AUTODESK **UNIVERSITY** 2015

IT10489

Harnessing the Power of the AutoCAD COM APIs

Lee Ambrosius
Autodesk, Inc.

Learning Objectives

- Learn how to link database records to objects in a drawing
- Learn how to create and modify sheets and sheet sets
- Learn how to check for and fix issues related to drawing standards
- Learn how to create a basic transmittal package from a drawing

Description

AutoCAD software's COM automation API used for Microsoft VBA isn't the only COM API that is exposed for use with AutoCAD software. There are 4 lesser-known COM APIs that have been lost in time; they were designed to automate some of the CAD management-related tasks in the AutoCAD drawing environment. The COM APIs that we will cover during this session enable you to create a transmittal package, work with sheet sets, define custom-CAD-standards plug-ins, and link objects to database records. This session will provide an overview of each API and will give you lots of sample code to take back for experimentation.

Your AU Experts

Lee Ambrosius is a Principal Learning Content Developer at Autodesk, Inc., for the AutoCAD software and AutoCAD LT software products. He works primarily on the CAD administration, customization, and developer documentation. Lee has also worked on the user documentation for AutoCAD on Windows and Mac. He has presented on a wide range of topics at Autodesk University over the past 10 years, from general AutoCAD customization to ObjectARX technology. Lee has authored a number of AutoCAD-related books, with his most recent projects being AutoCAD Platform Customization: User Interface, AutoLISP, VBA, and Beyond and AutoCAD 2015 and AutoCAD LT 2015 Bible. When he's not writing, you can find him roaming various community forums, posting articles on his blog, or tweeting AutoCAD-related information.

Twitter: *http://twitter.com/leeambrosius*

Email: *lee.ambrosius@autodesk.com*

Blog: *http://hyperpics.blogs.com*

Contents

1 Introduction

ActiveX is a programming framework that allows a developer to create modular libraries and form controls that can be used by multiple programs. ActiveX was first introduced by Microsoft in 1996 as a way to simplify application creation and build rich websites. In 2015, Microsoft officially ended ActiveX support with the introduction of Microsoft Edge; the web browser that replaced Microsoft Internet Explorer.

While ActiveX is now a deprecated technology, it is still used by third-party developers and applications that support Visual Basic for Applications (VBA) such as the Microsoft Word and AutoCAD programs. Many modern programming languages such as VB.NET and C# also support ActiveX, even legacy languages such as C++ even support ActiveX. In addition to mainstream programming languages, AutoLISP, or rather Visual LISP, also supports the use of ActiveX on Windows.

The AutoCAD Type library is the most common ActiveX library that the AutoCAD program supports; this is the library referenced by VBA projects created with the AutoCAD VBA IDE. The AutoCAD Type library can be used to:

- Create an in-memory instance of the AutoCAD program
- Manipulate the AutoCAD application window and preferences
- Create and modify drawing objects

Note: The AutoCAD/ObjectDBX Common Type library can be used to open a drawing file in-memory without actually opening the drawing file in the AutoCAD drawing environment. Opening the drawing file outside of the AutoCAD drawing environment requires less memory and is more efficient.

For applications developed with VB.NET/C# or ObjectARX/C++, the AutoCAD Type library and AutoCAD/ObjectDBX Common Type library do not provide access to many features that cannot already be accessed with the ObjectARX and AutoCAD Managed .NET APIs. Beyond the AutoCAD Type and AutoCAD/ObjectDBX Common Type libraries, there are four other ActiveX libraries that the AutoCAD program supports which do allow access to features that are not part of the ObjectARX SDK or AutoCAD Managed .NET APIs.

The following lists the four feature specific ActiveX libraries that the AutoCAD program supports:

- **Connectivity Automation Object Library** - Connectivity Automation Object (CAO) library allows you to link database records to objects in a drawing file.
- **Sheet Set Object Type Library** - Sheet Set Object (SSO) library allows you to create and modify the components in a sheet set (DST) file and access all open sheet sets in the Sheet Set Manager.
- **Standards Manager Type Library** - Standards Manager library, more commonly known as the CAD Standards Plug-ins (CSP) library, allows you to define a CAD standard plug-in that can be used to check for and fix drawing standards violations.
- **Autodesk Transmittal Type Library** - Transmittal Type (ATO) library allows you to create a transmittal package from a drawing file and its referenced files that, or an archival package from a sheet set and all the files that it references.

This session provides a basic introduction of the four previously mentioned ActiveX libraries. As part of the introduction, you will learn many of the concepts related to implementing and using the objects exposed by the libraries. All of the sample code is shown in VB.NET, but it can be translated to C# using

an online convertor such as Convert VB.NET to C# on the developerFusion website (*http://www.developerfusion.com/tools/convert/vb-to-csharp/*).

2 What You Need Before Getting Started

Many of the application programming interfaces (APIs) mentioned during this session can be utilized with the VBA programming language and AutoCAD VBA IDE with one exception; the CAD Standards Plug-ins (CSP) library which requires the creation of a standalone DLL.

You will want to use the following to get the most out of the APIs covered in this session:

- **Microsoft Visual Studio** – While the AutoCAD VBA IDE can be used with most of the APIs covered in this session, you will need Microsoft Visual Studio to utilize the CAD Standards Plug-ins (CSP) library. The release of Visual Studio you need to use to build an application depends on the release of the AutoCAD program you are targeting.
    - AutoCAD 2015 and AutoCAD 2016 - Visual Studio 2012 or 2013
    - AutoCAD 2013 and AutoCAD 2014 - Visual Studio 2010
    - AutoCAD 2010 through AutoCAD 2012 - Visual Studio 2008
    - AutoCAD 2007 through AutoCAD 2009 - Visual Studio 2005

  You can download and use Visual Studio Express, but you might have issues with debugging and supporting 64-bit applications. Visual Studio Express for Desktop can be downloaded at *http://www.microsoft.com/express/downloads*. You should be able to download and install the Windows SDK to build 64-bit applications.

- **Windows SDK** – If you are going to use Visual Studio Express, you will need to download and install the Windows SDK. The *Tlbimp.exe* file from the Windows SDK will also be needed need to generate the required file for working with the CAD Standards Plug-ins (CSP) library if you don't have Microsoft Visual Studio installed. The Windows SDK can also be helpful to developing and debugging applications even if you have Visual Studio installed.

  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.

- **ObjectARX Software Development Kit (SDK)** – The ObjectARX SDK contains all of the COM libraries mentioned in this session along with the CAD Standards Plug-ins library (*AcStMgr.tlb*). In addition to the COM libraries mentioned in this session, the documentation for the CAD Standards Plug-ins (CSP) and Transmittal libraries.

  You can download the ObjectARX SDK from *http://www.autodesk.com/objectarx*. Click License & Download and then follow the onscreen instructions.

- **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 *http://help.autodesk.com/view/ACD/2016/ENU/files/homepage_dev.htm*.

- **AutoCAD ActiveX Reference Guide** – The AutoCAD ActiveX documentation will be useful when working with the ActiveX/COM libraries mentioned in this session. The ActiveX/COM libraries

mentioned in this session return and expect database and drawing objects based on ActiveX/COM objects; these are not the same as those that are part of the AutoCAD Managed .NET library.

You can access the AutoCAD ActiveX Reference Guide by browsing to *C:\Program Files\Common Files\Autodesk Shared\acadauto.chm* or from the Developer's Home page in the AutoCAD Online help at *http://help.autodesk.com/view/ACD/2016/ENU/files/homepage_dev.htm*.

3 Accessing the AutoCAD Application

Most of the ActiveX libraries that the AutoCAD program supports require the use and understanding of two additional libraries: AutoCAD Type Library and AutoCAD/ObjectDBX Common Type Library. The AutoCAD Type Library and AutoCAD/ObjectDBX Common Type Library are used to access the components of the AutoCAD program and the objects stored in a drawing file. When you reference either the AutoCAD Type Library or AutoCAD/ObjectDBX Common Type Library, you need to know which version of the library to use. The library version you need to reference is dependent on the AutoCAD release in which your program will interact.

Note: It is possible to reference a version of the library that doesn't correspond to a release(s) of the AutoCAD program installed on your computer. The ObjectARX SDK can be used to reference the ActiveX libraries that should be used to interact with a specific release of the AutoCAD program. You can download the ObjectARX SDK at *http://www.objectarx.com*. While you can reference other versions of an ActiveX library than those installed with an AutoCAD release, be sure to test your programs with all target releases.

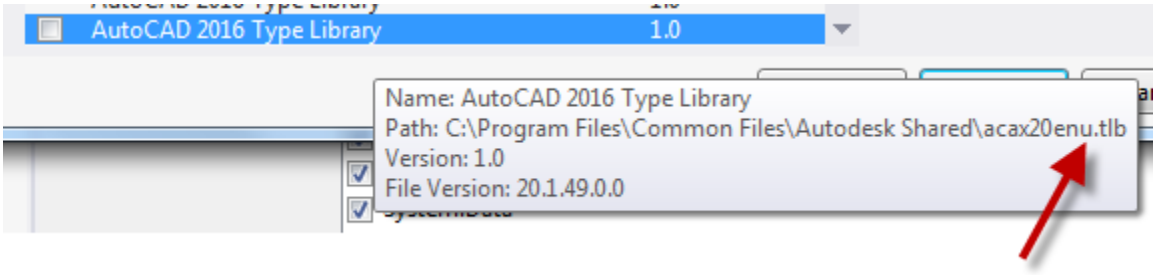
The following table lists the file name and version of the ActiveX libraries you need to reference by AutoCAD release.

AutoCAD Release	AutoCAD Type Library	AutoCAD/ObjectDBX Common Type Library
AutoCAD 2016 and AutoCAD 2015	*acax20enu.tlb*	*axdb20enu.tlb*
AutoCAD 2014 and AutoCAD 2013	*acax19enu.tlb*	*axdb19enu.tlb*
AutoCAD 2012, AutoCAD 2011, and AutoCAD 2010	*acax18enu.tlb*	*axdb18enu.tlb*
AutoCAD 2009, AutoCAD 2008, and AutoCAD 2007	*acax17enu.tlb*	*axdb17enu.tlb*

The following steps explain how to add references to the AutoCAD Type Library in a VB.NET project:

1. In Visual Studio, on the menu bar, click Project ➢ Add Reference.
2. In the Reference Manager dialog box, click COM along the left side.
3. In the ActiveX Library list, click the check box to the left of the AutoCAD *<release>* Type Library to reference.
   Multiple AutoCAD <release> Type Library entries will be listed, any one of them can be selected. The different entries represent each of the languages available for the installed library. Selecting one of the libraries results in the selection of all the libraries when the library you selected is added to the project. Positioning the cursor over one of the library entries allows

you to see the language of the library; look at the end of the file name for the library's language, "enu" indicates the library is defined for the English language.



4. Click OK to add the selected library references to the project.

If you need to reference the AutoCAD/ObjectDBX Common Type Library, in Step 3, select AutoCAD/ObjectDBX Common *<release>* Type Library from the ActiveX Library list.

Once you add a reference to an ActiveX library in your project file, you need to import or indicate that you are using the library in a code module. The following code snippet imports the AutoCAD Type Library and AutoCAD/ObjectDBX Common Type Library in a VB.NET code module:

```
' AutoCAD Type Library namespace
Imports AutoCAD

' AutoCAD/ObjectDBX Common Type Library namespace
Imports AXDBLib
```

Note: Alternatively, you can also reference the *Autodesk.AutoCAD.Interop.dll* and *Autodesk.AutoCAD.Interop.Common.dll* files in the ObjectARX SDK which provide similar functionality.

```
' AutoCAD Type Library Interop/COM namespace
Imports Autodesk.AutoCAD.Interop

' AutoCAD/ObjectDBX Common Type Library Interop/COM namespace
Imports Autodesk.AutoCAD.Interop.Common
```

4 Connecting Objects to a Database

Attributes in blocks are a great way to provide information about the real-world object or objects that a block might represent in a design. Using the text assigned to the attributes of a block, it is possible to generate a bill of materials (BOM) based on the blocks inserted into a drawing through the process of extracting the attributes in a block. The values in an attribute are static which isn't always ideal, static attribute values require you to modify a drawing to change their value. What if it was possible to connect a block or any object in a drawing to an external data source where values could be updated without modifying the drawing.

The Database Connectivity feature allows you to attach a database record to one or more objects in a drawing file. The link between an object and a database recorded is maintained in a drawing, while all of the actual values are stored in the database itself. Database connections in a drawing can be managed with the:

- dbConnect Manager palette in the AutoCAD user interface

- Connectivity Automation Object (CAO) library via a custom program

The Connectivity Automation Object (CAO) library allows you to perform the same tasks as those available in the dbConnect Manager. You can use the Connectivity Automation Object library to:

- Attach or remove between a database record and a drawing object
- Work with link templates and queries stored in a drawing
- Create label templates and attach a label to an object

What You Need Before Getting Started

Before you start linking to objects to a database record, you will need to obtain the following:

- **Connectivity Automation Object Library (*cao20enu.tlb*)** – File is installed with the AutoCAD program.
- **Connectivity Automation Object Documentation** – The Developer Home page in the AutoCAD Online Help system contains the reference and concept topics related to the Connectivity Automation Object library. On the Developer's Home Page, click the Connectivity Automation Object (CAO) Reference link. To access the online help, go to: *http://help.autodesk.com/view/ACD/2016/ENU/files/homepage_dev.htm*.
- **AutoCAD ActiveX Documentation** – The documentation for the AutoCAD Type Library is installed with the AutoCAD program. You can access the AutoCAD ActiveX Reference and AutoCAD ActiveX Developers guides by browsing to *C:\Program Files\Common Files\Autodesk Shared* and opening the files *acadauto.chm* and *acad_aag.chm*. The topics can also be found in the AutoCAD Online Help system, on the Developer's Home Page, click the AutoCAD ActiveX Library Reference and AutoCAD ActiveX Developer's Guide links.

Taking a Look at the Connectivity Automation Object Library

The first thing that you need to know is how to reference the Connectivity Automation Object library. The Connectivity Automation Object library is stored in a file named *cao<version><language>.tlb* under the Autodesk Shared subfolder in the *Program Files\Common Files* folder; in the Reference Manager it is listed as CAO 1.0 Type Library. When working with the Connectivity Automation Object library, you need to know the version of the library required based on the release of the AutoCAD program in which your program will interact. You most likely will reference the English version of the library, but the appropriate library is loaded based on which language of Windows is being used. The *cao20enu.tlb* file represents the base library version of 20 with the English language.

The following table lists the file name and version of the ActiveX libraries you need to reference by AutoCAD release.

AutoCAD Release	Connectivity Automation Object Library	AutoCAD/ObjectDBX Common Type Library
AutoCAD 2016 and AutoCAD 2015	*cao20<language>.tlb*	*axdb20<language>.tlb*
AutoCAD 2014 and AutoCAD 2013	*cao16<language>.tlb*	*axdb19<language>.tlb*
AutoCAD 2012, AutoCAD 2011, and AutoCAD 2010	*cao16<language>.tlb*	*axdb18<language>.tlb*

AutoCAD 2009, AutoCAD 2008, and AutoCAD 2007	*cao16<language>.tlb*	*axdb17<language>.tlb*

Once you have referenced the library, you need to reference the namespace that contains the Connectivity Automation Object library. The namespace of the library is similar to the name of the referenced TLB file. The following statement imports the Connectivity Automation Object library in a VB.NET code module:

```
Imports CAO
```

After the Connectivity Automation Object library namespace has been imported, you must create a reference to the Database Connection object which is represented by the `DbConnect` object. The following code demonstrates how to reference the Database Connection object:

```
' Get a reference to the Database Connection object
Dim g_oCAO As New CAO.DbConnect
```

With the Database Connection object, you can

- Connect and disconnect from a data source(`Connect/Disconnect`)
- Check for the connection of a data source (`IsConnected`)
- Create and modify the link templates stored in a drawing (`GetLinkTemplates`)
- Get and attach a link between objects in a drawing to a database record (`GetLinks/GetLinksByKey/GetLinksByLabelTemplate/GetLinksBySQLCond`)
- Create and modify the label templates stored in a drawing (`GetLabelTemplates`)
- Reload the labels of objects in a drawing (`ReloadLabels`)
- Define and change the queries stored in a drawing (`GetQueries`)

Creating a UDL File

Before you can work with the database components stored in a drawing, or the links between objects in a drawing and database records you need to connect to a data source. A data source must be defined in a UDL file before it can be used by the Database Connection object. A UDL file contains the database provider and connection string needed to access a data source.

You can create a UDL file using the DBCCONFIGURE command or create a Unicode file using VB.NET or C#. The UDL file created with the DBCCONFIGURE command can be copied between workstations as long as the database referenced is accessible to the user logged into the computer. If you create the UDL file with VB.NET or C# or another programming language, the file must be saved as a Unicode encoded file.

The following sample code creates a UDL file and makes a connection to the data source with the `Connect()` method.

```
Private g_oCAO As New CAO.DbConnect
Private g_linkTemplates As CAO.LinkTemplates
Private g_labelTemplates As CAO.LabelTemplates
Private g_queries As CAO.Queries
Private g_links As CAO.Links
Private g_sDataSource As String = "AU2015"

' Creates a UDL file based on the name assigned to the g_sDataSource variable.
' A connection to the datasource is then made with the Database Connection object.
<CommandMethod("CAO_CreateUDLFile")> _
```

```
Public Sub CAO_CreateUDLFile()
    ' Gets the editor object of the current drawing
    Dim acEditor As Editor = Application.DocumentManager.MdiActiveDocument.Editor

    ' Defines the location and name for the new UDL file
    Dim sDatabasePath As String = HostApplicationServices. _
        Current.GetEnvironmentVariable("DatabaseWorkSpacePath")
    Dim sFilename As String = sDatabasePath + "\" + g_sDataSource + ".udl"

    ' Determines whethere the UDL file already exists
    If FileIO.FileSystem.FileExists(sFilename) = False Then
        ' Creates the UDL file on disk
        Dim oFile As System.IO.FileStream = System.IO.File.Create(sFilename)
        oFile.Dispose()

        ' Writes the provider and connection string information to the UDL file
        Dim oWriter As New System.IO.StreamWriter(sFilename, False, _
                                                  System.Text.Encoding.Unicode)
        oWriter.WriteLine("[oledb]")
        oWriter.WriteLine("; Everything after this line is an OLE DB initstring")
        oWriter.WriteLine("Provider=Microsoft.ACE.OLEDB.12.0;Data " + _
                          "Source=C:\Datasets\Thursday\IT10489 - " + _
                          "Harnessing the Power of the AutoCAD COM APIs\" + _
                          "AU2015.accdb;Persist Security Info=False")
        oWriter.Close()
    End If

    ' Determines whether a valid Database Connection object was created during
    ' the loading of the class
    If IsNothing(g_oCAO) Then
        acEditor.WriteMessage(vbLf + "CAO.dbConnect wasn't created.")
        Exit Sub
    End If

    Try
        ' Connects to the data source
        g_oCAO.Connect(g_sDataSource, "", "", False)

        ' Lists information about the current database connection
        acEditor.WriteMessage(vbLf + "  DataSourceLocation: " + _
                              g_oCAO.DataSourceLocation)
        acEditor.WriteMessage(vbLf + "  IsConnected:        " + _
                              g_oCAO.IsConnected(g_sDataSource).ToString())
        acEditor.WriteMessage(vbLf + "  Version:            " + g_oCAO.Version)
    Catch es As SystemException
        ' Displays information about the error generated by an invalid data source
        acEditor.WriteMessage(vbLf + es.Message)
    End Try
End Sub
```

The UDL file that is created is stored in the location identified by the environment variable "DatabaseWorkSpacePath"; this environment variable contains the location defined under the Data Sources Location node on the Files tab of the Options dialog box.

The default location for UDL files is: *%AppData%\Autodesk\AutoCAD 2016\R20.1\enu\Data Links*

Here is what the contents of the UDL file contains after executing the previous sample:

```
[oledb]
; Everything after this line is an OLE DB initstring
Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Datasets\Thursday\IT10489 - Harnessing the
Power of the AutoCAD COM APIs\AU2015.accdb;Persist Security Info=False
```

After the connection to the data source is made, the information about the connection is displayed in the Command Line History window. The following is an example of the values output:

```
DataSourceLocation:  C:\Users\ambrosl\appdata\roaming\autodesk\autocad 2016\r20.1\enu\data links
IsConnected:         True
Version:             AutoCAD SQL Environment T
```

Defining Link Templates

A link between a drawing object and database record can be created, but before a link can be created a link template must be defined. A link template is used to identify the unique field of a database record that can be used to map an object back to the database; this is often an index number or a key name. This is possible to use two or more fields to identify a single database record.

You must know the following information to create a link template:

- Data source; name of the UDL file without the file extension
- Optionally, the catalog and schema for the database
- Name of the table in the database that contains the data which you want to link
- Name of the link template to create
- Names of the fields in the table that contain the unique values to identify a database record

A link template is created using the `Add()` method of the `LinkTemplates` collection. The `LinkTemplates` collection can be obtained with the `GetLinkTemplates()` method of the Database Connectivity object (`DbConnect`).

The following sample code creates a link template named ProjectsLink.

```
' Creates a new link template named "ProjectsLink" if it does not already exist.
' A link template is required to link a drawing object with a database record.
<CommandMethod("CAO_CreateLinkTemplate")> _
Public Sub CAO_CreateLinkTemplate()
    ' Gets the editor object of the current drawing
    Dim acEditor As Editor = Application.DocumentManager.MdiActiveDocument.Editor
    ' Gets the ActiveX object that represents the active drawing
    Dim acAxDoc As AcadDocument = _
        DocumentExtension.GetAcadDocument( _
            Application.DocumentManager.MdiActiveDocument)

    ' Determines whether a valid Database Connection object was created during
    ' the loading of the class
    If IsNothing(g_oCAO) Then
        acEditor.WriteMessage(vbLf + "CAO.dbConnect wasn't created.")
        Exit Sub
    End If

    ' Determines whether the Database Connection object is connected to the data source
```

```
    If g_oCAO.IsConnected(g_sDataSource) = False Then
        g_oCAO.Connect(g_sDataSource)
    End If

    ' Gets the link templates stored in the current drawing
    g_linkTemplates = g_oCAO.GetLinkTemplates(acAxDoc)

    ' Checks to see if the returned object is valid
    If Not IsNothing(g_linkTemplates) Then
        Dim linkTemplate As CAO.LinkTemplate = Nothing

        ' Sets the default name for the link template
        Dim sLinkTemplateName As String = "ProjectsLink"

        Try
            ' Tries to get the link template named "ProjectsLink"
            linkTemplate = g_linkTemplates.Item(sLinkTemplateName)
        Catch es As SystemException
            ' The following is executed if the label template
            ' named "ProjectsLabel" does not exist

            ' Sets the field that contains the index for the database records
            Dim keyDescs As New KeyDescriptions
            keyDescs.Add("ID", CaoDataType.kCaoTypeInteger, 0, 0)

            ' Creates the link template named "ProjectsLink"
            linkTemplate = g_linkTemplates.Add(g_sDataSource, "", "", _
                                               "Projects", sLinkTemplateName, keyDescs)
        End Try
    End If
End Sub
```

Note: Link templates are stored in a drawing via the named dictionary “CONLDEFDictionary”.

Linking Objects to a Database Record

After a link template has been created, you can create a link between an object in a drawing and a record in the table referenced by the link template. The creation of a link requires you to know the following:

- Link template to use
- Object ID of the object in which to link to the database record
- Fields and values that identify the database record that should be linked to the object

You create a link between an object and a database record with the `CreateLink()` method. The `CreateLink()` method is a member of the `LinkTemplate` object. You can get the link template you want to work with using the `LinkTemplates` collection. The `LinkTemplates` collection can be obtained with the `GetLinkTemplates()` method of the Database Connectivity object (`DbConnect`).

The following sample code creates a new link between the database record with the unique ID of 1 and a user specified object.

```
' Creates a link between a drawing object and a database record.
<CommandMethod("CAO_CreateLink")> _
```

```
Public Sub CAO_CreateLink()
    ' Gets the editor object of the current drawing
    Dim acEditor As Editor = Application.DocumentManager.MdiActiveDocument.Editor

    ' Gets the ActiveX object that represents the active drawing
    Dim acAxDoc As AcadDocument = _
        DocumentExtension.GetAcadDocument( _
            Application.DocumentManager.MdiActiveDocument)

    ' Determines whether a valid Database Connection object was created during
    ' the loading of the class
    If IsNothing(g_oCAO) Then
        acEditor.WriteMessage(vbLf + "CAO.dbConnect wasn't created.")
        Exit Sub
    End If

    ' Determines whether the Database Connection object
    ' is connected to the data source
    If g_oCAO.IsConnected(g_sDataSource) = False Then
        g_oCAO.Connect(g_sDataSource)
    End If

    ' Prompts the user to select an object
    Dim prGetEnt As PromptEntityResult = _
        acEditor.GetEntity(vbLf + "Select an object to add a link: ")

    ' Determines whether the user selected an object
    If prGetEnt.ObjectId <> ObjectId.Null Then

        ' Defines the field and value which should be used
        ' to link the object and database record
        Dim keyVals As New CAO.KeyValues
        keyVals.Add("ID", 1)

        ' Gets the link template to use
        Dim linkTemplate As CAO.LinkTemplate = _
            g_oCAO.GetLinkTemplates(acAxDoc).Item("ProjectsLink")

        ' Creates the link between the object and database record
        Dim link As CAO.Link = _
            linkTemplate.CreateLink(prGetEnt.ObjectId.OldIdPtr.ToInt64(), keyVals)

        ' Determines whether the link was successfully created
        If Not IsNothing(link) Then
            acEditor.WriteMessage(vbLf + "Link was created.")
        Else
            acEditor.WriteMessage(vbLf + "Link wasn't created.")
        End If
    End If
End Sub
```

The link between an object and a database record is represented by extended data (Xdata) attached to an object. It is possible to remove or modify the Xdata after it has been attached to an object. The xdata for a link looks similar to the following:

```
(-3 ("DCO15" (1071 . 5) (1071 . 5) (1000 . "ProjectsLink") (1004 . "03000000")))
```

It is possible for an object to be linked to more than one database records. Linking an object to more than one database records can be helpful if building an application that tracks the history of an object or the ownership of an asset. When you want to create multiple links on a single object, you simply execute the `CreateLink()` method more than once.

The following sample code creates multiple links on a single object; the records that are linked to the object have the values of 1 and 4 in the ID field.

```
' Creates multiple links between a drawing object and several database records.
<CommandMethod("CAO_CreateMultipleLinks")> _
Public Sub CAO_CreateMultipleLinks()
    ' Gets the editor object of the current drawing
    Dim acEditor As Editor = Application.DocumentManager.MdiActiveDocument.Editor

    ' Gets the ActiveX object that represents the active drawing
    Dim acAxDoc As AcadDocument = _
        DocumentExtension.GetAcadDocument( _
            Application.DocumentManager.MdiActiveDocument)

    ' Determines whether a valid Database Connection object was created during
    ' the loading of the class
    If IsNothing(g_oCAO) Then
        acEditor.WriteMessage(vbLf + "CAO.dbConnect wasn't created.")
        Exit Sub
    End If

    ' Determines whether the Database Connection object is connected to the data source
    If g_oCAO.IsConnected(g_sDataSource) = False Then
        g_oCAO.Connect(g_sDataSource)
    End If

    ' Prompts the user to select an object
    Dim prGetEnt As PromptEntityResult = _
        acEditor.GetEntity(vbLf + "Select an object to add links: ")

    ' Determines whether the user selected an object
    If prGetEnt.ObjectId <> ObjectId.Null Then

        ' Defines the field and value which should be used
        ' to link the object and database record
        Dim keyVals As New CAO.KeyValues
        keyVals.Add("ID", 1)

        ' Gets the link template to use
        Dim linkTemplate As CAO.LinkTemplate = _
            g_oCAO.GetLinkTemplates(acAxDoc).Item("ProjectsLink")

        ' Creates the link between the object and database record
        Dim link As CAO.Link = _
            linkTemplate.CreateLink(prGetEnt.ObjectId.OldIdPtr.ToInt64(), keyVals)

        ' Determines whether the link was successfully created
        If Not IsNothing(link) Then
```

```
            acEditor.WriteMessage(vbLf + "First link was created.")
        Else
            acEditor.WriteMessage(vbLf + "First link wasn't created.")
        End If

        ' Creates the second link between the object and database record
        keyVals = New CAO.KeyValues
        keyVals.Add("ID", 4)
        link = linkTemplate.CreateLink(prGetEnt.ObjectId.OldIdPtr.ToInt64(), keyVals)

        ' Determines whether the link was successfully created
        If Not IsNothing(link) Then
            acEditor.WriteMessage(vbLf + "Second link was created.")
        Else
            acEditor.WriteMessage(vbLf + "Second link wasn't created.")
        End If
    End If
End Sub
```

After a link is created, it is added to the `Links` collection. Unlike most collections, links are attached directly to an object and not centralized in a drawing. There are two approaches that can be used to identify objects that have a link attached to them:

- Xdata attached to an object can be used to identify and filter out objects that have links attached to them along with which link template was used. Using the Xdata, it is possible to create a selection filter that can be used when prompting the user to select objects in a drawing.
- The Connectivity Automation Object (CAO) library provides a direct approach to filtering out objects that have links attached to them with the `GetLinks()` method. The `GetLinks()` method doesn't require the use of any selection methods, so it is possible to access links without the drawing being open in the editor.

The `GetLinks()` method is the preferred approach to getting objects with links in a drawing. You must know the following to use the `GetLinks()` method:

- Link template used to create the link between an object and database record
- An array of object IDs that you want to filter to identify the objects with links that match the expected criteria
- Link types to filter on; entities, attached and freestanding labels

The following sample code updates a link to use the database record with a value of 4 in the ID field instead of the database record with the value of 1 in the ID field.

```
' Replaces the key value of an object link
<CommandMethod("CAO_EditLink")> _
Public Sub CAO_EditLink()
    ' Gets the editor object of the current drawing
    Dim acEditor As Editor = Application.DocumentManager.MdiActiveDocument.Editor

    ' Gets the ActiveX object that represents the active drawing
    Dim acAxDoc As AcadDocument = _
        DocumentExtension.GetAcadDocument( _
            Application.DocumentManager.MdiActiveDocument)

    ' Determines whether a valid Database Connection object was created during
```

```
    ' the loading of the class
    If IsNothing(g_oCAO) Then
        acEditor.WriteMessage(vbLf + "CAO.dbConnect wasn't created.")
        Exit Sub
    End If

    ' Determines whether the Database Connection object is connected to the data source
    If g_oCAO.IsConnected(g_sDataSource) = False Then
        g_oCAO.Connect(g_sDataSource)
    End If

    ' Prompts the user to select an object
    Dim prGetEnt As PromptEntityResult = _
        acEditor.GetEntity(vbLf + "Select object to edit link: ")

    ' Determines whether the user selected an object
    If prGetEnt.ObjectId <> ObjectId.Null Then

        ' Defines the field and value which should be used
        ' to link the object and database record
        Dim keyVals As New CAO.KeyValues
        keyVals.Add("ID", 4)

        ' Creates an array that contains the object with the link to be labeled
        Dim lObjIds(0) As Long
        lObjIds(0) = prGetEnt.ObjectId.OldIdPtr.ToInt64()

        ' Gets the link template to use
        Dim linkTemplate As CAO.LinkTemplate = _
            g_oCAO.GetLinkTemplates(acAxDoc).Item("ProjectsLink")
        ' Filters out the selected objects that don't match
        ' the link template and specified link types
        g_links = g_oCAO.GetLinks(linkTemplate, lObjIds, _
                                  CAO.LinkType.kEntityLinkType + _
                                  CAO.LinkType.kAttachedLabelType + _
                                  CAO.LinkType.kFSLabelType)

        ' Determines whether there is one or more links that match the filter criteria
        If g_links.Count > 0 Then

            ' Steps through each link and update its key values
            For Each link As CAO.Link In g_links
                link.KeyValues = keyVals

                ' Gets the label of the link object
                Dim label As CAO.Label = link.Label

                ' If a valid label was found, refresh and reload the label to
                ' display the new values of the new record
                If IsNothing(label) = False Then
                    link.Label.Refresh()
                    link.Label.Reload()
                End If
            Next
        Else
```

```
            acEditor.WriteMessage(vbLf + "No link found on selected object.")
        End If
    End If
End Sub
```

In addition to the `GetLinks()` method, there are three other methods that can be used to obtain a `Links` collection that represents objects with links in a drawing. The following methods can be used to identify objects in a drawing with one or more links to a database record:

- `GetLinksByKey` – Returns a `Links` collection that contains the objects in a drawing with links that use a specific link template and have a specific key value.
- `GetLinksByLabelTemplate` – Returns a `Links` collection that contains the objects in a drawing with label links that use a specific label template.
- `GetLinksBySQLCond` – Returns a `Links` collection that contains the objects in a drawing with links that match the values of the SQL statement.

A drawing can contain objects with links that were created using multiple link templates. The methods previously mentioned return a `Links` collection containing links created with a specific link template. It is possible to merge two `Links` collections or remove `Link` objects from a `Links` collection using three special functions. These functions perform Boolean like functionality on a `Links` collection. The Boolean like functions for a `Links` collection are:

- `Intersect` – Returns a `Links` collection that only contains the linked objects whose links overlap between two `Links` collections.
- `Subtract` – Returns a `Links` collection that only contains the linked objects that do not exist in two `Links` collections.
- `Union` – Returns a `Links` collection that contains the linked objects in two `Links` collection.

Defining Label templates

A label is a leader and/or multiline text (Mtext) object that displays one or more values from the database record in which the label is linked. You can create two types of labels: attached and freestanding. An attached label contains a leader that points to the object which it is annotating along with an Mtext object that describes the object based on values from the data source. A freestanding label does not have a leader; it is simply an Mtext object. Both attached and freestanding labels contain links that reference back to a database record.

A label template is used to identify the fields from a database record that should be used to populate the content of a label. In addition to specifying the content of a label, a label template defines the format in which the text appears in a label along with some basic properties of the leader created for an attached label.

You must know the following information to create a label template:

- Data source; name of the UDL file without the file extension
- Optionally, the catalog and schema for the database
- Name of the table in the database that contains the data for the label
- Name of the label template to create
- Names of the fields that represent the values to use in the label from the database record

A label template is created using the `Add()` method of the `LabelTemplates` collection. The `LabelTemplates` collection can be obtained with the `GetLabelTemplates()` method of the Database Connectivity object (`DbConnect`).

The following sample code creates a label template named ProjectsLabel.

```
' Creates a new label template named "ProjectsLabel" if it does not already exist.
' A label template is required to label a linked object or place a freestanding
' label in a drawing.
' A label displays values from the linked database record.
<CommandMethod("CAO_CreateLabelTemplate")> _
Public Sub CAO_CreateLabelTemplate()
    ' Gets the editor object of the current drawing
    Dim acEditor As Editor = Application.DocumentManager.MdiActiveDocument.Editor

    ' Gets the ActiveX object that represents the active drawing
    Dim acAxDoc As AcadDocument = _
        DocumentExtension.GetAcadDocument( _
            Application.DocumentManager.MdiActiveDocument)

    ' Determines whether a valid Database Connection object was created during
    ' the loading of the class
    If IsNothing(g_oCAO) Then
        acEditor.WriteMessage(vbLf + "CAO.dbConnect wasn't created.")
        Exit Sub
    End If

    ' Determines whether the Database Connection object is connected to the data source
    If g_oCAO.IsConnected(g_sDataSource) = False Then
        g_oCAO.Connect(g_sDataSource)
    End If

    ' Gets the label templates stored in the current drawing
    g_labelTemplates = g_oCAO.GetLabelTemplates(acAxDoc)

    ' Checks to see if the returned object is valid
    If Not IsNothing(g_labelTemplates) Then
        Dim labelTemplate As CAO.LabelTemplate = Nothing

        ' Sets the default name for the label template
        Dim sLabelTemplateName As String = "ProjectsLabel"

        ' Specifies the fields in the table that will be used in the label template
        Dim sFieldNames() As String = {"CompanyName", "Address1", "Address2", _
                                       "City", "State", "ZIPCode"}

        Try
            ' Tries to get the label Template named "ProjectsLabel"
            labelTemplate = g_linkTemplates.Item(sLabelTemplateName)
        Catch es As SystemException
            ' The following is executed if the label template
            ' named "ProjectsLabel" does not exist

            ' Creates the label template named "ProjectsLabel"
            labelTemplate = g_labelTemplates.Add(g_sDataSource, "", "", _
```

```
                                                "Projects", "ProjectsLabel", _
                                                sFieldNames)

            ' Sets the format for the label text
            ' - Field names are formatted as #(<field name>)
            ' - \P Indicates the start of a new line\paragraph
            labelTemplate.Format = "#(CompanyName)\P#(Address1)\P" + _
                                   "#(Address2)\P#(City), #(State)  #(ZIPCode)"

            ' Sets the leader attachment style
            labelTemplate.LeaderAttachment = _
                AutoCAD.AcAttachmentPoint.acAttachmentPointMiddleCenter

            ' Sets the leader offset; distance between the start of
            ' the leader (arrowhead) and the end point of the landing
            Dim dLeaderOffset() As Double = {1.0, 1.0, 0.0}
            labelTemplate.LeaderOffset = dLeaderOffset

            ' Sets the color and linetype for the leader line
            labelTemplate.Color = AutoCAD.AcColor.acByLayer
            labelTemplate.Linetype = "ByLayer"

            ' Sets the rotation, text size, and text style for the label
            labelTemplate.Rotation = 0
            labelTemplate.TextHeight = 0.2
            labelTemplate.TextStyle = "Standard"

            ' Sets the offset from which the leader would normally start
            ' The origin of the leader is typically the center of the object's extents
            Dim dTipOffset() As Double = {0.0, 0.0, 0.0}
            labelTemplate.TipOffset = dTipOffset
        End Try
    End If
End Sub
```

Note: Label templates are stored in a drawing via the named dictionary “CONLABELDictionary”.

Attaching Labels to an Object and Creating Freestanding Labels

After a label template has been created, you can create a label that is attached to a linked object or a freestanding label that references a database record. The creation of a label requires you to know the following:

- The object to label when creating an attached label or a coordinate value when creating a freestanding label
- Fields and values that identify the database record that should be used for the label
- Label template that controls the format of the label

You create a label with the `CreateLabel()` method. The `CreateLabel()` method is a member of the `LinkTemplate` object. You can get the link and label templates you want to work with using the `LinkTemplates` and `LabelTemplates` collections. The `LinkTemplates` and `LabelTemplates` collection can be obtained with the `GetLinkTemplates()` and `GetLabelTemplates()` methods of the Database Connectivity object (`DbConnect`).

The following sample code creates a new attached label by prompting the user to select an entity with a database link.

```
' Creates a label that is attached to an object that is linked to a database record
<CommandMethod("CAO_CreateLabelAttached")> _
Public Sub CAO_CreateLabelAttached()
    ' Gets the editor object of the current drawing
    Dim acEditor As Editor = Application.DocumentManager.MdiActiveDocument.Editor

    ' Gets the ActiveX object that represents the active drawing
    Dim acAxDoc As AcadDocument = _
        DocumentExtension.GetAcadDocument( _
            Application.DocumentManager.MdiActiveDocument)

    ' Determines whether a valid Database Connection object was created during
    ' the loading of the class
    If IsNothing(g_oCAO) Then
        acEditor.WriteMessage(vbLf + "CAO.dbConnect wasn't created.")
        Exit Sub
    End If

    ' Determines whether the Database Connection object is connected to the data source
    If g_oCAO.IsConnected(g_sDataSource) = False Then
        g_oCAO.Connect(g_sDataSource)
    End If

    ' Prompts the user to select an object
    Dim prGetEnt As PromptEntityResult = _
        acEditor.GetEntity(vbLf + "Select an object to add a label: ")

    ' Determines whether the user selected an object
    If prGetEnt.ObjectId <> ObjectId.Null Then

        ' Creates an array that contains the object with the link to be labeled
        Dim lObjIds(0) As Long
        lObjIds(0) = prGetEnt.ObjectId.OldIdPtr.ToInt64()

        ' Gets the link template to filter on
        Dim linkTemplate As CAO.LinkTemplate = _
            g_oCAO.GetLinkTemplates(acAxDoc).Item("ProjectsLink")

        ' Gets the objects that contain links based on
        ' the link template named "ProjectsLink"
        g_links = g_oCAO.GetLinks(linkTemplate, lObjIds, CAO.LinkType.kEntityLinkType)

        ' Determines whether there is one link that matches the filter criteria
        If g_links.Count > 0 Then

            ' Gets the label template named "ProjectsLabel"
            Dim labelTemplate As CAO.LabelTemplate = _
                g_oCAO.GetLabelTemplates(acAxDoc).Item("ProjectsLabel")

            ' Gets the link object from the Links collection
            Dim link As CAO.Link = g_links(0)
```

```
            ' Creates the label for the selected object
            Dim label As CAO.Label = _
                linkTemplate.CreateLabel(lObjIds(0), link.KeyValues, labelTemplate)

            ' If a valid label was created, refresh and reload the label to
            ' display the values of the record
            If Not IsNothing(label) Then
                acEditor.WriteMessage(vbLf + "Label was created.")

                label.Reload()
                label.Refresh()
            Else
                acEditor.WriteMessage(vbLf + "Label wasn't created.")
            End If
        Else
            acEditor.WriteMessage(vbLf + "No link found on selected object.")
        End If
    End If
End Sub
```

The label that is created in a drawing is defined as a standard leader with a multiline text (Mtext) object, or a standalone Mtext object, but those objects also include some additional extended data (Xdata). When an attached label is created, there are three relationships defined:

- The object linked to the database with the attached label contains a reference to the leader of the label via the handle of the leader object. The linked object also contains a reference to the label template that was used.

  Drawing object with link and attached label

  ((-1 . <Entity name: 7ffffb04e00>) (0 . "LINE") (330 . <Entity name: 7ffffb039f0>) **(5 . "1F0")** (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbLine") (10 29.6659 4.87227 0.0) (11 34.1323 13.5292 0.0) **(210 0.0 0.0 1.0) (-3 ("DCO15" (1071 . 2) (1071 . 2) (1000 . "ProjectsLink") (1004 . "01000000") (1000 . "ProjectsLabel") (1005 . "1F7") (1071 . 1) (1000 . "ProjectsLink") (1004 . "01000000")))**)

- The leader object contains a reference to the Mtext object as part of its object definition.

  Leader of the attached label

  ((-1 . <Entity name: 7ffffb04e70>) (0 . "LEADER") (330 . <Entity name: 7ffffb039f0>) **(5 . "1F7")** (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbLeader") (3 . "Standard") (71 . 1) (72 . 0) (73 . 0) (74 . 0) (75 . 1) (40 . 1.2472) (41 . 2.90982) (76 . 3) (10 31.8991 9.20074 0.0) (10 32.6291 10.2007 0.0) (10 32.8091 10.2007 0.0) **(340 . <Entity name: 7ffffb04e60>)** (211 1.0 0.0 0.0) (210 0.0 0.0 1.0) (212 0.0 0.0 0.0) (213 0.0 0.0 0.0))

- The Mtext object has Xdata attached to it that references to the object which it annotates.

  Mtext of the attached label

  ((-1 . <Entity name: 7ffffb04e60>) (0 . "MTEXT") (5 . "1F6") (102 . "{ACAD_REACTORS") (330 . <Entity name: 7ffffb04e70>) (102 . "}") (330 . <Entity name: 7ffffb039f0>) (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbMText") (10 32.8991 10.2007 0.0) (40 . 0.2) (41 . 0.0) (46 . 0.0) (71 . 1) (72 . 5) (1 . "ABC Company, Inc.\\P123 Print Ave.\\P\\PSmall Town, CA

```
94123") (7 . "Standard") (210 0.0 0.0 1.0) (11 1.0 0.0 0.0) (42 . 2.90982) (43 . 1.2472) (50 . 0.0)
(73 . 1) (44 . 1.0) (-3 ("DCO15" (1005 . "1F0")))))
```

Note: The text string of the Mtext object contains the text of the label template plus the static values from the database record. When the label is reloaded, the text string of the Mtext object is recreated from the label template and database record.

The process of creating a freestanding label is similar to creating an attached label with the exception of providing a point in the drawing as opposed to the entity that that should be annotated. The following sample code creates a new freestanding label by prompting the user for a point in the drawing.

```
' Creates a label that is not attached to a specific object in the drawing
<CommandMethod("CAO_CreateLabelFreestanding")> _
Public Sub CAO_CreateLabelFreestanding()
    ' Gets the editor object of the current drawing
    Dim acEditor As Editor = Application.DocumentManager.MdiActiveDocument.Editor

    ' Gets the ActiveX object that represents the active drawing
    Dim acAxDoc As AcadDocument = _
        DocumentExtension.GetAcadDocument( _
            Application.DocumentManager.MdiActiveDocument)

    ' Determines whether a valid Database Connection object was created during
    ' the loading of the class
    If IsNothing(g_oCAO) Then
        acEditor.WriteMessage(vbLf + "CAO.dbConnect wasn't created.")
        Exit Sub
    End If

    ' Determines whether the Database Connection object is connected to the data source
    If g_oCAO.IsConnected(g_sDataSource) = False Then
        g_oCAO.Connect(g_sDataSource)
    End If

    ' Prompts the user to select an object
    Dim prGetPt As PromptPointResult = _
        acEditor.GetPoint(vbLf + "Specify the insertion point for the label: ")

    ' Determines whether the user specified a point
    If prGetPt.Status = PromptStatus.OK Then

        ' Defines the fields which should be used to link the label
        Dim keyVals As New CAO.KeyValues
        keyVals.Add("ID", 1)

        ' Gets the link template to link the label
        Dim linkTemplate As CAO.LinkTemplate = _
            g_oCAO.GetLinkTemplates(acAxDoc).Item("ProjectsLink")

        ' Gets the label template named "ProjectsLabel"
        Dim labelTemplate As CAO.LabelTemplate = _
            g_oCAO.GetLabelTemplates(acAxDoc).Item("ProjectsLabel")
```

```
        ' Creates the label at the specified point
        Dim label As CAO.Label = _
            linkTemplate.CreateLabel(prGetPt.Value.ToArray(), keyVals, labelTemplate)

        ' If a valid label was created, refresh and reload the label to
        ' display the values of the record
        If Not IsNothing(label) Then
            acEditor.WriteMessage(vbLf + "Label was created.")

            label.Reload()
            label.Refresh()
        Else
            acEditor.WriteMessage(vbLf + "Label wasn't created.")
        End If
    Else
        acEditor.WriteMessage(vbLf + "No point specified.")
    End If
End Sub
```

The content of a label can be updated by reloading database values with the `Reload()` method and then the updating the display of the object in the drawing with the `Refresh()` method. It is possible to reload the labels associated with a number of objects in a drawing using the `ReloadLabels()` method of the Database Connectivity object (`DbConnect`). The `ReloadLabels()` method expects an array of object IDs that identify the labels to be reloaded.

The following sample code reloads all labels in Model space.

```
' Reloads the labels for all objects in the current drawing
<CommandMethod("CAO_ReloadAllLabels")> _
Public Sub CAO_ReloadAllLabels()
    ' Gets the editor object of the current drawing
    Dim acEditor As Editor = Application.DocumentManager.MdiActiveDocument.Editor

    ' Gets the ActiveX object that represents the active drawing
    Dim acAxDoc As AcadDocument = _
        DocumentExtension.GetAcadDocument( _
            Application.DocumentManager.MdiActiveDocument)

    ' Gets the database of the current drawing
    Dim acCurDb As Database = Application.DocumentManager.MdiActiveDocument.Database

    ' Determines whether a valid Database Connection object was created during
    ' the loading of the class
    If IsNothing(g_oCAO) Then
        acEditor.WriteMessage(vbLf + "CAO.dbConnect wasn't created.")
        Exit Sub
    End If

    ' Determines whether the Database Connection object is connected to the data source
    If g_oCAO.IsConnected(g_sDataSource) = False Then
        g_oCAO.Connect(g_sDataSource)
    End If

    ' Defines an empty array of longs to be used for object IDs
    Dim lObjIds() As Long = Nothing
```

```
    ' Creates a transaction to step through Model space
    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

        ' Opens the Block table record for read
        Dim acBlkTbl As BlockTable
        acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId, _
                                     OpenMode.ForRead)
        ' Opens the Block table record Model space for read
        Dim acBlkTblRec As BlockTableRecord
        acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                        OpenMode.ForRead)

        ' Creates a counter variable
        Dim lCount As Long = 0

        ' Steps through each object in Model space
        For Each acObjId As ObjectId In acBlkTblRec

            ' Redimensions the object ID array
            ReDim Preserve lObjIds(0 To lCount)

            ' Adds the object's ID to the array
            lObjIds(lCount) = acObjId.OldIdPtr.ToInt64()

            ' Increments the counter
            lCount = lCount + 1
        Next

        ' Ignore the changes made in the transaction
        acTrans.Abort()
    End Using

    ' Reloads the labels in the current drawing
    g_oCAO.ReloadLabels()
End Sub
```

In addition to reloading and refreshing labels, labels can be hidden from the drawing and then reshown later. Hiding a label doesn't remove the label from a drawing. A label can be hidden or shown using these methods of a `Label` or `LabelTemplate` object:

- **`Hide`** - Marks the label as invisible
- **`Show`** - Marks the label as visible

Creating Queries

Queries are one of the least used objects in the Connectivity Automation Object (CAO) library, but they can be useful if you want to allow the user to view or edit subsets of data in the database represented by the data source. If you want to work with subsets of data in your program, you will want to use a programming library such as ADO or DAO.

You must know the following information to create a query:

- Data source; name of the UDL file without the file extension
- Optionally, the catalog and schema for the database

- Name of the table in the database that contains the data to query
- Name of the query to create
- SQL statement that should be executed by the query

A query is created using the `Add()` method of the `Queries` collection. The `Queries` collection can be obtained with the `GetQueries()` method of the Database Connectivity object (`DbConnect`).

The following sample code creates a query named ProjectsQuery.

```
' Creates a new query named "ProjectsQuery" if it does not already exist.
' A query can be used to view and edit database records in the AutoCAD UI.
<CommandMethod("CAO_CreateQuery")> _
Public Sub CAO_CreateQuery()
    ' Gets the editor object of the current drawing
    Dim acEditor As Editor = Application.DocumentManager.MdiActiveDocument.Editor

    ' Gets the ActiveX object that represents the active drawing
    Dim acAxDoc As AcadDocument = _
        DocumentExtension.GetAcadDocument( _
            Application.DocumentManager.MdiActiveDocument)

    ' Determines whether a valid Database Connection object was created during
    ' the loading of the class
    If IsNothing(g_oCAO) Then
        acEditor.WriteMessage(vbLf + "CAO.dbConnect wasn't created.")
        Exit Sub
    End If

    ' Determines whether the Database Connection object is connected to the data source
    If g_oCAO.IsConnected(g_sDataSource) = False Then
        g_oCAO.Connect(g_sDataSource)
    End If

    ' Gets the queries stored in the current drawing
    g_queries = g_oCAO.GetQueries(acAxDoc)

    ' Checks to see if the returned object is valid
    If Not IsNothing(g_queries) Then
        Dim query As CAO.Query = Nothing

        ' Sets the default name and SQL statement for the query
        Dim sQueryName As String = "ProjectsQuery"
        Dim sSQLStatement As String = "SELECT * FROM Projects WHERE ID = 1 And ID = 4"

        Try
            ' Tries to get the query named "ProjectsQuery"
            query = g_queries.Item(sQueryName)
        Catch es As SystemException
            ' Creates the query named "ProjectsQuery"
            query = g_queries.Add(g_sDataSource, "", "", _
                                  "Projects", sQueryName, sSQLStatement)
        End Try
    End If
End Sub
```

Note: Queries are stored in a drawing via the named dictionary “CONQUERYDictionary”.

Listing Components Stored in a Drawing

The Connectivity Automation Object (CAO) library provides access to many of the objects it creates through collections, those collections represent objects stored in a drawing as both graphical and non-graphical objects; named object dictionaries and extended data (Xdata) attached to objects.

The following collection objects are used to access the objects created by the CAO library and stored in a drawing:

- **`LabelTemplates`** – Collection of label templates for creating attached and freestanding labels.
- **`LinkTemplates`** – Collection of link templates for creating links between drawing objects and database records.
- **`Links`** – Collection of objects that have one or more links to database records.
- **`Queries`**– Collection of queries used to access a subset of records in a data source.

All of the previously mentioned collections have been used in sample code shown in this handout to check for the existence of a template or query before creating an object of the same kind along with getting links in a drawing.

The following sample code obtains and then lists the templates and queries stored in a drawing with the `GetLabelTemplates()`, `GetLinkTemplates()`, and `GetQueries()` methods.

```
' Displays information about the Connectivity Automation objects stored in the drawing
<CommandMethod("CAO_ListStoredComponents")> _
Public Sub CAO_ListStoredComponents()
    ' Gets the editor object of the current drawing
    Dim acEditor As Editor = Application.DocumentManager.MdiActiveDocument.Editor

    ' Gets the ActiveX object that represents the active drawing
    Dim acAxDoc As AcadDocument = _
        DocumentExtension.GetAcadDocument( _
            Application.DocumentManager.MdiActiveDocument)

    ' Determines whether a valid Database Connection object was created during
    ' the loading of the class
    If Not IsNothing(g_oCAO) Then
        acEditor.WriteMessage(vbLf + "CAO.dbConnect was created; Version = " + _
                              g_oCAO.Version)
    Else
        acEditor.WriteMessage(vbLf + "CAO.dbConnect wasn't created.")
        Exit Sub
    End If

    ' Determines whether the Database Connection object is connected to the data source
    If g_oCAO.IsConnected(g_sDataSource) = False Then
        g_oCAO.Connect(g_sDataSource)
    End If

    ' Lists the label templates stored in the current drawing
    ' Gets the LabelTemplates collection
    g_labelTemplates = g_oCAO.GetLabelTemplates(acAxDoc)
```

```
' Determines whether a valid collection exists
If Not IsNothing(g_labelTemplates) Then

    ' Checks to see if at least one label template exists in the collection
    If g_labelTemplates.Count > 0 Then

        ' Outputs information about each label template
        acEditor.WriteMessage(vbLf + "Label templates stored in drawing:")
        For Each labelTemplate As CAO.LabelTemplate In g_labelTemplates
            acEditor.WriteMessage(vbLf + "  Name:       " + labelTemplate.Name)
            acEditor.WriteMessage(vbLf + "  Catalog:    " + labelTemplate.Catalog)
            acEditor.WriteMessage(vbLf + "  Datasource: " + labelTemplate.Datasource)
            acEditor.WriteMessage(vbLf + "  Schema:     " + labelTemplate.Schema)
            acEditor.WriteMessage(vbLf + "  Table:      " + labelTemplate.Table)

            ' Gets the field names from the table used in the label
            acEditor.WriteMessage(vbLf + "      DisplayableAttributes: ")
            For Each sAttName As String In labelTemplate.DisplayableAttributes
                acEditor.WriteMessage(vbLf + "      - " + sAttName)
            Next

            ' Gets the format for the text of the label
            acEditor.WriteMessage(vbLf + "      Format:           " + _
                                  labelTemplate.Format)
            acEditor.WriteMessage(vbLf + "      LeaderAttachment: " + _
                                  labelTemplate.LeaderAttachment.ToString())

            ' Gets the X,Y,Z for the LeaderOffset
            acEditor.WriteMessage(vbLf + "      LeaderOffset:     ")
            For Each dLdrOffset As Double In labelTemplate.LeaderOffset
                acEditor.WriteMessage(vbLf + "      - " + dLdrOffset.ToString())
            Next

            ' Gets the color and linetype of the leader line
            acEditor.WriteMessage(vbLf + "      Color:            " + _
                                  labelTemplate.Color.ToString())
            acEditor.WriteMessage(vbLf + "      Linetype:         " + _
                                  labelTemplate.Linetype)

            ' Gets the rotation, text attachment type, text height,
            ' and text style used by the label
            acEditor.WriteMessage(vbLf + "      Rotation:         " + _
                                  labelTemplate.Rotation.ToString())
            acEditor.WriteMessage(vbLf + "      TextAttachment:   " + _
                                  CStr(labelTemplate.TextAttachment))
            acEditor.WriteMessage(vbLf + "      TextHeight:       " + _
                                  labelTemplate.TextHeight.ToString())
            acEditor.WriteMessage(vbLf + "      TextStyle:        " + _
                                  labelTemplate.TextStyle)

            ' Gets the X,Y,Z for the TipOffset
            acEditor.WriteMessage(vbLf + "      TipOffset:        ")
            For Each dTipOffset As Double In labelTemplate.TipOffset
                acEditor.WriteMessage(vbLf + "      - " + dTipOffset.ToString())
```

```
            Next

            ' Forces a blank line in the output
            acEditor.WriteMessage(vbLf)
        Next
    Else
        acEditor.WriteMessage(vbLf + "No label templates stored in drawing.")
    End If
End If

' Lists the link templates stored in the current drawing
' Gets the LinkTemplates collection
g_linkTemplates = g_oCAO.GetLinkTemplates(acAxDoc)

' Determines whether a valid collection exists
If Not IsNothing(g_linkTemplates) Then

    ' Checks to see if at least one link template exists in the collection
    If g_linkTemplates.Count > 0 Then

        ' Outputs information about each link template
        acEditor.WriteMessage(vbLf + "Link templates stored in drawing:")
        For Each linkTemplate As CAO.LinkTemplate In g_linkTemplates
            acEditor.WriteMessage(vbLf + "  Name:       " + linkTemplate.Name)
            acEditor.WriteMessage(vbLf + "  Catalog:    " + linkTemplate.Catalog)
            acEditor.WriteMessage(vbLf + "  Datasource: " + linkTemplate.Datasource)
            acEditor.WriteMessage(vbLf + "  Schema:     " + linkTemplate.Schema)
            acEditor.WriteMessage(vbLf + "  Table:      " + linkTemplate.Table)

            ' Gets information about the database fields used to link to an object
            Dim keyDescs As KeyDescriptions = linkTemplate.KeyDescriptions
            For Each keyDesc As KeyDescription In keyDescs
                acEditor.WriteMessage(vbLf + "     DefinedSize:  " + _
                                      keyDesc.DefinedSize.ToString())
                acEditor.WriteMessage(vbLf + "     FieldName:    " + _
                                      keyDesc.FieldName)
                acEditor.WriteMessage(vbLf + "     NumericScale: " + _
                                      keyDesc.NumericScale.ToString())
                acEditor.WriteMessage(vbLf + "     Precision:    " + _
                                      keyDesc.Precision.ToString())
                acEditor.WriteMessage(vbLf + "     Type:         " + _
                                      keyDesc.Type.ToString())
            Next

            ' Forces a blank line in the output
            acEditor.WriteMessage(vbLf)
        Next
    Else
        acEditor.WriteMessage(vbLf + "No link templates stored in drawing.")
    End If
End If

' Lists the queries stored in the current drawing
' Gets the Queries collection
g_queries = g_oCAO.GetQueries(acAxDoc)
```

```
    ' Determines whether a valid collection exists
    If Not IsNothing(g_queries) Then

        ' Checks to see if at least one query exists in the collection
        If g_queries.Count > 0 Then

            ' Outputs information about each query
            acEditor.WriteMessage(vbLf + "Queries stored in drawing:")
            For Each query As CAO.Query In g_queries
                acEditor.WriteMessage(vbLf + "  Name:       " + query.Name)
                acEditor.WriteMessage(vbLf + "  Catalog:    " + query.Catalog)
                acEditor.WriteMessage(vbLf + "  Datasource: " + query.Datasource)
                acEditor.WriteMessage(vbLf + "  Schema:     " + query.Schema)
                acEditor.WriteMessage(vbLf + "  Table:      " + query.Table)
                acEditor.WriteMessage(vbLf + "  Text:       " + query.Text)

                ' Forces a blank line in the output
                acEditor.WriteMessage(vbLf)
            Next
        Else
            acEditor.WriteMessage(vbLf + "No queries stored in drawing.")
        End If
    End If
End Sub
```

The following sample code obtains and then lists information about the links attached to drawing objects with the `GetLinks()` method.

```
' Displays information about the links attached to each object in Model space
<CommandMethod("CAO_ListLinks")> _
Public Sub CAO_ListLinks()
    ' Gets the editor object of the current drawing
    Dim acEditor As Editor = Application.DocumentManager.MdiActiveDocument.Editor

    ' Gets the ActiveX object that represents the active drawing
    Dim acAxDoc As AcadDocument = _
        DocumentExtension.GetAcadDocument( _
            Application.DocumentManager.MdiActiveDocument)

    ' Gets the database of the current drawing
    Dim acCurDb As Database = Application.DocumentManager.MdiActiveDocument.Database

    ' Determines whether a valid Database Connection object was created during
    ' the loading of the class
    If IsNothing(g_oCAO) Then
        acEditor.WriteMessage(vbLf + "CAO.dbConnect wasn't created.")
        Exit Sub
    End If

    ' Determines whether the Database Connection object is connected to the data source
    If g_oCAO.IsConnected(g_sDataSource) = False Then
        g_oCAO.Connect(g_sDataSource)
    End If
```

```
    ' Defines an empty array of longs to be used for object IDs
    Dim lObjIds() As Long = Nothing

    ' Creates a transaction to step through Model space
    Using acTrans As Transaction = acCurDb.TransactionManager.StartTransaction()

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

        ' Opens the Block table record Model space for read
        Dim acBlkTblRec As BlockTableRecord
        acBlkTblRec = acTrans.GetObject(acBlkTbl(BlockTableRecord.ModelSpace), _
                                        OpenMode.ForRead)

        ' Creates a counter variable
        Dim lCount As Long = 0

        ' Steps through each object in Model space
        For Each acObjId As ObjectId In acBlkTblRec

            ' Redimensions the object ID array
            ReDim Preserve lObjIds(0 To lCount)

            ' Adds the object's ID to the array
            lObjIds(lCount) = acObjId.OldIdPtr.ToInt64()

            ' Increments the counter
            lCount = lCount + 1
        Next

        ' Ignore the changes made in the transaction
        acTrans.Abort()
    End Using

    ' Gets the link template named "ProjectsLink"
    Dim linktemplate As CAO.LinkTemplate = _
        g_oCAO.GetLinkTemplates(acAxDoc).Item("ProjectsLink")

    ' Filters out the selected objects that don't match
    ' the link template and specified link types
    g_links = g_oCAO.GetLinks(linktemplate, lObjIds, _
                                            CAO.LinkType.kEntityLinkType + _
                                            CAO.LinkType.kAttachedLabelType + _
                                            CAO.LinkType.kFSLabelType)

    ' Determines whether a valid collection was returned
    If Not IsNothing(g_links) Then

        ' Determines whether there is one or more links that
        ' match the name of the link template
        If g_links.Count > 0 Then

            ' Displays information about the objects in the Links collection
```

```
            acEditor.WriteMessage(vbLf + "Links to objects in the drawing:")
            For Each link As CAO.Link In g_links

                ' Gets the fields which are used to link the object to the database
                Dim keyVals As CAO.KeyValues
                keyVals = link.KeyValues

                ' Displays information about the fields and values
                ' used to link the object to the database
                acEditor.WriteMessage(vbLf + "  Key Values: ")
                For Each keyVal As CAO.KeyValue In keyVals
                    acEditor.WriteMessage(vbLf + "     Field: " + _
                                          keyVal.FieldName)
                    acEditor.WriteMessage(vbLf + "     Value: " + _
                                          keyVal.Value.ToString())
                Next

                ' Determines if a valid label exists, if so output
                ' information about the label
                If Not IsNothing(link.Label) Then
                    acEditor.WriteMessage(vbLf + "  Label: ")
                    acEditor.WriteMessage(vbLf + "     LeaderObjectId:     " + _
                                          link.Label.LeaderObjectId.ToString())
                    acEditor.WriteMessage(vbLf + "     MTextObjectId:      " + _
                                          link.Label.MTextObjectId.ToString())
                    acEditor.WriteMessage(vbLf + "     LabelTemplate Name: " + _
                                          link.Label.LabelTemplate.Name)

                    ' Displays the database fields used in the label
                    acEditor.WriteMessage(vbLf + "     Display Values: ")
                    For Each dispVal As String In link.Label.DisplayValues
                        acEditor.WriteMessage(vbLf + "          " + dispVal)
                    Next
                End If

                ' Displays information about the link template and the object linked
                acEditor.WriteMessage(vbLf + "  LinkTemplate Name: " + _
                                      link.LinkTemplate.Name)
                acEditor.WriteMessage(vbLf + "  LinkType:          " + _
                                      link.LinkType.ToString())
                acEditor.WriteMessage(vbLf + "  ObjectID:          " + _
                                      link.ObjectID.ToString())
                acEditor.WriteMessage(vbLf + "  Updatable:         " + _
                                      link.Updatable.ToString())

                ' Forces a blank line in the output
                acEditor.WriteMessage(vbLf)
            Next
        Else
            acEditor.WriteMessage(vbLf + "No links to objects in the drawing.")
        End If
    End If
End Sub
```

Getting Notified of Changes

The Connectivity Automation Object (CAO) library allows you to monitor changes to the objects that can be created and modified using the methods of the CAO library through the use of event handlers. The following events are members of the Database Connectivity (`DbConnect`) object:

- **`OnConnect`** – Triggered when a connection to a data source is made
- **`OnDisconnect`** – Triggered when a connection to a data source is terminated
- **`OnLabelTemplateCreated`** – Triggered when a label template is added to a drawing
- **`OnLabelTemplateDeleted`** – Triggered when a label template is removed from a drawing
- **`OnLabelTemplateModified`** – Triggered when a label template is modified
- **`OnLinkTemplateCreated`** – Triggered when a link template is added to a drawing
- **`OnLinkTemplateDeleted`** – Triggered when a link template is removed from a drawing
- **`OnLinkTemplateModified`** – Triggered when a link template is modified
- **`OnQueryCreated`** – Triggered when a query is added to a drawing
- **`OnQueryDeleted`** – Triggered when a query is removed from a drawing
- **`OnQueryModified`** – Triggered when a query is modified

You can determine when a link or label is added to a drawing by monitoring for changes using the following database events and then evaluating the objects that are created or modified:

- **`ObjectAppended`** – Triggered when an object is added to a database
- **`ObjectErased`** – Triggered when an object is removed from a database
- **`ObjectModified`** – Triggered when an object is modified

5 Managing Sheet Sets

Sheet sets were first introduced with the AutoCAD 2005 program and are used to organize multiple drawing files into a virtual set of papers. A sheet set is primarily made up of sheet components; a sheet component is a reference to a named layout defined in a drawing file. Since a sheet represents a named layout, you can access the layout to create individual viewports or place a sheet view on the layout. A sheet view is a special component that is made up of a viewport and named view.

When a sheet view is placed on a sheet, the drawing that contains the objects and named view is referenced to the drawing containing the sheet as an exteral reference (xref). After the drawing is referenced, a viewport is created that fits the size of the objects in the named view and at a specific scale. Users manage the components in or referenced by a sheet set file with the Sheet Set Manager from the AutoCAD user interface.

The Sheet Set Object (SSO) library allows you to perform many of the tasks available in the Sheet Set Manager with the exception of publishing, transmitting, and archiving the drawing files referenced in a sheet set file. You can use the Sheet Set Object library to:

- Work with sheet set (DST) files open in the Sheet Set Manager
- Create a sheet set from scratch or based on a template DST file
- Populate properties of a sheet set or sheet from an external data source
- Access objects in drawings referenced by a sheet set

Note: You can publish the drawings in a sheet set with the Managed .NET Publishing API and replicate the transmittal and archival features with the Transmittal Object (ATO) library mentioned in this session.

What You Need Before Getting Started

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

- **Sheet Set Object Library (*AcSmComponents20.tlb*)** – File is part of the ObjectARX SDK. You can download the ObjectARX SDK from *http://www.autodesk.com/objectarx*. Click License & Download and then follow the onscreen instructions
- **Sheet Set Object Documentation** – The Developer Home page in the AutoCAD Online Help system contains the reference and concept topics related to the Sheet Set Object library. On the Developer's Home Page, click the Sheet Set Object (SSO) Reference link. To access the online help, go to: *http://help.autodesk.com/view/ACD/2016/ENU/files/homepage_dev.htm*.
- **AutoCAD ActiveX Documentation** – The documentation for the AutoCAD Type Library is installed with the AutoCAD program. You can access the AutoCAD ActiveX Reference and AutoCAD ActiveX Developers guides by browsing to *C:\Program Files\Common Files\Autodesk Shared* and opening the files *acadauto.chm* and *acad_aag.chm*. The topics can also be found in the AutoCAD Online Help system, on the Developer's Home Page, click the AutoCAD ActiveX Library Reference and AutoCAD ActiveX Developer's Guide links.

Taking a Look at the Sheet Set Object Library

The first thing that you need to know is how to reference the Sheet Set Object library. The Sheet Set Object library is stored in a file named *AcSmComponents<version>.tlb* under the Autodesk Shared subfolder in the *Program Files\Common Files* folder; in the Reference Manager it is listed as Autodesk Standards Manager Type Library. When working with the Sheet Set Object library, you need to know the version of the library required based on the release of the AutoCAD program in which your program will interact.

The following table lists the file name and version of the ActiveX libraries you need to reference by AutoCAD release.

AutoCAD Release	Sheet Set Object Library	AutoCAD/ObjectDBX Common Type Library
AutoCAD 2016 and AutoCAD 2015	*acsmcomponents20.tlb*	*axdb20<language>.tlb*
AutoCAD 2014 and AutoCAD 2013	*acsmcomponents19.tlb*	*axdb19<language>.tlb*
AutoCAD 2012, AutoCAD 2011, and AutoCAD 2010	*acsmcomponents18.tlb*	*axdb18<language>.tlb*
AutoCAD 2009, AutoCAD 2008, and AutoCAD 2007	*acsmcomponents17.tlb*	*axdb17<language>.tlb*

Once you have referenced the library, you need to reference the namespace that contains the Sheet Set Manager object. The namespace of the library is similar to the name of the referenced TLB file. The following statement imports the Sheet Set Object library for AutoCAD 2016 or AutoCAD 2015 in a VB.NET code module:

```
Imports ACSMCOMPONENTS20Lib
```

After the Sheet Set Object library namespace has been imported, you must create a reference to the Sheet Set Manager object which is represented by the `IAcSmSheetSetMgr` interface. The following code demonstrates how to reference the Sheet Set Manager object:

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

With the Sheet Set Manager object, you can

- Access any of the sheet set files open in the Sheet Set Manager (`GetDatabaseEnumerator`)
- Create a new sheet set file (`CreateDatabase`)
- Open a sheet set file (`OpenDatabase`)
- Close a sheet set file (`Close`)

Below are some code examples that demonstrate the common methods used for working with the Sheet Set Manager object and sheet sets.

`OpenDatabase()` – Opens an existing sheet set (DST) file.

`Close()` – Closes an open sheet set (DST) file.

```
' Open a Sheet Set
<CommandMethod("SSO_OpenSheetSet")> _
Public Sub SSO_OpenSheetSet()
    ' Get a reference to the Sheet Set Manager object
    Dim sheetSetManager As IAcSmSheetSetMgr
    sheetSetManager = New AcSmSheetSetMgr

    ' Open a Sheet Set file
    Dim sheetSetDatabase As AcSmDatabase
    sheetSetDatabase = sheetSetManager.OpenDatabase(c_SampleFolder & _
                          "Sheet Sets\Architectural\IRD Addition.dst", False)

    ' Return the name and description of the sheet set
    MsgBox("Sheet Set Name: " + sheetSetDatabase.GetSheetSet().GetName() & vbCrLf + _
           "Sheet Set Description: " + sheetSetDatabase.GetSheetSet().GetDesc())

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

`CreateDatabase()` – Creates a new sheet set file from scratch or based on an existing DST file as a template.

```
' Create a new sheet set
<CommandMethod("SSO_CreateSheetSet")> _
Public Sub SSO_CreateSheetSet()
    ' Get a reference to the Sheet Set Manager object
    Dim sheetSetManager As IAcSmSheetSetMgr
    sheetSetManager = New AcSmSheetSetMgr

    ' Create a new sheet set file
    Dim sheetSetDatabase As AcSmDatabase
    sheetSetDatabase = sheetSetManager.CreateDatabase(c_DatasetFolder & c_DSTFile, _
                                                      "", True)
```

```
    ' Get the sheet set from the database
    Dim sheetSet As AcSmSheetSet
    sheetSet = sheetSetDatabase.GetSheetSet()

    ' Set the name and description of the sheet set
    ' Next line causes an expected exception
    sheetSet.SetName("IT10489")
    sheetSet.SetDesc("AU2015 Sheet Set Object Demo")

    ' Return the name and description of the sheet set
    MsgBox("Sheet Set Name: " + sheetSetDatabase.GetSheetSet().GetName() + vbCrLf + _
           "Sheet Set Description: " + sheetSetDatabase.GetSheetSet().GetDesc())

    ' Close the sheet set
    sheetSetManager.Close(sheetSetDatabase)

    ' Open the sheet set in the Sheet set Manager
    Application.DocumentManager.MdiActiveDocument. _
        Editor.Command("-opensheetset", c_DatasetFolder & c_DSTFile)
End Sub
```

While the above example code shows the logic of creating a new sheet set and changing its properties, an exception occurs when it is executed. The exception occurs because a sheet set must be locked before it can be modified. The reason behind it needing to be locked is because more than one user could be working with the sheet set file at the same time. So prior to changing the Name and Description properties, the `LockDb()` method must be called and then `UnlockDb()` must be called to unlock the database once all modifications are made.

The `GetLockStatus` method is used to return the current lock status of a sheet set database. The returned lock status will be in the form of one of the following constants:

`AcSmLockStatus_UnLocked`	Sheet set is not locked, write access denied
`AcSmLockStatus_Locked_Local`	Sheet set is locked, write access enabled
`AcSmLockStatus_Locked_Remote`	Sheet set is locked by another user, write access denied
`AcSmLockStatus_Locked_ReadOnly`	Sheet set is read-only, write access denied
`AcSmLockStatus_Locked_AccessDenied`	Write access denied due to lack of user rights
`AcSmLockStatus_Locked_NotConnected`	Connection sheet set was lost, write access enabled

Note: The top four statuses are the most commonly encountered out of all possible values.

Below is a custom function that wraps the functionality of locking and unlocking a sheet set database before modifying any of its properties or content.

```
' Used to lock/unlock a sheet set database
Private Function LockDatabase(ByRef database As AcSmDatabase, _
                              ByVal lockFlag As Boolean) As Boolean
    Dim dbLock As Boolean = False
```

```
    ' If lockFlag equals True then attempt to lock the database, otherwise
    ' attempt to unlock it.
    If lockFlag = True And _
        database.GetLockStatus() = AcSmLockStatus.AcSmLockStatus_UnLocked Then
        database.LockDb(database)
        dbLock = True
    ElseIf lockFlag = False And _
        database.GetLockStatus = AcSmLockStatus.AcSmLockStatus_Locked_Local Then
        database.UnlockDb(database)
        dbLock = True
    Else
        dbLock = False
    End If

    LockDatabase = dbLock
End Function
```

To correct the problem with the `SSO_CreateSheetSet()` method; you would add the `LockDatabase()` function with the database to be locked and the Boolean value of `True` before calling the `SetName()` and `SetDesc()` methods. After modifying the properties, you would make a second call to the `LockDatabase()` method with the Boolean value of `False` to unlock the database. The code changes might look similar to the following:

```
…
    sheetSet = sheetSetDatabase.GetSheetSet()

    ' Attempt to lock the database
    If LockDatabase(sheetSetDatabase, True) = True Then
        ' Set the name and description of the sheet set
        sheetSet.SetName("IT10489")
        sheetSet.SetDesc("AU2015 Sheet Set Object Demo")

        ' Unlock the database
        LockDatabase(sheetSetDatabase, False)

        ' Return the name and description of the sheet set
        MsgBox("Sheet Set Name: " + sheetSetDatabase.GetSheetSet().GetName() & vbCrLf + _
               "Sheet Set Description: " + sheetSetDatabase.GetSheetSet().GetDesc())
    Else
        ' Display error message
        MsgBox("Sheet set could not be opened for write.")
    End If

    ' Close the sheet set
…
```

`GetDatabaseEnumerator()` – Allows you to step through all open sheet set files in the current AutoCAD session.

```
' Step through all open sheet sets
<CommandMethod("SSO_StepThroughTheOpenSheetSets")> _
Public Sub SSO_StepThroughTheOpenSheetSets()
    ' Get a reference to the Sheet Set Manager object
    Dim sheetSetManager As IAcSmSheetSetMgr
    sheetSetManager = New AcSmSheetSetMgr
```

```
    ' Get the loaded databases
    Dim enumDatabase As IAcSmEnumDatabase
    enumDatabase = sheetSetManager.GetDatabaseEnumerator()

    ' Get the first open database
    Dim item As IAcSmPersist
    item = enumDatabase.Next()

    Dim customMessage As String = ""

    ' If a database is open continue
    If Not item Is Nothing Then
        Dim count As Integer = 0

        ' Step through the database enumerator
        Do While Not item Is Nothing
            ' Append the file name of the open sheet set to the output string
            customMessage = customMessage + vbLf + _
                            item.GetDatabase().GetFileName()

            ' Get the next open database and increment the counter
            item = enumDatabase.Next()
            count = count + 1
        Loop

        customMessage = "Sheet sets open: " + count.ToString() + _
                        customMessage
    Else
        customMessage = "No sheet sets are currently open."
    End If

    ' Display the custom message
    MsgBox(customMessage)
End Sub
```

Adding Content to the Sheet Set File

It is great to be able to open a sheet set file or even create a new one from scratch, but the purpose of a sheet set is to help organize the drawing files that make up a project. Sheet sets use subsets to help organize the sheets/drawing layouts added to a sheet set file. A subset can represent a physical folder on a network or local drive, or a virtual folder that can be used to just organize the sheets in a sheet set. A subset is represented by the `AcSmSubset` object. A subset has the following main properties:

- Name (`SetName` and `GetName`)
- Description (`SetDesc` and `GetDesc`)
- Prompt for Template (`SetPromptForDWT` and `GetPromptForDWT`)
- Storage Location for New Sheets (`SetNewSheetLocation` and `GetNewSheetLocation`)
- Default Template for New Sheets (`SetDefDwtLayout` and `GetDefDwtLayout`)

The following custom method demonstrates how to create a new subset and set its properties.

```
' Used to add a subset to a sheet set
Private Function CreateSubset(ByVal sheetSetDatabase As AcSmDatabase, _
```

```
                                ByVal name As String, _
                                ByVal description As String, _
                                Optional ByVal newSheetLocation As String = "", _
                                Optional ByVal newSheetDWTLocation As String = "", _
                                Optional ByVal newSheetDWTLayout As String = "", _
                                Optional ByVal promptForDWT As Boolean = False) _
                                As AcSmSubset

    ' Create a subset with the provided name and description
    Dim subset As AcSmSubset = sheetSetDatabase.GetSheetSet(). _
                                    CreateSubset(name, description)

    ' Get the folder the sheet set is stored in
    Dim sheetSetFolder As String
    sheetSetFolder = Mid(sheetSetDatabase.GetFileName(), _
                         1, InStrRev(sheetSetDatabase.GetFileName(), "\"))

    ' Create a reference to a File Reference object
    Dim fileReference As IAcSmFileReference
    fileReference = subset.GetNewSheetLocation()

    ' Check to see if a path was provided, if not default
    ' to the location of the sheet set
    If newSheetLocation <> "" Then
        fileReference.SetFileName(newSheetLocation)
    Else
        fileReference.SetFileName(sheetSetFolder)
    End If

    ' Set the location for new sheets added to the subset
    subset.SetNewSheetLocation(fileReference)

    ' Create a reference to a Layout Reference object
    Dim layoutReference As AcSmAcDbLayoutReference
    layoutReference = subset.GetDefDwtLayout

    ' Check to see that a default DWT location and name was provided
    If newSheetDWTLocation <> "" Then
        ' Set the template location and name of the layout
        'for the Layout Reference object
        layoutReference.SetFileName(newSheetDWTLocation)
        layoutReference.SetName(newSheetDWTLayout)

        ' Set the Layout Reference for the subset
        subset.SetDefDwtLayout(layoutReference)
    End If

    ' Set the Prompt for Template option of the subset
    subset.SetPromptForDwt(promptForDWT)

    CreateSubset = subset
End Function
```

The following code example shows how the `CreateSubset()` method can be used to add a subset in the `SSO_CreateSheetSet()` method.

```
…
    sheetSet.SetName("IT10489")
    sheetSet.SetDesc("AU2015 Sheet Set Object Demo")

    ' Create two new subsets
    Dim subset As AcSmSubset
    subset = CreateSubset(sheetSetDatabase, "Plans", "Building Plans", "", _
                          c_DatasetFolder & c_DWTFile, "Sheet", False)

    subset = CreateSubset(sheetSetDatabase, "Elevations", "Building Elevations", "", _
                          c_DatasetFolder & c_DWTFile, "Sheet", True)

    ' Unlock the database
    LockDatabase(sheetSetDatabase, False)
…
```

Now that there are some subsets in the sheet set, you can add sheets to them. You have already seen part of the functionality that is required to add a sheet. This functionality was shown in the creation of a subset. Sheets can exist both as part of a subset or outside of a subset. A sheet in a sheet set is represented by the `AcSmSheet` object. A sheet has the following main properties:

- Name (`SetName` and `GetName`)
- Description (`SetDesc` and `GetDesc`)
- Title (`SetTitle` and `GetTitle`)
- Number (`SetNumber` and `GetNumber`)

The following method demonstrates how to add a new sheet to a sheet set or subset.

```
' Used to add a sheet to a sheet set or subset
' Note: This function is dependent on a Default Template and Storage location
' being set for the sheet set or subset.
Private Function AddSheet(ByVal component As IAcSmComponent, _
                          ByVal name As String, _
                          ByVal description As String, _
                          ByVal title As String, _
                          ByVal number As String) As AcSmSheet

    Dim sheet As AcSmSheet

    ' Check to see if the component is a sheet set or subset,
    ' and create the new sheet based on the component's type
    If component.GetTypeName = "AcSmSubset" Then
        Dim subset As AcSmSubset = component
        sheet = subset.AddNewSheet(name, description)

        ' Add the sheet as the first one in the subset
        subset.InsertComponent(sheet, Nothing)
    Else
        sheet = component.GetDatabase().GetSheetSet().AddNewSheet(name, _
                                                                  description)

        ' Add the sheet as the first one in the sheet set
        component.GetDatabase().GetSheetSet().InsertComponent(sheet, Nothing)
    End If
```

```
    ' Set the number and title of the sheet
    sheet.SetNumber(number)
    sheet.SetTitle(title)

    AddSheet = sheet
End Function
```

To place the new sheet in the sheet set you use the `InsertComponent` or `InsertComponentAfter` methods.

The `SSO_CreateSheetSet()` method does not set a default drawing template or storage location. The process of changing these properties is similar to those used in the `CreateSubset()` method. The following code shows how to set the default template and storage location for a sheet set.

```
' Set the default properties of a sheet set
Private Sub SetSheetSetDefaults(ByVal sheetSetDatabase As AcSmDatabase, _
                                ByVal name As String, _
                                ByVal description As String, _
                                Optional ByVal newSheetLocation As String = "", _
                                Optional ByVal newSheetDWTLocation As String = "", _
                                Optional ByVal newSheetDWTLayout As String = "", _
                                Optional ByVal promptForDWT As Boolean = False)

    ' Set the Name and Description for the sheet set
    sheetSetDatabase.GetSheetSet().SetName(name)
    sheetSetDatabase.GetSheetSet().SetDesc(description)

    ' Check to see if a Storage Location was provided
    If newSheetLocation <> "" Then
        ' Get the folder the sheet set is stored in
        Dim sheetSetFolder As String
        sheetSetFolder = Mid(sheetSetDatabase.GetFileName(), 1, _
                             InStrRev(sheetSetDatabase.GetFileName(), "\"))

        ' Create a reference to a File Reference object
        Dim fileReference As IAcSmFileReference
        fileReference = sheetSetDatabase.GetSheetSet().GetNewSheetLocation()

        ' Set the default storage location based on the location of the sheet set
        fileReference.SetFileName(sheetSetFolder)

        ' Set the new Sheet location for the sheet set
        sheetSetDatabase.GetSheetSet().SetNewSheetLocation(fileReference)
    End If

    ' Check to see if a Template was provided
    If newSheetDWTLocation <> "" Then
        ' Set the Default Template for the sheet set
        Dim layoutReference As AcSmAcDbLayoutReference
        layoutReference = sheetSetDatabase.GetSheetSet().GetDefDwtLayout()

        ' Set the template location and name of the layout
        ' for the Layout Reference object
        layoutReference.SetFileName(newSheetDWTLocation)
```

```
        layoutReference.SetName(newSheetDWTLayout)

        ' Set the Layout Reference for the sheet set
        sheetSetDatabase.GetSheetSet().SetDefDwtLayout(layoutReference)
    End If

    ' Set the Prompt for Template option of the subset
    sheetSetDatabase.GetSheetSet().SetPromptForDwt(promptForDWT)
End Sub
```

The following code shows the use of the `AddSheet()` and `SetSheetSetDefaults()` methods.

```
…
    ' Attempt to lock the database
    If LockDatabase(sheetSetDatabase, True) = True Then
        ' Get the folder the sheet set is stored in
        Dim sheetSetFolder As String
        sheetSetFolder = Mid(sheetSetDatabase.GetFileName(), 1, _
                         InStrRev(sheetSetDatabase.GetFileName(), "\"))

        ' set the default values of the sheet set
        SetSheetSetDefaults(sheetSetDatabase, _
                            "IT10489", "AU2015 Sheet Set Object Demo", _
                            sheetSetFolder, _
                            c_DatasetFolder & c_DWTFile, "Sheet")

        AddSheet(sheetSetDatabase, "Title Page", "Project Title Page", _
                 "Title Page", "T1")

        ' Create two new subsets
        Dim subset As AcSmSubset
        subset = CreateSubset(sheetSetDatabase, "Plans", "Building Plans", "", _
                              c_DatasetFolder & c_DWTFile, "Sheet", False)

        AddSheet(subset, "North Plan", "Northern section of building plan", _
                 "North Plan", "P1")

        subset = CreateSubset(sheetSetDatabase, "Elevations", "Building Elevations", _
                              "", c_DatasetFolder & c_DWTFile, "Sheet", True)

        ' Unlock the database
        LockDatabase(sheetSetDatabase, False)
    Else
…
```

It is possible to import an existing layout into a sheet set. The process of importing a sheet versus adding a new sheet is not very different. To import a sheet you can use the `ImportSheet` or `AddSheet` method. The following method demonstrates how to import an existing layout from a drawing as a sheet.

```
' Import a sheet into a sheet set or subset
Private Function ImportASheet(ByVal component As IAcSmComponent, _
                              ByVal title As String, _
                              ByVal description As String, _
                              ByVal number As String, _
```

```
                                ByVal fileName As String, _
                                ByVal layout As String) As AcSmSheet

    Dim sheet As AcSmSheet

    ' Create a reference to a Layout Reference object
    Dim layoutReference As New AcSmAcDbLayoutReference
    layoutReference.InitNew(component)

    ' Set the layout and drawing file to use for the sheet
    layoutReference.SetFileName(fileName)
    layoutReference.SetName(layout)

    ' Import the sheet into the sheet set
    ' Check to see if the Component is a Subset or Sheet Set
    If component.GetTypeName = "AcSmSubset" Then
        Dim subset As AcSmSubset = component

        sheet = subset.ImportSheet(layoutReference)
        subset.InsertComponent(sheet, Nothing)
    Else
        Dim sheetSetDatabase As AcSmDatabase = component

        sheet = sheetSetDatabase.GetSheetSet().ImportSheet(layoutReference)
        sheetSetDatabase.GetSheetSet().InsertComponent(sheet, Nothing)
    End If

    ' Set the properties of the sheet
    sheet.SetDesc(description)
    sheet.SetTitle(title)
    sheet.SetNumber(number)

    ImportASheet = sheet
End Function
```

Adding Sheet Set and Sheet Properties

Custom properties are a great way to ensure consistency among the values in title blocks for a project, as well as being able to track project status. There are two types of custom properties: sheet and sheet set. To create a custom property you need to create a reference to an `AcSmCustomPropertyBag` object. The `AcSmCustomPropertyBag` object is used as a container to hold any custom properties that are added to a sheet or sheet set. Once a reference to an `AcSmCustomPropertyBag` object is made, you can then create a reference to an `AcSmCustomPropertyValue` object; which is the actual custom property that you want to create.

You use a constant to determine the type of property that will be created. The values that use are:

- CUSTOM_SHEET_PROP – Sheet property
- CUSTOM_SHEETSET_PROP – Sheet set property

The following method demonstrates how to add a custom property to a sheet or sheet set.

Note: It is best to create your properties before adding any sheets programmatically to a sheet set so the sheets that are added inherit the properties from the sheet set, otherwise you will need to step through each sheet in the sheet set and add the custom property to each sheet one at a time.

```
' Set/create a custom sheet or sheet set property
Private Sub SetCustomProperty(ByVal owner As IAcSmPersist, _
                              ByVal propertyName As String, _
                              ByVal propertyValue As Object, _
                              ByVal sheetSetFlag As PropertyFlags)

    ' Create a reference to the Custom Property Bag
    Dim customPropertyBag As AcSmCustomPropertyBag

    If owner.GetTypeName() = "AcSmSheet" Then
        Dim sheet As AcSmSheet = owner
        customPropertyBag = sheet.GetCustomPropertyBag()
    Else
        Dim sheetSet As AcSmSheetSet = owner
        customPropertyBag = sheetSet.GetCustomPropertyBag()
    End If

    ' Create a reference to a Custom Property Value
    Dim customPropertyValue As AcSmCustomPropertyValue = New AcSmCustomPropertyValue()
    customPropertyValue.InitNew(owner)

    ' Set the flag for the property
    customPropertyValue.SetFlags(sheetSetFlag)

    ' Set the value for the property
    customPropertyValue.SetValue(propertyValue)

    ' Create the property
    customPropertyBag.SetProperty(propertyName, customPropertyValue)
End Sub
```

The following code shows the use of the `SetCustomProperty()` method.

```
…
    ' Set the default values of the sheet set
    SetSheetSetDefaults(sheetSetDatabase, _
                        "IT10489", "AU2015 Sheet Set Object Demo", _
                        sheetSetFolder, _
                        c_DatasetFolder & c_DWTFile, "Sheet")

    ' Create a sheet set property
    SetCustomProperty(sheetSet, "Project Approved By", "AU15", _
                      PropertyFlags.CUSTOM_SHEETSET_PROP)

    ' Create sheet properties
    SetCustomProperty(sheetSet, "Checked By", "LAA", _
                      PropertyFlags.CUSTOM_SHEET_PROP)

    SetCustomProperty(sheetSet, "Complete Percentage", "0%", _
                      PropertyFlags.CUSTOM_SHEET_PROP)
```

```
    AddSheet(sheetSetDatabase, "Title Page", "Project Title Page", _
             "Title Page", "T1")
…
```

As previously mentioned, if you create a new sheet property and you have existing sheets in the sheet set it becomes your responsibility to add the property to each sheet in the sheet set. To add the property to all the sheets in the sheet set you need to step through each of the sheets using a loop. To get all the objects that are contained in the sheet set you can use the `GetEnumerator()` method and then use the `GetTypeName()` method to determine which type of object is returned.

The following method demonstrates how to step through all the properties associated with a sheet set and make sure the ones that are flagged as a sheet property are added to each sheet in the sheet set.

```
' Synchronize the properties of a sheet with the sheet set
Private Sub SyncProperties(ByVal sheetSetDatabase As IAcSmDatabase)
    ' Get the objects in the sheet set
    Dim enumerator As IAcSmEnumPersist
    enumerator = sheetSetDatabase.GetEnumerator()

    ' Get the first object in the Enumerator
    Dim item As IAcSmPersist
    item = enumerator.Next()

    ' Step through all the objects in the sheet set
    Do While Not item Is Nothing
        Dim sheet As IAcSmSheet = Nothing

        ' Check to see if the object is a sheet
        If item.GetTypeName() = "AcSmSheet" Then
            sheet = item

            ' Create a reference to the Property Enumerator for
            ' the Custom Property Bag
            Dim enumeratorProperty As IAcSmEnumProperty
            enumeratorProperty = item.GetDatabase().GetSheetSet(). _
            GetCustomPropertyBag().GetPropertyEnumerator()

            ' Get the values from the Sheet Set to populate to the sheets
            Dim name As String = ""
            Dim customPropertyValue As AcSmCustomPropertyValue = Nothing

            ' Get the first property
            enumeratorProperty.Next(name, customPropertyValue)

            ' Step through each of the properties
            Do While Not customPropertyValue Is Nothing
                ' Check to see if the property is for a sheet
                If customPropertyValue.GetFlags() = _
                   PropertyFlags.CUSTOM_SHEET_PROP Then

                    SetCustomProperty(sheet, name, _
                                      customPropertyValue.GetValue(), _
                                      customPropertyValue.GetFlags())
                End If
```

```
                ' Get the next property
                enumeratorProperty.Next(name, customPropertyValue)
            Loop
        End If

        ' Get the next Sheet
        item = enumerator.Next()
    Loop
End Sub
```

Working with Sheet Set Events

The events available for the Sheet Set object must be registered and then unregistered when they are no longer needed. The Sheet Set object utilizes a single event handler which is designed to return a number of different constant values. Below is basic implementation of a custom class that implements the events for the Sheet Set object and two methods used to register and unregister the event handler.

```
' Custom event handler class
Public Class MySSmEventHandler
    Implements IAcSmEvents

    Private Sub IAcSmEvents_OnChanged(ByVal ev As AcSmEvent, _
                                      ByVal comp As IAcSmPersist) _
                                      Implements IAcSmEvents.OnChanged

        Dim activeDocument As Autodesk.AutoCAD.ApplicationServices.Document
        activeDocument = Autodesk.AutoCAD.ApplicationServices.Application. _
                         DocumentManager.MdiActiveDocument()

        Dim oSheet As AcSmSheet
        Dim oSubset As AcSmSubset

        If ev = AcSmEvent.ACSM_DATABASE_OPENED Then
            activeDocument.Editor.WriteMessage(vbLf & comp.GetDatabase(). _
                                               GetFileName() & " was opened.")
        End If
        If ev = AcSmEvent.ACSM_DATABASE_CHANGED Then
            activeDocument.Editor.WriteMessage(vbLf & comp.GetDatabase(). _
                                               GetFileName() & " database changed.")
        End If
        If ev = AcSmEvent.SHEET_DELETED Then
            oSheet = comp
            activeDocument.Editor.WriteMessage(vbLf & oSheet.GetName() & _
                                               " was deleted.")
        End If
        If ev = AcSmEvent.SHEET_SUBSET_CREATED Then
            oSubset = comp
            activeDocument.Editor.WriteMessage(vbLf & oSubset.GetName() & _
                                               " was created.")
        End If
        If ev = AcSmEvent.SHEET_SUBSET_DELETED Then
            oSubset = comp
            activeDocument.Editor.WriteMessage(vbLf & oSubset.GetName() & _
                                               " was deleted.")
        End If
```

```
    End Sub
End Class

' Register the event handlers
<CommandMethod("SSO_AddSSmEvents")> _
Public Sub SSO_AddSSmEvents()

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

    Try
        ' Open a Sheet Set file
        Dim sheetSetDatabase As IAcSmDatabase
        sheetSetDatabase = sheetSetManager. _
                           FindOpenDatabase(c_DatasetFolder & c_DSTFile)

        ' Get the sheet set from the database
        Dim sheetSet As IAcSmSheetSet
        sheetSet = sheetSetDatabase.GetSheetSet()

        AddEvents(sheetSetManager, sheetSetDatabase, sheetSet)
    Catch ex As Autodesk.AutoCAD.Runtime.Exception
        MsgBox(ex.Message, MsgBoxStyle.Critical, "Error")
    Catch ex As System.Exception
        MsgBox(ex.Message, MsgBoxStyle.Critical, "Error")
    End Try
End Sub

' Remove the registered event handlers
<CommandMethod("SSO_RemoveSSmEvents")> _
Public Sub SSO_RemoveSSmEvents()
    RemoveEvents()
End Sub

' Register event handlers with the Sheet Set Manager,
' database, and sheet set
Private Sub AddEvents(ByVal sheetSetManager As IAcSmSheetSetMgr, _
                      ByVal sheetSetDatabase As IAcSmDatabase, _
                      ByVal sheetSet As IAcSmSheetSet)
    On Error Resume Next
    If Not eventHandler Is Nothing Then
        Exit Sub
    End If

    eventHandler = New MySSmEventHandler

    m_sheetSetManager = sheetSetManager
    m_sheetSetDatabase = sheetSetDatabase
    m_sheetSet = sheetSet

    ' Register an event for the Sheet Set Manager
    eventSSMCookie = m_sheetSetManager.Register(eventHandler)

    ' Register a database event
```

```
    eventDbCookie = m_sheetSetDatabase.Register(eventHandler)

    ' Register for sheet set event
    eventSSetCookie = m_sheetSet.Register(eventHandler)
End Sub

' Remove the registered event handlers for the Sheet Set Manager,
' database, and sheet set
Private Sub RemoveEvents()
    On Error Resume Next
    If eventHandler Is Nothing Then
        Exit Sub
    End If

    m_sheetSetManager.Unregister(eventSSMCookie)
    m_sheetSetDatabase.Unregister(eventDbCookie)
    m_sheetSet.Unregister(eventSSetCookie)

    eventHandler = Nothing
End Sub
```

Additional Custom Procedures

The following methods demonstrate how to manipulate other features of the Sheet Set Object (SSO) library, and even how to create a custom sheet set property that holds the total number of sheets in the sheet set.

```
' Counts up the sheets for all the open sheet sets
<CommandMethod("SSO_SetSheetCount")> _
Public Sub SSO_SetSheetCount()
    Dim nSheetCount As Integer = 0

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

    ' Get the loaded databases
    Dim enumDatabase As IAcSmEnumDatabase
    enumDatabase = sheetSetManager.GetDatabaseEnumerator()

    ' Get the first open database
    Dim item As IAcSmPersist
    item = enumDatabase.Next()

    Dim sheetSetDatabase As AcSmDatabase

    Do While Not item Is Nothing
        sheetSetDatabase = item

        ' Attempt to lock the database
        If LockDatabase(sheetSetDatabase, True) = True Then

            On Error Resume Next

            Dim enumerator As IAcSmEnumPersist
            Dim itemSheetSet As IAcSmPersist
```

```
            ' 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
                    nSheetCount = nSheetCount + 1
                End If

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

            ' Create a sheet set property
            SetCustomProperty(sheetSetDatabase.GetSheetSet(), _
                              "Total Sheets", CStr(nSheetCount), _
                              PropertyFlags.CUSTOM_SHEETSET_PROP)

            ' Unlock the database
            LockDatabase(sheetSetDatabase, False)

            ' Clear and check for the next open sheet set
            nSheetCount = 0
        Else
            MsgBox("Unable to access " & sheetSetDatabase.GetSheetSet().GetName())
        End If

        item = enumDatabase.Next
    Loop

    ' Open the sheet set in the Sheet set Manager
    Application.DocumentManager.MdiActiveDocument. _
        Editor.Command("-opensheetset", c_DatasetFolder & c_DSTFile)
 End Sub
```

The following method demonstrates how to define a sheet selection set.

```
' Add a sheet selection set
Function AddSheetSelectionSet(ByVal sheetSetDatabase As AcSmDatabase, _
                              ByVal name As String, _
                              ByVal description As String) As IAcSmSheetSelSet

    ' Get the sheet selection sets for the sheet set
    Dim sheetSelSets As IAcSmSheetSelSets
    sheetSelSets = sheetSetDatabase.GetSheetSet().GetSheetSelSets()

    ' Add the new sheet selection set
    Dim sheetSelSet As IAcSmSheetSelSet = Nothing
    sheetSelSets.Add(name, description, sheetSelSet)

    ' Return the selection set added
    AddSheetSelectionSet = sheetSelSet
End Function
```

The following method demonstrates how to setup a resource location.

```
' Add a drawing resource location to the sheet set
Public Sub AddResourceFileLocation(ByVal sheetSetDatabase As AcSmDatabase, _
                                   ByVal resourceLoc As String)
    Dim resources As IAcSmResources
    Dim fileReference As New AcSmFileReference

    ' Create a reference to a resource location
    resources = sheetSetDatabase.GetSheetSet().GetResources()
    fileReference.InitNew(sheetSetDatabase)
    fileReference.SetFileName(resourceLoc)

    ' Add the drawing resource location
    resources.Add(fileReference)
End Sub
```

The following method demonstrates how to setup a callout block.

```
' Add a callout block to the sheet set
Public Sub AddCalloutBlock(ByVal sheetSetDatabase As AcSmDatabase, _
                           ByVal blockName As String, _
                           ByVal drawingFile As String)

    Dim calloutBlocks As IAcSmCalloutBlocks
    Dim calloutBlockReference As New AcSmAcDbBlockRecordReference

    ' Get the callout blocks for the sheet set
    calloutBlocks = sheetSetDatabase.GetSheetSet().GetCalloutBlocks()

    ' Create a new reference to a callout block and define the block to use
    calloutBlockReference.InitNew(sheetSetDatabase)
    calloutBlockReference.SetFileName(drawingFile)
    calloutBlockReference.SetName(blockName)

    ' Add the new block to the callout blocks for the sheet set
    calloutBlocks.Add(calloutBlockReference)
End Sub
```

The following method demonstrates how to setup a label block.

```
' Add a label block to the sheet set
Public Sub AddLabelBlock(ByVal sheetSetDatabase As AcSmDatabase, _
                         ByVal blockName As String, _
                         ByVal drawingFile As String)

    Dim labelBlockReference As New AcSmAcDbBlockRecordReference

    ' Create a reference to a layout label block
    labelBlockReference.InitNew(sheetSetDatabase)
    labelBlockReference.SetFileName(drawingFile)
    labelBlockReference.SetName(blockName)

    ' Set the label block for the sheet set
    sheetSetDatabase.GetSheetSet().SetDefLabelBlk(labelBlockReference)
End Sub
```

The following method demonstrates how to specify the location for page setup overrides.

```
' Add a page setup override to the sheet set
Public Sub AddPageSetupOverride(ByVal sheetSetDatabase As AcSmDatabase, _
                                ByVal pageSetupName As String, _
                                ByVal drawingFile As String)

    Dim fileReference As New AcSmFileReference

    ' Create a new reference to the page setup overrides to use
    fileReference.InitNew(sheetSetDatabase)
    fileReference.SetFileName(drawingFile)

    ' Set the drawing template that contains the page setup overrides
    sheetSetDatabase.GetSheetSet().SetAltPageSetups(fileReference)
End Sub
```

6 Enforcing CAD Standards

CAD standards are always different based on who you talk to, but one thing is always agreed upon though; that is they need to be enforced. Autodesk first implemented CAD standards back in AutoCAD 2002 which was a great start for improving CAD standards and enforcing consistency across many drawings. The problem with a feature like CAD standards though is the magnitude of what CAD standards means to each company and to the degree that they should be followed.

While the depth of the feature might not seem very great, it does introduce an extensible platform that companies or third-parties can leverage. This session goes into the basics of the CAD Standards Plug-in library which should give you an idea of how to implement your own CAD standards with a custom plug-in.

The CAD Standards feature consists of components inside and outside of the AutoCAD program:

- CAD Standards and Check Standards in the AutoCAD program
- Batch Standards Checker outside of AutoCAD

The plug-ins that ship with the AutoCAD program are used to validate four of the named objects a drawing can contain: layers, linetypes, text styles, and dimension styles. You are not limited to creating plug-ins for just validating named objects. It is possible to create a plug-in to work with graphical objects as well, such as moving a block to a specific layer based on its name or maybe even making sure a spline is 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.

What You Need Before Getting Started

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

- **CAD Standards Manager Library (*AcStMgr.tlb*)** – File is part of the ObjectARX SDK. You can download the ObjectARX SDK from *http://www.autodesk.com/objectarx*. Click License & Download and then follow the onscreen instructions
- **CAD Standards Manager Documentation** – The documentation for the CAD Standards Manager Library is part of the ObjectARX SDK. Once the ObjectARX SDK is installed, open the *arxdev.chm* file in the *docs* subfolder.

- **AutoCAD ActiveX Documentation** – The documentation for the AutoCAD Type Library is installed with the AutoCAD program. You can access the AutoCAD ActiveX Reference and AutoCAD ActiveX Developers guides by browsing to *C:\Program Files\Common Files\Autodesk Shared* and opening the files *acadauto.chm* and *acad_aag.chm*.

  The topics can also be found in the AutoCAD Online Help system, on the Developer's Home Page, click the AutoCAD ActiveX Library Reference and AutoCAD ActiveX Developer's Guide links. To access the online help, go to: *http://help.autodesk.com/view/ACD/2016/ENU/files/homepage_dev.htm*.
- **Windows SDK** – Based on the version of Visual Studio you are using, you might need to download and install the Windows SDK. From the Windows SDK, you will need to use *Tlbimp.exe* to generate a required file for your project. 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.

To 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 Manager library; this gives you the necessary DLL to link to your VB.NET project.

Note: If you are using Visual Studio Express, make sure you have downloaded and installed the Windows SDK as you will need access to *Tlbimp.exe*.

If you are using Microsoft Visual Studio 2012,

1. Click Start menu ➢ All Programs ➢ Microsoft Visual Studio 2012 ➢ Visual Studio Tools ➢ Developer Command Prompt for VS2012.
2. In the Command Prompt window, enter **cd "c:\ObjectARX 2016\inc-win32"** and press Enter. Make sure to specify the correct location of the *inc-win32* folder.
3. Type **tlbimp acstmgr.tlb** and press Enter.

   The *AcStMgr.dll* file is added to the *inc-win32* folder.
4. In the Command Prompt window, enter **cd "c:\ObjectARX 2016\inc-x64"** and press Enter. Make sure to specify the correct location of the *inc-x64* folder.
5. Type **tlbimp acstmgr.tlb** and press Enter.

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

If the *tlbimp.exe* file isn't available from the version of Visual Studio installed on your computer, follow these steps after installing the Windows SDK:

1. Click Start menu ➢ All Programs ➢ Accessories ➢ Command Prompt.
2. In the Command Prompt window, enter **cd "c:\ObjectARX 2016\inc-win32"** and press Enter. Make sure to specify the correct location of the *inc-win32* folder.
3. Type **"C:\Program Files\Microsoft SDKs\Windows\v7.1\Bin\tlbimp.exe" acstmgr.tlb** and press Enter.

   Make sure to specify the correct location of the Windows SDK installed. The file *AcStMgr.dll* is added to the *inc-win32* folder.

4. In the Command Prompt window, enter **cd "c:\ObjectARX 2016\inc-x64"** and press Enter. Make sure to specify the correct location of the *inc-x64* folder.
5. Type **"C:\Program Files\Microsoft SDKs\Windows\v7.1\Bin\tlbimp.exe" acstmgr.tlb** and press Enter.

   Make sure to specify the correct location of the Windows SDK installed. The file *AcStMgr.dll* is added to the *inc-x64* folder.

Defining a Plug-in

A plug-in is similar to that of a .NET assembly that can be loaded directly into AutoCAD, except you need to make sure specific properties are enabled so it can be loaded by the CAD Standards framework and the Batch Standards Checker. When you create a new project, you use the Class Library template. After you create the new project, you need to reference the AutoCAD Type and CAD Standards Plug-in libraries.

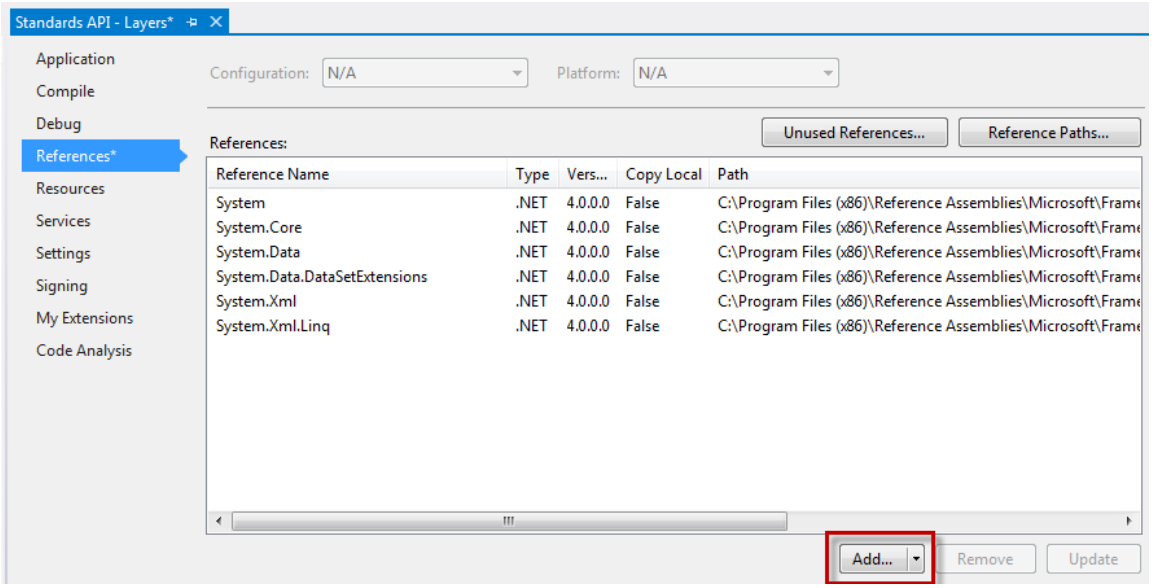
Create a New Class Library Project for a Plug-in

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

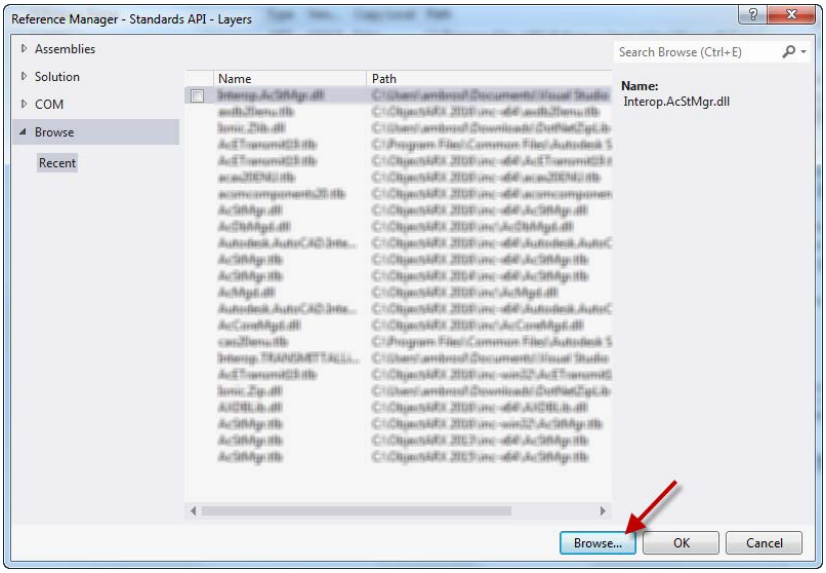
1. In Microsoft Visual Studio 2012, click File menu ➢ New Project.
2. In the New Project dialog box, select Visual Basic from Project Types and then select Class Library from Templates.
3. In the Name field, enter a name from your project, such as AULayersPlugin, and click OK.
4. Click Project menu ➢ *<ProjectName>* Properties.

The following steps explain how to reference CAD Standards Manager Library, AutoCAD/ObjectDBX Common Type Library, and Microsoft XML Type library.

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

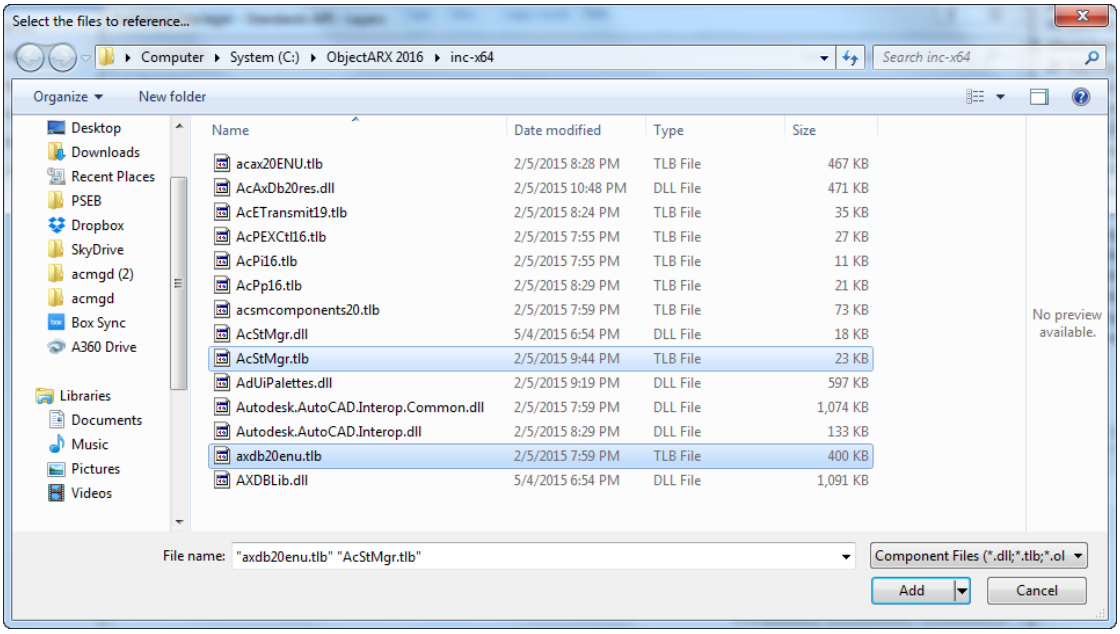


2. In Reference Manager dialog box, click Browse.

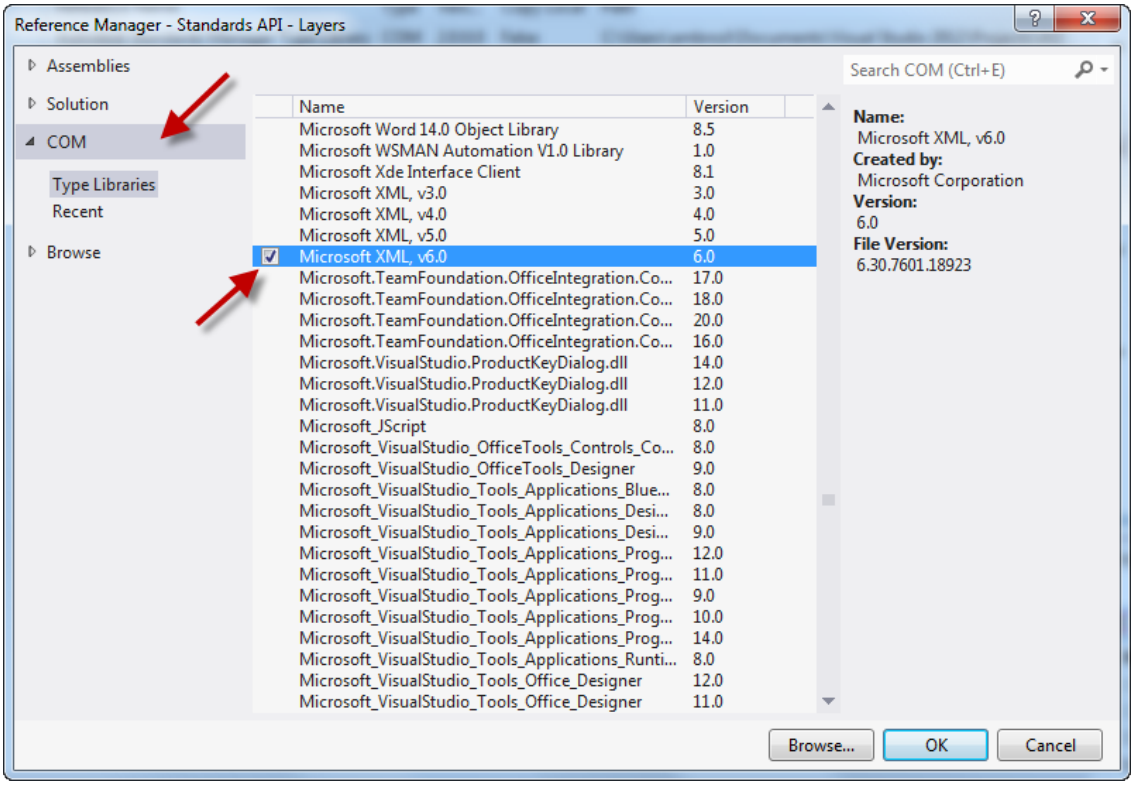


3. In the Select the Files to Reference dialog box, to *c:\ObjectARX 2016\inc-win32* or *c:\ObjectARX 2016\inc-x64*.

4. Press and hold Ctrl, and then select *AcStMgr.tlb* (in the same folder as the *AcStMgr.dll* which you should have created earlier) and *axdb20enu.tlb*. Click Add and then click OK.

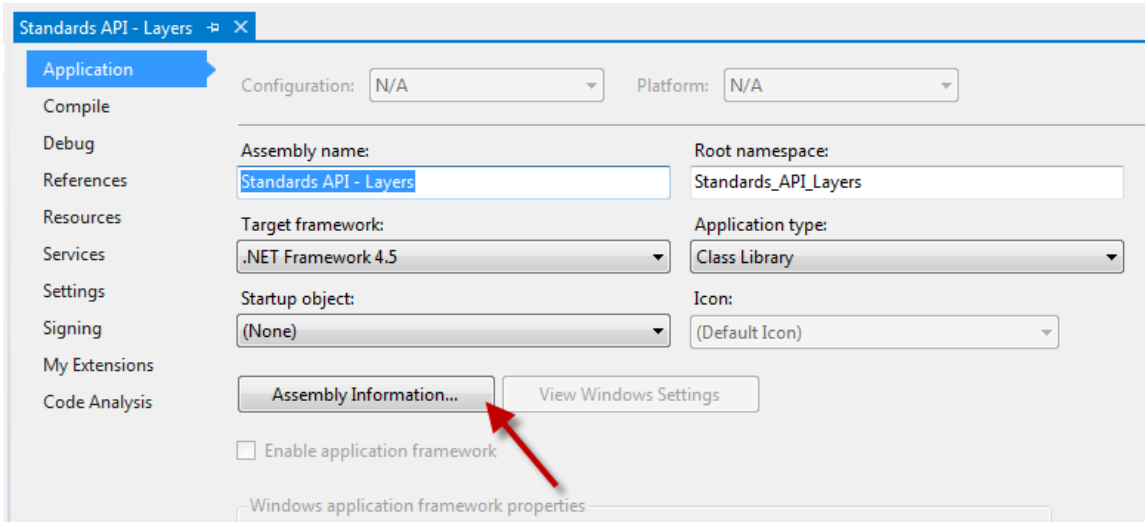


5. Click Add again. On the COM tab, scroll to and select Microsoft XML Type library (Microsoft XML, v6.0). Click OK.

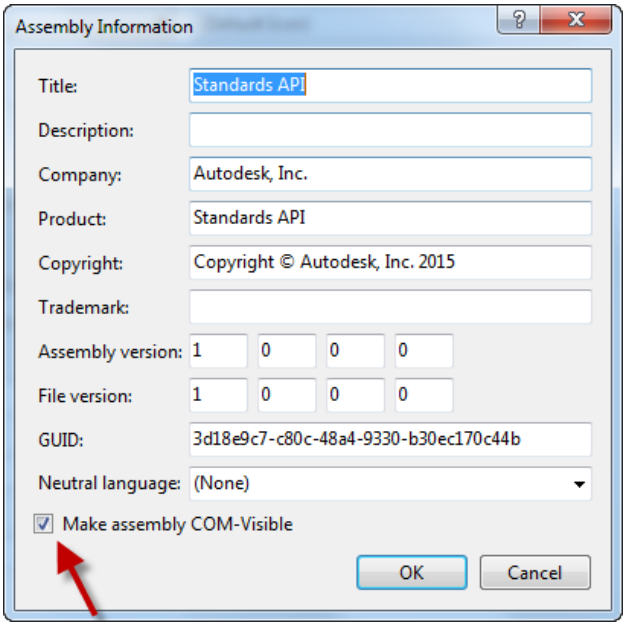


The following steps explain how to update the assembly information of the project.

1. On the Application tab, click Assembly Information.



2. In the Assembly Information dialog box, click Make Assembly COM-Visible (should be checked now) and click OK.



3. Click OK to apply the changes.

Note: Take note of the Root Namespace on the Application tab of the Properties Manager as it will be needed later when you have to register the plug-in.

The following steps explain how to import the namespaces of the referenced libraries in the code module.

1. In the Solution Explorer, double-click the code module in the project to open it for edit.
2. In the code editor window, click in the General Declaration area at the top of the file and type the following:

```
' Autodesk CAD Standards library
Imports AcStMgr

' Autodesk ObjectDBX library
Imports AXDBLib

' Microsoft XML library
Imports MSXML2
```

3. On the Visual Studio menu bar, click File ➢ Save All.

Basics of a CAD Standards Plug-ins Library

The CAD Standards Plug-ins library is small but can be a bit confusing at first. The `IAcStPlugin2` interface is used to implement a number of properties and methods required to define a CAD Standards 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 library 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-ins.

The methods and properties of the CAD Standards Plug-ins library fall into the following groupings:

- CAD Standard plug-in properties
- Initialization
- Error iterations
- Retrieve error and fix objects
- Fix errors
- Report errors

CAD Standard Plug-in Properties

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

- **`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 documentation on the plug-in can be located
- **`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

```
' Returns the Author of the plug-in, usually Company name.
' Required: Yes (Used by CAD Standards/Batch Standards Checker)
Public ReadOnly Property Author() As String _
                Implements IAcStPlugin2.Author
    Get
        Return "Lee Ambrosius, ABC Add-ons"
    End Get
End Property

' Returns the Description of the plug-in.
' Required: Yes (Used by CAD Standards/Batch Standards Checker)
Public ReadOnly Property Description() As String _
                Implements IAcStPlugin2.Description
    Get
        Return "Basic example of a layers CAD Standards plug-in."
    End Get
End Property

' Returns the URL where the plug-in can be downloaded from.
' Required: Yes (Used by CAD Standards/Batch Standards Checker)
Public ReadOnly Property HRef() As String _
                Implements IAcStPlugin2.HRef
    Get
        Return "http://www.abcaddons.com/layers/"
```

```
    End Get
End Property

' Returns the HICON property (icon) for the plug-in.
' Required: Yes (Used by CAD Standards/Batch Standards Checker)
Public ReadOnly Property Icon() As Long _
                Implements IAcStPlugin2.Icon
    Get
        Return 1
    End Get
End Property

' Returns the name of the plug-in; appears in the
' CAD Standards dialog box and the Batch Standards Checker.
' Required: Yes (Used by CAD Standards/Batch Standards Checker)
Public ReadOnly Property Name() As String _
                Implements IAcStPlugin2.Name
    Get
        Return "Layers Plug-in - Basic Sample"
    End Get
End Property

' Returns the version of the plug-in.
' Required: Yes (Used by CAD Standards/Batch Standards Checker)
Public ReadOnly Property Version() As String _
                Implements 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 can audit and receive notifications for. The array needs to be of the String data type and be the DXF name of the object type(s) to filter. It is recommended to filter on a single object type. If the method is not implemented, all object types are audited and notifications are performed when any object is changed.

```
' Defines an array of the type of objects (class names) that it can check
' It is recommended to use only one type of object for each plug-in
' Required: Yes (Used by CAD Standards/Batch Standards Checker)
Public Function GetObjectFilter() As Object _
       Implements IAcStPlugin2.GetObjectFilter

    ' Array of object types to check
    Dim sFilterArray(0) As String

    ' << Change based on standards implementation >>
    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 the AutoCAD program or the Batch Standards Checker. When a plug-in gets initialized varies slightly based the following conditions:

- The opening of a drawing with an associated DWS file and the plug-in is enabled for checking along with real-time checking enabled.
- The plug-in is checked while the CAD Standards dialog box or Batch Standards Checker is displayed.

The `Initialize()` method is the entry point of the 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.

After the plug-in has been initialized, the `SetupForAudit()` method is called to setup the drawing that should be checked along with which drawings the plug-in should obtain the standards information from. 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.

```
' Initializes the plugin
' Required: Yes (Used by CAD Standards/Batch Standards Checker)
Public Sub Initialize(ByVal pStdsMgr As AcStManager) _
           Implements IAcStPlugin2.Initialize

    ' Store the pointer to the manager object
    m_pManager = pStdsMgr
    m_pPlugin = Me
End Sub

' Sets the objects to examine when iterating over errors
' Required: Yes (Used by CAD Standards/Batch Standards Checker)
Public Sub SetupForAudit(ByVal pDb As AcadDatabase, _
                         ByVal szPathName As String, _
                         ByVal stdNameArray As Object, _
                         ByVal stdPathArray As Object, _
                         ByVal stdDbArray As Object) _
                               Implements IAcStPlugin2.SetupForAudit

    ' Used to define the context which the plug-in will operate.

    ' The drawing to be checked and the DWS files being used are passed
    ' to the plug-in at this point. Store the DWS standards definitions
    ' and the objects to be checked. The GetObjectFilter() method is used
    ' to specify the object type that the plug-in should check for errors.

    ' WARNING!: Do not save a reference to the database object,
    ' as it may not be valid going forward.

    If Not IsNothing(pDb) Then
        ' Store a reference to the DWG to be checked
        m_pCheckDatabase = pDb
```

```
        ' Verify the reference to the DWG is valid object
        If Not IsNothing(m_pCheckDatabase) Then
            ' << Change based on standards implementation >>
            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

        Dim iDWS As Integer

        ' Iterate each of the DWS files and cache the objects to use to fix errors
        For iDWS = 0 To UBound(stdDbArray)
            ' Get the DWS database
            m_pDWSDatabase = stdDbArray(iDWS)

            ' << Change based on standards implementation >>
            Dim pStdLayer As AcadLayer

            ' Iterate the layers in the database
            Dim i As Integer = 0
            For Each pStdLayer In m_pDWSDatabase.Layers
                Dim layerCache As New LayerCache()
                ' Cache the properties of the layers in the DWS database
                layerCache.Name = pStdLayer.Name
                layerCache.Color = pStdLayer.color
                layerCache.Lineweight = pStdLayer.Lineweight

                layerCache.StandardFileName = stdNameArray(iDWS)

                ' Create a fix object to be used later to correct a standards violation
                Dim pFix As New 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.

The `SetContext()` method requires an array of objects which contains the objects that should be checked in real-time for standards violations and a boolean that defines if the entire drawing or just the provided objects are checked.

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 of the error objects. After an error is checked, the `Done()` method is called to determine if there are any more errors, if there are more errors the `Next()` method is called to continue checking for 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 `SetContext()`, `Start()`, `Next()`, `Done()`, and `Clear()` methods.

```
' Sets the context for which the plug-in should be used to check a drawing
' Required: No, unless runtime checking is required (Used by CAD Standards)
Public Sub SetContext(ByVal objIdArray As Object, ByVal bUseDb As Boolean) _
                      Implements IAcStPlugin2.SetContext

    ' If useDb is set to "true" (default), or if objIdArray is blank,
    ' the context is set to the database (all ObjectIds in the drawing).
    ' Otherwise, the supplied array contains the objects to check.
    m_ContextList.SetContext(bUseDb, objIdArray)
End Sub

' Initializes the checking mechanism
' Required: No, unless runtime checking is required (Used by CAD Standards)
Public Sub PlugIn_Start(ByVal pStartError As AcStError) _
                        Implements IAcStPlugin2.Start

    ' pStartError is not an actual programming error that needs to be handled,
    ' but a reference to an object that does not match the standards
    ' in the DWS file.

    ' If pStartError is an object, checking should start
    ' from that error, not the beginning of the current context.
    If IsNothing(pStartError) = False Then

        ' Get the ObjectId of the first object to check
        Dim badId As Long = pStartError.BadObjectId

        ' Find the index for ObjectId in the context list stored in m_objIDArray
        For Me.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
        ' A valid AcStError object was not passed to the method.
        ' Checking should start at the beginning of the context list.
        m_curIndex = -1
        PlugIn_Next()
    End If
End Sub

' Get to the next error in the current context
' Required: No, unless runtime checking is required (Used by CAD Standards)
Public Sub PlugIn_Next() _
           Implements IAcStPlugin2.Next
    m_pError = Nothing

    If m_ContextList.Count > 0 Then

        ' Check to see if there are objects to check
        ' << Change based on standards implementation >>
        If m_LayerCacheArray.Length() > 0 Then

            Dim bFoundError As Boolean

            ' Check to see if this is the last object to be checked
            If m_curIndex < m_ContextList.Count - 1 Then
                m_curIndex = m_curIndex + 1
                bFoundError = False

                ' << Change based on standards implementation >>
                Dim layerObj As AcadLayer
                Dim iCache As Integer

                ' Iterate the objects to be checked
                While m_curIndex < m_ContextList.Count

                    ' Get the ObjectId of the object to be checked
                    layerObj = m_pCheckDatabase.ObjectIdToObject( _
                        m_ContextList.Item(m_curIndex))

                    ' Iterate through each of the standards objects
                    ' and check for a match based on name
                    For iCache = LBound(m_LayerCacheArray) To UBound(m_LayerCacheArray)
                        ' Compare the names of the two layers
                        If (layerObj.Name.CompareTo( _
                            m_LayerCacheArray(iCache).Name) <> 0) Then

                            ' Check to see if the object is a match,
                            ' if not flag as possible error
                            bFoundError = True
                        Else
                            bFoundError = False

                            ' Layer names matched, compare color and linetype values
                            If layerObj.color.ToString() <> _
```

```
                                m_LayerCacheArray(iCache).Color.ToString() Or _
                               layerObj.Lineweight.ToString() <> _
                                m_LayerCacheArray(iCache).Lineweight.ToString() Then

                                bFoundError = True
                            End If

                            ' Exit For loop since a match by name was found
                            Exit For
                        End If
                    Next

                    ' No match or a property difference was found
                    If bFoundError = True Then
                        Dim pError As New AcStError()

                        ' Assign a description to the error
                        pError.Description = "Layer is non-standard"

                        ' Assign an ObjectId to the error
                        pError.BadObjectId = layerObj.ObjectID

                        ' Assign the name of the object, plug-in, and error type
                        pError.BadObjectName = layerObj.Name
                        pError.Plugin = m_pPlugin
                        pError.ErrorTypeName = "Layer "

                        ' Assign a result status to the error
                        pError.ResultStatus = AcStResultStatus.acStResFlagsNone

                        ' Assign the object's property values to the error for checking
                        If pError.PropertyCount = 0 Then
                            pError.PropertyValuePut("Color", layerObj.color)
                            pError.PropertyValuePut("Lineweight", layerObj.Lineweight)
                        End If

                        m_pError = pError
                        bFoundError = False

                        Exit While
                    End If

                    ' Increment the counter
                    m_curIndex = m_curIndex + 1
                End While
            End If
        End If
    End If
End Sub

' Returns true if all errors have been fixed or ignored
' Required: No, unless runtime checking is required (Used by CAD Standards)
Public Function PlugIn_Done() As Boolean Implements IAcStPlugin2.Done
    Return IsNothing(m_pError)
End Function
```

```
' Cleans up after the plug-in, called before the plug-in is released from memory
' Required: No, unless runtime checking is required (Used by CAD Standards)
Public Sub PlugIn_Clear() Implements IAcStPlugin2.Clear

    ' If the XML Document object exist, the plug-in is being used
    ' by the Batch Standards Checker.
    ' Output the standards items that were used for comparison to
    ' the Standards Items section.
    If IsNothing(m_xmlDoc) = False Then
        WriteStandardsItemsInfo()
    End If

    ' Clean up the global variables
    m_xmlDoc = Nothing
    m_pPlugin = Nothing
    m_curIndex = -1
    m_RecommendedFixIndex = -1
    m_FixCnt = 0
    m_pManager = Nothing
    m_pDWSDatabase = Nothing
    m_pCheckDatabase = Nothing

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

    ' << Change based on standards implementation >>
    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 the 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 the 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 list of all fixes were already setup in the `SetupForAudit()` method, but they need to be returned as part of an `IAcStFix` data type array.

Along with a listing 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 being compared, comparison might be as simply as just comparing the names of non-graphical objects or a geometric property like radius.

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.

```
' Returns the current error
' Required: Yes  (Used by CAD Standards/Batch Standards Checker)
Public Function GetError() As AcStError Implements IAcStPlugin2.GetError
    Return m_pError
End Function

' Returns an array of all the possible fixes for the given error (IAcStFix objects)
' Note: Objects in the array need to be released
' Required: Yes  (Used by CAD Standards)
Public Sub GetAllFixes(ByVal pError As AcStError, _
                       ByRef fixArray As Object, _
                       ByRef recommendedFixIndex As Integer) _
                       Implements IAcStPlugin2.GetAllFixes

    ' << Change based on standards implementation >>
    If IsNothing(pError) = False Then
        Dim arr(UBound(m_LayerCacheArray)) As IAcStFix

        Dim i As Integer
        recommendedFixIndex = -1
        m_FixCnt = 0

        ' Check the cache of fixes
        If m_LayerCacheArray.Length > 0 Then

            ' Get the properties of the fix
            For i = LBound(m_LayerCacheArray) To UBound(m_LayerCacheArray)
                arr(i) = m_LayerCacheArray(i).StandardsFix
            Next

            fixArray = arr
            m_FixArray = fixArray
            Dim tmpFix As New AcStFix()

            ' Find the recommended fix for the error;
            ' the fix object does need to be retained
            If Not GetRecommendedFix(pError) Is Nothing Then
                recommendedFixIndex = m_RecommendedFixIndex
            End If
        End If
```

```
        ' Was a recommended fix found
        If recommendedFixIndex = -1 Then
            ' No recommended fix found, set the proper
            ' result status flag for the error object
            pError.ResultStatus = AcStResultStatus.acStResNoRecommendedFix
        End If
    End If
End Sub

' Gets the recommended fix object based on the specified error object
' Required: No, Optional  (Used by CAD Standards)
Public Function GetRecommendedFix(ByVal pError As AcStError) As AcStFix _
                                  Implements IAcStPlugin2.GetRecommendedFix

    Dim pRecommendedFix As New AcStFix()

    ' << Change based on standards implementation >>
    If m_LayerCacheArray.Length = 0 Then
        pError.ResultStatus = AcStResultStatus.acStResNoRecommendedFix
        pRecommendedFix = Nothing
    Else
        ' Get the ObjectId of the error object and the associated layer object
        Dim tmpObjID As Long = pError.BadObjectId()
        Dim tmpLayer As AcadLayer = m_pCheckDatabase.ObjectIdToObject(tmpObjID)

        ' Get the name of the layer
        Dim nameToBeChecked As String = tmpLayer.Name
        tmpLayer = Nothing
        Dim layCache As LayerCache = m_LayerCacheArray(0)

        m_RecommendedFixIndex = -1

        ' Attempt to get the recommended fix from
        ' the cached array by matching layer names
        Dim i As Integer
        For i = 0 To UBound(m_LayerCacheArray)
            If m_LayerCacheArray(i).Name = nameToBeChecked Then
                layCache = m_LayerCacheArray(i)
                m_RecommendedFixIndex = i

                ' Exit the For loop after a match is found
                Exit For
            End If
        Next

        ' Validate if a recommended fix was found
        If m_RecommendedFixIndex <> -1 Then
            ' Populate the properties for the recommended fix object
            pRecommendedFix.Description = "Layer fix"
            pRecommendedFix.StandardFileName = _
                m_LayerCacheArray(m_RecommendedFixIndex).StandardFileName
            pRecommendedFix.FixObjectName = _
                m_LayerCacheArray(m_RecommendedFixIndex).Name
            Debug.Print("Fix objectID:" & pRecommendedFix.FixObjectId.ToString())
```

```
            If pRecommendedFix.PropertyCount = 0 Then
                pRecommendedFix.PropertyValuePut("Color", m_LayerCacheArray( _
                                                 m_RecommendedFixIndex).Color)
                pRecommendedFix.PropertyValuePut("Lineweight", m_LayerCacheArray( _
                                                 m_RecommendedFixIndex).Lineweight)
            End If
        Else
            pRecommendedFix = Nothing
        End If
    End If

    GetRecommendedFix = pRecommendedFix
End Function

' Gets the property names and values to compare, and the
' fix objects available for the specified error
' Required: Yes  (Used by CAD Standards)
Public Sub GetPropertyDiffs(ByVal pError As AcStError, _
                            ByVal pFix As AcStFix, _
                            ByRef pPropNames As Object, _
                            ByRef pErrorValues As Object, _
                            ByRef pFixValues As Object, _
                            ByRef pFixableStatuses As Object) _
                            Implements IAcStPlugin2.GetPropertyDiffs

    If IsNothing(pError) = False And IsNothing(pFix) = False Then

        ' Define the arrays that will hold the property values to compare
        Dim sPropNames(0) As String
        Dim sErrorValues(0) As String
        Dim sFixValues(0) As String
        Dim bFixableStatuses(0) As Boolean

        Dim sPropName As String = ""
        Dim vErrorVal As Object = Nothing
        Dim vFixVal As Object = Nothing
        Dim i As Integer, nPropCnt As Integer = 0

        ' Iterate the properties of the error object
        For i = 0 To pError.PropertyCount - 1
            ' Get a property name and value from the error
            pError.PropertyGetAt(i, sPropName, vErrorVal)
            m_sPropName = sPropName

            Try
                ' Retrieve the corresponding property value for the fix object
                pFix.PropertyValueGet(sPropName, vFixVal)

                ' Compare the value of the error and fix objects
                If (vErrorVal.CompareTo(vFixVal) <> 0) Then
                    ' Resize the arrays and add the new values
                    ReDim Preserve sPropNames(nPropCnt + 1)
                    ReDim Preserve sErrorValues(nPropCnt + 1)
                    ReDim Preserve sFixValues(nPropCnt + 1)
```

```
                    ReDim Preserve bFixableStatuses(nPropCnt + 1)

                    ' Store the property names and values in arrays
                    sPropNames(nPropCnt) = sPropName
                    sErrorValues(nPropCnt) = vErrorVal.ToString
                    sFixValues(nPropCnt) = vFixVal.ToString
                    bFixableStatuses(nPropCnt) = True

                    ' Increment the property counter
                    nPropCnt = nPropCnt + 1
                End If
            Catch
                ' Catch and handle the error as needed
            End Try
        Next

        ' Return the arrays
        pPropNames = sPropNames
        pErrorValues = sErrorValues
        pFixValues = sFixValues
        pFixableStatuses = bFixableStatuses

        ' Clear the local arrays
        Erase sPropNames
        Erase sErrorValues
        Erase sFixValues
        Erase bFixableStatuses

        ' Increment the number of fixes
        m_FixCnt = m_FixCnt + 1
    End If
End Sub
```

Fix errors

After a standards violation has been identified as part of 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 as part of the `FixError()` method, you need to update the `ResultStatus` property of the error object with the value `AcStMgr.AcStResultStatus.acStResFixed`. Use the value `AcStMgr.AcStResultStatus.acStResFixFailed` when there is a problem updating the object. The following shows an example of the `FixError()` method.

```
' Attempts to use a fix object to change an error object
' Required: Yes  (Used by CAD Standards)
Public Sub FixError(ByVal pError As AcStError, _
                    ByVal pFix As AcStFix, _
                    Optional ByRef pFailedReason As String = "0") _
                    Implements IAcStPlugin2.FixError

    ' << Change based on standards implementation >>
    If IsNothing(pError) = False Then
```

```
        Dim sFixClrVal As ACAD_COLOR
        Dim sFixLWVal As ACAD_COLOR

        ' Get the drawing object to fix
        Dim badObjID As Long = pError.BadObjectId
        Dim badObj As AcadLayer = m_pCheckDatabase.ObjectIdToObject(badObjID)

        ' If no fix is provided, try the recommended fix
        If IsNothing(pFix) Then
            Dim tmpFix As New AcStFix()
            tmpFix = GetRecommendedFix(pError)

            If IsNothing(tmpFix) Then
                ' Set the result status of the error to Failed and No Recommended Fix
                pError.ResultStatus = AcStResultStatus.acStResFixFailed + _
                    AcStResultStatus.acStResNoRecommendedFix
            Else
                ' Fix the bad object
                pFix = tmpFix
                tmpFix = Nothing

                ' Fix the color of the layer
                pFix.PropertyValueGet("Color", sFixClrVal)
                Try
                    badObj.color = sFixClrVal
                    pError.ResultStatus = AcStResultStatus.acStResFixed
                Catch m_ex As Exception
                    pError.ResultStatus = AcStResultStatus.acStResFixFailed
                End Try

                ' Fix the Lineweight of the layer
                pFix.PropertyValueGet("Lineweight", sFixLWVal)
                Try
                    badObj.Lineweight = sFixLWVal
                    pError.ResultStatus = AcStResultStatus.acStResFixed
                Catch m_ex As Exception
                    pError.ResultStatus = AcStResultStatus.acStResFixFailed
                End Try
            End If

            tmpFix = Nothing
        Else
            ' Fix the color of the layer
            pFix.PropertyValueGet("Color", sFixClrVal)
            Try
                badObj.color = sFixClrVal
                pError.ResultStatus = AcStResultStatus.acStResFixed
            Catch m_ex As Exception
                pError.ResultStatus = AcStResultStatus.acStResFixFailed
            End Try

            ' Fix the Lineweight of the layer
            pFix.PropertyValueGet("Lineweight", sFixLWVal)
            Try
```

```
                badObj.Lineweight = sFixLWVal
                pError.ResultStatus = AcStResultStatus.acStResFixed
            Catch m_ex As Exception
                pError.ResultStatus = AcStResultStatus.acStResFixFailed
            End Try
        End If
    End If
End Sub
```

Report errors

Reporting errors is done in two different ways based on if you are checking standards in the AutoCAD program 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 the Next button is clicked, the Check Standards dialog box updates the `ResultStatus` property for the Standard 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 the automatic fixes performed using the recommended fixes. From the aspect of the library, the Check Standards interface keeps track of the total number of fixes that it is passed and the number that are fixed or ignored and reports the correct number 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 also handled for you. The one thing that the library and the Batch Standards Checker do 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.

```
' Used by the reporting feature of the Batch Standards Checker
' to get information about the plug-in.
' Required: No, Optional (Used by Batch Standards Checker)
Public Sub WritePluginInfo(ByVal pPluginInfoSectionNode As Object) _
                           Implements IAcStPlugin2.WritePluginInfo

    ' Sets the Section node for the XML file output
    Dim pSectionNode As IXMLDOMNode = pPluginInfoSectionNode

    ' Creates a new element
    Dim xmlElem As IXMLDOMElement = _
        pSectionNode.ownerDocument.createElement("AcStPluginInfo")

    ' Appends the new element to the XMl file
    Dim pPluginInfoElement As IXMLDOMElement = _
        pSectionNode.appendChild(xmlElem)

    ' Writes the plug-in information to the XML file
```

```
    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")

    ' Save a reference to the XML document in a global variable
    ' to allow additional information to be added to the XML file
    m_xmlDoc = pSectionNode.ownerDocument
End Sub

' Used to output information about the Standards items, called from Plugin_Clear() method
' Required: No, Optional (Used by Batch Standards Checker)
Public Sub WriteStandardsItemsInfo()

    If IsNothing(m_xmlDoc) = False Then

        ' Check for the AcStDWSSection section element in the XML file
        ' Get the first, should be the only instance, of AcStDWSSection
        Dim pAcStDWSSection As IXMLDOMNode = _
                              m_xmlDoc.getElementsByTagName("AcStDWSSection")(0)

        If IsNothing(pAcStDWSSection) = False Then
            For Each child As IXMLDOMNode In pAcStDWSSection.childNodes
                If child.nodeName = "AcStDWSFile" Then

                    For Each grandchild As 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
                                ' << Change based on standards implementation >>
                                For Each pFix As LayerCache In m_LayerCacheArray

                                    ' Creates a new element for a standards item
                                    Dim xmlElem As IXMLDOMElement = grandchild. _
                                        ownerDocument.createElement("AcStDWSItem")

                                    ' Appends the new element to the XML file
                                    Dim pAcStDWSItemElement As IXMLDOMElement = _
                                        grandchild.appendChild(xmlElem)
```

```
                                ' Writes information about the standards item
                                ' to the XML file
                                pAcStDWSItemElement. _
                                    setAttribute("DWSId", pDWSId)
                                pAcStDWSItemElement. _
                                    setAttribute("PluginId", pPluginId)
                                pAcStDWSItemElement. _
                                    setAttribute("ItemName", pFix.Name)

                                ' Creates a new element for the Color property
                                Dim xmlElemProp1 As IXMLDOMElement = _
                                    pAcStDWSItemElement.ownerDocument. _
                                    createElement("Property")

                                ' Appends the new element to the XML file
                                Dim pAcStDWSItemElementProp1 As IXMLDOMElement = _
                                    pAcStDWSItemElement.appendChild(xmlElemProp1)

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

                                ' Creates a new element for the Lineweight property
                                Dim xmlElemProp2 As IXMLDOMElement = _
                                    pAcStDWSItemElement.ownerDocument. _
                                    createElement("Property")

                                ' Appends the new element to the XML file
                                Dim pAcStDWSItemElementProp2 As IXMLDOMElement = _
                                    pAcStDWSItemElement.appendChild(xmlElemProp2)

                                pAcStDWSItemElementProp2. _
                                    setAttribute("ItemDescription", _
                                                 "Lineweight = " + _
                                                 pFix.Lineweight.ToString())
                            Next
                        End If
                    End If
                Next
            End If
        Next
    End If
End Sub
```

Other Methods to Implement

In addition to the methods previously mentioned, you need to implement the `CheckSysvar()` and `StampDatabse()` methods even though they are not required or used.

```
' Checks the value of a system variable
' Required: Yes (Used by CAD Standards)
Public Sub CheckSysvar(ByVal syvarName As String, _
                       ByVal bGetAllFixes As Boolean, _
                       ByRef bPassFail As Boolean) _
                       Implements IAcStPlugin2.CheckSysvar
```

```
    ' Not implemented
End Sub

' Specifies if the plug-in uses information from the database for checking
' Required: Yes (Used by CAD Standards)
Public Sub StampDatabase(ByVal pDb As AcadDatabase, _
                         ByRef pStampIt As Boolean) _
                         Implements IAcStPlugin2.StampDatabase

    ' << Change based on standards implementation >>
    ' If the DWS contains layers, return true
    ' Otherwise, return false
    pStampIt = False

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

Load and Use a Plug-in

Once you have defined and built a plug-in, it needs to be registered with both AutoCAD and Windows before it can be used in either the CAD Standards Checker or Batch Standards Checker. Once the plug-in is registered, it can be loaded by the AutoCAD program or the Batch Standards Checker the next time they are loaded.

Register and Load a Plug-in

The registration of the plug-in requires two parts; it needs to be registered with Windows and AutoCAD. The following outlines how to register a plug-in:

- **Windows** – You register your assembly using RegAsm.exe. The following steps explain how to register the assembly with Windows:

  1. Click Start menu ➢ All Programs ➢ Accessories ➢ Command Prompt.
  2. In the Windows Command Prompt, use the RegAsm.exe to register the DLL built.
     For example,
     "C:\Windows\Microsoft.NET\Framework\v4.0.30319\RegAsm.exe" /codebase /regfile:"Standards API - Layers.reg" "bin\x64\Debug\Standards API - Layers.dll"

     Replace *bin\x64\Debug\Standards API - Layers.dll* with the location of your plug-in. The location of your DLL can be found by displaying the Output window in Visual Studio (View menu > Other Windows > Output).

- **AutoCAD** – You register your plug-in using a Windows Registry entry. The Registry key you need to modify is: HKEY_LOCAL_MACHINE\SOFTWARE\Autodesk\Drawing Check\Plugins2

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

  - AutoLISP using the `vl-registry-write` function
  - VB.NET using the `SaveSetting` method
  - Define the registry key in a REG file

To define a REG file, do the following:

1. Create a new file in Notepad (click Start menu ➢ Accessories ➢ Notepad) and enter the following:

   REGEDIT4

   [HKEY_LOCAL_MACHINE\SOFTWARE\Autodesk\Drawing Check\Plugins2\CustomPlugins.BasicLayers]
   @="Standards_API_Layers"

   CustomPlugins.BasicLayers in the above example must be replaced with the ProgId assigned to your class in the VB.NET project.

2. Save the file in Notepad with an REG file extension. (Click File menu ➢ Save As, and then select All Files from the Save As Type drop-down list. Enter the file name with .reg added to the end of the file name and click Save.)

3. In Windows Explorer ir File Explorer, double-click the file. Click Yes and then OK in the messages boxes that are displayed.

Use and Test a Plug-in

Each plug-in should support the functionality of the following workflows:

- Checking standards from the AutoCAD program
- Applying automatic/recommended fixes when checking standards in AutoCAD
- Real-time checking in the AutoCAD program via notifications
- Checking standards with the Batch Standards Checker

Use the following steps to test checking and fixing 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, check the plug-in.
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 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, check the plug-in.
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, select the standards file to use for automatic fixes.
7. Click OK.
8. Click Check Standards.
9. Verify all automatic fixes where 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, check the plug-in. Click OK.
4. Modify the drawing so it goes against the standards that are being monitored.

Use the following steps to batch check the standards of a drawing against your plug-in.

1. On the Start menu, click All Programs ➢ Autodesk ➢ AutoCAD 2016 ➢ 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, check the plug-in.
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.

Create a Graphical Plug-in

The example code that has been shown up until now has been focused on how to implement a plug-in that compares layers in a drawing against those in a DWS file. A plug-in can be designed to compare graphical objects in Model space or Paper space.

The only major difference between supporting graphical vs non-graphical standards checking is to adjust the object filter to work with the graphical objects of interest as part of the `SetupForAudit()` method. Once you have filtered the objects you want to validate, you then identify the properties of the graphical objects that need to be compared. You might compare the scale of blocks or the layers that they are inserted on along with where they are inserted in a drawing.

You could create a plug-in that checks the placement of geometry in relationship to other objects. For example, you could create a basic validation tool to make sure objects were snapped together correctly. I wrote a utility a number of years ago that checked the condition of blocks in relationship to other

blocks and could have leveraged the CAD Standards framework for some of the functionality that I implemented.

The properties you want to compare for graphical objects affects how you go about implementing part of the plug-in. These handouts are accompanied with a sample plug-in that compares a block (title block to be more specific) in Paper space against the same block in a DWS file. 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 do not 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 if needed though.

The Layers plug-in can be used as the foundation for a Title Block plug-in, and here is how:

- Changed the `LayerStandardsHelpers` class to `TitleBlocksStandardsHelpers` in *StandardsHelp.vb*
- Class name, ProgId, a few of the 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 easy to create a new plug-in once you have your first one created.

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 the AutoCAD program before.

Debug a Plug-in in AutoCAD

The following steps explain how to debug a plug-in from inside of AutoCAD.

1. In Microsoft Visual Studio 2012, click Project menu ➢ *<Project>* Properties.
2. In the Properties dialog, click the References tab.
    a. Click Reference Paths.
    b. In the Reference Paths dialog box, click [..] to the right of the Folder text box.
    c. In the Select Reference Path dialog box, browse to and select the *inc-win32* folder. Click OK.

       By default *inc-win32* is located under *c:\ObjectARX 2016*.
    d. Click Add Folder and then click OK.
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 *acad.exe* for AutoCAD 2016 is located under *C:\Program Files\Autodesk\AutoCAD 2016*.

4. Register your plug-in in the Windows Registry using a REG file.
5. In Microsoft Visual Studio 2012, set the desired breakpoints in the code window.

   Remember that the Initialize method is the entry point of your CAD Standard plug-in.
6. Click Debug menu ➢ Start Debugging.

   AutoCAD 2016 starts up.
7. In AutoCAD 2016, open the drawing file that you want to check.
8. On the ribbon, Manage tab ➢ CAD Standards panel ➢ Configure.
9. In the Configure Standards dialog box, Standards tab, click Add and select the DWS files you want to use.
10. On the Plug-ins tab, select your CAD Standard plug-in so it is checked.
11. Save the drawing file being checked.
12. Click Check Standards to start checking the drawing.

    Focus is shifted to the breakpoint in Visual Studio.

7 Creating Transmittal Packages

Sharing and archiving drawings can be a challenge even with features like eTransmit and archive for Sheet Sets. The eTransmit feature can be used with a single drawing file or a sheet set, and the archival feature only works with sheet sets which not all companies use. Both the eTransmit and archive features rely on the use of the Transmittal Object (ATO) library. The Transmittal Object library can be used to emulate much of the functionality that the ETRANSMIT and ARCHIVE commands support.

The Transmittal Object library can be used to:

- Identify the reference files that a drawing file relies on to display or plot correctly.
- Gather and copy drawing files and their referenced files to a destination folder; can work with individual drawing files, sheets from a sheet set, a sheet selection set from a sheet set, or a drawing set (DST) file.
- Add supplemental files to a transmittal set that are required beyond the drawing files of a project, such as general specifications, definition of the layer standards used in the drawings, or a report of all the files in the transmittal package.
- Monitor files being added to a transmittal set created with the ETRANSMIT command or the Transmittal Object library.

The ETRANSMIT command supports some features that are not part of the Transmittal Object library. The features that are not part of the Transmittal Object library, but are part of the ETRANSMIT command are:

- Creation of a ZIP file that contains the contents of a transmittal package
- Sending an email that contains the transmittal package as an attachment

What You Need Before Getting Started

Before you start working with the Transmittal library, you will need to obtain the following:

- **Transmittal Library (*AcETransmit19.tlb*)** – File is installed with the AutoCAD program or is part of the ObjectARX SDK.
- **Transmittal Library Documentation** – The Transmittal library documentation is part of the ObjectARX SDK documentation. You can download the ObjectARX documentation at *http://www.autodesk.com/objectarx/*. The reference documentation can be found under ObjectARX Reference Guide ➢ eTransmit Interfaces (WIN) in the file *arxref.chm* and the developer concepts can be found under Using COM for ObjectARX Development ➢ eTransmit COM API in the file *arxdev.chm*.
- **AutoCAD ActiveX Documentation** – The documentation for the AutoCAD Type Library is installed with the AutoCAD program. You can access the AutoCAD ActiveX Reference and AutoCAD ActiveX Developers guides by browsing to *C:\Program Files\Common Files\Autodesk Shared* and opening the files *acadauto.chm* and *acad_aag.chm*. The topics can also be found in the AutoCAD Online Help system, on the Developer's Home Page, click the AutoCAD ActiveX Library Reference and AutoCAD ActiveX Developer's Guide links.

Taking a Look at the Transmittal Object Library

Before you can automate tasks related to the eTransmit and archival features, you must reference the Transmittal Object library. The Transmittal Object library is stored in a file named *AcETransmit <version>.tlb* under the Autodesk Shared subfolder in the *Program Files\Common Files* folder; in the Reference Manager it is listed as Transmittal *<Version>* Type Library.

The following table lists the file name and version of the ActiveX libraries you need to reference by AutoCAD release.

AutoCAD Release	Transmittal Object Library	AutoCAD/ObjectDBX Common Type Library
AutoCAD 2016 and AutoCAD 2015	*AcETransmit19.tlb*	*axdb20<language>.tlb*
AutoCAD 2014 and AutoCAD 2013	*AcETransmit18.tlb*	*axdb19<language>.tlb*
AutoCAD 2012, AutoCAD 2011, and AutoCAD 2010	*AcETransmit18.tlb*	*axdb18<language>.tlb*
AutoCAD 2009, AutoCAD 2008, and AutoCAD 2007	*AcETransmit16.tlb*	*axdb17<language>.tlb*

Once you have referenced the library, you need to reference the namespace that contains the Transmittal Object library. The following statement imports the Transmittal Object library in a VB.NET code module:

```
Imports TRANSMITTALLib
```

After the Transmittal Object library namespace has been imported, you must create a reference to the Transmittal Operation object which is represented by the `TransmittalOperatrion` object. The following code demonstrates how to reference the Transmittal Operation object:

```
' Create a transmittal operation
Dim tro As TransmittalOperation = New TransmittalOperation()
```

With the Transmittal Operation object, you can

- Get the settings related to how drawing and referenced files are to be handled (`getTransmittalInfoInterface`)
- Add drawing and other files to a transmittal set (`addDrawingFile/addFile`)
- Add sheets from a sheet to a transmittal set (`addSheets/addSheetSelset/archiveSheetSet`)
- Get the drawing and reference files in a transmittal set (`getTransmittalGraphInterface`)
- Modify and get the transmittal report (`addToReport/getTransmittalReport`)
- Create a transmittal package from the transmittal set (`createTransmittalPackage`)
- Recalculate the paths of files (`recalcTargetSubPath`)
- Monitor the progress of creating a transmittal package (`setProgressMeter`)
- Monitor changes to the files in a transmittal set

Configuring the Transmittal Interface

Before adding drawings and other files to a transmittal set, you should configure the settings that will be used to create the resulting transmittal package. The settings for a transmittal set are stored in a `TransmittalInfo` object; each `TransmittalOperation` object created has its own `TransmittalInfo` object. The `getTransmittalInfoInterface()` method is used to obtain the `TransmittalInfo` object of a `TransmittalOperation` object.

Once you have a `TransmittalInfo` object, you can change the behavior in which a transmittal package is created. Here are some of the settings that can be changed for a transmittal set:

- Specify the destination folder and whether the folder structure of the source files is preserved
- Maintain absolute paths for referenced files or use relative paths
- Control the inclusion of attached reference, plot, shape, photometric web, and material files
- Specify the paths of plot configuration and style files
- Purge unused named styles

The following sample code is of a custom function that configures the settings of a `TransmittalOperation` object by modifying the properties of its `TransmittalInfo` object.

```
' Custom function used to setup a TransmittalInfo object
Private Shared Function TransInfo(ti As TransmittalInfo, destination As String) _
    As TransmittalInfo

    ' Specify the folder that the files should be copied to
    If System.IO.Directory.Exists(destination) = False Then
        System.IO.Directory.CreateDirectory(destination)
    End If

    ' Assign the location which the transmittal package should be stored
    ti.destinationRoot = destination

    ' Maintain sub directories (0 - False, 1 - True)
    ti.preserveSubdirs = vbTrue
```

```
    ' Maintain folder paths
    ' ti.organizedFolder = 0 and ti.preserveSubdirs = 0 -
    '             All files are placed in a single folder which is specified
    '             by the destinationRoot property
    ' ti.organizedFolder = 1 and ti.preserveSubdirs = 0 -
    '             Files are placed in a single folder, but maintain one level
    '             of subdirectory strcuture
    ' ti.organizedFolder = 0 and ti.preserveSubdirs = 1 -
    '             Folder structure is maintained, but the paths for
    '             reference files are not
    ' ti.organizedFolder = 1 and ti.preserveSubdirs = 1 -
    '             Folder structure and reference paths are maintained
    ti.organizedFolder = vbTrue

    ' Change referenced files to use relative paths (0 - False, 1 - True)
    ti.dataLinkRelativePath = vbTrue
    ti.DGNUnderlayRelativePath = vbTrue
    ti.DWFUnderlayRelativePath = vbTrue
    ti.imageRelativePath = vbTrue
    ti.PDFUnderlayRelativePath = vbTrue
    ti.xrefRelativePath = vbTrue

    ' Set the default drawing file format for all DWG, DWT, and DWS files
    ' SaveDwgFormat.eAutoCAD14 - AutoCAD R14
    ' SaveDwgFormat.eAutoCAD2000 - AutoCAD 2000
    ' SaveDwgFormat.eAutoCADR18 - AutoCAD 2004
    ' SaveDwgFormat.eAutoCADR21 - AutoCAD 2007
    ' SaveDwgFormat.eAutoCADR24 - AutoCAD 2010
    ' SaveDwgFormat.eAutoCADR27 - AutoCAD 2013
    ' SaveDwgFormat.eNoConversion - Keep current format
    ti.saveVersion = SaveDwgFormat.eNoConversion

    ' Maintain visual fidelity (SAVEFIDELITY system variable)
    ' 0 - Saved without visual fidelity
    ' 1 - Annotation objects assigned annotative scales are saved to
    '     separate layers for each scale used.
    '     Only affects drawings when saving to AutoCAD 2007 or earlier file formats.
    ti.visualFidelity = vbTrue

    ' Include DWG Xrefs (Unloaded and Unreferenced: 0 - False, 1 - True)
    ti.includeXrefDwg = vbTrue
    ti.includeUnloadedReferences = vbTrue
    ti.includeUnloadedXrefDwg = vbTrue
    ti.includeNestedOverlayXrefDwg = vbTrue

    ' Bind xrefs attached to a drawing (0 - False, 1 - True)
    ' BindType.eBind - Use the Bind type to bind the xrefs
    ' BindType.eInsert - Use the Insert type to bind the xrefs
    ti.BindType = BindType.eInsert

    ' Remove paths from xrefs (0 - False, 1 - True)
    ti.depathXrefs = vbTrue

    ' Include images, underlays, and datalink file (0 - False, 1 - True)
    ti.includeImageFile = vbTrue
```

```
    ti.includeDGNUnderlay = vbTrue
    ti.includeDWFUnderlay = vbTrue
    ti.includePDFUnderlay = vbTrue
    ti.includeDataLinkFile = vbTrue

    ' Include font (SHX/TTF) and font map (FMP) files (0 - False, 1 - True)
    ti.includeFontFile = vbTrue

    ' Reset plotters for layouts in the drawings to None,
    ' and include plot style (CTB/STB) files (0 - False, 1 - True)
    ti.resetPlotter = vbFalse
    ti.includePlotFile = vbTrue

    ' Specify where the plot files are located so the API can resolve the file
    ' paths (used when includePlotFile = vbTrue or 1)
    ti.plotConfigFilePath = Environment.GetEnvironmentVariable("UserProfile") + _
             "\AppData\Roaming\Autodesk\AutoCAD 2016\R20.1\enu\Plotters"
    ti.plotStyleTablePath = Environment.GetEnvironmentVariable("UserProfile") + _
             "\AppData\Roaming\Autodesk\AutoCAD 2016\R20.1\enu\Plotters\Plot Styles"

    ' Get the photometric Web and material files used by lights
    ' and 3D objects for rendering (0 - False, 1 - True)
    ti.includePhotometricWebFile = vbTrue
    ti.includeMaterialTextureFile = vbTrue

    ' Purge unused named objects for all drawing files in the
    ' Transmittal(Package(0 - False, 1 - True))
    ti.purgeDatabase = vbFalse

    Return ti
End Function
```

Adding Files to a Transmittal Set

Drawing files are the focus of a transmittal set and without them there is less to transmit. The first file you add to a transmittal set becomes the root file and should be a drawing file. When a drawing file is added, the transmittal engine searches the drawing file for all referenced files that are required to properly display and plot the drawing file. A drawing file is added to a transmittal set using the `addDrawingFile()` method.

Once a drawing file is added, you can add additional drawing files that are needed to complete a project. Non-drawing files can also be added to a transmittal set. The `addFile()` method can be used to add non-drawing files such as a word processor document that might contain bid specifications, a spreadsheet that contains the parts or items that are required for the completed design, or even an HTML file that explains the layer standards used in the drawings to a transmittal set. The difference between the `addDrawingFile()`and `addFile()` methods is that the transmittal engine doesn't try to search files added with the `addFile()` method for referenced files.

The following sample code demonstrates how to add a drawing and non-drawing file to a transmittal set.

```
' Enum used to define the file type being used
' with the TransAddFile and TransAddSheetComponent methods
Enum FileType
```

```
    Unknown = 0
    DWG = 1
    Sheets = 2
    SheetSet = 3
    SheetSetSel = 4
End Enum

' Custom command to create a transmittal package with a drawing file
<CommandMethod("eTransmitDWGFile")> _
Public Shared Sub eTransmitDWGFile()

    ' Create a transmittal operation
    Dim tro As TransmittalOperation = New TransmittalOperation()

    ' Setup the transmittal behavior
    Dim ti As TransmittalInfo = _
        TransInfo(tro.getTransmittalInfoInterface(), c_destinationRoot)

    ' Add a DWG file
    Dim tf As TransmittalFile = Nothing
    tro.addDrawingFile(c_dwgFile, tf)

    ' Reclaculate the sub path for all files in the transmittal package
    tro.recalcTargetSubPath()

    ' Create the transmittal package
    ' Files are copied and resaved to the path specified by the destinationRoot property
    ' and the other settings of the TransmittalInfo object.
    tro.createTransmittalPackage()
End Sub

' Custom command to create a transmittal package with a non-drawing file
<CommandMethod("eTransmitNonDWGFile")> _
Public Shared Sub eTransmitNonDWGFile()

    ' Create a transmittal operation
    Dim tro As TransmittalOperation = New TransmittalOperation()

    ' Setup the transmittal behavior
    Dim ti As TransmittalInfo = _
        TransInfo(tro.getTransmittalInfoInterface(), c_destinationRoot)

    ' Add a non-drawing file
    Dim tf As TransmittalFile = Nothing
    tro.addFile(c_otherfile1, Nothing, 0, tf)

    ' Create the transmittal package
    ' Files are copied and resaved to the path specified by the destinationRoot property
    ' and the other settings of the TransmittalInfo object.
    tro.createTransmittalPackage()
End Sub
```

Prior to adding a file to a transmittal set, you should verify that the file exists. The following custom function verifies whether the file being added exists, and wraps the `addDrawingFile()` and `addFile()` methods.

```
' Custom method for adding files to a transmittal package
Private Shared Function TransAddFile(tro As TransmittalOperation, filename As String, _
                                     flag As FileType, _
                                     Optional tfParent As TransmittalFile = Nothing) _
                                  As TransmittalFile

    ' Define a variable for the return value
    Dim tf As TransmittalFile = Nothing

    If System.IO.File.Exists(filename) = True Then
        ' Determine which type of file to add
        Select Case flag
            Case FileType.DWG
                ' Add a drawing file
                tro.addDrawingFile(filename, tf)
            Case FileType.Unknown
                ' Add a file other than a drawing or one that is referenced
                ' to the drawings in the transmittal package
                tro.addFile(filename, tfParent, vbFalse, tf)
        End Select
    End If

    Return tf
End Function
```

Working with Sheet Sets

Sheet sets can contain a large number of drawing and resource files that are needed to collaborate on a project. You can add one or more individual sheets, all of the sheets in a sheet selection, or all of the sheets in a sheet set to a transmittal set. The approach you take to add content from a sheet set to a transmittal set depends on which files in a sheet set you want to transmit. When adding sheets from a sheet set to a transmittal set, you need to work with the Sheet Set Object library.

You add individual sheet component (`AcSmSheet`) objects to a transmittal set with the `addSheets()` method. The `addSheets()` method expects an array of object IDs that represents each sheet to be added to the transmittal set. The object ID of a sheet can be obtained with the `GetObjectId()` method of an `AcSmSheet` object. While less commonly used, the sheet selection feature of sheet sets can be used to specify which sheets to add from a sheet set to a transmittal set. If the sheets you want to add are part of a sheet selection defined in a sheet set, you can use the `addSheetSelset()` method. The `addSheetSelset()` method expects an `AcSmSheetSelSet` object which represents the sheet selection set that contains the sheets to be added to the transmittal set.

The `addSheets()` and `addSheetSelset()` methods are helpful to control which sheets from a sheet set are added to a transmittal set, but that level of control isn't often needed for automating the task of transmitting or archiving an entire project. You can add all the sheets from a sheet set to a transmittal set with the `ArchiveSheetSet()` method.

Note: The SSFOUND system variable can be used to determine if a drawing belongs to a sheet set and which one.

The following sample code demonstrates how to add a single sheet, and its reference and resource files to a transmittal set with the `addSheets()` method.

```
' Custom command to create a transmittal package from the sheets in a sheet set
<CommandMethod("eTransmitDSTFile_Sheets")> _
Public Shared Sub eTransmitDSTFile_Sheets()

    ' Create a transmittal operation
    Dim tro As TransmittalOperation = New TransmittalOperation()

    ' Setup the transmittal behavior
    Dim ti As TransmittalInfo = _
        TransInfo(tro.getTransmittalInfoInterface(), c_destinationRoot)

    ' Add a sheet selection set from a sheet set and
    ' the drawings contained in it
    ' 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_dstFile, False)

    ' Get the enumerator for the objects in the
    ' sheet set and then set the first object
    Dim enumerator As IAcSmEnumPersist = sheetSetDatabase.GetEnumerator()
    Dim itemSheetSet As IAcSmPersist = enumerator.Next()

    Dim sheetNames(0) As String
    Dim sheetSetSelOrSheetName As String = _
        "T-01 TITLE SHEET;A-01 MAIN AND SECOND FLOOR PLAN"

    ' Semi-colon detected, multiple sheet names are being passed
    If sheetSetSelOrSheetName.Contains(";") = True Then
        sheetNames = sheetSetSelOrSheetName.Split(";")
    Else
        sheetNames(0) = sheetSetSelOrSheetName
    End If

    ' Create an array of the sheets based on their names
    Dim sheetArray(0) As IAcSmObjectId
    Dim sheet As AcSmSheet, nCount As Integer = 0

    ' Step through the objects in the sheet set
    Do While Not itemSheetSet Is Nothing

        ' Check to see if the object is a Sheet
        If itemSheetSet.GetTypeName() = "AcSmSheet" Then

            ' A sheet was found, now to see if it is the one needed
            sheet = itemSheetSet

            ' Step through each name provided
            For Each sheetName As String In sheetNames

                ' Check to see if the sheet name matches the sheet
                If UCase(sheet.GetName()).Equals(UCase(sheetName)) = True Then
                    ReDim Preserve sheetArray(0 To nCount)
```

```
                    ' Match was found, add the sheet's object Id to the array
                    sheetArray(nCount) = sheet.GetObjectId()

                    ' Increment the counter by 1
                    nCount = nCount + 1
                End If
            Next
        End If

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

    ' Get the sheets from the database and process them
    ' 0 - Do not include DST file or support files referenced
    '     by the DST file (templates, blocks, ...)
    ' 1 - Include DST file, all files referenced by the
    '     DST file, its sheets and references
    If sheetArray.Length > 0 And sheetArray(0).Equals("") = False Then
        tro.addSheets(sheetArray, vbTrue)
    End If

    ' Close the sheet set
    sheetSetManager.Close(sheetSetDatabase)

    ' Create the transmittal package
    ' Files are copied and resaved to the path specified by the destinationRoot property
    ' and the other settings of the TransmittalInfo object.
    tro.createTransmittalPackage()
End Sub
```

The following sample code demonstrates how to add all the sheets from a sheet selection set, and its reference and resource files to a transmittal set with the `addSheetSelset()` method.

```
' Custom command to create a transmittal package
' from a sheet selection set in a sheet set
<CommandMethod("eTransmitDSTFile_SheetSelSet")> _
Public Shared Sub eTransmitDSTFile_SheetSelSet()

    ' Create a transmittal operation
    Dim tro As TransmittalOperation = New TransmittalOperation()

    ' Setup the transmittal behavior
    Dim ti As TransmittalInfo = _
        TransInfo(tro.getTransmittalInfoInterface(), c_destinationRoot)

    ' Add a sheet selection set from a sheet set
    ' and the drawings contained in it
    ' 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_dstFile, False)
```

```
    ' Get all the SheetSelSets in the sheet set
    Dim sheetSelSetEnum As IAcSmEnumSheetSelSet = _
        sheetSetDatabase.GetSheetSet().GetSheetSelSets().GetEnumerator()

    ' Get the first SheetSelSet
    Dim sheetSelSet As AcSmSheetSelSet = sheetSelSetEnum.Next()
    Dim sheetSelSetFound As AcSmSheetSelSet = Nothing

    ' Check to the see if it is the requested SheetSelSet,
    ' if not get the next and check
    While Not sheetSelSet Is Nothing
        If UCase(sheetSelSet.GetName()). _
                        Equals("ELEVATIONS") = True Then
            sheetSelSetFound = sheetSelSet
        End If

        sheetSelSet = sheetSelSetEnum.Next()
    End While

    ' Get the sheet set from the database and process it
    ' 0 - Do not include DST file or support files referenced
    '     by the DST file (templates, blocks, ...)
    ' 1 - Include DST file, all files referenced by the
    '     DST file, its sheets and references
    If Not sheetSelSetFound Is Nothing Then
        tro.addSheetSelset(sheetSelSetFound, 0)
    End If

    ' Close the sheet set
    sheetSetManager.Close(sheetSetDatabase)

    ' Create the transmittal package
    ' Files are copied and resaved to the path specified by the destinationRoot property
    ' and the other settings of the TransmittalInfo object.
    tro.createTransmittalPackage()
End Sub
```

The following sample code demonstrates how to add all the sheets and resource files from a sheet set to a transmittal set with the `ArchiveSheetSet()` method.

```
' Custom command to create a transmittal package from a sheet set
<CommandMethod("eTransmitDSTFile")> _
Public Shared Sub eTransmitDSTFile()

    ' Create a transmittal operation
    Dim tro As TransmittalOperation = New TransmittalOperation()

    ' Setup the transmittal behavior
    Dim ti As TransmittalInfo = _
        TransInfo(tro.getTransmittalInfoInterface(), c_destinationRoot)

    ' Add a file notification handler
    Dim fileHandler As New MyTransFileNotifier
    tro.subscribeToAddFileNotification(fileHandler)
```

```
    ' Setup the progress meter for use with sheet selection sets, _
    ' sheet sets, and the creation of a transmittal package
    Dim pmHandler As New MyProgressHandler
    tro.setProgressMeter(pmHandler, 1)

    ' Add a sheet set and the drawings referenced in it
    ' Get a reference to the Sheet Set Manager object
    Dim sheetSetManager As IAcSmSheetSetMgr = New AcSmSheetSetMgr

    ' Open a sheet set (DST) file
    Dim sheetSetDatabase As AcSmDatabase = _
        sheetSetManager.OpenDatabase(c_dstFile, False)

    ' Get the sheet set from the database and process it
    ' 0 - Do not include DST file or support files referenced
    '     by the DST file (templates, blocks, ...)
    ' 1 - Include DST file, all files referenced by the DST file,
    '     its(sheets And references)
    tro.archiveSheetSet(sheetSetDatabase.GetSheetSet(), vbTrue)

    ' Close the sheet set
    sheetSetManager.Close(sheetSetDatabase)

    ' Remove the file notification handler
    tro.cancelSubscriptionToAddFileNotification(fileHandler)

    ' Remove the progress meter handler
    pmHandler = Nothing

    ' Create the transmittal package
    ' Files are copied and resaved to the path specified by the destinationRoot property
    ' and the other settings of the TransmittalInfo object.
    tro.createTransmittalPackage()
End Sub
```

The following custom function wraps the functionality of adding sheets and the sheets from a sheet selection set into a single function with some basic error handling. The function wraps the `archiveSheetSet()`, `addSheetSelset()`, and `addFile()` methods.

```
' Custom method for adding sheet components to a transmittal package
Private Shared Function TransAddSheetComponent(tro As TransmittalOperation, _
                                    filename As String, flag As FileType, _
                                    Optional tfParent As TransmittalFile = Nothing, _
                                    Optional sheetSetSelOrSheetName As String = "") _
                                As TransmittalFile

    ' Define a variable for the return value
    Dim tf As TransmittalFile = Nothing

    If System.IO.File.Exists(filename) = True Then
        ' Determine which type of file to add
        Select Case flag
            Case FileType.Sheets
                ' Add a sheet selection set from a sheet set and
```

```
            ' the drawings contained in it
            ' 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(filename, False)

            ' Get the enumerator for the objects in the
            ' sheet set and then set the first object
            Dim enumerator As IAcSmEnumPersist = sheetSetDatabase.GetEnumerator()
            Dim itemSheetSet As IAcSmPersist = enumerator.Next()

            Dim sheetNames(0) As String

            ' Semi-colon detected, multiple sheet names are being passed
            If sheetSetSelOrSheetName.Contains(";") = True Then
                sheetNames = sheetSetSelOrSheetName.Split(";")
            Else
                sheetNames(0) = sheetSetSelOrSheetName
            End If

            ' Create an array of the sheets based on their names
            Dim sheetArray(0) As IAcSmObjectId
            Dim sheet As AcSmSheet, nCount As Integer = 0

            ' Step through the objects in the sheet set
            Do While Not itemSheetSet Is Nothing

                ' Check to see if the object is a Sheet
                If itemSheetSet.GetTypeName() = "AcSmSheet" Then

                    ' A sheet was found, now to see if it is the one needed
                    sheet = itemSheetSet

                    ' Step through each name provided
                    For Each sheetName As String In sheetNames

                        ' Check to see if the sheet name matches the sheet
                        If UCase(sheet.GetName()). _
                            Equals(UCase(sheetName)) = True Then
                            ReDim Preserve sheetArray(0 To nCount)

                            ' Match was found, add the sheet's object Id to the array
                            sheetArray(nCount) = sheet.GetObjectId()

                            ' Increment the counter by 1
                            nCount = nCount + 1
                        End If
                    Next
                End If

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

```
            ' Get the sheets from the database and process them
            ' 0 - Do not include DST file or support files referenced
            '     by the DST file (templates, blocks, ...)
            ' 1 - Include DST file, all files referenced by the
            '     DST file, its sheets and references
            If sheetArray.Length > 0 And sheetArray(0).Equals("") = False Then
                tro.addSheets(sheetArray, vbTrue)
            End If

            ' Close the sheet set
            sheetSetManager.Close(sheetSetDatabase)

            ' No value to return, so return Nothing
            tf = Nothing
        Case FileType.SheetSetSel
            ' Add a sheet selection set from a sheet set
            ' and the drawings contained in it
            ' 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(filename, False)

            ' Get all the SheetSelSets in the sheet set
            Dim sheetSelSetEnum As IAcSmEnumSheetSelSet = _
                sheetSetDatabase.GetSheetSet().GetSheetSelSets().GetEnumerator()

            ' Get the first SheetSelSet
            Dim sheetSelSet As AcSmSheetSelSet = sheetSelSetEnum.Next()
            Dim sheetSelSetTemp As AcSmSheetSelSet = Nothing

            ' Check to the see if it is the requested SheetSelSet,
            ' if not get the next and check
            If Not sheetSelSet Is Nothing Then
                While Not sheetSelSet Is Nothing
                    If sheetSelSet.GetName().ToUpper() = _
                        sheetSetSelOrSheetName.ToUpper() Then
                        sheetSelSetTemp = sheetSelSet

                        Exit While
                    End If

                    sheetSelSet = sheetSelSetEnum.Next()
                End While

                ' Get the sheet set from the database and process it
                ' 0 - Do not include DST file or support files referenced
                '     by the DST file (templates, blocks, ...)
                ' 1 - Include DST file, all files referenced by the
                '     DST file, its sheets and references
                If Not sheetSelSetTemp Is Nothing Then
                    tro.addSheetSelset(sheetSelSetTemp, 0)
                End If
```

```
                End If

                ' Close the sheet set
                sheetSetManager.Close(sheetSetDatabase)

                ' No value to return, so return Nothing
                tf = Nothing
            Case FileType.SheetSet
                ' Add a sheet set and the drawings referenced in it
                ' Get a reference to the Sheet Set Manager object
                Dim sheetSetManager As IAcSmSheetSetMgr = New AcSmSheetSetMgr

                ' Open a sheet set (DST) file
                Dim sheetSetDatabase As AcSmDatabase = _
                    sheetSetManager.OpenDatabase(filename, False)

                ' Get the sheet set from the database and process it
                ' 0 - Do not include DST file or support files referenced
                '     by the DST file (templates, blocks, ...)
                ' 1 - Include DST file, all files referenced by the DST file,
                '     its(sheets And references)
                tro.archiveSheetSet(sheetSetDatabase.GetSheetSet(), vbTrue)

                ' Close the sheet set
                sheetSetManager.Close(sheetSetDatabase)

                ' No value to return, so return Nothing
                tf = Nothing
        End Select
    End If

    Return tf
End Function
```

Obtaining Information about Files in a Transmittal Set

After a file has been added to a transmittal set, information can be gained about each file in the transmittal set whether the file was added using the Transmittal Object library or automatically by the transmittal engine. All of the files in a transmittal set are represented by a structure known as a *files graph*. A files graph is a hierarchical structure that represents each file and its relationships to other files in the transmittal set.

The files graph of a transmittal set is represented by a `TransmittalFilesGraph` object. The `getTransmittalGraphInterface()` method is used to obtain the files graph of a transmittal set. You can use the `getAt()` method to step through each file in the files graph. The `getAt()` method returns a `TransmittalFile` object which can be used to get information about the file on disk and understand how many files are dependees or dependents of the file.

The following information can be obtained from a `TransmittalFile` object:

- Source and target paths
- Number of dependents and dependees
- Existence of the file
- Size, type, and version of the file

- Inclusion in the transmittal set
- Last modified date and time

Note: While working with the files graph, you can add and remove relationships between files in the transmittal set.

The following sample code demonstrates how to step through and list each file in the files graph of a transmittal set.

```
' Custom command that opens and accesses the dependents of the files in a transmittal set
<CommandMethod("FileDependents")> _
Public Shared Sub FileDependents()
    ' Get the current editor
    Dim acEditor As Editor = Application.DocumentManager.MdiActiveDocument.Editor

    ' Create a transmittal operation
    Dim tro As TransmittalOperation = New TransmittalOperation()

    ' Setup the transmittal behavior
    Dim ti As TransmittalInfo = TransInfo(tro.getTransmittalInfoInterface(), _
                                          c_destinationRoot)

    ' Define file to add to the transmittal
    Dim tf As TransmittalFile = Nothing
    Dim dwgFile As String = c_dwgFile

    ' Add file to transmittal and parse its information
    If tro.addDrawingFile(dwgFile, tf) = AddFileReturnVal.eFileAdded Then
        ' Get each of the referenced files in the file being added
        Dim tfg As TransmittalFilesGraph = tro.graphInterfacePtr()
        Dim rootTF As TransmittalFile = tfg.getRoot()

        Dim tfList As New List(Of TransmittalFile)()
        tfList.Add(rootTF)

        ' Step through each of the referenced files identified
        While tfList.Count > 0
            tf = tfList(0)
            tfList.RemoveAt(0)

            Dim numberOfDependents As Integer = tf.numberOfDependents
            For i As Integer = 0 To numberOfDependents - 1
                Dim childTF As TransmittalFile = tf.getDependent(i)
                tfList.Add(childTF)

                ' Output the filenames attached to the drawing
                Application.DocumentManager.MdiActiveDocument.Editor. _
                    WriteMessage(Environment.NewLine + childTF.sourcePath)

                ' See if the dependent file is a drawing or not
                If System.IO.Path.GetExtension( _
                    childTF.sourcePath).ToUpper() = ".DWG" Then

                    ' Open the drawing into memory
                    Dim acDb As New AxDbDocument
```

```
                    acDb.Open(childTF.sourcePath)

                    ' If the drawing is not null, display the
                    ' names of each layer in the drawing
                    If Not acDb Is Nothing Then
                        acEditor.WriteMessage(Environment.NewLine & "Layer Names:")

                        ' Step through each of the layers
                        For Each acLyr As AcadLayer In acDb.Layers
                            ' Output the names of each layer
                            acEditor.WriteMessage(Environment.NewLine & _
                                                  "  " & acLyr.Name)
                        Next
                    End If

                    ' Set the object to Nothing to close the drawing
                    acDb = Nothing
                End If
            Next
        End While
    End If

    ' Release the transmittal operation
    tro = Nothing
End Sub
```

The following sample code demonstrates how to list information about a drawing file after it has been added to a transmittal set.

```
' Custom command that displays information about a file
<CommandMethod("FileInfo")> _
Public Shared Sub FileInfo()
    ' Create a transmittal operation
    Dim tro As TransmittalOperation = New TransmittalOperation()

    ' Setup the transmittal behavior
    Dim ti As TransmittalInfo = TransInfo(tro.getTransmittalInfoInterface(), _
                                          c_destinationRoot)

    ' Define file to add to the transmittal
    Dim tf As TransmittalFile = Nothing
    Dim dwgFile As String = c_SampleFolder & "Sheet Sets\Architectural\A-01.dwg"

    ' Add file to transmittal and parse its information
    If tro.addDrawingFile(dwgFile, tf) = AddFileReturnVal.eFileAdded Then
        ' Get the current editor
        Dim acEditor As Editor = Application.DocumentManager.MdiActiveDocument.Editor

        ' Output information about the file in the transmittal set
        ' Source and full target path to the file
        acEditor.WriteMessage(Environment.NewLine & "Source Path: " & tf.sourcePath)
        acEditor.WriteMessage(Environment.NewLine & "Full Path: " & tf.fullPathForTarget)
        acEditor.WriteMessage(Environment.NewLine & "Target SubPath: " & _
                              tf.targetSubPath)
```

```
        ' See if the file is a drawing
        If System.IO.Path.GetExtension(tf.sourcePath).ToUpper() = ".DWG" Then
            ' Number of dependents and dependees
            acEditor.WriteMessage(Environment.NewLine & "# of Dependents: " & _
                                  tf.numberOfDependents.ToString())
            acEditor.WriteMessage(Environment.NewLine & "# of Dependees: " & _
                                  tf.numberOfDependees.ToString())

            ' Get the first dependent of the drawing
            If tf.numberOfDependents > 0 Then
                acEditor.WriteMessage(Environment.NewLine & _
                                      "Name of First Dependent: " & _
                                      tf.getDependent(0).sourcePath)
            Else
                acEditor.WriteMessage(Environment.NewLine & "No dependents")
            End If

            ' Get the first dependee of the drawing
            If tf.numberOfDependees > 0 Then
                acEditor.WriteMessage(Environment.NewLine & _
                                      "Name of First Dependee: " & _
                                      tf.getDependee(0).sourcePath)
            Else
                acEditor.WriteMessage(Environment.NewLine & "No dependees")
            End If
        End If

        ' Output file system information about the file
        acEditor.WriteMessage(Environment.NewLine & "File Exists: " & _
                              tf.fileExists.ToString())
        acEditor.WriteMessage(Environment.NewLine & "File Size: " & _
                              tf.fileSize.ToString())
        acEditor.WriteMessage(Environment.NewLine & "File Type: " & _
                              tf.FileType.ToString())
        acEditor.WriteMessage(Environment.NewLine & "Include in transmittal: " & _
                              tf.includeInTransmittal.ToString())
        acEditor.WriteMessage(Environment.NewLine & "Date/Time: " & _
                              tf.lastModifiedTime.ToLongDateString() & "  " & _
                              tf.lastModifiedTime.ToLongTimeString())
        acEditor.WriteMessage(Environment.NewLine & "Type: " & _
                              tf.type.ToString())
        acEditor.WriteMessage(Environment.NewLine & "Version: " & _
                              tf.version.ToString())
    End If

    ' Release the transmittal operation
    tro = Nothing
End Sub
```

The following sample code demonstrates basic manipulation of the files graph for a transmittal set.

```
' Custom command to work with the transmittal file graph
<CommandMethod("FilesGraph")> _
Public Shared Sub FilesGraph()

    ' Create a transmittal operation
```

```
    Dim tro As TransmittalOperation = New TransmittalOperation()

    ' Setup the transmittal behavior
    Dim ti As TransmittalInfo = _
        TransInfo(tro.getTransmittalInfoInterface(), c_destinationRoot)

    ' Get the root file of the transmittal file graph
    Dim tfRoot As TransmittalFile = tro.getTransmittalGraphInterface().getRoot()

    ' Add a drawing file to a transmittal set
    Dim tfFirst As TransmittalFile = Nothing
    tro.addDrawingFile(c_SampleFolder & "Sheet Sets\Architectural\A-01.dwg", tfFirst)

    ' Add a second drawing file to a transmittal set
    Dim tfSecond As TransmittalFile = Nothing
    tro.addDrawingFile(c_SampleFolder & "Sheet Sets\Architectural\A-02.dwg", tfSecond)

    ' Add a third file to a transmittal set
    Dim tfThird As TransmittalFile = Nothing
    tro.addFile(c_SampleFolder & "Sheet Sets\Architectural\IRD Addition.dwf", _
                tfRoot, vbTrue, tfThird)

    ' Display the name and path of the first file of the transmittal file graph
    MsgBox("File name and path: " & _
           tro.getTransmittalGraphInterface().getAt(0).sourcePath)

    ' Display the number of files in the transmittal file graph
    MsgBox("Number of Files in Graph: " & tro.getTransmittalGraphInterface(). _
                                          getNumberOfFiles(vbTrue, vbTrue).ToString())

    ' Display the number of dependees
    MsgBox("Number of Dependees: " & tfFirst.numberOfDependees.ToString())

    ' Create a reference from the first file to the newly added file
    tro.getTransmittalGraphInterface().addEdge(tfSecond, tfFirst)

    ' Display the number of dependees
    MsgBox("Number of Dependees: " & tfFirst.numberOfDependees.ToString())

    ' Remove a reference from the first file
    tro.getTransmittalGraphInterface().removeEdge(tfSecond, tfFirst)

    ' Display the number of dependees
    MsgBox("Number of Dependees: " & tfFirst.numberOfDependees.ToString())

    ' Resets the transmittal graph
    tro.getTransmittalGraphInterface().reset()

    ' Display the number of files in the transmittal file graph
    MsgBox("Number of Files in Graph: " & tro.getTransmittalGraphInterface(). _
                                          getNumberOfFiles(vbTrue, vbTrue).ToString())

    ' Release the transmittal operation
    tro = Nothing
End Sub
```

Creating a Transmittal Package

After a transmittal set has been created, its settings configured, and those files that should be part of the transmittal package added, you can create a transmittal package. The `createTransmittalPackage()` method is used to start the transmittal engine and create a transmittal package. When a request to create a transmittal package is made, the transmittal engine creates the folder structure for the transmittal package and it copies the files from their source locations to the locations defined by the transmittal set. The location and structure of the transmittal package is defined by the properties of the `TransmittalInfo` object.

Prior to creating the transmittal package, you can read the *transmittal report*. The transmittal report is a text string that contains general information about the files in the transmittal package. The text in the transmittal report isn't written to a text file nor is it added to the transmittal package with the Transmittal Object library. You can use the `getTransmittalReport()` method to get the contents of the transmittal report and add custom content to the report using the `addToReport()` method. After the transmittal report is generated, you can write it out to a text file and then add that file to the transmittal set before creating the transmittal package.

The following sample code demonstrates how to write and add the transmittal report to a transmittal set. The custom `writeReport()` function is used to write the contents of the transmittal report string to a file named *transmittal report.txt*.

```
' Custom command that creates a transmittal package and adds a transmittal report
<CommandMethod("eTransmitCustomReport")> _
Public Shared Sub eTransmitCustomReport()

    ' Create a transmittal operation
    Dim tro As TransmittalOperation = New TransmittalOperation()

    ' Setup the transmittal behavior
    Dim ti As TransmittalInfo = _
        TransInfo(tro.getTransmittalInfoInterface(), c_destinationRoot)

    ' Add a sheet set to the transmittal set
    TransAddSheetComponent(tro, c_dstFile, FileType.SheetSet)

    ' Append text to the transmittal report
    Dim nFirstSection As Integer = -1
    nFirstSection = tro.addToReport("Custom First Section:", nFirstSection)
    tro.addToReport(Environment.NewLine & "First line of first custom section.", _
                    nFirstSection)

    ' Append text to the transmittal report
    Dim nSecondSection As Integer = -1
    nSecondSection = tro.addToReport(Environment.NewLine & "Custom Second Section:", _
                                     nSecondSection)
    tro.addToReport(Environment.NewLine & "First line of second custom section.", _
                    nSecondSection)

    ' Append text to the first section
    tro.addToReport(Environment.NewLine & "Second line of first custom section.", _
                    nFirstSection)
```

```
    ' Write the transmittal report to the file "transmittal report.txt" in the
    ' destination root of the transmittal package
    writeReport(Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments) + _
                "\transmittal report.txt", tro.getTransmittalReport())

    ' Add the report to the transmittal package
    Dim tf As TransmittalFile = TransAddFile(tro, _
                Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments) + _
                "\transmittal report.txt", FileType.Unknown, Nothing)

    ' Change the path of the transmittal report
    tf.targetSubPath = ""

    ' Create the transmittal package
    ' Files are copied and resaved to the path specified by the destinationRoot property
    ' and the other settings of the TransmittalInfo object.
    tro.createTransmittalPackage()

    ' Release the transmittal operation
    tro = Nothing
End Sub
```

The following is the definitions of the custom functions used by the previous sample code.

```
' Write the report file to a text (TXT) file
Private Shared Sub writeReport(filename As String, reportText As String)
    On Error Resume Next

    ' Remove file it is exists
    If System.IO.File.Exists(filename) = True Then
        System.IO.File.Delete(filename)
    End If

    ' Create the text (TXT) file
    Dim sw As System.IO.StreamWriter = System.IO.File.CreateText(filename)
    sw.Write(reportText)
    sw.Flush()
    sw.Close()
End Sub
```

Getting Notified of Changes by the Transmittal Engine

The Transmittal Object library allows you to monitor the files in a transmittal set through the use of event handlers. The events available for monitoring files in a transmittal set are implemented by first overriding the event handler classes. The event handler classes used to monitor the files in a transmittal set are:

- **`TransmittalAddFileNotificationHandler`** – Contains event handlers to monitor when files are added to a transmittal set and the files graph of a transmittal set is created (beginning/ending)
- **`TransmittalDatabaseNotificationHandler`** – Contains event handlers used to monitor changes to drawing files as a result of the files being processed by the transmittal engine

- **`TransmittalSheetSetNotification`** – Contains event handlers used to monitor the processing of the sheets in a sheet set
- **`TransmittalProgressHandler`** – Contains event handlers used to monitor the progress of the files being processed for lengthy operations that occur as a result of the `archiveSheetSet()`, `addSheetSelset()`, and `createTransmittalPackage()` methods

Once the override event handler classes have been defined, those classes must be registered with the `TransmittalOperation` object. The methods used to register and enable the use of the event handlers are:

- **`subscribeToAddFileNotification()/cancelSubscriptionToAddFileNotification()`** – Registers/unregisters a `TransmittalAddFileNotificationHandler` class
- **`subscribeDatabaseNotification()/cancelDatabaseNotification()`** – Registers/unregisters a `TransmittalDatabaseNotificationHandler` class
- **`subscribeSheetSetNotification()/cancelSheetSetNotification()`** – Registers/unregisters a `ITransmittalSheetSetNotification` class
- **`setProgressMeter()`** – Registers a `TransmittalProgressHandler` class

The following sample code shows how to implement event handlers that are part of the Transmittal Object library. The event handlers can be used to monitor the usage and progress of your custom applications that use the Transmittal Object library and the AutoCAD ETRANSMIT/ARCHIVE commands.

```
' Global variables
Private Shared g_dbHandler As New MyTransDbNotifier
Private Shared g_fileHandler As New MyTransFileNotifier
Private Shared g_ssmHandler As New MyTransSSMNotifier
Private Shared g_tro As New TransmittalOperation

' Custom command that subscribes to eTransmit notifications
<CommandMethod("ETNotifications")> _
Public Shared Sub ETNotifications()
    ' Create multiple notification handlers
    Dim fileHandler As New MyTransFileNotifier
    Dim pmHandler As New MyProgressHandler
    Dim dbHandler As New MyTransDbNotifier
    Dim ssmHandler As New MyTransSSMNotifier

    ' Create a transmittal operation
    Dim tro As TransmittalOperation = New TransmittalOperation()

    ' Register the file, database, and sheet set notification handlers
    tro.subscribeToAddFileNotification(fileHandler)
    tro.subscribeDatabaseNotification(dbHandler)
    tro.subscribeSheetSetNotification(ssmHandler)

    ' Setup the progress meter handler
    tro.setProgressMeter(pmHandler, 3)

    ' Setup the transmittal behavior
    Dim ti As TransmittalInfo = TransInfo(tro.getTransmittalInfoInterface(), _
```

```
                                        c_destinationRoot)

    ' Add a DWG file to the transmittal set
    Dim tfRoot As TransmittalFile = TransAddFile(tro, c_dwgFile, FileType.DWG)

    ' Add a non-drawing file to the transmittal set
    Dim tf As TransmittalFile = TransAddFile(tro, c_otherfile1, FileType.Unknown, tfRoot)

    ' Write the transmittal report to the file "transmittal report.txt" in the
    ' destination root of the transmittal package
    writeReport(Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments) + _
                "\transmittal report.txt", tro.getTransmittalReport())

    ' Add the report to the transmittal package
    tf = TransAddFile(tro, _
                Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments) + _
                "\transmittal report.txt", FileType.Unknown, Nothing)

    ' Change the path of the transmittal report
    tf.targetSubPath = ""

    ' Create the transmittal package
    ' Files are copied and resaved to the path specified by the destinationRoot property
    ' and the other settings of the TransmittalInfo object.
    tro.createTransmittalPackage()

    ' Termninate the file, database, and sheet set notification handlers
    tro.cancelSubscriptionToAddFileNotification(fileHandler)
    tro.cancelDatabaseNotification(dbHandler)
    tro.cancelSheetSetNotification(ssmHandler)

    ' Release the transmittal operation
    tro = Nothing
End Sub

' Custom command that subscribes to eTransmit notifications
' Works with the transmittal API and ETRANSMIT/ARCHIVE commands
<CommandMethod("ETNotificationsStart")> _
Public Shared Sub ETNotificationsStart()
    ' Add the handlers
    g_tro.subscribeToAddFileNotification(g_fileHandler)
    g_tro.subscribeDatabaseNotification(g_dbHandler)
    g_tro.subscribeSheetSetNotification(g_ssmHandler)
End Sub

' Custom command that unsubscribes to eTransmit notifications
' Works with the transmittal API and ETRANSMIT/ARCHIVE commands
<CommandMethod("ETNotificationsEnd")> _
Public Shared Sub ETNotificationsEnd()
    ' Remove the handlers
    g_tro.cancelSubscriptionToAddFileNotification(g_fileHandler)
    g_tro.cancelDatabaseNotification(g_dbHandler)
    g_tro.cancelSheetSetNotification(g_ssmHandler)
End Sub
```

The following sample code shows how to implement custom classes that override the event handlers available with the Transmittal Object library. These classes are used in the previous sample code.

```
' Custom class used to monitor the processing of a drawing
Friend Class MyTransDbNotifier
    Implements TransmittalDatabaseNotificationHandler

    Public Function convertDatabase(pDatabase As Object, _
                        pFile As TRANSMITTALLib.TransmittalFile, _
                        pTransmit As TRANSMITTALLib.TransmittalOperation) _
                    As Integer _
            Implements TRANSMITTALLib. _
                ITransmittalDatabaseNotificationHandler.convertDatabase

        Application.DocumentManager.MdiActiveDocument.Editor. _
            WriteMessage(Environment.NewLine & "Database converting: " & _
                         pFile.sourcePath)

        Return 1
    End Function

    Public Function postDatabaseProcessing(pDatabase As Object, _
                        pFile As TRANSMITTALLib.TransmittalFile, _
                        pTransmit As TRANSMITTALLib.TransmittalOperation) _
                    As Integer _
            Implements TRANSMITTALLib. _
                ITransmittalDatabaseNotificationHandler.postDatabaseProcessing

        Application.DocumentManager.MdiActiveDocument.Editor. _
            WriteMessage(Environment.NewLine & "Database post processing: " & _
                         pFile.sourcePath)

        Return 1
    End Function

    Public Function preDatabaseProcessing(pDatabase As Object, _
                        pFile As TRANSMITTALLib.TransmittalFile, _
                        pTransmit As TRANSMITTALLib.TransmittalOperation) _
                    As Integer _
            Implements TRANSMITTALLib. _
                ITransmittalDatabaseNotificationHandler.preDatabaseProcessing

        Application.DocumentManager.MdiActiveDocument.Editor. _
            WriteMessage(Environment.NewLine & "Database preprocessing: " & _
                         pFile.sourcePath)

        Return 1
    End Function
End Class

' Custom class used to monitor files being added to the transmittal package
Friend Class MyTransFileNotifier
    Implements TransmittalAddFileNotificationHandler

    ' Used to get a notification when a drawing or
```

```
    ' file is added to the transmittal package
    Public Sub addFileNotificationHandler(pFile As TRANSMITTALLib.TransmittalFile, _
                   pTransmit As TRANSMITTALLib.TransmittalOperation) _
           Implements TRANSMITTALLib. _
               ITransmittalAddFileNotificationHandler.addFileNotificationHandler

        Application.DocumentManager.MdiActiveDocument.Editor. _
            WriteMessage(Environment.NewLine & _
                         "File added to transmittal package: " & pFile.sourcePath)
    End Sub

    Public Sub beginFilesGraphCreation(pTransmit As  _
                                       TRANSMITTALLib.TransmittalOperation) _
        Implements TRANSMITTALLib. _
           ITransmittalAddFileNotificationHandler.beginFilesGraphCreation

        Application.DocumentManager.MdiActiveDocument.Editor. _
            WriteMessage(Environment.NewLine & "File graph being created.")
    End Sub

    Public Sub endFilesGraphCreation(pTransmit As TRANSMITTALLib.TransmittalOperation) _
        Implements TRANSMITTALLib. _
           ITransmittalAddFileNotificationHandler.endFilesGraphCreation

        Application.DocumentManager.MdiActiveDocument.Editor. _
            WriteMessage(Environment.NewLine + "File graph creation ended.")
    End Sub
End Class

' Custom class used to monitor the processing of a sheet set
Friend Class MyTransSSMNotifier
    Implements ITransmittalSheetSetNotification

    Public Sub postSheetSetProcessing(pISheetSet As Object, _
                   pFile As TRANSMITTALLib.TransmittalFile, _
                   pTransmit As TRANSMITTALLib.TransmittalOperation) _
        Implements TRANSMITTALLib.ITransmittalSheetSetNotification.postSheetSetProcessing

        Application.DocumentManager.MdiActiveDocument.Editor. _
            WriteMessage(Environment.NewLine & "Sheet Set post processing: " & _
                         pFile.sourcePath)
    End Sub

    Public Sub preSheetSetProcessing(pISheetSet As Object, _
                   pFile As TRANSMITTALLib.TransmittalFile, _
                   pTransmit As TRANSMITTALLib.TransmittalOperation) _
        Implements TRANSMITTALLib.ITransmittalSheetSetNotification.preSheetSetProcessing

        Application.DocumentManager.MdiActiveDocument.Editor. _
            WriteMessage(Environment.NewLine & "Sheet Set preprocessing: " & _
                         pFile.sourcePath)
    End Sub
End Class

' Custom class used to monitor the progress of lengthy operations
```

```
Friend Class MyProgressHandler
    Implements TransmittalProgressHandler

    Public Sub isCanceled(ByRef cancelled As Integer) _
        Implements TRANSMITTALLib.ITransmittalProgressHandler.isCanceled

        Application.DocumentManager.MdiActiveDocument.Editor. _
            WriteMessage(Environment.NewLine & "Transmital process cancelled status")
    End Sub

    Public Sub processingFile(fname As String) _
        Implements TRANSMITTALLib.ITransmittalProgressHandler.processingFile

        Application.DocumentManager.MdiActiveDocument.Editor. _
            WriteMessage(Environment.NewLine & "    ProgressMeter: " & fname)
    End Sub
End Class

' Custom class used to monitor the processing of drawing files
Friend Class MyFileOpOverwrite
    Implements ITransmittalFileOperationOverride

    Public Sub Open(ByRef fileName As UShort, attemptToRecover As Integer, _
                    ByRef ppDb As System.IntPtr) _
        Implements ITransmittalFileOperationOverride.Open

        Application.DocumentManager.MdiActiveDocument.Editor. _
            WriteMessage(Environment.NewLine & "Operation Override: File Open")
    End Sub

    Public Sub Close(pDb As System.IntPtr) _
        Implements ITransmittalFileOperationOverride.Close

        Application.DocumentManager.MdiActiveDocument.Editor. _
            WriteMessage(Environment.NewLine & "Operation Override: File Close")
    End Sub

    Public Sub PreProcessHandling(digitalSignedFiles As System.Array, _
                                  ByRef isCancelled As Integer) _
        Implements ITransmittalFileOperationOverride.PreProcessHandling

        Application.DocumentManager.MdiActiveDocument.Editor. _
            WriteMessage(Environment.NewLine & "Operation Override: PreProcessing")
    End Sub
End Class
```

Emulating the ZIP and Send Email Functionality of the ETRANSMIT and ARCHIVE Commands

The Transmittal Object library doesn't contain support for zipping or emailing a transmittal package like the ETRANSMIT or ARCHIVE commands. Support for zipping and emailing can be implemented using other namespaces and third-party libraries. Here are a few examples of the namespaces and third-party libraries that can be used to add zip file and email support to your custom applications:

- **`System.IO.Packaging` namespace** – Used to create a ZIP file containing the files and folder structure of a transmittal package. The ZIP file also contains an XML manifest file that

lists the contents of the ZIP file; the namespace is part of the .NET Framework in the WindowsBase library.

- **DotNetZip library** – Used to create a ZIP file containing the files and folder structure from the transmittal package. The ZIP file does not contain an XML manifest file like that created with the `Packaging` namespace in the .NET Framework; available from CodePlex (*http://dotnetzip.codeplex.com/*).
- **`System.Net.Mail` namespace** – Used to send emails using SMTP; the namespace is part of the .NET Framework.
- **Outlook Application library** – Used to send emails using Microsoft Outlook; Outlook must be installed.

The following sample code demonstrates how to implement the DotNetZip library to zip up a transmittal package and email it using the `System.Net.Mail` namespace. The functionality to perform both tasks are wrapped in several custom functions.

```
' Required to work with and create ZIP packages
' ZIP packages do not preserve spaces in file names, so it is not
' completely ideal for AutoCAD files that might be used as references or contain
references.
Imports System.IO
Imports System.IO.Packaging

' Required to support the e-mail functionality
Imports System.Net.Mail

' Required to work with the DotNetZip library (http://dotnetzip.codeplex.com/)
Imports Ionic.Zip

' Custom command to create a transmittal package
<CommandMethod("eTransmitCustom")> _
Public Shared Sub eTransmitCustom()

    ' Create a transmittal operation
    Dim tro As TransmittalOperation = New TransmittalOperation()

    ' Setup the transmittal behavior
    Dim ti As TransmittalInfo = _
        TransInfo(tro.getTransmittalInfoInterface(), c_destinationRoot)

    ' Add a DWG file
    Dim tfRoot As TransmittalFile = TransAddFile(tro, c_dwgFile, FileType.DWG)

    ' Add a file notification handler
    Dim fileHandler As New MyTransFileNotifier
    tro.subscribeToAddFileNotification(fileHandler)

    ' Add another file other than a DWG file
    Dim tf As TransmittalFile = TransAddFile(tro, c_otherfile1, FileType.Unknown, tfRoot)

    ' Setup the progress meter for use with sheet selection sets, _
    ' sheet sets, and the creation of a transmittal package
    Dim pmHandler As New MyProgressHandler
    tro.setProgressMeter(pmHandler, 1)
```

```
' Add a sheet set
'TransAddSheetComponent(tro, c_dstFile, FileType.SheetSet)

' Add a sheet selection set
'TransAddSheetComponent(tro, c_dstFile, FileType.SheetSetSel, Nothing, "Elevations")

' Add individual sheets from a sheet set
TransAddSheetComponent(tro, c_dstFile, FileType.Sheets, _
                       Nothing, "T-01 TITLE SHEET;A-01 MAIN AND SECOND FLOOR PLAN")

' Append text to the transmittal report
Dim nFirstSection As Integer = -1
nFirstSection = tro.addToReport("Custom First Section:", nFirstSection)
tro.addToReport(Environment.NewLine & "First line of first custom section.", _
                nFirstSection)

' Append text to the transmittal report
Dim nSecondSection As Integer = -1
nSecondSection = tro.addToReport(Environment.NewLine & "Custom Second Section:", _
                                 nSecondSection)
tro.addToReport(Environment.NewLine & "First line of second custom section.", _
                nSecondSection)

' Append text to the first section
tro.addToReport(Environment.NewLine & "Second line of first custom section.", _
                nFirstSection)

' Write the report to the file report.txt in the transmittal package
' destination root and add it to the transmittal package
writeReport(Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments) + _
            "\transmittal report.txt", tro.getTransmittalReport())

tf = TransAddFile(tro, _
            Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments) + _
            "\transmittal report.txt", FileType.Unknown, tfRoot)

' Remove the file notification handler
tro.cancelSubscriptionToAddFileNotification(fileHandler)

' Place the report at the root of the transmittal package
tf.targetSubPath = ""

' Create the transmittal package
' Files are copied and resaved to the path specified by the destinationRoot property
' and the other settings of the TransmittalInfo object.
tro.createTransmittalPackage()

' Create a ZIP file of the files from the transmittal package
If g_createZip = 1 Then
    ' Use the Packaging namespace that is part of the .NET Framework
    'Package_ZipFile(ti.destinationRoot, ti.destinationRoot + _
    '                                    "Package_Zip_myzip.zip")

    ' Use the DotNetZip library
```

```
        Iconic_ZipFile(ti.destinationRoot, ti.destinationRoot + _
                                           "Ionic_Zip_myzip.zip")
    End If

    ' Send the ZIP file via an e-mail
    If g_sendZip = 1 Then
        'sendEmail(ti.destinationRoot + "Package_Zip_myzip.zip")
        sendEmail(ti.destinationRoot + "Ionic_Zip_myzip.zip")
    End If

    ' Release the transmittal operation
    tro = Nothing
End Sub
```

The following sample code is of custom functions that create a ZIP file using the DotNetZip library.

```
' Create ZIP using DotNetZip library
Private Shared Sub Iconic_ZipFile(dirToFiles As String, zipFilename As String)
    ' Create a new ZIP file
    Dim zipPack As ZipFile = New ZipFile(zipFilename)

    ' Start adding the specified directory to the ZIP file
    Iconic_AddDirToZip(zipPack, dirToFiles, dirToFiles, zipFilename)

    ' Close the ZIP file
    zipPack.Save()
End Sub

' Recurse the provided directory and add files to the ZIP file and any subdirectores
Private Shared Sub Iconic_AddDirToZip(zipPack As ZipFile, ByVal dir As String, _
                                      ByVal parentDir As String, zipFilename As String)
    ' Add the files from the directory
    Dim fileEntries As String() = Directory.GetFiles(dir)
    Dim fileName As String
    For Each fileName In fileEntries

        ' Make sure you are not adding the ZIP file to itself
        If zipFilename.CompareTo(fileName) <> 0 Then
            Iconic_AppendToZipFile(zipPack, fileName, parentDir, zipFilename)
        End If
    Next fileName

    ' Recurse the subdirectories
    Dim subdirs As String() = Directory.GetDirectories(dir)
    Dim subdir As String
    For Each subdir In subdirs
        Iconic_AddDirToZip(zipPack, subdir, parentDir, zipFilename)
    Next subdir
End Sub

' Add the file to the ZIP
Private Shared Sub Iconic_AppendToZipFile(zipPack As ZipFile, filename As String, _
                                          parentDir As String, zipFilename As String)
    ' Get relative path based on parent
    Dim filenameTemp As String = filename.Replace(parentDir, "")
```

```
    ' Append a forward slash "/" since the URI always starts at the root
    Dim partialURL As String = String.Concat("/", filenameTemp)

    ' Check to see if the file already exists in the ZIP,
    ' if so remove it before adding the new file
    Dim file As ZipEntry = zipPack.Item(partialURL)
    If IsNothing(file) = False Then
        zipPack.RemoveEntry(partialURL)
    End If

    ' Add the file to the ZIP file
    zipPack.AddFile(filename, Path.GetDirectoryName(partialURL))
End Sub
```

The following sample code is of custom functions that create a ZIP file using the `System.IO.Packaging` namespace.

```
' Create the ZIP file using System.IO.Packaging
Private Shared Sub Package_ZipFile(dirToFiles As String, zipFilename As String)

    ' Open the ZIP package if it exists, else create a new one
    Dim zipPack As Package = ZipPackage.Open(zipFilename, FileMode.OpenOrCreate, _
                                             FileAccess.ReadWrite)

    ' Start adding the specified directory to the ZIP package
    Package_AddDirToZip(zipPack, dirToFiles, dirToFiles, zipFilename)

    ' Close the ZIP package
    zipPack.Close()
End Sub

' Recurse the provided directory and add files to the ZIP package and any subdirectores
Private Shared Sub Package_AddDirToZip(zipPack As Package, ByVal dir As String, _
                                       ByVal parentDir As String, zipFilename As String)
    ' Add the files from the directory
    Dim fileEntries As String() = Directory.GetFiles(dir)
    Dim fileName As String
    For Each fileName In fileEntries

        ' Make sure you are not adding the ZIP package to itself
        If zipFilename.CompareTo(fileName) <> 0 Then
            Package_AppendToZipFile(zipPack, fileName, parentDir, zipFilename)
        End If
    Next fileName

    ' Recurse the subdirectories
    Dim subdirs As String() = Directory.GetDirectories(dir)
    Dim subdir As String
    For Each subdir In subdirs
        Package_AddDirToZip(zipPack, subdir, parentDir, zipFilename)
    Next subdir
End Sub

' Add the file to the ZIP package
Private Shared Sub Package_AppendToZipFile(zipPack As Package, filename As String, _
```

```
                                        parentDir As String, zipFilename As String)
    ' Get relative path based on parent
    Dim filenameTemp As String = filename.Replace(parentDir, "")

    ' Append a forward slash "/" since the URI always starts at the root
    Dim uriURL As String = String.Concat("/", filenameTemp)

    ' Create the new uri for the file being added
    Dim partUri As Uri = PackUriHelper.CreatePartUri(New Uri(uriURL, UriKind.Relative))

    ' Check to see if the file already exists in the ZIP package,
    ' if so remove it before adding the new file
    If zipPack.PartExists(partUri) = True Then
        zipPack.DeletePart(partUri)
    End If

    ' Create the PackagePart related to the file being added
    Dim part As PackagePart = zipPack.CreatePart(partUri, _
                    System.Net.Mime.MediaTypeNames.Application.Zip, _
                    CompressionOption.Normal)

    ' Read and write the bytes of the file to the ZIP package
    Dim bytes() As Byte = File.ReadAllBytes(filename)
    part.GetStream().Write(bytes, 0, bytes.Length)
End Sub
```

The following sample code is of a custom function that emails a ZIP file using the `System.Net.Mail` namespace.

```
' Base function to define and send an e-mail message
Private Shared Sub sendEmail(fileAttachment As String)
    ' Define the mail message and attach the ZIP file
    Dim message As MailMessage = New MailMessage()
    message.To.Add(c_EmailTo)
    message.Subject = "This is the Subject line"
    message.From = New MailAddress(c_UserName)
    message.Body = "This is the message body"
    message.Attachments.Add(New Attachment(fileAttachment))

    ' Define the mail client
    Dim client As SmtpClient = New SmtpClient()
    client.Host = "smtp.gmail.com"
    client.Port = 587
    client.EnableSsl = True
    client.UseDefaultCredentials = False
    client.Credentials = New System.Net.NetworkCredential(c_UserName, c_Password)
    client.Send(message)
End Sub
```

8 Where to Get More Information

When you are first starting to learn a new skill, you will have questions and where you go to find those answers might not be clear. The following is a list of resources that you can use to get help:

- **AutoCAD Online Help System** – The Developer Home page in the AutoCAD Online Help system contains a lot of information on working with the COM APIs covered in this session. You can find reference topics related to the classes exposed by the APIs along with topics that explain how to work with the APIs. To access the online help, go to: *http://help.autodesk.com/view/ACD/2016/ENU/files/homepage_dev.htm*.
- **ObjectARX SDK** – While it is named the ObjectARX SDK, it contains many samples for the AutoCAD Managed .NET API. For information on the ObjectARX SDK, see the URL *http://www.autodesk.com/objectarx*.
- **Beyond the UI (Blog)** – My personal blog that I post AutoCAD developer related topics. You can find my blog at *http://hyperpics.blogs.com/*.
- **Through the Interface (Blog)** – Kean Walmsley (SW Architect), offers a wide range of information on using the various programming APIs that are available for use with AutoCAD. Visit Kean's blog at *http://through-the-interface.typepad.com*.
- **AutoCAD DevBlog** – The AutoCAD Developers blog covers a wide range of topics using all of the programming languages that the AutoCAD program supports. You can find the AutoCAD Developer blog at *http://adndevblog.typepad.com/autocad/*.
- **Autodesk Discussion Forums** – The Autodesk forums provide peer-to-peer networking that allows you to ask a question about anything related to AutoCAD and get a response from a fellow user or Autodesk employee. To access the Autodesk discussion forums, go to *http://forums.autodesk.com*, click Forums near the top of the page and then click AutoCAD. Click the appropriate subgroup link.
- **AUGI Forums** – The AUGI forums provide peer-to-peer networking where you can ask questions about virtually anything in AutoCAD and get a response from a fellow user. Visit AUGI at *http://www.augi.com/*.
- **Industry Events and Classes** – Industry events such as CAD Midwest Tech Expo and Autodesk University can be great venues to learn about features in an Autodesk product. Along with industry events, you might also be able to find classes at a local technical college/university or an Autodesk Authorized Training Center (ATC).
- **Internet** – There are tutorials on the Internet to learn many of the customization and programming options that are supported in the AutoCAD program. Use your favorite search engine, such as Google or Bing and search on the topic of interest.
- **Books** – There are many general and specialized books cover AutoCAD customization and programming. To find a book, use *amazon.com* or *barnesandnoble.com* to locate a book online or visit your local Barnes and Noble store.