Autodesk University 2012

Plotting and Publishing Drawings with .NET

Lee Ambrosius – Autodesk, Inc.

CP2071 The tasks of plotting and publishing drawings occur every day as firms send information to a client or generate prints for review. The AutoCAD® .NET API allows you to create viewports, configure layouts with the desired page setup settings, and then plot or publish the layout using the desired device and settings. Creating custom programs that automate the process of setting up and plotting a layout can save time and ensure that plots are created consistently. You should have an understanding of the AutoCAD .NET API when taking this class.

Learning Objectives

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

- Create and import named layouts
- Create and set the scale of viewport objects
- Access and assign plot configurations, plot styles, and page setups to a layout
- Plot and publish individual drawings and sheet set drawing files

About the Speaker

Lee is one of the technical writers on the AutoCAD team at Autodesk and has been an AutoCAD® user for over 15 years in the fields of architecture and facilities management. He has been teaching AutoCAD users for over a decade at both the corporate and college level. He is best known for his expertise in programming and customizing AutoCAD-based products, and has 10+ years of experience programming with AutoLISP®, VBA, Microsoft® .NET, and ObjectARX®. Lee has written articles for AUGI® publications and white papers for Autodesk on customization. He is the author of several books on AutoCAD and has been an active technical editor for AutoCAD books in the Bible and For Dummies series.

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

1 Introduction

Plotting is one of the more complex areas of AutoCAD for new users to work with because of the need to understand a wide range of concepts. These concepts include plot styles, plot and viewport scales, and different output formats. While templates do go a long way to make things easier for users by defining page setups, layouts, and viewports among other settings; trying to include everything in a single drawing template can make things more complicated for a user.

The Managed .NET API allows you to automate many of the tasks related to outputting a drawing to hardcopy or electronic file format. The areas of functionality that this session covers are:

- Creating a new layout in a drawing and importing an existing layout from an external drawing
- Creating and modifying viewports on a named layout
- Listing the available plot configurations and plot styles, and assigning them to a layout or page setup
- Working with render presets and visual styles
- Creating and assigning a named page setup
- Plotting a single layout to a hardcopy or electronic file
- Publishing specified layouts in multiple drawings or the drawings in a sheet set

Namespaces you will need to work with are:

- Autodesk.AutoCAD.ApplicationServices
- Autodesk.AutoCAD.Colors
- Autodesk.AutoCAD.DatabaseServices
- Autodesk.AutoCAD.Geometry
- Autodesk.AutoCAD.GraphicsInterface
- Autodesk.AutoCAD.PlottingServices
- Autodesk.AutoCAD.Runtime

2 Named Layouts

Named layouts are used when outputting a design and represent a physical sheet of paper in a digital drafting environment. Viewports are placed on a named layout to display Model space geometry at a specified scale. In addition to viewports, named layouts also might also contain annotation objects. The annotation objects that are commonly found on a named layout are:

- Title blocks (BlockReference objects)
- Bill of materials (Table, Lines, Text, Mtext, and BlockReference objects)
- Design dimensions (Dimension objects)
- General notes that apply to a project (MLeader, Text, Mtext, and Table objects)

From the API point-of-view, a named layout is a container object that holds a pointer to a block table record that represents one of the *PaperSpace*N* blocks in the drawing (PaperSpace0, PaperSpace1, …) and information related to output settings (PlotSettings object).

When you want to work with a layout in a drawing, you will need to work with:

- The **LayoutManager** interface class which allows you to work with the Layout Manager.
- The Layout dictionary object which allows you to step through each of the layouts in a drawing.

The **LayoutManager** interface is probably the most common way to work with layout objects, simply because most of the commands that you create will most likely operate on the current layout. You can use the **LayoutManager** interface to:

- Return the name of the current layout
- Create a new layout
- Duplicate (copy or clone) a layout
- Delete a layout
- Rename a layout

If you need to access the settings of any layout and not just the current layout, then you will need to work with the Layout dictionary of the drawing.

Class(es) you will need to work with:

- **DBDictionary** - Autodesk.AutoCAD.DatabaseServices.DBDictionary
- **DBDictionaryEntry** - Autodesk.AutoCAD.DatabaseServices -> DBDictionaryEntry Structure
- **LayoutManager** - Autodesk.AutoCAD.DatabaseServices.LayoutManager
- **Layout** - Autodesk.AutoCAD.DatabaseServices.Layout

The following samples show how to list the layouts in the current drawing, create a new layout, and import a layout and its contents from an external drawing.

```
' List all the layouts in the current drawing
<CommandMethod("LayoutList")> _
Public Shared Sub LayoutList()
    ' Get the current document and database
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
    Dim acCurDb As Database = acDoc.Database

    ' Get the layout dictionary of the current database
    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
        Dim lays As DBDictionary = _
            acTrans.GetObject(acCurDb.LayoutDictionaryId, OpenMode.ForRead)

        acDoc.Editor.WriteMessage(vbLf & "Layouts:")

        ' Step through and list each named layout and Model
        For Each item As DBDictionaryEntry In lays
            acDoc.Editor.WriteMessage(vbLf & "  " & item.Key)
        Next

        ' Abort the changes to the database
        acTrans.Abort()
    End Using
End Sub

' Create a new layout with the LayoutManager
<CommandMethod("LayoutCreate")> _
Public Shared Sub LayoutCreate()
    ' Get the current document and database
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
    Dim acCurDb As Database = acDoc.Database

    ' Get the layout and plot settings of the named pagesetup
    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
        ' Reference the Layout Manager
        Dim acLayoutMgr As LayoutManager = LayoutManager.Current

        ' Create the new layout with default settings
        Dim objID As ObjectId = acLayoutMgr.CreateLayout("newLayout")

        ' Open the layout
        Dim acLayout As Layout = acTrans.GetObject(objID, _
                                                   OpenMode.ForRead)

        ' Set the layout current if it is not already
        If acLayout.TabSelected = False Then
            acLayoutMgr.CurrentLayout = acLayout.LayoutName
        End If

        ' Output some information related to the layout object
        acDoc.Editor.WriteMessage(vbLf & "Tab Order: " & acLayout.TabOrder & _
                                  vbLf & "Tab Selected: " & acLayout.TabSelected & _
                                  vbLf & "Block Table Record ID: " & _
                                  acLayout.BlockTableRecordId.ToString())

        ' Save the changes made
```

```
            acTrans.Commit()
        End Using
    End Sub

    ' Import a layout from an external drawing
    <CommandMethod("LayoutImport")> _
    Public Shared Sub LayoutImport()
        ' Get the current document and database
        Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
        Dim acCurDb As Database = acDoc.Database

        ' Specify the layout name and drawing file to work with
        Dim layoutName As String = "MAIN AND SECOND FLOOR PLAN"
        Dim filename As String = "C:\Program Files\Autodesk\AutoCAD 2013\" & _
                                 "Sample\Sheet Sets\Architectural\A-01.dwg"

        ' Create a new database object and open the drawing into memory
        Dim acExDb As Database = New Database(False, True)
        acExDb.ReadDwgFile(filename, FileOpenMode.OpenForReadAndAllShare, True, "")

        ' Create a transaction for the external drawing
        Using acTransEx As Transaction = acExDb.TransactionManager.StartTransaction()

            ' Get the layouts dictionary
            Dim layoutsEx As DBDictionary = _
                acTransEx.GetObject(acExDb.LayoutDictionaryId, OpenMode.ForRead)

            ' Check to see if the layout exists in the external drawing
            If layoutsEx.Contains(layoutName) = True Then

                ' Get the layout and block objects from the external drawing
                Dim layEx As Layout = _
                    layoutsEx.GetAt(layoutName).GetObject(OpenMode.ForRead)
                Dim blkBlkRecEx As BlockTableRecord = _
                    acTransEx.GetObject(layEx.BlockTableRecordId, OpenMode.ForRead)

                ' Get the objects from the block associated with the layout
                Dim idCol As ObjectIdCollection = New ObjectIdCollection()
                For Each id As ObjectId In blkBlkRecEx
                    idCol.Add(id)
                Next

                ' Create a transaction for the current drawing
                Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

                    ' Get the block table and create a new block
                    ' then copy the objects between drawings
                    Dim blkTbl As BlockTable = _
                        acTrans.GetObject(acCurDb.BlockTableId, OpenMode.ForWrite)

                    Using blkBlkRec As New BlockTableRecord
                        blkBlkRec.Name = "*Paper_Space" & CStr(layoutsEx.Count() - 1)
                        blkTbl.Add(blkBlkRec)
                        acTrans.AddNewlyCreatedDBObject(blkBlkRec, True)
                        acExDb.WblockCloneObjects(idCol, _
```

```
                                blkBlkRec.ObjectId, _
                                New IdMapping(), _
                                DuplicateRecordCloning.Ignore, _
                                False)

        ' Create a new layout and then copy properties between drawings
        Dim layouts As DBDictionary = _
            acTrans.GetObject(acCurDb.LayoutDictionaryId, OpenMode.ForWrite)

        Using lay As New Layout
            lay.LayoutName = layoutName
            lay.AddToLayoutDictionary(acCurDb, blkBlkRec.ObjectId)
            acTrans.AddNewlyCreatedDBObject(lay, True)
            lay.CopyFrom(layEx)

            Dim plSets As DBDictionary = _
                acTrans.GetObject( _
                    acCurDb.PlotSettingsDictionaryId, _
                    OpenMode.ForRead)

            ' Check to see if a named page setup was assigned to the layout,
            ' if so then copy the page setup settings
            If lay.PlotSettingsName <> "" Then

                ' Check to see if the page setup exists
                If plSets.Contains(lay.PlotSettingsName) = False Then
                    plSets.UpgradeOpen()

                    Using plSet As New PlotSettings(lay.ModelType)

                        plSet.PlotSettingsName = lay.PlotSettingsName
                        plSet.AddToPlotSettingsDictionary(acCurDb)
                        acTrans.AddNewlyCreatedDBObject(plSet, True)

                        Dim plSetsEx As DBDictionary = _
                            acTransEx.GetObject( _
                                acExDb.PlotSettingsDictionaryId, _
                                OpenMode.ForRead)

                        Dim plSetEx As PlotSettings = _
                            plSetsEx.GetAt( _
                                lay.PlotSettingsName).GetObject( _
                                OpenMode.ForRead)

                        plSet.CopyFrom(plSetEx)
                    End Using
                End If
            End If
        End Using
    End Using

    ' Regen the drawing to get the layout tab to display
    acDoc.Editor.Regen()

    ' Save the changes made
```

```
                acTrans.Commit()
            End Using
        Else
            ' Display a message if the layout could not be found in the specified drawing
            acDoc.Editor.WriteMessage(vbLf & "Layout '" & layoutName & _
                                      "' could not be imported from '" & filename & "'.")
        End If

        ' Discard the changes made to the external drawing file
        acTransEx.Abort()
    End Using

    ' Close the external drawing file
    acExDb.Dispose()
End Sub
```

3 Floating Viewports

Floating viewports are created in a paper space block reference, and are used to display and scale Model space geometry for output. Floating viewports in the .NET API are represented by the **Viewport** object.

Once you create or obtain a valid Viewport object, you can change the following properties among others:

- Annotation and viewport scale
- Layer to control if the border of the viewport is plotted
- Size and view magnification
- Adjust the border of the viewport
- Switch between model and paper space

Class(es) you will need to work with:

- **Viewport** - Autodesk.AutoCAD.DatabaseServices.Viewport

The following sample shows how to create a new rectangular and non-rectangular viewport, and set some of the viewport's basic properties.

```
' Used to set a viewport current
<DllImport("accore.dll", CallingConvention:=CallingConvention.Cdecl, _
 EntryPoint:="?acedSetCurrentVPort@@YA?AW4ErrorStatus@Acad@@PEBVAcDbViewport@@@Z")> _
Public Shared Function acedSetCurrentVPort(ByVal AcDbVport As IntPtr) As IntPtr
End Function

' Used to create a rectangular and nonrectangular viewports
<CommandMethod("ViewportsCreateFloating")> _
Public Sub ViewportsCreateFloating()
    ' Get the current document and database, and start a transaction
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
    Dim acCurDb As Database = acDoc.Database

    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
```

```
' Open the Block table for read
Dim acBlkTbl As BlockTable = acTrans.GetObject(acCurDb.BlockTableId, _
                                               OpenMode.ForRead)

' Open the Block table record Paper space for write
Dim acBlkTblRec As BlockTableRecord = _
    acTrans.GetObject(acBlkTbl(BlockTableRecord.PaperSpace), _
                      OpenMode.ForWrite)

' Switch to the previous Paper space layout
Application.SetSystemVariable("TILEMODE", 0)
acDoc.Editor.SwitchToPaperSpace()

' Remove any viewports that already exist
For Each objId As ObjectId In acBlkTblRec
    Dim dbObj As DBObject = acTrans.GetObject(objId, OpenMode.ForRead)

    ' Remove any viewports in the block
    If TypeOf dbObj Is DatabaseServices.Viewport Then
        dbObj.UpgradeOpen()
        dbObj.Erase(True)
    End If
Next

' Create a Viewport
Using acVport1 As DatabaseServices.Viewport = New DatabaseServices.Viewport()
    ' Set the center point and size of the viewport
    acVport1.CenterPoint = New Point3d(3.75, 4, 0)
    acVport1.Width = 7.5
    acVport1.Height = 7.5

    ' Lock the viewport
    acVport1.Locked = True

    ' Set the scale to 1" = 4'
    acVport1.CustomScale = 48

    ' Set visual style
    Dim vStyles As DBDictionary = _
        acTrans.GetObject(acCurDb.VisualStyleDictionaryId, _
                          OpenMode.ForRead)

    acVport1.SetShadePlot(ShadePlotType.VisualStyle, _
                          vStyles.GetAt("Sketchy"))

    ' Add the new object to the block table record and the transaction
    acBlkTblRec.AppendEntity(acVport1)
    acTrans.AddNewlyCreatedDBObject(acVport1, True)

    ' Change the view direction and enable the viewport
    acVport1.ViewDirection = New Vector3d(-1, -1, 1)
    acVport1.On = True

    ' Create a rectangular viewport to change to a non-rectangular viewport
    Using acVport2 As DatabaseServices.Viewport = New DatabaseServices.Viewport()
```

```
acVport2.CenterPoint = New Point3d(9, 6.5, 0)
acVport2.Width = 2.5
acVport2.Height = 2.5

' Set the scale to 1" = 8'
acVport2.CustomScale = 96

' Set render preset
Dim namedObjs As DBDictionary = _
    acTrans.GetObject(acCurDb.NamedObjectsDictionaryId, _
                      OpenMode.ForRead)

' Check to see if the Render Settings dictionary already exists
Dim renderSettings As DBDictionary
If namedObjs.Contains("ACAD_RENDER_PLOT_SETTINGS") = True Then
    renderSettings = acTrans.GetObject( _
        namedObjs.GetAt("ACAD_RENDER_PLOT_SETTINGS"), _
        OpenMode.ForWrite)
Else
    ' If it does not exist, create it and add it to the drawing
    namedObjs.UpgradeOpen()
    renderSettings = New DBDictionary
    namedObjs.SetAt("ACAD_RENDER_PLOT_SETTINGS", renderSettings)
    acTrans.AddNewlyCreatedDBObject(renderSettings, True)
End If

' Create the new render preset, based on the settings
' of the Medium render preset
Using renderSetting As New MentalRayRenderSettings
    GetDefaultRenderPreset(renderSetting, "Medium")

    renderSetting.Name = "Medium"
    renderSettings.SetAt("Medium", renderSetting)
    acTrans.AddNewlyCreatedDBObject(renderSetting, True)

    acVport2.SetShadePlot(ShadePlotType.RenderPreset, _
                          renderSetting.ObjectId)
End Using

' Create a circle
Using acCirc As Circle = New Circle()
    acCirc.Center = acVport2.CenterPoint
    acCirc.Radius = 1.25

    ' Add the new object to the block table record and the transaction
    acBlkTblRec.AppendEntity(acCirc)
    acTrans.AddNewlyCreatedDBObject(acCirc, True)

    ' Clip the viewport using the circle
    acVport2.NonRectClipEntityId = acCirc.ObjectId
    acVport2.NonRectClipOn = True
End Using

' Add the new object to the block table record and the transaction
acBlkTblRec.AppendEntity(acVport2)
```

```
                acTrans.AddNewlyCreatedDBObject(acVport2, True)

                ' Change the view direction
                acVport2.ViewDirection = New Vector3d(0, 0, 1)

                ' Enable the viewport
                acVport2.On = True
            End Using

            ' Activate model space
            acDoc.Editor.SwitchToModelSpace()

            ' Set the new viewport current via an imported ObjectARX function
            acedSetCurrentVPort(acVport1.UnmanagedObject)
        End Using

        ' Save the new objects to the database
        acTrans.Commit()
    End Using
End Sub
```

4 Visual Styles and Render Presets

Visual styles and render presets are used to control the way objects appear on screen and during output. Both can be assigned to the Shade Plot setting of a viewport or plot settings (aka page setup). Plot settings affect the current values used by the Plot dialog box and those stored with a layout or page setup.

Visual styles allow you to affect the way 2D and 3D objects appear on screen and during output. Standard and custom visual styles are stored in the Visual Style dictionary. All visual styles can be customized, so you cannot expect each drawing that you might work with will contain all the standard visual styles that are part of the drawing templates that come with AutoCAD. Once you open the Visual Style dictionary, you can step through each visual style in the drawing or create a new visual style depending on your needs.

Render presets allow you to define the settings that should be used when rendering 3D objects in a viewport or a drawing during output. Render presets are stored in two different places; with the application and in a named dictionary within a drawing. The five standard render presets (Draft, Low, Medium, High, and Presentation) are stored with the application and cannot be modified with the .NET API. Custom render presets are stored in the "ACAD_RENDER_SETTINGS" named dictionary as of the MentalRayRenderSettings object type.

There is another named dictionary that you will need to work with when using render presets, this named dictionary is "ACAD_RENDER_PLOT_SETTINGS". Both of the named dictionaries are important and serve different purposes. You can think of the "ACAD_RENDER_SETTINGS" named dictionary as a way to store your render preset recipes, and the "ACAD_RENDER_PLOT_SETTINGS" named dictionary is product created from the recipe.

So what does this mean, you first create a render preset as part of the "ACAD_RENDER_SETTINGS" named dictionary and then you make a copy of it and add it to the "ACAD_RENDER_PLOT_SETTINGS" named dictionary. Yes, a copy of it, you cannot reference back to it like you work a TextStyleTableRecord or DimStyleTableRecord object. You then reference the copied render preset that is added to the "ACAD_RENDER_PLOT_SETTINGS" named dictionary for a Viewport, Layout, or PlotSettings object.

Classes you will need to work with:

- **DBDictionary** - Autodesk.AutoCAD.DatabaseServices.DBDictionary
- **DBDictionaryEntry** - Autodesk.AutoCAD.DatabaseServices -> DBDictionaryEntry Structure
- **DBVisualStyle** - Autodesk.AutoCAD.DatabaseServices.DBVisualStyle
- **MentalRayRenderSettings** - Autodesk.AutoCAD.DatabaseServices.MentalRayRenderSettings

```
' Lists the available visual styles
<CommandMethod("VisualStyleList")> _
Public Shared Sub VisualStyleList()
    ' Get the current document and database, and start a transaction
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
    Dim acCurDb As Database = acDoc.Database

    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
        Dim vStyles As DBDictionary = _
            acTrans.GetObject(acCurDb.VisualStyleDictionaryId, _
                              OpenMode.ForRead)

        ' Output a message to the Command Line history
        acDoc.Editor.WriteMessage(vbLf & "Visual styles: ")

        ' Step through the dictionary
        For Each entry As DBDictionaryEntry In vStyles
            ' Get the dictionary entry
            Dim vStyle As DBVisualStyle = _
                vStyles.GetAt(entry.Key).GetObject(OpenMode.ForRead)

            ' If the visual style is not marked for internal use then output its name
            If vStyle.InternalUseOnly = False Then
                ' Output the name of the visual style
                acDoc.Editor.WriteMessage(vbLf & "  " & vStyle.Name)
            End If
        Next
    End Using
End Sub

' Creates a new visual style
<CommandMethod("VisualStyleCreate")> _
Public Shared Sub VisualStyleCreate()
    ' Get the current document and database, and start a transaction
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
```

```
Dim acCurDb As Database = acDoc.Database

Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
    Dim vStyles As DBDictionary = _
        acTrans.GetObject(acCurDb.VisualStyleDictionaryId, _
                          OpenMode.ForRead)

    Try
        ' Check to see if the "MyVS" exists or not
        Dim vStyle As DBVisualStyle
        If vStyles.Contains("MyVS") = True Then
            vStyle = acTrans.GetObject(vStyles.GetAt("MyVS"), _
                                       OpenMode.ForWrite)
        Else
            vStyles.UpgradeOpen()

            ' Create the visual style
            vStyle = New DBVisualStyle
            vStyles.SetAt("MyVS", vStyle)

            ' Add the visual style to the dictionary
            acTrans.AddNewlyCreatedDBObject(vStyle, True)
        End If

        ' Set the description of the visual style
        vStyle.Description = "My Visual Style"
        vStyle.Type = VisualStyleType.Custom

        ' Face Settings (Opacity, Face Style, Lighting Quality, Color,
        '                Monochrome color, Opacity, and Material Display)
        vStyle.SetTrait(VisualStyleProperty.FaceModifier, _
                        VSFaceModifiers.FaceOpacityFlag, _
                        VisualStyleOperation.Set)
        vStyle.SetTrait(VisualStyleProperty.FaceLightingModel, _
                        VSFaceLightingModel.Gooch, _
                        VisualStyleOperation.Set)
        vStyle.SetTrait(VisualStyleProperty.FaceLightingQuality, _
                        VSFaceLightingQuality.PerPixelLighting, _
                        VisualStyleOperation.Set)
        vStyle.SetTrait(VisualStyleProperty.FaceColorMode, _
                        VSFaceColorMode.ObjectColor, _
                        VisualStyleOperation.Set)
        vStyle.SetTrait(VisualStyleProperty.FaceMonoColor, _
                        Color.FromColorIndex(ColorMethod.ByAci, 1), _
                        VisualStyleOperation.Set)
        vStyle.SetTrait(VisualStyleProperty.FaceOpacity, 0.5, _
                        VisualStyleOperation.Set)
        vStyle.SetTrait(VisualStyleProperty.DisplayStyle, _
                        VSDisplayStyles.MaterialsFlag + _
                        VSDisplayStyles.TexturesFlag, _
                        VisualStyleOperation.Set)

        ' Lighting (Enable Highlight Intensity,
        '           Highlight Intensity, and Shadow Display)
        vStyle.SetTrait(VisualStyleProperty.FaceModifier, _
```

```
                vStyle.GetTrait(VisualStyleProperty.FaceModifier).Int + _
                VSFaceModifiers.SpecularFlag, _
                VisualStyleOperation.Set)
vStyle.SetTrait(VisualStyleProperty.DisplayStyle, _
                vStyle.GetTrait(VisualStyleProperty.DisplayStyle).Int + _
                VSDisplayStyles.LightingFlag, _
                VisualStyleOperation.Set)
vStyle.SetTrait(VisualStyleProperty.FaceSpecular, _
                45.0, VisualStyleOperation.Set)
vStyle.SetTrait(VisualStyleProperty.DisplayShadowType, _
                VSDisplayShadowType.Full, _
                VisualStyleOperation.Set)

' Environment Settings (Backgrounds)
vStyle.SetTrait(VisualStyleProperty.DisplayStyle, _
                vStyle.GetTrait(VisualStyleProperty.DisplayStyle).Int + _
                VSDisplayStyles.BackgroundsFlag, _
                VisualStyleOperation.Set)

' Edge Settings (Show, Number of Lines, Color, and Always on Top)
vStyle.SetTrait(VisualStyleProperty.EdgeModel, VSEdgeModel.Isolines, _
                VisualStyleOperation.Set)
vStyle.SetTrait(VisualStyleProperty.EdgeIsolines, _
                6, VisualStyleOperation.Set)
vStyle.SetTrait(VisualStyleProperty.EdgeColor, _
                Color.FromColorIndex(ColorMethod.ByAci, 2), _
                VisualStyleOperation.Set)
vStyle.SetTrait(VisualStyleProperty.EdgeModifier, _
                vStyle.GetTrait(VisualStyleProperty.EdgeModifier).Int + _
                VSEdgeModifiers.AlwaysOnTopFlag, _
                VisualStyleOperation.Set)

' Occluded Edges (Show, Color, and Linetype)
If Not (vStyle.GetTrait(VisualStyleProperty.EdgeStyle).Int And _
        VSEdgeStyles.ObscuredFlag) > 0 Then
    vStyle.SetTrait(VisualStyleProperty.EdgeStyle, _
                    vStyle.GetTrait(VisualStyleProperty.EdgeStyle).Int + _
                    VSEdgeStyles.ObscuredFlag, _
                    VisualStyleOperation.Set)
End If
vStyle.SetTrait(VisualStyleProperty.EdgeObscuredColor, _
                Color.FromColorIndex(ColorMethod.ByAci, 3), _
                VisualStyleOperation.Set)
vStyle.SetTrait(VisualStyleProperty.EdgeObscuredLinePattern, _
                VSEdgeLinePattern.DoubleMediumDash, _
                VisualStyleOperation.Set)

' Intersection Edges (Color and Linetype)
If Not (vStyle.GetTrait(VisualStyleProperty.EdgeStyle).Int And _
        VSEdgeStyles.IntersectionFlag) > 0 Then
    vStyle.SetTrait(VisualStyleProperty.EdgeStyle, _
                    vStyle.GetTrait(VisualStyleProperty.EdgeStyle).Int + _
                    VSEdgeStyles.IntersectionFlag, _
                    VisualStyleOperation.Set)
End If
```

```
            vStyle.SetTrait(VisualStyleProperty.EdgeIntersectionColor, _
                            Color.FromColorIndex(ColorMethod.ByAci, 4), _
                            VisualStyleOperation.Set)
            vStyle.SetTrait(VisualStyleProperty.EdgeIntersectionLinePattern, _
                            VSEdgeLinePattern.ShortDash, _
                            VisualStyleOperation.Set)

            ' Silhouette Edges (Color and Width)
            If Not (vStyle.GetTrait(VisualStyleProperty.EdgeStyle).Int And _
                    VSEdgeStyles.SilhouetteFlag) > 0 Then
                vStyle.SetTrait(VisualStyleProperty.EdgeStyle, _
                                vStyle.GetTrait(VisualStyleProperty.EdgeStyle).Int + _
                                VSEdgeStyles.SilhouetteFlag, _
                                VisualStyleOperation.Set)
            End If
            vStyle.SetTrait(VisualStyleProperty.EdgeSilhouetteColor, _
                            Color.FromColorIndex(ColorMethod.ByAci, 5), _
                            VisualStyleOperation.Set)
            vStyle.SetTrait(VisualStyleProperty.EdgeSilhouetteWidth, 2, _
                            VisualStyleOperation.Set)

            ' Edge Modifiers (Enable Line Extensions, Enable Jitter,
            '                 Line Extensions, Jitter, Crease Angle,
            '                 and Halo Gap)
            If Not (vStyle.GetTrait(VisualStyleProperty.EdgeModifier).Int And _
                    VSEdgeModifiers.EdgeOverhangFlag) > 0 Then
                vStyle.SetTrait(VisualStyleProperty.EdgeModifier, _
                                vStyle.GetTrait(VisualStyleProperty.EdgeModifier).Int + _
                                VSEdgeModifiers.EdgeOverhangFlag, _
                                VisualStyleOperation.Set)
            End If
            If Not (vStyle.GetTrait(VisualStyleProperty.EdgeModifier).Int And _
                    VSEdgeModifiers.EdgeJitterFlag) > 0 Then
                vStyle.SetTrait(VisualStyleProperty.EdgeModifier, _
                                vStyle.GetTrait(VisualStyleProperty.EdgeModifier).Int + _
                                VSEdgeModifiers.EdgeJitterFlag, _
                                VisualStyleOperation.Set)
            End If
            vStyle.SetTrait(VisualStyleProperty.EdgeOverhang, 3, _
                            VisualStyleOperation.Set)
            vStyle.SetTrait(VisualStyleProperty.EdgeJitterAmount, _
                            VSEdgeJitterAmount.JitterMedium, _
                            VisualStyleOperation.Set)
            vStyle.SetTrait(VisualStyleProperty.EdgeCreaseAngle, _
                            0.3, VisualStyleOperation.Set)
            vStyle.SetTrait(VisualStyleProperty.EdgeHaloGap, _
                            5, VisualStyleOperation.Set)
        Catch es As Autodesk.AutoCAD.Runtime.Exception
            MsgBox(es.Message)
        Finally
            acTrans.Commit()
        End Try
    End Using
End Sub
```

```
' Utility to output a property value
' Usage: OutputVSPropValue(vStyle, VisualStyleProperty.EdgeHaloGap)
Private Shared Sub OutputVSPropValue(vStyle As DBVisualStyle, _
                                     vsProp As VisualStyleProperty)
    ' Get the current document
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

    Dim gIntVar As Autodesk.AutoCAD.GraphicsInterface.Variant = vStyle.GetTrait(vsProp)

    acDoc.Editor.WriteMessage(vbLf & gIntVar.Type.ToString() & ": ")

    If gIntVar.Type = Autodesk.AutoCAD.GraphicsInterface.VariantType.Boolean Then
        acDoc.Editor.WriteMessage(gIntVar.Boolean.ToString())
    ElseIf gIntVar.Type = Autodesk.AutoCAD.GraphicsInterface.VariantType.Color Then
        acDoc.Editor.WriteMessage(gIntVar.Color.ColorIndex.ToString())
    ElseIf gIntVar.Type = Autodesk.AutoCAD.GraphicsInterface.VariantType.Double Then
        acDoc.Editor.WriteMessage(gIntVar.Double.ToString())
    ElseIf gIntVar.Type = Autodesk.AutoCAD.GraphicsInterface.VariantType.Int Then
        acDoc.Editor.WriteMessage(gIntVar.Int.ToString())
    ElseIf gIntVar.Type = Autodesk.AutoCAD.GraphicsInterface.VariantType.String Then
        acDoc.Editor.WriteMessage(gIntVar.String.ToString())
    End If
End Sub

' Lists the available render presets
<CommandMethod("RenderPresetsList")> _
Public Shared Sub RenderPresetsList()
    ' Get the current document and database, and start a transaction
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
    Dim acCurDb As Database = acDoc.Database

    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
        Dim namedObjs As DBDictionary = _
            acTrans.GetObject(acCurDb.NamedObjectsDictionaryId, _
                              OpenMode.ForRead)

        ' Output a message to the Command Line history
        acDoc.Editor.WriteMessage(vbLf & "Default render presets: ")

        ' List the default render presets that are defined
        ' as part of the application
        acDoc.Editor.WriteMessage(vbLf & "  Draft")
        acDoc.Editor.WriteMessage(vbLf & "  Low")
        acDoc.Editor.WriteMessage(vbLf & "  Medium")
        acDoc.Editor.WriteMessage(vbLf & "  High")
        acDoc.Editor.WriteMessage(vbLf & "  Presentation")

        ' Check to see if the "ACAD_RENDER_SETTINGS" named dictionary exists
        If namedObjs.Contains("ACAD_RENDER_SETTINGS") = True Then
            ' Open the named dictionary
            Dim renderSettings As DBDictionary = _
                acTrans.GetObject(namedObjs.GetAt("ACAD_RENDER_SETTINGS"), _
                                  OpenMode.ForRead)

            ' Output a message to the Command Line history
```

```
            acDoc.Editor.WriteMessage(vbLf & "Custom render presets: ")

            ' Step through and list each of the custom render presets
            For Each entry As DBDictionaryEntry In renderSettings
                Dim renderSetting As MentalRayRenderSettings = _
                    acTrans.GetObject(entry.Value, OpenMode.ForRead)

                ' Output the name of the custom render preset
                acDoc.Editor.WriteMessage(vbLf & "   " & renderSetting.Name)
            Next
        Else
            ' If no custom render presets exist, then output the following message
            acDoc.Editor.WriteMessage(vbLf & _
                                      "No custom render presets available.")
        End If

        ' Check to see if the "ACAD_RENDER_PLOT_SETTINGS" named dictionary exists
        If namedObjs.Contains("ACAD_RENDER_PLOT_SETTINGS") = True Then
            ' Open the named dictionary
            Dim renderSettings As DBDictionary = _
                acTrans.GetObject(namedObjs.GetAt("ACAD_RENDER_PLOT_SETTINGS"), _
                                  OpenMode.ForRead)

            ' Output a message to the Command Line history
            acDoc.Editor.WriteMessage(vbLf & "Custom render plot presets: ")

            ' Step through and list each of the custom render presets
            For Each entry As DBDictionaryEntry In renderSettings
                Dim renderSetting As MentalRayRenderSettings = _
                    acTrans.GetObject(entry.Value, OpenMode.ForRead)

                ' Output the name of the custom render preset
                acDoc.Editor.WriteMessage(vbLf & "   " & renderSetting.Name)
            Next
        Else
            ' If no custom render plot presets exist, then output the following message
            acDoc.Editor.WriteMessage(vbLf & _
                                      "No custom render plot presets available.")
        End If

        ' Discard any changes
        acTrans.Abort()
    End Using
End Sub

' Creates a new render preset
<CommandMethod("RenderPresetsCreate")> _
Public Shared Sub RenderPresetCreate()
    ' Get the current document and database, and start a transaction
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
    Dim acCurDb As Database = acDoc.Database

    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
        Dim namedObjs As DBDictionary = _
            acTrans.GetObject(acCurDb.NamedObjectsDictionaryId, OpenMode.ForRead)
```

```
        Try
            ' Check to see if the Render Settings dictionary already exists
            Dim renderSettings As DBDictionary
            If namedObjs.Contains("ACAD_RENDER_SETTINGS") = True Then
                renderSettings = _
                    acTrans.GetObject(namedObjs.GetAt("ACAD_RENDER_SETTINGS"), _
                                      OpenMode.ForWrite)
            Else
                ' If it does not exist, create it and add it to the drawing
                namedObjs.UpgradeOpen()
                renderSettings = New DBDictionary
                namedObjs.SetAt("ACAD_RENDER_SETTINGS", renderSettings)
                acTrans.AddNewlyCreatedDBObject(renderSettings, True)
            End If

            ' Create the new render preset, based on
            ' the settings of the Medium render preset
            If renderSettings.Contains("MyPreset") = False Then
                Using renderSetting As New MentalRayRenderSettings
                    GetDefaultRenderPreset(renderSetting, "Medium")

                    renderSetting.Name = "MyPreset"
                    renderSetting.Description = "Custom new render preset"
                    renderSettings.SetAt("MyPreset", renderSetting)
                    acTrans.AddNewlyCreatedDBObject(renderSetting, True)
                End Using
            End If

            ' Set the new render preset current
            Application.UIBindings.RenderEngine.CurrentRenderPresetName = _
                "MyPreset"

        Catch es As Autodesk.AutoCAD.Runtime.Exception
            MsgBox(es.Message)
        Finally
            acTrans.Commit()
        End Try
    End Using
End Sub

Private Shared Sub GetDefaultRenderPreset( _
                  ByRef renderPreset As MentalRayRenderSettings, _
                  ByVal name As String)
    ' Set the values common to multiple default render presets
    renderPreset.BackFacesEnabled = False
    renderPreset.DiagnosticBackgroundEnabled = False
    renderPreset.DiagnosticBSPMode = _
        DiagnosticBSPMode.Depth
    renderPreset.DiagnosticGridMode = _
        New MentalRayRenderSettingsTraitsDiagnosticGridModeParameter( _
        DiagnosticGridMode.Object, 10.0)

    renderPreset.DiagnosticMode = _
        DiagnosticMode.Off
```

```
renderPreset.DiagnosticPhotonMode = _
    DiagnosticPhotonMode.Density
renderPreset.DisplayIndex = 0
renderPreset.EnergyMultiplier = 1.0
renderPreset.ExportMIEnabled = False
renderPreset.ExportMIFileName = ""
renderPreset.FGRayCount = 100

' FGSampleRadius cannot be set, it returns invalid input
renderPreset.FGSampleRadiusState = _
    New MentalRayRenderSettingsTraitsBoolParameter( _
        False, False, False)

renderPreset.FinalGatheringEnabled = False
renderPreset.FinalGatheringMode = _
    FinalGatheringMode.FinalGatherOff
renderPreset.GIPhotonsPerLight = 1000
renderPreset.GISampleCount = 500
renderPreset.GISampleRadius = 1.0
renderPreset.GISampleRadiusEnabled = False
renderPreset.GlobalIlluminationEnabled = False
renderPreset.LightLuminanceScale = 1500.0
renderPreset.MaterialsEnabled = True
renderPreset.MemoryLimit = 1048

renderPreset.PhotonTraceDepth = _
    New MentalRayRenderSettingsTraitsTraceParameter( _
        5, 5, 5)
renderPreset.PreviewImageFileName = ""
renderPreset.RayTraceDepth = _
    New MentalRayRenderSettingsTraitsTraceParameter( _
        3, 3, 3)
renderPreset.RayTracingEnabled = False
renderPreset.Sampling = _
    New MentalRayRenderSettingsTraitsIntegerRangeParameter( _
        -2, -1)
renderPreset.SamplingContrastColor = _
    New MentalRayRenderSettingsTraitsFloatParameter( _
        0.1, 0.1, 0.1, 0.1)
renderPreset.SamplingFilter = _
    New MentalRayRenderSettingsTraitsSamplingParameter( _
        Filter.Box, 1.0, 1.0)

renderPreset.ShadowMapsEnabled = False
renderPreset.ShadowMode = ShadowMode.Simple
renderPreset.ShadowSamplingMultiplier = _
    ShadowSamplingMultiplier.SamplingMultiplierZero
renderPreset.ShadowsEnabled = True
renderPreset.TextureSampling = False
renderPreset.TileOrder = TileOrder.Hilbert
renderPreset.TileSize = 32

Select Case name.ToUpper()
    ' Assigns the values to match the Draft render preset
    Case "DRAFT"
```

```
        renderPreset.Description = _
            "The lowest rendering quality which entails no raytracing, " & _
            "no texture filtering and force 2-sided is inactive."
        renderPreset.Name = "Draft"
    Case ("LOW")
        renderPreset.Description = _
            "Rendering quality is improved over Draft. " & _
            "Low anti-aliasing and a raytracing depth of 3 " & _
            "reflection/refraction are processed."
        renderPreset.Name = "Low"

        renderPreset.RayTracingEnabled = True

        renderPreset.Sampling = _
            New MentalRayRenderSettingsTraitsIntegerRangeParameter( _
                -1, 0)
        renderPreset.SamplingContrastColor = _
            New MentalRayRenderSettingsTraitsFloatParameter( _
                0.1, 0.1, 0.1, 0.1)
        renderPreset.SamplingFilter = _
            New MentalRayRenderSettingsTraitsSamplingParameter( _
                Filter.Triangle, 2.0, 2.0)

        renderPreset.ShadowSamplingMultiplier = _
            ShadowSamplingMultiplier.SamplingMultiplierOneFourth
    Case "MEDIUM"
        renderPreset.BackFacesEnabled = True
        renderPreset.Description = _
            "Rendering quality is improved over Low to include " & _
            "texture filtering and force 2-sided is active. " & _
            "Moderate anti-aliasing and a raytracing depth of " & _
            "5 reflections/refractions are processed."

        renderPreset.FGRayCount = 200
        renderPreset.FinalGatheringMode = _
            FinalGatheringMode.FinalGatherAuto
        renderPreset.GIPhotonsPerLight = 10000

        renderPreset.Name = "Medium"
        renderPreset.RayTraceDepth = _
            New MentalRayRenderSettingsTraitsTraceParameter( _
                5, 5, 5)
        renderPreset.RayTracingEnabled = True
        renderPreset.Sampling = _
            New MentalRayRenderSettingsTraitsIntegerRangeParameter( _
                0, 1)
        renderPreset.SamplingContrastColor = _
            New MentalRayRenderSettingsTraitsFloatParameter( _
                0.05, 0.05, 0.05, 0.05)
        renderPreset.SamplingFilter = _
            New MentalRayRenderSettingsTraitsSamplingParameter( _
                Filter.Gauss, 3.0, 3.0)

        renderPreset.ShadowSamplingMultiplier = _
            ShadowSamplingMultiplier.SamplingMultiplierOneHalf
```

```
        renderPreset.TextureSampling = True
    Case "HIGH"
        renderPreset.BackFacesEnabled = True
        renderPreset.Description = _
            "Rendering quality is improved over Medium. " & _
            "High anti-aliasing and a raytracing depth of 7 " & _
            "reflections/refractions are processed."

        renderPreset.FGRayCount = 500
        renderPreset.FinalGatheringMode = _
            FinalGatheringMode.FinalGatherAuto
        renderPreset.GIPhotonsPerLight = 10000

        renderPreset.Name = "High"
        renderPreset.RayTraceDepth = _
            New MentalRayRenderSettingsTraitsTraceParameter( _
                7, 7, 7)
        renderPreset.RayTracingEnabled = True
        renderPreset.Sampling = _
            New MentalRayRenderSettingsTraitsIntegerRangeParameter( _
                0, 2)
        renderPreset.SamplingContrastColor = _
            New MentalRayRenderSettingsTraitsFloatParameter( _
                0.05, 0.05, 0.05, 0.05)
        renderPreset.SamplingFilter = _
            New MentalRayRenderSettingsTraitsSamplingParameter( _
                Filter.Mitchell, 4.0, 4.0)

        renderPreset.ShadowSamplingMultiplier = _
            ShadowSamplingMultiplier.SamplingMultiplierOne
        renderPreset.TextureSampling = True
    Case "PRESENTATION"
        renderPreset.BackFacesEnabled = True
        renderPreset.Description = _
            "Rendering quality is improved over High. " & _
            "Very high anti-aliasing and a raytracing depth of 9 " & _
            "reflections/refractions are processed."

        renderPreset.FGRayCount = 1000
        renderPreset.FinalGatheringMode = _
            FinalGatheringMode.FinalGatherAuto
        renderPreset.GIPhotonsPerLight = 10000

        renderPreset.Name = "Presentation"
        renderPreset.RayTraceDepth = _
            New MentalRayRenderSettingsTraitsTraceParameter( _
                9, 9, 9)
        renderPreset.RayTracingEnabled = True
        renderPreset.Sampling = _
            New MentalRayRenderSettingsTraitsIntegerRangeParameter( _
                1, 2)
        renderPreset.SamplingContrastColor = _
            New MentalRayRenderSettingsTraitsFloatParameter( _
                0.05, 0.05, 0.05, 0.05)
        renderPreset.SamplingFilter = _
```

```
                New MentalRayRenderSettingsTraitsSamplingParameter( _
                    Filter.Lanczos, 4.0, 4.0)

            renderPreset.ShadowSamplingMultiplier = _
                ShadowSamplingMultiplier.SamplingMultiplierOne
            renderPreset.TextureSampling = True
    End Select
End Sub
```

5 Plot Configurations and Plot Styles

Plot configurations are used to define the properties of an output device that represents a physical or virtual device. A plot configuration is saved in a PC3 file and can be created using the Plotter Manager (PLOTTERMANAGER command) or the SaveToPC3 method of the **PlotConfig** class. While a PC3 file can be created with the .NET API, I do not cover creating PC3 files in this session but it is typically more practical to create them using the Plotter Manager. When automating plotting tasks though, you will need to know which PC3 files are available and assign one of the available files to the **PlotSettings**, **Layout**, **PlotConfig** and **PlotInfo** objects used for page setups, layouts, and the plotting or publishing of a layout.

Plot styles control the appearance of objects on screen and during output, and they come in two different types: color-dependent and named styles. Color-dependent plot styles allow you to control the appearance of linework based on an object's assigned color, where as named plot styles use names to define a set of properties used to control how an object might appear instead of using an object's color. Color-dependent is by far the most popular plot style, and is also the oldest way of controlling the appearance of objects during output. Named plot styles are a bit harder to setup sine they are not supported for complex objects such as dimensions and tables.

The way a drawing is created determines which type of plot style it uses. Your custom programs need to account for both types and not assume that all drawings use color-dependent plot styles. The **PlotStyleMode** property of the Database class can be used to determine which type of plot style is currently being used in a drawing.

When determining which plot configurations and plot styles are available to your custom programs and AutoCAD, you will need to work with the **PlotSettingsValidator** class.

Class(es) you will need to work with:

- **PlotSettingsValidator** - Autodesk.AutoCAD.DatabaseServices.PlotSettingsValidator
- **PlotSettings** - Autodesk.AutoCAD.DatabaseServices.PlotSettings

```
' Lists the available plotters (plot configuration [PC3] files)
<CommandMethod("PlotterList")> _
Public Shared Sub PlotterList()
    ' Get the current document and database, and start a transaction
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

    acDoc.Editor.WriteMessage(vbLf & "Plot devices: ")
```

```
    For Each plotDevice As String In PlotSettingsValidator.Current.GetPlotDeviceList()
        ' Output the names of the available plotter devices
        acDoc.Editor.WriteMessage(vbLf & "  " & plotDevice)
    Next
End Sub

' Lists the available media sizes for a specified plot configuration (PC3) file
<CommandMethod("PlotterMediaList")> _
Public Shared Sub PlotterMediaList()
    ' Get the current document and database, and start a transaction
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

    Using plSet As PlotSettings = New PlotSettings(True)
        Dim acPlSetVdr As PlotSettingsValidator = PlotSettingsValidator.Current

        ' Set the Plotter and page size
        acPlSetVdr.SetPlotConfigurationName(plSet, "DWF6 ePlot.pc3", _
                                            "ANSI_A_(8.50_x_11.00_Inches)")

        acDoc.Editor.WriteMessage(vbLf & "Available media sizes: ")

        For Each mediaName As String In acPlSetVdr.GetCanonicalMediaNameList(plSet)
            ' Output the names of the available media for the specified device
            acDoc.Editor.WriteMessage(vbLf & "  " & mediaName)
        Next
    End Using
End Sub

' Lists the available plot styles
<CommandMethod("PlotStyleList")> _
Public Shared Sub PlotStyleList()
    ' Get the current document and database, and start a transaction
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument

    acDoc.Editor.WriteMessage(vbLf & "Plot styles: ")

    For Each plotStyle As String In PlotSettingsValidator.Current.GetPlotStyleSheetList()
        ' Output the names of the available plot styles
        acDoc.Editor.WriteMessage(vbLf & "  " & plotStyle)
    Next
End Sub
```

6 Page Setups

Page setups are named objects that store plot settings used to control the display of objects on a layout and during the output of a drawing. The main reason to use page setups is to enforce consistency across the layouts in multiple drawings, and to make it easy to publish a single drawing at full size or half-size for review.

The **PlotSettings** class is used to represent a page setup, and is the base class for the **Layout** class as it inherits many of its settings from the **PlotSettings** class. This allows a layout to have

its own plot settings without having to create a named page setup, and to apply plot settings as overrides during the plotting or publishing a layout.

When you create or modify a **PlotSettings** object, you will also need to work with the Plot Settings Validator (**PlotSettingsValidator** class) when you set certain properties such as the plot configuration name, plot style, and several others. Page setups that are defined within a drawing are stored in the Plot Settings dictionary.

Class(es) you will need to work with:

- **DBDictionary** - Autodesk.AutoCAD.DatabaseServices.DBDictionary
- **DBDictionaryEntry** - Autodesk.AutoCAD.DatabaseServices -> DBDictionaryEntry Structure
- **PlotSettingsValidator** - Autodesk.AutoCAD.DatabaseServices.PlotSettingsValidator
- **PlotSettings** - Autodesk.AutoCAD.DatabaseServices.PlotSettings
- **LayoutManager** - Autodesk.AutoCAD.DatabaseServices.LayoutManager
- **Layout** - Autodesk.AutoCAD.DatabaseServices.Layout

```
' Lists the available page setups
<CommandMethod("PageSetupList")> _
Public Shared Sub PageSetupList()
    ' Get the current document and database
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
    Dim acCurDb As Database = acDoc.Database

    ' Start a transaction
    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
        Dim plSettings As DBDictionary = _
            acTrans.GetObject(acCurDb.PlotSettingsDictionaryId, OpenMode.ForRead)

        acDoc.Editor.WriteMessage(vbLf & "Page Setups: ")

        ' List each named page setup
        For Each item As DBDictionaryEntry In plSettings
            acDoc.Editor.WriteMessage(vbLf & "  " & item.Key)
        Next

        ' Abort the changes to the database
        acTrans.Abort()
    End Using
End Sub

' Creates a new page setup or edits the page set if it exists
<CommandMethod("PageSetupCreateEdit")> _
Public Shared Sub PageSetupCreate()
    ' Get the current document and database, and start a transaction
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
    Dim acCurDb As Database = acDoc.Database

    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

        Dim plSets As DBDictionary = _
```

```
    acTrans.GetObject(acCurDb.PlotSettingsDictionaryId, OpenMode.ForRead)
Dim vStyles As DBDictionary = _
    acTrans.GetObject(acCurDb.VisualStyleDictionaryId, OpenMode.ForRead)

Dim acPlSet As PlotSettings
Dim createNew As Boolean = False

' Reference the Layout Manager
Dim acLayoutMgr As LayoutManager = LayoutManager.Current

' Get the current layout and output its name in the Command Line window
Dim acLayout As Layout = _
    acTrans.GetObject(acLayoutMgr.GetLayoutId(acLayoutMgr.CurrentLayout), _
                      OpenMode.ForRead)

' Check to see if the page setup exists
If plSets.Contains("MyPageSetup") = False Then
    createNew = True

    ' Create a new PlotSettings object:
    '    True - model space, False - named layout
    acPlSet = New PlotSettings(acLayout.ModelType)
    acPlSet.CopyFrom(acLayout)

    acPlSet.PlotSettingsName = "MyPageSetup"
    acPlSet.AddToPlotSettingsDictionary(acCurDb)
    acTrans.AddNewlyCreatedDBObject(acPlSet, True)
Else
    acPlSet = plSets.GetAt("MyPageSetup").GetObject(OpenMode.ForWrite)
End If

Dim acPlSetVdr As PlotSettingsValidator = PlotSettingsValidator.Current

' Update the PlotSettings object
Try
    ' Set the Plotter and page size
    acPlSetVdr.SetPlotConfigurationName(acPlSet, _
                                        "DWF6 ePlot.pc3", _
                                        "ANSI_B_(17.00_x_11.00_Inches)")

    ' Set to plot to the current display
    If acLayout.ModelType = False Then
        acPlSetVdr.SetPlotType(acPlSet, _
                               DatabaseServices.PlotType.Layout)
    Else
        acPlSetVdr.SetPlotType(acPlSet, _
                               DatabaseServices.PlotType.Extents)

        acPlSetVdr.SetPlotCentered(acPlSet, True)
    End If

    ' Use SetPlotWindowArea with PlotType.Window
    'acPlSetVdr.SetPlotWindowArea(plSet, _
    '                             New Extents2d(New Point2d(0.0, 0.0), _
    '                             New Point2d(9.0, 12.0)))
```

```
        ' Use SetPlotViewName with PlotType.View
        'acPlSetVdr.SetPlotViewName(plSet, "MyView")

        ' Set the plot offset
        acPlSetVdr.SetPlotOrigin(acPlSet, _
                                 New Point2d(0, 0))

        ' Set the plot scale
        acPlSetVdr.SetUseStandardScale(acPlSet, True)
        acPlSetVdr.SetStdScaleType(acPlSet, StdScaleType.ScaleToFit)
        acPlSetVdr.SetPlotPaperUnits(acPlSet, PlotPaperUnit.Inches)
        acPlSet.ScaleLineweights = True

        ' Specify if plot styles should be displayed on the layout
        acPlSet.ShowPlotStyles = True

        ' Rebuild plotter, plot style, and canonical media lists
        ' (must be called before setting the plot style)
        acPlSetVdr.RefreshLists(acPlSet)

        ' Specify the shaded viewport options
        acPlSet.ShadePlot = PlotSettingsShadePlotType.AsDisplayed

        acPlSet.ShadePlotResLevel = ShadePlotResLevel.Normal

        ' Specify the plot options
        acPlSet.PrintLineweights = True
        acPlSet.PlotTransparency = False
        acPlSet.PlotPlotStyles = True
        acPlSet.DrawViewportsFirst = True

        ' Use only on named layouts - Hide paperspace objects option
        ' plSet.PlotHidden = True

        ' Specify the plot orientation
        acPlSetVdr.SetPlotRotation(acPlSet, PlotRotation.Degrees000)

        ' Set the plot style
        If acCurDb.PlotStyleMode = True Then
            acPlSetVdr.SetCurrentStyleSheet(acPlSet, "acad.ctb")
        Else
            acPlSetVdr.SetCurrentStyleSheet(acPlSet, "acad.stb")
        End If

        ' Zoom to show the whole paper
        acPlSetVdr.SetZoomToPaperOnUpdate(acPlSet, True)
    Catch es As Autodesk.AutoCAD.Runtime.Exception
        MsgBox(es.Message)
    End Try

    ' Save the changes made
    acTrans.Commit()

    If createNew = True Then
```

```
            acPlSet.Dispose()
        End If
    End Using
End Sub

' Assigns a page setup to a layout
<CommandMethod("PageSetupAssignToLayout")> _
Public Shared Sub PageSetupAssignToLayout()
    ' Get the current document and database, and start a transaction
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
    Dim acCurDb As Database = acDoc.Database

    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
        ' Reference the Layout Manager
        Dim acLayoutMgr As LayoutManager = LayoutManager.Current

        ' Get the current layout and output its name in the Command Line window
        Dim acLayout As Layout = _
            acTrans.GetObject(acLayoutMgr.GetLayoutId(acLayoutMgr.CurrentLayout), _
                              OpenMode.ForRead)

        Dim plSets As DBDictionary = _
            acTrans.GetObject(acCurDb.PlotSettingsDictionaryId, OpenMode.ForRead)

        ' Check to see if the page setup exists
        If plSets.Contains("MyPageSetup") = True Then
            Dim plSet As PlotSettings = _
                plSets.GetAt("MyPageSetup").GetObject(OpenMode.ForRead)

            ' Update the layout
            acLayout.UpgradeOpen()
            acLayout.CopyFrom(plSet)

            ' Save the new objects to the database
            acTrans.Commit()
        Else
            ' Ignore the changes made
            acTrans.Abort()
        End If
    End Using

    ' Update the display
    acDoc.Editor.Regen()
End Sub

' Changes the plot settings for a layout directly
<CommandMethod("LayoutChangePlotSettings")> _
Public Shared Sub LayoutChangePlotSettings()
    ' Get the current document and database, and start a transaction
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
    Dim acCurDb As Database = acDoc.Database

    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
        ' Reference the Layout Manager
        Dim acLayoutMgr As LayoutManager = LayoutManager.Current
```

```
        ' Get the current layout and output its name in the Command Line window
        Dim acLayout As Layout = _
            acTrans.GetObject(acLayoutMgr.GetLayoutId(acLayoutMgr.CurrentLayout), _
                              OpenMode.ForRead)

        ' Output the name of the current layout and its device
        acDoc.Editor.WriteMessage(vbLf & "Current layout: " & _
                                  acLayout.LayoutName)

        acDoc.Editor.WriteMessage(vbLf & "Current device name: " & _
                                  acLayout.PlotConfigurationName)

        ' Get a copy of the PlotSettings from the layout
        Dim acPlSet As PlotSettings = New PlotSettings(acLayout.ModelType)
        acPlSet.CopyFrom(acLayout)

        ' Update the PlotConfigurationName property of the PlotSettings object
        Dim acPlSetVdr As PlotSettingsValidator = PlotSettingsValidator.Current
        acPlSetVdr.SetPlotConfigurationName(acPlSet, "DWG To PDF.pc3", _
                                            "ANSI_B_(11.00_x_17.00_Inches)")

        ' Zoom to show the whole paper
        acPlSetVdr.SetZoomToPaperOnUpdate(acPlSet, True)

        ' Update the layout
        acLayout.UpgradeOpen()
        acLayout.CopyFrom(acPlSet)

        ' Output the name of the new device assigned to the layout
        acDoc.Editor.WriteMessage(vbLf & "New device name: " & _
                                  acLayout.PlotConfigurationName)

        ' Save the new objects to the database
        acTrans.Commit()
    End Using

    ' Update the display
    acDoc.Editor.Regen()
End Sub
```

7 Plot a Layout

It is not uncommon that a drawing will need to be shared at some point in type during its life with someone that does not have AutoCAD or should not have access to the actual model. Plotting or publishing are often used to share a drawing in a hardcopy or electronic format. Plotting a layout from a drawing is handled with many of the same classes that have been previously mentioned in this session: **Layout**, **Plotsettings**, and **PlotSettingsValidator**.

You will also be working with additional classes that are used to collect information about the layouts you are going to plot and access the Plot Engine. The Plot Engine (**PlotEngine** object) is what you will be working with when you want to plot or preview a layout from a drawing. You

use the ***CreatePublishEngine*** method when you want to plot a layout or ***CreatePreviwEngine*** when you want to preview a layout.

The **PlotInfo** utility class is used to define the plot settings that are passed to the Plot Engine when plotting a layout. When setting up the **PlotInfo** object, you use the **PlotInfoValidator** object to ensure the object is defined properly; similar to the **PlotSettingsValidator** object used to validate a **PlotSettings** object.

The **PlotEngine** object is affected by the BACKGROUNDPLOT system variable, which determines if it operates in the foreground or background like the PLOT command. You use the **PlotFactory** class to obtain a reference to the instance of the Plot Engine for the application and also check the current status of the Plot Engine; see if a layout is already currently being plotted which is important if a plot is happening in the background.

Class(es) you will need to work with:

- **PlotSettingsValidator** - Autodesk.AutoCAD.DatabaseServices.PlotSettingsValidator
- **PlotSettings** - Autodesk.AutoCAD.DatabaseServices.PlotSettings
- **PlotInfoValidator** - Autodesk.AutoCAD.DatabaseServices.PlotInfoValidator
- **PlotInfo** - Autodesk.AutoCAD.DatabaseServices.PlotInfo
- **LayoutManager** - Autodesk.AutoCAD.DatabaseServices.LayoutManager
- **Layout** - Autodesk.AutoCAD.DatabaseServices.Layout
- **PlotEngine** - Autodesk.AutoCAD.PlottingServices.PlotEngine
- **PlotProgressDialog** - Autodesk.AutoCAD.PlottingServices.PlotProgressDialog

```
' Plots the current layout to a DWF file
<CommandMethod("PlotLayout")> _
Public Shared Sub PlotLayout()
    ' Get the current document and database, and start a transaction
    Dim acDoc As Document = Application.DocumentManager.MdiActiveDocument
    Dim acCurDb As Database = acDoc.Database

    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()
        ' Reference the Layout Manager
        Dim acLayoutMgr As LayoutManager = LayoutManager.Current

        ' Get the current layout and output its name in the Command Line window
        Dim acLayout As Layout = _
            acTrans.GetObject(acLayoutMgr.GetLayoutId(acLayoutMgr.CurrentLayout), _
                              OpenMode.ForRead)

        ' Get the PlotInfo from the layout
        Using acPlInfo As PlotInfo = New PlotInfo()
            acPlInfo.Layout = acLayout.ObjectId

            ' Get a copy of the PlotSettings from the layout
            Using acPlSet As PlotSettings = New PlotSettings(acLayout.ModelType)
                acPlSet.CopyFrom(acLayout)

                ' Update the PlotSettings object
```

```
Dim acPlSetVdr As PlotSettingsValidator = _
    PlotSettingsValidator.Current

' Set the plot type
acPlSetVdr.SetPlotType(acPlSet, _
                       DatabaseServices.PlotType.Extents)

' Set the plot scale
acPlSetVdr.SetUseStandardScale(acPlSet, True)
acPlSetVdr.SetStdScaleType(acPlSet, StdScaleType.ScaleToFit)

' Center the plot
acPlSetVdr.SetPlotCentered(acPlSet, True)

' Set the plot device to use
acPlSetVdr.SetPlotConfigurationName(acPlSet, "DWF6 ePlot.pc3", _
                                    "ANSI_A_(8.50_x_11.00_Inches)")

' Set the plot info as an override since it will
' not be saved back to the layout
acPlInfo.OverrideSettings = acPlSet

' Validate the plot info
Using acPlInfoVdr As PlotInfoValidator = New PlotInfoValidator()
    acPlInfoVdr.MediaMatchingPolicy = MatchingPolicy.MatchEnabled
    acPlInfoVdr.Validate(acPlInfo)

    ' Check to see if a plot is already in progress
    If PlotFactory.ProcessPlotState = _
        ProcessPlotState.NotPlotting Then

        Using acPlEng As PlotEngine = _
            PlotFactory.CreatePublishEngine()

            ' Track the plot progress with a Progress dialog
            Using acPlProgDlg As PlotProgressDialog = _
                New PlotProgressDialog(False, 1, True)

                Using (acPlProgDlg)
                    ' Define the status messages to display
                    ' when plotting starts
                    acPlProgDlg.PlotMsgString( _
                        PlotMessageIndex.DialogTitle) = "Plot Progress"
                    acPlProgDlg.PlotMsgString( _
                        PlotMessageIndex.CancelJobButtonMessage) = _
                                            "Cancel Job"
                    acPlProgDlg.PlotMsgString( _
                        PlotMessageIndex.CancelSheetButtonMessage) = _
                                            "Cancel Sheet"
                    acPlProgDlg.PlotMsgString( _
                        PlotMessageIndex.SheetSetProgressCaption) = _
                                            "Sheet Set Progress"
                    acPlProgDlg.PlotMsgString( _
                        PlotMessageIndex.SheetProgressCaption) = _
                                            "Sheet Progress"
```

```
' Set the plot progress range
acPlProgDlg.LowerPlotProgressRange = 0
acPlProgDlg.UpperPlotProgressRange = 100
acPlProgDlg.PlotProgressPos = 0

' Display the Progress dialog
acPlProgDlg.OnBeginPlot()
acPlProgDlg.IsVisible = True

' Start to plot the layout
acPlEng.BeginPlot(acPlProgDlg, Nothing)

' Define the plot output
acPlEng.BeginDocument(acPlInfo, _
                      acDoc.Name, _
                      Nothing, _
                      1, _
                      True, _
                      "c:\myplot")

' Display information about the current plot
acPlProgDlg.PlotMsgString( _
    PlotMessageIndex.Status) = _
    "Plotting: " & acDoc.Name & _
    " - " & acLayout.LayoutName

' Set the sheet progress range
acPlProgDlg.OnBeginSheet()
acPlProgDlg.LowerSheetProgressRange = 0
acPlProgDlg.UpperSheetProgressRange = 100
acPlProgDlg.SheetProgressPos = 0

' Plot the first sheet/layout
Using acPlPageInfo As PlotPageInfo = _
    New PlotPageInfo()
    acPlEng.BeginPage(acPlPageInfo, _
                      acPlInfo, _
                      True, _
                      Nothing)
End Using

acPlEng.BeginGenerateGraphics(Nothing)
acPlEng.EndGenerateGraphics(Nothing)

' Finish plotting the sheet/layout
acPlEng.EndPage(Nothing)
acPlProgDlg.SheetProgressPos = 100
acPlProgDlg.OnEndSheet()

' Finish plotting the document
acPlEng.EndDocument(Nothing)

' Finish the plot
acPlProgDlg.PlotProgressPos = 100
```

```
                                    acPlProgDlg.OnEndPlot()
                                    acPlEng.EndPlot(Nothing)
                                End Using
                            End Using
                        End Using
                    End If
                End Using
            End Using
        End Using
    End Using
End Sub
```

8 Publish Multiple Layouts or a Sheet Set

Publishing multiple layouts is different from plotting a layout, but you will still be working with many of the same classes though. When publishing layouts, you need to make sure all the layouts that you want to output have been properly setup or you can override the plot settings assigned to the layout.

To get started, you need to gather information about each layout you want to publish. The information you gather must be placed in a **DsdEntryCollection** object. A **DsdEntryCollection** object contains one or more **DsdEntry** object, and each **DsdEntry** object represents a layout you want to publish. A **DsdEntry** object contains the name of the layout and the drawing it can be found in, along with which page setup override should be used during publishing. If no override is provided, then it uses the plot settings saved with the layout.

After a **DsdEntryCollection** object has been populated, you then create a **DsdData** object which defines the settings that should be used to publish the specified layouts. Once the **DsdData** object is defined, you can save the settings to a DSD file that can be used by the PUBLISH command or the **Publisher** object.

The **Publisher** object is used to generate the output based on the settings and files listed in the DSD file. Like the Publish dialog box, you can publish a layout using the plot device and settings defined with each layout or assigned via an override page setup in the **DsdEntry** object using the **PublishDsd** method, or override the device assigned to a layout using the **PublishExecute** method. The **PublishExecute** method is affected by the BACKGROUNDPLOT system variable like the PUBLISH command, while the **PublishDsd** method is not.

Class(es) you will need to work with:

- **DsdEntryCollection** - Autodesk.AutoCAD.PlottingServices.DsdEntryCollection
- **DsdEntry** - Autodesk.AutoCAD.PlottingServices.DsdEntry
- **DsdData** - Autodesk.AutoCAD.PlottingServices.DsdData
- **PlotProgressDialog** - Autodesk.AutoCAD.PlottingServices.PlotProgressDialog
- **Publisher** - Autodesk.AutoCAD.Publishing.Publisher
- **PlotConfig** - Autodesk.AutoCAD.PlottingServices.PlotConfig
- **PlotConfigInfo** - Autodesk.AutoCAD.PlottingServices.PlotConfigInfo

- **PlotConfigManager** - Autodesk.AutoCAD.PlottingServices.PlotConfigManager

```
' Publishes all the drawings in a sheet set to a multi-sheet DWF
<CommandMethod("PublishSheetSet")> _
Public Sub PublishSheetSet()
    ' DSD filename
    Dim dsdFilename As String = Environment.GetFolderPath( _
                                Environment.SpecialFolder.MyDocuments) & _
                                "\batchdrawings.dsd"

    ' Delete the DSD file if it exists
    If File.Exists(dsdFilename) = True Then File.Delete(dsdFilename)

    ' Get a reference to the Sheet Set Manager object
    Dim sheetSetManager As IAcSmSheetSetMgr = New AcSmSheetSetMgr

    ' Open a Sheet Set file
    Dim sheetSetDatabase As AcSmDatabase = _
        sheetSetManager.OpenDatabase("C:\Program Files\Autodesk\AutoCAD 2013\Sample\" & _
                                     "Sheet Sets\Architectural\IRD Addition.dst", False)

    Dim sheetSet As AcSmSheetSet = sheetSetDatabase.GetSheetSet()

    Dim enumerator As IAcSmEnumPersist
    Dim itemSheetSet As IAcSmPersist

    Using dsdDwgFiles As New DsdEntryCollection()

        ' Get the enumerator for the objects in the sheet set
        enumerator = sheetSetDatabase.GetEnumerator()
        itemSheetSet = enumerator.Next()

        ' Step through the objects in the sheet set
        Do While Not itemSheetSet Is Nothing
            ' Increment the counter of the object is a sheet
            If itemSheetSet.GetTypeName() = "AcSmSheet" Then

                Dim sheet As AcSmSheet = itemSheetSet

                ' Add drawing file
                If File.Exists(sheet.GetLayout().GetFileName()) = True Then
                    Using dsdDwgFile As DsdEntry = New DsdEntry()

                        ' Set the file name and layout
                        dsdDwgFile.DwgName = sheet.GetLayout().GetFileName()
                        dsdDwgFile.Layout = sheet.GetLayout().GetName()
                        dsdDwgFile.Title = sheet.GetName()

                        ' Set the page setup override
                        dsdDwgFile.Nps = ""
                        dsdDwgFile.NpsSourceDwg = ""

                        dsdDwgFiles.Add(dsdDwgFile)
                    End Using
                End If
```

```
        End If

        ' Get next object
        itemSheetSet = enumerator.Next()
    Loop

    If dsdDwgFiles.Count > 0 Then
        ' Set the properties for the DSD file and then write it out
        Using dsdFileData As DsdData = New DsdData()

            ' Set the version of the DWF 6 format
            dsdFileData.MajorVersion = 1
            dsdFileData.MinorVersion = 1

            ' Set the target information for publishing
            dsdFileData.DestinationName = Environment.GetFolderPath( _
                Environment.SpecialFolder.MyDocuments) & "\" & _
                sheetSetDatabase.GetSheetSet.GetName() & ".dwf"
            dsdFileData.ProjectPath = Environment.GetFolderPath( _
                Environment.SpecialFolder.MyDocuments)

            ' Set the type of output that should be generated
            dsdFileData.SheetType = SheetType.MultiDwf

            ' Set the drawings that should be added to the publication
            dsdFileData.SetDsdEntryCollection(dsdDwgFiles)

            ' Sheet set properties
            dsdFileData.IsSheetSet = True
            dsdFileData.SheetSetName = sheetSetDatabase.GetName()
            dsdFileData.SelectionSetName = ""
            dsdFileData.IsHomogeneous = False

            ' Set the general publishing properties
             dsdFileData.CategoryName = ""
             dsdFileData.NoOfCopies = 1
             dsdFileData.LogFilePath = Environment.GetFolderPath( _
                Environment.SpecialFolder.MyDocuments) & "\myBatch.txt"
             dsdFileData.Password = ""
             dsdFileData.PlotStampOn = False
             dsdFileData.PromptForDwfName = False

             ' Set the DWF 3D options
             dsdFileData.Dwf3dOptions.GroupByXrefHierarchy = False
             dsdFileData.Dwf3dOptions.PublishWithMaterials = False

             ' Create the DSD file
             dsdFileData.WriteDsd(Environment.GetFolderPath( _
                Environment.SpecialFolder.MyDocuments) & _
                "\batchdrawings.dsd")

             ' Track the progress with a Progress dialog
             Using acPlProgDlg As PlotProgressDialog = _
                New PlotProgressDialog(False, dsdDwgFiles.Count, False)
```

```
                    ' Define the status messages to display
                    ' when plotting starts
                    acPlProgDlg.PlotMsgString( _
                        PlotMessageIndex.DialogTitle) = "Plot Progress"
                    acPlProgDlg.PlotMsgString( _
                        PlotMessageIndex.CancelJobButtonMessage) = _
                                        "Cancel Job"
                    acPlProgDlg.PlotMsgString( _
                        PlotMessageIndex.CancelSheetButtonMessage) = _
                                        "Cancel Sheet"
                    acPlProgDlg.PlotMsgString( _
                        PlotMessageIndex.SheetSetProgressCaption) = _
                                        "Sheet Set Progress"
                    acPlProgDlg.PlotMsgString( _
                        PlotMessageIndex.SheetProgressCaption) = _
                                        "Sheet Progress"

                    ' Set the plot progress range
                    acPlProgDlg.LowerPlotProgressRange = 0
                    acPlProgDlg.UpperPlotProgressRange = 100
                    acPlProgDlg.PlotProgressPos = 0

                    acPlProgDlg.LowerSheetProgressRange = 0
                    acPlProgDlg.UpperSheetProgressRange = 100

                    acPlProgDlg.IsVisible = True

                    Try
                        ' Publish the drawing using the DSD in the foreground
                        Application.Publisher.PublishDsd(Environment.GetFolderPath( _
                            Environment.SpecialFolder.MyDocuments) & _
                                    "\batchdrawings.dsd", acPlProgDlg)

                    Catch es As Autodesk.AutoCAD.Runtime.Exception
                        MsgBox(es.Message)
                    Finally
                        acPlProgDlg.Destroy()
                        acPlProgDlg.Dispose()
                    End Try
                End Using
            End Using
        End If
    End Using

    ' Close the sheet set
    sheetSetManager.Close(sheetSetDatabase)
End Sub

' Publishes a few drawings in the background to PDF file
<CommandMethod("PublishPDF")> _
Public Shared Sub PublishPDF()
    Using dsdDwgFiles As New DsdEntryCollection

        ' Add first drawing file
        Using dsdDwgFile1 As New DsdEntry
```

```
    ' Set the file name and layout
    dsdDwgFile1.DwgName = "C:\Program Files\Autodesk\AutoCAD 2013\Sample\" & _
        "Sheet Sets\Architectural\A-01.dwg"
    dsdDwgFile1.Layout = "MAIN AND SECOND FLOOR PLAN"
    dsdDwgFile1.Title = "A-01 MAIN AND SECOND FLOOR PLAN"

    ' Set the page setup override
    dsdDwgFile1.Nps = ""
    dsdDwgFile1.NpsSourceDwg = ""

    dsdDwgFiles.Add(dsdDwgFile1)
End Using

' Add second drawing file
Using dsdDwgFile2 As New DsdEntry

    ' Set the file name and layout
    dsdDwgFile2.DwgName = "C:\Program Files\Autodesk\AutoCAD 2013\Sample\" & _
        "Sheet Sets\Architectural\A-02.dwg"
    dsdDwgFile2.Layout = "ELEVATIONS"
    dsdDwgFile2.Title = "A-02 ELEVATIONS"

    ' Set the page setup override
    dsdDwgFile2.Nps = ""
    dsdDwgFile2.NpsSourceDwg = ""

    dsdDwgFiles.Add(dsdDwgFile2)
End Using

' Set the properties for the DSD file and then write it out
Using dsdFileData As New DsdData

    ' Set the target information for publishing
    dsdFileData.DestinationName = Environment.GetFolderPath( _
        Environment.SpecialFolder.MyDocuments) & "\MyPublish2.pdf"
    dsdFileData.ProjectPath = Environment.GetFolderPath( _
        Environment.SpecialFolder.MyDocuments) & "\"
    dsdFileData.SheetType = SheetType.MultiPdf

    ' Set the drawings that should be added to the publication
    dsdFileData.SetDsdEntryCollection(dsdDwgFiles)

    ' Set the general publishing properties
    dsdFileData.LogFilePath = Environment.GetFolderPath( _
        Environment.SpecialFolder.MyDocuments) & "\myBatch.txt"

    ' Create the DSD file
    dsdFileData.WriteDsd(Environment.GetFolderPath( _
                         Environment.SpecialFolder.MyDocuments) & _
                         "\batchdrawings2.dsd")

    Try
        ' Publish the specified drawing files in the DSD file, and
        ' honor the behavior of the BACKGROUNDPLOT system variable
```

```
                Using dsdDataFile As New DsdData
                    dsdDataFile.ReadDsd(Environment.GetFolderPath( _
                                        Environment.SpecialFolder.MyDocuments) & _
                                        "\batchdrawings2.dsd")

                    ' Get the DWG to PDF.pc3 and use it as a
                    ' device override for all the layouts
                    Dim acPlCfg As PlotConfig = _
                        PlotConfigManager.SetCurrentConfig("DWG to PDF.PC3")

                    Application.Publisher.PublishExecute(dsdDataFile, acPlCfg)
                End Using

            Catch es As Autodesk.AutoCAD.Runtime.Exception
                MsgBox(es.Message)
            End Try
        End Using
    End Using
End Sub
```

9 Where to Get More Information

When you are first start using a feature, you will have questions and where to find the answers you need might not be clear. The following is a list of resources that you can use to find additional information:

- **Help System** – The .NET Developer's and Reference Guides in the AutoCAD online Help system and ObjectARX SDKs contain information about working with named layouts, viewports, plot settings, visual styles, and the plot engine. To access the AutoCAD online Help system, go to: *http://docs.autodesk.com/ACD/2013/ENU/*. The ObjectARX SDK can be downloaded by going to *http://www.objectarx.com*.
- **Discussion Forums** – There are a number of peer-to-peer discussion forums on the Internet that are moderated by a range of users and Autodesk employees.
  - Autodesk Discussion Groups (*http://forums.autodesk.com*) – Click AutoCAD and then click the .NET link to access the subgroup. These forums are for users, but you can also find Autodesk employees taking time to reply to postings. The forums are moderated by Autodesk.
  - AUGI Forums (*http://www.augi.com*) – Peer-to-peer networking where you can ask questions about virtually anything in AutoCAD and get a response from fellow users.
  - The Swamp (*http://www.theswamp.org*) – Peer-to-peer networking site that is focused on AutoCAD customization and program. A great resource for advanced questions that you might have about the AutoCAD APIs. You can find fellow developers and some Autodesk employees lurking on the site.
- **Industry Events and Classes** – Industry events such as AUGI CAD Camp 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 a number of AutoCAD related blogs that cover programming with .NET and other AutoCAD related APIs. You can also use your favorite search engine, such as Google or Bing and search on the API you are interested in.
  - Through the Interface (*http://blogs.autodesk.com/through-the-interface*) - Kean Walmsley
  - AutoCAD DevBlog (*http://adndevblog.typepad.com/autocad/*) - The Autodesk Developer Network Team