AS322865

Managing CAD Standards with VB.NET

Lee Ambrosius
Autodesk, Inc.

Learning Objectives

- Learn how to locate and reference the CAD Standards COM API
- Learn about how to use and the limits of the CAD Standards COM API
- Learn how to create and load a plug-in into AutoCAD
- Learn how to check for and fix CAD standards errors with a plug-in

Description

The CAD Standards feature in AutoCAD software is often left untapped by many companies because the functionality is limited to four types of named objects. What if you could create your own standards plug-ins that extend the functionality of the CAD Standards and Batch Standards Checkers? This class will help explain the CAD Standards COM API that ships with AutoCAD and how to implement your own plug-ins. You'll learn which libraries you need to use, how to create and register a plug-in, and how to load and use the plug-in in AutoCAD. You should have a basic understanding of the CAD Standards feature in AutoCAD and know how to use the VB.NET programming language.

Speaker(s)

Lee Ambrosius is a Principal Learning Experience Designer at Autodesk, Inc., for the AutoCAD® and AutoCAD LT products on Windows and Mac. He works primarily on the customization, developer, and CAD administration documentation along with the user documentation. Lee has presented at Autodesk University for about 15 years on a range of topics, from general AutoCAD customization to programming with the ObjectARX technology. He has authored several AutoCAD-related books, with his most recent project being *AutoCAD Platform Customization: User Interface, AutoLISP, VBA, and Beyond*. When Lee isn't writing, you can find him roaming various AutoCAD community forums, posting articles on his or the AutoCAD blog, or tweeting information regarding the AutoCAD product.

Twitter: @leeambrosius
Email: lee.ambrosius@autodesk.com
Blog: http://hyperpics.blogs.com

1 Introduction

CAD standards vary based on who you talk to, but one thing is always agreed upon though; that is, they need to be enforced. Autodesk first implemented the concept of a CAD standards feature back in AutoCAD 2002 which was a great idea that could have had a huge impact on the quality of drawings that a company created. The problem with a feature like CAD standards though is the magnitude of what CAD standards meant by each company and to the degree that they should be followed.

While the depth of the feature was not very great, it did introduce a platform in which companies and third-parties could extend. As you might know or maybe this is the first time you are hearing about this functionality, it wasn't well documented until recently on how to go about creating a CAD standards plug-in. This session goes into the basics of the API, what you will need to do to implement a custom plug-in. The CAD Standards feature consists of components inside and outside of AutoCAD; CAD Standards and Check Standards in AutoCAD, and the Batch Standards Checker outside of AutoCAD.

The plug-ins that ship with AutoCAD are used to validate four named object types stored in a drawing: layers, linetypes, text styles, and dimension styles. AutoCAD 2020 Update 1 introduced support for multileader (mleader) styles. While the plug-ins that ship with AutoCAD help to enforce named object types, you aren't limited to creating plug-ins that just work with named objects. It is possible to create a plug-in to work with graphical objects as well, such as enforcing viewports and title blocks to be on a specific layer. Any object that you can access through COM or the .NET APIs can essentially be validated because you define the rules of what makes an object valid or invalid.

2 What You Need Before Getting Started

Before you start creating a custom plug-in, you will need to obtain the following:

- **ObjectARX Software Development Kit (SDK)** – The ObjectARX SDK contains code samples, project templates, AutoCAD Managed .NET library files, and the AutoCAD Managed .NET Reference Guide. Along with those, you will also need the file *AcStMgr.tlb*.

  You can download the ObjectARX SDK from https://autode.sk/2NtvbnL.

- **Development Environment** – Visual Studio 2017 Update 2 is what you will need if you plan on developing applications for AutoCAD 2019 or AutoCAD 2020. If you are working with an earlier AutoCAD release, you will need a different version of Visual Studio. For more information on which release of Visual Studio to use, see Which Edition of Microsoft Visual Studio to Use (.NET) in the AutoCAD Developer Documentation online.

  The free version of Visual Studio known as Visual Studio Community can be downloaded by going to https://visualstudio.microsoft.com/vs/community/, since it is recommended to use Visual Studio 2017 you will need to download that release by going to https://visualstudio.microsoft.com/vs/older-downloads/ and following the on screen directions.

- **AutoCAD ActiveX: Reference Guide** – The AutoCAD ActiveX documentation will be useful as you will be working with COM objects unless you get the objects and then open them as AutoCAD Managed .NET objects. This session focuses on comparing AutoCAD COM objects.

  You can access the AutoCAD ActiveX Reference Guide online at https://help.autodesk.com/view/OARX/2020/ENU/ or locally by browsing to *C:\Program Files\Common Files\Autodesk Shared\acadauto.chm*.

- **AutoCAD ActiveX: CAD Standards Plug-ins Reference** – The AutoCAD CAD Standards Plug-ins documentation will help explain the objects and member functions that make up the CAD Standards API. You can access the AutoCAD CAD Standards Plug-ins documentation at https://help.autodesk.com/view/OARX/2020/ENU/.

- **AutoCAD: Managed .NET Developer's Guide** – The AutoCAD Managed .NET Developer's Guide contains information on how to work with the AutoCAD Managed .NET API, and there are many code samples that show the different aspects of the API. You can access the AutoCAD Managed .NET Developer's Guide at https://help.autodesk.com/view/OARX/2020/ENU/ by expanding the ObjectARX: Managed .NET Developer's Guide node in the table of contents.

- **Type Library Importer** – The Type Library Importer (*Tlbimp.exe*) utility is part of Visual Studio and the Windows SDK. *Tlbimp.exe* is needed to generate a DLL file from the CAD Standards plug-in library TLB file. The DLL file is what is referenced to your Visual Studio project.

  Based on the version of Visual Studio you install, you may need to also download and install the Windows SDK. *Tlbimp.exe* is installed with Visual Studio 2017 and later, and Visual Studio Community 2017 and later, so there is no need to install the Windows SDK unless you want to learn more about the Windows SDK which isn't required for creating a custom plug-in. You can download and install the Windows SDK from http://msdn.microsoft.com/en-us/windows/bb980924.aspx. Click Install Now and follow the on-screen directions. While the install is for the Windows 7 SDK, that isn't important unless you plan on building apps for Windows.

Create an Interop Assembly of the CAD Standards Type Library (TLB)

Before you get started, you need to generate an interop assembly of the CAD Standards type library. This gives you the necessary DLL to link to your VB.NET project.

1. On the Windows Start menu, click [All] Programs > Visual Studio 2017 > Developer Command Prompt for VS 2017.
2. In the Developer Command Prompt for VS 2017 window, type **cd "C:\Autodesk\ObjectARX_for_AutoCAD_2020_Win_64_bit\inc-x64"** and press Enter.

   If you installed the ObjectARX SDK to a different location, make sure to specify the *inc-x64* folder under that location.

   Note: If you are building plug-ins for AutoCAD 32-bit, you will want to specify the *inc-win32* folder instead of *inc-x64*.

3. Type **tlbimp acstmgr.tlb /machine:Agnostic** and press Enter.

   The *AcStMgr.dll* file should be added to the *inc-x64* folder.

   If you are building the DLL for AutoCAD 32-bit, you can create the *AcStMgr.dll* file executing **tlbimp acstmgr.tlb** in the Developer Command Prompt for VS 2017 window.

4. Type **tlbimp axdb23enu.tlb /machine:Agnostic** and press Enter.

   The file *AXDBLib.dll* will be added to the *inc-x64* folder.

If you are building a plug-in for AutoCAD 32-bit, you don't need to use the /machine:Agnostic switch in the Developer Command Prompt for VS 2017 window.

Note: The *cadStdsInterop.bat* file included in the dataset for this session can be modified/used to create the *AcStMgr.dll* and *AXDBLib.dll* files.

3 Defining a Plug-in

A custom plug-in is defined as a COM library and can't be loaded directly into AutoCAD, but rather the CAD Standards framework which is part of the CAD Standards feature in AutoCAD and the Batch Standards Checker. When you create a project for a plug-in, you use the Class Library template. After you create the project, the next steps are to reference the AutoCAD ActiveX COM and Standards Manager libraries.

Create a New Class Library Project for a Plug-in

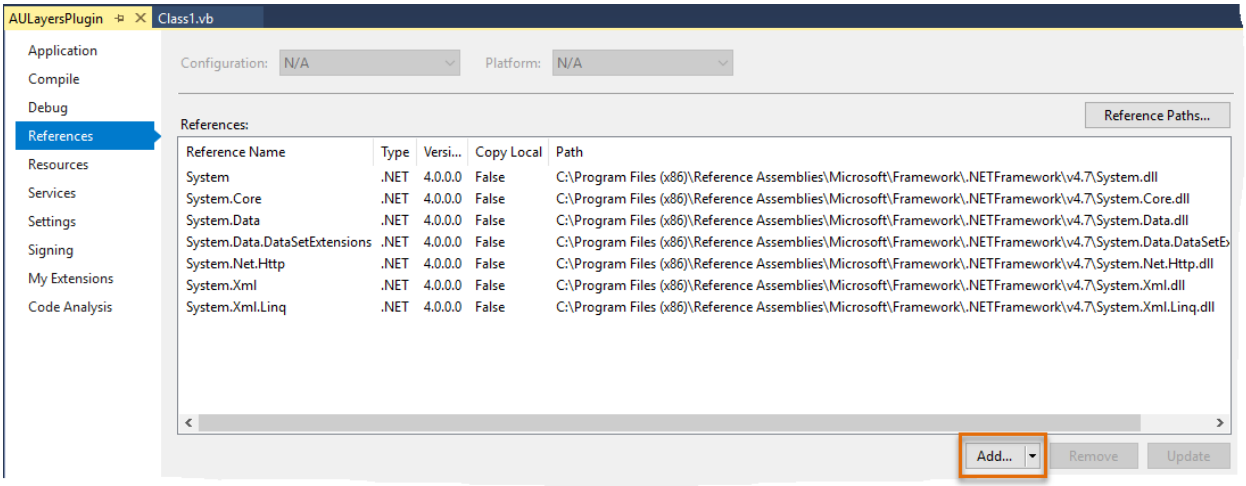
The following steps explain how to create a project for a new CAD Standard plug-in.

1. On the Windows Start menu, click [All] Programs > Visual Studio 2017.
2. In Microsoft Visual Studio 2017, click File menu > New Project.
3. In the New Project dialog box, under Installed, click Visual Basic and then select **Class Library (.NET Framework)**.
4. In the Name field, type a name from your project, such as **AULayersPlugin**.
5. In the Location field, specify a location for the new solution.
6. Clear the Create Directory for Solution check box.
7. Click the Framework drop-down list and choose .NET Framework 4.7. Click OK.

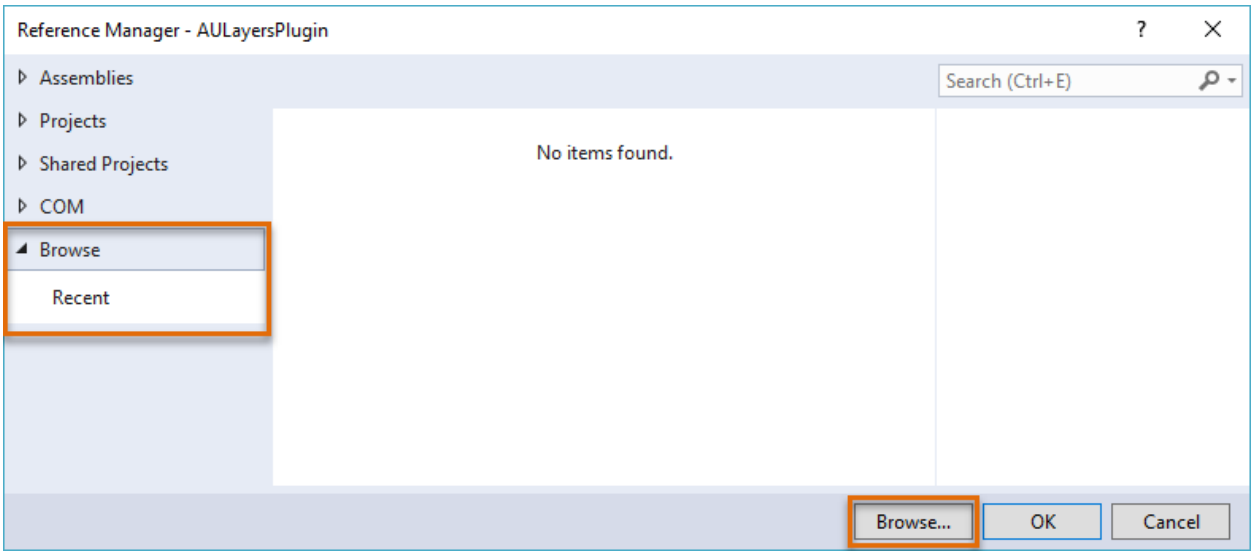
   If you are building a plug-in for AutoCAD 2018 or earlier, make sure to specify the appropriate version of the .NET Framework.

8. Click Project menu > AULayersPlugin Properties.

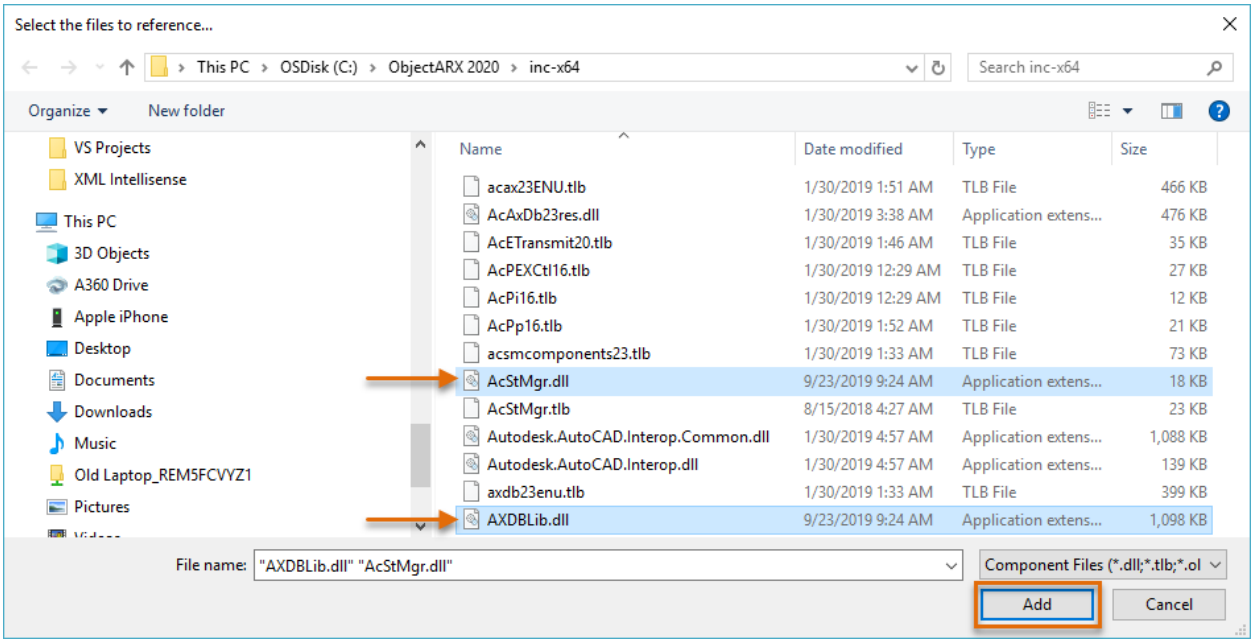
9. In the Properties dialog box, References tab, click Add.



10. In the Add Reference dialog box, click Browse and browse to *C:\Autodesk\ObjectARX_for_AutoCAD_2020_Win_64_bit\inc-x64* or the location in which the ObjectARX SDK was installed.

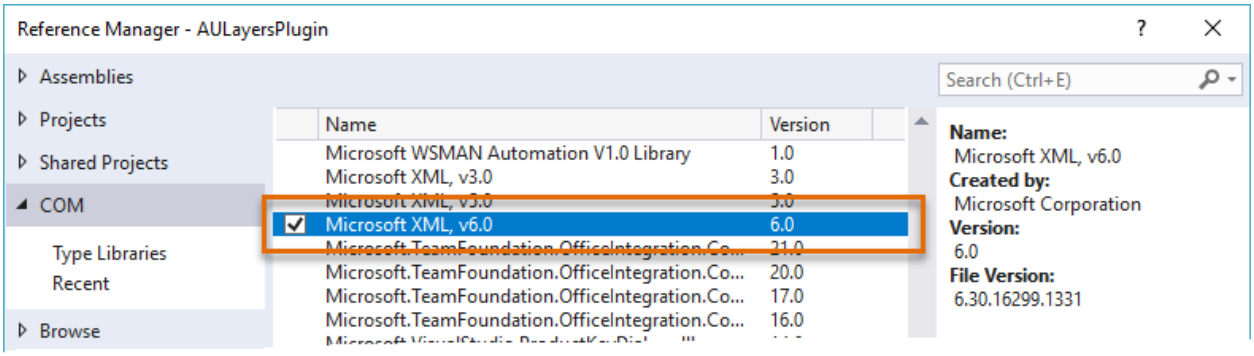


11. Press and hold Ctrl to select the *AcStMgr.dll* (was created earlier under Section 2) and *AXDBLib.dll* files. Click Add.

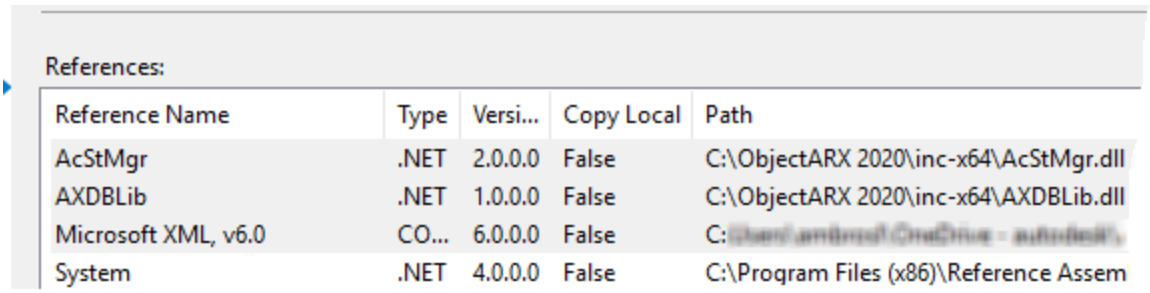


12. With the Reference Manager still open, on the COM tab, scroll to and check Microsoft XML Type library (Microsoft XML, v6.0).

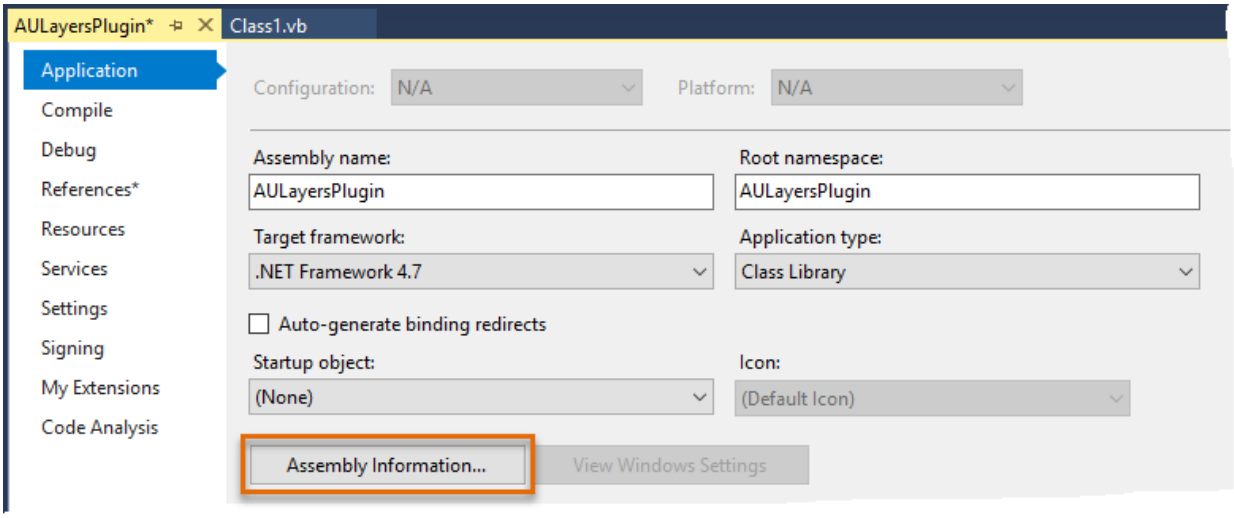
    This library will be used to manipulate the Batch Standards Checker report.



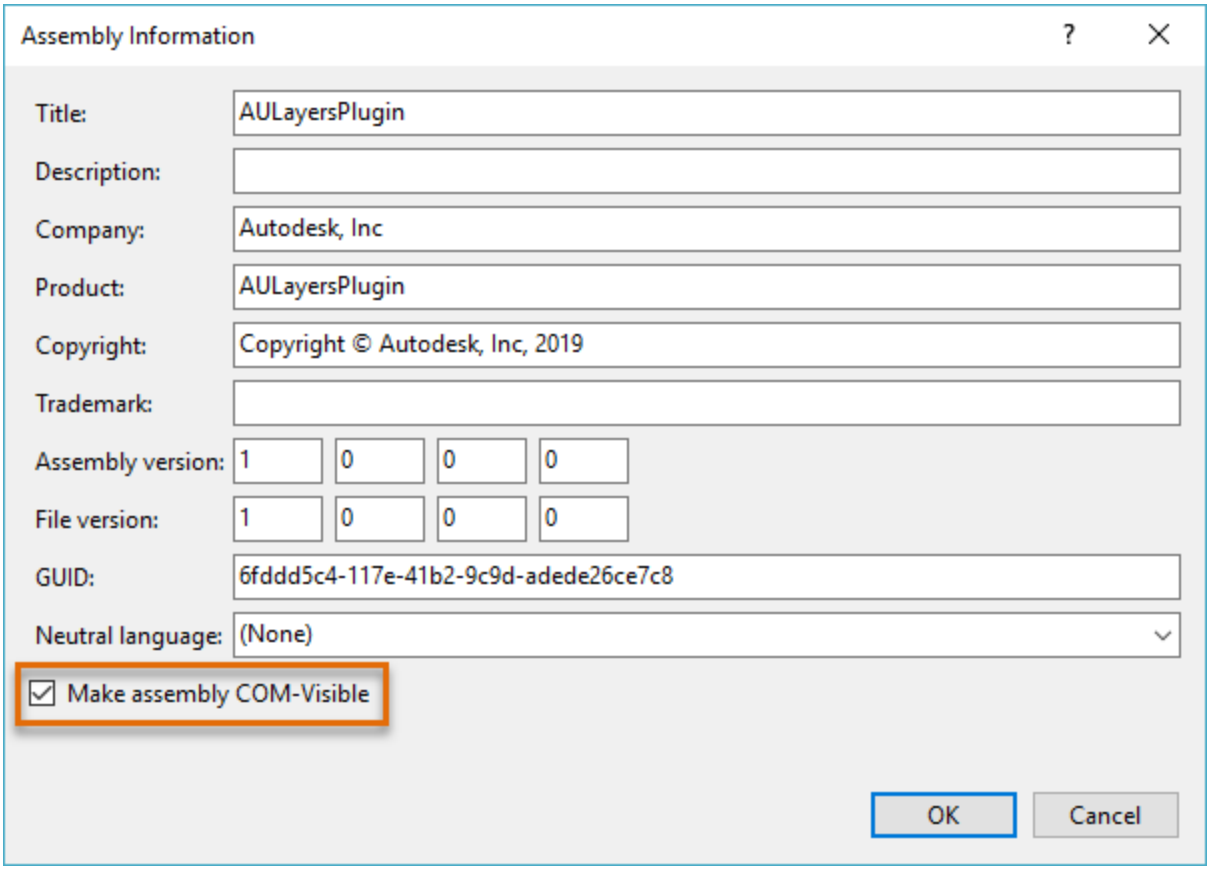
13. Click OK to close the Reference Manager.



14. In the Properties dialog box, on the Application tab, click Assembly Information.



15. In the Assembly Information dialog box, check Make Assembly COM-Visible.



16. Optionally, update other properties in the Assembly Information dialog box.
17. Click OK to save the changes and exit the dialog box.

18. Click File > Save All to save the Visual Studio project.

    The project is now ready for you to add the code that will define the behavior of the custom plug-in.

4 Basics of the CAD Standards API

The CAD Standards API is small but can be confusing. The `IAcStPlugin2` interface is used to implement several properties and methods required to define a plug-in.

These properties and methods are the same for each plug-in you might create and is how the CAD Standards framework is able to efficiently use each registered plug-in. While the API itself is not organized into different namespaces or groupings of methods and properties, there is a logical order in the way you should implement the necessary methods and properties for your custom plug-in.

The methods and properties fall into the following groupings:

- CAD Standards properties
- Initialization
- Error iterations; an error being a standards violation
- Retrieve errors and fixes
- Fix errors
- Report errors

CAD Standard Properties

Each plug-in must support a standard set of properties that are used to help the user identify what type of standards a plug-in might be used for, who created it, and its version. These properties must be implemented as part of a plug-in:

- `Author()` – Returns the name of the author for the plug-in, usually a company name
- `Description()` – Returns the description of the plug-in
- `HRef()` – Returns the URL where the plug-in can be downloaded from or information about the plug-in can be obtained
- `Icon()` – Returns the HICON property (icon) for the plug-in
- `Name()` – Returns the name of the plug-in
- `Version()` – Returns the version of the plug-in

```
Public ReadOnly Property Author() As String _
                Implements AcStMgr.IAcStPlugin2.Author
    Get
        Return "Lee Ambrosius, Autodesk, Inc."
    End Get
End Property
```

```
Public ReadOnly Property Description() As String _
                Implements AcStMgr.IAcStPlugin2.Description
    Get
        Return "Basic example of a Layer CAD Standards plug-in."
    End Get
End Property

Public ReadOnly Property HRef() As String _
                Implements AcStMgr.IAcStPlugin2.HRef
    Get
        Return "http://www.hyperpics.com/cadplugins"
    End Get
End Property

Public ReadOnly Property Icon() As Long _
                Implements AcStMgr.IAcStPlugin2.Icon
    Get
        Return 1
    End Get
End Property

Public ReadOnly Property Name() As String _
                Implements AcStMgr.IAcStPlugin2.Name
    Get
        Return "Basic Layers (AU 2019)"
    End Get
End Property

Public ReadOnly Property Version() As String _
                Implements AcStMgr.IAcStPlugin2.Version
    Get
        Return "1.0"
    End Get
End Property
```

In addition to the previously mentioned properties, you need to implement the `GetObjectFilter()` method which returns an array of the types of objects that the plug-in checks and receives notifications for. The array needs to be of the String data type and be the DXF name of the object type(s) to filter on. It is recommended to filter on a single object type. If you don't override the function or an object type is not provided, checks and notifications are performed on all object types which can result in performance problems on large complex drawings.

```
Public Function GetObjectFilter() As Object _
       Implements AcStMgr.IAcStPlugin2.GetObjectFilter

    Dim sFilterArray(0) As String
    sFilterArray(0) = "AcDbLayerTableRecord"

    Return sFilterArray
End Function
```

Initialization

The initialization of a plug-in is the very first thing that happens when it gets loaded into AutoCAD or the Batch Standards Checker. When a plug-in is initialized varies slightly based on one of the following conditions:

- Opening a drawing with an associated DWS file and the plug-in is enabled for checking, and real-time checking is enabled
- Enabling a plug-in in the CAD Standards dialog box or the Batch Standards Checker

The `Initialize()` method is the main entry point of a plug-in. This method is passed a reference to the Standards Manager object and is your opportunity to save a reference to this object along with the plug-in for future reference while the plug-in is active.

```
Public Sub Initialize(ByVal pStdsMgr As AcStMgr.AcStManager) _
           Implements AcStMgr.IAcStPlugin2.Initialize

    m_pManager = pStdsMgr
    m_pPlugin = Me
End Sub
```

After the plug-in is initialized, the `SetupForAudit()` method is called to setup the drawing that should be checked along with which drawings the plug-in should obtain standards information. The first parameter is the `Database` object of the drawing to be checked followed by the file path to the drawing file that is represented by the `Database` object. The third and fourth parameters are arrays that represent the names and paths of the drawing standards (DWS) files being used. The last parameter is an array of `Database` objects that represent each of the DWS files being used.

```
Public Sub SetupForAudit(ByVal pDb As AcadDatabase, _
                         ByVal szPathName As String, _
                         ByVal stdNameArray As Object, _
                         ByVal stdPathArray As Object, _
                         ByVal stdDbArray As Object) _
           Implements AcStMgr.IAcStPlugin2.SetupForAudit

    If Not IsNothing(pDb) Then
        Dim pStdLayer As AcadLayer
```

```
        Dim i, iDWS As Integer

        m_pCheckDatabase = pDb

        If Not IsNothing(m_pCheckDatabase) Then
            For Each acObj As AcadObject In m_pCheckDatabase.Layers
                If acObj.ObjectName = "AcDbLayerTableRecord" Then
                    m_ContextList.Add(acObj.ObjectID, True)
                End If
            Next
        End If

        i = 0
        For iDWS = 0 To UBound(stdDbArray)
            m_pDWSDatabase = stdDbArray(iDWS)

            For Each pStdLayer In m_pDWSDatabase.Layers
                Dim layerCache As New LayerCache()
                layerCache.Name = pStdLayer.Name
                layerCache.Color = pStdLayer.color
                layerCache.Lineweight = pStdLayer.Lineweight
                layerCache.StandardFileName = stdNameArray(iDWS)

                Dim pFix As New AcStMgr.AcStFix()
                pFix.Description = "Layer fix"
                pFix.StandardFileName = layerCache.StandardFileName
                pFix.FixObjectName = pStdLayer.Name

                If pFix.PropertyCount = 0 Then
                   pFix.PropertyValuePut("Color", pStdLayer.color)
                   pFix.PropertyValuePut("Lineweight", _
                                         pStdLayer.Lineweight)
                End If

                layerCache.StandardsFix = pFix
                ReDim Preserve m_LayerCacheArray(i)
                m_LayerCacheArray(i) = layerCache
                layerCache = Nothing
                pFix = Nothing
                i = i + 1
            Next

            m_pDWSDatabase = Nothing
        Next
    End If
End Sub
```

Error Iterations

You use a plug-in to check and fix violations based on a given set of standards. Before you can fix any errors, the errors must first be identified. Each plug-in must implement its own series of checks or rules to determine which conditions constitute an error. Errors can be found during the checking of an entire drawing or in real-time as objects are added to or modified in a drawing. The `SetContext()` method is used to control which objects should be checked and must be implemented for real-time notifications.

`SetContext()` accepts an array of objects that contains the objects to be checked as the result of a real-time standards violation and a Boolean that defines if the entire drawing or just the provided objects should be checked.

```
Public Sub SetContext(ByVal objIdArray As Object, _
                      ByVal bUseDb As Boolean) _
           Implements AcStMgr.IAcStPlugin2.SetContext

    m_ContextList.SetContext(bUseDb, objIdArray)
End Sub
```

The `Start()` method initializes the error checking process and is passed an error object that represents the first standards violation. Once checking is started, the `Next()` method is called to step through each error object. After an error is checked, the `Done()` method is called to determine if there are any more errors, if there are more errors `Next()` is called to continue checking errors. The `Clear()` method while related to iterating errors, is actually not called until the plug-in is about to be unloaded from memory.

The following shows an example of the `Start()`, `Next()`, `Done()`, and `Clear()` methods.

```
Public Sub PlugIn_Start(ByVal pStartError As AcStMgr.AcStError) _
           Implements AcStMgr.IAcStPlugin2.Start

    If IsNothing(pStartError) = False Then
        Dim badId As Long = pStartError.BadObjectId

        For m_curIndex = 0 To m_ContextList.Count - 1
            If m_ContextList.Item(m_curIndex) = badId Then
                m_curIndex = m_curIndex - 1
                PlugIn_Next()
            End If
        Next
    Else
        m_curIndex = -1
        PlugIn_Next()
    End If
End Sub

Public Sub PlugIn_Next() _
```

```
        Implements AcStMgr.IAcStPlugin2.Next
m_pError = Nothing

If m_ContextList.Count > 0 Then
    Dim layerObj As AcadLayer
    Dim iCache As Integer
    Dim bFoundError As Boolean, bNameFoundError As Boolean

    If m_LayerCacheArray.Length() > 0 Then
        If m_curIndex < m_ContextList.Count - 1 Then
            m_curIndex = m_curIndex + 1
            bFoundError = False
            bNameFoundError = False

            While m_curIndex < m_ContextList.Count
                layerObj = m_pCheckDatabase. _
                 ObjectIdToObject(m_ContextList.Item(m_curIndex))

                For iCache = LBound(m_LayerCacheArray) To _
                             UBound(m_LayerCacheArray)
                    If (layerObj.Name.CompareTo _
                      (m_LayerCacheArray(iCache).Name) <> 0) Then
                        bFoundError = True
                        bNameFoundError = True
                    Else
                        If layerObj.color.ToString() <> _
                                    m_LayerCacheArray(iCache). _
                                    Color.ToString() Or _
                           layerObj.Lineweight.ToString() <> _
                                    m_LayerCacheArray(iCache). _
                                    Lineweight.ToString() Then

                            bFoundError = True
                            bNameFoundError = False
                        Else
                            bFoundError = False
                            bNameFoundError = False
                        End If
                        Exit For
                    End If
                Next

                If bFoundError = True Then
                    Dim pError As New AcStMgr.AcStError()
                    pError.Description = "Layer is non-standard"
                    pError.BadObjectId = layerObj.ObjectID
```

```
                        pError.BadObjectName = layerObj.Name
                        pError.Plugin = m_pPlugin
                        pError.ErrorTypeName = "Layer "
                        pError.ResultStatus = _
                            AcStMgr.AcStResultStatus.acStResFlagsNone

                        If pError.PropertyCount = 0 Then
                            pError.PropertyValuePut("Color", _
                                                    layerObj.color)
                            pError.PropertyValuePut("Lineweight", _
                                                    layerObj.Lineweight)
                        End If

                        m_pError = pError
                        bFoundError = False
                        bNameFoundError = False
                        Exit While
                    End If

                    m_curIndex = m_curIndex + 1
                End While
            End If
        End If
    End If
End Sub

Public Function PlugIn_Done() As Boolean _
                Implements AcStMgr.IAcStPlugin2.Done
    Return IsNothing(m_pError)
End Function

Public Sub PlugIn_Clear() _
           Implements AcStMgr.IAcStPlugin2.Clear

    If IsNothing(m_xmlDoc) = False Then
        WriteStandardsItemsInfo()
    End If

    m_xmlDoc = Nothing
    m_pPlugin = Nothing
    m_curIndex = -1
    m_RecommendedFixIndex = -1
    m_FixCnt = 0
    m_sPropClrName = ""
    m_sPropLWName = ""
    m_pManager = Nothing
    m_pDWSDatabase = Nothing
```

```
    m_pCheckDatabase = Nothing
    m_ex = Nothing

    If IsNothing(m_pError) = False Then
        m_pError.Reset()
        m_pError = Nothing
    End If

    If IsNothing(m_LayerCacheArray) = False Then
        Dim i As Integer
        For i = 0 To UBound(m_LayerCacheArray)
          If IsDBNull(m_LayerCacheArray(i).StandardsFix) = False Then
                m_LayerCacheArray(i).StandardsFix.Reset()
                m_LayerCacheArray(i).StandardsFix = Nothing
          End If
        Next

        Erase m_LayerCacheArray
    End If

    If IsNothing(m_FixArray) = False Then
        Erase m_FixArray
    End If

    m_ContextList.Clear()
End Sub
```

Retrieve Errors and Fixes

Before you can fix a standards violation (an error), the violation must be identified and compared against all available fixes. The `GetError()` method is used to return the current error object. Once the current error object is returned, you can get a list of all possible fixes using the `GetAllFixes()` method. You pass the current error object, an empty array, and a variable that represents the recommended fix. When the `GetAllFixes()` method ends, the array and recommended fix variables are updated accordingly.

The recommended fix is used to apply an automatic fix to a standards violation. The `GetAllFixes()` method is not used by the Batch Standards Checker. The array of all fixes is defined by the `SetupForAudit()` method, but they need to be returned as part of an `IAcStFix` data type array.

Along with a list of all possible fixes, you will want to define a recommended fix which is defined by the `GetRecommendedFix()` method. How the recommended fix is determined is based on the type of object you are comparing, you might compare the name of an object or a geometric value like radius or length.

The `GetRecommendedFix()` method is not required for a plug-in, but is recommended so automatic fixes can be supported. The `GetPropertyDiffs()` method is used to provide a list

of the property differences based on the suggested or selected fix in the Check Standards dialog box against the error being checked.

```
Public Function GetError() As AcStMgr.AcStError _
                Implements AcStMgr.IAcStPlugin2.GetError
    Return m_pError
End Function

Public Sub GetAllFixes(ByVal pError As AcStMgr.AcStError, _
                       ByRef fixArray As Object, _
                       ByRef recommendedFixIndex As Integer) _
           Implements AcStMgr.IAcStPlugin2.GetAllFixes

    If IsNothing(pError) = False Then
        Dim arr(UBound(m_LayerCacheArray)) As AcStMgr.IAcStFix
        Dim vErrorClrVal As ACAD_COLOR
        Dim vErrorLWVal As ACAD_LWEIGHT

        Dim i As Integer
        recommendedFixIndex = -1
        m_FixCnt = 0

        If m_LayerCacheArray.Length > 0 Then
            For i = LBound(m_LayerCacheArray) To _
                    UBound(m_LayerCacheArray)
                vErrorClrVal = pError.PropertyValueGet("Color")
                vErrorLWVal = pError.PropertyValueGet("Lineweight")
                arr(i) = m_LayerCacheArray(i).StandardsFix
            Next

            fixArray = arr
            m_FixArray = fixArray

            Dim tmpFix As New AcStMgr.AcStFix()
            tmpFix = GetRecommendedFix(pError)
            tmpFix = Nothing
            recommendedFixIndex = m_RecommendedFixIndex
        End If

        If recommendedFixIndex = -1 Then
            pError.ResultStatus = _
                AcStMgr.AcStResultStatus.acStResNoRecommendedFix
        End If
    End If
End Sub
```

```
Public Function GetRecommendedFix(
                        ByVal pError As AcStMgr.AcStError) _
                                As AcStMgr.AcStFix _
                Implements AcStMgr.IAcStPlugin2.GetRecommendedFix

    Dim pRecommendedFix As New AcStMgr.AcStFix()

    Dim nameToBeChecked As String
    If m_LayerCacheArray.Length = 0 Then
        pError.ResultStatus = _
               AcStMgr.AcStResultStatus.acStResNoRecommendedFix
    Else
        Dim tmpLayer As AcadLayer
        Dim layCache As LayerCache
        Dim tmpObjID As Long
        Dim i As Integer

        tmpObjID = pError.BadObjectId()

        tmpLayer = m_pCheckDatabase.ObjectIdToObject(tmpObjID)
        nameToBeChecked = tmpLayer.Name
        tmpLayer = Nothing
        layCache = m_LayerCacheArray(0)

        m_RecommendedFixIndex = 0

        For i = 0 To UBound(m_LayerCacheArray)
            If m_LayerCacheArray(i).Name = nameToBeChecked Then
                layCache = m_LayerCacheArray(i)
                m_RecommendedFixIndex = i
            End If
        Next

        pRecommendedFix.Description = "Layer fix"
        pRecommendedFix.StandardFileName = _
            m_LayerCacheArray(m_RecommendedFixIndex).StandardFileName

            pRecommendedFix.FixObjectName = _
                    m_LayerCacheArray(m_RecommendedFixIndex).Name

        If pRecommendedFix.PropertyCount = 0 Then
            pRecommendedFix.PropertyValuePut("Color", _
                 m_LayerCacheArray(m_RecommendedFixIndex).Color)
            pRecommendedFix.PropertyValuePut("Lineweight", _
                 m_LayerCacheArray(m_RecommendedFixIndex).Lineweight)
        End If
    End If
```

```
    GetRecommendedFix = pRecommendedFix
End Function

Public Sub GetPropertyDiffs(ByVal pError As AcStMgr.AcStError, _
                            ByVal pFix As AcStMgr.AcStFix, _
                            ByRef pPropNames As Object, _
                            ByRef pErrorValues As Object, _
                            ByRef pFixValues As Object, _
                            ByRef pFixableStatuses As Object) _
           Implements AcStMgr.IAcStPlugin2.GetPropertyDiffs

    If IsNothing(pError) = False And IsNothing(pFix) = False Then
        Dim sPropNames(1) As String
        Dim sErrorValues(1) As String
        Dim sFixValues(1) As String
        Dim bFixableStatuses(1) As Boolean

        Dim sPropName As String = ""
        Dim vErrorVal As Object = Nothing
        Dim vFixVal As Object = Nothing
        Dim i As Integer

        For i = 0 To pError.PropertyCount - 1
            pError.PropertyGetAt(i, sPropName, vErrorVal)
            m_sPropName = sPropName

            Try
                pFix.PropertyValueGet(sPropName, vFixVal)

                If (vErrorVal.CompareTo(vFixVal) <> 0) Then
                    sPropNames(i) = sPropName
                    sErrorValues(i) = vErrorVal.ToString
                    sFixValues(i) = vFixVal.ToString
                    bFixableStatuses(i) = True
                End If
            Catch
            End Try
        Next

        pPropNames = sPropNames
        pErrorValues = sErrorValues
        pFixValues = sFixValues
        pFixableStatuses = bFixableStatuses

        Erase sPropNames
        Erase sErrorValues
```

```
        Erase sFixValues
        Erase bFixableStatuses

        m_FixCnt = m_FixCnt + 1
    End If
End Sub
```

Fix Errors

After a standards violation has been identified within the `Next()` method, the `FixError()` method is used to apply the fix to the error so it matches the appropriate standard. If the `FixError()` method is not part of the plug-in, the user is not able to fix a standards violation and is presented with a message box letting them know errors cannot be fixed. The `FixError()` method is not used by the Batch Standards Checker.

When you fix an error in the `FixError()` method, you need to update the `ResultStatus` property of the error object with the value `AcStMgr.AcStResultStatus.acStResFixed`. If the error object couldn't be updated, you should set the property to the value of `AcStMgr.AcStResultStatus.acStResFixFailed`. The following shows an example of the `FixError()` method.

```
Public Sub FixError(ByVal pError As AcStMgr.AcStError, _
                    ByVal pFix As AcStMgr.AcStFix, _
                    Optional ByRef pFailedReason As String = "0") _
            Implements AcStMgr.IAcStPlugin2.FixError

    If IsNothing(pError) = False Then
        Dim badObj As AcadLayer
        Dim sFixClrVal As ACAD_COLOR
        Dim sFixLWVal As ACAD_COLOR
        Dim badObjID As Long

        badObjID = pError.BadObjectId

        badObj = m_pCheckDatabase.ObjectIdToObject(badObjID)

        If IsNothing(pFix) Then
            Dim tmpFix As New AcStMgr.AcStFix()
            tmpFix = GetRecommendedFix(pError)

            If IsNothing(tmpFix) Then
                pError.ResultStatus = _
                    AcStMgr.AcStResultStatus.acStResNoRecommendedFix
            Else
                pFix = tmpFix
                tmpFix = Nothing
```

```
                pFix.PropertyValueGet("Color", sFixClrVal)
                Try
                    badObj.color = sFixClrVal
                    pError.ResultStatus = _
                           AcStMgr.AcStResultStatus.acStResFixed
                Catch m_ex As Exception
                    pError.ResultStatus = _
                           AcStMgr.AcStResultStatus.acStResFixFailed
                End Try

                pFix.PropertyValueGet("Lineweight", sFixLWVal)
                Try
                    badObj.Lineweight = sFixLWVal
                    pError.ResultStatus = _
                           AcStMgr.AcStResultStatus.acStResFixed
                Catch m_ex As Exception
                    pError.ResultStatus = _
                           AcStMgr.AcStResultStatus.acStResFixFailed
                End Try
            End If
        Else
            pFix.PropertyValueGet("Color", sFixClrVal)
            Try
                badObj.color = sFixClrVal
                pError.ResultStatus = _
                       AcStMgr.AcStResultStatus.acStResFixed
            Catch m_ex As Exception
                pError.ResultStatus = _
                       AcStMgr.AcStResultStatus.acStResFixFailed
            End Try

            pFix.PropertyValueGet("Lineweight", sFixLWVal)
            Try
                badObj.Lineweight = sFixLWVal
                pError.ResultStatus = _
                       AcStMgr.AcStResultStatus.acStResFixed
            Catch m_ex As Exception
                pError.ResultStatus = _
                       AcStMgr.AcStResultStatus.acStResFixFailed
            End Try
        End If
    End If
End Sub
```

The `UpdateStatus()` method is used to update the status of the provided error object. Typically, there is no reason to override the default behavior, but you can use the method to determine which error object is being updated. The following shows an example of the `UpdateStatus()` method.

```
Public Sub UpdateStatus(ByVal pError As AcStMgr.AcStError) _
                        Implements AcStMgr.IAcStPlugin2.UpdateStatus

End Sub
```

Report Errors

Reporting errors is done in two different ways based on if you are checking standards in AutoCAD or through the Batch Standards Checker. As you fix a standards violation with the `FixError()` method, you are responsible with updating the `ResultStatus` property for the Standard Error object that you attempt to fix when the user clicks the Fix button or an automatic fix is applied.

If the 'Mark this Problem as Ignored' option is checked before Next is clicked, the Check Standards dialog box updates the `ResultStatus` property of the error object accordingly. After all violations have been reviewed, a message box is displayed. This message box is populated based on the responses of the user and automatic fixes performed using the recommended fixes. From the aspect of the API, the Check Standards dialog box keeps track of and reports on the total number of errors fixed or ignored once checking has been completed.

For the Batch Standards Checker, you must implement the `WritePluginInfo()` method to write related plug-in information out. Information about standards violations and which standards are ignored is handled for you. The one thing that the API and the Batch Standards Checker does not seem to handle is the outputting of information about the standards items that were used to validate the objects in the drawings that were checked.

The following shows an implementation of the `WritePluginInfo()` method and a custom method named `WriteStandardsItemsInfo()` which is used to output information about the standards items to the Standards Items section of the report. The `WriteStandardsItemsInfo()` method is called in the `Clear()` method before the plug-in is removed from memory.

```
Public Sub WritePluginInfo(ByVal pPluginInfoSectionNode As Object) _
           Implements AcStMgr.IAcStPlugin2.WritePluginInfo

    Dim pSectionNode As MSXML2.IXMLDOMNode = pPluginInfoSectionNode
    Dim xmlElem As MSXML2.IXMLDOMElement = _
           pSectionNode.ownerDocument.createElement("AcStPluginInfo")
    Dim pPluginInfoElement As MSXML2.IXMLDOMElement = _
           pSectionNode.appendChild(xmlElem)

    pPluginInfoElement.setAttribute("PluginName", Name())
    pPluginInfoElement.setAttribute("Version", Version())
    pPluginInfoElement.setAttribute("Description", Description())
    pPluginInfoElement.setAttribute("Author", Author())
```

```
    pPluginInfoElement.setAttribute("HRef", HRef())
    pPluginInfoElement.setAttribute("DWSName", "")
    pPluginInfoElement.setAttribute("Status", "1")

    m_xmlDoc = pSectionNode.ownerDocument
End Sub

Public Sub WriteStandardsItemsInfo()
    If IsNothing(m_xmlDoc) = False Then

        Dim nCnt As Integer = 0
        Dim pAcStDWSSection As MSXML2.IXMLDOMNode = _
              m_xmlDoc.getElementsByTagName("AcStDWSSection")(0)

        If IsNothing(pAcStDWSSection) = False Then
            For Each child As MSXML2.IXMLDOMNode In _
                                  pAcStDWSSection.childNodes
                If child.nodeName = "AcStDWSFile" Then
                    For Each grandchild As MSXML2.IXMLDOMNode In _
                                        Child.childNodes
                        If grandchild.nodeName = "AcStDWSPlugin" Then

                            Dim pDWSId As String = _
                                grandchild.attributes.getNamedItem( _
                                "DWSId").nodeValue.ToString()

                            Dim pPluginId As String = _
                                grandchild.attributes.getNamedItem(_
                                "PluginId").nodeValue.ToString()

                            Dim pPluginName As String = _
                                grandchild.attributes.getNamedItem( _
                                "PluginName").nodeValue.ToString()

                            Dim pPluginVersion As String = _
                                grandchild.attributes.getNamedItem( _
                                "PluginVersion").nodeValue.ToString()

                            If Name() = pPluginName And _
                               Version() = pPluginVersion Then
                                For Each pFix As LayerCache In _
                                                    m_LayerCacheArray

                                    Dim xmlElem As _
                                      MSXML2.IXMLDOMElement = _
                                      grandchild.ownerDocument. _
                                      createElement("AcStDWSItem")
```

```
                    Dim pAcStDWSItemElement As _
                        MSXML2.IXMLDOMElement = _
                      grandchild.appendChild(xmlElem)

                    pAcStDWSItemElement. _
                      setAttribute( _
                        "DWSId", pDWSId)
                    pAcStDWSItemElement. _
                      setAttribute( _
                        "PluginId", pPluginId)
                    pAcStDWSItemElement. _
                      setAttribute( _
                        "ItemName", pFix.Name)

                    Dim xmlElemProp1 As _
                        MSXML2.IXMLDOMElement = _
                        pAcStDWSItemElement. _
                        ownerDocument. _
                        createElement("Property")

                    Dim pAcStDWSItemElementProp1 As _
                        MSXML2.IXMLDOMElement = _
                        pAcStDWSItemElement. _
                        appendChild(xmlElemProp1)

                    pAcStDWSItemElementProp1. _
                        setAttribute( _
                        "ItemDescription", _
                        "Color = " + _
                        pFix.Color.ToString())

                    Dim xmlElemProp2 As _
                        MSXML2.IXMLDOMElement = _
                        pAcStDWSItemElement. _
                        ownerDocument. _
                        createElement("Property")

                    Dim pAcStDWSItemElementProp2 As _
                        MSXML2.IXMLDOMElement = _
                        pAcStDWSItemElement. _
                        appendChild(xmlElemProp2)

                    pAcStDWSItemElementProp2. _
                        setAttribute( _
                        "ItemDescription", _
                        "Lineweight = " + _
```

```
                                        pFix.Lineweight.ToString())
                                End If
                            End If
                        Next
                    End If
                Next
            End If
        End If
    End Sub
```

Other Methods to Implement

In addition to the methods previously mentioned, you need to implement the `CheckSysvar()` and `StampDatabse()` methods.

```
Public Sub CheckSysvar(ByVal syvarName As String, _
                       ByVal bGetAllFixes As Boolean, _
                       ByRef bPassFail As Boolean) _
           Implements AcStMgr.IAcStPlugin2.CheckSysvar

End Sub

Public Sub StampDatabase(ByVal pDb As AcadDatabase, _
                         ByRef pStampIt As Boolean) _
           Implements AcStMgr.IAcStPlugin2.StampDatabase

    pStampIt = False
    If pDb.Layers.Count > 0 Then
        pStampIt = True
    End If
End Sub
```

5 Load and Use a Plug-in

Once you have defined and compiled your plug-in, you need to register the DLL with both AutoCAD and Windows before it can be used in either the CAD Standards or the Batch Standards Checker. Once the plug-in has been registered, it is automatically loaded by AutoCAD or the Batch Standards Checker the next time they are loaded.

The registration of the plug-in is done in two parts:

- Register the DLL assembly with Windows as a COM object
- Registry the ProgId assigned to the plug-in with AutoCAD

Register a Plug-in with Windows

You register a plug-in with Windows through the merging of a Registry file that is created from your assembly DLL using the *RegAsm.exe* utility.

Note: You can't register the DLL using the *RegSrv32.exe* utility.

The following steps explain how to create a REG file for your plug-in so it can be registered properly with Windows:

1. On the Windows Start menu, click [All] Programs > Visual Studio 2017 > Developer Command Prompt for VS 2017.

   Note: Based on your system, you might need to run Developer Command Prompt for VS 2017 as an Administrator.

2. In the Developer Command Prompt for VS 2017 window, type **cd "C:\CAD Plug-ins"** and press Enter.

   Substitute *C:\CAD Plug-ins* with the location of your plug-in. The location of your DLL can be found in the Output window after building it in Visual Studio.

3. Type **RegAsm.exe /codebase /regfile:AULayersPluginCOM.reg AULayersPlugin.dll** and press Enter.

   Note: The REG file generated will contain the current location of the DLL, so you will need to make changes to the file if you want to install the file elsewhere. While you can use a network location to deploy the DLL file, the file will be locked upon being loaded into memory on a workstation and make it difficult to update in the future. It is recommended to deploy the files locally using a plug-in bundle or a startup script.

4. In Windows Explorer or File Explorer, double-click the REG file. Click Yes to allow the file to be merged and then OK in the messages boxes that are displayed.

   If you were creating a plug-in bundle or an application installer, make sure to add the REG file as part of the plug-in bundle or installer.

Tip: Build events in Visual Studio can be used to execute *RegAsm.exe* to create the REG file each time the DLL is compiled. The following shows an example using *RegAsm.exe* and *RegEdt32.exe* in build events to create and merge the REG file:

```
C:\Windows\Microsoft.NET\Framework\v4.0.30319\RegAsm.exe /codebase
/regfile:"$(TargetDir)AULayersPluginCOM.reg" "$(TargetPath)"
regedt32.exe /S "$(TargetDir)AULayersPluginCOM.reg"
```

Register a Plug-in with AutoCAD

After you register your plug-in with Windows, you must update the following Windows Registry key to tell AutoCAD and the Batch Standards Checker about the plug-in:

HKEY_LOCAL_MACHINE\SOFTWARE\Autodesk\Drawing Check\Plugins2

You can define the Widows Registry key using one of the following:

- AutoLISP using the *vl-registry-write* function
- VB.NET using the *SaveSetting* method
- Define and merge a REG file with Windows manually, a plug-in bundle, or an installer

The following steps explain how to create a REG file to register the plug-in with AutoCAD:

1. Create a new file in Notepad (On the Windows Start menu, click [All] Programs > Windows Accessories > Notepad) and enter the following:

   REGEDIT4

   [HKEY_LOCAL_MACHINE\SOFTWARE\Autodesk\Drawing Check\Plugins2\AUPlugins.BasicLayers]

   AUPlugins.BasicLayers in the above example will need to be updated with the ProgId assigned to your class in the VB.NET project. For the AULayersPlugin project in the *Dataset* folder, the ProgId is AUPlugins.BasicLayers which is specified in the *AULayersPlugin.vb* file.

```
55      Imports AULayersPlugin.StandardsHelpers
56
57      <ProgId("AUPlugins.BasicLayers"), ComClass()> _
        1 reference
58      Public Class BasicLayers
59          Implements AcStMgr.IAcStPlugin2
```

2. Save the file in Notepad with a REG file extension. (Click File menu > Save As, and then choose All Files from the Save As Type drop-down list. Enter the file name with **.reg** added to the end and click Save.)
3. In Windows Explorer or File Explorer, double-click the file. Click Yes to allow the file to be merged and then OK in the messages boxes that are displayed.

Use and Test a Plug-in

Each plug-in should support the following workflows:

- CAD Standards checking in AutoCAD
- Automatic fixes when checking standards in AutoCAD
- Real-time checking in AutoCAD, notifications
- Batch Standards Checker

Use the following steps to test the checking and fixing of errors with a plug-in:

1. In AutoCAD, on the ribbon, click Manage tab > CAD Standards panel > Configure.
2. In the Configure Standards dialog box, Standards tab, click Add and select the DWS files you want to use.
3. On the Plug-ins tab, choose your plug-in so it is checked.
4. Click Check Standards to start checking the drawing against the associated DWS files.
5. Ignore some issues, fix some, and skip others to test all functionality.

Use the following steps to test the automatic fixing of errors with a plug-in:

1. In AutoCAD, on the ribbon, click Manage tab > CAD Standards panel > Configure.
2. In the Configure Standards dialog box, Standards tab, click Add and select the DWS files you want to use.
3. On the Plug-ins tab, choose your plug-in so it is checked.
4. Click Settings.
5. In the CAD Standards Settings dialog box, under Check Standards Settings, click Automatically Fix Non-Standard Properties.
6. In the Preferred Standards File to Use for Replacements drop-down list, choose the standards file to use for automatic fixes.
7. Click OK.
8. Click Check Standards.
9. Verify all automatic fixes were applied, and that only defects that do not match a recommended fix are being displayed.

Use the following steps to test the real-time checking of a plug-in:

1. In AutoCAD, on the ribbon, click Manage tab > CAD Standards panel > Configure.
2. In the Configure Standards dialog box, Standards tab, click Add and select the DWS files you want to use.
3. On the Plug-ins tab, choose your plug-in so it is checked. Click OK.
4. Modify the drawing so it goes against the standards that are being monitored.

Use the following steps to test a plug-in with the Batch Standards Checker:

1. On the Windows Start menu, click [All] Programs > AutoCAD 2020 > Batch Standards Checker.
2. In the Batch Standards Checker, Drawings tab, click Add and specify the drawings to be checked.
3. On the Standards tab, click Check Each Drawing Using Its Associated Standards Files.

   If you want to specify a set of drawing standards files, click Check All Drawings Using the Following Standards Files and then add each DWS file to use.
4. On the Plug-ins tab, choose your plug-in so it is checked.
5. Click Start Check and save the CHX file.

   The CHX file stores which drawing files to check, the standards files and plug-ins to use, and the final output report.
6. View the output report and then close the Batch Standards Checker.

6 Debugging a Plug-in

Debugging a plug-in created with VB.NET is handled slightly differently than other .NET assemblies that you might have created and loaded into AutoCAD before.

Debug a Plug-in in AutoCAD

The following explains how to setup a plug-in for debug:

1. Register your plug-in if you haven't already done so.

   Section 5 of this handout mentions how to create the necessary REG files needed to register a plug-in.
2. In Microsoft Visual Studio 2017, click Project menu > <*Project*> Properties.
3. In the Properties dialog, click the Debug tab.
   a. Click Start External Program and click [..] to the right of the text box.
   b. In the Select File dialog box, browse to and select *acad.exe*. Click Open.

      By default, the *acad.exe* for AutoCAD 2020 is located under *C:\Program Files\Autodesk\AutoCAD 2020*.
4. In the code editor window, set the desired breakpoints.

   Remember that the `Initialize` method is the main entry point of a plug-in.
5. Click Debug menu > Start Debugging.

   AutoCAD 2020 should launch.
6. In AutoCAD 2020, open the drawing file that you want to check.
7. On the ribbon, click Manage tab > CAD Standards panel > Configure.
8. In the Configure Standards dialog box, Standards tab, click Add and select the DWS files you want to use.
9. On the Plug-ins tab, choose your CAD Standard plug-in so it is checked.
10. Click Check Standards to start checking the drawing.

    Focus is shifted to the first breakpoint in Visual Studio.

7 Create a Graphical Plug-in

The example project that has been shown throughout this handout has focused on how to implement a plug-in that compares layers in a drawing against those in a DWS file. A plug-in can also be designed to compare graphical objects in a drawing against those in a DWS file.

The main difference between a plug-in that compares non-graphical vs graphical objects is the objects being filtered upon as part of the `SetupForAudit()` method. Once you have filtered the objects you want to check, you then identify the properties of the graphical objects that should be compared. For example, you might compare the scale of blocks or the layers on which they are inserted along with the coordinates at which they are inserted.

You could also create a plug-in that checks the placement of geometry in relationship to other objects. For example, you could create a validation tool to make sure objects were position and orientated together correctly. I have written utilities in the past that checked the relationships of blocks and could have leveraged the CAD Standards framework for some of the functionality I implemented.

The properties of a graphical objects that want to compare affects how you might go about implementing part of a plug-in. A sample plug-in project that compares a block (title block to be more specific) in Paper space against the same block in the DWS file has been included with the dataset for this session. It makes sure that the layer and insertion points of the two blocks match, and if not, then the CAD Standards framework lets you know they don't match.

The insertion point comparison is done by comparing two string values. String values are often the easiest thing to compare against at times, but you could compare the values through different means.

For the graphical Title Block plug-in sample, I used the Layers plug-in sample and made these changes:

- Changed the `LayerCache` class to the `BlockCache` class in *StandardsHelp.vb*
- Class name, ProgId, along with some global variables
- `Description()` and `Name()` methods
- `GetObjectFilter()` method
- `SetupForAudit()` method
- `PlugIn_Next()` and `PlugIn_Clear()` methods
- `GetAllFixes()` and `GetRecommendedFix()` methods
- `FixError()` method
- `WriteStandardsItemsInfo()` method

You should notice that quite a bit of the functionality is consistent between both plug-ins, which makes it rather easy to create a new plug-in once you have created your first one.

8 Where to Get More Information

When you are new to an API library or feature, you will most likely have questions of which you will need answers. The following is a list of resources that you can use to get help:

- **AutoCAD ActiveX and CAD Standards Plug-in References** – The AutoCAD ActiveX and CAD Standards plug-in documentation will be helpful if you are new to both. You can access these online at: https://help.autodesk.com/view/OARX/2020/ENU/
- **AutoCAD Managed .NET Developer's Guide** – The AutoCAD Managed .NET Developer's Guide in the AutoCAD Developer Documentation contains a lot of information on using the AutoCAD Managed .NET API to create custom programs. To access the help online, go to: https://help.autodesk.com/view/OARX/2020/ENU/
- **ObjectARX SDK** – While it is named the ObjectARX SDK, it contains many samples for the AutoCAD Managed .NET API along with the CAD Standards plug-in library that you will need. For information on the ObjectARX SDK, see https://autode.sk/2NtvbnL

- **Through the Interface (Blog)** – Kean Walmsley (AutoCAD Software Architect) writes a wide range of articles on using AutoCAD APIs along with other technologies. You can visit Kean's blog at http://through-the-interface.typepad.com
- **AutoCAD DevBlog (Blog)** – The Developer Technical Services (DevTech) team offers a wide range of articles on using the various programming APIs that are available for use with AutoCAD and specialized toolsets. Visit the ADN blog at https://adndevblog.typepad.com/autocad/
- **Autodesk Developer's Network** – If you are serious about developing applications for AutoCAD and specialized toolsets, you should consider becoming a registered Autodesk Developer. For information on registering as an Autodesk Developer, see https://www.autodesk.com/adn
- **Autodesk Discussion Forums** – The Autodesk forums provide peer-to-peer networking and some interaction with Autodesk moderators. You can ask a question about anything AutoCAD related and get a response from a fellow user or Autodesk employee.

  To access the .NET forums, go to https://forums.autodesk.com/t5/net/bd-p/152

  To access the ActiveX forums, go to https://forums.autodesk.com/t5/visual-basic-customization/bd-p/33
- **AUGI Forums** – The AUGI forums provide peer-to-peer networking where you can ask questions about virtually anything in AutoCAD or Autodesk and get a response from a fellow user. Visit AUGI at https://www.augi.com/
- **Industry Events and Classes** – Industry events such as Midwest University and Autodesk University are great places to learn about new features in an Autodesk product. Along with industry events, you might also be able to find classes at your local technical college or Autodesk Authorized Training Center (ATC).
- **Internet** – There are tutorials and information scattered across the Internet that cover the basics of ActiveX and VB.NET along with specific information related to using the ActiveX and Managed .NET APIs for AutoCAD. Use your favorite search engine, such as Google or Bing, to find this information.
- **Books** – There are many books out there that cover ActiveX and Managed .NET APIs along with programming languages like VB.NET. Search an online book reseller, such as Amazon (amazon.com) or Barnes & Noble (bn.com), for books that cover topics of interest.