
    thisDb.TransactionManager.StartTransaction()
      Try
        Dim objID As Autodesk.AutoCAD.DatabaseServices.ObjectId = _
        DrawingSetupVariables.GetInstance(thisDb, False)
        Dim aecDwgSetupVars As DrawingSetupVariables = _
        trans.GetObject(objID, OpenMode.ForRead)
        trans.Commit()
        theResult = New TypedValue(LispDataType.Double, _
                                   aecDwgSetupVars.TextHeight)
      Catch myErr As Autodesk.AutoCAD.Runtime.Exception
        trans.Abort()
      Finally
        trans.Dispose()
      End Try
    End Using
  Else
    thisDoc.Editor.WriteMessage(vbLf & "too many arguments ")
  End If

  Return theResult
End Function
```

Writing Data

Writing data to the database obviously is going to require at least one argument provide by the AutoLISP function. So the .NET code to read the data coming from AutoLISP will be more complicated than the code only for reading data.

```
<LispFunction("sinc:SetAecTextHeight")> _
Public Function SetAnnotationPlotSize(ByVal rb As ResultBuffer)
  Dim thisDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim thisDb As Database = thisDoc.Database
  Dim annoPlotSize As Double
  Dim theResult As TypedValue = New TypedValue(LispDataType.Nil)

  If Not rb = Nothing Then
    If rb.AsArray.Length = 1 Then
      Dim val As TypedValue = rb.AsArray(0)
      If val.TypeCode = LispDataType.Double Then
```

```
        annoPlotSize = val.Value
      Else
        thisDoc.Editor.WriteMessage(vbLf & "bad argument type ")
        Return theResult
      End If

      Using trans As Transaction = _
      thisDb.TransactionManager.StartTransaction()
        Try
          Dim objID As Autodesk.AutoCAD.DatabaseServices.ObjectId = _
          DrawingSetupVariables.GetInstance(thisDb, False)
          Dim aecDwgSetupVars As DrawingSetupVariables = _
          trans.GetObject(objID, OpenMode.ForWrite)
          aecDwgSetupVars.TextHeight = annoPlotSize
          trans.Commit()
          theResult = New TypedValue(LispDataType.Double, annoPlotSize)
        Catch myErr As Autodesk.AutoCAD.Runtime.Exception
          trans.Abort()
        Finally
          trans.Dispose()
        End Try
      End Using

    Else
      thisDoc.Editor.WriteMessage(vbLf & "too many arguments ")
    End If
  Else
    thisDoc.Editor.WriteMessage(vbLf & "too few arguments ")
  End If

  Return theResult
End Function
```

Conclusion

As you can see, the biggest hurdle to moving to .NET programming is really just locating the information needed to write the code. Trial-and-error can help you wade thru the documentation but searching on the Internet may yield quicker results.

With this class and handout you will have a concrete example of how to write AutoLISP functions in .NET to expose ACA/MEP data that is otherwise concealed to AutoLISP.

Appendix

Full Code

```
Option Explicit On
Option Compare Text

Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.EditorInput
Imports Autodesk.AutoCAD.Geometry
Imports Autodesk.Aec.ApplicationServices
Imports Autodesk.Aec.DatabaseServices
Imports Autodesk.Aec.Building.DatabaseServices
Imports Autodesk.Aec.PropertyData.DatabaseServices
Imports Autodesk.Aec.Building.Elec.DatabaseServices

Public Class sincMEP

  <LispFunction("sinc:GetAecTextHeight")> _
  Public Function GetAnnotationPlotSize(ByVal rb As ResultBuffer)
    Dim thisDoc As Document = Application.DocumentManager.MdiActiveDocument
    Dim thisDb As Database = thisDoc.Database
    Dim theResult As TypedValue = New TypedValue(LispDataType.Nil)

    If rb = Nothing Then
      Using trans As Transaction = thisDb.TransactionManager.StartTransaction()
        Try
          Dim objID As Autodesk.AutoCAD.DatabaseServices.ObjectId = DrawingSetupVariables.GetInstance(thisDb, False)
          Dim aecDwgSetupVars As DrawingSetupVariables = trans.GetObject(objID, OpenMode.ForRead)
          trans.Commit()
          theResult = New TypedValue(LispDataType.Double,aecDwgSetupVars.TextHeight)
        Catch myErr As Autodesk.AutoCAD.Runtime.Exception
          trans.Abort()
        Finally
          trans.Dispose()
        End Try
      End Using
    Else
      thisDoc.Editor.WriteMessage(vbLf & "too many arguments ")
    End If

    Return theResult
  End Function

  <LispFunction("sinc:SetAecTextHeight")> _
  Public Function SetAnnotationPlotSize(ByVal rb As ResultBuffer)
    Dim thisDoc As Document = Application.DocumentManager.MdiActiveDocument
    Dim thisDb As Database = thisDoc.Database
    Dim annoPlotSize As Double
    Dim theResult As TypedValue = New TypedValue(LispDataType.Nil)

    If Not rb = Nothing Then
      If rb.AsArray.Length = 1 Then
        Dim val As TypedValue = rb.AsArray(0)
        If val.TypeCode = LispDataType.Double Then
          annoPlotSize = val.Value
        Else
          thisDoc.Editor.WriteMessage(vbLf & "bad argument type ")
          Return theResult
        End If

        Using trans As Transaction = thisDb.TransactionManager.StartTransaction()
          Try
            Dim objID As Autodesk.AutoCAD.DatabaseServices.ObjectId = DrawingSetupVariables.GetInstance(thisDb, False)
            Dim aecDwgSetupVars As DrawingSetupVariables = trans.GetObject(objID, OpenMode.ForWrite)
            aecDwgSetupVars.TextHeight = annoPlotSize
            trans.Commit()
            theResult = New TypedValue(LispDataType.Double, annoPlotSize)
          Catch myErr As Autodesk.AutoCAD.Runtime.Exception
            trans.Abort()
          Finally
            trans.Dispose()
          End Try
        End Using

      Else
        thisDoc.Editor.WriteMessage(vbLf & "too many arguments ")
      End If
    Else
      thisDoc.Editor.WriteMessage(vbLf & "too few arguments ")
    End If

    Return theResult
  End Function

  <LispFunction("sinc:GetTagData")> _
  Public Function GetTagData(ByVal rb As ResultBuffer)

    If rb.AsArray.Length = 1 Then
      Dim val As TypedValue = rb.AsArray(0)
      If val.TypeCode = LispDataType.ObjectId Then
        Dim objID As Autodesk.AutoCAD.DatabaseServices.ObjectId = val.Value
        Dim test = GetTagLocation(objID)
        If Not test = Nothing Then
          Dim tagLocation As Point3d = test
          Dim hostLocation As Point3d = GetHostLocation(objID)
```

```
        Dim theResult As New ResultBuffer
        theResult.Add(New TypedValue(LispDataType.ListBegin))
        theResult.Add(New TypedValue(LispDataType.ListBegin))
        theResult.Add(New TypedValue(LispDataType.Text, "Tag"))
        theResult.Add(New TypedValue(LispDataType.Point3d, tagLocation))
        theResult.Add(New TypedValue(LispDataType.DottedPair))
        theResult.Add(New TypedValue(LispDataType.ListBegin))
        theResult.Add(New TypedValue(LispDataType.Text, "Host"))
        theResult.Add(New TypedValue(LispDataType.Point3d, hostLocation))
        theResult.Add(New TypedValue(LispDataType.DottedPair))
        theResult.Add(New TypedValue(LispDataType.ListEnd))
        Return theResult
      Else
        Return New TypedValue(LispDataType.Nil)
      End If
    Else
      Return New TypedValue(LispDataType.Nil)
    End If
  Else
    Return New TypedValue(LispDataType.Nil)
  End If

End Function

Private Function GetTagLocation(ByVal ObjID As Autodesk.AutoCAD.DatabaseServices.ObjectId)
  Dim thisDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim thisDb As Database = thisDoc.Database

  Using trans As Transaction = thisDb.TransactionManager.StartTransaction()
    Try
      Dim testObject As Object = trans.GetObject(ObjID, OpenMode.ForRead)
      If testObject.GetType.ToString = "Autodesk.Aec.DatabaseServices.MultiViewBlockReference" Then
        Dim tagobject As MultiViewBlockReference = testObject
        Dim tagLocation As Point3d = tagobject.Location
        Return tagLocation
      Else
        thisDoc.Editor.WriteMessage(vbLf & "Error: Selected object is not a tag.")
        Return Nothing
      End If
      trans.Commit()
    Catch myErr As Exception
      thisDoc.Editor.WriteMessage(vbLf & myErr.ToString)
      trans.Abort()
      Return Nothing
    End Try
  End Using

End Function

Private Function GetHostLocation(ByVal ObjID As Autodesk.AutoCAD.DatabaseServices.ObjectId)
  Dim thisDoc As Document = Application.DocumentManager.MdiActiveDocument
  Dim thisDb As Database = thisDoc.Database
  Dim hostLocation As Point3d
  Dim hostName As String

  Using trans As Transaction = thisDb.TransactionManager.StartTransaction()
    Try
      Dim tagObject As MultiViewBlockReference = trans.GetObject(ObjID, OpenMode.ForRead)
      If tagObject.IsAnchored Then
        Dim tagAnchor As AnchorExtendedTagToEntity = trans.GetObject(tagObject.AnchorId, OpenMode.ForRead)
        Dim hostID As Autodesk.AutoCAD.DatabaseServices.ObjectId = tagAnchor.ReferencedEntityId
        Dim hostObj As Object = trans.GetObject(hostID, OpenMode.ForRead)
        If hostObj.GetType.ToString = "Autodesk.Aec.Building.Elec.DatabaseServices.Device" Then
          Dim deviceObj As Device = hostObj
          hostLocation = deviceObj.Location
          hostName = "Device"
        Else
          hostName = hostObj.GetType.ToString
          thisDoc.Editor.WriteMessage(vbLf & "Error: " & hostName & " not supported by GetHostLocation.")
        End If
      End If
      trans.Commit()
    Catch myErr As Exception

      thisDoc.Editor.WriteMessage(vbLf & myErr.ToString)
      trans.Abort()
    End Try
  End Using

  Return hostLocation
End Function

End Class
```