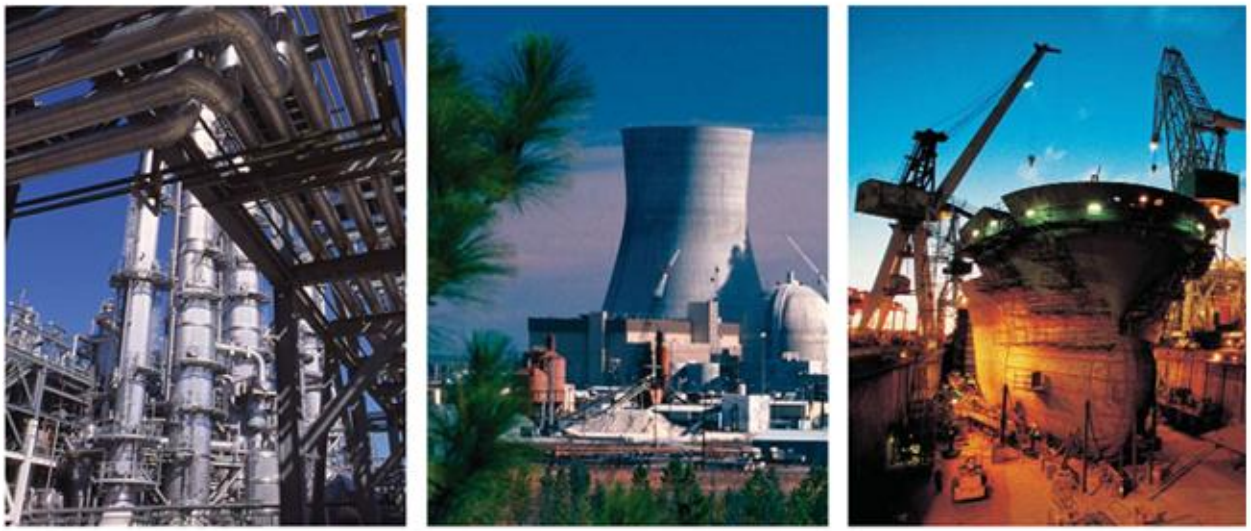
SmartPlant P&ID 2009

Automation Programming with VB Labs

Process, Power & Marine



INTERGRAPH

Version 2009 **September 2009** **DPID2-TP-100023C**

Copyright

Restricted Rights Legend

Intergraph Corporation
Huntsville, Alabama 35894-0001

Warranties and Liabilities

All warranties given by Intergraph Corporation about equipment or software are set forth in your purchase contract, and nothing stated in, or implied by, this document or its contents shall be considered or deemed a modification or amendment of such warranties. Intergraph believes the information in this publication is accurate as of its publication date.

The information and the software discussed in this document are subject to change without notice and are subject to applicable technical product descriptions. Intergraph Corporation is not responsible for any error that may appear in this document.

The software discussed in this document is furnished under a license and may be used or copied only in accordance with the terms of this license.

No responsibility is assumed by Intergraph for the use or reliability of software on equipment that is not supplied by Intergraph or its affiliated companies. THE USER OF THE SOFTWARE IS EXPECTED TO MAKE THE FINAL EVALUATION AS TO THE USEFULNESS OF THE SOFTWARE IN HIS OWN ENVIRONMENT.

Trademarks

Table of Contents

Preface

This document is a user's guide for SmartPlant P&ID 2007 automation programming with VB labs.

Send documentation comments or suggestions to PPMdoc@intergraph.com

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. 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 plant, then create a new drawing and place another Vessel in it.

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

Set datasource = New LMADataSource
datasource.SiteNode = "C:\HostSite\SmartPlantV4.ini"
datasource.ProjectNumber = "Lin_Plant3!Lin_Plant3"

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

Set datasource = Nothing
Set objVessel = Nothing
```

3. 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
```

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

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
```

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

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
```

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. ROLLBACK

a) Purpose

To rollback a transaction by automation program

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
```

8. 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
```

9. ACCESS LMAATTRIBUTES COLLECTION

Purpose

To access LMAAttributes collection of LLAMA object.

Problem Statement

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

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. 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
```

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

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
```

13. 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
```

14. 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
```

15. 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
```

16. 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
```

17. CREATE FILTER WITH SELECT LIST DATA IN CRITERIA

a) Purpose

To create a filter with select list data in criteria, learn how to resolve the select data to its index dynamically.

b) Problem Statement

Place couple piping valves in drawing. Write a standalone application to collect all Ball Valves.

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 = "PipingCompType"
criterion.ValueAttribute = datasource.CodeList("Piping Component
Type").EnumeratedAttributes.GetItemIndex("Ball valve")
criterion.Operator = "="
objFilter.ItemType = "PipingComp"
objFilter.Criteria.Add criterion

Dim PipingComp As LMPipingComp
Dim PipingComps As LMPipingComps
Set PipingComps = New LMPipingComps
PipingComps.Collect datasource, Filter:=objFilter

Debug.Print "Number of Ball Valves retrieved = " & PipingComps.Count


Set datasource = Nothing
Set objFilter = Nothing
Set criterion = Nothing
Set PipingComp = Nothing
Set PipingComps = 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(30 - 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(30 - 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

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, TEMA 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

a) Purpose

To get familiar with relationship between PlantItemGroup and PlantItem

c) 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

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.

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

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.

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

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

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

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

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 LMAActiveProject
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

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

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

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

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


'PIDSetCopyPropertiesFlag is not working, always copy the properties according to Rule
''''    '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

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

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

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

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

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

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

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.

91. CREATE A DRAWING VALIDATE PROGRAM

Purpose

Enable the Drawing validation when a drawing event is triggered.

d) Problem Statement

System admin wants to log the time user name when a drawing is opened, closed or printed. This example writes an Active-X dll (DrawingValidation.dll) implementing the DoValidateItem method when ac drawing event (Open, Close, Print, Create, Modify) is detected. The ProgID, **DrawingValidation.Validate** needs to be assigned to Drawing object in DataDictionary Manage -> DataBase Itemtypes table.

e) Solution

◊ **Example code**

```
Option Explicit

Implements ILMForeignCalc
Implements IPrintValidation

Private Function ILMForeignCalc_DoCalculate(DataSource As Llama.LMADataSource, Items As
Llama.LMAItems, PropertyName As String, Value As Variant) As Boolean

End Function

'Creates a new file each time an event is fired
'File name is context type dependent
'Code has also been added for AutoGap on Drawing Close Event

Private Function ILMForeignCalc_DoValidateItem(DataSource As Llama.LMADataSource, Items As
Llama.LMAItems, Context As LMForeignCalc.ENUM_LMAValidateContext) As Boolean

   Dim sFileName As String
   Dim strCurrentUser As String
   Dim strDwgName As String
   On Error GoTo ErrHndl

   Select Case Context
      Case LMForeignCalc.ENUM_LMAValidateContext.LMAValidateOpen
         sFileName = "Drawing Opened
      Case LMForeignCalc.ENUM_LMAValidateContext.LMAValidateClose
        sFileName = "Drawing Closed "
        'Begin AutoGap on active drawing
        ' Need to reference AutoGapAll.dll and RAD2d.dat for this example
         Dim auto As New AutoGapAll.AutoGapAllCmd
         auto.GapAll DataSource.PIDMgr.Application.RADApplication
         'End AutoGap on active drawing

      Case LMForeignCalc.ENUM_LMAValidateContext.LMAValidateCreate
         sFileName = "Drawing Created
      Case LMForeignCalc.ENUM_LMAValidateContext.LMAValidateDelete
        sFileName = "Drawing Deleted "
      Case LMForeignCalc.ENUM_LMAValidateContext.LMAValidateModify
         sFileName = "Drawing Modified
```

```
    End Select

    'Create file
    If Len(sFileName) > 0 Then CreateFile sFileName, Items

    ILMForeignCalc_DoValidateItem = True
    Exit Function

ErrHndl:
     MsgBox Err.Description
    ILMForeignCalc_DoValidateItem = False

End Function

'Create file for the Context Type (Create, Delete, Open, Close, Modify, or Print) in the "Environ("Temp")" Directory
and log drawing info.
'Create the "Environ("Temp")" Directory if doesn't already exist.

Private Sub CreateFile(sFileName As String, Optional Items As Llama.LMAItems)

    Dim lFileNum As Long
    Dim fso As New FileSystemObject
    Dim sFolder As String

    sFolder = Environ("Temp")

    On Error GoTo ErrHndl

    'Create folder if doesn't exist
    If Not fso.FolderExists(sFolder) Then
       fso.CreateFolder sFolder
    End If

    sFileName = sFolder & "\" & sFileName & ".txt"

    lFileNum = FreeFile
    Dim f, fs
    Set fs = CreateObject("Scripting.FileSystemObject")
    Set f = fs.OpenTextFile(sFileName, ForAppending, -2)

    f.WriteLine (DateTime.Time)
    f.WriteLine ("  Item Type: " & Items.Nth(1).ItemType)
    f.WriteLine ("  Name: " & Items.Nth(1).Attributes("Name"))
    f.WriteLine ("  UserName: " & Environ("UserName") & vbCrLf)
    f.Close
    Set fs = Nothing
    Set fso = Nothing
    Exit Sub

ErrHndl:

    Err.Raise Err.Number, "Validate.CreateFile", Err.Description

End Sub
```

```
Private Function ILMForeignCalc_DoValidateProperty(DataSource As Llama.LMADataSource, Items As
Llama.LMAItems, PropertyName As String, Value As Variant) As Boolean
End Function

Private Sub ILMForeignCalc_DoValidatePropertyNoUI(DataSource As Llama.LMADataSource, Items As
Llama.LMAItems, PropertyName As String, Value As Variant)
End Sub

'Create a new file each time a print event is fired

Private Function IPrintValidation_DoValidatePrint(ByVal DrawingSPID As String, ByVal DrawingName As String,
DataSource As Llama.LMADataSource, Items As Llama.LMAItems) As Boolean
   On Error GoTo ErrHndl
   If IPrintValidation_UseDataSourceOnPrint Then
      CreateFile "Drawing Printed with DB ", Items
   Else
      CreateFile "Drawing Printed "
   End If
   IPrintValidation_DoValidatePrint = True
   Exit Function

ErrHndl:
   IPrintValidation_DoValidatePrint = False
End Function

Private Property Get IPrintValidation_UseDataSourceOnPrint() As Boolean
   IPrintValidation_UseDataSourceOnPrint = True
End Property
```

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

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

Purpose

To become familiar with looking up libraries in Visual Basic

Problem Statement

Use the Object Browser to examine the classes, methods, and properties of the Microsoft Excel library and the SmartPlant P&ID library.

Solution

◊ **Open a Standard Executable in Visual Basic**

3. From the Start menu, click on Programs folder/Microsoft Visual Basic 6./Visual Basic 6.0.
4. Open a Standard.exe project.

◊ **Reference Microsoft Excel into the Project**

5. From the Project/References, select Microsoft Excel 8.0 Object Library

◊ **Browse the Microsoft Excel library using the Object Browser**

6. Click on the Object Browser icon and select Excel in the pull down list.
7. Click on each class on the left hand side to view its methods and properties on the right.
8. Click on each method or property for more information on them.

◊ **Reference Llama into the Project**

9. 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

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**

10. Select Project\Add Class Module
11. Rename the Class Module to "Customer"
12. Dimension three private variables: strName as string, strAddress as string, intAge as integer
13. Create three Property procedures with Get and Let definitions: Name, Address, Age.
14. In the Let procedure for the Age property, include an IF structure to ensure that the age is never negative.
15. 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**

16. Create a executable application reference the Active-X DLL
17. Create a form in the executable application
18. Create a command on the form and double click it to enter code into its click event.
19. 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

Purpose

To practice writing Active-X server components which support interfaces

Problem Statement

Create an Active-X interface, its implementation, and a client driver program to use the two libraries.

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.
   20. IAccount has one function AccountBalance and IPurchase has one function LastPurchaseAmount, each returning a double datatype.
   21. Save the project and run it.

◊ **Create a project called Implementation with a single class called CreditCard**

   22. Create an Active-X dll project called Implementation.
   23. Reference the Interface dll.
   24. Open the class module and name it CreditCard.
   25. Implement the two interfaces in the class.
   26. Create two private variables to hold the lastPurchaseAmount and the accountBalance.
   27. Add a property called paymentAmount which reduces the account balance by the given amount in the Let definition.
   28. 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.
   29. Implement the two functions from the interfaces to return the values of the two appropriate variables.

◊ **Create a project called Client with a Form**

   30. Create a standard executable project called Client.
   31. Reference the two dlls created above.
   32. Dimension one variable each for the three classes developed above.
   33. Create a new creditcard object and assign it some payment and purchase using the creditcard object's properties.
   34. Set the two interface variables to point to the creditcard object.
   35. 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. MODIFY PROPERTY BASED ON CONSTRUCTION STATUS

Purpose

Get familiar with LLAMA

Problem Statement

(1) Place two vessels, one with Construction Status = NEW, the other with Construction Status = EXISTING
(2) Write an EXE to get two vessels, modify Description property to "New Construction Status" if the Construction Status = NEW, and modify Description property to "Existing Construction Status" if the Construction Status = EXISTING.

Solution

◊ **Example code**

8. SERACH ITEMS AND MODIFY PROPERTY

a) Purpose

Get familiar with LLAMA

b) Problem Statement

(1) Place some valves with different types, make sure some of them are Ball Valves
(2) Write an EXE to filter for Ball valves only, and set Nominal Diameter to 2" if the original value is not set yet.

c) Solution

◊ **Example code**

9. MODIFY CASE PROCESS DATA

a) Purpose

Get familiar with LLAMA

b) Problem Statement

(1) Place a Vessel, assign some process data to the vassal, such as max operating pressure, temperature, etc. When enter value for process data, intentionally some use default unit, some not.
(2) Write an EXE to access this vessel, and change the process data's display value to project default format if not yet.

c) Solution

Example code

10. FIND IMPLIED ITEMS AND MODIFY THEIR PROPERTY

a) Purpose

Get familiar with LLAMA

b) Problem Statement

(1) Place some off-line instruments with implied components
(2) Write an EXE to filter for implied items, check if it is an Instrument Root Valve, if so, set its Nominal Diameter to 2"

c) Solution

◊ **Example code**

11. COUNT NOZZLES ON A VESSEL

a) Purpose

Get familiar with LLAMA

b) Problem Statement

(1) Place a vessel, and place several nozzles on it
(2) Write an EXE to count nozzles on the vessel, and set the vessel's Description property value to "Total number of nozzles on this vessel is: <the count of nozzles>"

c) Solution

◊ **Example code**

12. SEARCH ITEMS ACTIVE DRAWING STOCKPILE

a) Purpose

Get familiar with LLAMA

b) Problem Statement

(1) Place some vessels in the drawing, then remove some of them to drawing stockpile
(2) Write an EXE to filter for all vessels in drawing stockpile, and set the vessel's Description property value to "In drawing stockpile of the drawing: <DrawingName>".

c) Solution

Example code

13. NAVIGATE FROM OFF-LINE INSTRUMENT TO PROCESS PIPERUN

a) Purpose

Get familiar with LLAMA

b) Problem Statement

(1) Place an off-line instrument, connect the instrument to the process PipeRun with Connect-To-Process line.
(2) Write an EXE to navigate from off-line instrument the process PipeRun, set instrument's Description property value to "Connected Process PipeRun's Item tag is <Piperun ItemTag>.

c) Solution

Example code

14. 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**

15. HOW TO CHECK IF A DRAWING BELONG TO ACTIVE SITE

a) Purpose

How to determine if a drawing is belong to the active site in workshare environment? Or just a read only drawing?

b) Problem Statement

Create a workshare environment, with host site and satellite site, publish a drawing from host to satellite, where the drawing is read-only. Write a code to check if the drawing is read-only?

c) Solution

Get the PlantGroup object for the root item, which is the plant, then get the Workshare Site ID from the PlantGroup.
Get the drawing's DrawingSite object, then get the Workshare Site ID from DrawingSite.
Compare two Workshare Site IDs, if they are same, which means the drawing belongs to the active site, otherwise, it's a read-only drawing.

◊ **Example code**

16. LABEL FIND AND REPLACE 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**

17. 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**

18. NC/NO VALVES REPLACEMENT UTILITY

a) Purpose

To apply the knowledge you learned in this cource for both LLAMA and PLACEMENT

b) Problem Statement

Considering following workflow: In the beginning of project, same piping valve, for example ball valve, is used as for Normal Close (NC) and Normal Open (NO), by setting the attribute "Opening Action", to NC or NO. Later in the project, it is required to have different symbols for NC or NO valves. It is OK to keep the original valve as NO, but needs a new symbol for NC. Problem: how to update all placed valves in different drawings?

Write a standalone application to search all placed valves, using ball valve for example, then depends on its "Opening Action" is NC or NO, if it is NC, then to replace it with new valve.

You may start the project by open a drawing, and run the utility against the active drawing only. Later, enhance the utility to have function to batch process all drawings.

You need to create one new ball valve symbol as NC for this lab.

c) Solution

◊ **Example code**

19. 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**

20. 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**

21. 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**

22. 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**

23. 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**

24. 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**

25. 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**

26. 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**

27. 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**

28. 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**

29. 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. Then print out the ItemTag of the connected item if it is a Piperun, or the ItemTag of its parent if it is Nozzle.

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**

30. 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**