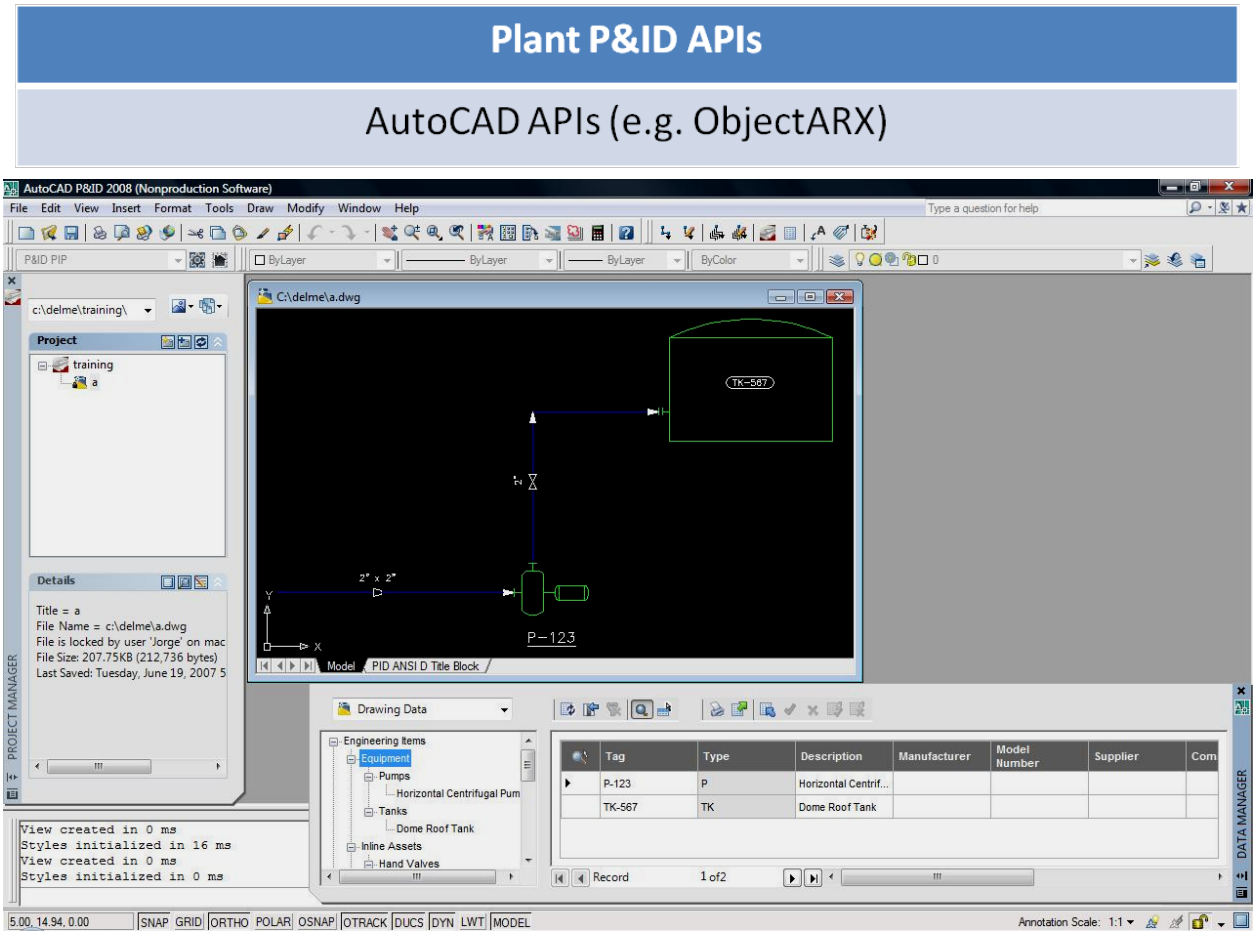
Customize AutoCAD® P&ID to Your Needs

Use the power of AutoCAD P&ID and extend its capabilities to suit your needs by writing custom commands, features, and solutions.

Getting Started

The P&ID Software Developer's Kit, PnPSDK, extends the AutoCAD ObjectARX SDK. P&ID-specific features are available as extensions to ObjectARX and the ObjectARX managed extensions for .NET. You can view the P&ID development environment as having everything AutoCAD has to offer plus an additional library of P&ID specific APIs.



The PnPSDK requires AutoCAD 2008 ObjectARX be installed. You can download ObjectARX from http://www.autodesk.com/objectarx.

PnPSDK files are installed with AutoCAD P&ID into the [install folder]\PnPSDK. You may copy the contents of the PnPSDK folder to the ObjectARX folder, which is C:\ObjectARX 2008\ by default. The PnPSDK files share the existing ObjectARX folder structure, but no files are overwritten.

The .\inc subdirectory contains PID class declarations. The .\lib subdirectory contains the PID import library file names PnPProjectManager.lib, AcPnpDataLinks.lib, and PnIDObjects.lib. DLL

Autodesk®

files PnIDMgd.dll, PnPDataLinks.dll, PnPProjectManagerMgd.dll, and PnPDataObjects.dll in .\inc are .NET managed assemblies.

Using the Project

The Project class contains services to access project level information. This includes the project settings, the DWG files in the project, and links to other data files and objects such as the symbol & style library, the data links manager to access the project database information, etc.

To use the class, add a reference to the PnPProjectManagerMgd.dll .NET assembly.

NOTE: At this time there are no APIs to control Project Manager UI level operations.

Below is code that asks for the current project object and then lists all the drawings of the project in the AutoCAD console window.

C# Code

```
using AcadApp = Autodesk.AutoCAD.ApplicationServices.Application;
using Autodesk.ProcessPower.ProjectManager;

[CommandMethod("LISTDWGS", CommandFlags.Modal)]
public static void ListProjectDrawings()
{
    Project oPrj = PlantApplication.CurrentProject.ProjectParts[“PnId”];
    List<PnPProjectDrawing> oDwgList = oPrj.GetPnPDrawingFiles();
    Editor oEditor = AcadApp.DocumentManager.MdiActiveDocument.Editor;

    foreach (PnPProjectDrawing oDwg in oDwgList)
    {
        string strMsg = "\n" + oDwg.AbsoluteFileName;
        oEditor.WriteMessage(strMsg);
    }
}
```

P&ID Objects in DWG Files

The P&ID custom objects and entities are exposed via the PnIdMgd.dll .NET assembly. This assembly is used any time data P&ID objects are to be read or written to .DWG files. The classes in PnIdMgd.dll are modeled after AutoCAD’s Managed APIs and are used in the same manner. Please refer to the ObjectARX Developer’s Guide under Using .NET for AutoCAD Development and the ObjectARX Managed Class Reference for more information.

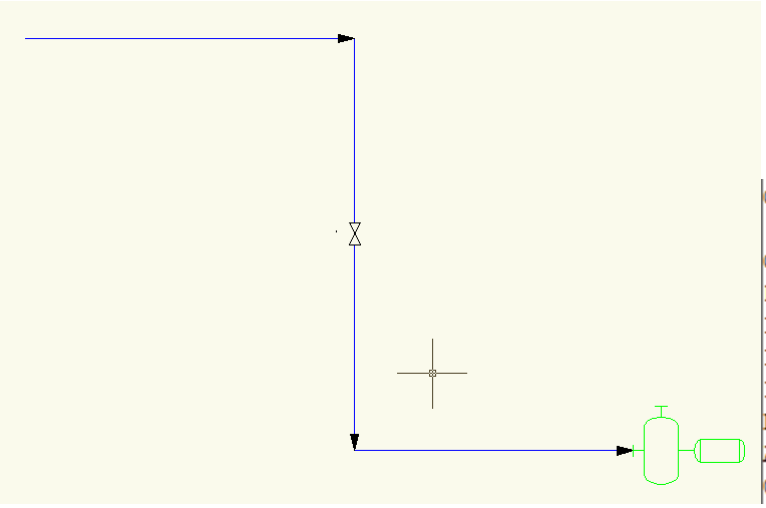
Below is a function that checks every entity in the model space of a drawing and if it is a P&ID entity then it displays the entity’s class name in the AutoCAD console window.

C# Code

```
using Autodesk.ProcessPower.PnIDObjects;
using AcadApp = Autodesk.AutoCAD.ApplicationServices.Application;

[CommandMethod("LISTPNP", CommandFlags.Modal)]
public static void ListPnPEntities()
```

```
{
    Editor oEditor = AcadApp.DocumentManager.MdiActiveDocument.Editor;
    Database oDB = AcadApp.DocumentManager.MdiActiveDocument.Database;

    TransactionManager oTxMgr = oDB.TransactionManager;
    using(Transaction oTx = oTxMgr.StartTransaction())
    {
        BlockTable oBT = (BlockTable)oTx.GetObject(oDB.BlockTableId,
            OpenMode.ForRead);
        BlockTableRecord oMS =(BlockTableRecord)oTx.GetObject(
            oBT["*Model_Space"], OpenMode.ForRead);

        BlockTableRecordEnumerator iter = oMS.GetEnumerator();
        while (iter.MoveNext())
        {
            DBObject oDbObj = oTx.GetObject(iter.Current, OpenMode.ForRead);

            /* PnId entities are either LineSegment or Asset (possibly
               derived). Annotations use normal AutoCAD text or block
               reference entities and can be identified via class
               AnnotationUtils method IsAnnotation().
             */
            LineSegment oLS = oDbObj as LineSegment;
            if (oLS != null)
            {
                string strMsg = "\n" + oLS.ClassName;
                oEditor.WriteMessage(strMsg);
            }

            Asset oAsset = oDbObj as Asset;
            if (oAsset != null)
            {
                string strMsg = "\n" + oAsset.ClassName;
                oEditor.WriteMessage(strMsg);
            }
        }

        oTx.Commit();
    }
}
```

Sample drawing and output:



```
Command: listpnp

Command:
Primary
FlowArrow
FlowArrow
FlowArrow
HorizontalCentrifugalPump
AssumedNozzle
GateValve
```

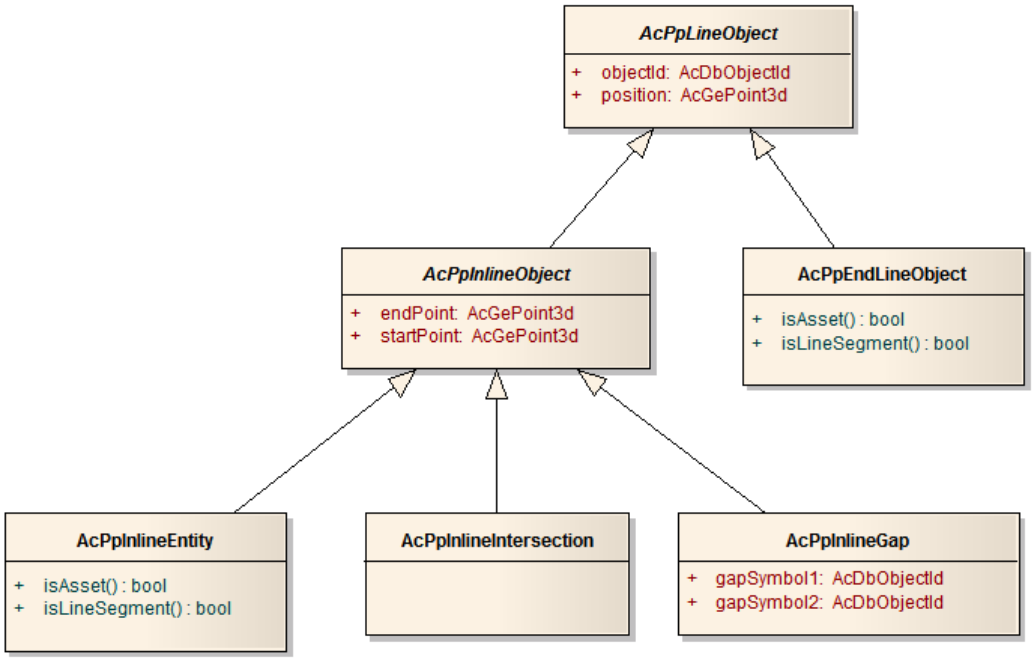
For a comprehensive list of all P&ID objects and entities, refer to the AutoCAD P&ID Developer's Guide that comes with PnPSDK.

Walking the Objects on a Line

Entities in DWG files are always owned by block table records. This is true for entities that are seemingly owned by other P&ID entities in AutoCAD Plant P&ID. Examples of these entities include inline assets connected to a line or owned assets such as a nozzle that is connected to a tank.

The ownership is simulated within the implementation of our custom entities using relationships stored in the project database. The project database and relationships mechanism will be described later in this document.

The LineSegment entity tracks a list of metadata packages about other entities that are connected to it. These packages translate into instances of a "line object" class. Below is a diagram of the available classes with a few key properties identified.



- Inline objects represent valves and other inline components. The line entity does not draw itself through the space used by an inline entity. Inline entities are also moved / copied along with the line they are on.
- End-line objects represent any connection at the start or end of a line. The connection can be to another line or equipment entity.
- Intersection objects are visualized as either gaps or loops based on a project setting and occurs where two lines cross each other.
- Inline gap objects represent a user created gap via SLEDIT.

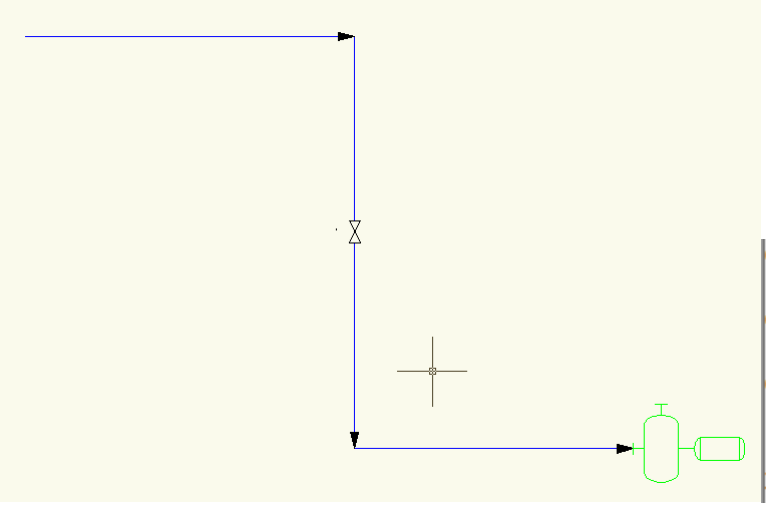
Below is code that asks the user to select a line and then displays the objects connected to it.

C# Code

```
using Autodesk.ProcessPower.PnIDObjects;
using AcadApp = Autodesk.AutoCAD.ApplicationServices.Application;

[CommandMethod("LISTLINEOBJS", CommandFlags.Modal)]
public static void ListLineObjects()
{
    Editor oEditor = AcadApp.DocumentManager.MdiActiveDocument.Editor;

    PromptEntityOptions oPromptOptions = new
       PromptEntityOptions("Select an object");

    PromptEntityResult oPromptResult = oEditor.GetEntity(oPromptOptions);
    if (oPromptResult.Status != PromptStatus.OK)
    {
        return;
    }

    Database oDB = AcadApp.DocumentManager.MdiActiveDocument.Database;
    TransactionManager oTxMgr = oDB.TransactionManager;
    using (Transaction oTx = oTxMgr.StartTransaction())
    {
        DBObject obj = oTx.GetObject(oPromptResult.ObjectId,
            OpenMode.ForRead);

        LineSegment oLS = obj as LineSegment;
        if (oLS != null)
        {
            /* No filter is specified so all line objects will be returned.
               It is possible to specify a filter object that is called with
               each object before it is to be added. This allows for objects
               to be filtered out easily. For example, if you do not want to
               get Flow Arrow objects in the collection, then implement a
               filter that recognizes them and disallow them from being added
               to the collection.
            */
            LineObjectCollection los = oLS.GetLineObjectCollection(null);
            /* Line objects are metadata classes that represent "something"
               on the line. They are in order from the start to end of the
               line.
             */
            foreach (LineObject lo in los)
            {
                if (lo is InlineGap)    // user gaps have nothing to display
                    continue;

                DBObject oDbObj = oTx.GetObject(lo.ObjectId,
                   OpenMode.ForRead);

                LineSegment ls = oDbObj as LineSegment;
                if (ls != null)
                {
                    string strMsg = "\n" + ls.ClassName;
                    oEditor.WriteMessage(strMsg);
                }

                Asset oAsset = oDbObj as Asset;
```

```
                if (oAsset != null)
                {
                    string strMsg = "\n" + oAsset.ClassName;
                    oEditor.WriteMessage(strMsg);
                }
            }
        }

        oTx.Commit();
    }
}
```

Sample drawing and output:

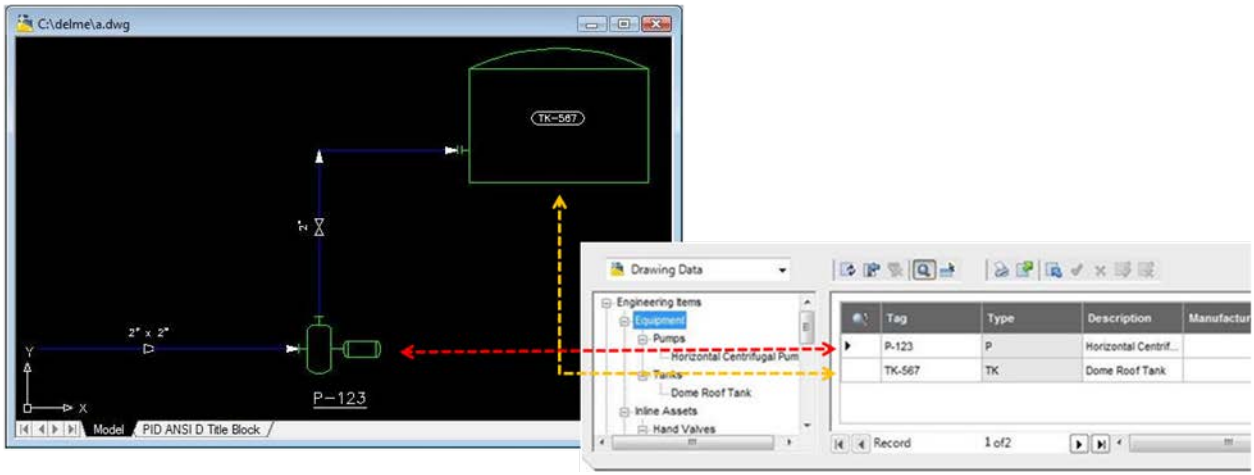


```
Command: listlineobjs

Command: Select an object:
FlowArrow
GateValve
FlowArrow
FlowArrow
HorizontalCentrifugalPump
```

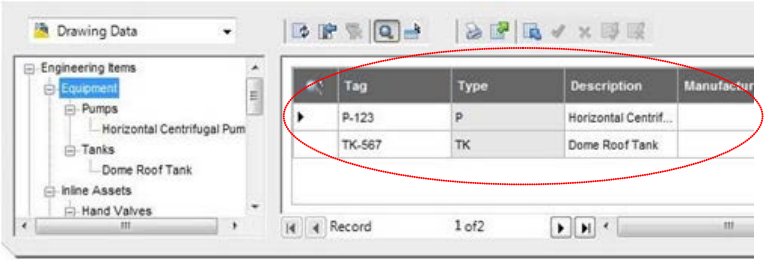
Using the Data Links Manager

The data links manager is a class that is used to work with the project database. The functionality breaks down into three main areas:

- Link between objects in the drawing files and records in the project database



- Non-graphical properties of entities in the drawing (e.g. tag, description, etc)



- Relationship between objects (e.g. pump is end asset of a line)

Relationships are not visible in the drawing but they are critical for proper graphical editing of the P&ID drawings. For example, if a line is connected (e.g. related) to a pump and the pump or line is moved, then the line elbows or stretches to maintain the connection between the two objects. These relationships are stored in the project database and the P&ID custom entities query for this information necessary.

The previous code example asked a line for its “line objects” and they were returned in the order they are encountered along with intersections and user gaps. If ordering is not important and we are interested in other connected lines or assets only, then the project database can be queried for this information, even if the drawing is not loaded in AutoCAD.

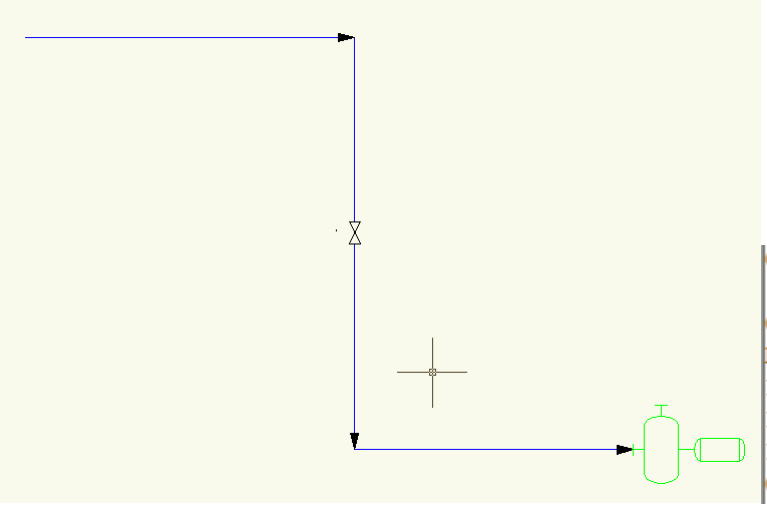
Below is the previous code example changed to query the project database directly and display the list of related objects. This will touch on the main three areas of the DataLinksManager.

C# Code

```
using AcadApp = Autodesk.AutoCAD.ApplicationServices.Application;
using Autodesk.ProcessPower.ProjectManager;
using Autodesk.ProcessPower.DataLinks;
using Autodesk.ProcessPower.DataObjects;

[CommandMethod("LISTRELOBJS", CommandFlags.Modal)]
public static void ListRelatedObjects()
{
    Editor oEditor = AcadApp.DocumentManager.MdiActiveDocument.Editor;

    PromptEntityOptions oPromptOptions =
       new PromptEntityOptions("Select an object");

    PromptEntityResult oPromptResult = oEditor.GetEntity(oPromptOptions);
    if (oPromptResult.Status != PromptStatus.OK)
    {
        return;
    }

    Project oPrj = PlantApplication.CurrentProject.ProjectParts[“PnId”];
    DataLinksManager oDLMgr = oPrj.DataLinksManager;

    /* Get the project database id for the selected entity
    */
    int nRowId = oDLMgr.FindAcPpRowId(oPromptResult.ObjectId);

    /* Ask for specific related objects
    */
    PnPRowIdArray relIds = new PnPRowIdArray();
```

```
    PnPRowIdArray startIds =
       oDLMgr.GetRelatedRowIds("LineStartAsset", "Line", nRowId, "Asset");
    PnPRowIdArray endIds =
       oDLMgr.GetRelatedRowIds("LineEndAsset", "Line", nRowId, "Asset");
    PnPRowIdArray flowIds =
       oDLMgr.GetRelatedRowIds("LineFlowArrow", "Line", nRowId, "Arrow");
    PnPRowIdArray inlineIds =
       oDLMgr.GetRelatedRowIds("LineInlineAsset", "Line", nRowId, "Asset");

    /* Combine all the lists into one
    */
    foreach (int id in startIds)
    {
        relIds.AddLast(id);
    }
    foreach (int id in endIds)
    {
        relIds.AddLast(id);
    }
    foreach (int id in flowIds)
    {
        relIds.AddLast(id);
    }
    foreach (int id in inlineIds)
    {
        relIds.AddLast(id);
    }

    StringCollection propNames = new StringCollection();
    propNames.Add("Description");

    StringCollection propVals = null;
    foreach (int nRelId in relIds)
    {
        /* Ask for the description property and display it
           to the console window.
        */
        propVals = oDLMgr.GetProperties(nRelId, propNames, true);

        string strMsg = "\n" + propVals[0];
        oEditor.WriteMessage(strMsg);
    }
}
```

Sample drawing and output:

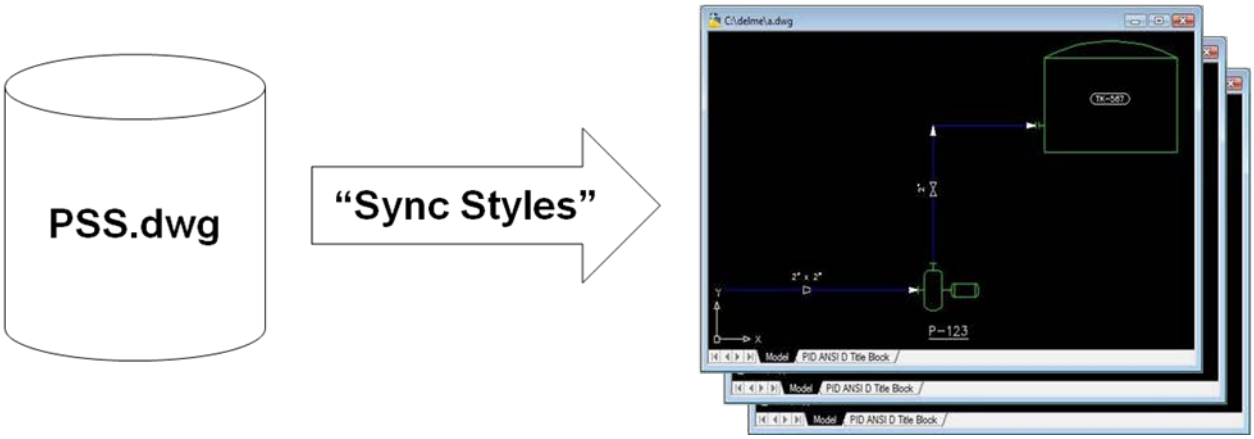


```
Command: listrelobjs

Command: Select an object:
Horizontal Centrifugal Pump
Flow Arrow
Flow Arrow
Flow Arrow
Gate Valve
```

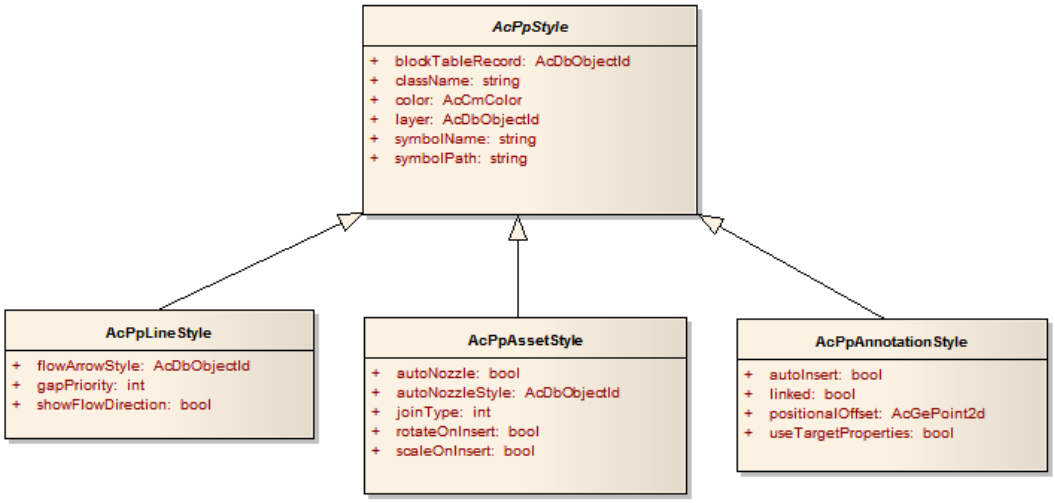
The description property and class name match up so the output is almost identical. The code can be easily changed to get and display different properties. For a comprehensive list of all the relationships, refer to the AutoCAD P&ID Developer's Guide that comes with PnPSDK.

Styles

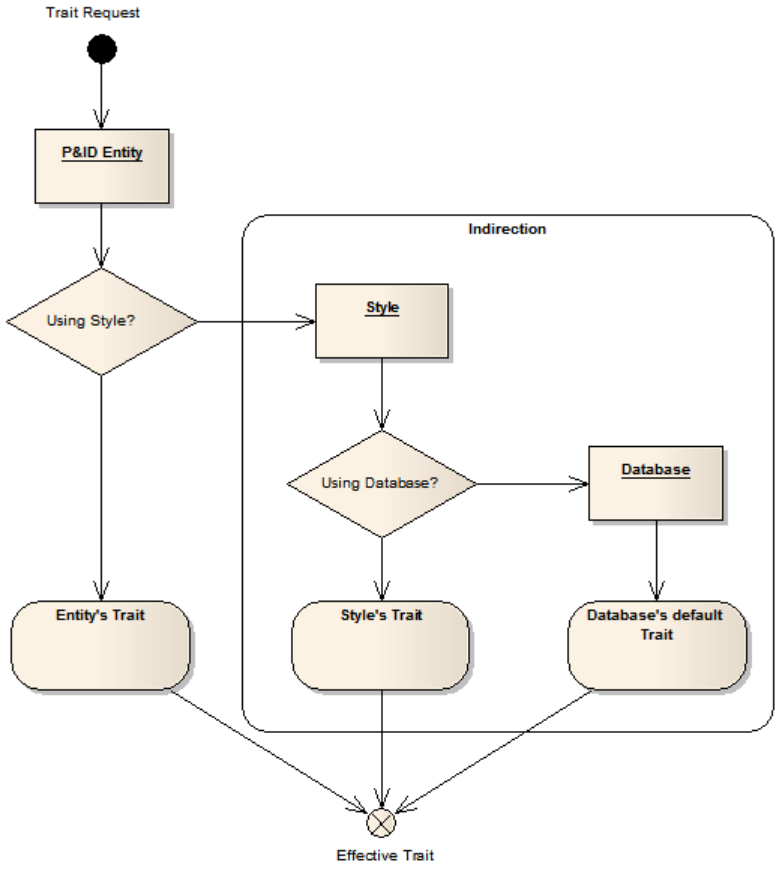
All P&ID custom entities point to a style object that defines the appearance and other graphical properties of the entity. By using styles, the graphical standards of the project can be managed in a single location, the projSymbolStyle.dwg of the project, and propagated accordingly.



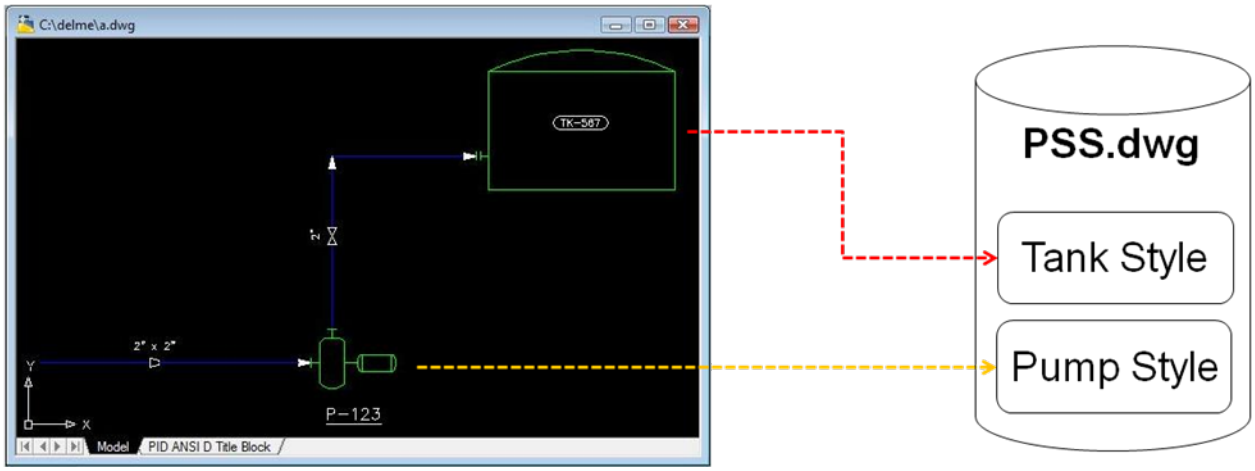
There are a number of style classes that are used by different types of P&ID entities. The class hierarchy for styles is shown below.



P&ID entities can override style settings and therefore the following decision tree is used for each graphical trait when the entity is being rendered.



The asset style contains a pointer to the block definition that is used to display the entity. Below is an illustration where one instance of class Asset looks like a pump while another like a tank.



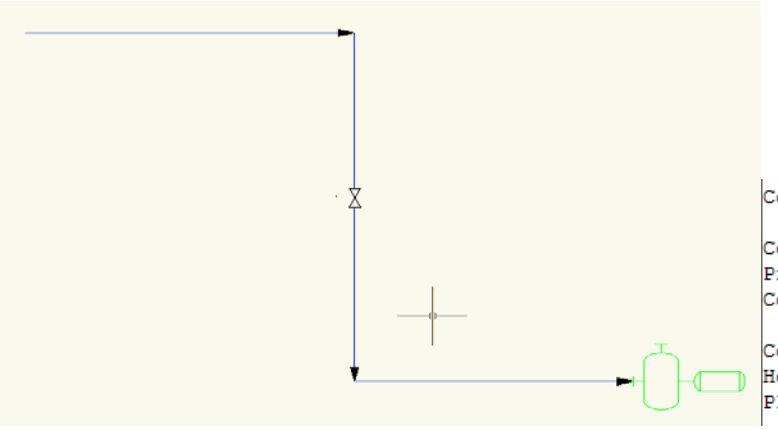
Now let's look at some code that allows a user to select a P&ID entity in the drawing and then displays the name of the style it is pointing to and the name of the block definition (if applicable).

C# Code

```
using Autodesk.ProcessPower.PnIDObjects;
using Autodesk.ProcessPower.Styles;
using AcadApp = Autodesk.AutoCAD.ApplicationServices.Application;

[CommandMethod("LISTSTYLE", CommandFlags.Modal)]
public static void ListStyle()
{
    Editor oEditor = AcadApp.DocumentManager.MdiActiveDocument.Editor;

    PromptEntityOptions oPromptOptions =
      new PromptEntityOptions("Select an object");
    PromptEntityResult oPromptResult = oEditor.GetEntity(oPromptOptions);
    if (oPromptResult.Status != PromptStatus.OK)
    {
        return;
    }

    Database oDB = AcadApp.DocumentManager.MdiActiveDocument.Database;
    TransactionManager oTxMgr = oDB.TransactionManager;
    using (Transaction oTx = oTxMgr.StartTransaction())
    {
        DBObject obj = oTx.GetObject(oPromptResult.ObjectId,
            OpenMode.ForRead);
        LineSegment oLS = obj as LineSegment;
        Asset oAsset = obj as Asset;

        /* Get the style's object id
        */
        ObjectId styleId = new ObjectId();
        if (oLS != null)
        {
            styleId = oLS.StyleID;
        }

        if (oAsset != null)
        {
            styleId = oAsset.StyleId;
        }
```

```
        if (!styleId.IsNull)
        {
            /* Display the style's name
            */
            Style oStyle = (Style)oTx.GetObject(styleId, OpenMode.ForRead);

            string strMsg = "\n" + oStyle.Name;
            oEditor.WriteMessage(strMsg);

            /* If it is an asset style then display the block name
            */
            AssetStyle oAssetStyle = oStyle as AssetStyle;
            if (oAssetStyle != null)
            {
                ObjectId btrId = oAssetStyle.BlockTableRecord;
                BlockTableRecord oBTR = (BlockTableRecord)
                   oTx.GetObject(btrId, OpenMode.ForRead);

                strMsg = "\n" + oBTR.Name;
                oEditor.WriteMessage(strMsg);
            }
        }

        oTx.Commit();
    }
}
```

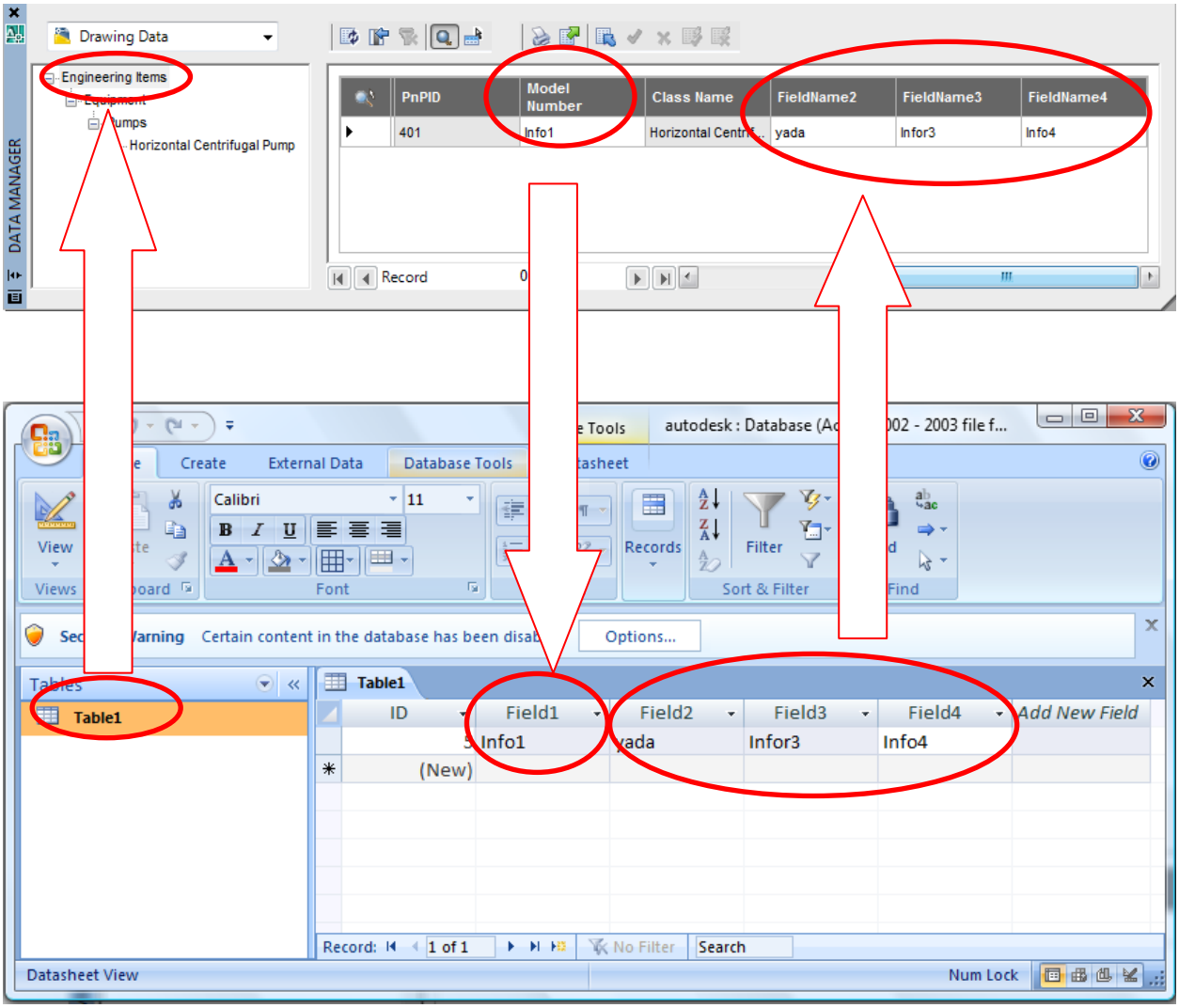
Sample drawing and output:



```
Command: liststyle

Command: Select an object:
Primary Style
Command: liststyle

Command: Select an object:
Horizontal Centrifugal Pump Style
PIP HORIZONTAL CENTRIFUGAL PUMP
```

Extension Properties Sample

Below is a sample database extension where columns from an external database (`Autodesk.mdb`) are added dynamically to the project database.



External database: `Autodesk.mdb`
External table: Table1

Key column: Field1
Extension columns: Field2, Field3, Field4

C# Code

```
/* This is a sample of adding additional columns to a table that are
 * defined in an external database.
 */
[CommandMethod("xdb_sample", CommandFlags.Modal)]
public static void xdb()
{
    try
    {
        /* The first thing we have to do is ask for the current data
         * links manager. From this object, we can get to the
         * lower-level database layer API known as PnPDataObjects.
         */
        StringCollection names = DataLinksManager.GetLinkManagerNames();
        DataLinksManager dlm = DataLinksManager.GetManager(names[0]);

        /* The XDb reference manager tracks all the external data
         * sources and references. References can be set up to use the
         * "source" model that means to copy data from fields in the
         * external database into the data-cache or the "extension"
         * model where additional columns are added to a table in
         * the data-cache.
         */
        PnPDatabase db = dlm.GetPnPDatabase();
        PnPXDbReferenceManager mgr = db.XDbReferenceManager;

        /* Data source defines how to connect to a database. */
        PnPXDbDatasource ds = new PnPXDbDatasource("autodesk_datasource");

        /* Connect to an Access database */
        string strCS = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=";
        strCS += "c:\\Training\\autodesk.mdb;Jet OLEDB:Engine Type=5;";
        strCS += "mode=Share Deny None";
        ds.ConnectionString = strCS;
        ds.Connect(string.Empty, string.Empty);

        /* Add the newly created data source to the XDb manager */
        mgr.Datasources.Add(ds);

        /* Create the reference to external data. This sample is
         * using the "extension model" where fields from the
         * external data source are displayed in the data manager.
         * The foreign key maps to the primary key in the external
         * database and this is how the two tables are kept
         * synchronized.
         */
        PnPXDbReference dbref_ext = new PnPXDbReference("testref");
        dbref_ext.IsExtension = true;

        /* Project database table and field */
        dbref_ext.Referencing.TableName = "EngineeringItems";
        dbref_ext.Referencing.ForeignKey.Add("ModelNumber");

        /* External database table and field */
        dbref_ext.Referenced.DatasourceName = "autodesk_datasource";
        dbref_ext.Referenced.TableName = "Table1";
        dbref_ext.Referenced.PrimaryKey.Add("Field1");

        /* The fields in the external database to extend the
```

```
             * project database with. The first parameter is the
             * local name in the project manager (displayed as column
             * header).
             */
            PnPXDbColumnMapCollection cm = dbref_ext.ColumnMap;
            cm.Add(new PnPXDbColumnMapEntry("FieldName2", "Field2"));
            cm.Add(new PnPXDbColumnMapEntry("FieldName3", "Field3"));
            cm.Add(new PnPXDbColumnMapEntry("FieldName4", "Field4"));

            /* Add the reference to the XDb manager */
            mgr.References.Add(dbref_ext);

            /* Once the XDb manager is tracking the data source and
             * the reference then we add the "extension" to a specific
             * table. In this example, the EngineeringItems table gets
             * the extension. All tables derived from EngineeringItems
             * will also display the fields.
             */
            PnPTable tbl = db.Tables["EngineeringItems"];
            tbl.AddRuntimeExtension("testref");
        }
        catch (System.Exception e)
        {
            System.Windows.Forms.MessageBox.Show(e.Message.ToString());
        }
        finally
        {
        }
    }
```