Autodesk University 2007

Introduction to the AutoCAD MEP .NET API

Martin Schmid, P.E. – Autodesk

ME104-2L This lab will provide some practical examples of using the AutoCAD MEP .NET API to configure drawing and environment settings, as well as accessing element data for design analysis.

About the Speaker:
Before accepting the role of Project Consultant at Autodesk, Martin worked as a mechanical and electrical systems designer and, most recently, as an engineering coordinator. Martin has been using AutoCAD since Release 10, and is fluent in the customization of Autodesk products, including AutoCAD Architecture, AutoCAD MEP .NET, and Revit MEP using .NET.

Martin has worked closely with both the AutoCAD MEP and the Revit MEP development team to share insights gained while consulting with customers, and has developed custom solutions for customers built on AutoCAD Revit MEP Suite to streamline documentation and analysis workflows.

martin.schmid@autodesk.com

Autodesk®

Table of Contents

Introduction

In this lab, we will look at what it takes to quickly get started to build a .NET API application for AutoCAD MEP. Three separate applications will be developed, one focusing on drawing settings, one focusing on piping components, one focusing on electrical components. The concepts from the piping sample are applicable to duct, conduit, as well as cable tray. The concepts in the electrical example are applicable to MvParts. The samples in this class will utilize C#; there are C#<->VB translators available on the web if you prefer VB.

This class assumes some familiarity with:

- Visual Studio 2005 or Visual Studio Express
- Microsoft .NET Framework 2.0
- AutoCAD .NET API

This is a 'Power User' class, extensive knowledge of AutoCAD MEP is assumed. This is NOT an intro to .NET, Visual Studio, or the AutoCAD .NET API… each of those topics could easily be a class (or course) themselves. The purpose of this class is to provide the knowledge to be able to successfully build an AutoCAD MEP .NET application from scratch, by using the provided sample code and the instructions contained herein.

Note that there are additional samples that come with AutoCAD MEP. Armed with the information in this class, you should be able to successfully run those samples, and learn more about the AutoCAD MEP .NET API using your language of choice.
C:\Program Files\AutoCAD MEP 2008\Sample\CS.NET
C:\Program Files\AutoCAD MEP 2008\Sample\VB.NET

In many cases, the provided sample code omits error checking for the sake of brevity.

The Hello Project

To get a quick intro to setting up a project, you will create a sample project in the following sections. These steps will demonstrate how to setup an AutoCAD MEP C#.NET Project. Further, we will create an AutoCAD script file that will run every time we start debugging. This script will load our .NET assembly .dll file for us. Additionally, the script will open a .dwg file for us… because it is very handy to have a drawing that already has information in it to test our .NET project against, instead of having to create the drawing each time, or load it manually.

Creating a New Project in Visual Studio 2005

1. From the File menu, select New > Project.
2. In the Project types list, select Visual C# > Windows
   Depending on your Visual Studio configuration, Visual C# may show up under Other Languages.

3. In the Templates list, select Class Library.
4. Specify
    a. Project Name: Project
    b. Location: C:\ME104-2L
    c. Solution Name: Solution
5. Click OK.

Creating a New Project in Visual C# 2005 Express Edition

1. From the File menu, select New Project.
2. In the New Project, select Class Library.
3. Specify the name ME104-2L, then click OK.
4. From the File menu, select Save All
5. Specify
    a. Project Name: Project
    b. Location: C:\ME104-2L
    c. Solution Name: Solution
6. Click Save.

Defining References for a Project

When developing within the .NET framework, references are utilized to access classes within assemblies. To work with AutoCAD, AutoCAD Architecture, and AutoCAD MEP functionality, it is necessary to reference the managed assemblies from the AutoCAD MEP installation.

1. In the Visual Studio Solution Explorer, right click on the project name, and select Add Reference…
2. Select the Browse tab.
3. Browse to the AutoCAD MEP installation folder, the default is:
   C:\Program Files\AutoCAD MEP 2008
4. In the File Name box, enter:
   *mgd*dll
5. Select all the .dll files in the list using Shift+Click: acdbmgd.dll through AecStructureMgd.dll
   You may not need all the assemblies for a particular project, but by selecting them all, you can ensure that all class assemblies are available.
6. Click OK.
7. In the Solution Explorer, under References, select acdbmgd through AecStructureMgd, and in the Properties section, set Copy Local to False.

Configure Debugging in Visual Studio 2005

Visual Studio can launch AutoCAD MEP for interactive debugging. You can use a script to automatically load the latest build of your .NET assembly, and automatically open test .dwg files if so desired.

1. In the Visual Studio Solution Explorer, right click on the project name, and select Properties.
2. Click the Debug tab.
3. Set the Start Action to Start external program, and set the program to:
   C:\Program Files\AutoCAD MEP 2008\acad.exe
4. Under Start Options, set the Command line arguments to:
   /ld "C:\Program Files\AutoCAD MEP 2008\AecBase.dbx" /b
   "C:\ME104_2L\Solution\Project\Test.scr"
5. Set the Working Directory to:
   C:\Program Files\AutoCAD MEP 2008\UserDataCache

Note, the C:\ME104_2L\ME104_2L\ME104_2L\ path may seem a bit odd… and it is. The folder structure is built up as follows:

<drive>:\<development folder>\<solution name>\<project name>

As your .NET development progresses, your work may be built into a series of Solutions that each contains multiple Projects. For this class, everything will be in one project.

Configure Debugging in Visual C# 2005 Express Edition

The Express Edition doesn't allow you to configure debugging within the Studio directly; however, you can configure debugging by manually defining a .csproj.user file:

1. In the Solution Explorer, right click on the project name ME104_2L, and select Add > New Item…
2. Select Text File, specify the name ME104_2L.csproj.user
3. Enter the following XML text, making changes as necessary for your path and project info (note: some lines wrap, I've shown ↵ where you should place actual returns.

```
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">↵
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">↵
    <StartAction>Program</StartAction>↵
    <StartProgram>C:\Program Files\AutoCAD MEP 2008\acad.exe</StartProgram>↵
    <StartWorkingDirectory>C:\Program Files\AutoCAD MEP
2008\UserDataCache\</StartWorkingDirectory>↵
    <StartArguments>/ld "C:\Program Files\AutoCAD MEP 2008\AecBase.dbx" /b
"C:\ME104_2L\ME104_2L\ME104_2L\Test.scr"</StartArguments>↵
  </PropertyGroup>↵
</Project>↵
```

4. Select File > Close Solution, and save all modified files.
5. Reopen the project from the Recent Projects list.
6. From the Debug menu, select Start Debugging.

This should launch AutoCAD MEP from the path specified in the .csproj.user file. You should get a message that the test.scr file cannot be found (because we haven't created it yet).

7. Click OK.
8. Since we're going to use this file later, go ahead and draw some pipes, fittings, valves, mvparts, etc…
9. From the AutoCAD File menu, select Save As…
10. Browse to C:\ME104_2L\ME104_2L\ME104_2L
11. Save the empty file as Test.dwg.
12. Close AutoCAD.

Note: Anytime you make a change to the .csproj.user file, you will have to close and reopen the solution for the change to take effect.

Sample Script File

As indicated above, a script may be used to automatically load .NET assemblies and .dwg files for testing.

1. In the Solution Explorer, right click on the project name ME104_2L, and select Add > New Item…
2. Select Text File, specify the name Test.scr
3. Enter the following:

```
(command "netload"
"C:\\ME104_2L\\ME104_2L\\ME104_2L\\bin\\Debug\\ME104_2L.dll")↵
open↵
"C:\ME104_2L\ME104_2L\ME104_2L\Test.dwg"↵
```

Note: The lines above word-wrapped in this document. The hard Returns have been indicated above with ↵.

4. From the Debug menu, select Start Debugging.

AutoCAD should launch, the script should run, and load the Test.dwg file.

Create and Test a .NET Command

We will create a simple command to demonstrate that the .dll file is actually being loaded.

1. In the Visual Studio Project Explorer, open the Class1.cs file by double clicking on it.

2. Above the namespace line, add the following two lines:

```
using AcadApp = Autodesk.AutoCAD.ApplicationServices.Application;
using Autodesk.AutoCAD.EditorInput;
```

3. Inside the class definition brackets, add the following lines:

```
[Autodesk.AutoCAD.Runtime.CommandMethod("Hello")]
public void HelloCommand()
{
    Editor ed = AcadApp.DocumentManager.MdiActiveDocument.Editor;
    ed.WriteMessage("Hello!");
}
```

4. The entire text of the Class1.cs file should be as follows:

```
using System;
using System.Collections.Generic;
using System.Text;

using AcadApp = Autodesk.AutoCAD.ApplicationServices.Application;
using Autodesk.AutoCAD.EditorInput;

namespace ME104_2L
{
    public class Class1
    {
        [Autodesk.AutoCAD.Runtime.CommandMethod("Hello")]
        public void HelloCommand()
        {
            Editor ed = AcadApp.DocumentManager.MdiActiveDocument.Editor;
            ed.WriteMessage("Hello!");
        }
    }
}
```

5. From the Debug menu, select Start Debugging

   AutoCAD MEP should launch, and the Test.dwg file should automatically open based on the .scr file defined in the previous section.

6. At the Command prompt, enter:

   Hello↵

   You should see Hello! at the command line.

Now that you have the basics of setting up a AutoCAD MEP .NET project in Visual Studio, we will now do something a little more useful, by configuring a couple of drawing settings using a command.

Modifying Drawing Settings

Let's take a look at some sample code that makes some changes to the current drawing's settings. In this sample we'll modify the Pipe and Duct insulation settings, the Hidden (Haloed) line gap, and the disconnect marker visibility.

Basically, what we need to do is get a reference to the current drawing, known in the code as a Database. Once we have the database, we start a transaction, and get the AutoCAD MEP settings, also known in the code as BuildingDBVariables.

With the BuildingDBVariables, we can modify the settings as desired, then Commit the transaction to 'save' the changes to the drawing (this is not the same as saving the file, i.e., this doesn't save the file, it just tells the Transaction that we're done, and want to retain our changes).

```
[Autodesk.AutoCAD.Runtime.CommandMethod("Setup")]
public void Setup()
{
  Autodesk.AutoCAD.DatabaseServices.Database db =
Autodesk.AutoCAD.DatabaseServices.HostApplicationServices.WorkingDatabase;
  Autodesk.AutoCAD.DatabaseServices.ObjectId oid =
Autodesk.Aec.Building.ApplicationServices.BuildingDBVariables.GetInstance(db, false);
  using (Autodesk.AutoCAD.DatabaseServices.Transaction trans =
db.TransactionManager.StartTransaction())
  {
    Autodesk.Aec.Building.ApplicationServices.BuildingDBVariables dbv = trans.GetObject(oid,
Autodesk.AutoCAD.DatabaseServices.OpenMode.ForWrite) as
Autodesk.Aec.Building.ApplicationServices.BuildingDBVariables;

      // DuctPreferences
      dbv.DuctInsulationThickness = 2;
      dbv.EnableDuctInsulation = true;

      // Options > MEP Display Control > B - Gap Width
      dbv.HaloedLineGap = 4;

      // PipePreferences
      dbv.PipeInsulationThickness = 1;
      dbv.EnablePipeInsulation = false;

      // View > Show Disconnect markers
      dbv.ShowBrokenConnectionMarkers = true;

     // Options > MEP Layout Rules > Collision Detection > Alert
      dbv.InterferenceDetection = true;
```

```
            trans.Commit();
        }
    }
```

Calculating Pipe System Volume

Now that we have seen some basics of how to modify settings in the drawing, we will look at getting information about the elements in our drawing. We will now work on a solution to address a question I came across on the Autodesk Discussion Groups (http://discussion.autodesk.com/thread.jspa?threadID=603868). The question was how to come up with a schedule that would itemize the total pipe length for each size of pipe. Additionally, this addresses a common request to tabulate the total system volume.

The following sample demonstrates how to extract this information from a pipe system. The user is prompted to select a pipe, and then every interconnected component is inspected. If it is a Pipe object, the volume of the pipe is approximated. Valves, Pipe Fittings, and MvParts may exist in the pipe network, but for simplicity, their volumes are not considered.

Classes

As this is a little more complex than the 'Hello' project, it is structured a bit differently. Instead of one file (Class1.cs), it is broken up into three Classes, each in its own file.

1. Right click on the folder, and select Add > Class…
2. Select Class, and enter the name Utilities.cs
3. Repeat the above to create Selection.cs, and GetPipeSystemVolumeCommand.cs

Utilities.cs – Defines a couple of access functions for commonly used AutoCAD .NET API statements, and a couple of conversion factors.

Selection.cs – Provides the selection functionality, and recursively iterates to find connected components.

GetPipeSystemVolumeCommand.cs – Defines the command to extract the pipe length and volume info.

The full code for these classes is attached to the end of this document.

Add the Command Method

Additionally, we need to modify the Class1.cs file to add our command.

```
[Autodesk.AutoCAD.Runtime.CommandMethod("GetPipeSystemVolume")]
public void GetPipeSystemVolume()
{
```

```
    GetPipeSystemVolumeCommand.Execute();
}
```

Test and Debug

Now, we should be able to start debugging, and test our command.

1. From the Debug menu, select Start Debugging.
   Note the Test.dwg file has opened, and you should see your earlier work.
2. Enter the command GetPipeSystemVolume.
3. Pick on a Pipe, and you should see the result of the calculation:

```
Command: GetPipeSystemVolume
Select a pipe:
64.6' of 8" pipe
53.4' of 2" pipe
300.7' of 0.75" pipe
179.0' of 20" pipe
Total Volume in Gallons: 3105.67
```

Your results will likely be different.

Summary

Note that this calculation is approximate based on the pipe segments only, and it is using the nominal pipe diameter. By default, the AutoCAD MEP pipe catalogs do not have interior diameter information for the various pipe classes and materials. Rest assured, however, that it is possible to add this information to the pipe catalog, and access it from the API to get more accurate pipe volume calculations if necessary.

From this point, you can use the AutoCAD .NET API to generate a table object, and insert the pipe length and volume information directly in the drawing. Since that is not specific to the AutoCAD MEP .NET API, it is outside the scope of this class.

Checking Electrical Connections

In this example, we will check electrical devices to automate the process of checking drawings. We will check to make sure that each connection meets the following criteria:

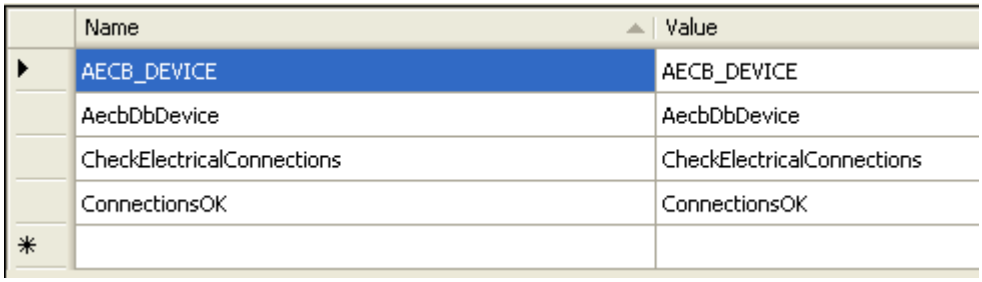
1. Verify that all device connectors are connected to a circuit.
2. Verify that all connectors have a load assigned.
3. Voltage and number of poles of all device connectors match the voltage and number of poles of the circuit to which they are connected.

4. Verify that Circuit is in Circuit Database.

Add Constants

In this sample, we're going to be creating a Property Set Definition, and selecting all Device objects. To help with these tasks, we're going to create some constants in Visual Studio. This will help eliminate string literals in our code, and make it easier to maintain.

1. Right click on the ME104_2L project name, and select Add > New Item…
2. Select Resources File.
3. Enter the name Constants, and then click Add.
4. Enter the name/value pairs as you see here:



Name	Value
AECB_DEVICE	AECB_DEVICE
AecbDbDevice	AecbDbDevice
CheckElectricalConnections	CheckElectricalConnections
ConnectionsOK	ConnectionsOK

Add the Command Method

There is another method added to Selection.cs called GetAllOfType().

Add command to Class1.cs:

```
[Autodesk.AutoCAD.Runtime.CommandMethod("CheckDeviceConnections")]
public void CheckDeviceConnections()
{
   ElectricalDeviceConnectionCheckerCommand.Execute();
}
```

Test the Electrical Connection Checker

1. Start Debug
2. Place a few devices (i.e., light fixtures, receptacles, etc.)
3. Place a panel (create circuits).
4. Connect *some* of the devices.
5. Assign load to *some* of the devices (some that are circuited, and some that are not circuited).
6. Save the file (just in case something is not quite right in your code, and a crash ensues).
7. Enter the command "CheckDeviceConnections"

On Your Own

Note that you can now use a Display Theme to visually show connected vs. disconnected devices. Alternatively, you can create a schedule. Or, you may want to create a user interface that lists the disconnected devices, and lets you pick on an element in the list, which will then zoom you to the disconnected device.

References

The following are books that I have recently read, and would recommend:

Professional C# 2005
http://www.wrox.com/WileyCDA/WroxTitle/productCd-0764575341.html

Code Complete, 2nd Edition
http://www.cc2e.com/

Pro .NET 2.0 Windows Forms and Custom Controls in C#
http://www.apress.com/book/view/1590594398

Professional .NET 2.0 Generics
http://www.wrox.com/WileyCDA/WroxTitle/productCd-0764559885.html

Questions / Notes

Sample Code

```
using System;
using System.Collections.Generic;
using System.Text;
using AcadApp = Autodesk.AutoCAD.ApplicationServices.Application;
using Autodesk.AutoCAD.EditorInput;

namespace ME104_2L
{
    public class Class1
    {
        [Autodesk.AutoCAD.Runtime.CommandMethod("Hello")]
        public void HelloCommand()
        {
            Editor ed = AcadApp.DocumentManager.MdiActiveDocument.Editor;
            ed.WriteMessage("Hello!");
        }

        [Autodesk.AutoCAD.Runtime.CommandMethod("Setup")]
        public void Setup()
        {
            Autodesk.AutoCAD.DatabaseServices.Database db = Autodesk.AutoCAD.
    DatabaseServices.HostApplicationServices.WorkingDatabase;
            Autodesk.AutoCAD.DatabaseServices.ObjectId oid = Autodesk.Aec.Building.
    ApplicationServices.BuildingDBVariables.GetInstance(db, false);
            using (Autodesk.AutoCAD.DatabaseServices.Transaction trans = db.
    TransactionManager.StartTransaction())
            {
                Autodesk.Aec.Building.ApplicationServices.BuildingDBVariables dbv = trans.
    GetObject(oid, Autodesk.AutoCAD.DatabaseServices.OpenMode.ForWrite) as Autodesk.Aec.
    Building.ApplicationServices.BuildingDBVariables;

                // DuctPreferences
                dbv.DuctInsulationThickness = 2;
                dbv.EnableDuctInsulation = true;

                // Options > MEP Display Control > B - Gap Width
                dbv.HaloedLineGap = 4;

                // PipePreferences
                dbv.PipeInsulationThickness = 1;
                dbv.EnablePipeInsulation = false;

                // View > Show Disconnect markers
                dbv.ShowBrokenConnectionMarkers = true;

                // Options > MEP Layout Rules > Collision Detection > Alert
                dbv.InterferenceDetection = true;

                trans.Commit();
            }
        }

        [Autodesk.AutoCAD.Runtime.CommandMethod("GetPipeSystemVolume")]
        public void GetPipeSystemVolume()
        {
            GetPipeSystemVolumeCommand.Execute();
        }


        [Autodesk.AutoCAD.Runtime.CommandMethod("CheckDeviceConnections")]
        public void CheckDeviceConnections()
        {
            ElectricalDeviceConnectionCheckerCommand.Execute();
        }
    }
}
```

```
using System.Collections.Generic;
using System.Diagnostics;

// AutoCAD Specific
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;

// MEP Specifics
using Autodesk.Aec.Building.DatabaseServices;
using Autodesk.Aec.Building.Elec.DatabaseServices;
using Autodesk.Aec.Building.ApplicationServices;

// AutoCAD Architecture Specific
using Autodesk.Aec.PropertyData.DatabaseServices;

// create an alias to the AutoCAD Application object
using AcadApp = Autodesk.AutoCAD.ApplicationServices.Application;

namespace ME104_2L
{
    class ElectricalDeviceConnectionCheckerCommand
    {
        static Transaction m_trans;
        static ObjectIdCollection m_uncircuitedList;
        static ObjectIdCollection m_voltsPolesMismatchList;
        static ObjectIdCollection m_noLoadList;
        static ObjectIdCollection m_circuitNotFoundInProjectDatabaseList;

        enum ConnectionProperty { Null =-3, ByCircuit = -2, Undefined = -1 };

        public static void Execute()
        {

            m_uncircuitedList = new ObjectIdCollection();
            m_voltsPolesMismatchList = new ObjectIdCollection();
            m_noLoadList = new ObjectIdCollection();
            m_circuitNotFoundInProjectDatabaseList = new ObjectIdCollection();

            ObjectId psdId = PropertySetHelpers.                                    ↙
    CreateCheckElectricalConnectionsPropertySet();

            ObjectIdCollection devIdColl = Selection.GetAllOfType(Constants.AECB_DEVICE);

            Database db = HostApplicationServices.WorkingDatabase;
            using (m_trans = db.TransactionManager.StartTransaction())
            {
                foreach (ObjectId devId in devIdColl)
                {
                    Device dev = m_trans.GetObject(devId, OpenMode.ForRead) as Device;
                    DeviceStyle devStyle = m_trans.GetObject(dev.StyleId, OpenMode.     ↙
    ForRead) as DeviceStyle;

                    int connectionCount = devStyle.MultiViewPartConnectors.Count;
                    CheckDeviceConnections(dev, connectionCount);
                    CheckDeviceConnectionLoads(dev, connectionCount);
                }
                m_trans.Commit();
            }

            ObjectIdCollection connectionsNotOk = null;
            ObjectIdCollection connectionsOk = null;
            GenerateConnectionStatusLists(devIdColl, ref connectionsNotOk, ref          ↙
    connectionsOk);
            SetDevicePropertySetInfo(psdId, connectionsOk, connectionsNotOk);

        }
```

```
        private static void GenerateConnectionStatusLists(ObjectIdCollection devIdColl,
    ref ObjectIdCollection connectionsNotOk, ref ObjectIdCollection connectionsOk)
        {
            connectionsNotOk = new ObjectIdCollection();
            foreach (ObjectId oid in m_uncircuitedList)
            {
                if (!connectionsNotOk.Contains(oid))
                {
                    connectionsNotOk.Add(oid);
                }
            }

            foreach (ObjectId oid in m_voltsPolesMismatchList)
            {
                if (!connectionsNotOk.Contains(oid))
                {
                    connectionsNotOk.Add(oid);
                }
            }

            foreach (ObjectId oid in m_noLoadList)
            {
                if (!connectionsNotOk.Contains(oid))
                {
                    connectionsNotOk.Add(oid);
                }
            }

            foreach (ObjectId oid in m_circuitNotFoundInProjectDatabaseList)
            {
                if (!connectionsNotOk.Contains(oid))
                {
                    connectionsNotOk.Add(oid);
                }
            }

            connectionsOk = new ObjectIdCollection();
            {
                foreach (ObjectId oid in devIdColl)
                {
                    if (!connectionsNotOk.Contains(oid) && !connectionsOk.Contains(oid))
                    {
                        connectionsOk.Add(oid);
                    }
                }
            }
        }

        private static void SetDevicePropertySetInfo(ObjectId psdId, ObjectIdCollection
    connectionsOk, ObjectIdCollection connectionsNotOk)
        {
            Database db = HostApplicationServices.WorkingDatabase;

            using (Transaction trans = db.TransactionManager.StartTransaction())
            {
                foreach (ObjectId oid in connectionsOk)
                {
                    ObjectId psId = PropertySetHelpers.GetPropertySet(oid, psdId);
                    PropertySet ps = trans.GetObject(psId, OpenMode.ForWrite) as
    PropertySet;
                    int id = ps.PropertyNameToId(Constants.ConnectionsOK);
                    ps.SetAt(id, true);
                }
                foreach (ObjectId oid in connectionsNotOk)
                {
                    ObjectId psId = PropertySetHelpers.GetPropertySet(oid, psdId);
```

```
                    PropertySet ps = trans.GetObject(psId, OpenMode.ForWrite) as
PropertySet;
                    int id = ps.PropertyNameToId(Constants.ConnectionsOK);
                    ps.SetAt(id, false);
                }
                trans.Commit();
            }
        }



        private static void CheckDeviceConnectionLoads(Device dev, int connectionCount)
        {
            ConnectionComponentMemberCollection ccmc = dev.ConnectionComponentMembers;

            for (int indx = 1; indx <= connectionCount; indx++)
            {
                ConnectionComponentMember currentConnection = dev.
ConnectionComponentMembers[indx - 1];

                if (currentConnection.SystemType == SystemType.PowerAndLightingSystemType)
                {
                    double load = GetConnectorPropertyDouble(dev, indx, Context.
ConnectionPropertyLoad);
                    if (load <= 0)
                    {
                        m_noLoadList.Add(dev.ObjectId);
                    }
                }

            }
        }


        private static void CheckDeviceConnections(Device dev, int connectionCount)
        {
            ConnectionComponentMemberCollection ccmc = dev.ConnectionComponentMembers;

            for (int indx = 1; indx <= connectionCount; indx++)
            {
                Circuit circuit = CheckConnectorCircuit(dev, indx);
                if (circuit != null)
                {
                    ConnectionComponentMember currentConnection = dev.
ConnectionComponentMembers[indx - 1];

                    if (currentConnection.SystemType == SystemType.
PowerAndLightingSystemType)
                    {
                        CheckVoltsAndPoles(circuit, dev, indx);
                    }
                }
            }
        }

        private static void CheckVoltsAndPoles(Circuit circuit, Device dev, int indx)
        {
            bool voltageOk = false;
            bool polesOk = false;

            double circuitVoltage = circuit.GetPropertyDouble(Context.
ConnectionPropertyVoltage);
            int circuitNumberOfPoles = circuit.GetPropertyInt(Context.
ConnectionPropertyNoOfPoles);

            double connectorVoltage = GetConnectorPropertyDouble(dev, indx, Context.
ConnectionPropertyVoltage);
```

```
            int connectorNumberOfPoles = GetConnectorPropertyNumberOfPoles(dev, indx);

            if (circuitVoltage == connectorVoltage || connectorVoltage ==(double)
    ConnectionProperty.Undefined || connectorVoltage==(double)ConnectionProperty.
    ByCircuit)
            {
                voltageOk = true;
            }

            if (circuitNumberOfPoles == connectorNumberOfPoles || connectorNumberOfPoles =
    =(int)ConnectionProperty.Undefined || connectorNumberOfPoles==(int)ConnectionProperty.
    ByCircuit)
            {
                polesOk = true;
            }

            if (!voltageOk || !polesOk)
            {
                m_voltsPolesMismatchList.Add(dev.ObjectId);
            }
        }

        private static double GetConnectorPropertyDouble(Device dev, int index, Context
    context)
        {
            double value = (double)ConnectionProperty.Null;

            DataRecord styleData = PartManager.GetPartData(dev);
            DataField dfStyle = null;

            DataRecord instanceData = PartManager.GetEngineeringData(dev);
            DataField dfInstance = null;

            try
            {
                dfInstance = instanceData.DataFields.FindByContextAndIndex(context, index)
    ;
                value = dfInstance.ValueDouble;
            }
            catch
            {
            }

            try
            {
                dfStyle = styleData.DataFields.FindByContextAndIndex(context, index);
                value = dfStyle.ValueDouble;
                if (dfStyle.AllowOverride == true)
                {
                    value = dfInstance.ValueDouble;
                }
            }
            catch
            {
            }

            Debug.Assert(value != (double)ConnectionProperty.Null);
            return value;
        }

        private static int GetConnectorPropertyNumberOfPoles(Device dev, int index)
        {
            Context context = Context.ConnectionPropertyNoOfPoles;
            int value = (int)ConnectionProperty.Null;

            DataRecord styleData = PartManager.GetPartData(dev);
            DataField dfStyle = null;
```

```
            DataRecord instanceData = PartManager.GetEngineeringData(dev);
            DataField dfInstance = null;

            try
            {
                dfInstance = instanceData.DataFields.FindByContextAndIndex(context, index)
    ;
                value = dfInstance.ValueInteger;
            }
            catch
            {
            }

            try
            {
                dfStyle = styleData.DataFields.FindByContextAndIndex(context, index);
                if (dfStyle.AllowOverride == false)
                {
                    value = dfStyle.ValueInteger;
                }
            }
            catch
            {
            }

            Debug.Assert(value != (int)ConnectionProperty.Null);
            return value;
        }


        private static Circuit CheckConnectorCircuit(Device dev, int index)
        {
            Circuit ckt = null;
            DataRecord instanceData = PartManager.GetEngineeringData(dev);

            try
            {
                DataField dfCircuit = instanceData.DataFields.FindByContextAndIndex
    (Context.ConnectionPropertyCircuit, index);
                if (dfCircuit.ValueObjectId != ObjectId.Null)
                {
                    ckt = m_trans.GetObject(dfCircuit.ValueObjectId, OpenMode.ForRead) as
    Circuit;
                    if (ckt.CircuitType == CircuitType.CopyProjectDatabaseMissing || ckt.
    CircuitType == CircuitType.CopyDeleted)
                    {
                        m_circuitNotFoundInProjectDatabaseList.Add(dev.ObjectId);
                    }
                }
                else
                {
                    m_uncircuitedList.Add(dev.ObjectId);
                }
            }
            catch
            {
                m_uncircuitedList.Add(dev.ObjectId);
            }
            return ckt;
        }
    }
}
```

```
using System;
using System.Collections.Generic;

// AutoCAD Specific
using Autodesk.AutoCAD.DatabaseServices;

// MEP Specifics
using Autodesk.Aec.Building.ApplicationServices;
using Autodesk.Aec.Building.Piping.DatabaseServices;

// create an alias to the AutoCAD Application object
using AcadApp = Autodesk.AutoCAD.ApplicationServices.Application;
using ObjectId = Autodesk.AutoCAD.DatabaseServices.ObjectId;

namespace ME104_2L
{
    class GetPipeSystemVolumeCommand
    {
        static Dictionary<double, double> m_pipeLengthInfo; // key = nominal inches; value
 = length in inches
        static Selection m_selection;

        public static void Execute()
        {
            m_selection = new Selection();
            int pipeCount = m_selection.m_pipeObjectIds.Count;
            int otherCount = m_selection.m_otherObjectIds.Count;
            if (pipeCount > 0)
            {
                GetPipeSegmentInfo();
                SummarizePipeInfo();
            }
        }


        private static void GetPipeSegmentInfo()
        {
            // student to fill in code starting here
            m_pipeLengthInfo = new Dictionary<double, double>();

            using (Transaction trans = Utilities.StartTransaction())
            {
                foreach (ObjectId oid in m_selection.m_pipeObjectIds)
                {
                    Pipe pipe = trans.GetObject(oid, OpenMode.ForRead) as Pipe;
                    Double length = pipe.Length;

                    DataRecord rec = PartManager.GetPartData(pipe);

                    foreach (DataField df in rec.DataFields)
                    {
                        if (df.Name == "ND1")
                        {
                            Double nominalDiameter = df.ValueDouble;
                            if (!m_pipeLengthInfo.ContainsKey(nominalDiameter))
                            {
                                m_pipeLengthInfo.Add(nominalDiameter, length);
                            }
                            else
                            {
                                m_pipeLengthInfo[nominalDiameter] += length;
                            }
                        }
                    }
                }
                trans.Commit();
            }
```

```
            // student code ends here
        }



        private static double CalculateSystemVolume()
        {
            double totalVolumeCuIn = 0;

            if (m_pipeLengthInfo == null)
            {
                return 0;
            }

            foreach (double diameter in m_pipeLengthInfo.Keys)
            {
                double length = m_pipeLengthInfo[diameter];
                double radius = diameter / 2.0;
                totalVolumeCuIn += Math.PI * Math.Pow(radius, 2) * length;
            }

            double totalVolumeGallons = totalVolumeCuIn * Utilities.CubicInchesToGallons;
            return totalVolumeGallons;
        }

        private static void SummarizePipeInfo()
        {
            foreach (double diameter in m_pipeLengthInfo.Keys)
            {
                double length = m_pipeLengthInfo[diameter];
                length = length * Utilities.InchesToFeet;
                Utilities.Editor.WriteMessage(String.Format("\n{0}\' of {1}\" pipe",      ↙
    length.ToString("0.0"), diameter));
            }

            double volume = CalculateSystemVolume();
            Utilities.Editor.WriteMessage(String.Format("\nTotal Volume in Gallons: {0}", ↙
    volume.ToString("0.00")));
        }
    }
}
```

```
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
    <StartAction>Program</StartAction>
    <StartProgram>C:\Program Files\AutoCAD MEP 2008\acad.exe</StartProgram>
    <StartWorkingDirectory>C:\Program Files\AutoCAD MEP 2008\UserDataCache\</
    StartWorkingDirectory>
    <StartArguments>/ld "C:\Program Files\AutoCAD MEP 2008\AecBase.dbx" /b "C:\ME104_2L\
    ME104_2L\ME104_2L\Test.scr"</StartArguments>
  </PropertyGroup>
</Project>
```

```
using System.Collections.Specialized;

// AutoCAD Specific
using Autodesk.AutoCAD.DatabaseServices;

// ACA Specific
using Autodesk.Aec.PropertyData.DatabaseServices;

using ObjectId = Autodesk.AutoCAD.DatabaseServices.ObjectId;
using DBObject = Autodesk.AutoCAD.DatabaseServices.DBObject;

namespace ME104_2L
{
    class PropertySetHelpers
    {

        public static ObjectId GetPropertySet(ObjectId oid, ObjectId psdId)
        {
            ObjectId propSetId = ObjectId.Null;
            Database db = HostApplicationServices.WorkingDatabase;
            using (Transaction trans = db.TransactionManager.StartTransaction())
            {
                DBObject obj = trans.GetObject(oid, OpenMode.ForWrite);

                try
                {
                    propSetId = PropertyDataServices.GetPropertySet(obj, psdId);
                }
                catch
                {
                    // object doesn't have PSD attached, so attach it
                    PropertyDataServices.AddPropertySet(obj, psdId);
                    propSetId = PropertyDataServices.GetPropertySet(obj, psdId);
                }
                trans.Commit();
            }
            return propSetId;
        }

        public static ObjectId CreateCheckElectricalConnectionsPropertySet()
        {
            ObjectId psdefId = ObjectId.Null;
            string psName = Constants.CheckElectricalConnections;

            Database db = HostApplicationServices.WorkingDatabase;
            DictionaryPropertySetDefinitions dictPSDs = new                    ↙
    DictionaryPropertySetDefinitions(db);
            PropertySetDefinition psdef;

            using (Transaction trans = db.TransactionManager.StartTransaction())
            {
                if (!(dictPSDs.Has(psName, trans)))
                {
                    StringCollection sColl = new StringCollection();
                    sColl.Add(Constants.AecbDbDevice);

                    psdef = new PropertySetDefinition();
                    psdef.SetAppliesToFilter(sColl, false);

                    dictPSDs.AddNewRecord(psName, psdef);
                    trans.AddNewlyCreatedDBObject(psdef, true);
                }

                psdefId = dictPSDs.GetAt(psName);
                psdef = trans.GetObject(psdefId, OpenMode.ForWrite) as         ↙
    PropertySetDefinition;
```

```
                PropertyDefinition pd = new PropertyDefinition();
                pd.DataType = Autodesk.Aec.PropertyData.DataType.TrueFalse;
                pd.IsReadOnly = true;
                pd.DefaultData = false;
                pd.Description = "Use the CheckElectricalConnections command to update
    this field.";
                pd.Name = Constants.ConnectionsOK;
                if (!(psdef.Definitions.Contains(pd)))
                {
                    psdef.Definitions.Add(pd);
                }

                trans.Commit();
            }
            return psdefId;
        }
    }
}
```

```
// AutoCAD Specific
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;

// MEP Specifics
using Autodesk.Aec.Building.DatabaseServices;
using Autodesk.Aec.Building.Piping.DatabaseServices;

// create an alias to the AutoCAD Application object
using AcadApp = Autodesk.AutoCAD.ApplicationServices.Application;

namespace ME104_2L
{
    class Selection
    {
        public ObjectIdCollection m_pipeObjectIds;
        public ObjectIdCollection m_otherObjectIds;
        Transaction m_trans;

        // this is used for the Pipe Sample
        public Selection()
        {
            m_pipeObjectIds = new ObjectIdCollection();
            m_otherObjectIds = new ObjectIdCollection();
            GetPipeNetworkObjectIds();
        }

        // this is used for the Pipe Sample
        public void GetPipeNetworkObjectIds()
        {
            // store pipes separate from other elements (valves, fittings, etc...)

            // set up the entity prompt options, note that we only want a pipe object
            PromptEntityOptions prOptions = new PromptEntityOptions("Select a pipe");
            prOptions.SetRejectMessage("\nObject must be a pipe.");
            prOptions.AddAllowedClass(typeof(Pipe), false);

            // prompt the user to select a member
            Editor ed = Utilities.Editor;
            PromptEntityResult prResult = ed.GetEntity(prOptions);

            if (prResult.Status != PromptStatus.OK)
            {
                return;
            }

            // get the ObjectId from the selected object
            Autodesk.AutoCAD.DatabaseServices.ObjectId oidSelected = prResult.ObjectId;
            m_pipeObjectIds.Add(oidSelected);

            try
            {
                using (m_trans = AcadApp.DocumentManager.MdiActiveDocument.Database.
    TransactionManager.StartTransaction())
                {
                    Member mem = m_trans.GetObject(oidSelected, OpenMode.ForRead) as
    Member;
                    GetConnectedObjects(mem);
                    m_trans.Commit();
                }
            }
            catch
            {
            }
        }

        // this is used for the Pipe Sample
```

```
        private void GetConnectedObjects(Member member)
        {
            ConnectionComponentMemberCollection connComps = member.
    ConnectionComponentMembers;
            foreach (ConnectionComponentMember ccm in connComps)
            {
                Autodesk.AutoCAD.DatabaseServices.ObjectIdCollection oids = member.
    GetObjectsAtConnectionComponent(ccm);

                foreach (Autodesk.AutoCAD.DatabaseServices.ObjectId id in oids)
                {
                    if (!m_pipeObjectIds.Contains(id) && !m_otherObjectIds.Contains(id))
                    {
                        Member iterMember = m_trans.GetObject(id, OpenMode.ForRead, false)
     as Member;
                        if (iterMember.GetType() == typeof(Pipe))
                        {
                            m_pipeObjectIds.Add(id);
                        }
                        else
                        {
                            m_otherObjectIds.Add(id);
                        }
                        GetConnectedObjects(iterMember);
                    }
                }
            }
        }

        // this is for the Electrical Device sample
        public static ObjectIdCollection GetAllOfType(string dxfEntityType)
        {
            TypedValue[] filterlist = new TypedValue[1];
            filterlist[0] = new TypedValue(0, dxfEntityType);

            SelectionFilter filter = new SelectionFilter(filterlist);

            PromptSelectionResult prResult;
            Editor ed = AcadApp.DocumentManager.MdiActiveDocument.Editor;
            prResult = ed.SelectAll(filter);

            if (prResult.Status != PromptStatus.OK)
            {
                return null;
            }

            SelectionSet ss = prResult.Value;
            ObjectIdCollection oidc = new ObjectIdCollection(ss.GetObjectIds());
            return oidc;
        }
    }
}
```

```
(command "netload" "C:\\ME104_2L\\ME104_2L\\ME104_2L\\bin\\Debug\\ME104_2L.dll")
open
"C:\ME104_2L\ME104_2L\ME104_2L\Test.dwg"
```

```
using System;

using AcadApp = Autodesk.AutoCAD.ApplicationServices.Application;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.DatabaseServices;

namespace ME104_2L
{
    class Utilities
    {
        public static Editor Editor
        {
            get { return AcadApp.DocumentManager.MdiActiveDocument.Editor; }
        }

        public static Transaction StartTransaction()
        {
            return AcadApp.DocumentManager.MdiActiveDocument.Database.TransactionManager. ↙
    StartTransaction();
        }

        public static Double CubicInchesToGallons
        {
            get { return 0.00432900431; }
        }

        public static Double InchesToFeet
        {
            get { return 1.0 / 12.0; }
        }

    }
}
```