INSTRUCTOR: BIN LIN

SMARTPLANT P&ID V4.3

AUTOMATION PROGRAMMING

USING

VISUAL BASIC

COURSE GUIDE

CHAPTER 1: REVIEW OF VISUAL BASIC CONCEPTS

1. OBJECTIVES

In this chapter you will review:

◊ How to write simple standard executables

◊ The Visual Basic environment

◊ Some object-oriented concepts

2. ADVANTAGES OF USING VISUAL BASIC

◊ Visual Basic is an extension of the BASIC programming language utilizing the object-oriented technology.

◊ It provides an easy-to-use development environment that simplifies the creation of applications requiring graphical user interfaces.

◊ It supports OLE Automation programming because it is COM compliant.

  ◊ COM (Component Object Model) specifies binary level interfaces how components are created and how client applications connect to components.

  ◊ OLE (Object Linking and Embedding) Automation further specifies interfaces that manipulate an application's objects from outside of the application.

3. CREATING A STANDARD EXECUTABLE BASIC PROGRAM USING VB

3.1. *Module-Level Code*

◊ Standard Basic code is written in Modules. A module is a code segment that defines functions and subroutines.

3.2. *Programming features in VB*

3.2.1. Start-up Object

◊ Start-up function is the Sub Main() in the Module

3.2.2.Loop structures

- ◊ Do ...Loop While
- ◊ Do...Loop Until
- ◊ Do While ...Loop
- ◊ Do Until ...Loop
- ◊ For ... Next
- ◊ Exit statement can be used to exit a Do or For loop

3.2.3. Conditional IF structure

- ◊ IF ... THEN
- ◊ IF ... THEN ... ELSE ... ENDIF
- ◊ IF ... THEN ... ELSEIF ... THEN ... ELSE ... ENDIF

3.2.4.Select Case Statements

- ◊ SELECT CASE .. CASE .. CASE ... END SELECT

3.2.5.String concatenation

- ◊ Use the '&' symbol to concatenate strings into a variable.

3.2.6.Help feature

- ◊ Place the cursor on a keyword and press the F1 key to obtain help on the keyword

3.3. *Debugging a standard executable*

- ◊ Use the Breakpoints to stop a program.
- ◊ Use Watch expressions
- ◊ Step into, Step over, and Step out of.
- ◊ View the Call Stack.
- ◊ Run commands in the Immediate window.
- ◊ Examine objects in the Locals window.
- ◊ The Debug object with the Print method.
- ◊ The Err object and its properties: Number, Description, Source, Clear, Raise.
- ◊ The On Error statement.

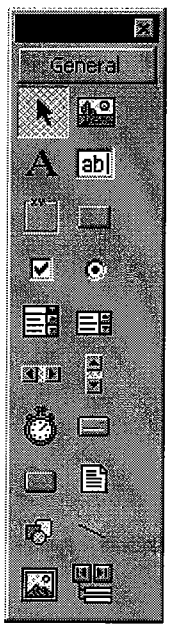
3.4. *Compile and Run a Standard executable*

◊ Select File->Make->...

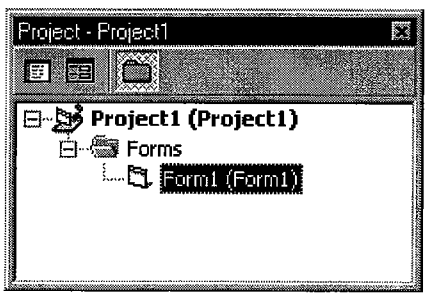
◊ Enter the name of the executable file

4. VISUAL BASIC ENVIRONMENT

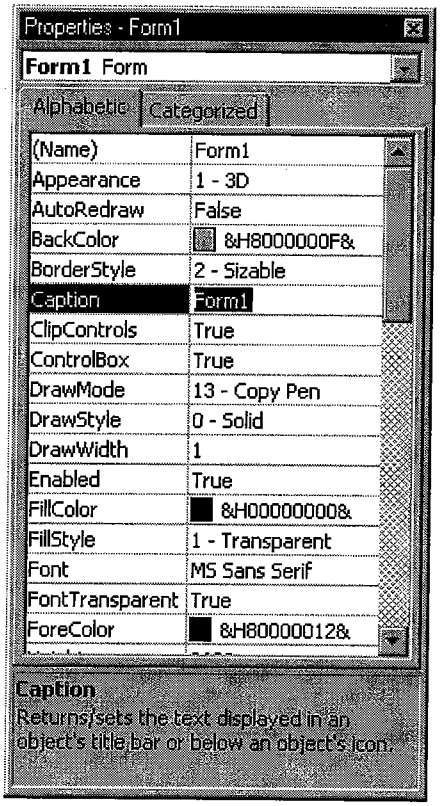
4.1. *Toolbox*



4.2. *Project Explorer Window*



4.3. *Properties Window*



4.4. *Object Browser for Browsing the Libraries*

The Object Browser can be opened from the Standard Toolbar or from the View command on the menu. The Object Browser is used to examine the methods and properties of the classes in object libraries.

4.5. *Immediate, Locals, and Watch Window for Debugging*

The Immediate, Locals, and Watch Windows can be opened from the Debug Toolbar or from the View command on the menu. These windows are used to examine the values of variables and objects at run time.

4.6. *References*

The Project/References... command pulls up a dialog to select libraries that can be referenced into the current project. By referencing these libraries, the type information of the classes in these libraries are available to the programmer during code development and compilation.

4.7. *Components*

The Project/Components... command pulls up a dialog to select the ActiveX components that can be included into the current project. ActiveX components can be used to provide user-interface and code level functionalities to the project.

4.8. *Project Properties*

- ◊ Convert between standard EXE and Active-X dlls.
- ◊ Select start-up component.
- ◊ Set compatibility.

5. USING VISUAL BASIC

5.1. *Controls*

- ◊ Controls are predefined classes with graphical representations.
- ◊ Controls can be added to the project by dragging onto a form from the Toolbox.
- ◊ Every instance of a control has a unique name within the form

5.2. *Forms*

- ◊ Forms are like Controls, with properties, methods, and events.
- ◊ Forms are containers for Active-X controls.

5.3. *Procedures*

- ◊ Two types: Event Procedures and General Procedures
- ◊ General Procedures include Function procedures and Subroutine procedures

5.4. *Variables and Constants: Declaration, Scope, and Life*

- ◊ Always use the keywords Option Explicit at the top of every module. This will ensure that variables are always defined before they are used.
- ◊ Use Dim, Private, or Public to declare variables.
- ◊ Constants are declared with the CONST keyword
- ◊ Procedure-Level Variables are declared in a procedure and is recognized only within the procedure. Procedure-level are created at the start of the procedure and destroyed at the close of the procedure.

- ◊ Form-Level Variables are declared outside of procedures but inside a General Declarations section of the form and is global within the form. Public variables alone will be accessible to procedures outside of the form as a property of the form. They have to be referenced using the name of the form, such as frmMyForm.Property1. They are available from the instant the form is created until it is destroyed (`set frmMyForm = Nothing`).
- ◊ Standard Module-level Variables are declared within the General Declarations section of the Module. Private variables are available only within the Module, while Public variables are available as global variables within the whole project.

5.5. *DataTypes*

- ◊ Data Types include Integers, Longs, Double, String, Variant, etc.
- ◊ Variants hold all datatypes but not objects. A variant can have a NULL value, indicating that it is undefined.

6. FILES IN THE VISUAL BASIC PROJECT

6.1. *Project*

- ◊ A Project consists of all of the form modules, standard modules, class modules, controls, references, and settings required to compile and run the application.
- ◊ It is saved as a *.vbp file.
- ◊ A project is synonymous with Library or Program, depending on the context.

6.2. *Standard Module*

- ◊ It is a standard code module.
- ◊ It is saved as a *.bas file.

6.3. *Class Module*

- ◊ It is a template for an object.
- ◊ An interface is an abstract class.
- ◊ It is saved as a *.cls file.

6.4. *Form Module*

- ◊ A Form is a Class definition.

- ◊ It is a container for controls.
- ◊ It has properties and methods like other classes.
- ◊ It has pre-defined Events just as for Controls.
- ◊ It is saved as a *.frm file.

7. OBJECT-ORIENTED CONCEPTS

7.1. *Object*

An object is a collection of data with related functions and subroutines that can be grouped as a single unit.

7.2. *Class*

- ◊ A class is a template for an object. It contains the structure and definitions of all of the parts of an object.
- ◊ A class is initialized when an object instance of the class is created. Eg:
  `set obj1 = New Class1`
- ◊ A class is terminated when the object instance of the class is set to nothing. Eg:
  `set obj1 = Nothing`
- ◊ The name of the class can be edited in the Property Box for the class module. This name can be different from the filename under which the class module is saved.

7.3. *Methods and Properties of Classes*

- ◊ Methods and Properties of a class are those variables and procedures that cannot be accessed other than within the context of an object instance of the class.
- ◊ The methods and properties of a class are accessed using the dot (.) extension to the class name. For instance, if the class `Object1` has a property called `Name` and `obj1` is an object instance of that class, then the property can be accessed using the syntax `obj1.Name`.
- ◊ Properties and Methods can be either Public or Private, depending on whether they must be visible to objects outside this class.
- ◊ Properties can also be defined as procedures to provide more access and validation control. Such procedures are prefixed by the keyword 'Property'.
- ◊ Properties have three procedure definitions: Get, Let, and Set.

- The Get procedure returns the value of the property and is called when the code is reading the value of the property such as in `Debug.Print obj1.Name`
- The Let procedure is called by VB when the property value is being assigned a value in the code. Eg. `Obj1.Name = "Object 1"`
- A Get procedure without the implementation of the Let procedure creates a Read-only property.
- A Set procedure is just like the Let procedure, except that it is used associate an object variable to an object. In this case the Set keyword must be used. Eg. `set obj1 = New Object1`
- Sample Code:

```
Public Property Let Name(vdata As String)
    strname = vdata
End Property

Public Property Get Name() As String
    Name = strname
End Property

Public Property Set Myobject(vdata as object)
    Set mvarMyObject = vdata
End Property
```

- Methods are prefixed by either the keyword 'Function' or 'Sub'. Subroutines do not have a return value.
- Events of a control are suffixed into the name of the control with an underbar (_). Eg. Command1_Click() is the Event method for the Click event of the Command1 command button.
- Every class has a Class_Initialize() and a Class_Terminate() event that is executed at the initialization and termination stages of the class, respectively.

8. CREATING CLASSES

8.1. *Create a Class Module*

- Class modules contain code that define the properties and methods of the class.
- Classes can be created using the Class Builder Utility under the Add-in Manager.

8.2. *Create Properties and Methods*

◊ Properties and Methods can be added using the Class Builder Utility or they can be manually edited into the class module.

9. LAB

Optional Lab 1: Write a simple VB code and debug it.

Optional Lab 2: Write a simple VB code using a Form.

10. REVIEW

CHAPTER 2: CREATING AND RUNNING ACTIVE-X CLIENT COMPONENTS

1. OBJECTIVES

In this chapter you will review the steps needed to develop:

◊ Active-X clients that use Automation objects

2. INTRODUCTION

◊ A client application is code that accesses the properties and methods of other Automation objects.

◊ The Object Variable must be declared using the ProgID of the object. The ProgID is made up of the ProjectName and the ClassName concatenated with the dot (.) symbol. Eg. "Excel.Application"

◊ Standard applications, such as MS Word and MS Excel, can provide Automation objects that can be accessed by client applications.

◊ Client applications get information about Automation objects through the type libraries associated with the Automation object. Type libraries provide information about the Interfaces supported by the object, descriptions of the properties, methods, and events provided by the object, the return types and parameter types of the methods and events, etc. This information for Active-X components can be viewed through the Object Browser.

◊ The structure of the inter-relationships of objects within the type library is depicted by the Object Models.

3. CREATING A CLIENT COMPONENT

◊ Create a Standard executable VB Project

◊ Set a reference to the component library

◊ Declare and create an object variable representing the component

◊ Use the object's methods and properties

4. Declaring and Creating an Object Variable of Server Component

◊ Use CreateObject(ProgID) method to instantiate an object.

```
Dim obj as Excel.Application
Set obj = CreateObject("Excel.Applicaton")
```

◊ Use the New keyword.

```
Dim obj as Excel.Application
Set obj = New Excel.Applicaton
```

5. Lab

Optional Lab 3: Using object browser to view Automation objects.

Optional Lab 4: Write VB client application to access Microsoft Excel's Automation objects.

6. Review

CHAPTER 3: CREATING AND RUNNING ACTIVE-X SERVER COMPONENTS

1. OBJECTIVES

In this chapter you will review the steps needed to develop:

◊ Active-X server components in the Visual Basic (6.0) development environment.

2. ACTIVE-X SERVER COMPONENTS

◊ Active-X server components are libraries of objects that can be used by client components.

◊ These have a Project Type of Active-X DLL.

3. CREATING AN ACTIVE-X SERVER COMPONENT

◊ Create an Active-X dll project

◊ Create a Class Module

◊ Create functions and properties (defined in the abstract class) in the new class.

◊ Compile the component or run the VB project to register the class for debugging.

4. CREATE A CLIENT APPLICATION THAT USES THE ACTIVE-X SERVER

◊ Create a standard executable which references the Active-X Server

◊ Dimension and create an instance of the class.

◊ Access functions and properties of the class.

5. LAB

Optional Lab 5: Create an active-x server and a client application.

6. REVIEW

CHAPTER 4: CREATING ACTIVE-X SERVER COMPONENTS SUPPORTS INTERFACES

1. OBJECTIVES

In this chapter you will review the steps needed to develop:

◊ Active-X server components with an interface in the Visual Basic (6.0) development environment.

2. ACTIVE-X SERVER COMPONENTS

◊ Active-X server components are libraries of objects that can be used by client components.

◊ These have a Project Type of Active-X DLL.

3. INTERFACES

◊ Interfaces are abstract classes. Abstract classes act as templates because they provide function and property definitions without any implementations.

◊ An interface specifies the functionalities guaranteed by an object.

◊ Interfaces allow objects to grow in functionality without breaking the earlier functionalities, as long as they continue to implement the original interfaces.

4. CREATING AN ACTIVE-X SERVER COMPONENT WITH INTERFACE

4.1. *Create an abstract class for the interface*

◊ Create an Active-X dll project.

◊ Create a class module with the name of the abstract class and interface.

◊ Create the functionalities of the interface by adding methods and properties to the abstract class. No implementation code must be included.

◊ Compile the code or run the VB project to register the class for debugging .

4.2. *Create an implementation of the abstract class or interface*

- ◊ Create an Active-X dll project that references the abstract class created above.
- ◊ Use the 'Implements' keyword followed by the name of the abstract class inside the implementation class module. This class then implements the interface described by the abstract class.
- ◊ Implement the functions and properties (defined in the abstract class) in the new class.
- ◊ Compile the component or run the VB project to register the class for debugging.

4.3. *Create a client application that uses the implementation*

- ◊ Create a standard executable which references the abstract class and the implementation class.
- ◊ Dimension variables of the abstract class as well as the implementation class.
- ◊ Set the abstract class variables to point to the existing implementation class objects.
- ◊ The functionalities available at each object depends on its class definition, even though they point to the same object.

5. LAB

Optional Lab 5: Create an interface, an implementation, and a client application.

6. REVIEW

CHAPTER 5: DATA MODEL - FUNDAMENTALS

1. OBJECTIVE

In this chapter, you will learn

- ◊ the Data Model terminology
- ◊ how to interpret the UML diagrams

2. DATA MODEL IN OBJECT-ORIENTED TERMINOLOGY

- ◊ Every software works with data using methods or routines. Data and methods can be grouped to form self-contained objects that interact with other objects.
- ◊ A class is a template for an object. Multiple objects of a given class can be used in a project. A project may also include a number of different classes. These classes can be related to one another. The relationship among the classes represents the Data Model.
- ◊ The Static Data Model describes the nature and inter-relationships of the various classes in a project. The Data Model can be represented as a Class Diagram using Unified Modeling Language (UML). UML is an emerging language for creating models of object-oriented computer software.

3. DATA MODEL REPRESENTED USING UML

3.1. *Object Classes*

- ◊ The Data Model deals with classes of objects.
- ◊ An object class is a template for an object while an instance refers to a particular object created during the use of the software.
- ◊ An object is made up of attributes (member variables) and operations (member functions and subroutines).
- ◊ An object class is represented by a box called the class icon. It is a rectangle divided into three compartments.

- The topmost compartment contains the name of the class. Typically, the second compartment lists the attributes of the class, and the third compartment lists the operations. Normally, only relevant attributes and operations are listed in these compartments for space considerations.

3.2. *Relationships*

- Object classes in the Data Model are normally related to other object classes.
- These relationships include simple associations, aggregate relationships, and inheritance and are always denoted by a relationship line.
- Most of the relationships are maintained by member variables. One member variable in one of the objects will represent the unique identifier of the related object.

3.2.1. Association (Open Arrow)

- A simple association is a reference to an object inside another object.
- The direction of the open arrow indicates the direction of the navigation. The object class that maintains the information about the relationship is at the base of the arrow.
- Instances of both objects must exist before the relationship can be created.

3.2.2. Inheritance (Closed Triangular Arrow)

- In the inheritance relationship, the arrow is at the parent object, which is the superclass.
- The child object, the sub-class, derives from the parent class and inherits all of the attributes and operations of the parent class.
- The sub-class is a more specific kind of the superclass. The derived classes retain pointers to their parent classes.

3.2.3. Aggregation and Composition (Diamonds)

- Diamonds indicate Parent-Child relationships. The entity with the diamond is the Parent.
- The child exists only if the parent exists. For example, a Nozzle must have a parent Equipment. If the parent Equipment is deleted, the child Nozzle cannot exist.

3.2.4. Text

- The text on the relationship line shows the nature of the relationship.

3.2.5. Numbers

◊ Numbers at the ends of the relationship line indicate "multiplicity" or the number of instances of the respective object class that can exist in the relationship. For instance, the numbers between ModelItem and Representation indicate that one ModelItem may be related to (associated with) multiple (0...*) Representations. Some ModelItems do not have any representations (0), while some can have multiple representations (*).

4. REVIEW

CHAPTER 6: LOGICAL MODEL AUTOMATION (LLAMA)

1. OBJECTIVES

In this chapter, you will learn

◊ interpret the Logical Model Automation classes and properties from the Data Model

2. INTRODUCTION

The Logical Model Automation (Llama) is primarily an object model based on the SmartPlant P&ID DataModel. In order to facilitate proper use of Automation, Llama also incorporates some non-DataModel items.

3. ITEMS FROM THE DATAMODEL

3.1. *Object Classes*

◊ Each object class in the DataModel has two Visual Basic (VB) classes. The first represents the object class while the second represents a collection of the object class. The names of these classes are prefixed by an "LM". The LM stands for the Logical Model. For example, the PipeRun object class in the DataModel has two classes, namely LMPipeRun and LMPipeRuns, in VB.

◊ Llama does not provide classes for the *Join objects. These objects contain connection informaton for a many-to-many relationship between two object classes. For instance, the PlantItemGroupJoin object class holds connection information between the PlantItem and PlantItemGroup object classes, which are related through a many-to-many relationship. Llama provides this information in the same way that it supports relationships. This will be discussed in the section on Relationships.

◊ Certain relationships shown in the Data Model are not implemented in SPPID. For instance, the relationship between Nozzle and PipeRun and the many-to-many relationships indicated by the PipeRunJoin and SignalRunJoin tables are currently not implemented.

3.2. *Attributes*

◊ Every object has one or more attributes.

◊ Although many objects may use similar attributes (e.g. SP_ID), the attributes of an object are unique.

◊ Attributes from the DataModel are implemented in three different ways in Llama, depending on the datatypes of the attribute. These are the simple datatypes, the enumerated datatypes, and the unitted datatypes.

3.2.1. Simple Data Types

◊ The simple datatype attributes, such as string, date, long, etc, are implemented as standard Properties of the VB class.

◊ The Let and Get functions are typically defined for the Property, enabling the VB user to both assign the value and to read the value.

3.2.2.Enumerated Data Types

◊ The enumerated attributes are codelisted attributes. Each codelist has a number (an integer part) and a name (a string part) and these are listed in the 'Enumerations' table in the DataDictionary. The members of each codelist are listed in the 'codelists' table in the DataDictionary. They have a set of predefined values with corresponding indices for each valid value. The values and the indices together form the codelist associated with the attribute.

◊ In Llama, the enumerated attributes are named after the codelists that they represent.

◊ Each enumerated attribute has two Properties associated with it in Llama. One has the name of the attribute, and it can be used to either get or set the name (string part) of the value. The other has the name of the attribute suffixed with the word "Index", and can be used to get or set the integer part (index) of the value. The index zero (0) indicates a null value.

3.2.3.Unitted Attributes

◊ The unitted attributes are the formatted attributes.

◊ They are composed of a numeric value and a text string depicting the unit of measure. The numeric value is an arbitrary number selected by the user, but the unit of measure must be recognized by SmartPlant P&ID as one of the defined formats.

- In Llama, the unitted attribute has two Properties associated with it. One has the name of the attribute, and it is the concatenation of the floating point value and the unit of measure as a single string. It can be used to set and get the value of the attribute in string form. The other property has the name of the attribute suffixed with the word "SI" and it yields the numeric value converted into the SI units for that quantity. This latter function is read-only and has only the Get part of the property.

3.3. *ItemAttribution*

- The relationship between objects allow objects to own attributes that are unique to other objects through various relationships. These derived attributes are part of the 'Item Attributions' of that item.
- The inheritance relationship naturally includes the attributes of the superclass into the Item Attributions of the subclass. For instance, the ItemAttributions of the Equipment class include attributes of PlantItem because of the inheritance relationship between Equipment and PlantItem.
- An association relationship between two items can permit one or more attributes of one class to be Item Attributes of the other. These Item Attributes have to be explicitly added in order for the attribute to be available from the object. The relationship between Case and ModelItem allows Equipment to have some Case attributes in its ItemAttributions list because of the ItemAttributions created between the two classes. However, other object classes that inherit from ModelItem do not have to own the same Case attributes and therefore do not have these attributes as part of their itemAttributions list.
- Each item attribution has a Name, a DisplayName, and an ID among other properties. If the item attribution is displayable through the property grid, then the DisplayName will be displayed in the property grid for the particular item.
- The ItemAttribution name is generally based on the path used to navigate to the property from a given item. However, this format is not a requirement. The only requirement is that the name must be unique for a given object class.
- Item Attributions not directly owned by the item are maintained in memory for performance reasons. Starting V3.0, first time user tries to access these Item Attributions will make these Item Attributions available for the object, no special trigger is needed. In addition, if user get the object using PIDDatasource, then all Item Attributions will be in the collection by default.

3.4.

3.5. *Relationships*

◊ Llama implements the relationships defined in the DataModel through properties. These properties appear in the 'LM' class of the object class. The type of property that appears depends on the multiplicity of the specific end of the relationship.

3.5.1. One-to-Many Relationship

◊ In a one-to-many relationship, the LM class on the 'one' end of the relationship will have a property that returns a collection of the item that appears on the 'many' end of the relationship. As an example, in the one-to-many relationship between Equipment and Nozzles, LMEquipment has a property called Nozzles, which returns all of the nozzles associated with the given instance of the LMEquipment.

◊ In the item on the many end of the relationship, there is a property that returns the unique item that it is related to. In the above example, the LMNozzle has a property that returns the EquipmentObject that it is related to.

3.5.2. Inheritance Relationship

◊ Inheritance relationships are visible through the 'additional' properties inherited by the sub-class LM object:

◊ The sub-class LM object contains every property of the superclass. For example, LMEquipment has a property called ItemTag, which it inherited from its superclass, LMPlantItem. LMVessel, which is a sub-class of LMEquipment, also inherits the ItemTag property from LMPlantItem.

◊ The subclass LM object inherits relationships of its superclass. For example, LMPlantItem has a property called Cases, which returns a collection of Cases, because it inherits the one-to-many relationship between its superclass ModelItem and Case in the DataModel.

3.5.3. Many-to-Many Relationships

◊ In a many-to-many relationship, the LM classes associated with both the object classes have properties that return a collection of the other LM class. For example, LMPlantItem has a property called PlantItemGroups, and LMPlantItemGroup has a property called PlantItems.

3.5.4. Group Relationships

- ◊ In a group or parent-child aggregation relationship, a child object has only one parent while a parent can have many children. Consequently, most parent-child aggregations are typically one-to-many relationships.
- ◊ Llama provides a property that returns a collection of the 'many' object in the group relationship. Although some parent-child relationships have a one-to-one relationship or even a one-to-none relationship, a collection is still provided. For example, the PipingComp and InlineComp have a parent-child aggregation relationship with multiplicity of 0..1. This implies that an InlineComp can have at most one PipingComp parent and the PipingComp can have at most one InlineComp child. Although the latter condition suggests a single InlineComp for a PipingComp, Llama still supports a property that returns a collection of LMInlineComps for a given PipeRun. This collection may contain only one object, consistent with the multiplicity of the relationship.

4. ITEMS OUTSIDE OF THE DATA MODEL (LMA)

In addition to the object classes provided in the DataModel, Llama provides some additional classes to enable the use of Automation.

4.1. *LMADatasource*

- ◊ The LMADatasource is the primary object in Llama because it provides the connection to the database.
- ◊ The LMADatasource establishes connection to the database using the Plant Name in SPManager. The ProjectNumber is the property that is set equal to the active SPManager Plant Name as default. To switch to a new SmartPlant P&ID Plant, user must set a the exactly name of the Plant to the ProjectNumber of the LMADatasource
- ◊ There are three sources to get LMADatasource. First is by setting a new LMADatasource, second is get PIDDatasource from the active drawing, third is get from SPPID when running validation program.
- ◊ A New LMADatasource must be created in a standalone project, unless the datasource can be obtained from another process. In Calculation/Validation, a New Datasource does not have to be created because the Active-X dll receives an initialized LMADatasource from the SPPID.

◊ The three properties IPile, Pile, and PIDMgr have non-Llama return types and can be used to get the connections to the Data Coordinator and PIDObjectManager or to set the connections to existing connections. The typical Llama user will not have to use any of these properties. These are for advanced uses.

4.2. *LMAItem*

◊ The LMAItem is a generic item in Llama. It has an interface that is supported by every object class in Llama. Therefore, an LMVessel can be converted to LMAItem.

◊ Every LMAItem represents the object class it was converted from. Typically, one would use the ItemType to determine the type of item to 'get' from the datasource. For instance, LMVessel as an LMAItem will yield an ItemType of 'Vessel' while the same vessel item when retrieved as an LMEquipment and then converted into LMAItem will yield an ItemType of 'Equipment'. However, this is not TRUE anymore since V4. In V4, all model item object will be obtained in its concrete level. For example, even retrieved as n LMEquipment, its ItemType is "Vessel". In addition, the object will have all ItemAttributions collection as its concrete object.

4.3. *LMAAttribute*

◊ An LMAAttribute is a generic attribute in Llama and uses the ItemAttribution. It has a Name and a Value. These are applicable for all types of attributes.

◊ Additional properties such as Index and SIValue are applicable for enumerated (codelisted) and unitted (formatted) attributes, respectively.

◊ The ISPAttribute is a property is reserved for advanced uses and is not supported in Llama.

◊ Typically, an LMAAttribute can be identified from the LMAAttributes collection by using the Name as listed in the Item Attributions table.

4.4. *Labs*

Lab 1: Initialize LMADataSource and access its properties.

Lab 2: Identify an Item and read its properties.

Lab 3: Modify the properties of an Item.

Lab 4: Propagation.

Lab 5: Rollback.

Lab 6: Init Objects Read Only.

Lab 7: Get LMAItem from LMVessel.

Lab 8: Access ItemTypes.

Lab 9: Access LMAAttributes Collection.

Lab 10: Access ItemAttributions in Details.

Lab 11: Change Site and Plant

4.5. *LMACriterion*

The LMACriterion is an object class that is typically used with the LMAFilter class.

- ◊ A criterion relates an attribute with a value. The criterion can be fully defined by setting values for at least three properties, namely SourceAttributeName, ValueAttribute, and Operator.
- ◊ The SourceAttributeName is the name of the attribute from the ItemAttributions table.
- ◊ The ValueAttribute is the value of the attribute, it also can be a string containing a symbol such as "%","_" etc.
- ◊ The Operator indicates the relationship between the attribute and its value. The operator is a string containing a symbol such as "=", "<", ">=", "like", "is" etc.
- ◊ The SourceAttributeID is the ID of the ItemAttribution.
- ◊ The SIValue can be specified in the case of formatted attributes.
- ◊ The ValueDisplayString is value of formatted attributes as displayed in the PropertyGrid.
- ◊ The Conjunctive must be set to True if this LMACriterion must be AND-ed with other LMACriterions.

4.6. *LMAFilter*

- ◊ LMAFilter.Criteria.Add Criterion adds the Criterion into the Criteria collection in the Filter.
- ◊ LMAFilter.Criteria.AddNew without an argument will create an new LMACriterion to the filter.
- ◊ The LMAFilter.Criteria collection has default indices starting with 1.
- ◊ If a specific *vntindexkey* is to be assigned, then it can be given as LMAFilter.Criteria.AddNew("3") or LMAFilter.Criteria.AddNew("Special").

- ◊ A counter key is still available for an LMACriterion based on the number of Criteria that existed prior to its creation.
- ◊ If a Criterion is deleted from the list, the counter key is renumbered but the list maintains the same sequence.

4.7. *LMAEnumAttList*

- ◊ An LMAEnumAttList in Llama is referred to a SelectList data.
- ◊ Property EnumeratedAttributes is a collection of SelectList value of this SelectList Data.
- ◊ User can get a collection of LMAEnumAttList from LMADataSource.CodeLists property.

4.8. *LMAEnumeratedAttribute*

- ◊ An LMAEnumberatedAttribute in Llama is referred to a SelectList Value in a Select data.
- ◊ Properties of LMAEnumberatedAttribute, such as Name and Index will return the Name and Index of a select value. This will be very useful when Llama user creates a LMAFilter in program, since Llama only takes Index value in Criterion.

5. GENERAL ISSUES

5.1. *Properties and Methods of Collections*

- ◊ *Property Count*: returns the number of items in the collection. For an empty collection, the Count is zero.
- ◊ *Property Item(ID as long)*: returns an item of the type that this collection holds. The argument is the IndexKey of the item. The IndexKey is generally the SP_ID of the item. If the IndexKey is wrong, the returned item is a Nothing.
- ◊ *Property Nth(index as long)*: selects the item in the sequence that it is stored in the collection.
- ◊ *Property Datasource*: returns the Datasource that was used to create the collection. This is important when a number of datasources are in use, each pointing to different plants or databases.
- ◊ *Property TypeName As String:* Returns the item type name of the items in the collection.
- ◊ *Sub Remove(ID as long)*: removes the item with the given ID, if it exists in the collection.

- ◊ *Sub Collect([DataSource As LMADataSource], [Parent As LMAItem], [RelationshipName As String], [Filter As LMAFilter]):* collects items of the kind represented by the collection from the specified DataSource. The filter can be used to identify specific properties for the collected items. The RelationshipName is the name given on the relationship lines in the DataModel.
- ◊ *Sub Clear():* Clears the collection.
- ◊ *Sub AddCollection(Items As ...)*: Add all items from the indicated collection into the current collection.
- ◊ *Function Add([Item As ...]) As ...:* Adds an item to the collection. Note that the argument and the return value are of the added Llama object.
- ◊ *Function AsLMAItems() As LMAItems*: Returns the LMAItems interface of the collection.
- ◊ The LMACriterions collection has a special *Function Add(NewObject As LMACriterion)* which returns a Boolean.

6. LABS

Lab 12: Access an item using filters with single criterion.

Lab 13: Access items using filters with multiple criteria.

Lab 14: Access items using filters with criterion on Select List Data.

Lab 15: Using Compound Filter

Lab 16: Access all filters defined in the plant.

Lab 17: Access SelectList Data.

7. REVIEW

CHAPTER 7: SPPID DATA MODEL

1. OBJECTIVES

In this chapter, you will learn

◊ the general structure of the SmartPlant P&ID Data Model and its implementation in the Llama object model

◊ highlights the relationships between the classes that need additional attention

2. INTRODUCTION

The objects exposed through Llama span all the three schemas used in the database. Objects in each grouping may come from different schemas. However, the object model relationships allow the user to access the related objects and obtain the appropriate properties without being concerned about the schema they belong to. Therefore, the most important task for the user of Automation is fully understand the various relationships between objects. This chapter addresses some objects and relationships that requires special attention from the user.

3. MODEL DATA

The Model Data maintains the engineering data. It is centered around the ModelItem and the PlantItem, which is also a ModelItem. Refer to the Data Model diagram for the relationships between the various objects. This section describes only the specially noteworthy and the not-so-obvious characteristics of the objects.

3.1. *ModelItem*

The ModelItem can own one or more instances of Alias, History, Event, Note, Source, and Status. In addition, it can own one or more Representations from the Drawing Data Model. Currently the software creates History records of HistoryType 'Creation' and 'Last Modification'. The user should not modify any of the History records maintained by the system nor create new ones of these types. However, the user can create new History records of a different type after creating new types in the datadictionary, if necessary. The Automation user can also create new instances of Alias, Event, Note, Source, and Status.

In addition, a ModelItem can own one or more Cases. These Cases are maintained by the software using the information provided in the DataDictionary and the ItemAttributions. The Automation user should not attempt to add new Cases, CaseProcesses, or CaseControls through code.

The ModelItem is the superclass from which all other Model Data classes are derived. The immediate subclasses of the ModelItem are the ItemNote, PipingPoint, PlantItem, SignalPoint, and OPC. Of these, the most significant is the PlantItem.

3.2. *Labs*

Lab 18: Read History Property of ModelItem.

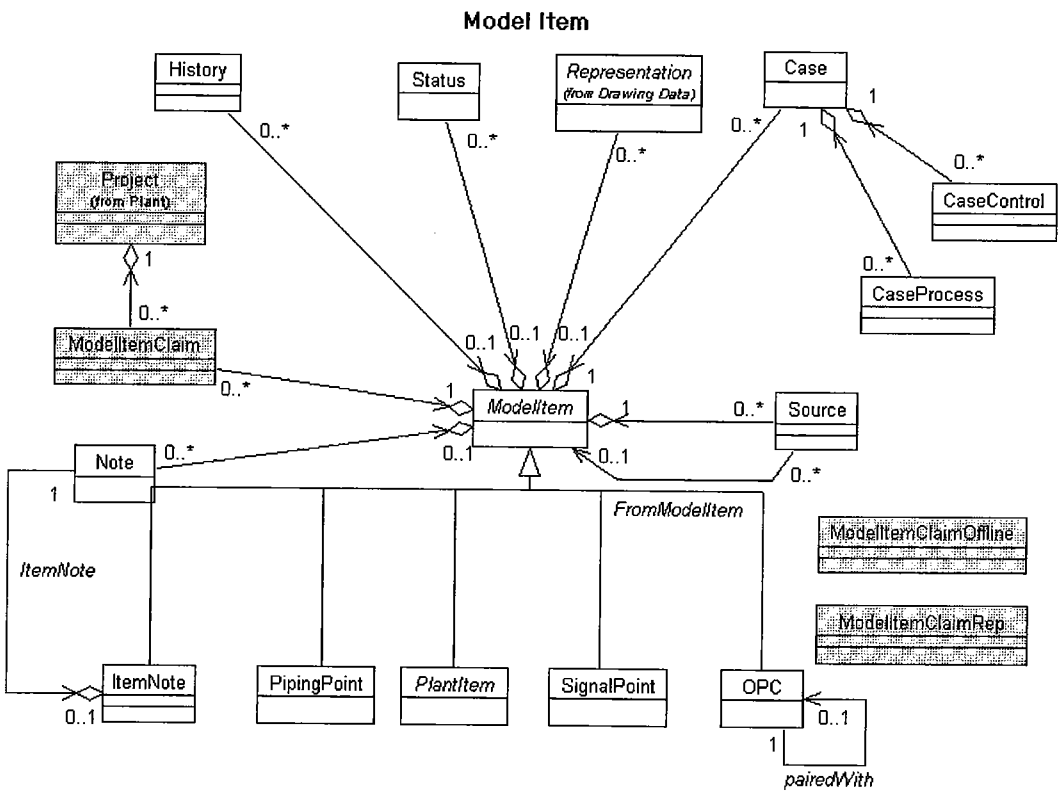
Lab 19: Read Status Property of ModelItem.

Lab 20: Read Case Property of ModelItem.

Lab 21: Access ItemNote.

Lab 22: Access OPC.

Lab 23: Filter for All Histories



Model Data: Model Item

3.3. *PlantItem*

Most items that are placed on drawings can be traced back to PlantItem. Subclasses of the PlantItem are Equipment, Nozzle, PipingComp, Instrument, Pipeline, Piperun, SignalLine, SignalRun, and PlantItemGroup.

3.3.1. PartOfPlantItemID and PartOfPlantItemObject

Returns the PlantItem that this current PlantItem is a Part Of. Examples are implied items, TEMA ends, and trays. An implied item is related to its parent item through this relationship. It also has a PartOf type = 'Implied'. A TEMA end is a part of the TEMA Shell that it is placed on and has a PartOf type = 'Composite'. Although a tray is a legitimate part of the vessel it is placed on, it does not have a PartOf type value. The value is NULL for a tray.

3.3.2. PlantItemGroup and PlantItem

PlantItemGroup is a subclass of PlanItem, concrete PlantItemGroup includes: Package, SafetyClass, System, InstrLoop, PlantItemGroupOther, and AreaBreak. PlantItemGroup has many to many relationship with PlantItem. For example, a Package can have two vessels and many piperuns, while one vessel can belong to multiple Packages.

3.3.3. Labs

Lab 24: Change properties at different object levels.

Lab 25: Read Case Property of Vessel.

Lab 26: Read Flow Direction of PipeRun

Lab 27: Access Piping Point.

Lab 28: Access Signal Point.

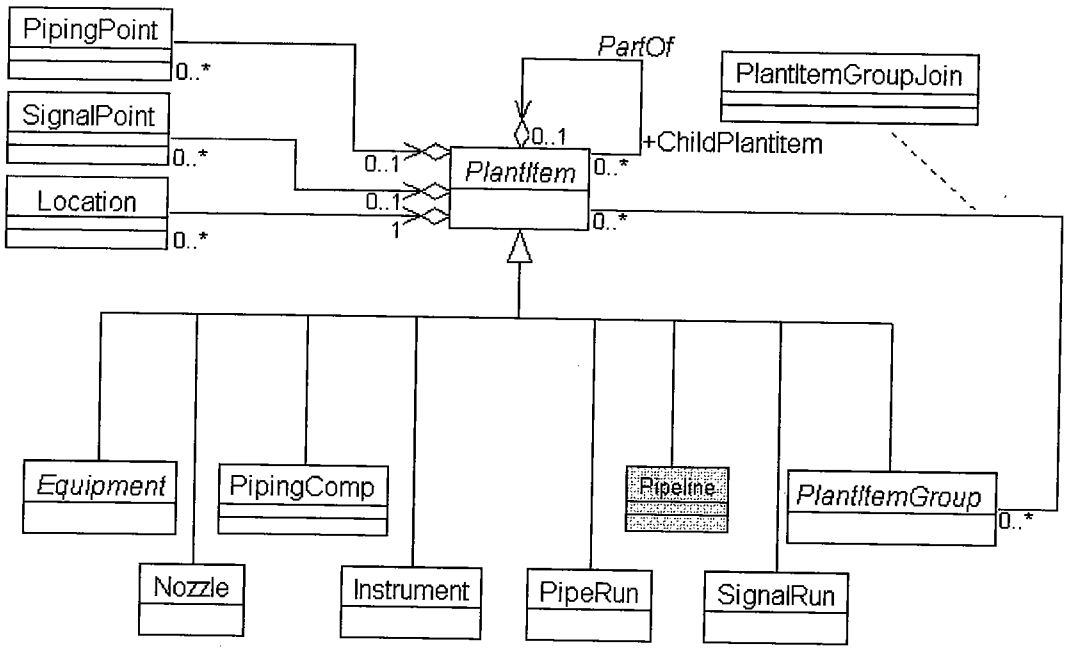
Lab 29: Implied Items.

Lab 30: Part of PlantItem relationships.

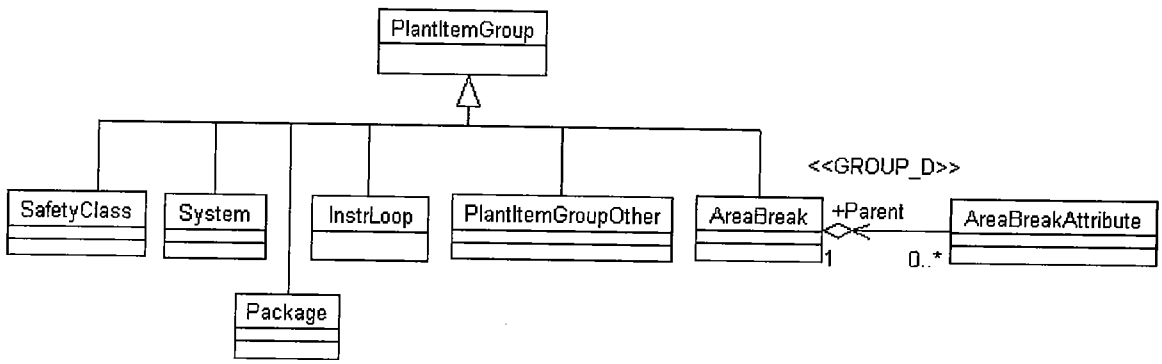
Lab 31: Access Instrument Loop.

Lab 32: LoadInstruments.

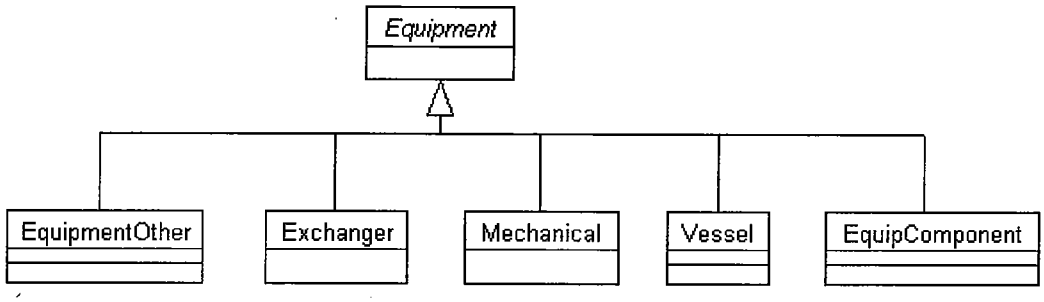
Plant Item Model



Model Data: PlantItem



Model Data: PlantItemGroup



Model Data: Equipment

3.4. *Logical Connectivity*

3.4.1. Nozzle - Equipment

A Nozzle is always owned by an Equipment object.

3.4.2.PipingComp – InlineComp

Every PipingComp has an associated InlineComp. The PipingComp owns its associated InlineComp. However, only the InlineComp has the information about the PipeRun that the PipingComp resides on.

3.4.3.Instrument – InlineComp

An Instrument can own an InlineComp if it is an Inline instrument. In this case, the Instrument owns its associated InlineComp. The InlineComp maintains the relationship to the PipeRun that the Instrument is on.

3.4.4. Instrument – PipeRun

When a PipeRun with PiperunClass equals to Instrument is connected with an off-line instrument, SP_PipeRunID in T_Instrument table will be populated with SP_ID of the PipeRun.

3.4.5. Instrument – SignalRun

This relationship is not implemented in the software.

3.4.6.Labs

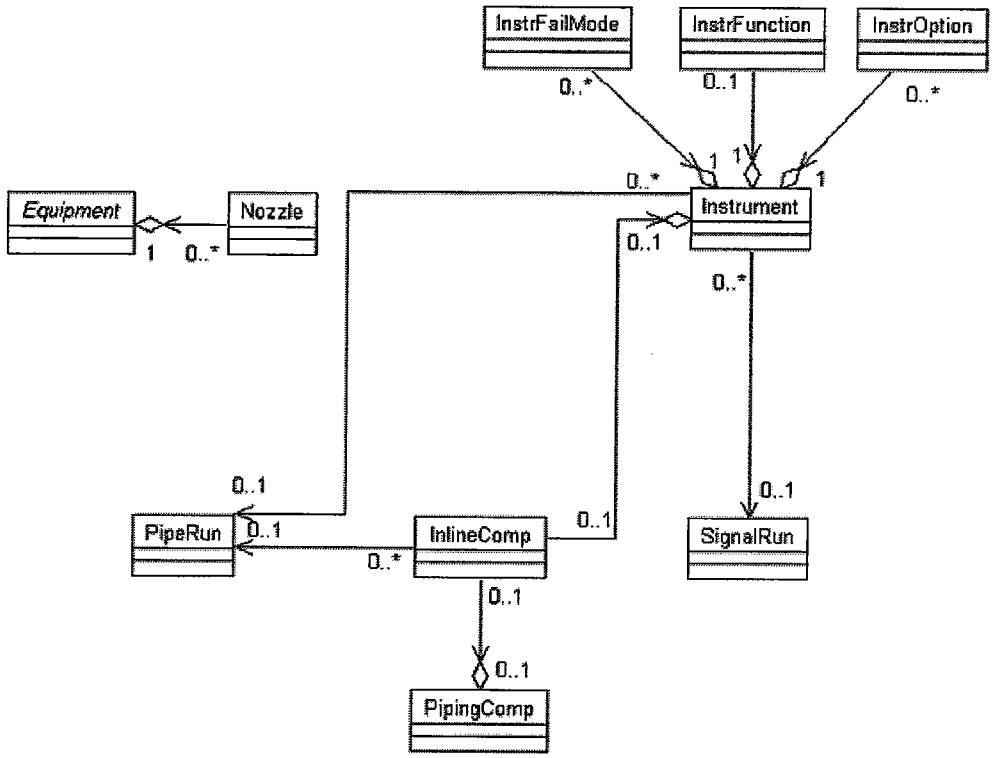
Lab 33: Identify Nozzle and Equipment.

Lab 34: PipingComp and InlineComp.

Lab 35: Instrument and InlineComp

Lab 36: Offline Instrument and SignalRun.

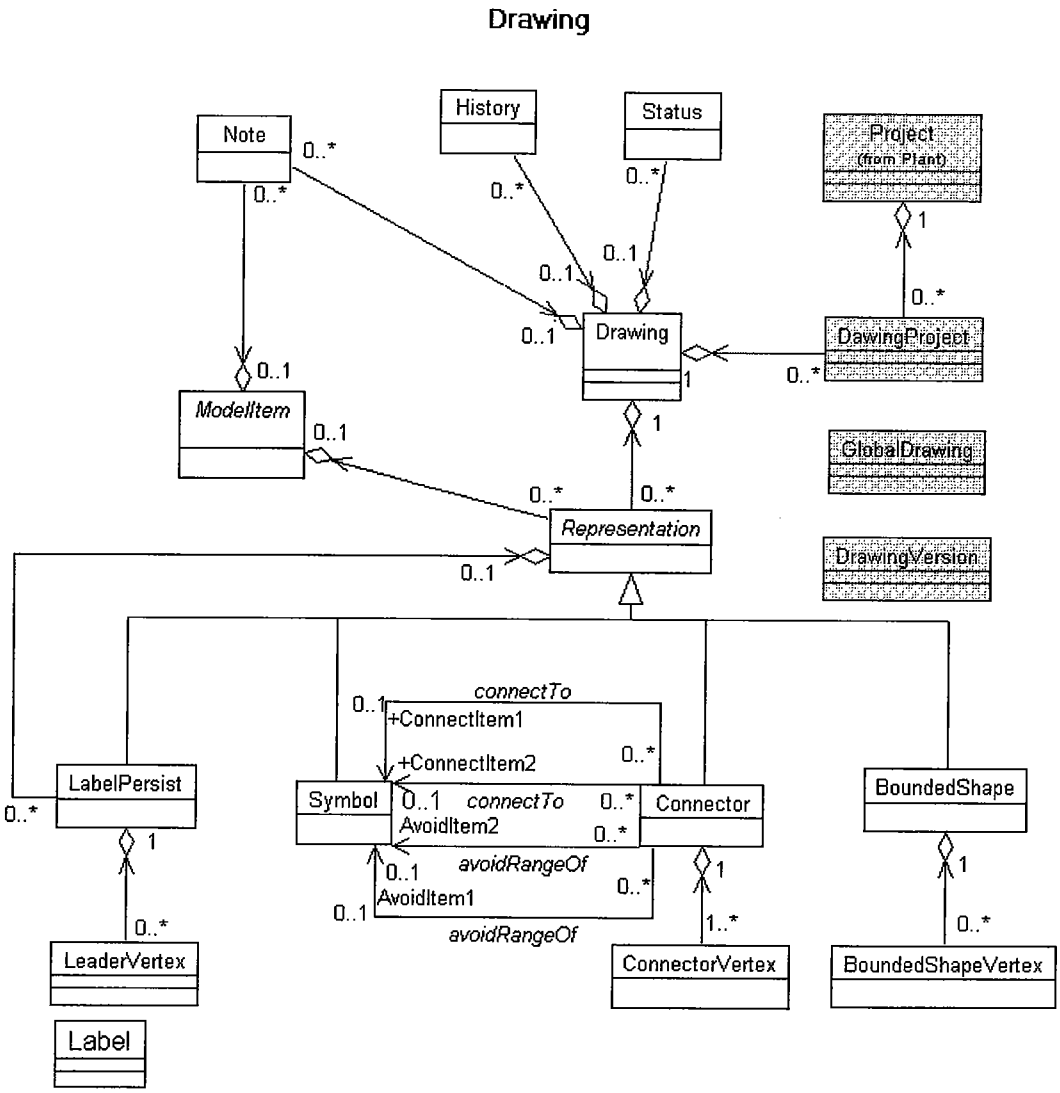
Lab 37: Access Instrument Functions.



Model Data: Connectivity

4. DRAWING DATA

The Drawing Data Model maintains the graphical representation data. It is centered around the Drawing and the Representation objects.



Drawing Data: Package Overview

4.1. *Drawing*

Drawing object can own one or more History, Status and Note. Currently the software creates History records of HistoryType 'Creation' and 'Last Modification'. The user should not modify any of the History records maintained by the system nor create new ones of these types.

4.2. *Representation*

A Representation belongs to a Drawing or Stockpile. Most representations also belong to some ModelItem. There are four types of Representations, namely LabelPersist, Symbol, Connector, and BoundedShape.

4.3. *LabelPersist*

A LabelPersist representation is created every time a label is placed on a drawing. This is the only representation that does not belong to a ModelItem. The LabelPersist representation can have a RepresentationType = 'Label'.

A LabelPersist not only is a Representation, it is also 'Related To' another Representation. This latter representation belongs to the ModelItem on which the label is placed.

For labels on Symbol, the label is labeling the Symbol. However, for labels on PipeRun, actually the label is labeling the Connector.

4.3.1. Labs

Lab 38: Identify connectors of a Piperun.

Lab 39: Find File Name of a Symbol.

Lab 40: Find X, Y Coordinates of Symbol.

Lab 41: Find X, Y Coordinates of Piperun.

Lab 42: Find Labels of a Symbol.

Lab 43: Find Parent Representation of a Label.

Lab 44: Find Parent Drawing of a Symbol.

Lab 45: Find Active Drawing and PlantItems in it.

Lab 46: Filter for Items In Stockpile.

4.4. *Symbol*

A Symbol representation is created every time an item is placed on a drawing. Every PlantItem, except Pipeline and SignalLine, has an associated Symbol representation. In addition, ModelItems such as OPCs and ItemNotes (annotations) also have symbols. When a PipeRun or a SignalRun is placed on a drawing, multiple representations are created. A Symbol representation is one of the representations placed at the creation of a run. A Symbol representation can have the following RepresentationTypes, namely Gap, OPC, Symbol, and Branch. 'Gap' stands for a gapping symbol placed at the junction of crossing runs. An 'OPC' type representation is placed for OPC ModelItems. Typically, most other

items have a Symbol representation of RepresentationType, 'Symbol'. A branch point on a PipeRun or a SignalRun produces a Symbol of RepresentationType, 'Branch'. The 'Branch' symbol behaves just like a regular symbol.

4.4.1.Connect1Connectors

A collection of connectors that are connected to this current Symbol representation by their End1 ends.

4.4.2.Connect2Connectors

A collection of connectors that are connected to this current Symbol representation by their End2 ends.

4.4.3. ConnectorVertices

A collection of Connector Vertex that holds graphic information for the Connector, such as X, Y Coordinates of the Connector. Pay special attention, X, Y Coordinates are not necessary always stored in the Connector Vertex, sometimes, depending on how the Connector is drawn, the Symbol connected with the Connector holds such information.

4.5. *Connector*

A Connector representation is created along with a Symbol representation when either a PipeRun or a SignalRun is placed, leading to a total of two representations. If an existing connector is split by placing an inline component on the run, then the original connector is deleted and replaced by two connectors, one on each side of the inline component. If the Piperun or SignalRun is deleted into the Stockpile, then all the Connector representations are deleted.

Each connector has two end points, namely the End1 and End2. The ends are named 1 and 2 in the direction that the connector was drawn. Each connector can be attached to a symbol on each of the two ends.

4.5.1.ConnectItem1SymbolID and ConnectItem1SymbolObject

This is the symbol representation to which the End1 of this current connector is attached.

4.5.2.ConnectItem2SymbolID and ConnectItem2SymbolObject

This is the symbol representation to which the End2 of this current connector is attached.

4.6. *BoundedShape*

A BoundedShape representation is created when placing an AreaBreak or Revision Cloud.

4.7. *Labs*

Lab 47: Identify items connected to a PipeRun.

Lab 48: Identify the PipeRun associated with the PipingComp.

Lab 49: Navigate down a piperun to a vessel.

Lab 50: Navigate through a branch point on a piperun.

Lab 51: Navigate through OPC

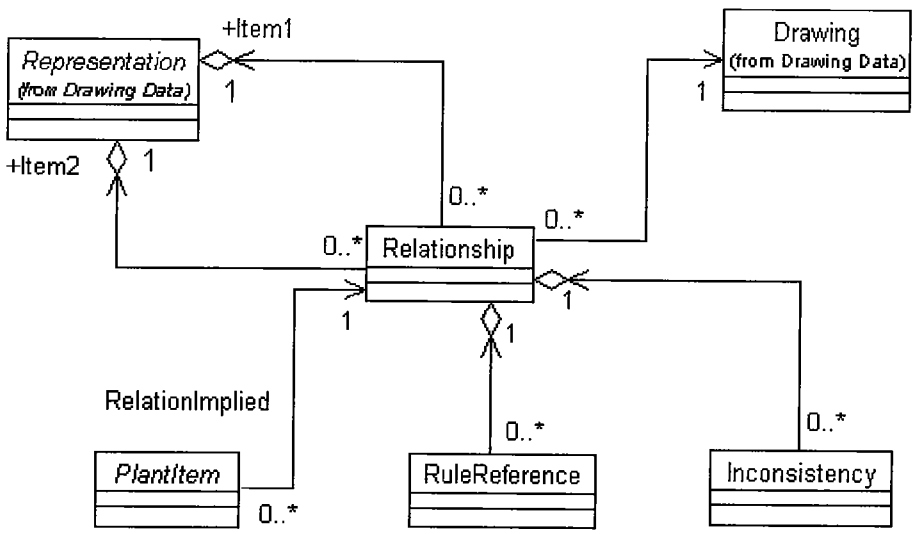
Optional Lab 7: Find OPC and From/To.

5. ADDITIONAL DATA MODEL

5.1. *Relationship*

A Relationship is created automatically between representations when

(1) Two graphics are connected. Example: PipeRun connected to Valve, Nozzle on a Vessel.
(2) When a symbol is placed that has piping points or signal points.



Relationship Model

5.2. *Labs*

Lab 52: Access Relationship from Representation

Lab 53: Access Inconsistency

Lab 54: Access RuleReference

5.3. *PlantGroup*

PlantGroup is a superclass with eight subclasses, namely Site, Plant, Unit, BusinessSector, Level, PlantSystem, SubSystem and Area. These are designated by PlantGroupTypes enumerated list in the DataDictionary. Unlike all other objects in the SPPID data model, new subclasses can be added to PlantGroups. These classes are persisted as tables with a 'SPM' prefix after the prefix 'T_'. For example, a user-defined PlantGroup SubArea corresponding table is T_SPMSubArea. These objects can be obtained as generic LMAItems. For instance, if SubArea is a user-defined PlantGroup that was included into a hierarchy and used in a plant structure, then the following code can be used to access the SubArea that has an ID="SP_ID":

```
Dim objItem As LMAItem
Set objItem = objDatasource.GetItem("SPMSubArea", "SP_ID")
```

The attributes of the SubArea will need to be obtained through the Attributes collection because they are user-defined.

5.3.1. ParentID and ParentType

ParentID returns the ID of the Parent PlantGroup of the current PlantGroup. It returns a negative 1 (-1) if it is the top-most item in the hierarchy. The ParentType is a number describing the type of the Parent PlantGroup.

5.3.2. PlantGroupType

It is an enumerated list describing the type of the PlantGroup.

5.3.3. PlantItems

A collection of PlantItems that have been associated with this PlantGroup. One example is when an Equipment Vessel is associated with a Unit. The Equipment would then appear in the Unit's PlantItems collection.

5.3.4. Drawings

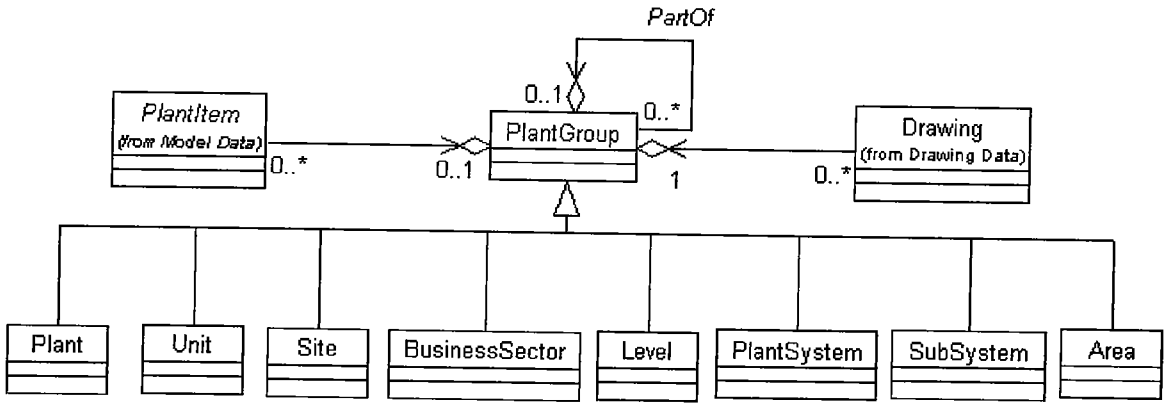
A collection of Drawings that are directly under the current PlantGroup.

5.3.5. Labs

Lab 55: Access PlantGroup from PlantItem.

Lab 56: Access PlantGroup from Drawing.

Lab 57: Access Customized PlantGroup.



Plant Hierarchy

5.4. *Workshare*

Workshare may be set up to share SPPID data among multiple locations or to have a Task/Master Plant. Workshare entities are created in PLANT schema. LLAMA is aware of the workshare. For example, if user tries to modify a property of an item that read-only because of workshare, the modification will not be persisted to database. However, there is no public interface exposed in LLAMA that user can check such as ownership of a drawing, or user's access right.

5.4.1.WSSite

Every plant, by default, is a workshare site. User can create multiple satellite workshare sites in the plant.

5.4.2.DrawingSite

This is a join table between T_WSSite table and T_Drawing table. Through this table, drawing knows who is it owner workshare site, and workshare site knows how many drawings belongs to it.

5.4.3.ActiveWSSite

A plant may have multiple workshare sites, but it can have only one active workshare site.

5.4.4.PlantItemGroup

A table contains plant item group information that related to a workshare site.

5.4.5.OPC

A table contains OPC information that related to a workshare site.

5.4.6.PlantGroup

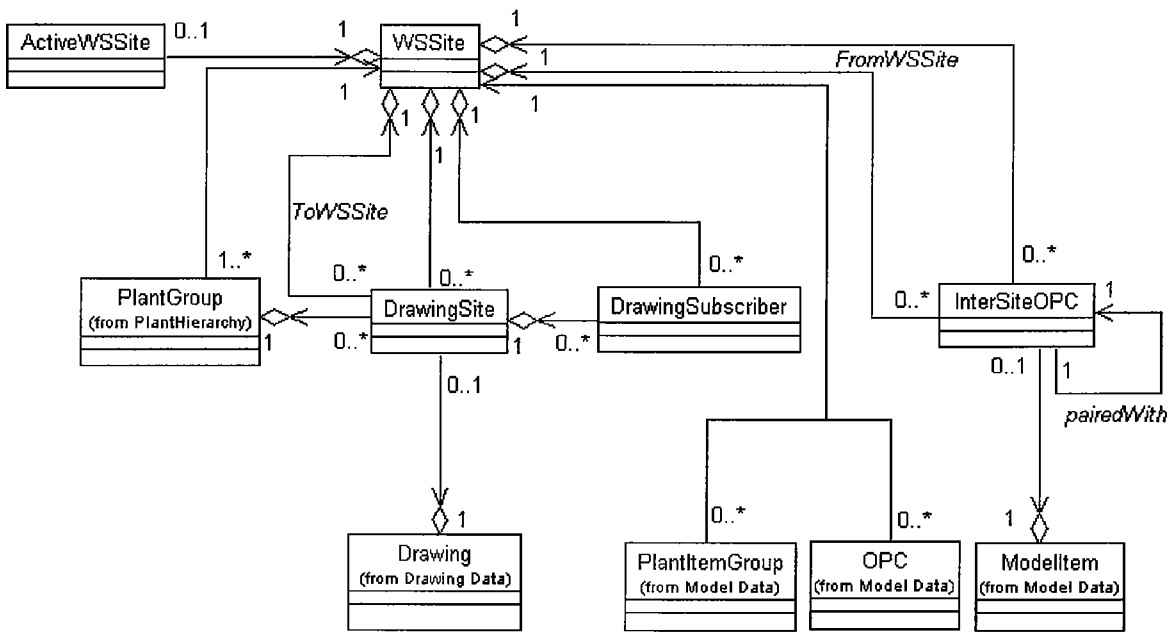
T_PlantGroup has a point that points back to the workshare site.

5.4.7.Labs

Lab 58: Access Workshare Site

Lab 59: Access DrawingSite

Lab 60: Workshare Awareness in LLAMA



Workshare Model

5.5. *As-Build/Project*

5.5.1. Project

Every plant, by default, is The Plant. Then, User can create multiple projects in the plant.

5.5.2. ActiveProject

The current in-use project is the active project. User can access active project through LMADatasource.GetActiveProject function. It can be The Plant or projects.

5.5.3. Project Status

Project status is a select list data with following lists: Active, Completed, Merged, Finished, Cancelled, Terminated, None, Deleted. The Plant status is None. Projects can have different statuses.

5.5.4. Claim Status

In As-Build/Project environment, an item can have different claim status: Not Claimed, Claimed by active project, and Claimed by Others.

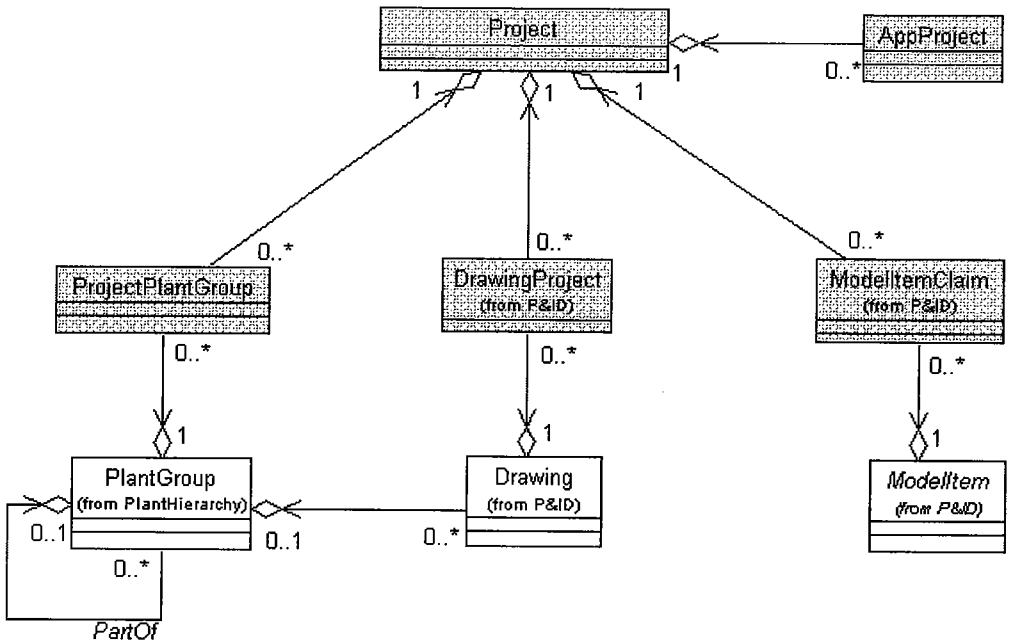
5.5.5. Labs

Lab 61: Access Active Project

Lab 62: Access Plant from Project

Lab 63: Access Claim Status of Items

Project Model



Project Model

5.6. *T_OptionSetting Labs*

Lab 64: Access OptionSettings

5.7. *Special Issues*

5.7.1. Model Item always returns in its concrete level

Since V4, all model item object will be obtained in its concrete level. For example, even retrieved as n LMEquipment, its ItemType is "Vessel". In addition, the object will have all ItemAttributions collection as its concrete object.

CHAPTER 8: PLACEMENT AUTOMATION

1. OBJECTIVES

In this chapter, you will learn

◊ an overview of the Placement Automation library

2. PLAICE LIBRARY

2.1. *Create an Item in the Stockpile*

Function PIDCreateItem(DefinitionFile As String, [DrawingID="0"]) As LMAItem

DefinitionFile: the path and name of symbol file

DrawingID: SP_ID of drawing, the item will be created in its stockpile. If user leaves it blank, by default, value is 0, which means plant stockpile. If you user specify a valid drawing SP_ID, the item will be created in that drawing's stockpile.

2.2. *Place an Symbol on the Drawing*

Function PIDPlaceSymbol(DefinitionFile As String, X As Double, Y As Double, [Mirror], [Rotation], [ExistingItem As LMAItem], [TargetItem]) As LMSymbol

DefinitionFile: the path and name of symbol file

X: x-coordinate of where the symbol locates on Drawing, unit in METER

Y: y-coordinate of where the symbol locates on Drawing, unit in METER

[Mirror]: True or False

[Rotation]: 1.57=90 Degree

[ExistingItem]: Item as LMAItem in stockpile/drawing to be placed on Drawing

[TargetItem]: Related item as LMRepresentation of the symbol to be placed, for example, when place a nozzle symbol, a Equipment item must be specified as TargetItem. (When

TargetItem is specified, given X, Y coordinates may be ignored, software will locate a new X, Y coordinates for the symbol to be located on the edge of TargetItem, if PIDSnapToTarget is set to TRUE)

2.3. *Place Labels*

Function PIDPlaceLabel(DefinitionFile As String, Points() As Double, [Mirror], [Rotation], [LabeledItem As LMRepresentation], [IsLeaderVisible As Boolean = False]) As LMLabelPersist

DefinitionFile: the path and name of label file

Points(): a dynamic array, specifies the X, Y coordinates for Label to be placed on drawing

[Mirror]: True or False

[Rotation]: 1.57=90 Degree

[LabeledItem]: LMRepresentation of targetitem on which the label to be placed

[IsLeaderVisible]: specifies the Leader of the label to be placed is visible or not

2.3.1. Labs

Lab 65: Create a Vessel and place into StockPile.

Lab 66: Place a Vessel on a Drawing.

Lab 67: Place Nozzles and Trays on a Vessel.

Lab 68: Place Labels on a Vessel.

Lab 69: Place OPC

Lab 70: Place OPC from StockPile

2.4. *Place Piperun*

Function PIDPlaceRun(StockpileItem As LMAItem, Inputs As PlaceRunInputs) As LMConnector

StockpileItem: Piperun placed in the stockpile as LMAItem

Inputs: Collection of input parameters

2.5. *Auto Join Piperuns*

Sub PIDAutoJoin(RunItem As LMAItem, AutoJoinEnd As AutoJoinEndConstants, SurvivorItem As LMAItem)

RunItem: Exiting Piperun as LMAItem to seek AutoJoin other piperuns

AutoJoinEnd: Enum data as values are: autojoin_both, autojoin_start, autojoin_end, and autojoin_none.

SurvivorItem: When two piperuns auto joined, one piperun will disappear while the other will survive, the survived one is the SurvivorItem.

2.6. *Place Bounded Shapes*

Function PIDPlaceBoundedShape(DefinitionFile As String, Points() As Double, [ExistingItem As LMBoundedShape]) As LMBoundedShape

DefinitionFile: the path and name of symbol file

Points: a dynamic array, specifies the X, Y coordinates for BoundedShape to be placed on drawing

ExistingItem: Item in stockpile to be placed on Drawing

2.7. *Place Assemblies*

Function PIDPlaceAssembly(AssemblyFile As String, X As Double, Y As Double) As LMAItems

AssemblyFile: the path and name of symbol file

X: x-coordinate of where the symbol locates on Drawing, unit in METER

Y: y-coordinate of where the symbol locates on Drawing, unit in METER

2.8. *Place Gaps*

Function PIDPlaceGap(DefinitionFile As String, GapX As Double, GapY As Double, ParamLeft As Double, ParamRight As Double, [objLMConnector As LMConnector], [RotationAngle], [ExistingRun As LMAItem]) As LMSymbol

DefinitionFile: the path and name of symbol file

GapX: x-coordinate of where the Gap locates on Drawing, unit in METER

GapY: y-coordinate of where the Gap locates on Drawing, unit in METER

ParamLeft: length of Gap to the Left, unit in METER

ParamRight: length of Gap to the Right, unit in METER

[ObjLMConnector]: target PipeRun on which the Gap is placed

[RatationAngle]: 1.57=90 Degree

[ExistingRun]**: not working now

2.8.1. Labs

Lab 71: Using PIDPlaceRun to Place Piperun

Lab 72: Using PIDAutoJoin to Auto Join Two Piperuns

Lab 73: Place Gap.

Lab 74: Place BoundedShape

Lab 75: Place Assembly.

2.9. *Remove a Symbol from the Drawing*

Function PIDRemovePlacement(Representation As LMRepresentation) As Boolean

Representation: LMRepresentation of the symbol on Drawing to be removed into stockpile

2.10. *Delete an Item from Model*

Function PIDDeleteItem(Item As LMAItem) As Boolean

Item: LMAItem of the symbol (on Drawing) or item (in stockpile) to be delete from Model

2.11. *Replace Symbol*

Function PIDReplaceSymbol(NewSymbolFileName As String, ExistingSymbol As LMSymbol) As LMSymbol

NewSymbolFileName: the path and name of symbol file used to replace the existing symbol

ExistingSymbol: LMSymbol of symbol on the Drawing to be replaced

2.12. *Replace Label*

Function PIDReplaceLabel(NewSymbolFileName As String, ExistingLabel As LMLabelPersist) As LMLabelPersist

NewSymbolFileName: the path and name of label file used to replace the existing Label

ExistingLabel: LMLabelPersist of Label on the Drawing to be replaced

2.13. *Replace OPC*

Function PIDReplaceOPC(NewSymbolFileName As String, objExistingOPC As LMSymbol) As LMSymbol **Not working now

NewSymbolFileName: the path and name of OPC used to replace the exising OPC

objExistingOPC: LMSymbol of OPC on the Drawing to be replaced

(However, pairOPC is not replaced)

2.14. *Apply Parameters to Parametric Symbols*

Sub PIDApplyParameters(Representation As LMRepresentation, Names() As String, Values() As String)

Representation: LMRepresentation of parametric symbol on Drawing, whose parameters are to be modified

Names(): a dynamic array, specifies the names of parameter, for example: TOP, RIGHT

Values(): a dynamic array, specifies the values of parameter

2.15. *Locate Connect Point*

Function PIDConnectPointLocation(Symbol As LMSymbol, ConnectPointNumber As Long, X As Double, Y As Double) As Boolean

Symbol(): LMSymbol of the symbol to be located of connect point

ConnectPointNumber: Index number of the connect point on a symbol

X: x-coordinate of the connect point on drawing, unit in METER

Y: y-coordinate of the connect point on drawing, unit in METER

2.16. *Datasource*

Property PIDDataSource As LMADataSource

Property returns a LMADataSource object

2.17. *PIDSnapToTarget*

Property PIDSnapToTarget As Boolean

Property specifies item to be placed snap to targetitem or not, TRUE by default

2.18. *SetCopyPropertiesFlag*

Function PIDSetCopyPropertiesFlag(bDoCopy As Boolean) As Boolean

Property specifies the properties to be copied or not as defined in the Rule across the Items that are connected.

2.18.1. Labs

Lab 76: Remove Vessel.

Lab 77: Delete Vessel.

Lab 78: Replace Symbol.

Lab 79: Replace Label.

Lab 80: Replace OPC.

Lab 81: Modify Parametric Symbol.

Lab 82: Using PIDConnectPointLocation to Locate X, Y Coordinates.

Lab 83: Create Instrument Loop.

Lab 84: Find and Replace Label.

2.19. *Datasource*

Property PIDDataSource As LMADataSource

Property returns a LMADataSource object

2.20. *Place Connectors*

Function PIDPlaceConnector(DefinitionFile As String, Points() As Double, [OrthogonalStart As Boolean = True], [OrthogonalEnd As Boolean = True], [ExistingItem As LMAItem], [OriginItem], [DestinationItem], [OriginIndex As Long = 1], [DestinationIndex As Long = 1], [MinSegLength As Double = 0.001], [RangeClearance As Double = 0.004]) As LMConnector

DefinitionFile: the path and name of symbol file

Points(): a dynamic array, specifies the X, Y coordinates for Connector to be placed on drawing

[OrthogonalStart]: specifies if Orthogonal to OriginItem

[OrthogonalEnd]: specifies if Orthogonal to DestinationItem

[ExistingItem]: LMAItem of existing piperun in the same drawing

[OriginalItem]: LMRepresentation of original item where connector starts

[DestinationItem]: LMRepresentation of Destination item where connector ends

[OriginIndex]**: not working now

[DestinationIndex]**: not working now

[MinSegLength]: specifies the minimum segment length to drawing the connector

[RangeClearance]: specifies the avoid range when place the connector

(when specifies the OriginalItem and DestinationItem, the first point's and last point's X, Y coordinates will be ignored)

(When placing a connector to a symbol, how to control which connect point on symbol the connector should be connected? Instead of using OriginIndex and DestinationIndex, specifying the X, Y coordinate for connector point you wish to connect with in Points(), which will make the connector to connect the connect point of the symbol)

3. Misc.

When placing a piping component on a piperun connector, the original connector is given an ItemStatus of 'Delete – Pending' or ItemStatusIndex = 4 and two new connectors are created.

When placing the next item on the connector, you cannot use the reference to the original connector because it is no longer valid. We have to find the two new connectors and then use their references. This can be achieved by looking at the symbol placed and finding the item1connectors or item2connectors.

When place a Gap on a piperun, the symbol does not break the original connector either. It creates two more new connects and sits on the top of the piperun. When user select not to see the Gap, the Gap along with two connectors it created will go to hidden layer and original connector is showing.

4. LABS

Optional Lab 8: Find and Replace

CHAPTER 9: USING PIDAUTOMATION TO CREATE, OPEN AND CLOSE DRAWINGS

1. OBJECTIVES

In this chapter, you will learn

◊ Using PIDAutomation to create, open and close drawings.

2. THREE IMPORTANT FUNCTIONS

2.1. *PIDAutomation.Application.Drawings.Add*

Function Add(PlantGroupName As String, TemplateFileName As String, DrawingNumber As String, DrawingName As String, [DocumentTypeValue], [DocumentCategoryValue], [bVisible As Boolean = True]) As Drawing

PlantGroupName: Name of PlantGroup that is just above drawing in Plant Structure Hierarchy

TemplateFileName: Full path and name of drawing template file

DrawingNumber: Drawing number of the drawing to be created

DrawingName: Drawing name of the drawing to be created

DocumentTypeValue: Optional argument, select list value for document type, default value is 631 stands for P&Ids

DocumentCategoryValue: Optional argument, select list value for document category, default value is 6 stands for Piping Documents

BVisible: Optional argument, Boolean value as True or False for the visibility of drawing when opening, default as True

2.2. *PIDAutomation.Application.Drawings.OpenDrawing*

Function OpenDrawing(DrawingName As String, [Visible = True]) As Drawing

DrawingName: Pure name of drawing

Visible: Optional argument, Boolean value as True or False for the visibility of drawing when opening, default as True

2.3. *PIDAutomation.Drawing.CloseDrawing*

Sub CloseDrawing(SaveFile As Boolean)

SaveFile: Boolean value as True or False to save drawing when closing drawing.

3. LABS

Lab 85: Open and close an Existing Drawing

Lab 86: Create, Open and Close a New Drawing

Lab 87: Comprehensive Automation Lab

Optional Lab 9: Automatically Create New Drawings

CHAPTER 10: CALCULATION/VALIDATION USING ACTIVE-X SERVER COMPONENTS

1. OBJECTIVES

In this chapter, you will learn

◊ the special issues relevant to writing Calculation/Validation routines

2. ILMFOREIGNCALC INTERFACE

The ILMForeignCalc Interface supports four functions that are called by SmartPlant P&ID at specific events. This interface is supplied in the LMForeignCalc library. `IMPLEMENTS ILMFOREIGNCALC` statement incorporates this interface into the Active-X component. The functions are described below:

2.1. *DoCalculate*

◊ Function DoCalculate(DataSource As LMADataSource, Items As LMAItems, PropertyName As String, Value) As Boolean

◊ It is activated only if the ProgID of the class has been entered into the Calculation ID field of the property in the DataDictionary Manager.

◊ SPPID triggers a call to the function when the user clicks on the ellipsis button on the Property Grid next to the relevant property.

2.2. *DoValidateItem*

◊ Function DoValidateItem(DataSource As LMADataSource, Items As LMAItems, Context As ENUM_LMAValidateContext) As Boolean

◊ It is activated only if the ProgID of the class has been entered into the Validation Program field of the Database Item Type in the DataDictionary Manager.

- ◊ SPPID triggers a call to the function when the user place a new item, copy and paste and existing item, and delete and item to/from drawing
- ◊ SPPID triggers a call to the function when the user de-selects from the item after changing some property associated with the item.
- ◊ The various contexts for the triggering are listed by the enumerated variable LMForeignCalc.ENUM_LMAValidateContext, which has values of LMAValidateCreate, LMAValidateCopy, LMAValidateDelete, LMAValidateModify and LMAValidateFiltered**.
- ◊ The call is made to this function generally after every relationship has been created and the rules have been executed.

2.3. *DoValidateProperty*

- ◊ Function DoValidateProperty(DataSource As LMADataSource, Items As LMAItems, PropertyName As String, Value) As Boolean
- ◊ It is activated only if the ProgID has been entered into Validation ID field of the property in the DataDictionary Manager.
- ◊ SPPID triggers a call to the function when the user de-selects from the relevant property after having modified it.
- ◊ The call is made to this method before SPPID accepts the user's entry into the memory.

2.4. *DoValidatePropertyNoUI*

- ◊ Sub DoValidatePropertyNoUI(DataSource As LMADataSource, Items As LMAItems, PropertyName As String, Value)
- ◊ It is activated only if the ProgID has been entered into Validation ID field of the property in the DataDictionary Manager.
- ◊ SPPID triggers a call to this function when a rule modifies the property, such as at placement time.

3. OBJECTS PASSED FROM THE MODELER

3.1. *LMADataSource*

This LMADataSource is initialized to the Project associated with the Drawing. A ProjectNumber property will not return anything and it cannot be set to any other Project than the one associated with the Drawing.

3.2. *LMAItems*

The select set of items is passed to the Calculation/Validation routines as a collection of LMAItems.

3.3. *PropertyName*

This is the name as listed in the ItemAttributions table of the DataDictionary.

3.4. *Property Value*

This is a Variant and can contain a NULL value.

4. SPECIAL ISSUES

4.1. *Updating the Property Grid*

In order to update the value seen through the Property Grid, it is necessary to set the value of the 'Value' variable and to return a TRUE for the Boolean return value of the ILMForeignCalc functions. If the return value is FALSE (default), then it is assumed that the operation was not a success and the old values are retained in the property grid.

4.2. *Commit command*

A Commit command must be issued to the object in question if the changes to its properties other than the property being validated are to be made persistent.

4.3. *Automatically Fire Up Validation*

When user uses automation program to update a property, if the property has a validation ProgID in its validation ProgID field, the validation will be automatically called up. For example, when user update the Vessel's prefix to "P" and commit the change, then validation will be fired up to generate a new item tag for Vessel if the property TagPrefix has a validation ProgID in its validation ProgID field.

5. LABS

Lab 88: Create a Calculation Program.

Lab 89: Create a ValidateProperty Program.

Lab 90: Create a ValidateItem Program.

Optional Lab 10: Calculation Validation (1).

Optional Lab 11: Calculation Validation (2).

Optional Lab 12: Property Validation (1).

Optional Lab 13: Property Validation (2).

Optional Lab 14: Item Validation (1).

Optional Lab 15: Item Validation (2).

Optional Lab 16: Item Validation (3).

CHAPTER 11: USING LMAUTOMATIONUTIL TO GET FROM/TO INFORMATION

1. OBJECTIVES

In this chapter, you will learn

◊ the special issues relevant to use LMAutomationUtil to get from/to information for Piperun.

2. IMPORTANT METHODS

2.1. *RunsNavigation*

Sub RunsNavigation(objRun As LMPipeRun, FromCollection As Collection, ToCollection As Collection, [InDeterminateCol as Collection)

objRun: LMPipeRun of the Piperun object to report of its from/to

FromCollection: Collection of items that are returned as FROM Items

ToCollection: Collection of items that are returned as TO Items

InDeterminateCol: Collection of items that can not be determined ad FROM or TO items

2.2. *RunsNavigationAll*

Sub RunsNavigationAll(objDS As LMADataSource, objRuns As String, BuildInDeterminates As Boolean, OutDataCollection As Collection)

objDS: Current LMADataSource

objRuns: A string value with all SP_IDs of PipeRuns in format as 'SP_ID','SP_ID',...

BuildInDeterminates: A Boolean value, if it is TRUE, a FROM/TO item, such as PipeRun, will be reported even no flow direction is defined on that PipeRun

OutDataCollection: Collection of Collection that includes the FROM/TO items

Sub RunsNavigationAll(objDS As LMADataSource, objRuns As String, BuildInDeterminates As Boolean, OutDataCollection As Collection)

Each item, which is object as LMAutomationUtil.ToFromItem, in OutDataCollection is a collection of all from/to item related to a Piperun.

LMAutomationUtil.ToFromItem has following functions/methods/properties:

BucketType: data type as LONG, 1 means From, 2 means To, 3 means BiDir

Connected_FlowDirection: data type as LONG, code list index value of FlowDirection of the piperun that is reported as from/to item

Connected_ID: data type as string, from/to item's SP_ID

Connected_ItemTag: data type as string, from/to item's ItemTag

Connected_OPC_DrawingNumber: data type as string, if From/to item is OPC, and this OPC has pair OPC on a drawing, then the Drawing Number of that drawing where its pair is sitting on.

Equipment_ItemTag: data type as string, if from/to item is NOZZLE, then the itemtag of the Equipment the Nozzle is on.

Equipment_SP_ID: data type as string, if from/to item is NOZZLE, then the SP_ID of the Equipment the Nozzle is on.

Init: internal method

ItemType: data type as string, the ItemType of the from/to item

PipeRun_FlowDirection: data type as LONG, code list index value of FlowDirection of the piperun that is running from/to report

PipeRun_ID: data type as string, SP_ID of FlowDirection of the piperun that is running from/to report

3. LABS

Optional Lab 21: Improvement of from/to macro

SMARTPLANT P&ID

V4.3

AUTOMATION COURSE

LABS

LABS

GENERAL INSTRUCTIONS FOR LABS:

1. You will need to reference following dlls for your lab programs, which are located at "...\ Program Files\SmartPlant\P&ID Workstation\Program".

(1) Intergraph SmartPlant P&ID Logical Model Automation – LLAMA.DLL
(2) Intergraph SmartPlant P&ID Placement Automation – Plaice.DLL
(3) Intergraph SmartPlant P&ID Automation – PIDAuto.DLL
(4) Intergraph SmartPlant PID Foreign Calculation Adapter - LMForeignCalc.DLL

2. Most labs expect a Boolean variable to be defined to indicate user's choice of using PIDDatasource or New LMADatasource

```
Private blnUsePIDDatasource As Boolean
```

3. Some constants need to be defined to hold SP_ID of some items, of course you need to place these items first. I provide an assembly for you to place into a drawing at the beginning of this course. Examples are:

```
Private Const CONST_SPID_ModelItem As String = "C76EF274525A4345A6ACE1D179362899"

Private Const CONST_SPID_ItemNote As String = "9A3B02C271754A8BB46DC4D02F9F0954"

Private Const CONST_SPID_OPC As String = "A8EC5233227A4F3AB480E9AB39205BCC"

Private Const CONST_SPID_Vessel As String = "C76EF274525A4345A6ACE1D179362899"

Private Const CONST_SPID_PipeRun As String = "8B283FA8472F4E3BABB6AF573DF161F4"

Private Const CONST_SPID_PipingComp As String =
"59D6251324574734B9883C8E89E57B4E"

Private Const CONST_SPID_OfflineInstrument As String =
"7EAB72658BA04FD8BD67CFEB4D96DD37"

Private Const CONST_SPID_InlineInstrument As String =
"BC21A415E803496EBDA87129F5F5F540"

Private Const CONST_SPID_LabelPersist As String =
"B9E88D821E8145269E5B398B858555A8"
```

1. INITIALIZE LMADATASOURCE

a) Purpose

To initialize LMADataSource with different methods and access some properties of it.

b) Problem Statement

Write a standalone application to initialize the LMADataSource with New LMADatasource and PIDDatasource, then access some properties of it, such as ProjectNumber, SiteNote, etc.

c) Solution

1. Using Set new LMADataource or PIDDataSource to initialize LMADataSource.
2. Use Debug.Print method to print out the required properties.

◊ **Example Code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Debug.Print datasource.ProjectNumber
Debug.Print datasource.SiteNode
Debug.Print datasource.IsSatellite
Debug.Print datasource.GetSystemEditingToolbarSetting

Set datasource = Nothing
```

2. IDENTIFY AN ITEM IN THE DATABASE USING SP_ID AND READ ITS PROPERTIES

a) Purpose

To access a vessel using SP_ID values and read its properties

b) Problem Statement

Place a vessel. Write a standalone application to retrieve the following properties of the vessel: SP_ID, EquipmentSubClass, EquipmentType, aabbcc_code, Class, Item TypeName, volumeRating, and volumeRating in SI units.

c) Solution

2. Dim a LMVessel object and get the object using LMADataSource.GetVessle method.
3. Use Debug.Print method to print out the required properties.

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objVessel As LMVessel
Set objVessel = datasource.GetVessel(CONST_SPID_Vessel)   'get objVessel by id

'print out some objVessel's properties
Debug.Print "objVessel ID = " & objVessel.ID
Debug.Print "Equipment Subclass = " & objVessel.Attributes("EquipmentSubclass").Value
Debug.Print "Equipment Type = " & objVessel.Attributes("EquipmentType").Value
Debug.Print "aabbcc code = " & objVessel.Attributes("aabbcc_code").Value
Debug.Print "Class = " & objVessel.Attributes("Class").Value
Debug.Print "Item TypeName = " & objVessel.Attributes("ItemTypeName").Value
Debug.Print "Volume Rating =" & objVessel.Attributes("VolumeRating").Value
Debug.Print "Volume Rating in SI units = " & objVessel.Attributes("VolumeRating").SIValue

Set datasource = Nothing
Set objVessel = Nothing
```

3. IDENTIFY AN ITEM IN THE DATABASE AND MODIFY ITS PROPERTIES

a) Purpose

To modify its properties of items in the database

b) Problem Statement

Place a vessel. Write a standalone application to modify the following property of the vessel: Name

c) Solution

1. Dim a LMVessel object and get the object using LMADataSource.GetVessle method.
2. Change the value of required properties
3. Use LMVessel.Commit to commit the change to database

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

datasource.BeginTransaction

Dim objVessel As LMVessel
Set objVessel = datasource.GetVessel(CONST_SPID_Vessel)   'get vessel by id
objVessel.Attributes("Name").Value = "Vessel 7"           'assign value to vessel name
objVessel.Attributes("DesignBy").Value = "By B"
objVessel.Commit

datasource.CommitTransaction

Set objVessel = Nothing
Set datasource = Nothing
```

4. PROPAGATION

a) Purpose

To set propagation to True or False from automation program

b) Problem Statement

Place a PipeRun, then place couple branch PipeRuns to this piperun. Write a standalone application to modify the property "SupplyBy" of the first PipeRun with Propagation set to True and modify the property "CleaningReqmts" with Propagation set to False.

c) Solution

Example code

```
Dim datasource As LMADataSource
Dim objPipeRun As LMPipeRun

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If


datasource.BeginTransaction

Set objPipeRun = datasource.GetPipeRun(CONST_SPID_PipeRun)   'get PipeRun by id

datasource.PropagateChanges = True
objPipeRun.Attributes("SupplyBy").Value = "By D"          'assign value to PipeRun Supply By
objPipeRun.Commit

datasource.PropagateChanges = False
objPipeRun.Attributes("CleaningReqmts").Value = "CC1"          'assign value to PipeRun
Supply By
objPipeRun.Commit

datasource.CommitTransaction

Set objPipeRun = Nothing
Set datasource = Nothing
```

5. ROLLBACK

a) Purpose

To rollback a transaction by automation program

b) Problem Statement

Place a Piperun. Write a standalone application to get LMPiperun, then change the property "Name" of the piperun and CommitTransaction, then change the property "Name" again, but this time RollbackTransaction, check which value is commited.

c) Solution

1. Dim a LMVessel object and get the object using LMADataSource.GetVessle method.
2. Dim a LMEquipment object and get the object using LMADataSource.GetEquipment method.
3. Use LMVessel.AsLMAItem and LMEquipment.AsLMAEquipment method to transfer to LMAItem.

◊ **Example code**

```
Dim datasource As LMADataSource
Dim objPipeRun As LMPipeRun

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

datasource.BeginTransaction

Set objPipeRun = datasource.GetPipeRun(CONST_SPID_PipeRun)   'get PipeRun by id
objPipeRun.Attributes("Name").Value = "TEST1"          'assign value to PipeRun name
objPipeRun.Commit
datasource.CommitTransaction

datasource.BeginTransaction
objPipeRun.Attributes("Name").Value = "TEST2"          'assign value to PipeRun name
objPipeRun.Commit
datasource.RollbackTransaction

Set objPipeRun = Nothing
Set datasource = Nothing
```

6. INIT OBJECTS READ ONLY

a) Purpose

To use property of LMADatasource: InitObjectsReadonly

b) Problem Statement

Place a Piperun. Write a standalone application to get LMPiperun, then set the InitObjectsReadonly to True, and check if the property "Name" can be changed with drawing close and New LMADatasource is used.

c) Solution

Example code

```
Dim datasource As LMADataSource
Dim objPipeRun As LMPipeRun

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

datasource.InitObjectsReadonly = True

datasource.BeginTransaction

Set objPipeRun = datasource.GetPipeRun(CONST_SPID_PipeRun)    'get PipeRun by id
objPipeRun.Attributes("Name").Value = "TEST1"              'assign value to PipeRun name
objPipeRun.Commit

datasource.CommitTransaction

Set objPipeRun = Nothing
Set datasource = Nothing
```

7. GET LMAITEM FROM LMVESSEL

a) Purpose

To get LMAItem from LMAVessel and LMAEquipment

b) Problem Statement

Place a vessel. Write a standalone application to get LMVessel and LMEquipment, then transfer them to LMAItem, and compare the difference

c) Solution

3. Dim a LMVessel object and get the object using LMADataSource.GetVessle method.
4. Dim a LMEquipment object and get the object using LMADataSource.GetEquipment method.
3. Use LMVessel.AsLMAItem and LMEquipment.AsLMAEquipment method to transfer to LMAItem.

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objVessel As LMVessel
Dim objItem1 As LMAItem
Dim objEquipment As LMEquipment
Dim objItem2 As LMAItem

Set objVessel = datasource.GetVessel(CONST_SPID_Vessel)   'get vessel by id
Set objItem1 = objVessel.AsLMAItem
Set objEquipment = datasource.GetEquipment(CONST_SPID_Vessel) 'get equipment by id
Set objItem2 = objEquipment.AsLMAItem

'print out some properties of objItem1 and objItem2
Debug.Print "ItemType = " & objItem1.ItemType
Debug.Print "VolumeRating = " & objItem1.Attributes("VolumeRating").Value
Debug.Print "ItemType = " & objItem2.ItemType
Debug.Print "VolumeRating = " & objItem2.Attributes("VolumeRating").Value

Set datasource = Nothing
Set objVessel = Nothing
Set objItem1 = Nothing
Set objEquipment = Nothing
Set objItem2 = Nothing
```

8. ACCESS ALL ITEMTYPES

a) Purpose

To access all ItemTypes within a Plant

b) Problem Statement

Access to an LMADatasource, then print out all Item Types within that LMADatasource. There are total 41 Item Types in V4, which includes:

AreaBreak
Drawing
DrawingProject
DrawingVersion
EquipComponent
Equipment
EquipmentOther
Exchanger
GlobalDrawing
History
InstrLoop
Instrument
ItemNote
Label
LabelPersist
Mechanical
ModelItem
ModelItemClaim
ModelItemClaimOffline
ModelItemClaimRep
ModelItemLookup
Note
Nozzle
OPC
Package
PipeRun
Pipeline
PipingComp
PipingPoint
PlantItem
PlantItemGroup
PlantItemGroupOther
Representation
RepresentationLookup
SafetyClass
SignalPoint
SignalRun
System
Task
TaskItemProperty
Vessel

c) Solution

1. Get object LMADatasource

2. Loop through LMADatasource.ItemTypes

◊ **Example code**

```
Dim datasource As LMADataSource
Dim i As Integer

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If
Debug.Print "Total ItemTypes: " & datasource.TypeNames.Count
For i = 1 To datasource.TypeNames.Count
   Debug.Print datasource.TypeNames.item(i)
Next

Set datasource = Nothing
```

9. ACCESS LMAATTRIBUTES COLLECTION

a) Purpose

To access LMAAttributes collection of LLAMA object.

b) Problem Statement

Place a Vessel and get the LMVessel object, then loop through its Attributes collection.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objVessel As LMVessel

Set objVessel = datasource.GetVessel(CONST_SPID_Vessel)   'get vessel by id

Dim objAttr As LMAAttribute
Debug.Print "Total attributes for Vessel: " & objVessel.Attributes.Count
For Each objAttr In objVessel.Attributes
   Debug.Print "Attribute Name=" & objAttr.name & Space(50 - Len(objAttr.name)) & "
Value=" & objAttr.Value
Next

Debug.Print objVessel.Attributes.item("ProcessAlternateDesign.Max.Pressure").Value
Debug.Print "Total attributes for Vessel: " & objVessel.Attributes.Count
For Each objAttr In objVessel.Attributes
   Debug.Print "Attribute Name=" & objAttr.name & Space(50 - Len(objAttr.name)) & "
Value=" & objAttr.Value
Next

Set datasource = Nothing
Set objVessel = Nothing
Set objAttr = Nothing
```

10. ACCESS ITEMATTRIBUTIONS IN DETAILS

a) Purpose

To access ItemAttributions of different items in details

b) Problem Statement

Place all kinds of SmartPlant P&ID items, such as Vessel, Mechanical, Heat Exchanger, then print their ItemAttributions information in details in format of Excel, which includes attribution format, index if codelist, calculation ProgID and validation ProgID

c) Solution

1. Get Items
2. Needs access LMAAttribute.ISPAttribute
3. Print result in Excel

◊ **Example code**

```
Dim datasource As LMADataSource
Dim i As Integer
Dim objAttr As LMAAttribute
Dim vValue As Variant
Dim objVessel As LMVessel

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objExcel As Excel.Application

Set objExcel = CreateObject("Excel.Application")
objExcel.Visible = True

Dim xlWorkbook As Excel.Workbook
Set xlWorkbook = objExcel.Workbooks.Add

Dim xlWorksheet As Excel.Worksheet
Set xlWorksheet = xlWorkbook.Worksheets("SHEET1")

Dim Row As Long
Dim CodeListCount As Long

Row = 1

xlWorksheet.Cells(Row, 1) = "ItemType"

xlWorksheet.Cells(Row, 2) = "Attribute Name"

xlWorksheet.Cells(Row, 3) = "Format"

xlWorksheet.Cells(Row, 4) = "IsCodeList"

xlWorksheet.Cells(Row, 5) = "CodeList Index"
```

```
    xlWorksheet.Cells(Row, 6) = "Calculation ProgID"

    xlWorksheet.Cells(Row, 7) = "Validation ProgID"
    Row = Row + 1

    Set objVessel = datasource.GetVessel(CONST_SPID_Vessel)
    xlWorksheet.Cells(Row, 1) = "Total attributions for Vessel: " & objVessel.Attributes.Count
    Row = Row + 1
    For Each objAttr In objVessel.Attributes
      xlWorksheet.Cells(Row, 1) = objVessel.AsLMAItem.ItemType
      xlWorksheet.Cells(Row, 2) = objAttr.name
      xlWorksheet.Cells(Row, 3) = objAttr.ISPAttribute.Attribution.Format
On Error Resume Next
      CodeListCount = 0
      CodeListCount = objAttr.ISPAttribute.Attribution.ISPEnumAtts.Count
On Error GoTo 0
      If CodeListCount > 0 Then
        xlWorksheet.Cells(Row, 4) = "True"
      Else
        xlWorksheet.Cells(Row, 4) = "False"
      End If
      xlWorksheet.Cells(Row, 5) = objAttr.Index
      xlWorksheet.Cells(Row, 6) = objAttr.ISPAttribute.Attribution.CalculationProgID
      xlWorksheet.Cells(Row, 7) = objAttr.ISPAttribute.Attribution.ValidationProgID
      Row = Row + 1
    Next

    Row = Row + 1
On Error Resume Next
    vValue = objVessel.Attributes("ProcessDesign.Max.Pressure")
On Error GoTo 0
    xlWorksheet.Cells(Row, 1) = "Total attributions for Vessel: " & objVessel.Attributes.Count
    Row = Row + 1
    For Each objAttr In objVessel.Attributes
      xlWorksheet.Cells(Row, 1) = objVessel.AsLMAItem.ItemType
      xlWorksheet.Cells(Row, 2) = objAttr.name
      xlWorksheet.Cells(Row, 3) = objAttr.ISPAttribute.Attribution.Format
On Error Resume Next
      CodeListCount = 0
      CodeListCount = objAttr.ISPAttribute.Attribution.ISPEnumAtts.Count
On Error GoTo 0
      If CodeListCount > 0 Then
        xlWorksheet.Cells(Row, 4) = "True"
      Else
        xlWorksheet.Cells(Row, 4) = "False"
      End If
      xlWorksheet.Cells(Row, 5) = objAttr.Index
      xlWorksheet.Cells(Row, 6) = objAttr.ISPAttribute.Attribution.CalculationProgID
      xlWorksheet.Cells(Row, 7) = objAttr.ISPAttribute.Attribution.ValidationProgID
      Row = Row + 1
    Next

    Dim strFileName As String
    strFileName = Environ("TEMP") & "\ItemAttributions.xls"
    objExcel.Workbooks(1).SaveAs (strFileName)
```

```
xlWorkbook.Close True
objExcel.Quit

MsgBox "Done"

Set datasource = Nothing
Set objVessel = Nothing
Set xlWorksheet = Nothing
Set xlWorkbook = Nothing
Set objExcel = Nothing
```

11. CHANGE SITE AND PLANT

a) Purpose

To change active site and active plant within LLAMA program.

b) Problem Statement

Place a Vessel into active drawing, then close the drawing. Then switch the smartplant to another site and another place.

c) Solution

Change the site and plant within your program to get access to that Vessel.

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Debug.Print datasource.SiteNode
Debug.Print datasource.ProjectNumber

Dim objVessel As LMVessel
Set objVessel = datasource.GetVessel(CONST_SPID_Vessel)
Debug.Print objVessel.Attributes("ItemTag").Value

datasource.SiteNode = "C:\HostSite\SmartPlantV4.ini"
datasource.ProjectNumber = "Lin_Plant3"

Set objVessel = datasource.GetVessel("C76EF274525A4345A6ACE1D179362899")
Debug.Print objVessel.Attributes("ItemTag").Value

Set datasource = Nothing
Set objVessel = Nothing
```

12. COLLECT ITEMS FROM THE DATABASE USING FILTERS

a) Purpose

To access objects created through SPPID using filters

b) Problem Statement

Place a piperun and give it a TagSuffix value. Retrieve the piperun by filtering on the TagSuffix value = "P" and populate the Name property with value "P-Run"

c) Solution

1. Dim LMAFilter and LMACriterion
2. Add LMACriterion to LMAFilter
3. Call LMPipeRuns.Collect method by using the LMAFilter

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objFilter As LMAFilter
Dim criterion As LMACriterion
 Set criterion = New LMACriterion
Set objFilter = New LMAFilter

criterion.SourceAttributeName = "TagSuffix"
criterion.ValueAttribute = "P"
criterion.Operator = "="
objFilter.ItemType = "PipeRun"
objFilter.Criteria.Add criterion

Dim piperun As LMPipeRun
Dim piperuns As LMPipeRuns
Set piperuns = New LMPipeRuns
piperuns.Collect datasource, Filter:=objFilter

Debug.Print "Number of Piperuns retrieved = " & piperuns.Count
datasource.BeginTransaction
For Each piperun In piperuns
   Debug.Print piperun.Attributes("TagSuffix").Value

   Debug.Print "Piperun ID = " & piperun.ID
   piperun.Attributes("Name").Value = "P-Run"
   piperun.Commit
Next
datasource.CommitTransaction
Set datasource = Nothing
Set objFilter = Nothing
Set criterion = Nothing
Set piperun = Nothing
Set piperuns = Nothing
```

13. COLLECT ITEMS FROM THE DATABASE USING FILTERS WITH MULTIPLE CRITERIA

a) Purpose

To access objects created through SPPID using filters with multiple criteria

b) Problem Statement

Place three piperuns and set OperFluidCode="KD" for one piperun, TagSuffix = "PT" for another pipe run and Name="V" for another pipe run. Retrieve the three piperuns by filtering using Multiple Criteria.

c) Solution

1. Dim LMAFilter and LMACriterion
2. Add multiple LMACriterion to LMAFilter
3. Call LMPipeRuns.Collect method by using the LMAFilter

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objFilter As LMAFilter
Set objFilter = New LMAFilter

objFilter.Criteria.AddNew ("FirstOne")
objFilter.Criteria.item("FirstOne").SourceAttributeName = "ItemTag"
objFilter.Criteria.item("FirstOne").ValueAttribute = "%K%"
objFilter.Criteria.item("FirstOne").Operator = "like"
objFilter.ItemType = "PipeRun"

objFilter.Criteria.AddNew ("SecondOne")
objFilter.Criteria.item("SecondOne").SourceAttributeName = "TagSuffix"
objFilter.Criteria.item("SecondOne").ValueAttribute = "P_"
objFilter.Criteria.item("SecondOne").Operator = "like"
objFilter.Criteria.item("SecondOne").Conjunctive = False

objFilter.Criteria.AddNew ("ThirdOne")
objFilter.Criteria.item("ThirdOne").SourceAttributeName = "Name"
objFilter.Criteria.item("ThirdOne").ValueAttribute = Null
objFilter.Criteria.item("ThirdOne").Operator = "!="
objFilter.Criteria.item("ThirdOne").Conjunctive = False

Dim piperun As LMPipeRun
Dim piperuns As LMPipeRuns
Set piperuns = New LMPipeRuns
piperuns.Collect datasource, Filter:=objFilter

Debug.Print "Number of piperuns filtered = " & piperuns.Count
For Each piperun In piperuns
```

```
    Debug.Print "ID = " & piperun.ID
    Debug.Print "ItemTag = " & piperun.Attributes("ItemTag").Value
    Debug.Print "TagSuffix = " & piperun.Attributes("TagSuffix").Value
    Debug.Print "Name = " & piperun.Attributes("Name").Value
Next

Set datasource = Nothing
Set objFilter = Nothing
Set piperun = Nothing
Set piperuns = Nothing
```

14. USING FILTERS WITH CRITERIA ON SELECT LIST DATA

a) Purpose

To access objects created through SPPID using filters with multiple criteria

b) Problem Statement

Place two piperuns and set NominalDiameter=2" for the piperuns. Then delete one piperun from model. Retrieve the active piperun by filtering using Criteria on ItemStatus and NominalDiameter.

c) Solution

Need to find the index for ItemStatus="Active" and NominalDiameter=2".

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objFilter As LMAFilter
Set objFilter = New LMAFilter

objFilter.Criteria.AddNew ("FirstOne")
objFilter.Criteria.item("FirstOne").SourceAttributeName = "ItemStatus"
objFilter.Criteria.item("FirstOne").ValueAttribute = "1"
objFilter.Criteria.item("FirstOne").Operator = "="
objFilter.ItemType = "PipeRun"

objFilter.Criteria.AddNew ("SecondOne")
objFilter.Criteria.item("SecondOne").SourceAttributeName = "NominalDiameter"
objFilter.Criteria.item("SecondOne").ValueAttribute = "5064" '2"
objFilter.Criteria.item("SecondOne").Operator = "="
objFilter.Criteria.item("SecondOne").Conjunctive = True

Dim piperun As LMPipeRun
Dim piperuns As LMPipeRuns
Set piperuns = New LMPipeRuns
piperuns.Collect datasource, Filter:=objFilter

Debug.Print "Number of piperuns filtered = " & piperuns.Count
For Each piperun In piperuns
   Debug.Print "ID = " & piperun.ID
   Debug.Print "ItemStatus = " & piperun.Attributes("ItemStatus").Value
   Debug.Print "NominalDiameter = " & piperun.Attributes("NominalDiameter").Value
Next

Set datasource = Nothing
Set objFilter = Nothing
Set piperun = Nothing
Set piperuns = Nothing
```

15. USING COMPOUND FILTER

a) Purpose

To access objects created through SPPID using compound filter

b) Problem Statement

Place six piperuns and set NominalDiameter=1", 2", and 3" for the piperuns. Then, delete three piperuns from model. Retrieve the piperuns with ItemStatus="Active" and NominalDiameter equals 1" or 2" by using compound filter.

c) Solution

Comound allows conjunctive as both "And" and "Or".

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objFilter As LMAFilter
Dim objChildFilter1 As LMAFilter
Dim objChildFilter2 As LMAFilter

Set objFilter = New LMAFilter
Set objChildFilter1 = New LMAFilter
Set objChildFilter2 = New LMAFilter

objChildFilter1.ItemType = "PipeRun"
objChildFilter1.name = "Filter 1"
objChildFilter1.Criteria.AddNew ("FirstOne")
objChildFilter1.Criteria.item("FirstOne").SourceAttributeName = "ItemStatus"
objChildFilter1.Criteria.item("FirstOne").ValueAttribute = "1"
objChildFilter1.Criteria.item("FirstOne").Operator = "="

objChildFilter2.ItemType = "PipeRun"
objChildFilter2.name = "Filter 2"
objChildFilter2.Criteria.AddNew ("FirstOne")
objChildFilter2.Criteria.item("FirstOne").SourceAttributeName = "NominalDiameter"
objChildFilter2.Criteria.item("FirstOne").ValueAttribute = "5032" '1"
objChildFilter2.Criteria.item("FirstOne").Operator = "="

objChildFilter2.Criteria.AddNew ("SecondOne")
objChildFilter2.Criteria.item("SecondOne").SourceAttributeName = "NominalDiameter"
objChildFilter2.Criteria.item("SecondOne").ValueAttribute = 5064 '2"
objChildFilter2.Criteria.item("SecondOne").Operator = "="
objChildFilter2.Criteria.item("SecondOne").Conjunctive = False

objFilter.ItemType = "PipeRun"
objFilter.FilterType = 1 '1 for compound filter, 0 for simple filter
objFilter.ChildLMAFilters.Add objChildFilter1
objFilter.ChildLMAFilters.Add objChildFilter2
```

```
objFilter.Conjunctive = True

Dim piperun As LMPipeRun
Dim piperuns As LMPipeRuns
Set piperuns = New LMPipeRuns
piperuns.Collect datasource, Filter:=objFilter

Debug.Print "Number of piperuns filtered = " & piperuns.Count
For Each piperun In piperuns
   Debug.Print "ItemStatus = " & piperun.Attributes("ItemStatus").Value
   Debug.Print "NominalDiameter = " & piperun.Attributes("NominalDiameter").Value
Next

Set datasource = Nothing
Set objFilter = Nothing
Set objChildFilter1 = Nothing
Set objChildFilter2 = Nothing
Set piperun = Nothing
Set piperuns = Nothing
```

16. COLLECT FILTERS FROM DATASOURCE

a) Purpose

To collect all filters in SPPID from datasource

b) Problem Statement

Write a standalone application to retrieve all filters in SPPID from datasource. Display the Item Type and the first Criterion (if one exists) in the filter for those of ItemType = "Instrument".

c) Solution

1. Dim LMAFilter
2. Call LMADataSource.Filters method to get all LMAFilters in database
3. Use For ... Next to loop through the LMAFilters and print out required properties

◊ **Example code**

```
Dim datasource As LMADataSource
If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If
Dim objFiltersCollection As Collection
Set objFiltersCollection = datasource.Filters

Debug.Print "Number of filters = " & objFiltersCollection.Count
Dim objFilter As LMAFilter
For Each objFilter In datasource.Filters 'objFiltersCollection
   If objFilter.ItemType = "Instrument" Then
      If Not objFilter.Criteria Is Nothing Then
         If objFilter.Criteria.Count >= 1 Then
            Debug.Print "Filter item type = " & objFilter.ItemType & Space(20 -
Len(objFilter.ItemType)) _
            & "Filter name = " & objFilter.name & Space(50 - Len(objFilter.name)) _
            & objFilter.Criteria.item(1).SourceAttributeName & Space(30 -
Len(objFilter.Criteria.item(1).SourceAttributeName)) _
            & objFilter.Criteria.item(1).Operator & Space(5) _
            & objFilter.Criteria.item(1).ValueAttribute & Space(40 -
Len(objFilter.Criteria.item(1).ValueAttribute))
         End If
      End If
   End If
Next

'use the pre-defined filter
Set objFilter = datasource.Filters.item("Active Equipment")
Dim objEquipments As LMEquipments
Set objEquipments = New LMEquipments
objEquipments.Collect datasource, Filter:=objFilter
Debug.Print objEquipments.Count

Set datasource = Nothing
Set objFilter = Nothing
Set objFiltersCollection = Nothing
Set objEquipments = Nothing
```

17. ACCESS SELECTLIST DATA

a) Purpose

To get familiar with LMAEnumAttList and LMAEnumratedAttributes objects in LLAMA.

b) Problem Statement

Write a standalone application to retrieve all Select List Data in SPPID from datasource. Display properties, such as ListName, DependName, DependID. Then loop through all Select List Value of each Select List Data, display properties, such as Name and Index.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If


Dim objEnum As LMAEnumAttList
Dim objEnums As LMAEnumAttLists
Dim objEnumAttr As LMAEnumeratedAttribute
Dim objEnumAttrs As LMAEnumeratedAttributes

Set objEnums = datasource.CodeLists

Debug.Print "Total Select List Data found: " & objEnums.Count

For Each objEnum In objEnums
   Debug.Print ""
   Debug.Print "Select List Name = " & objEnum.ListName & Space(40 -
Len(objEnum.ListName)) _
   & "DependName = " & objEnum.DependName & Space(30 - Len(objEnum.DependName)) _
   & "DependID = " & objEnum.DependID
   Set objEnumAttrs = objEnum.EnumeratedAttributes
   For Each objEnumAttr In objEnumAttrs
      Debug.Print "Name = " & objEnumAttr.name & Space(65 - Len(objEnumAttr.name)) _
      & "Index = " & objEnumAttr.Index
   Next
Next

Set datasource = Nothing
Set objEnum = Nothing
Set objEnums = Nothing
Set objEnumAttr = Nothing
Set objEnumAttrs = Nothing
```

18. READ HISTORY PROPERTY OF MODELITEM

a) Purpose

To read the history data belongs to a modelitem.

b) Problem Statement

Place a Vessel. Write a standalone application to read the history data belongs to this Vessel.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objModelItem As LMModelItem
Set objModelItem = datasource.GetModelItem(CONST_SPID_ModelItem)

Dim objHistory As LMHistory
Dim objHistories As LMHistories
Dim objAttribute As LMAAttribute
Set objHistories = objModelItem.Histories

Debug.Print objHistories.Count

For Each objHistory In objHistories
   For Each objAttribute In objHistory.Attributes
      Debug.Print "Name: " & objAttribute.name & Space(20 - Len(objAttribute.name)) & "Value:
" & objAttribute.Value
   Next
Next

Set datasource = Nothing
Set objModelItem = Nothing
Set objHistory = Nothing
Set objHistories = Nothing
Set objAttribute = Nothing
```

19. READ STATUS PROPERTY OF MODELITEM

a) Purpose

To read the Status datas belongs to a modelitem

b) Problem Statement

Place a Vessel with some Status data populated. Write a standalone application to read the status data belongs to this Vessel.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objModelItem As LMModelItem
Set objModelItem = datasource.GetModelItem(CONST_SPID_ModelItem)

Dim objStatus As LMStatus
Dim objStatuses As LMStatuses
Set objStatuses = objModelItem.Statuses

Debug.Print objStatuses.Count

Dim objAttribute As LMAAttribute
For Each objStatus In objStatuses
   For Each objAttribute In objStatus.Attributes
      Debug.Print "Name: " & objAttribute.name & Space(20 - Len(objAttribute.name)) & "Value:
" & objAttribute.Value
   Next
Next

Set datasource = Nothing
Set objModelItem = Nothing
Set objStatus = Nothing
Set objStatuses = Nothing
Set objAttribute = Nothing
```

20. READ CASE PROPERTY OF MODELITEM

a) Purpose

To read Case data of a modelitem

b) Problem Statement

Place a Vessel with some Case data populated. Write a standalone application to read the case data belongs to this Vessel.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objModelItem As LMModelItem
Set objModelItem = datasource.GetModelItem(CONST_SPID_ModelItem)

Dim objCase As LMCase
Dim objCases As LMCases
Set objCases = objModelItem.Cases
Debug.Print objCases.Count

Dim objAttribute As LMAAttribute
Dim objCaseProcess As LMCaseProcess
Dim objCaseControl As LMCaseControl
For Each objCase In objCases
   For Each objAttribute In objCase.Attributes
      Debug.Print "Name: " & objAttribute.name & Space(20 - Len(objAttribute.name)) & "Value:
" & objAttribute.Value
   Next
   Debug.Print objCase.CaseProcesses.Count
   For Each objCaseProcess In objCase.CaseProcesses
      For Each objAttribute In objCaseProcess.Attributes
         Debug.Print "Name: " & objAttribute.name & Space(20 - Len(objAttribute.name)) &
"Value: " & objAttribute.Value
      Next
   Next
   Debug.Print objCase.CaseControls.Count
   For Each objCaseControl In objCase.CaseControls
      For Each objAttribute In objCaseControl.Attributes
         Debug.Print "Name: " & objAttribute.name & Space(20 - Len(objAttribute.name)) &
"Value: " & objAttribute.Value
      Next
   Next
Next

Set datasource = Nothing
Set objModelItem = Nothing
```

```
Set objCase = Nothing
Set objCases = Nothing
Set objAttribute = Nothing
Set objCaseProcess = Nothing
Set objCaseControl = Nothing
```

21. ACCESS ITEMNOTE

a) Purpose

To access an ItemNote.

b) Problem Statement

Place an ItemNote. Write a standalone application to read the properties of this ItemNote.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objItemNote As LMItemNote
Set objItemNote = datasource.GetItemNote(CONST_SPID_ItemNote)

Dim objAttribute As LMAAttribute
For Each objAttribute In objItemNote.Attributes
   Debug.Print "Name: " & objAttribute.name & Space(40 - Len(objAttribute.name)) & "Value: "
& objAttribute.Value
Next

Dim objNote As LMNote
Debug.Print objItemNote.Notes.Count

For Each objNote In objItemNote.Notes
   For Each objAttribute In objNote.Attributes
      Debug.Print "Name: " & objAttribute.name & Space(40 - Len(objAttribute.name)) & "Value:
" & objAttribute.Value
   Next
Next

Set datasource = Nothing
Set objItemNote = Nothing
Set objNote = Nothing
Set objAttribute = Nothing
```

22. ACCESS OPC

a) Purpose

To access an OPC

b) Problem Statement

Place an OPC and its PairOPC in another drawing. Write a standalone application to read the properties of this OPC and its PairOPC.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objOPC As LMOPC
Set objOPC = datasource.GetOPC(CONST_SPID_OPC)

Dim objAttribute As LMAAttribute
For Each objAttribute In objOPC.Attributes
   Debug.Print "Name: " & objAttribute.name & Space(20 - Len(objAttribute.name)) & "Value: "
& objAttribute.Value
Next

Dim objPairOPC As LMOPC
Set objPairOPC = objOPC.pairedWithOPCObject

For Each objAttribute In objPairOPC.Attributes
   Debug.Print "Name: " & objAttribute.name & Space(20 - Len(objAttribute.name)) & "Value: "
& objAttribute.Value
Next

Set datasource = Nothing
Set objOPC = Nothing
Set objPairOPC = Nothing
Set objAttribute = Nothing
```

23. FILTER FOR HISTORIES

a) Purpose

To filter for histories by TimeStamp and ItemType

b) Problem Statement

Set the active plant with some items placed. Write a standalone application to filter Histories.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objFilter As LMAFilter
Set objFilter = New LMAFilter

objFilter.Criteria.AddNew ("FirstOne")
objFilter.Criteria.item("FirstOne").SourceAttributeName = "TimeStamp"
objFilter.Criteria.item("FirstOne").ValueAttribute = "7/19/04 8:00:00 AM"
objFilter.Criteria.item("FirstOne").Operator = ">"
objFilter.ItemType = "History"

objFilter.Criteria.AddNew ("SecondOne")
objFilter.Criteria.item("SecondOne").SourceAttributeName = "ModelItem.ModelItemType"
objFilter.Criteria.item("SecondOne").ValueAttribute = 29 '29 is the index for 'Plant Item'
objFilter.Criteria.item("SecondOne").Operator = "="
objFilter.Criteria.item("SecondOne").Conjunctive = True
Dim objHistories As LMHistories

Set objHistories = New LMHistories
objHistories.Collect datasource, Filter:=objFilter

Debug.Print objHistories.Count

Dim objHistory As LMHistory

For Each objHistory In objHistories
   Debug.Print objHistory.Attributes("TimeStamp").Value
   Debug.Print objHistory.ModelItemObject.Attributes("ModelItemType").Value
Next

Set datasource = Nothing
Set objFilter = Nothing
Set objHistories = Nothing
Set objHistory = Nothing
```

24. CHANGE PROPERTIS AT DIFFERNET OBJECT LEVELS

a) Purpose

To access "Name" property at different object level.

b) Problem Statement

Place a vessel. Write a standalone application to change "Name" property at Equipment object level, and see how it changes the output for Vessel object

c) Solution

1. use LMADataSource.GetVessel and LMADataSource.GetEquipment methods to obtain object Vessel and Equipment with same SP_ID
2.change property "Name" value of Equipment object, then obtain Vessel object again to see how it changes the property "Name" value of Vessel

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If
datasource.BeginTransaction
Dim objVessel As LMVessel
Dim objEquipment As LMEquipment
Set objVessel = datasource.GetVessel(CONST_SPID_Vessel)
Set objEquipment = datasource.GetEquipment(CONST_SPID_Vessel)

Dim objAttr As LMAAttribute
Debug.Print "Total attributes for Vessel: " & objVessel.Attributes.Count
For Each objAttr In objVessel.Attributes
   Debug.Print "Attribute Name : Value     " & objAttr.name & " : " & objAttr.Value
Next

Debug.Print "Total attributes for Equipment: " & objEquipment.Attributes.Count
For Each objAttr In objEquipment.Attributes
   Debug.Print "Attribute Name : Value     " & objAttr.name & " : " & objAttr.Value
Next

Debug.Print objEquipment.name
Debug.Print objVessel.name

objEquipment.Attributes("Name").Value = "Lab-12"
objEquipment.Commit
Set objVessel = datasource.GetVessel(CONST_SPID_Vessel)
Debug.Print objEquipment.name
Debug.Print objVessel.name

datasource.CommitTransaction
Set datasource = Nothing
Set objVessel = Nothing
Set objEquipment = Nothing
Set objAttr = Nothing
```

25. READ CASEPROPERTY OF VESSEL

a) Purpose

To access case properties of Vessel.

b) Problem Statement

Place a vessel. Populate the some case property value of the vessel. Write a standalone application to access the case and caseprocess of the vessel and read properties of the case.

c) Solution

1. Dim LMVessel
2. Vessel is associated several LMCases, if Case Class is Case Process, then this Case can have two CaseProcesses associated with it, depends on Quality, which can be Maximun or Minimum, then a one to one filtered relationship is found for the Vessel and Case property

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If
Dim objVessel As LMVessel
Set objVessel = datasource.GetVessel(CONST_SPID_Vessel)

Debug.Print "Design.Min.Pressure = " &
objVessel.Attributes("ProcessDesign.Min.Pressure").Value

Set datasource = Nothing
Set objVessel = Nothing
```

26. READ FLOW DIRECTION OF PIPERUN

a) Purpose

To obtain Flow Direction of Piperun

b) Problem Statement

Place a Piperun. Write a standalone application to obtain Flow Direction information about the Piperun.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If
Dim objPiperun As LMPipeRun
Set objPiperun = datasource.GetPipeRun(CONST_SPID_PipeRun)

Debug.Print "Flow Direction = " & objPiperun.Attributes("FlowDirection").Value

Set datasource = Nothing
Set objPiperun = Nothing
```

27. ACCESS PIPING POINT

a) Purpose

To access a Piping Point.

b) Problem Statement

Place a Valve. Write a standalone application to access PipingPoint belongs to this Valve.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objPipingComp As LMPipingComp
Set objPipingComp = datasource.GetPipingComp(CONST_SPID_PipingComp)

Dim objAttribute As LMAAttribute
Dim objPipingPoint As LMPipingPoint
Debug.Print objPipingComp.PipingPoints.Count

For Each objPipingPoint In objPipingComp.PipingPoints
   For Each objAttribute In objPipingPoint.Attributes
      Debug.Print "Name: " & objAttribute.name & Space(25 - Len(objAttribute.name)) & "Value:
" & objAttribute.Value
   Next
Next

Set datasource = Nothing
Set objPipingComp = Nothing
Set objPipingPoint = Nothing
Set objAttribute = Nothing
```

28. ACCESS SIGNAL POINT

a) Purpose

To access a Signal Point.

b) Problem Statement

Place an offline Instrument. Write a standalone application to access PipingPoint belongs to this offline Instrument.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objInstrument As LMInstrument
Set objInstrument = datasource.GetInstrument(CONST_SPID_OfflineInstrument)

Dim objAttribute As LMAAttribute
Dim objSignalPoint As LMSignalPoint
Debug.Print objInstrument.SignalPoints.Count

For Each objSignalPoint In objInstrument.SignalPoints
   For Each objAttribute In objSignalPoint.Attributes
      Debug.Print "Name: " & objAttribute.name & Space(20 - Len(objAttribute.name)) & "Value:
" & objAttribute.Value
   Next
Next

Set datasource = Nothing
Set objInstrument = Nothing
Set objSignalPoint = Nothing
Set objAttribute = Nothing
```

29. IMPLIEDITEM

a) Purpose

To navigate the relationship between Implied item and its parent item.

b) Problem Statement

Place a Instrument off-line with implied item. Write a standalone application to obtain any items in the database that are Implied Item

c) Solution

1. Dim LMPlantItem, LMACriterion and LMAFilter
2. Implied item would have property "PartOfType" is equal to "Implied", which has the index number is 2.

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objFilter As LMAFilter
Dim criterion As LMACriterion
Set criterion = New LMACriterion
Set objFilter = New LMAFilter

criterion.SourceAttributeName = "PartOfType"
criterion.ValueAttribute = "2" 'implied item
criterion.Operator = "="
objFilter.ItemType = "PlantItem"
objFilter.Criteria.Add criterion

Dim objPlantItem As LMPlantItem
Dim objPlantItems As LMPlantItems
Set objPlantItems = New LMPlantItems
objPlantItems.Collect datasource, Filter:=objFilter
Debug.Print "Number of Implied Items retrieved = " & objPlantItems.Count

For Each objPlantItem In objPlantItems
   Debug.Print "PartOfType = " & objPlantItem.Attributes("PartOfType").Value
   Debug.Print "Parent Item Type = " &
objPlantItem.PartOfPlantItemObject.Attributes("ItemTypeName").Value
Next

Set datasource = Nothing
Set objFilter = Nothing
Set criterion = Nothing
Set objPlantItem = Nothing
Set objPlantItems = Nothing
```

30. PARTOFPLANTITEM RELATIONSHIPS

a) Purpose

To navigate the relationship between item and its parent item.

b) Problem Statement

Place a Instrument off-line with implied item, two nozzles, two trays, TEAM ends Write a standalone application to find all the items that have parent item in the database

c) Solution

1. Dim LMPlantItem, LMACriterion and LMAFilter
2. Implied item would have property "SP_PartOfID" is not NULL

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If


Dim objFilter As LMAFilter
Dim criterion As LMACriterion
Set criterion = New LMACriterion
Set objFilter = New LMAFilter

criterion.SourceAttributeName = "SP_PartOfID"
criterion.ValueAttribute = Null
criterion.Operator = "!="
objFilter.ItemType = "PlantItem"
objFilter.Criteria.Add criterion

Dim objPlantItem As LMPlantItem
Dim objPlantItems As LMPlantItems
Set objPlantItems = New LMPlantItems
objPlantItems.Collect datasource, Filter:=objFilter
Debug.Print "Number of child items retrieved = " & objPlantItems.Count

For Each objPlantItem In objPlantItems
   Debug.Print "PartOfType = " & objPlantItem.Attributes("PartOfType").Value
   Debug.Print "Parent Item Type = " &
objPlantItem.PartOfPlantItemObject.Attributes("ItemTypeName").Value
Next


Set datasource = Nothing
Set objFilter = Nothing
Set criterion = Nothing
Set objPlantItem = Nothing
Set objPlantItems = Nothing
```

31. ACCESS INSTRUMENT LOOP

c) Purpose

To get familiar with relationship between PlantItemGroup and PlantItem

d) Problem Statement

Use LMAFilter to search for Instrument Loops, then check how many PlantItems are associated with the Instrument Loop. At the end, try to associate an Instrument with this Instrument Loop.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objFilter As LMAFilter
Dim criterion As LMACriterion
Set criterion = New LMACriterion
Set objFilter = New LMAFilter

criterion.SourceAttributeName = "ItemTag"
criterion.ValueAttribute = "L-101L"
criterion.Operator = "="

objFilter.ItemType = "InstrLoop"
objFilter.Criteria.Add criterion

Dim objInstrLoops As LMInstrLoops
Dim objInstrLoop As LMInstrLoop
Set objInstrLoops = New LMInstrLoops
objInstrLoops.Collect datasource, Filter:=objFilter  'get InstrLoop by filter
If objInstrLoops.Count > 0 Then
   Set objInstrLoop = objInstrLoops.Nth(1)
Else

   Set datasource = Nothing
   Set objFilter = Nothing
   Set criterion = Nothing
   Set objInstrLoop = Nothing
   Set objInstrLoops = Nothing
   Exit Sub
End If

Dim objPlantItem As LMPlantItem
Dim objPlantItems As LMPlantItems
Set objPlantItems = objInstrLoop.PlantItems

Debug.Print "Number of plant items in the InstrLoop = " & objPlantItems.Count
```

```
    'print plantitems in the instrument loop
    Dim i As Integer
    i = 1
    For Each objPlantItem In objPlantItems
        Debug.Print "ItemTypeName No. " & i & " " & objPlantItem.ItemTypeName & " ID =" &
objPlantItem.ID
        i = i + 1
    Next

    'add an instrument to the instrument loop
    Dim objInstr As LMInstrument
    Set objInstr = datasource.GetInstrument(CONST_SPID_OfflineInstrument)
    Debug.Print "Number of PlantItemGroups that are associated with this instrument= " &
objInstr.PlantItemGroups.Count
    objInstr.PlantItemGroups.Add objInstrLoop.AsLMPlantItemGroup
    'objInstr.Commit
    Debug.Print "Number of PlantItemGroups that are associated with this instrument= " &
objInstr.PlantItemGroups.Count

    Set datasource = Nothing
    Set objFilter = Nothing
    Set criterion = Nothing
    Set objInstrLoops = Nothing
    Set objInstrLoop = Nothing
    Set objPlantItems = Nothing
    Set objPlantItem = Nothing
    Set objInstr = Nothing
```

32. LOADINSTRUMENTS

a) Purpose

To navigate the relationship between Instrloop and Instrument through LoadInstruments method.

b) Problem Statement

Place couple instrloops, and couple instruments, then make association between them. Write a standalone application to find instruments associated with instrloops through LoadInstruments method.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource
Dim objInstrLoops As LMInstrLoops
Dim objInstrLoop As LMInstrLoop
Dim objInstruments As LMInstruments
Dim objInstrument As LMInstrument

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Set objInstrLoops = New LMInstrLoops
objInstrLoops.Collect datasource

Debug.Print objInstrLoops.Count

Set objInstruments = objInstrLoops.LoadInstruments

Debug.Print objInstruments.Count

For Each objInstrument In objInstruments
   Debug.Print objInstrument.Attributes("ItemTag").Value
   Debug.Print objInstrument.PlantItemGroups.Nth(1).Attributes("ItemTag")

Next

For Each objInstrLoop In objInstrLoops
   Debug.Print objInstrLoop.Attributes("ItemTag").Value
   If Not objInstrLoop.Instruments Is Nothing Then
      Debug.Print objInstrLoop.Instruments.Count
   End If

Next

Set objInstrLoops = Nothing
Set objInstrLoop = Nothing
Set objInstruments = Nothing
Set objInstrument = Nothing
Set datasource = Nothing
```

33. IDENTIFY NOZZLE AND EQUIPMENT

a) Purpose

To access nozzles on a vessel by navigating the relationship between nozzles and vessels.

b) Problem Statement

Place a vessel. Place two different nozzles on the vessel. Write a standalone application to retrieve the following properties of the nozzle:
SP_ID, aabbcc code, ID of equipment that the nozzle is connected to, Flowdirection, Nozzle type.

c) Solution

1. obtain Vessel object by SP_ID
2. use LMAVessel.nozzles to get a collection of nozzle belong to this Vessel

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim vessel As LMVessel
Set vessel = datasource.GetVessel(CONST_SPID_Vessel)

Dim nozzles As LMNozzles
Set nozzles = vessel.nozzles

Debug.Print nozzles.Count
Dim nozzle As LMNozzle

For Each nozzle In nozzles
   Debug.Print "Nozzle ID = " & nozzle.ID
   Debug.Print "Nozzle's aabbcc code = " & nozzle.Attributes("aabbcc_code").Value
   Debug.Print "ID of equipment that the nozzle is connected to = " & nozzle.EquipmentID
   Debug.Print "Nozzle's flow direction = " & nozzle.Attributes("FlowDirection").Value

   Debug.Print "Nozzle type = " & nozzle.Attributes("NozzleType").Value
   Debug.Print
Next

Set datasource = Nothing
Set vessel = Nothing
Set nozzles = Nothing
Set nozzle = Nothing
```

34. PIPINGCOMP AND INLINECOMP

a) Purpose

To navigate the relationship between PipingComp and InlineComp.

b) Problem Statement

Place a Valve. Write a standalone application to navigate from pipingcomp to piperun and from piperun to pipingcomp through InlineComp.

c) Solution

1. use LMADataSource.GetPipingComp
2. loop LMPipingComp.InlineComps
3. use LMInlinecomp.PipeRunObject

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objPipingComp As LMPipingComp
Dim objPiperun As LMPipeRun
Dim objInlineComp As LMInlineComp

Set objPipingComp = datasource.GetPipingComp(CONST_SPID_PipingComp)

If objPipingComp.InlineComps.Count = 1 Then
   Set objPiperun = objPipingComp.InlineComps.Nth(1).PipeRunObject

   Debug.Print "PipeRun ItemTag: " & objPiperun.Attributes("ItemTag").Value

   For Each objInlineComp In objPiperun.InlineComps
      Set objPipingComp = Nothing
      Set objPipingComp = objInlineComp.PipingCompObject
      If Not objPipingComp Is Nothing Then
         Debug.Print "PipingComp Type: " &
objPipingComp.Attributes("PipingCompType").Value
      End If
   Next
End If

Set datasource = Nothing
Set objPipingComp = Nothing
Set objPiperun = Nothing
Set objInlineComp = Nothing
```

35. INSTURMENT AND INLINECOMP

a) Purpose

To navigate the relationship between Inline-Instrument and InlineComp.

b) Problem Statement

Place a Instrument Valve. Write a standalone application to navigate from inline-instrument to piperun and from piperun to inline-instrument through inlinecomp.

c) Solution

1. use LMADataSource.GetInstrument
2. loop LMInstrument.InlineComps
3. use LMInlinecomp.InstrumentObject

◊ **Example code**

```
'be aware of that only inline instrument associate with InlineComp object.

Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objInstrument As LMInstrument
Dim objPiperun As LMPipeRun
Dim objInlineComp As LMInlineComp

Set objInstrument = datasource.GetInstrument(CONST_SPID_InlineInstrument)

If objInstrument.Attributes("IsInline").Value = True Then
   Set objPiperun = objInstrument.InlineComps.Nth(1).PipeRunObject

   Debug.Print "PipeRun ItemTag: " & objPiperun.Attributes("ItemTag").Value

   For Each objInlineComp In objPiperun.InlineComps
      Set objInstrument = Nothing
      Set objInstrument = objInlineComp.InstrumentObject
      If Not objInstrument Is Nothing Then
         Debug.Print "Instrument Type: " & objInstrument.Attributes("InstrumentType").Value
      End If
   Next
End If

Set datasource = Nothing
Set objInstrument = Nothing
Set objInlineComp = Nothing
```

36. OFFLINE INSTRUMENT AND SIGNALRUN

a) Purpose

Explore the relationship between offline instrument and SignalRun.

b) Problem Statement

Place an offline instrument, and then place couple singalruns connected with it. Write a standalone application to navigate from offline-instrument to signalrun.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objInstrument As LMInstrument
Dim objSignalRun As LMSignalRun

Set objInstrument = datasource.GetInstrument(CONST_SPID_OfflineInstrument)
Set objSignalRun = objInstrument.SignalRunObject
Debug.Print objSignalRun.Attributes("SignalType").Value

Debug.Print objSignalRun.Instruments.Count
For Each objInstrument In objSignalRun.Instruments
   Debug.Print objInstrument.Attributes("InstrumentType").Value
Next

Set datasource = Nothing
Set objInstrument = Nothing
Set objSignalRun = Nothing
```

37. ACCESS INSTRUMETN FUNCTIONS

a) Purpose

Explore the relationship between instrument and its functions

b) Problem Statement

Place an instrument, and populate properties for its functions. Write a standalone application to access instrument functions.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objInstrument As LMInstrument
Dim objInstrFailMode As LMInstrFailMode
Dim objInstrFunction As LMInstrFunction
Dim objInstrOption As LMInstrFunction
Dim objAttribute As LMAAttribute

Set objInstrument = datasource.GetInstrument(CONST_SPID_OfflineInstrument)
Debug.Print objInstrument.InstrFailModes.Count
For Each objInstrFailMode In objInstrument.InstrFailModes
   For Each objAttribute In objInstrFailMode.Attributes
      Debug.Print "Attribute Name=" & objAttribute.name & Space(50 - Len(objAttribute.name))
& "  Value=" & objAttribute.Value
   Next
Next

Debug.Print objInstrument.InstrFunctions.Count
For Each objInstrFunction In objInstrument.InstrFunctions
   For Each objAttribute In objInstrFunction.Attributes
      Debug.Print "Attribute Name=" & objAttribute.name & Space(50 - Len(objAttribute.name))
& "  Value=" & objAttribute.Value
   Next
Next

Debug.Print objInstrument.InstrOptions.Count
For Each objInstrOption In objInstrument.InstrOptions
   For Each objAttribute In objInstrOption.Attributes
      Debug.Print "Attribute Name=" & objAttribute.name & Space(50 - Len(objAttribute.name))
& "  Value=" & objAttribute.Value
   Next
Next

Set datasource = Nothing
Set objInstrument = Nothing
Set objInstrFailMode = Nothing
```

```
Set objInstrFunction = Nothing
Set objInstrOption = Nothing
Set objAttribute = Nothing
```

38. IDENTIFY CONNECTORS OF A PIPERUN

a) Purpose

To traverse the relationships from the Model DataModel to the Drawing DataModel

b) Problem Statement

Place a piperun between two nozzles and place two valves on it. Populate the ItemTag of the piperun with a value (eg. 15L – GCD). Retrieve the piperun by filtering for the piperun's ItemTag and locate all of its representations and connector representations.

c) Solution

1. Dim LMPipeRun, LMConnector, LMRepresentation
2. LMConnector is subclass of LMRepresentation, and its RepresentationType is "Connector".

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objFilter As LMAFilter
Dim criterion As LMACriterion
Set criterion = New LMACriterion
Set objFilter = New LMAFilter

criterion.SourceAttributeName = "ItemTag"
criterion.ValueAttribute = "01110-GCD"
criterion.Operator = "="
objFilter.ItemType = "PipeRun"
objFilter.Criteria.Add criterion

Dim piperun As LMPipeRun
Dim piperuns As LMPipeRuns
Set piperuns = New LMPipeRuns
piperuns.Collect datasource, Filter:=objFilter

Dim connector As LMConnector
Dim representation As LMRepresentation

For Each piperun In piperuns
   For Each representation In piperun.Representations
      If representation.RepresentationType = "Connector" Then
         Set connector = datasource.GetConnector(representation.ID)
         Debug.Print connector.Attributes("ItemStatus").Value
         Debug.Print "Model itme type = " &
connector.ModelItemObject.Attributes("ItemTypeName").Value
      End If
   Next
```

```
Next

Set datasource = Nothing
Set objFilter = Nothing
Set criterion = Nothing
Set piperuns = Nothing
Set piperun = Nothing
Set connector = Nothing
Set representation = Nothing
```

39. FIND FILE NMAE OF A SYMBOL

a) Purpose

Find file name of a symbol

b) Problem Statement

Place a vessel, then navigate from vessel to symbol, and get the file name of the vessel from the symbol object.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objVessel As LMVessel
Dim objSymbol As LMSymbol

Set objVessel = datasource.GetVessel(CONST_SPID_Vessel)
Set objSymbol = datasource.GetSymbol(objVessel.Representations.Nth(1).ID)
Debug.Print objSymbol.Attributes("FileName").Value

'if you are required to find file name of a piperun, what should you do?
Set datasource = Nothing
Set objVessel = Nothing
Set objSymbol = Nothing
```

40. FIND X, Y COORDINATES OF SYMBOL

a) Purpose

Find X, Y Coordinates of symbol

b) Problem Statement

Place a vessel, then navigate from vessel to symbol, and get the X, Y Coordinates of the vessel from the symbol object.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objVessel As LMVessel
Dim objSymbol As LMSymbol

Set objVessel = datasource.GetVessel(CONST_SPID_Vessel)
Set objSymbol = datasource.GetSymbol(objVessel.Representations.Nth(1).ID)
Debug.Print "XCoordinate = " & objSymbol.Attributes("XCoordinate").Value
Debug.Print "YCoordinate = " & objSymbol.Attributes("YCoordinate").Value

Set datasource = Nothing
```

41. FIND X, Y COORDINATES OF PIPERUN

a) Purpose

Find X, Y Coordinates of Piperun

b) Problem Statement

Place a Piperun, then navigate from Piperun to Connector, and get the X, Y Coordinates of the Piperun from Connector object.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objPiperun As LMPipeRun
Dim objRep As LMRepresentation
Dim objConnector As LMConnector
Dim objConnectorVertex As LMConnectorVertex
Dim objSymbol As LMSymbol

Set objPiperun = datasource.GetPipeRun(CONST_SPID_PipeRun)

For Each objRep In objPiperun.Representations
   If objRep.Attributes("RepresentationType").Value = "Connector" And
objRep.Attributes("ItemStatus").Value = "Active" Then
      Set objConnector = datasource.GetConnector(objRep.ID)
      For Each objConnectorVertex In objConnector.ConnectorVertices
         Debug.Print "XCoordinate = " & objConnectorVertex.Attributes("XCoordinate").Value
         Debug.Print "YCoordinate = " & objConnectorVertex.Attributes("YCoordinate").Value
      Next
   End If
Next

'if the X, Y Coordinates are on the symbol that is connected with the Connector, what should
you do?
Set datasource = Nothing
Set objPiperun = Nothing
Set objRep = Nothing
Set objConnector = Nothing
Set objConnectorVertex = Nothing
Set objSymbol = Nothing
```

42. FIND LABELS OF A SYMBOL

a) Purpose

Find labels on a symbol

b) Problem Statement

Place a vessel, then place couple labels on it, then navigate from Vessel to Representation, then find labels on the Vessel.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objVessel As LMVessel
Dim objSymbol As LMSymbol
Dim objLabelPersist As LMLabelPersist
Dim objAttr As LMAAttribute
Dim objLeaderVertex As LMLeaderVertex

Set objVessel = datasource.GetVessel(CONST_SPID_Vessel)
Set objSymbol = datasource.GetSymbol(objVessel.Representations.Nth(1).ID)

Debug.Print "Total labels on this symbol: " & objSymbol.LabelPersists.Count

For Each objLabelPersist In objSymbol.LabelPersists
   For Each objAttr In objLabelPersist.Attributes
      Debug.Print "Attribute Name=" & objAttr.name & Space(50 - Len(objAttr.name)) & "
Value=" & objAttr.Value
   Next
   For Each objLeaderVertex In objLabelPersist.LeaderVertices
      Debug.Print "XCoordinate = " & objLeaderVertex.Attributes("XCoordinate").Value
      Debug.Print "YCoordinate = " & objLeaderVertex.Attributes("YCoordinate").Value
   Next
Next

'if you are required to find labels of a piperun, what should you do?
Set datasource = Nothing
Set objVessel = Nothing
Set objSymbol = Nothing
Set objLabelPersist = Nothing
Set objAttr = Nothing
Set objLeaderVertex = Nothing
```

43. FIND PARENT REPRESENTATION OF A LABEL

a) Purpose

Find Parent Representation of a label.

b) Problem Statement

Place a label on to a vessel, get label object first, then navigate from label to find Representation it labels, then navigate from the Representation to ModelItem.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objLabelPersist As LMLabelPersist
Dim objRep As LMRepresentation
Dim objModelItem As LMModelItem
Dim objAttr As LMAAttribute

Set objLabelPersist = datasource.GetLabelPersist(CONST_SPID_LabelPersist)
Set objRep = objLabelPersist.RepresentationObject
Set objModelItem = objRep.ModelItemObject

Debug.Print "Total labels on this symbol: " & objRep.LabelPersists.Count

For Each objAttr In objModelItem.Attributes
   Debug.Print "Attribute Name=" & objAttr.name & Space(50 - Len(objAttr.name)) & "
Value=" & objAttr.Value
Next

'if parent is piperun, what should you do?
Set datasource = Nothing
Set objLabelPersist = Nothing
Set objRep = Nothing
Set objModelItem = Nothing
Set objAttr = Nothing
```

44. FIND PARENT DRAWING FOR A SYMBOL

a) Purpose

Find parent drawing of a symbol.

b) Problem Statement

Place a vessel on a drawing. Write a standalone application to find drawing this vessel is on.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objVessel As LMVessel
Dim objSymbol As LMSymbol

Set objVessel = datasource.GetVessel(CONST_SPID_Vessel)
Set objSymbol = datasource.GetSymbol(objVessel.Representations.Nth(1).ID)

Dim objDrawing As LMDrawing
Set objDrawing = objSymbol.DrawingObject
Debug.Print objDrawing.Attributes("Name").Value

Set datasource = Nothing
Set objVessel = Nothing
Set objSymbol = Nothing
Set objDrawing = Nothing
```

45. FIND ACTIVE DRAWING AND PLANTITEMS IN IT

a) Purpose

Directly find the active drawing not through an item first, then find all PlantItems in it.

b) Problem Statement

Open a drawing. Write a standalone application to find what active drawing is and how many PlantItems in it.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

'have to user PIDDatasource
Set datasource = PIDDataSource

Dim objDrawing As LMDrawing
Set objDrawing = datasource.GetDrawing(datasource.PIDMgr.Drawing.ID)
Debug.Print objDrawing.Attributes("Name").Value

Dim objFilter As LMAFilter
Set objFilter = New LMAFilter

objFilter.Criteria.AddNew ("FirstOne")
objFilter.Criteria.item("FirstOne").SourceAttributeName = "Representation.Drawing.Name"
objFilter.Criteria.item("FirstOne").ValueAttribute = objDrawing.Attributes("Name").Value
objFilter.Criteria.item("FirstOne").Operator = "="
objFilter.ItemType = "PlantItem"

objFilter.Criteria.AddNew ("SecondOne")
objFilter.Criteria.item("SecondOne").SourceAttributeName = "ItemStatus"
objFilter.Criteria.item("SecondOne").ValueAttribute = 1
objFilter.Criteria.item("SecondOne").Operator = "="
objFilter.Criteria.item("SecondOne").Conjunctive = True

Dim objPlantItems As LMPlantItems
Set objPlantItems = New LMPlantItems
objPlantItems.Collect datasource, Filter:=objFilter

Debug.Print "Number of plantitems in active drawing: " & objPlantItems.Count

Set datasource = Nothing
Set objDrawing = Nothing
Set objFilter = Nothing
Set objPlantItems = Nothing
```

46. FILTER FOR ITEMS IN PLANT STOCKPILE

a) Purpose

To filter for all items in plant stockpile.

b) Problem Statement

Write a standalone application to get all items in plant stockpile.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If
Dim objFilter As LMAFilter
Set objFilter = New LMAFilter
objFilter.Criteria.AddNew ("FirstOne")
objFilter.Criteria.item("FirstOne").SourceAttributeName = "ItemStatus"
objFilter.Criteria.item("FirstOne").ValueAttribute = 1
objFilter.Criteria.item("FirstOne").Operator = "="
objFilter.ItemType = "Vessel"

objFilter.Criteria.AddNew ("SecondOne")
objFilter.Criteria.item("SecondOne").SourceAttributeName = "Representation.InStockpile"
objFilter.Criteria.item("SecondOne").ValueAttribute = 2 '2 is index stands for True
objFilter.Criteria.item("SecondOne").Operator = "="
objFilter.Criteria.item("SecondOne").Conjunctive = True

objFilter.Criteria.AddNew ("ThirdOne")
objFilter.Criteria.item("ThirdOne").SourceAttributeName = "Representation.SP_DrawingId"
objFilter.Criteria.item("ThirdOne").ValueAttribute = 0 '0 stands Plant Stockpile
objFilter.Criteria.item("ThirdOne").Operator = "="
objFilter.Criteria.item("ThirdOne").Conjunctive = True

Dim objVessels As LMVessels
Set objVessels = New LMVessels
objVessels.Collect datasource, Filter:=objFilter

Debug.Print objVessels.Count

Dim objVessel As LMVessel
For Each objVessel In objVessels
   Debug.Print objVessel.Attributes("ItemTag").Value
   Debug.Print objVessel.Representations.Nth(1).Attributes("InStockpile").Value
Next

Set datasource = Nothing
Set objFilter = Nothing
Set objVessels = Nothing
Set objVessel = Nothing
```

47. IDENTIFY ITEMS CONNECTED TO A PIPERUN

a) Purpose

To traverse the relationships from LMConnector to LMSymbol

b) Problem Statement

Place a piperun between two nozzles and place two valves on it. Populate the ItemTag of the piperun with a value (eg. unit1100-GCD). Retrieve the piperun by filtering for the piperun's ItemTag. Identify all of the items connected to the ends of the connectors of the piperun.

c) Solution

1. Dim LMPipeRun, LMConnector, LMRepresentation
2. LMConnector has properties "`ConnectItem1SymbolObject`" and "`ConnectItem2SymbolObject`", that returns the symbol object connected to the Connector

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objFilter As LMAFilter
Dim criterion As LMACriterion

Set criterion = New LMACriterion
Set objFilter = New LMAFilter

criterion.SourceAttributeName = "ItemTag"
criterion.ValueAttribute = "01110-GCD"
criterion.Operator = "="
objFilter.ItemType = "PipeRun"
objFilter.Criteria.Add criterion

Dim piperun As LMPipeRun
Dim piperuns As LMPipeRuns
Set piperuns = New LMPipeRuns
piperuns.Collect datasource, Filter:=objFilter

Dim connector As LMConnector
Dim representation As LMRepresentation
For Each piperun In piperuns
   For Each representation In piperun.Representations
      If representation.RepresentationType = "Connector" Then
         Set connector = datasource.GetConnector(representation.ID)
         If Not connector.ConnectItem1SymbolObject Is Nothing Then
            Debug.Print connector.ConnectItem1SymbolObject.ModelItemObject.ItemTypeName
_
               & " - ID: " & connector.ConnectItem1SymbolObject.ModelItemID
         End If
```

```
            If Not connector.ConnectItem2SymbolObject Is Nothing Then
              Debug.Print connector.ConnectItem2SymbolObject.ModelItemObject.ItemTypeName
_
                  & " - ID: " & connector.ConnectItem2SymbolObject.ModelItemID
            End If
          End If
        Next
      Next

      Set datasource = Nothing
      Set objFilter = Nothing
      Set criterion = Nothing
      Set piperuns = Nothing
      Set piperun = Nothing
      Set connector = Nothing
      Set representation = Nothing
```

48. IDENTIFY THE PIPERUN ASSOCIATED WITH THE PIPINGCOMP

a) Purpose

To traverse the relationships from LMPipingComp to LMPipeRun

b) Problem Statement

Place a piperun, then place a valve in the middle of the piperun. Assume you only know the SP_ID of the valve. Write a standalone application to obtain the PipeRun on which the valve is sitting, then read properties (ID and Name) of the piperun.

c) Solution

1. Dim LMPipingComp, LMSymbol, LMPipeRun
2. LMSymbol has a property "Connect1Connectors", that returns the collection of LMConnector object connected to the Symbol, then from LMConnector.ModelItemID, returns the ModelItemID of the Connector, which is the SP_ID of the PipeRun.
3. Alternate, LMPipingComp has method "InlineComps", which returns the collection of LMInlineComps associated with the PipingComp, then, LMInlineComp has a property "PipeRunID", which returns the SP_ID of the PipeRun, on which the PipingComp is sitting.

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objPipingComp As LMPipingComp
Set objPipingComp = datasource.GetPipingComp(CONST_SPID_PipingComp)

Dim objPipingCompSym As LMSymbol
Set objPipingCompSym = datasource.GetSymbol(objPipingComp.Representations.Nth(1).ID)

Dim objPiperun As LMPipeRun
Set objPiperun =
datasource.GetPipeRun(objPipingCompSym.Connect1Connectors.Nth(1).ModelItemID)
'or you can obtain PipeRun as following
'Set objPipeRun = 'datasource.GetPipeRun(objPipingComp.inlinecomps.Nth(1).PipeRunID)
Debug.Print "PipeRun ID = " & objPiperun.ID
Debug.Print "PipeRun ItemTag = " & objPiperun.Attributes("ItemTag").Value
'Note they are the same item
Debug.Print "PipingComp ID = " & objPipingComp.ID
Debug.Print "InlineComp ID = " & objPiperun.InlineComps.Nth(1).ID

Set datasource = Nothing
Set objPipingComp = Nothing
Set objPipingCompSym = Nothing
Set objPiperun = Nothing
```

49. NAVIGATE ITEMS TO GET PARENT ITEM

a) Purpose

To navigate items such as PipeRun, Connectors, nozzles, to get the parent item of nozzle – Equipment.

b) Problem Statement

Place a vessel with a nozzle on it, then place a piperun connected to the nozzle, then place a valve in the piperun. Write a standalone application to navigate from the piperun, through the piperun's connectors and the nozzle to arrive at the vessel. Print out some properties of the vessel.

c) Solution

1. Dim LMAFilter, LMACriterion, LMPipeRuns
2. From LMPipeRun.Representations, obtain LMConnector, whose RepresentationType is "Connector", then, from LMConnector.ConnnectItem1SymbolObject or LMConnector.ConnecItem2SymbolObject find the Symbol object connect to the Connector, then, from LMSymbol.ModelItemID find the SP_ID of the symbol, then the Nozzle object is located, and LMNozzle has a property "EquipmentObject", which returns the LMEquipment object, which is connected to the Nozzle

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objFilter As LMAFilter
Dim criterion As LMACriterion

Set objFilter = New LMAFilter
Set criterion = New LMACriterion
criterion.SourceAttributeName = "ItemTag"
criterion.ValueAttribute = "01110-GCD"
criterion.Operator = "="
objFilter.ItemType = "PipeRun"
objFilter.Criteria.Add criterion

Dim piperun As LMPipeRun
Dim piperuns As LMPipeRuns
Set piperuns = New LMPipeRuns
piperuns.Collect datasource, Filter:=objFilter

Debug.Print "Number of Piperuns retrieved = " & piperuns.Count

Dim representation As LMRepresentation
Dim connector As LMConnector
Dim nozzle As LMNozzle
Dim objEquipment As LMEquipment
```

```
    For Each piperun In piperuns
        For Each representation In piperun.Representations
            If representation.RepresentationType = "Connector" Then
                Set connector = datasource.GetConnector(representation.ID)
                If Not connector.ConnectItem1SymbolObject Is Nothing Then
                    If connector.ConnectItem1SymbolObject.ModelItemObject.ItemTypeName =
"Nozzle" Then
                        Set nozzle =
datasource.GetNozzle(connector.ConnectItem1SymbolObject.ModelItemID)
                        Exit For
                    End If
                End If
                If Not connector.ConnectItem2SymbolObject Is Nothing Then
                    If connector.ConnectItem2SymbolObject.ModelItemObject.ItemTypeName =
"Nozzle" Then
                        Set nozzle =
datasource.GetNozzle(connector.ConnectItem2SymbolObject.ModelItemID)
                        Exit For
                    End If
                End If
            End If
        Next
    Next

    Set objEquipment = nozzle.EquipmentObject
    Debug.Print "ID = " & objEquipment.ID
    Debug.Print "EquipmentType = " & objEquipment.EquipmentType
    Debug.Print "ItemTag = " & objEquipment.Attributes("ItemTag").Value
    Debug.Print "Nozzles belong to the vessel = " & objEquipment.nozzles.Count

    Set datasource = Nothing
    Set objFilter = Nothing
    Set criterion = Nothing
    Set objEquipment = Nothing
    Set nozzle = Nothing
    Set connector = Nothing
    Set representation = Nothing
    Set piperuns = Nothing
    Set piperun = Nothing
```

50. NAVIGATE THROUGH BRANCHPOINT

a) Purpose

To navigate through branch point on a piperun.

b) Problem Statement

Place a vessel with two nozzles on it with ItemTags N10 and N20 respectively. Generate the itemtag for the vessel by assigning a TagPrefix. Place a straight piperun starting from the nozzle (N10) and end it in space with no connection. Start a branch piperun from some point on the first piperun, and extend it to the second nozzle (N20). Write a standalone application to navigate from the first nozzle (N10), through the piperun connectors and the second nozzle to arrive back at the vessel.

c) Solution

1. Dim LMAFileter, LMACriterion, LMSymbol, LMPipeRun
2. BranchPoint is a Symbol Representation of the PipeRun on which it is sitting, BranchPoint's RepresentationType is "Branch"

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objFilter As LMAFilter
Dim criterion As LMACriterion
Set criterion = New LMACriterion
Set objFilter = New LMAFilter

criterion.SourceAttributeName = "ItemTag"
criterion.ValueAttribute = "N10"
criterion.Operator = "="
objFilter.ItemType = "Nozzle"
objFilter.Criteria.Add criterion

Dim nozzle As LMNozzle
Dim nozzles As LMNozzles
Set nozzles = New LMNozzles
nozzles.Collect datasource, Filter:=objFilter
'make sure only one nozzle is obtained
If nozzles.Count = 1 Then
   Set nozzle = nozzles.Nth(1)
Else

   Set datasource = Nothing
   Set nozzles = Nothing
   Set nozzle = Nothing
   Exit Sub
End If

'get nozzle symbol
```

```
    Dim symbol1 As LMSymbol
    Set symbol1 = datasource.GetSymbol(nozzle.Representations.Nth(1).ID)

    'check nozzle symbol's connect1connectors & connect2connectors information to find a
connector
    'connected to the nozzle
    Dim Tconnector As LMConnector
    Dim connector As LMConnector
    If symbol1.Connect1Connectors.Count >= 1 Then
       For Each Tconnector In symbol1.Connect1Connectors
          If Tconnector.ItemStatus = "Active" Then
             If Tconnector.ModelItemObject.ItemTypeName = "PipeRun" Then
                Set connector = Tconnector
             End If
          End If
       Next
    End If

    If connector Is Nothing And symbol1.Connect2Connectors.Count >= 1 Then
       For Each Tconnector In symbol1.Connect2Connectors
          If Tconnector.ItemStatus = "Active" Then
             If Tconnector.ModelItemObject.ItemTypeName = "PipeRun" Then
                Set connector = Tconnector
             End If
          End If
       Next
    End If

    'once the connector is found, check connectitem1symbolobject and connectitem2symbolobject
information
    'to find the BranchPoint.
    'The modelitem for the BranchPoint symbol is the piperun, but the representationtype is
"Branch"
    Dim branchsymbol As LMSymbol
    If Not connector.ConnectItem1SymbolObject Is Nothing Then
       If connector.ConnectItem1SymbolObject.ModelItemObject.ItemTypeName = "PipeRun"
Then
          If connector.ConnectItem1SymbolObject.AsLMRepresentation.RepresentationType =
"Branch" Then
             Set branchsymbol = connector.ConnectItem1SymbolObject
          End If
       End If
    End If
    If branchsymbol Is Nothing And Not connector.ConnectItem2SymbolObject Is Nothing Then
       If connector.ConnectItem2SymbolObject.ModelItemObject.ItemTypeName = "PipeRun"
Then
          If connector.ConnectItem2SymbolObject.AsLMRepresentation.RepresentationType =
"Branch" Then
             Set branchsymbol = connector.ConnectItem2SymbolObject
          End If
       End If
    End If

    'After the BranchPoint is located, use again the connect1connectors & connect2connectors
method to locate
```

```
    'the connector connected to the BranchPoint, and make sure this connector is point back to the
  new piperun
    Dim connector2 As LMConnector
    Dim connector3 As LMConnector
    If branchsymbol.Connect1Connectors.Count >= 1 Then
      For Each connector2 In branchsymbol.Connect1Connectors
        If connector2.ModelItemID <> connector.ModelItemID Then
          Set connector3 = connector2
          Exit For
        End If
      Next
    End If

    If connector3 Is Nothing And branchsymbol.Connect2Connectors.Count >= 1 Then
      For Each connector2 In branchsymbol.Connect2Connectors
        If connector2.ModelItemID <> connector.ModelItemID Then
          Set connector3 = connector2
          Exit For
        End If
      Next
    End If

    'After second connector is located, check connectitem1symbolobject &
  connectitem2symbolobject to find
    'the second nozzle
    Dim nozzle2 As LMNozzle
    If Not connector3.ConnectItem1SymbolObject Is Nothing Then
      If connector3.ConnectItem1SymbolObject.ModelItemObject.ItemTypeName = "Nozzle" Then
        Set nozzle2 =
  datasource.GetNozzle(connector3.ConnectItem1SymbolObject.ModelItemID)
      End If
    End If
    If nozzle2 Is Nothing And Not connector3.ConnectItem2SymbolObject Is Nothing Then
      If connector3.ConnectItem2SymbolObject.ModelItemObject.ItemTypeName = "Nozzle" Then
        Set nozzle2 =
  datasource.GetNozzle(connector3.ConnectItem2SymbolObject.ModelItemID)
      End If
    End If

    'Print out two nozzles' name and itmetag of the vessel they attached
    Debug.Print "Nozzle2 itemtag = " & nozzle2.Attributes("ItemTag").Value
    Debug.Print "Nozzle itemtag = " & nozzle.Attributes("ItemTag").Value
    Debug.Print "vessel nozzle2 attached = " &
  nozzle2.EquipmentObject.Attributes("ItemTag").Value
    Debug.Print "vessel nozzle attached =" & nozzle.EquipmentObject.Attributes("ItemTag").Value
    Set datasource = Nothing
    Set objFilter = Nothing
    Set criterion = Nothing
    Set nozzle = Nothing
    Set nozzles = Nothing
    Set nozzle2 = Nothing
    Set connector = Nothing
    Set connector2 = Nothing
    Set connector3 = Nothing
    Set branchsymbol = Nothing
```

51. NAVIGATE THROUGH OPC

a) Purpose

To get familiar with navigation through OPC

b) Problem Statement

Place an OPC, then place its pair OPC into another drawing, and connected the pair OPC to a piperun with itemtag populated. Then write a standalone application to navigate for OPC to its pairOPC, and print out the itemtag of piperun that the pair OPC is connected with.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objFilter As LMAFilter
Dim criterion As LMACriterion
Set criterion = New LMACriterion
Set objFilter = New LMAFilter

criterion.SourceAttributeName = "Representation.Drawing.Name"
criterion.ValueAttribute = "unit2d"
criterion.Operator = "="
objFilter.ItemType = "OPC"
objFilter.Criteria.Add criterion

Dim objOPC As LMOPC
Dim objOPCs As LMOPCs
Dim objpairOPC As LMOPC
Dim objRep As LMRepresentation
Dim objPiperun As LMPipeRun
Dim objSym As LMSymbol
Dim objConnector As LMConnector

Set objOPCs = New LMOPCs
objOPCs.Collect datasource, Filter:=objFilter

Debug.Print "Total OPCs were found: " & objOPCs.Count

For Each objOPC In objOPCs
   Set objpairOPC = objOPC.pairedWithOPCObject
   For Each objRep In objpairOPC.Representations
      If objRep.DrawingID > 0 Then
         Debug.Print "pairOPC is on drawing: " &
objRep.DrawingObject.Attributes("Name").Value
      End If
```

```
        Set objSym = datasource.GetSymbol(objRep.ID)
        For Each objConnector In objSym.Connect1Connectors
           Set objPiperun = datasource.GetPipeRun(objConnector.ModelItemID)
           Debug.Print "pairOPC is connected to Piperun: " &
objPiperun.Attributes("ItemTag").Value
        Next
        For Each objConnector In objSym.Connect2Connectors
           Set objPiperun = datasource.GetPipeRun(objConnector.ModelItemID)
           Debug.Print objPiperun.Attributes("ItemTag").Value
        Next
     Next
  Next

  Set datasource = Nothing
  Set objOPCs = Nothing
  Set objOPC = Nothing
  Set objpairOPC = Nothing
  Set objRep = Nothing
  Set objSym = Nothing
  Set objPiperun = Nothing
  Set objConnector = Nothing
```

52. ACCESS RELATIONSHIP FROM REPRESENTATION

a) Purpose

To access relationship object from representation object.

b) Problem Statement

Place piperun, then place a valve on the piperun. Write a standalone application to obtain the valve, then get the relationship objects belong to this valve.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objPipingComp As LMPipingComp
Dim objRepresentation As LMRepresentation
Dim objRelationships As LMRelationships
Dim objRelationship As LMRelationship
Dim objAttribute As LMAAttribute

Set objPipingComp = datasource.GetPipingComp(CONST_SPID_PipingComp)
Set objRepresentation = objPipingComp.Representations.Nth(1)
Set objRelationships = objRepresentation.Relation1Relationships
For Each objRelationship In objRelationships
   If Not objRelationship.Item1RepresentationObject Is Nothing Then
      Debug.Print
objRelationship.Item1RepresentationObject.ModelItemObject.AsLMAItem.ItemType
   End If
   If Not objRelationship.Item2RepresentationObject Is Nothing Then
      Debug.Print
objRelationship.Item2RepresentationObject.ModelItemObject.AsLMAItem.ItemType
   End If
   For Each objAttribute In objRelationship.Attributes
      Debug.Print "Name: " & objAttribute.name & Space(20 - Len(objAttribute.name)) & "Value:
" & objAttribute.Value
   Next
Next

Set objRelationships = objRepresentation.Relation2Relationships
For Each objRelationship In objRelationships
   If Not objRelationship.Item1RepresentationObject Is Nothing Then
      Debug.Print
objRelationship.Item1RepresentationObject.ModelItemObject.AsLMAItem.ItemType
   End If
   If Not objRelationship.Item2RepresentationObject Is Nothing Then
      Debug.Print
objRelationship.Item2RepresentationObject.ModelItemObject.AsLMAItem.ItemType
   End If
```

```
        For Each objAttribute In objRelationship.Attributes
            Debug.Print "Name: " & objAttribute.name & Space(20 - Len(objAttribute.name)) & "Value:
" & objAttribute.Value
        Next
    Next

    Set datasource = Nothing
    Set objPipingComp = Nothing
    Set objRepresentation = Nothing
    Set objRelationships = Nothing
    Set objRelationship = Nothing
    Set objAttribute = Nothing
```

53. ACCESS INCONSISTENCY

a) Purpose

To access the Inconsistency.

b) Problem Statement

Write a standalone application to get all relationship objects belong to a drawing, then access the Inconsistency from relationship.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objFilter As LMAFilter
Dim criterion As LMACriterion

Set criterion = New LMACriterion
Set objFilter = New LMAFilter

criterion.SourceAttributeName = "Name"
criterion.ValueAttribute = "Automation1"
criterion.Operator = "="
objFilter.ItemType = "Drawing"
objFilter.Criteria.Add criterion

Dim objDrawing As LMDrawing
Dim objDrawings As LMDrawings
Set objDrawings = New LMDrawings
objDrawings.Collect datasource, Filter:=objFilter

Dim objRelationships As LMRelationships
Dim objRelationship As LMRelationship
Dim objInconsistencies As LMInconsistencies
Dim objInconsistency As LMInconsistency
Dim objAttribute As LMAAttribute

For Each objDrawing In objDrawings
   Set objRelationships = objDrawing.Relationships
   For Each objRelationship In objRelationships
      Set objInconsistencies = objRelationship.Inconsistencies
      For Each objInconsistency In objInconsistencies
         For Each objAttribute In objInconsistency.Attributes
            Debug.Print "Name: " & objAttribute.name & Space(20 - Len(objAttribute.name)) &
"Value: " & objAttribute.Value
         Next
```

```
        Next
    Next
Next

Set datasource = Nothing
Set objDrawings = Nothing
Set objDrawing = Nothing
Set objRelationships = Nothing
Set objRelationship = Nothing
Set objInconsistencies = Nothing
Set objInconsistency = Nothing
```

54. ACCESS RULEREFERENCE

a) Purpose

To access the RuleReference.

b) Problem Statement

Write a standalone application to get all relationship objects belong to a drawing, then access the RuleReference from relationship.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objFilter As LMAFilter
Dim criterion As LMACriterion

Set criterion = New LMACriterion
Set objFilter = New LMAFilter

criterion.SourceAttributeName = "Name"
criterion.ValueAttribute = "Automation1"
criterion.Operator = "="
objFilter.ItemType = "Drawing"
objFilter.Criteria.Add criterion

Dim objDrawing As LMDrawing
Dim objDrawings As LMDrawings
Set objDrawings = New LMDrawings
objDrawings.Collect datasource, Filter:=objFilter

Dim objRelationships As LMRelationships
Dim objRelationship As LMRelationship
Dim objRuleReferences As LMRuleReferences
Dim objRuleReference As LMRuleReference
Dim objAttribute As LMAAttribute

For Each objDrawing In objDrawings
   Set objRelationships = objDrawing.Relationships
   For Each objRelationship In objRelationships
      Set objRuleReferences = objRelationship.RuleReferences
      For Each objRuleReference In objRuleReferences
         For Each objAttribute In objRuleReference.Attributes
            Debug.Print "Name: " & objAttribute.name & Space(20 - Len(objAttribute.name)) &
"Value: " & objAttribute.Value
         Next
```

```
        Next
    Next
Next

Set datasource = Nothing
Set objDrawings = Nothing
Set objDrawing = Nothing
Set objRelationships = Nothing
Set objRelationship = Nothing
Set objRuleReferences = Nothing
Set objRuleReference = Nothing
```

55. ACCESS PLANTGROUP FROM PLANTITEM

a) Purpose

To access the PlantGroup to which the PlantItem belongs.

b) Problem Statement

Place a vessel. Write a standalone application to get the plantgroup to which the PlantItem is associated.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objVessel As LMVessel
Set objVessel = datasource.GetVessel(CONST_SPID_Vessel)

Dim objPlantGroup As LMPlantGroup

'get the plantgroup just above the drawing, in our case, should be unit
Set objPlantGroup = objVessel.PlantGroupObject
Debug.Print "PlantGroup Name = " & objPlantGroup.Attributes("Name").Value

Dim strParentID As String
strParentID = objPlantGroup.Attributes("ParentID").Value

Dim objParentPlantGroup As LMPlantGroup
Set objParentPlantGroup = datasource.GetPlantGroup(strParentID)
Debug.Print "Parent PlantGroup Name = " & objParentPlantGroup.Attributes("Name").Value

Set datasource = Nothing
Set objVessel = Nothing
Set objPlantGroup = Nothing
Set objParentPlantGroup = Nothing
```

56. ACCESS PLANTGROUP FROM DRAWING

a) Purpose

To access the PlantGroup to which the Drawing belongs.

b) Problem Statement

Place a vessel. Write a standalone application to obtain the drawing associated with the vessel. Get the plantgroup to which the drawing belongs.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objVessel As LMVessel
Set objVessel = datasource.GetVessel(CONST_SPID_Vessel)

Dim objPlantGroup As LMPlantGroup

'get the plantgroup just above the drawing, in our case, should be unit
Set objPlantGroup = objVessel.Representations.Nth(1).DrawingObject.PlantGroupObject
Debug.Print "PlantGroup Name = " & objPlantGroup.Attributes("Name").Value

Dim strParentID As String
strParentID = objPlantGroup.Attributes("ParentID").Value

Dim objParentPlantGroup As LMPlantGroup
Set objParentPlantGroup = datasource.GetPlantGroup(strParentID)
Debug.Print "Parent PlantGroup Name = " & objParentPlantGroup.Attributes("Name").Value

Set datasource = Nothing
Set objVessel = Nothing
Set objPlantGroup = Nothing
Set objParentPlantGroup = Nothing
```

57. ACCESS CUSTOMIZED PLANTGROUP

a) Purpose

To access the customized property of a user defined PlantGroup type.

b) Problem Statement

Create a new PlantGroup type, "SubArea", in SmartPlant Engineering Manager, then create a new Hierarcy template using this new PlantGroup. Then create a new plant using this new Hierarcy template, after creation of new plant, add a new property "T1" to the new PlantGroup. Write a standalone application to read this new property "T1".

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objVessel As LMVessel
Set objVessel = datasource.GetVessel(CONST_SPID_Vessel)    'get objVessel by id
Debug.Print objVessel.PlantGroupObject.Attributes("T1")
Debug.Print datasource.GetItem("SPMSubArea", objVessel.PlantGroupID).Attributes("T1")

Set datasource = Nothing
Set objVessel = Nothing
```

58. ACCESS WORKSHARE STIE

a) Purpose

To get familiar with the workshare site object in LLAMA.

b) Problem Statement

Place a Vessel, find the workshare site to which this vessel belongs. Print out properties of the workshare site. Browser relateionship between workshare site and other entities, such as PlantGroup, PlantItemGroup, OPC, and DrawingSite.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If


Dim objVessel As LMVessel
Set objVessel = datasource.GetVessel(CONST_SPID_Vessel)

Dim objDrawing As LMDrawing
Set objDrawing = datasource.GetDrawing(objVessel.Representations.Nth(1).DrawingID)

Dim objPlantGroup As LMPlantGroup
'get the plantgroup just above the drawing, in our case, should be unit
Set objPlantGroup = objDrawing.PlantGroupObject

Dim objWSSite As LMWSSite

Set objWSSite = objPlantGroup.WSSiteObject

Dim objAttr As LMAAttribute
Debug.Print "How many attributes? " & objWSSite.Attributes.Count
For Each objAttr In objWSSite.Attributes
   Debug.Print "Attribute Name: " & objAttr.name & "    Attribute Value: " & objAttr.Value
Next

Dim objOPC As LMOPC
Debug.Print "Total OPCs in WS Site: " & objWSSite.OPCs.Count
For Each objOPC In objWSSite.OPCs
   Debug.Print objOPC.Attributes("OPCTag").Value
   Debug.Print objOPC.WSSiteObject.Attributes("Name").Value
Next

Dim objPlantItemGroup As LMPlantItemGroup
Debug.Print "Total PlantItemGroups in WS Site: " & objWSSite.PlantItemGroups.Count
For Each objPlantItemGroup In objWSSite.PlantItemGroups
   Debug.Print objPlantItemGroup.Attributes("PlantItemGroupType").Value
```

```
    Debug.Print objPlantItemGroup.WSSiteObject.Attributes("Name").Value
Next


Debug.Print "Total PlantGroups in WS Site: " & objWSSite.PlantGroups.Count
For Each objPlantGroup In objWSSite.PlantGroups
    Debug.Print objPlantGroup.Attributes("PlantGroupType").Value
    Debug.Print objPlantGroup.Attributes("Name").Value
    Debug.Print objPlantGroup.WSSiteObject.Attributes("Name").Value
Next

Dim objDrawingSite As LMDrawingSite
Debug.Print "Total DrawingSites in WS Site: " & objWSSite.DrawingSites.Count
For Each objDrawingSite In objWSSite.DrawingSites
    Debug.Print objDrawingSite.Attributes("Name").Value
    Debug.Print objDrawingSite.WSSiteObject.Attributes("Name").Value
Next

Set datasource = Nothing
Set objVessel = Nothing
Set objDrawing = Nothing
Set objPlantGroup = Nothing
Set objAttr = Nothing
Set objOPC = Nothing
Set objPlantItemGroup = Nothing
Set objDrawingSite = Nothing
```

59. ACCESS DRAWINGSITE

a) Purpose

To get familiar with the drawingsite object in LLAMA.

b) Problem Statement

Get a drawingsite object, the print out properties of the drawingsite.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If
Dim objVessel As LMVessel
Set objVessel = datasource.GetVessel(CONST_SPID_Vessel)

Dim objDrawing As LMDrawing
Set objDrawing = datasource.GetDrawing(objVessel.Representations.Nth(1).DrawingID)

Dim objDrawingSite As LMDrawingSite
Set objDrawingSite = objDrawing.DrawingSites.Nth(1)

Dim objAttr As LMAAttribute
Debug.Print "How many attributes? " & objDrawingSite.Attributes.Count
For Each objAttr In objDrawingSite.Attributes
   Debug.Print "Attribute Name: " & objAttr.name & "    Attribute Value: " & objAttr.Value
Next

Dim objDrawingSubscriber As LMDrawingSubscriber
Debug.Print "Total drawing subscriber: " & objDrawingSite.DrawingSubscribers.Count
For Each objDrawingSubscriber In objDrawingSite.DrawingSubscribers
   Debug.Print objDrawingSubscriber.DrawingSiteObject.Attributes("Name").Value
   Debug.Print objDrawingSubscriber.WSSiteObject.Attributes("Name").Value
Next

Debug.Print objDrawingSite.DrawingObject.Attributes("Name").Value
Debug.Print objDrawingSite.WSSiteObject.Attributes("Name").Value
If Not objDrawingSite.ToWSSiteWSSiteObject Is Nothing Then
   Debug.Print objDrawingSite.ToWSSiteWSSiteObject.Attributes("Name").Value
End If
Debug.Print objDrawingSite.PlantGroupObject.Attributes("Name").Value

Set datasource = Nothing
Set objVessel = Nothing
Set objDrawing = Nothing
Set objDrawingSite = Nothing
Set objDrawingSubscriber = Nothing
Set objAttr = Nothing
```

60. WORKSHARE AWARENESS IN LLAMA

a) Purpose

To check out the workshare awareness in LLAMA.

b) Problem Statement

Set a satellite site as active project, then access a vessel in a drawing which is read-only for this satellite site, try to modify the property of the vessel and commit to database. See what happens?

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

datasource.BeginTransaction

Dim objVessel As LMVessel
Set objVessel = datasource.GetVessel(CONST_SPID_Vessel)
Debug.Print objVessel.Attributes("Name").Value
'expects error in following code, "Read Only attribute"
objVessel.Attributes("Name").Value = "InWorkshare"
objVessel.Commit

datasource.CommitTransaction

Set datasource = Nothing
Set objVessel = Nothing
```

61. ACCESS ACTIVE PROJECT

a) Purpose

To access the active project

b) Problem Statement

Set The Plant or one of projects as active project, then use LMADatasource.GetActiveProject to obtain the active project. Then, print out all attributions of the active project, pay attention to the Project Status.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource
Dim objActiveProject As LMActiveProject
Dim objAttribute As LMAAttribute

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Set objActiveProject = datasource.GetActiveProject
For Each objAttribute In objActiveProject.Attributes
   Debug.Print "Name: " & objAttribute.name & Space(20 - Len(objAttribute.name)) & "Value: "
& objAttribute.Value
Next

Set datasource = Nothing
Set objActiveProject = Nothing
Set objAttribute = Nothing
```

62. How to access Plant from Project

a) Purpose

When user is in a project, how to find the project belongs to which The Plant?

b) Problem Statement

Set one of the projects as active project, then try to find The Plant.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource
Dim objPlantGroups As LMPlantGroups
Dim objPlantGroup As LMPlantGroup
Dim objAttribute As LMAAttribute

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Set objPlantGroups = New LMPlantGroups
objPlantGroups.Collect datasource

Debug.Print "Total PlantGroups: " & objPlantGroups.Count

For Each objPlantGroup In objPlantGroups
   For Each objAttribute In objPlantGroup.Attributes
      Debug.Print "Name: " & objAttribute.name & Space(20 - Len(objAttribute.name)) & "Value:
" & objAttribute.Value
      If objAttribute.name = "Depth" And objAttribute.Value = "0" Then
         MsgBox "ThePlant is: " & objPlantGroup.Attributes("Name")
      End If
   Next
Next

Set datasource = Nothing
Set objPlantGroups = Nothing
Set objPlantGroup = Nothing
Set objAttribute = Nothing
```

63. Access Claim Status of Items

a) Purpose

To access the items' claim status.

b) Problem Statement

Set items in different claim status, and access claim status by using function LMADatasource.GetModelItemClaimStatus

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objVessel As LMVessel
Set objVessel = datasource.GetVessel(CONST_SPID_Vessel)   'get objVessel by id

'check Claim status
Debug.Print datasource.GetModelItemClaimStatus(objVessel.AsLMAItem)

Set datasource = Nothing
Set objVessel = Nothing
```

64. ACCESS OPTIONSETTINGS

a) Purpose

To access the OptionSetting by Filter, and read value of OptionSetting

b) Problem Statement

Write a standalone application to obtain optionsetting (Default Assembly Path) by filter and read the value of the optionsetting.

c) Solution

1. Dim LMAFilter, LMACriterion, LMOptionSetting
2. LMOptionSetting is a independent object, which does not has any relationship with other objects in Data Model. To access LMOptionSetting, users need to know exactly what they are looking for, for example, in optionsettings, where is the "Default Assembly Path" ?

◊ **Example code**

```
Dim datasource As LMADataSource

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Dim objFilter As LMAFilter
Dim criterion As LMACriterion
Set criterion = New LMACriterion
Set objFilter = New LMAFilter

criterion.SourceAttributeName = "Name"
criterion.ValueAttribute = "Default Assembly Path"
criterion.Operator = "="

objFilter.ItemType = "OptionSetting"
objFilter.Criteria.Add criterion

Dim objOptionSettings As LMOptionSettings
Dim objOptionSetting As LMOptionSetting
Set objOptionSettings = New LMOptionSettings
'get "Default Assembly Path" from OptionSettings by filter
objOptionSettings.Collect datasource, Filter:=objFilter

Set objOptionSetting = objOptionSettings.Nth(1)
Debug.Print "Name = " & objOptionSetting.Attributes("Name").Value
Debug.Print "Value = " & objOptionSetting.Attributes("Value").Value

Set datasource = Nothing
Set objFilter = Nothing
Set criterion = Nothing
Set objOptionSettings = Nothing
Set objOptionSetting = Nothing
```

65. CREATE A VESSEL AND PLACE INTO STOCKPILE

a) Purpose

Use PIDCreateItem method to create a vessel in the stockpile

b) Problem Statement

Write a standard executable to create a vessel and place it in the stockpile.

c) Solution

◊ **Open the SmartPlant P&ID drawing.**

1. Create a drawing through SPManager.
2. Double-click on the drawing to open up SmartPlant P&ID

◊ **Create a standard executable VB project**

3. Select a standard exe project
4. Reference the "Logical Model Automation" and "Placement Automation" libraries

◊ **Add code to place a vessel into stockpile**

5. Use the Function Function **PIDCreateItem**(DefinitionFile As String) As **LMAItem**
6. Provide the DefinitionFile string indicating the location of the symbol on a server
7. Use the return value to future reference.

◊ **Example code**

```
Dim objPlacement As Placement
Set objPlacement = New Placement

Dim item As LMAItem
Dim dirpath As String

Dim VesselLocation As String
VesselLocation = "\Equipment\Vessels\Horizontal Drums\Horz Surge w-Horiz Dea.sym"
'create a vessel into stockpile
Set item = objPlacement.PIDCreateItem(VesselLocation)
If item Is Nothing Then
   MsgBox "unsuccessful placement"
End If

Set objPlacement = Nothing
Set item = Nothing
```

66. PLACE A VESSEL ON A DRAWING

a) Purpose

Use the PidPlaceSymbol method to place a vessel on a drawing

b) Problem Statement

Write a standard executable to place a vessel on a drawing.

c) Solution

◊ **Open the SmartPlant P&ID drawing.**

1. Create a drawing through SPManager.
2. Double-click on the drawing to open up SmartPlant P&ID

◊ **Create a standard executable VB project**

3. Select a standard exe project
4. Reference the "Logical Model Automation" and "Placement Automation" libraries

◊ **Add code to place a vessel**

5. Use the method Function **PIDPlaceSymbol**(DefinitionFile As String, X As Double, Y As Double, [Mirror], [Rotation], [ExistingItem As LMAItem], [TargetItem]) As **LMSymbol**
6. Provide the DefinitionFile string indicating the location of the symbol on a server
7. Provide the X and Y coordinates of the placement on the drawing.
8. Use the return value to future reference.

◊ **Example code**

```
Dim objPlacement As Placement
Set objPlacement = New Placement

Dim dirpath As String

Dim symbol As LMSymbol
Dim VesselLocation As String
VesselLocation = "\Equipment\Vessels\Horizontal Drums\Horz Surge w-Horiz Dea.sym"

'place a vessle into active drawing
Set symbol = objPlacement.PIDPlaceSymbol(VesselLocation, 0.3, 0.2)

If symbol Is Nothing Then
   MsgBox "unsuccessful placement"
End If

Set objPlacement = Nothing
Set symbol = Nothing
```

67. PLACE NOZZLES AND TRAYS ON A VESSEL

a) Purpose

Use PIDPlaceSymbol method to place equipment components on a vessel

b) Problem Statement

Write a standard executable to place nozzles and trays on a vessel.

c) Solution

◊ **Open the SmartPlant P&ID drawing.**

1. Create a drawing through SPManager.
2. Double-click on the drawing to open up SmartPlant P&ID

◊ **Create a standard executable VB project**

3. Select a standard exe project
4. Reference the "Logical Model Automation" and "Placement Automation" libraries

◊ **Add code to place a vessel**

5. Use the Function **PIDPlaceSymbol**(DefinitionFile As String, X As Double, Y As Double, [Mirror], [Rotation], [ExistingItem As LMAItem], [TargetItem]) As **LMSymbol**
6. Provide the DefinitionFile string indicating the location of the symbol on a server
7. Provide the X and Y coordinates of the placement on the drawing.
8. Provide the TargetItem as an LMAItem.
9. Use the return value to future reference.
10. Repeat place nozzles while set **PIDSnapToTarget** to TRUE

◊ **Example code**

```
Dim objPlacement As Placement
Set objPlacement = New Placement

Dim strdirpath As String
Dim nozzleName As String
Dim trayName As String

Dim xvessel As Double
Dim yvessel As Double
xvessel = 0.3
yvessel = 0.2
Dim symVessel As LMSymbol
Dim symbol2 As LMSymbol
Dim symbol3 As LMSymbol
Dim symbol4 As LMSymbol
Dim symbol5 As LMSymbol
Dim vesselName As String
Dim symbol21 As LMSymbol
Dim symbol31 As LMSymbol

vesselName = "\Equipment\Vessels\Vertical Drums\1D 1C 2to1.sym"
nozzleName = "\Equipment Components\Nozzles\Flanged Nozzle with blind.sym"
trayName = "\Equipment Components\Trays\Bubble Cap Trays\2-Pass Bubl Side.sym"

'place a vessel
Set symVessel = objPlacement.PIDPlaceSymbol(vesselName, xvessel, yvessel)
'set Cleaning Requirement for the Vessel
```

```
    Dim objVessel As LMVessel
    Set objVessel = objPlacement.PIDDataSource.GetVessel(symVessel.ModelItemID)
    objVessel.Attributes("CleaningReqmts").Value = "CC1"
    'place two nozzles on vessel
    Set symbol2 = objPlacement.PIDPlaceSymbol(nozzleName, xvessel - 0.2, yvessel + 0.05, _
            TargetItem:=symVessel.AsLMRepresentation)
    Set symbol3 = objPlacement.PIDPlaceSymbol(nozzleName, xvessel + 0.2, yvessel + 0.07, _
            TargetItem:=symVessel.AsLMRepresentation)
    'place two trays on vessel
    Set symbol4 = objPlacement.PIDPlaceSymbol(trayName, xvessel - 0.05, yvessel + 0.05, _
            TargetItem:=symVessel.AsLMRepresentation)
    Set symbol5 = objPlacement.PIDPlaceSymbol(trayName, xvessel + 0.05, yvessel + 0.1, _
            TargetItem:=symVessel.AsLMRepresentation)

''''    'place nozzles again use same X, Y coordinates, but this time set PIDSnapToTarget=false
''''    objPlacement.PIDSnapToTarget = False
''''    Set symbol21 = objPlacement.PIDPlaceSymbol(nozzleName, xvessel - 0.2, yvessel + 0.05,
_
''''            TargetItem:=symVessel.AsLMRepresentation)
''''    Set symbol31 = objPlacement.PIDPlaceSymbol(nozzleName, xvessel + 0.2, yvessel + 0.07,
_
''''            TargetItem:=symVessel.AsLMRepresentation)
''''    objPlacement.PIDSnapToTarget = True

''''    'place nozzles again use new X, Y coordinates, but this time set
PIDSetCopyPropertiesFlag=false
''''    objPlacement.PIDSetCopyPropertiesFlag False
''''    Set symbol21 = objPlacement.PIDPlaceSymbol(nozzleName, xvessel - 0.2, yvessel + 0.05,
_
''''            TargetItem:=symVessel.AsLMRepresentation)
''''    Set symbol31 = objPlacement.PIDPlaceSymbol(nozzleName, xvessel + 0.2, yvessel + 0.07,
_
''''            TargetItem:=symVessel.AsLMRepresentation)
''''    objPlacement.PIDSetCopyPropertiesFlag True

    Set objPlacement = Nothing
    Set symVessel = Nothing
    Set symbol2 = Nothing
    Set symbol3 = Nothing
    Set symbol4 = Nothing
    Set symbol5 = Nothing
''''    Set symbol21 = Nothing
''''    Set symbol31 = Nothing
```

68. PLACE LABELS ON A VESSEL

a) Purpose

Use PIDPlaceLabel method to place labels on equipment

b) Problem Statement

Write a standard executable to populate some properties of a vessel and then place labels to display them.

c) Solution

◊ **Open the SmartPlant P&ID drawing.**

1. Create a drawing through SPManager.
2. Double-click on the drawing to open up SmartPlant P&ID

◊ **Create a standard executable VB project**

3. Select a standard exe project
4. Reference the "Logical Model Automation" and "Placement Automation" libraries

◊ **Add code to place a vessel**

5. Use the Function **PIDPlaceSymbol**(DefinitionFile As String, X As Double, Y As Double, [Mirror], [Rotation], [ExistingItem As LMAItem], [TargetItem]) As **LMSymbol**
6. Provide the DefinitionFile string indicating the location of the symbol on a server
7. Provide the X and Y coordinates of the placement on the drawing.
8. Provide the TargetItem as an LMAItem when placing nozzles or trays.
9. Use the return value to future reference.

◊ **Add code to delete the vessel**

10. Use the Function **PIDPlaceLabel**(DefinitionFile As String, Points() As Double, [Mirror], [Rotation], [LabeledItem As LMRepresentation], [IsLeaderVisible As Boolean = False]) As **LMLabelPersist**
11. The Points array consists of the exact number of points (starting from index 1) necessary to place the label.
12. The LMRepresentation argument must be a representation of the parent item on the drawing.
13. The return object is the label object.

◊ **Example code**

```
Dim objPlacement As Placement
Set objPlacement = New Placement

Dim strdirpath As String
Dim nozzleName As String
Dim trayName As String
Dim labelName1 As String
Dim labelName2 As String
Dim labelName3 As String
Dim vessel As LMVessel
Dim labelpersist As LMLabelPersist

Dim xvessel As Double
Dim yvessel As Double
xvessel = 0.2
yvessel = 0.2

Dim symVessel As LMSymbol
```

```
Dim vesselName As String
Dim points(1 To 4) As Double
Dim twopoints(1 To 2) As Double

vesselName = "\Equipment\Vessels\Vertical Drums\1D 1C 2to1.sym"
labelName1 = "\Equipment\Labels - Equipment\Equipment Name.sym"
labelName2 = "\Equipment\Labels - Equipment\Insulation Purpose.sym"
labelName3 = "\Equipment\Labels - Equipment\Heat Tracing.sym"

'place vessel
Set symVessel = objPlacement.PIDPlaceSymbol(vesselName, xvessel, yvessel)
'get placed vessel and set some properties' value
Set vessel = objPlacement.PIDDataSource.GetVessel(symVessel.ModelItemID)
vessel.name = "Vessel for Label Placement"
vessel.InsulPurpose = "R15"
vessel.HTraceMedium = "SS"
vessel.HTraceMediumTemp = "300 F"
vessel.HTraceReqmt = "ET"

'place three different labels for the vessel
twopoints(1) = xvessel + 0.02
twopoints(2) = yvessel + 0.1
Set labelpersist = objPlacement.PIDPlaceLabel(labelName1, _
    twopoints, Labeleditem:=symVessel.AsLMRepresentation)
points(1) = xvessel
points(2) = yvessel
points(3) = xvessel - 0.05
points(4) = yvessel + 0.1
Set labelpersist = objPlacement.PIDPlaceLabel(labelName2, _
   points(), Labeleditem:=symVessel.AsLMRepresentation, _
   IsLeaderVisible:=True)

points(1) = xvessel
points(2) = yvessel
points(3) = xvessel + 0.05
points(4) = yvessel + 0.1
Set labelpersist = objPlacement.PIDPlaceLabel(labelName3, _
   points(), Labeleditem:=symVessel.AsLMRepresentation, _
  IsLeaderVisible:=True)

Set objPlacement = Nothing
Set symVessel = Nothing
Set labelpersist = Nothing
```

69. PLACE OPC

a) Purpose

Use PIDPlaceOPC method to place an OPC into drawing.

b) Problem Statement

Write a standard executable to place an OPC into drawing.

c) Solution

◊ **Example code**

```
Dim objPlacement As Placement
Set objPlacement = New Placement

Dim OPClocation As String
OPClocation = "\Piping\Piping OPC's\Off-Drawing.sym"

'place a vessle into active drawing
Dim symbol As LMSymbol
Set symbol = objPlacement.PIDPlaceSymbol(OPClocation, 0.1, 0.1)

If symbol Is Nothing Then
   MsgBox "unsuccessful placement"
End If

Set objPlacement = Nothing
Set symbol = Nothing
```

70. PLACE OPC FROM STOCKPILE

a) Purpose

Use PIDPlaceOPC method to place an OPC from StockPile into drawing.

b) Problem Statement

Place an OPC into a drawing, and place its pair OPC in plant stockpile, then open another drawing with a piperun placed, then write a standard executable to find the OPC, then find its pair OPC in StockPile, then place it pair OPC from StockPile into current drawing, and connect with the piperun.

c) Solution

◊ **Example code**

```
Dim objPlacement As Placement
Set objPlacement = New Placement

Dim OPClocation As String
OPClocation = "\Piping\Piping OPC's\Off-Drawing.sym"

Dim objOPC As LMOPC
Dim objpairOPC As LMOPC

Set objOPC = objPlacement.PIDDataSource.GetOPC(CONST_SPID_OPC)
Set objpairOPC = objOPC.pairedWithOPCObject

Dim objConnector As LMConnector
Dim objPiperun As LMPipeRun
Dim objRep As LMRepresentation

Set objPiperun = objPlacement.PIDDataSource.GetPipeRun(CONST_SPID_PipeRun)
For Each objRep In objPiperun.Representations
   If objRep.Attributes("RepresentationType").Value = "Connector" Then
      Set objConnector = objPlacement.PIDDataSource.GetConnector(objRep.ID)
      Exit For
   End If
Next

Dim X As Double
Dim Y As Double

X = objConnector.ConnectorVertices.Nth(1).Attributes("XCoordinate").Value
Y = objConnector.ConnectorVertices.Nth(1).Attributes("YCoordinate").Value
'place the OPC from stockpile into active drawing
Dim symbol As LMSymbol
Set symbol = objPlacement.PIDPlaceSymbol(OPClocation, X, Y, , , objpairOPC.AsLMAItem)

If symbol Is Nothing Then
   MsgBox "unsuccessful placement"
End If

Set objPlacement = Nothing
Set symbol = Nothing
```

71. PLACE PIPERUN WITH PIDPLACERUN

a) Purpose

Use PIDPlaceRun method to place a Piperun from stockpile into active drawing

b) Problem Statement

Write a standalone application to create a piperun in stockpile, then place this piperun from stockpile into active drawing. Then place a valve, and place a piperun connects first piperun and the valve.

c) Solution

◊ **Example code**

```
Dim objPlacement As Placement
Set objPlacement = New Placement

Dim PipeRunLocation As String
Dim objItem As LMAItem
Dim objConnector As LMConnector
Dim objInputs As PlaceRunInputs
Dim objSymbol As LMSymbol
Dim ValveLocation As String

PipeRunLocation = "\Piping\Routing\Process Lines\Primary Piping.sym"

Set objInputs = New PlaceRunInputs
objInputs.AddPoint 0.1, 0.1
objInputs.AddPoint 0.2, 0.1

Set objItem = objPlacement.PIDCreateItem(PipeRunLocation)
Set objConnector = objPlacement.PIDPlaceRun(objItem, objInputs)

ValveLocation = "\Piping\Valves\2 Way Common\Ball Valve.sym"
Set objSymbol = objPlacement.PIDPlaceSymbol(ValveLocation, 0.15, 0.3, , 1.57)

Set objInputs = New PlaceRunInputs
objInputs.AddConnectorTarget objConnector, 0.15, 0.1
objInputs.AddPoint 0.15, 0.15
objInputs.AddPoint 0.12, 0.15
objInputs.AddPoint 0.12, 0.2
objInputs.AddPoint 0.15, 0.2
objInputs.AddSymbolTarget objSymbol, 0.15, 0.3

Set objItem = objPlacement.PIDCreateItem(PipeRunLocation)
Set objConnector = objPlacement.PIDPlaceRun(objItem, objInputs)
'clean up
Set objPlacement = Nothing
Set objItem = Nothing
Set objConnector = Nothing
Set objSymbol = Nothing
Set objInputs = Nothing
```

72. JOIN TWO PIPERUNS

a) Purpose

Use PIDAutoJoin to auto join two piperuns

b) Problem Statement

Write a standalone application to createa piperun in stockpile, then place this piperun from stockpile into active drawing. Then place another piperun from middle of first piperun to have an end open, then place a vessel with a nozzle, then place a new piperun connects nozzle and second piperun, then use PIDAutoJoin to join second and third piperuns.

c) Solution

◊ **Example code**

```
Dim objPlacement As Placement
Set objPlacement = New Placement

Dim PipeRunLocation As String
Dim objItem As LMAItem
Dim objConnector As LMConnector
Dim objInputs As PlaceRunInputs
Dim objSymbol As LMSymbol
Dim VesselLocation As String
Dim NozzleLocation As String
Dim objPiperuns As LMPipeRuns
Dim objPiperun As LMPipeRun
Dim objSurvivorItem As LMAItem

Set objPiperuns = New LMPipeRuns

PipeRunLocation = "\Piping\Routing\Process Lines\Primary Piping.sym"

Set objInputs = New PlaceRunInputs
objInputs.AddPoint 0.1, 0.1
objInputs.AddPoint 0.2, 0.1

'first piperun
Set objItem = objPlacement.PIDCreateItem(PipeRunLocation)
Set objConnector = objPlacement.PIDPlaceRun(objItem, objInputs)

Set objInputs = New PlaceRunInputs
objInputs.AddLocatedTarget 0.15, 0.1
objInputs.AddPoint 0.15, 0.3

'second piperun
Set objItem = objPlacement.PIDCreateItem(PipeRunLocation)
Set objConnector = objPlacement.PIDPlaceRun(objItem, objInputs)
objPiperuns.Add objPlacement.PIDDataSource.GetPipeRun(objConnector.ModelItemID)

VesselLocation = "\Equipment\Vessels\Vertical Drums\1D 1C 2to1.sym"
NozzleLocation = "\Equipment Components\Nozzles\Flanged Nozzle with blind.sym"

Set objSymbol = objPlacement.PIDPlaceSymbol(VesselLocation, 0.15, 0.5)
```

```
    Set objSymbol = objPlacement.PIDPlaceSymbol(NozzleLocation, 0.15, 0.5 - 0.1,
TargetItem:=objSymbol.AsLMRepresentation)

    Set objInputs = New PlaceRunInputs
    objInputs.AddConnectorTarget objConnector, 0.15, 0.3
    objInputs.AddSymbolTarget objSymbol, objSymbol.Attributes("XCoordinate"),
objSymbol.Attributes("YCoordinate")

    'third piperun
    Set objItem = objPlacement.PIDCreateItem(PipeRunLocation)
    Set objConnector = objPlacement.PIDPlaceRun(objItem, objInputs)
    objPiperuns.Add objPlacement.PIDDataSource.GetPipeRun(objConnector.ModelItemID)

    'AutoJoin
    For Each objPiperun In objPiperuns
       objPlacement.PIDAutoJoin objPiperun.AsLMAItem, autoJoin_Both, objSurvivorItem
    Next

    MsgBox "Done!"
'clean up
    Set objPlacement = Nothing
    Set objItem = Nothing
    Set objConnector = Nothing
    Set objSymbol = Nothing
    Set objInputs = Nothing
```

73. PLACE GAP

a) Purpose

Use PIDPlaceGap method to place a Gap.

b) Problem Statement

Write a standalone application to place Connector, then place a Gap in the middle of the connector

c) Solution

1. PIDPlaceGap returns a LMSymbol object, whose RepresentationType is "GAP"

◊ **Example code**

```
Dim objPlacement As Placement
Set objPlacement = New Placement

Dim PipeRunLocation As String
PipeRunLocation = "\Piping\Routing\Process Lines\Primary Piping.sym"

Dim twopoints(1 To 4) As Double
twopoints(1) = 0.2
twopoints(2) = 0.2
twopoints(3) = 0.4
twopoints(4) = 0.2

Dim objConnector As LMConnector
Set objConnector = objPlacement.PIDPlaceConnector(PipeRunLocation, twopoints)


Dim gaplocation As String
gaplocation = "\Piping\Gaps\gap-lines.sym"
Dim objSymbol As LMSymbol
Set objSymbol = objPlacement.PIDPlaceGap(gaplocation, 0.3, 0.2, 0.02, 0.02, objConnector, -
1.57)

Set objConnector = Nothing
Set objSymbol = Nothing
Set objPlacement = Nothing
```

74. PLACE BOUNDEDSHAPE

a) Purpose

Use PIDPlaceBoundedShape method to place a BoundedShape (AreaBreak).

b) Problem Statement

Write a standalone application to place a BoundedShape (AreaBreak) and a vessel with nozzle. Add a vessel and assign it to be a part of the AreaBreak

c) Solution

1. PIDPlaceBoundedShape places a visual BoundedShape aroung the items rather than estbilish the relationship between BoundedShape and items inside of it.

◊ **Example code**

```
Dim objPlacement As Placement
Set objPlacement = New Placement

Dim datasource As LMADataSource
Set datasource = objPlacement.PIDDataSource

Dim VesselLocation As String
VesselLocation = "\Equipment\Vessels\Vertical Drums\1D 1C 2to1.sym"

'place a vessel
Dim objSymbol1 As LMSymbol
Set objSymbol1 = objPlacement.PIDPlaceSymbol(VesselLocation, 0.25, 0.25)

'place a nozzle on the vessel
Dim NozzleLocation As String
NozzleLocation = "\Equipment Components\Nozzles\Flanged Nozzle with blind.sym"
Dim objSymbol2 As LMSymbol
Set objSymbol2 = objPlacement.PIDPlaceSymbol(NozzleLocation, 0#, 0#, , , ,
objSymbol1.AsLMRepresentation)

Dim points(1 To 10) As Double
points(1) = 0.1
points(2) = 0.1
points(3) = 0.4
points(4) = 0.1
points(5) = 0.4
points(6) = 0.4
points(7) = 0.1
points(8) = 0.4
points(9) = 0.1
points(10) = 0.1

'place BoundedShape(it is a AreaBreak)
Dim boundedshapelocation As String
boundedshapelocation = "\Design\Area Break.sym"
Dim objBoundedShaped As LMBoundedShape
Set objBoundedShaped = objPlacement.PIDPlaceBoundedShape(boundedshapelocation,
points)
```

```
    'get the placed vessel
    Dim objVessel As LMVessel
    Set objVessel = datasource.GetVessel(objSymbol1.ModelItemID)

    'get the BoundedShpae(AreaBreak) as PlantItemGroup
    Dim objPlantItemGroup As LMPlantItemGroup
    Set objPlantItemGroup = datasource.GetPlantItemGroup(objBoundedShaped.ModelItemID)

    Debug.Print "The vessel belongs to how many plantitemgroups? = " &
objVessel.PlantItemGroups.Count

    'add the vessel to the PlantItemGroup
    objVessel.PlantItemGroups.Add objPlantItemGroup
    'objVessel.Commit
    Debug.Print "The vessel belongs to how many plantitemgroups? = " &
objVessel.PlantItemGroups.Count

    Set objPlacement = Nothing
    Set datasource = Nothing
```

75. PLACE ASSEMBLY

a) Purpose

Use PIDPlaceAssembly method to place assembly into drawing

b) Problem Statement

Create an assembly using the SmartPlant P&ID modeler. Write a standalone application to place an assembly into drawing.

c) Solution

If the Assembly's source is in a location that is not accessible, change the source to current machine first

◊ **Example code**

```
Dim objPlacement As Placement
Set objPlacement = New Placement

Dim assemblylocation As String
assemblylocation = "\Assemblies\test.pid"

Dim objItems As LMAItems
'place assembly
Set objItems = objPlacement.PIDPlaceAssembly(assemblylocation, 0#, 0#)
If objItems.Count <> 0 Then
   MsgBox "Place Assembly Completed"
End If

Set objPlacement = Nothing
Set objItems = Nothing
```

76. DELETE VESSEL FROM DRAWING

a) Purpose

Use PIDRemovePlacement method to delete vessel from drawing

b) Problem Statement

Write a standard executable to delete vessel from the drawing.

c) Solution

◊ **Open the SmartPlant P&ID drawing.**

1. Create a drawing through SPManager.
2. Double-click on the drawing to open up SmartPlant P&ID

◊ **Create a standard executable VB project**

3. Select a standard exe project
4. Reference the "Logical Model Automation" and "Placement Automation" libraries

◊ **Add code to delete the vessel from drawing**

5. Use the Function **PIDRemovePlacement**(Representation As LMRepresentation) As **Boolean**
6. The LMRepresentation argument must be a representation of the item on the drawing.
7. The boolean return value can be stored to determine success or failure.

◊ **Example code**

```
Dim objPlacement As Placement
Set objPlacement = New Placement

Dim objRep As LMRepresentation
Dim objVessel As LMVessel
Dim objSym As LMSymbol
Dim VesselLocation As String

VesselLocation = "\Equipment\Vessels\Horizontal Drums\Horz Surge w-Horiz Dea.sym"
'place a vessel into drawing
Set objSym = objPlacement.PIDPlaceSymbol(VesselLocation, 0.2, 0.2)

Set objVessel = objPlacement.PIDDataSource.GetVessel(objSym.ModelItemID)
Set objRep = objVessel.Representations.Nth(1)
'remove the vessel from drawing into stockpile
Dim success As Boolean
success = objPlacement.PIDRemovePlacement(objRep)
If success Then
   MsgBox "Symbol removed successfully"
Else
   MsgBox "RemovePlacement unsuccessful"
End If

Set objPlacement = Nothing
Set objVessel = Nothing
Set objRep = Nothing
```

77. DELETE VESSEL FROM MODEL

a) Purpose

Use PIDDeleteItem method to delete vessal from the project

b) Problem Statement

Write a standard executable to delete a vessel from project.

c) Solution

◊ **Open the SmartPlant P&ID drawing.**

8. Create a drawing through SPManager.
9. Double-click on the drawing to open up SmartPlant P&ID

◊ **Create a standard executable VB project**

10. Select a standard exe project
11. Reference the "Logical Model Automation" and "Placement Automation" libraries

◊ **Add code to delete the vessel from model**

12. Use the Function **PIDDeleteItem**(Item As LMAItem) As **Boolean** to remove from model
13. The LMRepresentation argument must be a representation of the item on the drawing.
14. The boolean return value can be stored to determine success or failure.

◊ **Example code**

```
Dim objPlacement As Placement
Set objPlacement = New Placement

Dim objItem As LMAItem
Dim success As Boolean
Dim objSym As LMSymbol
Dim VesselLocation As String

VesselLocation = "\Equipment\Vessels\Horizontal Drums\Horz Surge w-Horiz Dea.sym"
'place a vessel into drawing
Set objSym = objPlacement.PIDPlaceSymbol(VesselLocation, 0.2, 0.2)

Set objItem = objPlacement.PIDDataSource.GetVessel(objSym.ModelItemID).AsLMAItem
success = False
success = objPlacement.PIDDeleteItem(objItem)
If success Then
   MsgBox "deletion from drawing successfully"
Else
   MsgBox "deletion from drawing unsuccessful"
End If

Dim item As LMAItem
Set item = objPlacement.PIDCreateItem(VesselLocation)
Set objItem = objPlacement.PIDDataSource.GetVessel(item.ID).AsLMAItem
success = False
success = objPlacement.PIDDeleteItem(objItem)
If success Then
   MsgBox "deletion from stockpile successfully"
```

```
Else
   MsgBox "deletion from stockpile unsuccessful"
End If

Set objPlacement = Nothing
Set objItem = Nothing
```

78. REPLACE SYMBOL

a) Purpose

Use PIDRePlaceSymbol method to replace a vessel.

b) Problem Statement

Write a standalone application to place a vessel with a nozzle on it. Replace the vessel with different vessel. Note that the vessel is replaced and the nozzle is now on the new vessel.

c) Solution

◊ **Example code**

```
Dim objPlacement As Placement
Set objPlacement = New Placement

Dim VesselLocation As String
VesselLocation = "\Equipment\Vessels\Vertical Drums\1D 1C 2to1.sym"
'place a vessel
Dim objSymbol1 As LMSymbol
Set objSymbol1 = objPlacement.PIDPlaceSymbol(VesselLocation, 0.2, 0.2)

Dim NozzleLocation As String
NozzleLocation = "\Equipment Components\Nozzles\Flanged Nozzle with blind.sym"
'place a nozzle on the vessel
Dim objSymbol2 As LMSymbol
Set objSymbol2 = objPlacement.PIDPlaceSymbol(NozzleLocation, 0.3, 0.2, , , ,
objSymbol1.AsLMRepresentation)

'replace the vessel, note nozzle is still on the new vessel
Dim replacevesselname As String
replacevesselname = "\Equipment\Vessels\Vertical Drums\2to1Parametric V Drum.sym"

Dim objSymbol3 As LMSymbol
Set objSymbol3 = objPlacement.PIDReplaceSymbol(replacevesselname, objSymbol1)


Set objSymbol1 = Nothing
Set objSymbol2 = Nothing
Set objSymbol3 = Nothing
Set objPlacement = Nothing
```

79. REPLACE LABEL

a) Purpose

Use PIDRePlaceLabel method to replace a label.

b) Problem Statement

Write a standalone application to place a vessel with a nozzle on it, then replace the vessel with different vessel. Note vessel is replaced with nozzle now is sitting on the new vessel.

c) Solution

◊ **Example code**

```
Dim objPlacement As Placement
Set objPlacement = New Placement

Dim VesselLocation As String
VesselLocation = "\Equipment\Vessels\Vertical Drums\2to1Parametric V Drum.sym"
'place a vessel
Dim objSymbol1 As LMSymbol
Set objSymbol1 = objPlacement.PIDPlaceSymbol(VesselLocation, 0.2, 0.2)

'get the vessel and set some properties' value
Dim objVessel As LMVessel
Set objVessel = objPlacement.PIDDataSource.GetVessel(objSymbol1.ModelItemID)
objVessel.Attributes("Name").Value = "V1"
objVessel.Attributes("TagPrefix").Value = "T"
objVessel.Commit

Dim labelName1 As String
labelName1 = "\Equipment\Labels - Equipment\Equipment Name.sym"
Dim twopoints(1 To 4) As Double
twopoints(1) = 0.21
twopoints(2) = 0.25
twopoints(3) = 0.1
twopoints(4) = 0.1

'place a label on the vessel
Dim objLabelPersist1 As LMLabelPersist
Set objLabelPersist1 = objPlacement.PIDPlaceLabel(labelName1, twopoints, , ,
objSymbol1.AsLMRepresentation, True)

Dim replacelabelname As String
replacelabelname = "\Equipment\Labels - Equipment\Equipment ID.sym"

'replace the label with new label
Dim objLabelPersist2 As LMLabelPersist
Set objLabelPersist2 = objPlacement.PIDReplaceLabel(replacelabelname, objLabelPersist1)

Set objSymbol1 = Nothing
Set objLabelPersist1 = Nothing
Set objLabelPersist2 = Nothing
Set objPlacement = Nothing
```

80. REPLACE OPC

a) Purpose

Use PIDRePlaceOPC method to replace an OPC.

b) Problem Statement

Write a standalone application to replace an OPC on draiwng.

c) Solution

** PIDRePlaceOPC Not Working Now, using PIDReplaceSymbol instead

◊ **Example code**

```
Dim objPlacement As Placement
Set objPlacement = New Placement

Dim OPClocation As String
OPClocation = "\Piping\Piping OPC's\Off-Drawing-New.sym"

Dim objOPC As LMOPC

Set objOPC =
objPlacement.PIDDataSource.GetOPC("28B79FF6B52047DB98600BE313648290")

Dim objSymbol As LMSymbol
Set objSymbol =
objPlacement.PIDDataSource.GetSymbol(objOPC.Representations.Nth(1).ID)

Dim objSymbol1 As LMSymbol
Set objSymbol1 = objPlacement.PIDReplaceSymbol(OPClocation, objSymbol)

Set objSymbol = Nothing
Set objSymbol1 = Nothing
Set objOPC = Nothing
Set objPlacement = Nothing
```

81. MODIFY PARAMETRICSYMBOL

a) Purpose

Use PIDApplyParameters method to modifty a Parametric Symbol

b) Problem Statement

Write a standalone application to place a parametric vessel symbol, and place a nozzle on it. Then, Modifies the parameters of the vessel.

c) Solution

1. Names() are the Variables defined in Catalog Manager for the parametric symbol

◊ **Example code**

```
Dim objPlacement As Placement
Set objPlacement = New Placement

Dim VesselLocation As String
VesselLocation = "\Equipment\Vessels\Vertical Drums\2to1Parametric V Drum.sym"
Dim objSymbol1 As LMSymbol
Set objSymbol1 = objPlacement.PIDPlaceSymbol(VesselLocation, 0.2, 0.2, True, 1.57)

Dim NozzleLocation As String
NozzleLocation = "\Equipment Components\Nozzles\Flanged Nozzle with blind.sym"
Dim objSymbol2 As LMSymbol
Set objSymbol2 = objPlacement.PIDPlaceSymbol(NozzleLocation, 0.22, 0.4, , , ,
objSymbol1.AsLMRepresentation)

Dim names(1 To 2) As String
Dim values(1 To 2) As String
names(1) = "Top"
names(2) = "Right"
values(1) = "0.38"
values(2) = "0.2"

objPlacement.PIDApplyParameters objSymbol1.AsLMRepresentation, names, values

Set objSymbol1 = Nothing
Set objSymbol2 = Nothing
Set objPlacement = Nothing
```

82. Locate X, Y Coordinates of Signal Points on an Instrument

a) Purpose

Using PIDConnectPointLocation to locate X, Y coordinates of signal points on an instrument.

b) Problem Statement

Place an off-line instrument, connect a signal line to signal point on the instrument with index as 3.

c) Solution

Using PIDConnectPointLocation to find out X, Y coordinates first.

◊ **Example code**

```
Dim objPlacement As Placement
Set objPlacement = New Placement

Dim SignalRunLocation As String
Dim objItem As LMAItem
Dim objConnector As LMConnector
Dim objInputs As PlaceRunInputs
Dim objSymbol As LMSymbol
Dim InstrLocation As String
Dim blnSuccess As Boolean
Dim X As Double, Y As Double

SignalRunLocation = "\Instrumentation\Signal Line\Electric Binary.sym"


InstrLocation = "\Instrumentation\Off-Line\With Implied Components\Level\Discr Field Mounted
LC.sym"
Set objSymbol = objPlacement.PIDPlaceSymbol(InstrLocation, 0.3, 0.3)

blnSuccess = objPlacement.PIDConnectPointLocation(objSymbol, 3, X, Y)

Set objInputs = New PlaceRunInputs
objInputs.AddPoint 0.2, 0.3
objInputs.AddSymbolTarget objSymbol, X, Y

Set objItem = objPlacement.PIDCreateItem(SignalRunLocation)
Set objConnector = objPlacement.PIDPlaceRun(objItem, objInputs)

'clean up
Set objPlacement = Nothing
Set objItem = Nothing
Set objConnector = Nothing
Set objSymbol = Nothing
Set objInputs = Nothing
```

83. PLACE INSTRUMENT LOOP

a) Purpose

Use PIDCreateItem method to place an Instrument Loop in stockpile.

b) Problem Statement

Write a standalone application to place an Instrument Loop in stockpile and place a Piperun into drawing, then associate the Instrument Loop with the Piperun.

c) Solution

1. Known problem, the association won't be successful.

◊ **Example code**

```
Dim objPlacement As Placement
Set objPlacement = New Placement

Dim datasource As LMADataSource
Set datasource = objPlacement.PIDDataSource
Dim InstrumentLocation As String
InstrumentLocation = "\Instrumentation\Off-Line\With Implied Components\Pressure\Discr Field
Mounted PC.sym"
Dim objInstrSym As LMSymbol
Set objInstrSym = objPlacement.PIDPlaceSymbol(InstrumentLocation, 0.2, 0.2)

'get the placed instrument
Dim objInstr As LMInstrument
Set objInstr = datasource.GetInstrument(objInstrSym.ModelItemID)

'place an InstrLoop into stockpile
Dim InstrLoopLocation As String
InstrLoopLocation = "\Instrumentation\Loops\Pressure Loop.sym"
Dim objItem As LMAItem
Set objItem = objPlacement.PIDCreateItem(InstrLoopLocation)

Dim objInstrLoop As LMInstrLoop
Set objInstrLoop = datasource.GetInstrLoop(objItem.ID)
objInstrLoop.Attributes("TagSuffix").Value = "P"
objInstrLoop.Commit

Set objInstrLoop = datasource.GetInstrLoop(objItem.ID)
Debug.Print objInstrLoop.Attributes("ItemTag").Value

Debug.Print "The instrument belongs to how many plantitemgroups? = " &
objInstr.PlantItemGroups.Count
objInstr.PlantItemGroups.Add objInstrLoop.AsLMPlantItemGroup
objInstr.Commit
Debug.Print "The instrument belongs to how many plantitemgroups? = " &
objInstr.PlantItemGroups.Count

Set objPlacement = Nothing
Set datasource = Nothing
Set objItem = Nothing
Set objInstrLoop = Nothing
```

84. FIND AND REPLACE LABELS

a) Purpose

Comprehensive lab to practice filter for labels, and delete existing labels, then place new labels at the same X, Y Coordinates.

b) Problem Statement

Get collection of labels in the database, then loop through each label, and delete "\Piping\Segment Breaks\Construction Responsibility.sym" label, and place a new "\Piping\Segment Breaks\Construction Status.sym" label at the same X, Y Coordinate.

c) Solution

◊ **Example code**

```
Dim objPlacement As Plaice.Placement
Dim datasource As LMADataSource
Dim objFilter As LMAFilter
Dim objLabelPersists As LMLabelPersists
Dim objLabelPersist As LMLabelPersist
Dim objNewLabelPersist As LMLabelPersist
Dim X As Double, Y As Double
Dim objNewLabel As LMLabelPersist
Dim strFileName As String
Dim Points(1 To 4) As Double
Dim blnSuccess As Boolean
Dim strOLDFileName As String

Set objPlacement = New Plaice.Placement
Set datasource = objPlacement.PIDDataSource


Set objFilter = New LMAFilter

objFilter.Criteria.AddNew ("FirstOne")
objFilter.Criteria.item("FirstOne").SourceAttributeName = "ItemStatus"
objFilter.Criteria.item("FirstOne").ValueAttribute = 1
objFilter.Criteria.item("FirstOne").Operator = "="
objFilter.ItemType = "Vessel"


objFilter.Criteria.AddNew ("SecondOne")
objFilter.Criteria.item("SecondOne").SourceAttributeName = "SP_DrawingID"
objFilter.Criteria.item("SecondOne").ValueAttribute =
objPlacement.PIDDataSource.PIDMgr.Drawing.ID
objFilter.Criteria.item("SecondOne").Operator = "="
objFilter.Criteria.item("SecondOne").Conjunctive = True

objFilter.ItemType = "LabelPersist"

Set objLabelPersists = New LMLabelPersists

objLabelPersists.Collect datasource, Filter:=objFilter

strOLDFileName = "\Piping\Segment Breaks\Construction Responsibility.sym"
```

```
    strFileName = "\Piping\Segment Breaks\Construction Status.sym"
    For Each objLabelPersist In objLabelPersists
        If VBA.StrComp(objLabelPersist.Attributes("FileName").Value, strOLDFileName,
vbTextCompare) = 0 Then
            Points(1) = objLabelPersist.LeaderVertices.Nth(1).Attributes("XCoordinate").Value
            Points(2) = objLabelPersist.LeaderVertices.Nth(1).Attributes("YCoordinate").Value
            Points(3) = objLabelPersist.Attributes("XCoordinate").Value
            Points(4) = objLabelPersist.Attributes("YCoordinate").Value
            blnSuccess = False
            blnSuccess = objPlacement.PIDRemovePlacement(objLabelPersist.AsLMRepresentation)
            If blnSuccess Then
                Set objNewLabel = objPlacement.PIDPlaceLabel(strFileName, Points,
IsLeaderVisible:=True)
            End If
        End If
    Next

    MsgBox "Done!"

    Set objFilter = Nothing
    Set objNewLabel = Nothing
    Set objLabelPersist = Nothing
    Set objLabelPersists = Nothing
    Set datasource = Nothing
    Set objPlacement = Nothing
```

85. OPEN AND CLOSE AN EXISTING DRAWING

a) Purpose

Using PIDAutomation to open and close an existing drawing.

b) Problem Statement

Get collection of all drawings in the database, then loop through each drawing, open and close each drawing.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource
Dim objPIDAutoApp As PIDAutomation.Application
Dim objPIDADrawing As PIDAutomation.Drawing
Dim objDrawing As LMDrawing
Dim objDrawings As LMDrawings

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Set objDrawings = New LMDrawings
objDrawings.Collect datasource

Set objPIDAutoApp = CreateObject("PIDAutomation.Application")

For Each objDrawing In objDrawings
   If objDrawing.Attributes("ItemStatus").Index = 1 Then '1 stands for Active
      Set objPIDADrawing =
objPIDAutoApp.Drawings.OpenDrawing(objDrawing.Attributes("Name"))
      If Not objPIDADrawing Is Nothing Then
         MsgBox "Drawing " & objDrawing.Attributes("Name").Value & " is opened!"
         objPIDADrawing.CloseDrawing True
      End If
   End If
Next

objPIDAutoApp.Quit
Set objPIDAutoApp = Nothing
Set objPIDADrawing = Nothing
Set objDrawing = Nothing
Set objDrawings = Nothing
```

86. CREATE , OPEN AND CLOSE A NEW DRAWING

a) Purpose

Using PIDAutomation to create, open and close a new drawing.

b) Problem Statement

Create, open and close a new drawing until one of your Units.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource
Dim objPIDAutoApp As PIDAutomation.Application
Dim objPIDADrawing As PIDAutomation.Drawing
Dim PlantGroupName As String
Dim TemplateFileName As String
Dim DrawingNumber As String
Dim DrawingName As String

If Not blnUsePIDDatasource Then
   Set datasource = New LMADataSource
Else
   Set datasource = PIDDataSource
End If

Set objPIDAutoApp = CreateObject("PIDAutomation.Application")

PlantGroupName = "Unit01"
'considering accessing T_OptinSetting to read the template files path, which will be more
flexible
TemplateFileName = "\\blin\hostsite\lin_plant4\P&ID Reference Data\template files\C-Size.pid"
DrawingNumber = "TestCreateNewDrawing1"
DrawingName = "TestCreateNewDrawing1"
Set objPIDADrawing = objPIDAutoApp.Drawings.Add(PlantGroupName, TemplateFileName,
DrawingNumber, DrawingName)
If Not objPIDADrawing Is Nothing Then
   MsgBox "Drawing " & objPIDADrawing.name & " is opened!"
   objPIDADrawing.CloseDrawing True
End If

objPIDAutoApp.Quit
Set objPIDAutoApp = Nothing
Set objPIDADrawing = Nothing
```

87. COMPREHENSIVE AUTOMATION LAB

a) Purpose

To practice a comprehensive auatomation lab, including LLAMA, Placement and PIDAutomation.

b) Problem Statement

Write a standalone application to create a new drawing, then place an assembly into the drawing, then modify the piperuns placed by the assembly, set TagSequenceNo to 100. Then, close the drawing.

c) Solution

◊ **Example code**

```
Dim datasource As LMADataSource
Dim objPIDAutoApp As PIDAutomation.Application
Dim objPIDADrawing As PIDAutomation.Drawing
Dim PlantGroupName As String
Dim TemplateFileName As String
Dim DrawingNumber As String
Dim DrawingName As String
Dim objPlacement As Placement
Dim AssemblyLocation As String
Dim objItems As LMAItems
Dim objItem As LMAItem
Dim objConnector As LMConnector
Dim objPiperun As LMPipeRun

Set objPIDAutoApp = CreateObject("PIDAutomation.Application")

PlantGroupName = "Unit01"
'considering accessing T_OptinSetting to read the template files path, which will be more
flexible
TemplateFileName = "\\blin\hostsite\lin_plant3\P&ID Reference Data\template files\E-Size.pid"
DrawingNumber = "TestCreateNewDrawing2"
DrawingName = "TestCreateNewDrawing2"
Set objPIDADrawing = objPIDAutoApp.Drawings.Add(PlantGroupName, TemplateFileName,
DrawingNumber, DrawingName)
If Not objPIDADrawing Is Nothing Then
   Set objPlacement = New Placement
   Set datasource = objPlacement.PIDDataSource
   AssemblyLocation = "\Assemblies\Automation.pid"
   'place assembly
   Set objItems = objPlacement.PIDPlaceAssembly(AssemblyLocation, 0.2, 0.2)
   'change TagSequenceNo
   For Each objItem In objItems
      If objItem.ItemType = "Connector" Then
         Set objConnector = datasource.GetConnector(objItem.ID)
         If objConnector.ModelItemObject.AsLMAItem.ItemType = "PipeRun" Then
            Set objPiperun = datasource.GetPipeRun(objConnector.ModelItemID)
            objPiperun.Attributes("TagSequenceNo").Value = 100
            objPiperun.Commit
         End If
      End If
```

```
    Next
    objPIDADrawing.CloseDrawing True
End If

objPIDAutoApp.Quit
Set objPIDAutoApp = Nothing
Set objPIDADrawing = Nothing
```

88. CREATE A CALCULATION PROGRAM

a) Purpose

Enable the Calculation button at the customized property "XYCoordinates" at ModelItem level to show X, Y coordinates of the symbol in format of X/Y.

b) Problem Statement

Write an Active-X dll implementing the DoCalculate method to read the X, Y coordinates of a Symbol to the customized property ""XYCoordinates" at ModelItem level. The customized property ""XYCoordinates" should be added at ModelItem level, with datatype is String, format is Variable Length, Maximum Length is 40, and Category is Accessories.

c) Solution

◊ **Example code**

```
Implements ILMForeignCalc

Private Function ILMForeignCalc_DoCalculate(datasource As Llama.LMADataSource, _
   items As Llama.LMAItems, PropertyName As String, Value As Variant) As Boolean

   ILMForeignCalc_DoCalculate = ShowXYCoordinates(datasource, items, Value,
PropertyName)

End Function

Private Function ShowXYCoordinates(datasource As Llama.LMADataSource, _
   items As Llama.LMAItems, Value As Variant, PropertyName As String) As Boolean

   Dim Item As LMAItem
   Dim objSymbol As LMSymbol
   Dim objEquipment As LMEquipment

   'check if the selected Property is "XYCoordinates", then copy value from X, Y coordinates
   'to it

   ShowXYCoordinates = False
   For Each Item In items
      If PropertyName = "XYCoordinates" Then
On Error Resume Next
         Set objEquipment = datasource.GetEquipment(Item.Id)
On Error GoTo 0
         If Not objEquipment Is Nothing Then
            Set objSymbol =
datasource.GetSymbol(datasource.GetModelItem(Item.Id).Representations.Nth(1).Id)
            Value = objSymbol.Attributes("XCoordinate").Value & "/" &
objSymbol.Attributes("YCoordinate").Value
         End If
      End If
   Next
   ShowXYCoordinates = True

'clean up
```

```
    Set Item = Nothing
    Set objSymbol = Nothing
    Set objEquipment = Nothing
End Function
```

Save the Project and enter the ProgID in the Calculation ID field of the XYCoordinates Attribute in ModelItem through the DataDictionary Manager. Restart SPPID to find the button. Start the Project in Debug mode and then click on the button to step through your code. Then, compile the project, and click on the button again.

89. CREATE A VALIDATEPROPERTY PROGRAM

a) Purpose

Enable the Property validation at the ActuatorType attribute of InlineComp

b) Problem Statement

Write an Active-X dll implementing the DoValidateProperty method for placing corresponding actuator for an instrument valve when property ActuatorType is entered or changed.

c) Solution

◊ **Example code**

```
Implements ILMForeignCalc

Private Function ILMForeignCalc_DoValidateProperty(datasource As Llama.LMADataSource, _
   items As Llama.LMAItems, PropertyName As String, Value As Variant) As Boolean

   ILMForeignCalc_DoValidateProperty = AddActuator(datasource, items, PropertyName, Value)

End Function

'add actuator when property actuator type is changed on instrument
Private Function AddActuator(datasource As Llama.LMADataSource, _
   items As Llama.LMAItems, PropertyName As String, Value As Variant) As Boolean
'
On Error GoTo ErrHandler
'
   Dim Item As LMAItem
   Dim objPlacement As Placement
   Dim strFilePath As String
   Dim x As Double
   Dim y As Double
   Dim objSym As LMSymbol
   Dim objSymbol As LMSymbol

   Dim objInstr As LMInstrument
   Dim objPlantItem As LMPlantItem
   Dim objInstrActuator As LMInstrument
   Dim blnDelete As Boolean
   Dim blnNeedAdd As Boolean


   Set objPlacement = New Placement
   AddActuator = False
   For Each Item In items
      If Item.ItemType = "Instrument" And PropertyName = "ActuatorType" Then
         If Item.Attributes("InstrumentClass").Value = "Control valves and regulators" Then
            'get the instrument
            Set objInstr = datasource.GetInstrument(Item.Id)
            If objInstr.ChildPlantItemPlantItems.Count = 0 Then
               blnNeedAdd = True
            Else
```

```
            If objInstr.ChildPlantItemPlantItems.Count = 1 Then
               blnDelete =
objPlacement.PIDDeleteItem(objInstr.ChildPlantItemPlantItems.Nth(1).AsLMAItem)
               blnNeedAdd = True
            Else
               MsgBox "Wrong, there are more than 1 Child for this instrument!"
            End If
         End If
         If blnNeedAdd Then
            Select Case Value
               Case "Diaphragm"
                  strFilePath = "\Instrumentation\Actuators\Diaph Actuator.sym"
               Case "Single acting cylinder"
                  strFilePath = "\Instrumentation\Actuators\Single Action Cyl Act.sym"
               Case "Pilot operated cylinder"
                  strFilePath = "\Instrumentation\Actuators\Pilot Operated Cyl Act.sym"
               Case "Motor"
                  strFilePath = "\Instrumentation\Actuators\Motor Actuator.sym"
               Case "Digital"
                  strFilePath = "\Instrumentation\Actuators\Digital Actuator.sym"
               Case "Electro-hydraulic"
                  strFilePath = "\Instrumentation\Actuators\Electric-Hydraulic Act.sym"
               Case "Single solenoid"
                  strFilePath = "\Instrumentation\Actuators\Solenoid Actuator.sym"
               Case "Single solenoid w/reset"
                  strFilePath = "\Instrumentation\Actuators\Solenoid Act w-Man Reset.sym"
               Case "Double solenoid"
                  strFilePath = "\Instrumentation\Actuators\Double Solenoid Act.sym"
               Case "Pilot"
                  strFilePath = "\Instrumentation\Actuators\Pilot Actuator.sym"
               Case "Weight"
                  strFilePath = "\Instrumentation\Actuators\Weight Actuator.sym"
               Case "Manual"
                  strFilePath = "\Instrumentation\Actuators\Manual Actuator.sym"
               Case "Spring"
                  strFilePath = "\Instrumentation\Actuators\Spring Actuator.sym"
               Case "Capacitance sensor"
                  strFilePath = "\Instrumentation\Actuators\Capacitance Sensor Act.sym"
               Case "Ball float"
                  strFilePath = "\Instrumentation\Actuators\Ball Float Actuator.sym"
               Case "Displacement float"
                  strFilePath = "\Instrumentation\Actuators\Displacement Float Actuator.sym"
               Case "Paddle wheel"
                  strFilePath = "\Instrumentation\Actuators\Paddle Wheel Actuator.sym"
               Case "Diaphragm Rotary Actuator"
                  strFilePath = "\Instrumentation\Actuators\Diaph Actuator.sym"
                  Set objSym = datasource.GetSymbol(objInstr.Representations.Nth(1).Id)
                  x = objSym.XCoordinate
                  y = objSym.YCoordinate
               Case Else
                  'do nothing
                  strFilePath = ""
            End Select

            If strFilePath <> "" Then
               Set objSym = datasource.GetSymbol(objInstr.Representations.Nth(1).Id)
```

```
                x = objSym.Attributes("XCoordinate")
                y = objSym.Attributes("YCoordinate")
                Set objSymbol = objPlacement.PIDPlaceSymbol(strFilePath, x, y) '
targetitem:=objSym.AsLMAItem)
              Else
                MsgBox "Couldn't find corresponding Actuator"
              End If
            End If
            AddActuator = True
          End If
        End If

    Next

'clean up
    Set objPlacement = Nothing
    Set Item = Nothing
    Set objSym = Nothing
    Set objSymbol = Nothing
    Set objInstr = Nothing
    Set objPlantItem = Nothing
    Set objInstrActuator = Nothing

    Exit Function
ErrHandler:
    MsgBox "Error happened: " & Err.Description
'clean up
    Set objPlacement = Nothing
    Set Item = Nothing
    Set objSym = Nothing
    Set objSymbol = Nothing
    Set objInstr = Nothing
    Set objPlantItem = Nothing
    Set objInstrActuator = Nothing
End Function
```

Save the Project and enter the ProgID in the Validation ID field of the ActuatorType Attribute in InlineComp through the DataDictionary Manager. Restart SPPID. Start the Project in Debug mode and then select different Actuators through ActuatorType property to step through your code. Then, compile the Project, and change the property again.

90. CREATE A VALIDATEITEM PROGRAM

a) Purpose

Enable the Item validation when placing PipeRun.

b) Problem Statement

User added a new property "SystemCode" for Drawing, user want this property value to be copied to new Piperuns when placing them. Write an Active-X dll implementing the DoValidateItem method when placing PipeRun to make the copy from Drawing to PipeRun. You will notice a problem the a ProgID has existed for the PipeRun, learning how to call another validation program through your code.

c) Solution

◊ **Example code**

```
Implements ILMForeignCalc

Private Function ILMForeignCalc_DoValidateItem(DataSource As Llama.LMADataSource, _
   Items As Llama.LMAItems, Context As ENUM_LMAValidateContext) As Boolean

   Dim PlantItemValidate As ILMForeignCalc

   'Call PlantItemValidation.Validate
   Set PlantItemValidate = CreateObject("PlantItemValidation.Validate")
   If Not PlantItemValidate Is Nothing Then
      ILMForeignCalc_DoValidateItem = PlantItemValidate.DoValidateItem(DataSource, Items,
Context)
   End If

   'call function to place actuator
   ILMForeignCalc_DoValidateItem = CopySystemCode(DataSource, Items, Context)

End Function

Private Function CopySystemCode(DataSource As Llama.LMADataSource, _
   Items As Llama.LMAItems, Context As ENUM_LMAValidateContext) As Boolean
   Dim objLMAItem As LMAItem
   Dim objDrawing As LMDrawing
   Dim objModelItem As LMModelItem

   If Context = LMAValidateCreate Then
      For Each objLMAItem In Items
         Set objModelItem = DataSource.GetModelItem(objLMAItem.Id)
         Set objDrawing = objModelItem.Representations.Nth(1).DrawingObject
         objLMAItem.Attributes("SystemCode").Value =
objDrawing.Attributes("SystemCode").Value
         objLMAItem.Commit
      Next
   End If
End Function
```

Save the Project and enter the ProgID in the Validation Program field of the PipeRun through the DataDictionary Manager – DataBase Item Types. Restart SPPID. Start the Project in Debug mode and then place an PipeRun to step through your code. Then, compile the Project, and place PipeRun again.

OPTIONAL LABS

1. WRITE A SIMPLE VB CODE AND DEBUG IT

a) Purpose

To relate Visual Basic to procedural languages

b) Problem Statement

Create a "Hello World" program using the Sub main().

c) Solution

◊ **Open a Standard Executable in Visual Basic**

1. From the Start menu, click on Programs folder/Microsoft Visual Basic 6./Visual Basic 6.0.
2. Open a Standard.exe project.
3. Select the default form and delete it through Project\Remove Form1.
4. Add a Module to the project through Project\Add Module.
5. Set Project1 Properties show Sub Main as the start-up procedure. Select Project\Project Properties to get the dialog box.

◊ **Enter the following code:**

```
Sub Main()
    Debug.print "Hello World"
    Msgbox "Hello World"
End Sub
```

◊ **Save all the files associated with the Project**

◊ **Step through the program in Debug mode**

◊ **Compile and run the program**

2. WRITE A SIMPLE VB CODE USING A FORM

a) Purpose

To introduce the object-oriented features of Visual Basic

b) Problem Statement

Create a "Hello World" program using Form-level code.

c) Solution

◊ **Open a Standard Executable in Visual Basic**

1.From the Start menu, click on Programs folder/Microsoft Visual Basic 6./Visual Basic 6.0.
2.Open a Standard.exe project.
3.Create Command button on the default form.

4.Double-click the Command button to simulate the click Event and open the definition for the Click Event.

◊ **Enter the following code:**

```
Private Sub Command1_Click()
  MsgBox "Hello World"
End Sub
```

◊ **Save all the files associated with the Project**

◊ **Step through the program in Debug mode**

◊ **Compile and run the program**

◊ **Change the caption and name of the Command button and repeat the exercise**

3. USE THE OBJECT BROWSER TO VIEW AUTOMATION OBJECTS

a) Purpose

To become familiar with looking up libraries in Visual Basic

b) Problem Statement

Use the Object Browser to examine the classes, methods, and properties of the Microsoft Excel library and the SmartPlant P&ID library.

c) Solution

◊ **Open a Standard Executable in Visual Basic**

5. From the Start menu, click on Programs folder/Microsoft Visual Basic 6./Visual Basic 6.0.
6. Open a Standard.exe project.

◊ **Reference Microsoft Excel into the Project**

7. From the Project/References, select Microsoft Excel 8.0 Object Library

◊ **Browse the Microsoft Excel library using the Object Browser**

8. Click on the Object Browser icon and select Excel in the pull down list.
9. Click on each class on the left hand side to view its methods and properties on the right.
10. Click on each method or property for more information on them.

◊ **Reference Llama into the Project**

11. From the Project/References, select Intergraph SmartPlant P&ID Logical Model Automation

◊ **Browse the Llama library using the Object Browser**

4. WRITE VB CLIENT APPLICATION TO ACCESS MICROSOFT EXCEL'S AUTOMATION OBJECTS

a) Purpose

To practice writing an Active-X client component to access an Active-X servernents

b) Problem Statement

Write an Active-X client executable to interact with Microsoft Excel and perform the following operations:
Create a workbook and assign a value to a range of cells and save the workbook. Re-open the workbook and examine the contents of the cells.
(Note: Excel Application contains Workbooks which contain Worksheets. Each Worksheet contains Ranges, which contain Columns and Rows.)

c) Solution

◊ **Open a Standard Executable in Visual Basic**

1. From the Start menu, click on Programs folder/Microsoft Visual Basic 6./Visual Basic 6.0.
2. Open a Standard.exe project.
3. Create a Reference to the Excel object library

◊ **Add code to start up Excel Application and make it visible**

4. Use the CreateObject method to create an instance of Excel.Application
5. Make Excel Application visible by setting the boolean 'Visible' property to True
6. Examine the Excel instance visually. The Application may not have any open workbooks.

◊ **Expand code to Add a new workbook**

7. Use the 'Add' method in the Workbooks object in Excel to add a workbook.
8. Create an object variable to point to the new workbook.
9. Examine the Excel instance visually. The Workbook comes with 3 worksheets.

◊ **Expand code to select a range in a worksheet**

10. Use the Worksheets method in the Workbook object and set a variable pointing to "Sheet1".
11. Create a range in Sheet1 covering A1 to E10 using the 'range' method of the Worksheet object and set a variable pointing to it.

◊ **Include code to set a value in the range**

12. Use the methods in the Range object and print out the number of columns and rows in the range.
13. Use the 'Value' property of the Range object and assign a value to every cell in the range.
14. Examine the Excel Application visually.

◊ **Save and close the workbook**

15. Use the SaveAs method of the workbook to save the file. Give a filename as argument.
16. Use Close method to close the workbook.
17. Examine the Excel Application visually.

◊ **Re-open the Excel Workbook and examine the cells**

18. Use the Open method in the Workbooks object of ExcelApplication.
19. Select the Workbook from the Workbooks collection by name and assign a variable to it.
20. Select Sheet1 from the Worksheets and examine the 'Value' of cell (10, 5) using the Cells property.

◊ **Example Code**

```
Dim objExcel As Excel.Application
```

```
Set objExcel = CreateObject("Excel.Application")
objExcel.Visible = True

Dim xlWorkbook As Excel.Workbook
Set xlWorkbook = objExcel.Workbooks.Add

Dim xlWorksheet As Excel.Worksheet
Set xlWorksheet = xlWorkbook.Worksheets("SHEET1")

Dim range As range
Set range = xlWorksheet.range("A1", "E10")
range.Value = 10

Dim strFileName As String
strFileName = Environ("TEMP") & "\Excel1.xls"

objExcel.Workbooks(1).SaveAs (strFileName)
objExcel.Workbooks(1).Close

objExcel.Workbooks.Open (strFileName)
Set xlWorkbook = objExcel.Workbooks("Excel1.xls")
Debug.Print xlWorkbook.Sheets.Count

Set xlWorksheet = xlWorkbook.Sheets("SHEET1")
Debug.Print xlWorksheet.name
Debug.Print xlWorksheet.Cells(10, 5)

xlWorkbook.Close True
objExcel.Quit

Set xlWorksheet = Nothing
Set xlWorkbook = Nothing
Set objExcel = Nothing
```

5. CREATE AN ACTIVE-X SERVER AND A CLIENT APPLICATION

a) Purpose

To practice writing an Active-X server component that can be accessed by a client application

b) Problem Statement

Write an Active-X server in Visual Basic containing a class called Customer.
Provide the Customer class with Name, Address, and Age properties. Create Property procedures to set and get the values of these properties.
In the Age property, ensure that no age less than zero (0) can be assigned. The Age property should use a default value of zero in such cases.
Create a Sub called Display to display the whole Name, Address, and Age of the customer in a single message box.

c) Solution

◊ **Open a Active-X DLL in Visual Basic**

1. From the Start menu, click on Programs folder/Microsoft Visual Basic 6./Visual Basic 6.0.
2. Open a Active-X DLL project.

◊ **Create a Class Module**

12. Select Project\Add Class Module
13. Rename the Class Module to "Customer"
14. Dimension three private variables: strName as string, strAddress as string, intAge as integer
15. Create three Property procedures with Get and Let definitions: Name, Address, Age.
16. In the Let procedure for the Age property, include an IF structure to ensure that the age is never negative.
17. Create Sub called Display () that will concatenate the Name, Address and Age properties and displays it in a message box. [Use Msgbox (...) to display]

◊ **Create a executable application reference the Active-X DLL**

18. Create a executable application reference the Active-X DLL
19. Create a form in the executable application
20. Create a command on the form and double click it to enter code into its click event.
21. Dimension and create **three** instances of the Customer class. Use the Name, Address, and Age properties to input data into the object. Use the Display subroutine to display the combined information.

◊ **Example Code**

Sample code in Class Customer:

```
Dim strname As String
Dim strAddress As String
Dim intAge As Integer
Public Property Get name() As String
  name = strname
End Property
Public Property Let name(vdata As String)
  strname = vdata
End Property
```

```
Public Property Get Address() As String
   Address = strAddress
End Property

Public Property Let Address(ByVal vNewValue As String)
   strAddress = vNewValue
End Property

Public Property Get Age() As Integer
   Age = intAge
End Property

Public Property Let Age(ByVal vNewValue As Integer)
   If vNewValue >= 0 Then
      intAge = vNewValue
   End If
End Property
Public Sub Display()
   MsgBox "Name=" & strname & "   Address=" & strAddress & "   Age=" & intAge
End Sub
```

Sample code in Client application:

```
Dim obj1 As Customer
Dim obj2 As Customer
Dim obj3 As Customer

Set obj1 = New Customer
Set obj2 = obj1
Set obj3 = New Customer

obj1.Address = "309 Ball St., College Station, TX"
obj1.name = "Tom"
obj1.Age = 20

obj2.name = "Dave"
obj3.name = "David"
obj3.Age = -100


Debug.Print obj1.name
Debug.Print obj2.name
Debug.Print obj3.name

obj1.Display
obj2.Display
obj3.Display

Set obj1 = Nothing
Set obj2 = Nothing
Set obj3 = Nothing
```

6. CREATE AN INTERFACE, AN IMPLEMENTATION, AND A CLIENT APPLICATION

a) Purpose

To practice writing Active-X server components which support interfaces

b) Problem Statement

Create an Active-X interface, its implementation, and a client driver program to use the two libraries.

c) Solution

◊ **Create the Interface Project**

1. Create an Active-X dll project called Interface.
2. Create two class modules and name them IAccount and IPurchase.
   22. IAccount has one function AccountBalance and IPurchase has one function LastPurchaseAmount, each returning a double datatype.
   23. Save the project and run it.

◊ **Create a project called Implementation with a single class called CreditCard**

   24. Create an Active-X dll project called Implementation.
   25. Reference the Interface dll.
   26. Open the class module and name it CreditCard.
   27. Implement the two interfaces in the class.
   28. Create two private variables to hold the lastPurchaseAmount and the accountBalance.
   29. Add a property called paymentAmount which reduces the account balance by the given amount in the Let definition.
   30. Add a second property called PurchaseAmount which increases the account balance by the given amount and saves the given amount as the lastPurchaseAmount, in the Let definition.
   31. Implement the two functions from the interfaces to return the values of the two appropriate variables.

◊ **Create a project called Client with a Form**

   32. Create a standard executable project called Client.
   33. Reference the two dlls created above.
   34. Dimension one variable each for the three classes developed above.
   35. Create a new creditcard object and assign it some payment and purchase using the creditcard object's properties.
   36. Set the two interface variables to point to the creditcard object.
   37. Print out the AccountBalance and LastPurchaseAmount using the two variables.

◊ **Example Code**

IAccount interface:

```
Public Function AccountBalance() As Double

End Function
```

IPurchase Interface:

```
Public Property Get LastPurchaseAmount() As Double

End Property
```

Implementation Class: CreditCard

```
Option Explicit
```

```
Implements IAccount
Implements IPurchase

Private mvarLastPurchaseAmount As Double
Private mvarAccountBalance As Double

Private Function IAccount_AccountBalance() As Double
    IAccount_AccountBalance = mvarAccountBalance
End Function

Private Property Get IPurchase_LastPurchaseAmount() As Double
    IPurchase_LastPurchaseAmount = mvarLastPurchaseAmount
End Property

Public Property Let PurchaseAmount(vdata As Double)
    mvarLastPurchaseAmount = vdata
    mvarAccountBalance = mvarAccountBalance + vdata
End Property

Public Property Let paymentAmount(vdata As Double)
    mvarAccountBalance = mvarAccountBalance - vdata
End Property

Public Function AddFinanceCharge(percent As Double) As Double
    Dim interest As Double
    interest = percent / 100# * mvarAccountBalance
    mvarAccountBalance = mvarAccountBalance - interest
    AddFinanceCharge = interest
End Function
```

Client Driver Program:

```
    Private Sub Command1_Click()
    Dim accountInterface As IAccount
    Dim purchaseInterface As IPurchase

    Dim creditCard As creditCard
    Set creditCard = New creditCard
    creditCard.PaymentAmount = 500.34
    creditCard.PurchaseAmount = 400#
    Set accountInterface = creditCard
    Set purchaseInterface = creditCard
    Debug.Print accountInterface.AccountBalance
    Debug.Print purchaseInterface.LastPurchaseAmount

    creditCard.AddFinanceCharge 10.9
    Debug.Print accountInterface.AccountBalance
    Debug.Print purchaseInterface.LastPurchaseAmount
```

End Sub

7. FIND OPC AND FROM/TO

a) Purpose

Get familiar with LLAMA

b) Problem Statement

(1) place Vessel, with two nozzles on it and then draw a piperun from one of the nozzles and place a OPC to the open end of the piperun, then open another drawing, place a vessel with two nozzles on it, and then draw a piperun from one of the nozzles, and then place the pairedOPC to the piperun,
(2) write a standalone application start from the vessel in first drawing, navigate from vessel, to nozzle, to piperun, to OPC,
(3) continue the navigation, to pairedOPC (in other drawing), to piperun, to nozzle, and to the vessel.
(4) place a valve on the piperun in first drawing, then repeat step (2) & (3)
(5) place a two more nozzles on the vessel in second drawing, place an off-line instrumentation with implied components, such a Discr Field Mounted LC, then use SignalLine-Connect to Process to link nozzle with Instrumentation, navigate from vessel in first drawing until find the implied component (which is valve)
(6) continue the navigation from implied component to the vessel in second drawing
(7) change the property SupplyBy to "By A" for the vessel, nozzle, and piperun
(8) integrate this function to a validation code, which will be run when SupplyBy property of vessel is changed. (due to the limitation of the code, you may start from the vessel in first drawing)

c) Solution

◊ **Example code**

8. LABEL PLACEMENT AND REPLACEMENT UTILITY

a) Purpose

To apply the knowledge you learned in this cource for both LLAMA and PLACEMENT

b) Problem Statement

Write a standalone application to obtain all Vessel on a drawing, then check if the EquipmentID label is placed on the Vessel or not, if not, place the EquipmentID label to the Vessel, if the EquipmentID label is placed already, replace the label. In this way, the latest EquipmentID label will be placed on each Vessel.

c) Solution

1. Use LMAFilter to find all Vessels in current active drawing.
2. Use PIDPlaceLabel to place new label.
3. Use PIDReplaceLabel to replace existing label.

◊ **Example code**

9. AUTOMATICALLY CREATE NEW DRAWINGS

a) Purpose

To apply the knowledge you learned in this cource for LLAMA, PLACEMENT and PIDAutomation.

b) Problem Statement

Write a standalone application to create a new drawing, then place two assemblies into the new drawing (assemblies are pre-defined, with one assembly has a piperun with one open end, and the other assembly has an nozzle without any piperun connected). Then place a new Piperun to connect the open end of the piperun to the nozzle, and use PIDAutoJoin method to auto join the existing piperun and new placed piperun.

c) Solution

◊ **Example code**

10. CALCULATION VALIDATION (1)

a) Purpose

Get familiar with Calculation Validation

b) Problem Statement

Write an Active-X dll implementing the DoCalculate method for creating a value for the Name of Vessel. Ask user to enter the name they want to give to the vessel, then combine with SP_ID of the vessel to obtain the final name of the vessel.

c) Solution

◊ **Example code**

11. CALCULATION VALIDATION (2)

a) Purpose

Get familiar with Calculation Validation

b) Problem Statement

Write an Active-X dll implementing the DoCalculate method for placing an assembly. Create a new property called "Place Assembly" for PipingComp, placing an assembly when user click the Calculation button on the "Place Assembly" field and the item type is "Valve". Place the assembly somewhere outside of the border.

c) Solution

◊ **Example code**

12. PROPERTY VALIDATION (1)

a) Purpose

Get familiar with Property Validation

b) Problem Statement

Write an Active-X dll implementing the DoValidateProperty method for populating the value for the Name of Vessel when user enter the value for TagPrefix of vessel. Vessel name is combination of TagPrefix and SP_ID

c) Solution

◊ **Example code**

13. PROPERTY VALIDATION (2)

a) Purpose

Get familiar with Property Validation

b) Problem Statement

Write an Active-X dll implementing the DoValidateProperty method for populating the value for the "Pressure Drop" of Relief Device, "Pressure Drop" is a new property for Relief Device, which is difference of Oper Max Pressure between two piperuns connected to Relief Device. When one of Oper Max Pressures is changed, Validation code should be fired and calculate the value for the "Pressure Drop"

c) Solution

1. Some Relief Devices have more than two piperuns connected to them, select one with only two piperuns connected.

◊ **Example code**

14. ITEM VALIDATION (1)

a) Purpose

Get familiar with Item Validation

b) Problem Statement

Write an Active-X dll implementing the DoValidateItem method for creating a value for the Name of Vessel when a vessel is placed on drawing. Vessel name is combination of "T" and SP_ID

c) Solution

◊ **Example code**

15. ITEM VALIDATION (2)

a) Purpose

Get familiar with Item Validation

b) Problem Statement

Write an Active-X dll implementing the DoValidateItem method to clean the OperFluidCode if the line number label is deleted from the Piperun.

c) Solution

◊ **Example code**

16. ITEM VALIDATION (3)

a) Purpose

Get familiar with Item Validation

b) Problem Statement

Write an Active-X dll implementing the DoValidateItem method to write a log file with all information about who/when place, delete, and modify items.

c) Solution

◊ **Example code**

17. Modify PlantItem Validation

a) Purpose

Get familiar with PlantItem Validation Code.

b) Problem Statement

Modify the delivered PlantItem Validation code to keep the original tag sequence no when copy/paste assembly.

c) Solution

◊ **Example code**

18. Modify ItemTag Validation

a) Purpose

Get familiar with ItemTag Validation Code.

b) Problem Statement

Modify the delivered ItemTag Validate code to allow ItemTag of PipeRun including NominalDiameter.

c) Solution

◊ **Example code**

19. Modify Import Code

a) Purpose

Get familiar with Import Code.

b) Problem Statement

Modify the delivered Import code to allow import more properties for Equipment, new properties such as "Height" of vessel, and/or a user defined property.

c) Solution

◊ **Example code**

20. NEW MOCRO FOR INSTRUMENT REPORT

a) Purpose

Create new macro to enhance the functionarity of Instrument Report.

b) Problem Statement

Write a macro for Instrument Report to obtain what items that connected the instrumentation through "Connect to process" SignalRun.

c) Solution

1. "Connect to process" SignalRun is actually a special PipeRun, whose PipeRunType is "Conn to process/supply"
2. You may limit your code to only report the items if they are PipeRun or Nozzle

◊ **Example code**

21. IMPROVEMENT OF FROM/TO MACRO

a) Purpose

To improve the functionality of From/To Macro

b) Problem Statement

Modify the the From/To Macro to not reporting Branch Piperuns.

c) Solution

◊ **Example code**

SmartPlant P&ID Data Model