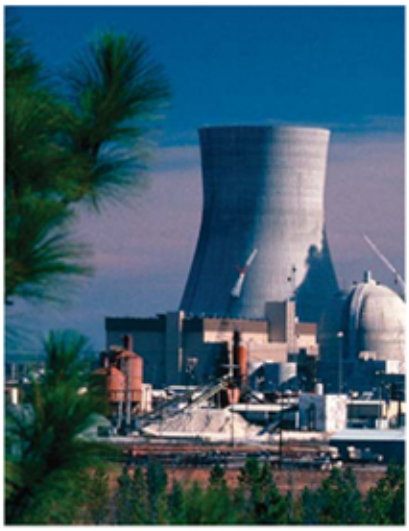
SmartPlant 3D Programming
Naming and Interference Rules

Student Workbook

Process, Power & Marine


INTERGRAPH

Version 2009 **August 2010** **N/A**

Copyright

Restricted Rights Legend

Use, duplication, or disclosure by the government is subject to restrictions as set forth below. For civilian agencies: This was developed at private expense and is “restricted computer software” submitted with restricted rights in accordance with subparagraphs (a) through (d) of the Commercial Computer Software - Restricted Rights clause at 52.227-19 of the Federal Acquisition Regulations (“FAR”) and its successors, and is unpublished and all rights are reserved under the copyright laws of the United States. For units of the Department of Defense (“DoD”): This is “commercial computer software” as defined at DFARS 252.227-7014 and the rights of the Government are as specified at DFARS 227.7202-3.

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.

Intergraph is not responsible for the accuracy of delivered data including, but not limited to, catalog, reference and symbol data. Users should verify for themselves that the data is accurate and suitable for their project work.

Trademarks

Table of Contents

Introduction

The Student workbook is designed as an aid for students attending the SP3D Programming I class presented by Intergraph Corporation, and it's a supplement to the standard product documentation.

Objective

This document is designed to provide a guideline for people who need to design symbol definitions and naming rules for the SmartPlant 3D application. This workbook includes, but is not limited to the following:

- Provides an overview of customization with the SmartPlant 3D software using standard Windows™ programming tools and languages like Visual Basic™.
- Describes some of the tools that can be used to design new symbol entities and naming rules.
- Provides examples of workflow customization.

Assumptions are made here that the user has a prerequisite knowledge of the SmartPlant 3D reference data.

Course description

- SmartPlant 3D Data Model
- Naming Rules
- Visual Basic Symbol Creation

Course Reference Material

SmartPlant 3D/IntelliShip Programmer's Guide
SmartPlant 3D Symbols Reference Data Guide
SmartPlant 3D Reference Data Guide

Understanding Smart Plant 3D Data Model

Overview

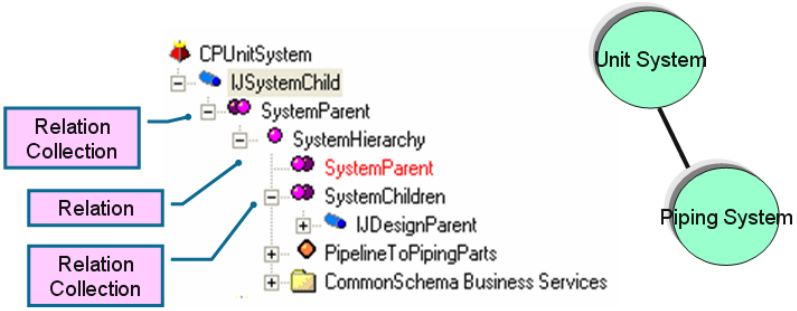
- Business Objects (BO's)
  - Business objects are COM objects designed to represent the various elements of the design model (Example: pipes, valves, etc..)
  - Business objects exist only in the Middle Tier and only for the lifetime of a transaction.
- Roles

  To be part of SmartPlant 3D framework, a business object must support specific roles by implementing one or more interfaces. These roles are:

  - Geometric - Has 3D geometries.
  - Persistent - Can be saved and restored.
  - Displayable - Can be viewed in the Client tier.
  - Relationship-enabled - Participates in relationships.
  - User attributes-enabled – Can add user attributes

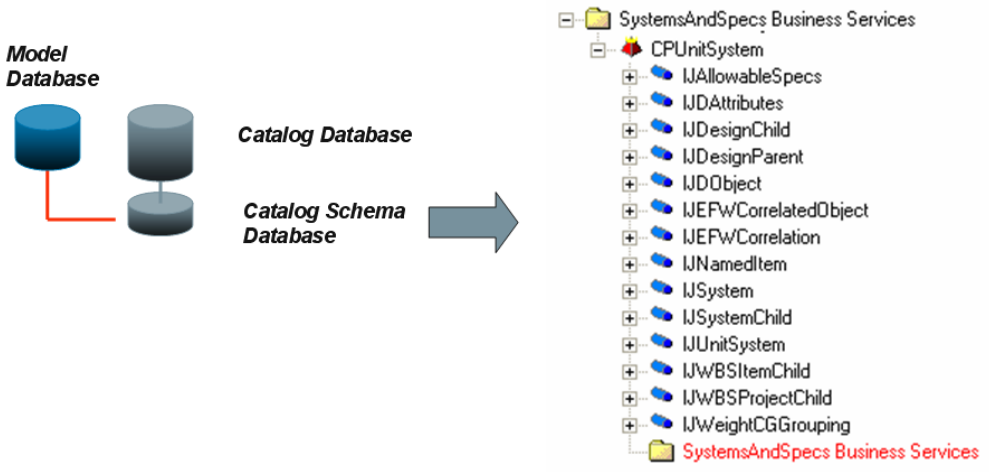
Overview

- Relationships
  - are the SmartPlant 3D business rules
  - define how BO's *Behave* with respect to each other
  - react to changes as they take place, ensuring data consistency
  - A relation is between two and only two business objects (entities)
  - These entities are known as **origin** and **destination** of the relation.

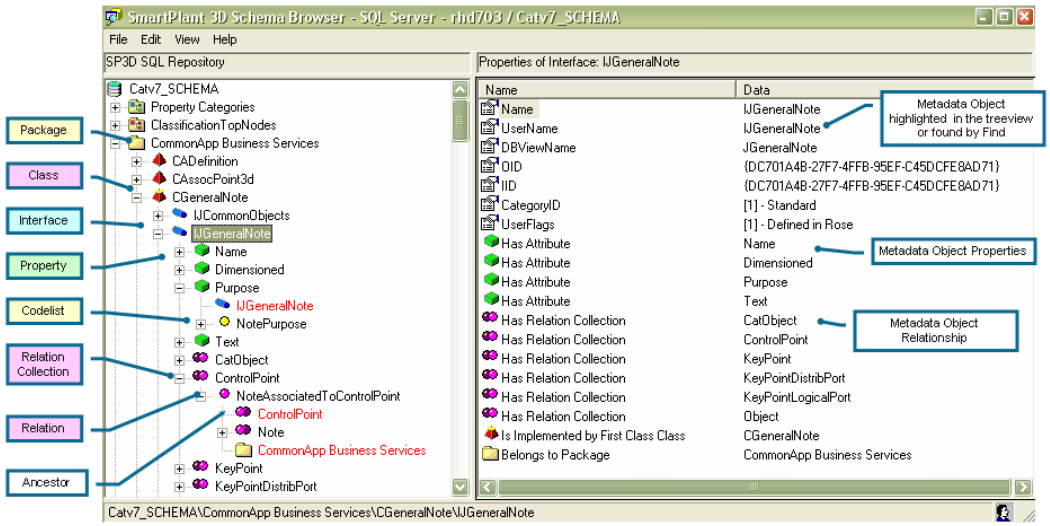


SmartPlant 3D Data Model

Use the Schema Browser Tool to view the data model



Schema Browser Tool

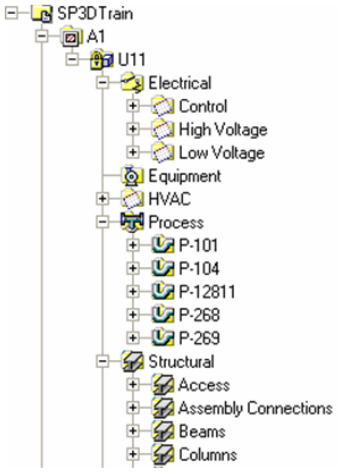


System Entity Data Model

Business Objects Defined in Systems And Specifications Application

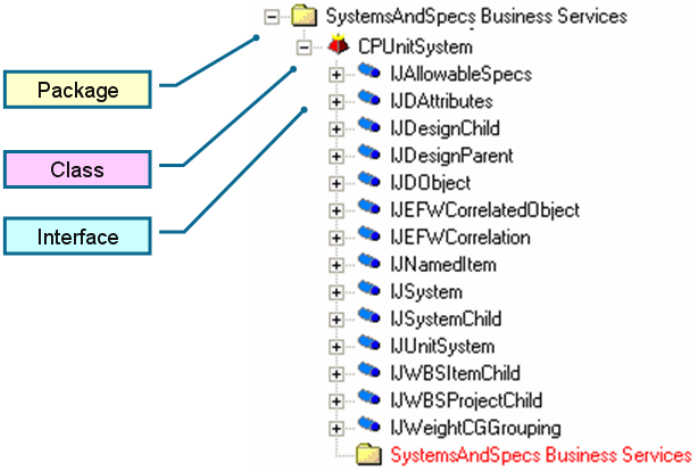
- The system hierarchy is a functional breakdown of the model (Plant/Ship) and is intended for design data management purposes.

- CPAreaSystem - Area System
- CPUnitSystem - Unit System
- CPMSystem - Generic System
- CPMachinerySystem - Equipment System
- CPPipingSystem - Piping System
- CPPipelineSystem - Pipeline System
- CPStructuralSystem - Structural System
- CPElectricalSystem - Electrical System
- CPConduitSystem - Conduit System
- CPDuctingSystem - HVAC System



System Entity Data Model

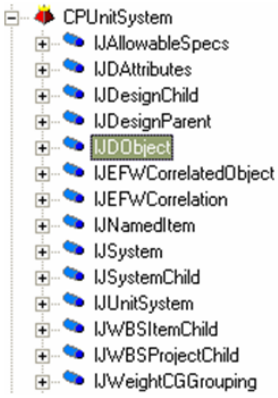
COM Classes and Interfaces



System Entity Data Model

Supported interfaces

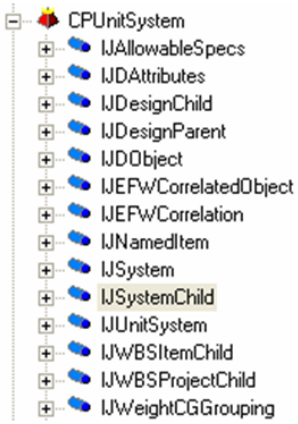
- IJDObject interface is a required interface for almost all objects and supports access control.
- IJNamedItem interface provides the name property for all named objects.
- IJDAttributes interface is required for the system object to support user-defined attributes.



System Entity Data Model

Supported interfaces

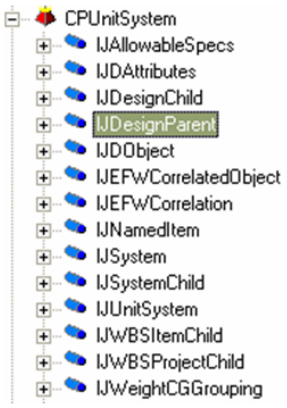
- IJSystemChild: Any entity that appears in the System Hierarchy and is capable of being a child of a system will implement this interface.
- IJAllowableSpecs: This interface is required in order to associate the system object to a collection of allowable specs.
- IJEFWCorrelation and IJEFWCorrelatedObject: This interface is required in order to associate the system object to the Engineering Framework Design Basis object and provides the EFW Correlation Properties.
- IJWeightCGGrouping :This IJWeightCGGrouping interface is to manage the weight and center of gravity (Weight&CG) for objects that represent logical groups of parts such as a system, an assembly, or a compartment.



System Entity Data Model

Supported interfaces

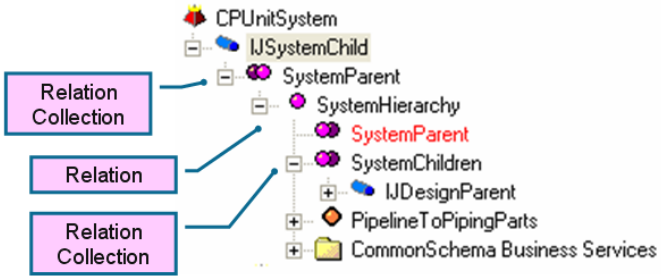
- IJSystem: This interface is used to provide a common interface for system objects.
- IJDesignParent and IJDesignChild:
  - An object in the system hierarchy that can have children must implement the IJDesignParent interface.
  - An object that can be the child of a parent must implement the IJDesignChild interface.
- IJWBSItemChild and IJWBSProjectChild:
  - An object in the workbreakdown structure hierarchy that can be the child of a parent must implement these interfaces.



System Entity Data Model

Use of Relationship

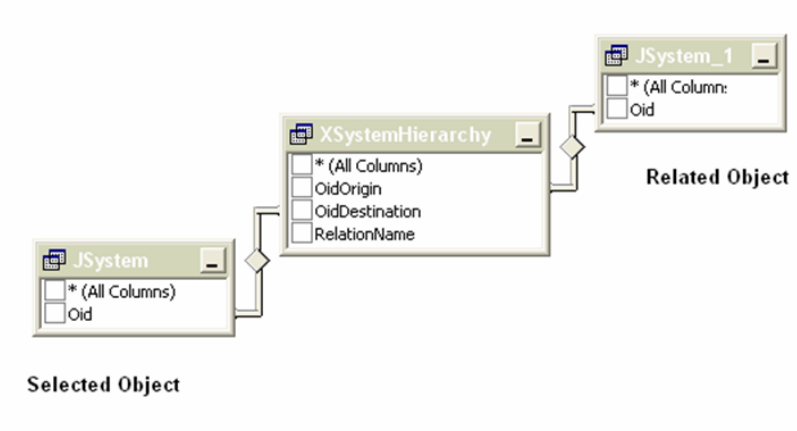
- Business objects must play many roles (Example: Relationship enable) by implementing one or more interfaces.
  - IJSystemChild: The interface destination of the relationship
  - IJDesignParent: The interface origin of the relationship
  - A relationship type (Applications define typed relationships)



System Entity Data Model

Accessing Relationships from SQL

- A relationship can be navigated; and the destination object can be retrieved by knowing the object identifier and the view relation table.

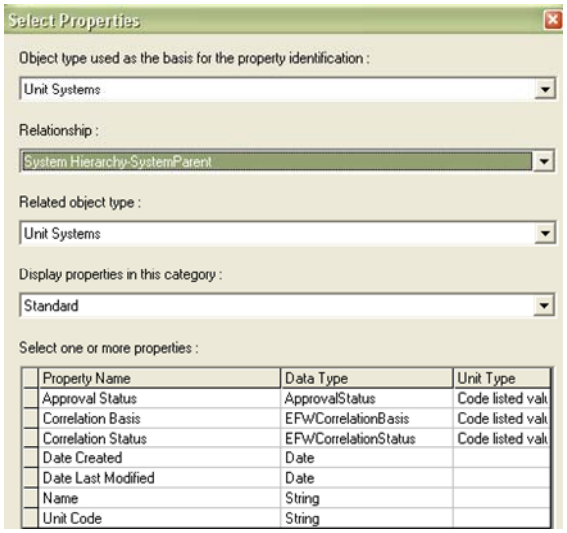


System Entity Data Model

Accessing Relationships from COM+

Graphical User Interface

- Provide access to properties of related objects



System Entity Data Model

Accessing Relationships from COM+

Label Editor generates XML files

- Use XML tags to access to related objects

```
<RETURNED_PROPERTY Name="ParentName" SQLType="BStr">
   <PATHS>
      <PATH
          SourceType="IJSystemChild"
          DestinationInterface="IJNamedItem"
          DestinationProperty="Name"
          Concatenate="No"
          PathSeparator="">
         <STROKES>
            <STROKE
                  Interface="IJSystemChild"
                  RelationCollection="SystemParent"
                  Recursive="No"
                  Filter="First"
                  IsVirtualRelationship="No" />
         </STROKES>
      </PATH>
   </PATHS>
</RETURNED_PROPERTY>
```

System Entity Data Model

Accessing Relationships from Visual Basic

- Relation Helper Service
  - Provides access to related objects
  - To use this service you must know the relation collection name and the interface name

Interface name: IJSystemChild

Relation collection name: SystemParent

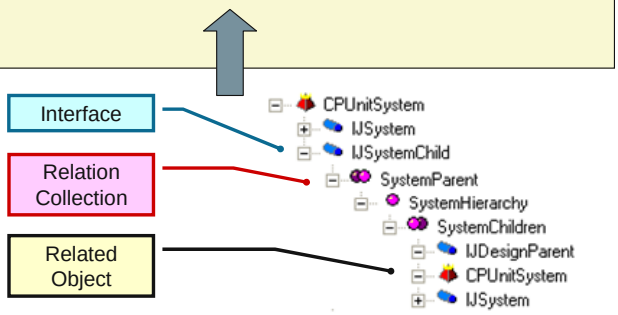
CollectionRelations(interfaceID, collectionName As String) As Object

System Entity Data Model

Accessing relationships from Visual Basic

Example:

```
Dim oRelationHelper As IMSRelation.DRelationHelper
Dim oCollection As IMSRelation.DCollectionHelper
Set oRelationHelper = oObject
Set oCollection = oRelationHelper.CollectionRelations("IJSystemChild", "SystemParent")
Set oSystem = oCollection.Item(1)
```



System Entity Data Model

- Interfaces and Properties
  - User attributes are tied to interfaces
  - User can extent the data model by adding custom interfaces and properties.

Custom Interface sheet

Head	InterfaceName	CategoryName	AttributeName	AttributeUserName	Type	UnitsType	PrimaryUnits	OnPropertyPage	ReadOnly	SymbolParameter
Start										
	IJUAHoldStatus		HoldStatus	Hold Status	Char			TRUE	FALSE	
End										

CustomClassInterfaceList sheet

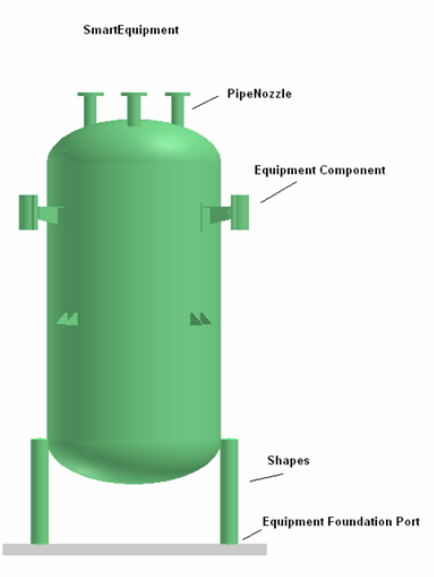
HEAD	ClassName	InterfaceName
Start		
	CPPipingSystem	IJUAHoldStatus
End		

Equipment Data Model

- Business Objects Defined in Equipment Application

First Class Business Objects

- CPSmartEquipment
- CPEquipmentComponent
- CPShape
- CPPrismaticShape
- CPUAImportedShapeOcc
- CPPipeNozzle
- CPCableTrayNozzle
- CPConduitNozzle
- CPCableNozzle
- CPHvacNozzle
- CPEqpFoundationPort



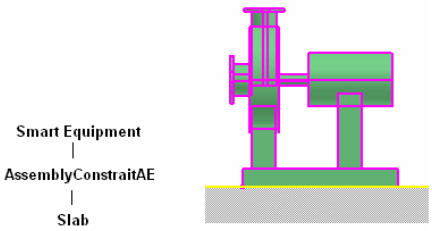
Equipment Data Model

- Business Objects Defined in Equipment Application

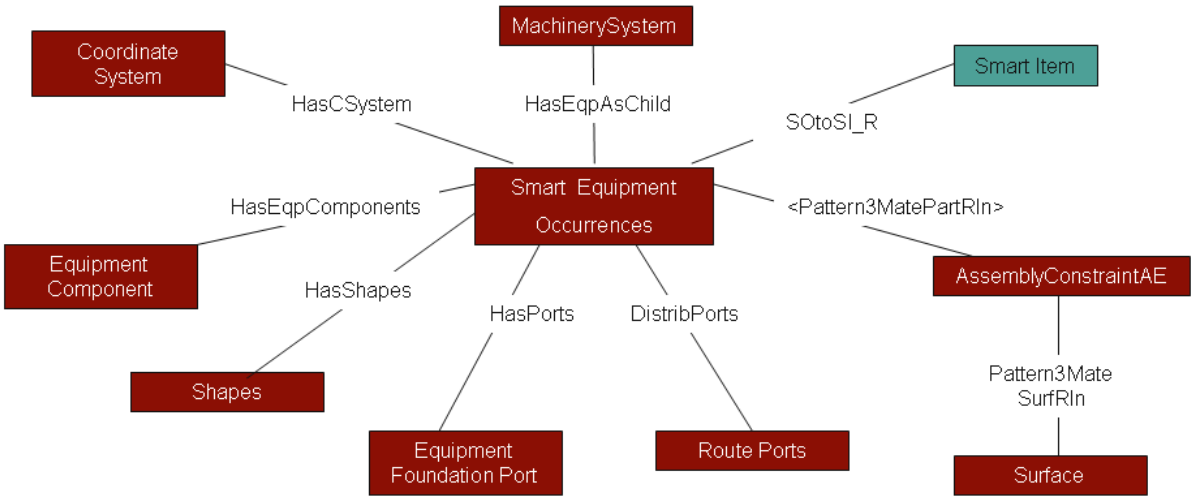
Non-First Class Business Objects

- CPAssemblyConstraintAE
- CPNozzleOrientation
- CPPipeNozzlePH
- CPCableTrayNozzlePH
- CPConduitNozzlePH
- CPCableNozzlePH
- CPHvacNozzlePH
- CPEqpFoundationPortPH

  Port Placeholder is a persistent object that holds the information about the actual port.

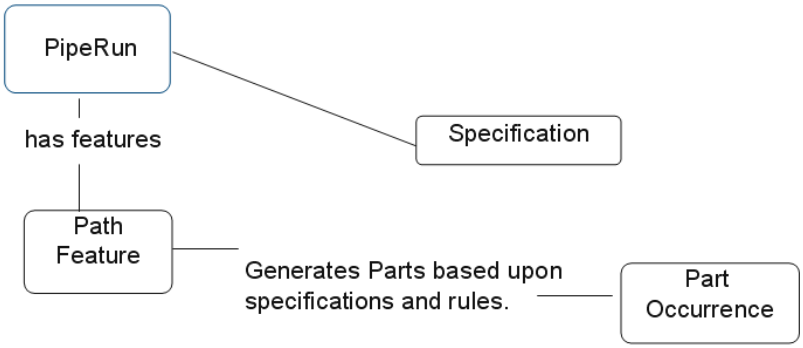


Equipment Data Model



Piping Data Model

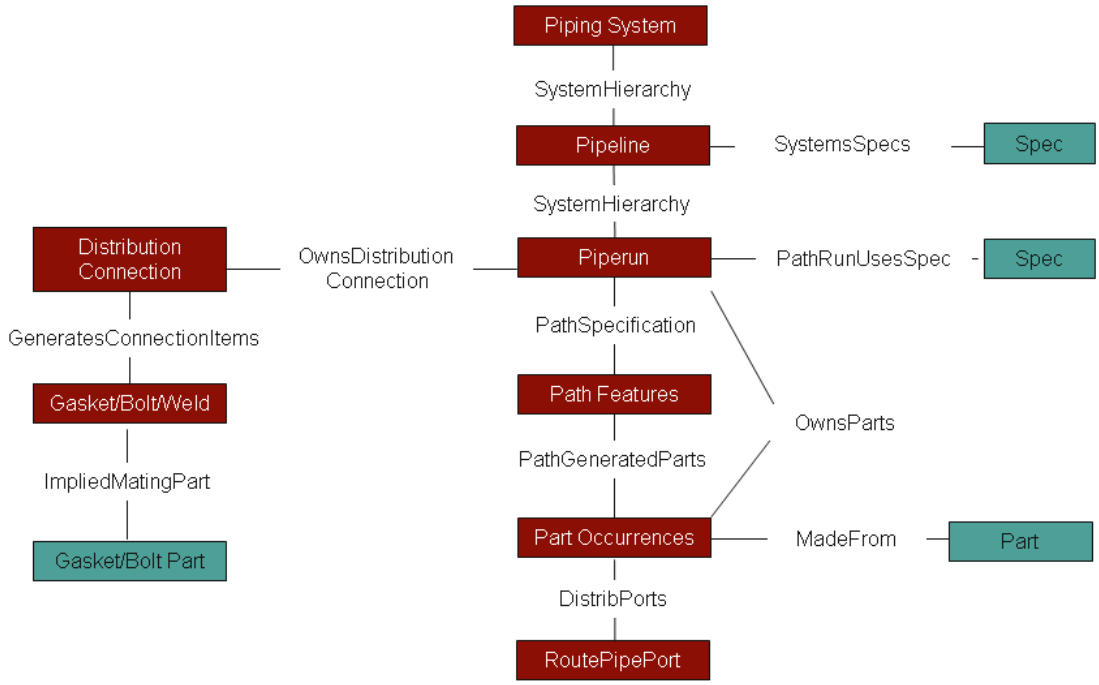
- The routing model is specification driven
  - It follows rules defined by the piping specifications.
  - It uses predefined catalog parts from Reference Data to define the part occurrences



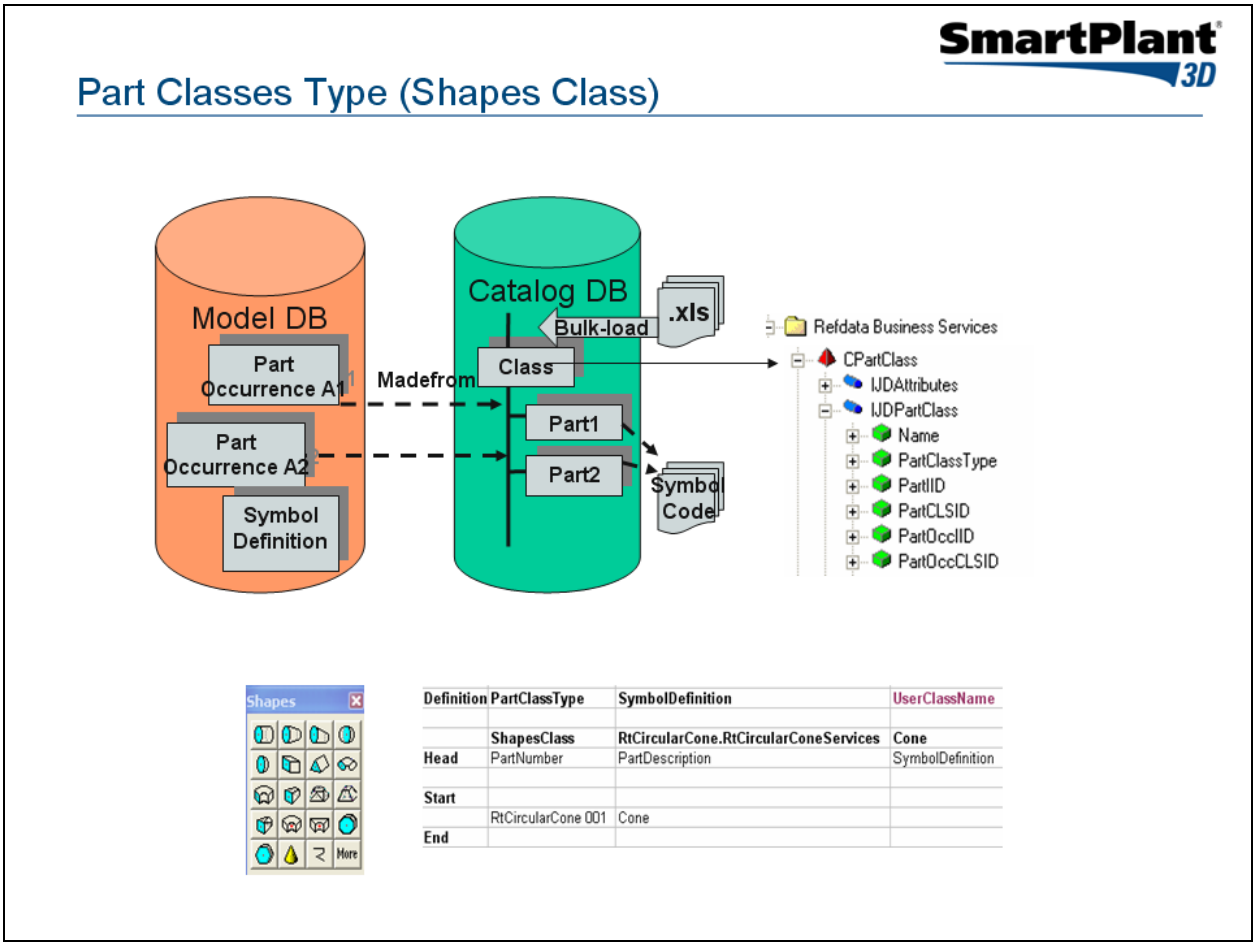
Piping Data Model

- Feature Based Model
- A Path Feature provides Geometry:
  - Path center line information: Start/End positions, Turns (Change of direction), etc
  - InLine path Features have their centerline making up a PathLeg and provide flow.
  - OffLine Path Features do not affect center line but may have AlongLeg behavior.
- A Path Feature provides Connectability:
  - Logically connect to equipment
  - Branch from any feature
  - Provide attachment connections to pipe supports.
- A Path Feature provides Functionality:
  - Capability to move geometry in 3D space with specific behaviors
  - Generates part occurrences based on specifications

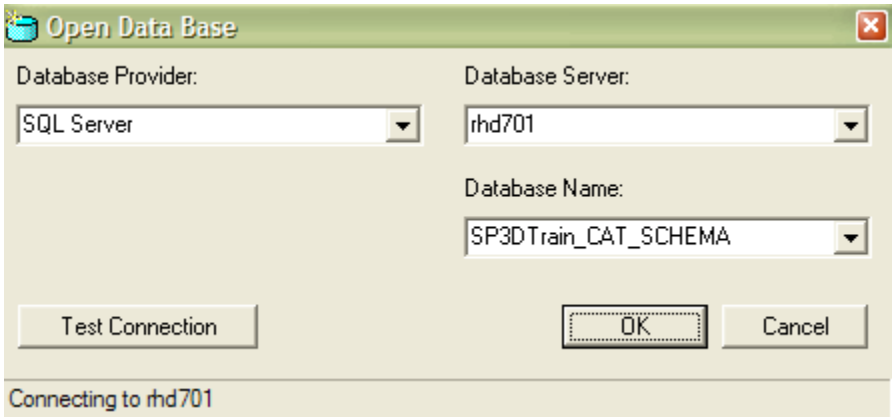
Piping Data Model



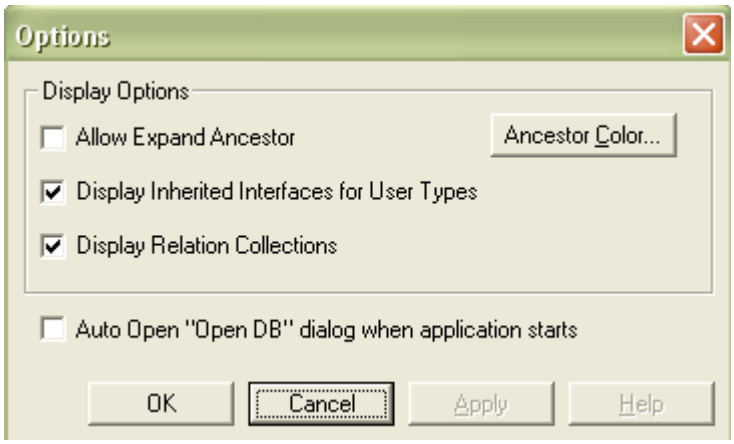
Lab 1: Create a query that returns all part classes of type ShapesClass defined in the catalog database



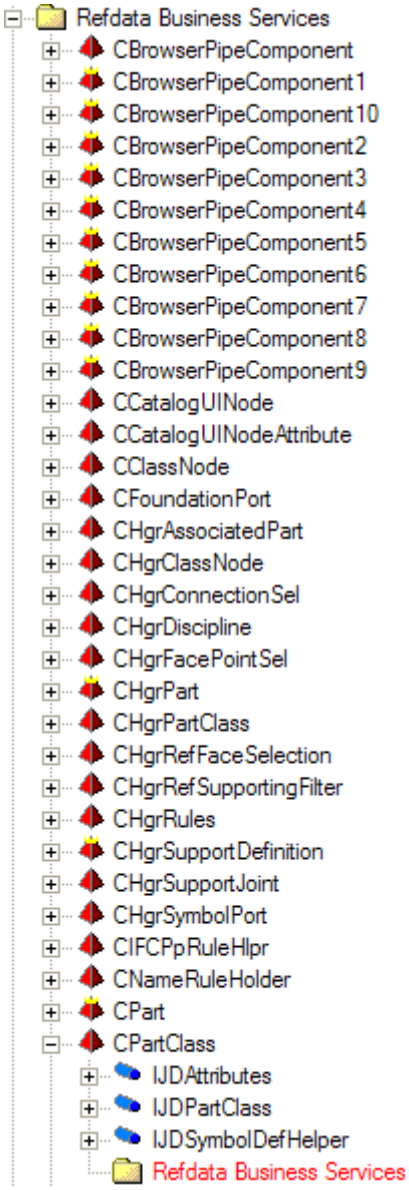
1. Open the SP3D Schema Browser and point to a catalog schema.



2. Select View -> Options to open the option dialog box. Enable the check box to displays Relation Collections.



3. Exit the SP3D Schema Browser and re-open it to read the change. We are interested in query part classes, thus we must start our navigation at Ref Data Business Services.



4. Expand CPartClass node. The tool shows a list of interfaces that are implemented by CPartClass. Since we are looking for the name of a part class, let us expand IJDPartClass.

5. Clicking on the PartClassType property in the tree view will show information about the selected item in the detail view. The DBViewName corresponding to IJDPartClass is PartClassType.

Properties of Attribute: PartClassType

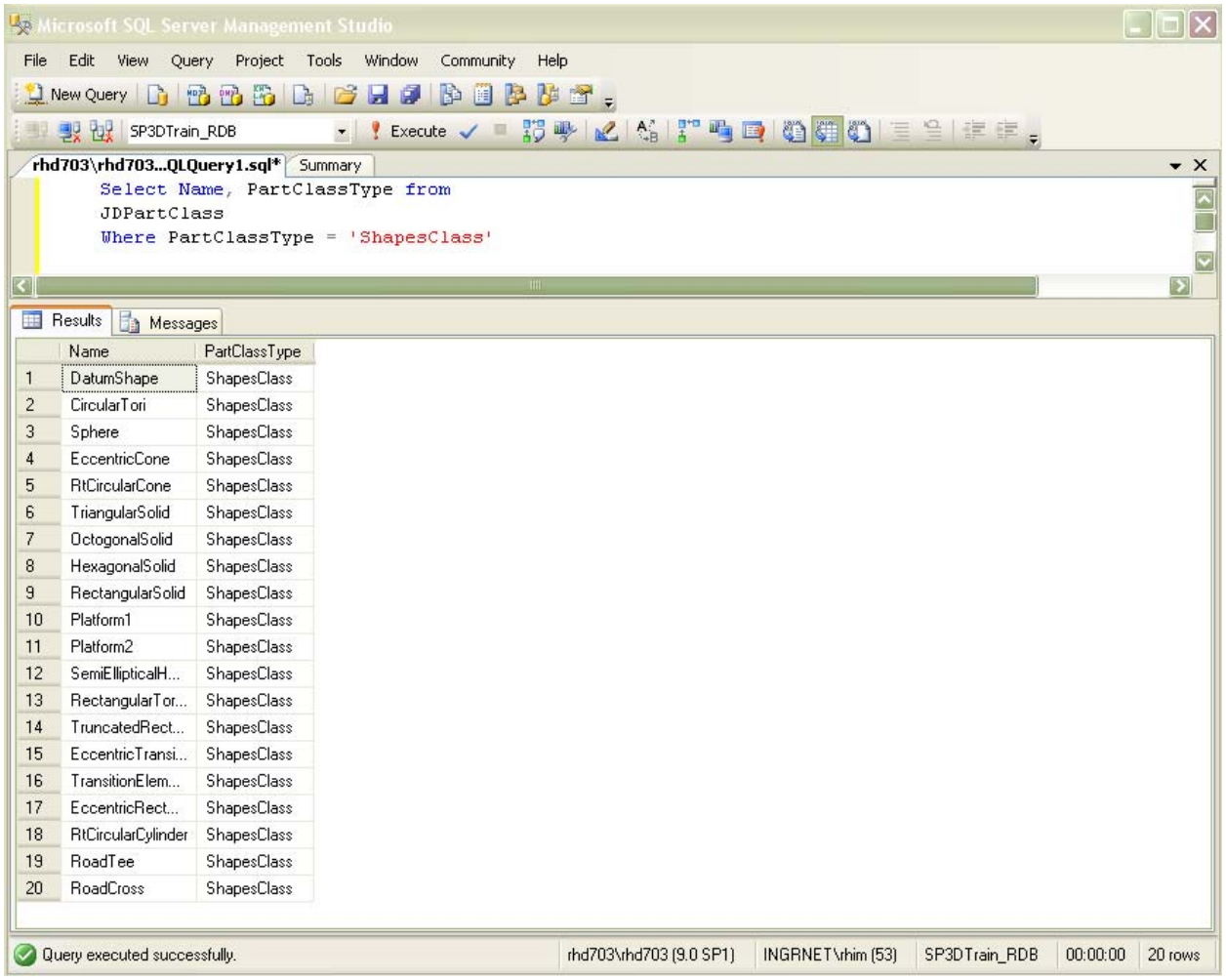
Name	Data
Name	PartClassType
UserName	Part Class Type
DBColumnName	PartClassType
OID	{2D90662E-67E6-11D4-B285-00104BCC2DC1}
Type	[1] - Char
CodeListTable	
ReadOnly	False
UnitsType	[0] - Undefined
IsValueRequired	False
ComplexTypeID	
UserFlags	[1] - Defined in Rose
Is Supported by Interface	IJDPartClass

6. To search for part classes in the catalog database, we must execute a SQL query that searches for all entries in the view PartClassType. We can do this using a SELECT statement on the report database. The SELECT query is as follows:

   ```
   Select Name, PartClassType from
   JDPartClass
   Where PartClassType = 'ShapesClass'
   ```

   This will return all part classes of type ShapesClass in the catalog database.

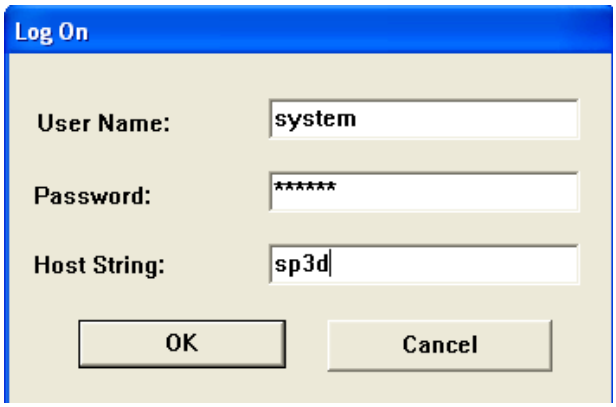
   If you are using Microsoft SQL 2005 to host the SP3D databases, then you can use Microsoft SQL Server Management Studio to run the SQL query. Set the report database to be the active database when running the query.



If you are using Oracle 10g to host the SP3D databases, then you can use SQL plus to run the SQL query:

Open Oracle SQL Plus or Oracle SQL Developer from the Start Menu.

Log on using the following data:

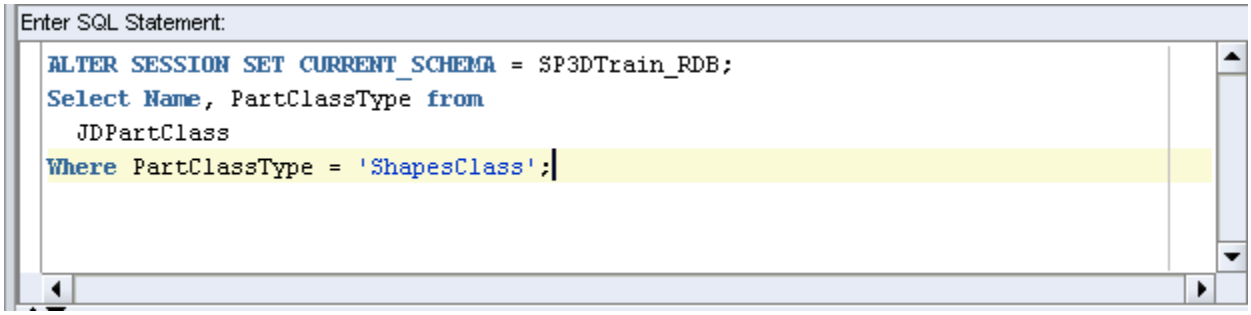


Note: Ask your instructor for the system user password.

From the SQL Command prompt, type the lines as shown here:

ALTER SESSION SET CURRENT_SCHEMA = SP3DTrain_RDB;

Select Name, PartClassType from
JDPartClass
Where PartClassType = 'ShapesClass';

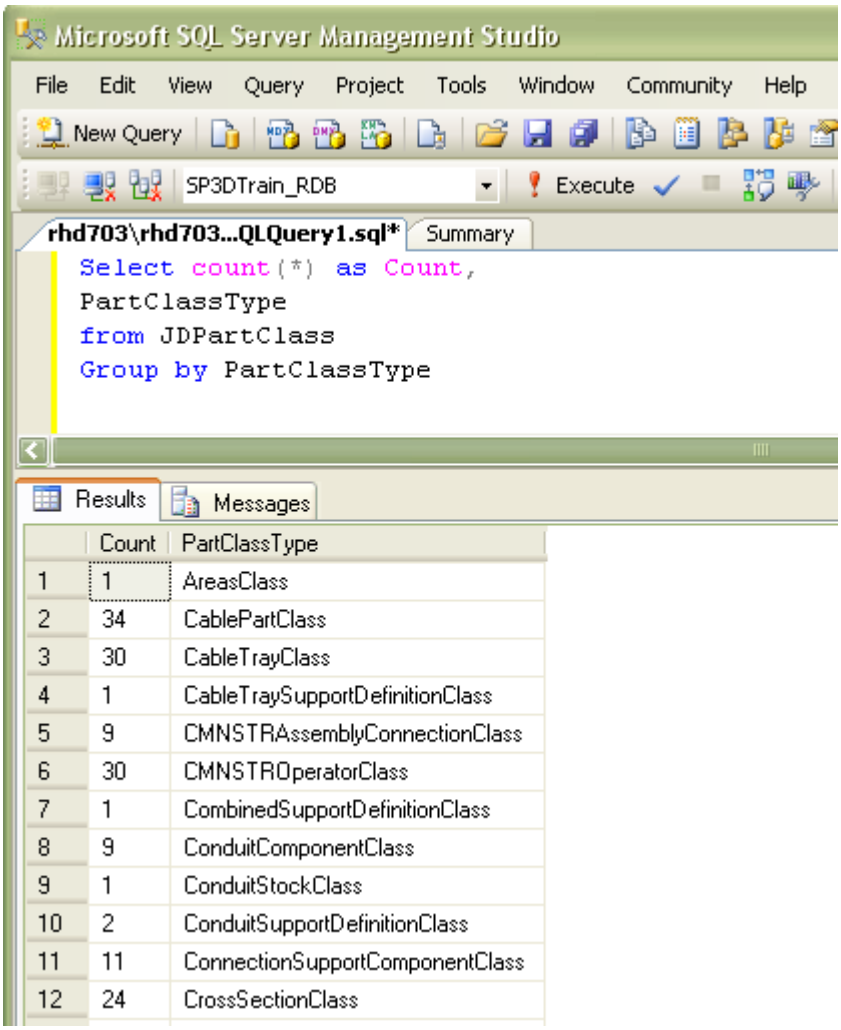


Lab 2: Create a query to find out the total number of part classes in the catalog database

1. Use the “SQL Group by” clause and the aggregate function “Count(*)” to get the total number of part classes in the catalog database. We can do this using a SELECT statement on the report database. The SELECT query is as follows:

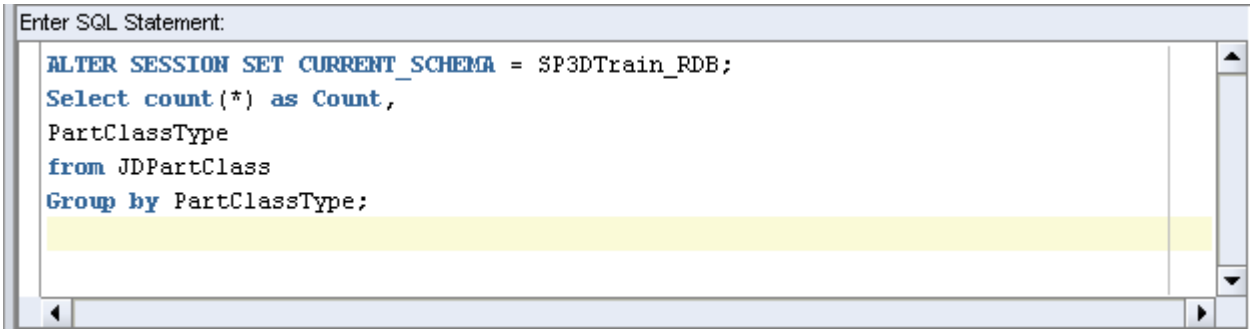
   Using Microsoft SQL Server Management Studio:

   ```
   Select count(*) as Count,
   PartClassType
   from JDPartClass
   Group by PartClassType
   ```



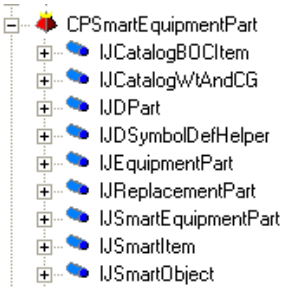
   Using Oracle SQL Plus or Oracle SQL Developer, the SELECT query is as follows:

```
ALTER SESSION SET CURRENT_SCHEMA = SP3DTrain_RDB;
    Select count(*) as Count,
    PartClassType
    from JDPartClass
    Group by PartClassType;
```



Lab 3: Create a query to list all smart equipment parts in the catalog database

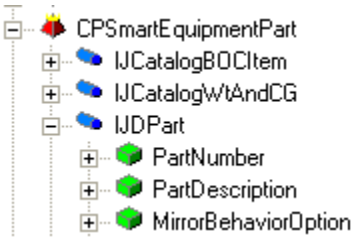
1. We are interested in query equipment parts defined in the catalog, thus we must start our navigation at Ref Data Business Services.



2. Expand Smart Equipment Part node. The tool shows a list of interfaces that are implemented by Smart Equipment Part. Thus to search for all equipment parts in the catalog database, we must execute a SQL query that searches for all entries in the view JSmartEquipmentPart. We can do this using a SELECT statement on the report database. The SELECT query is as follows:

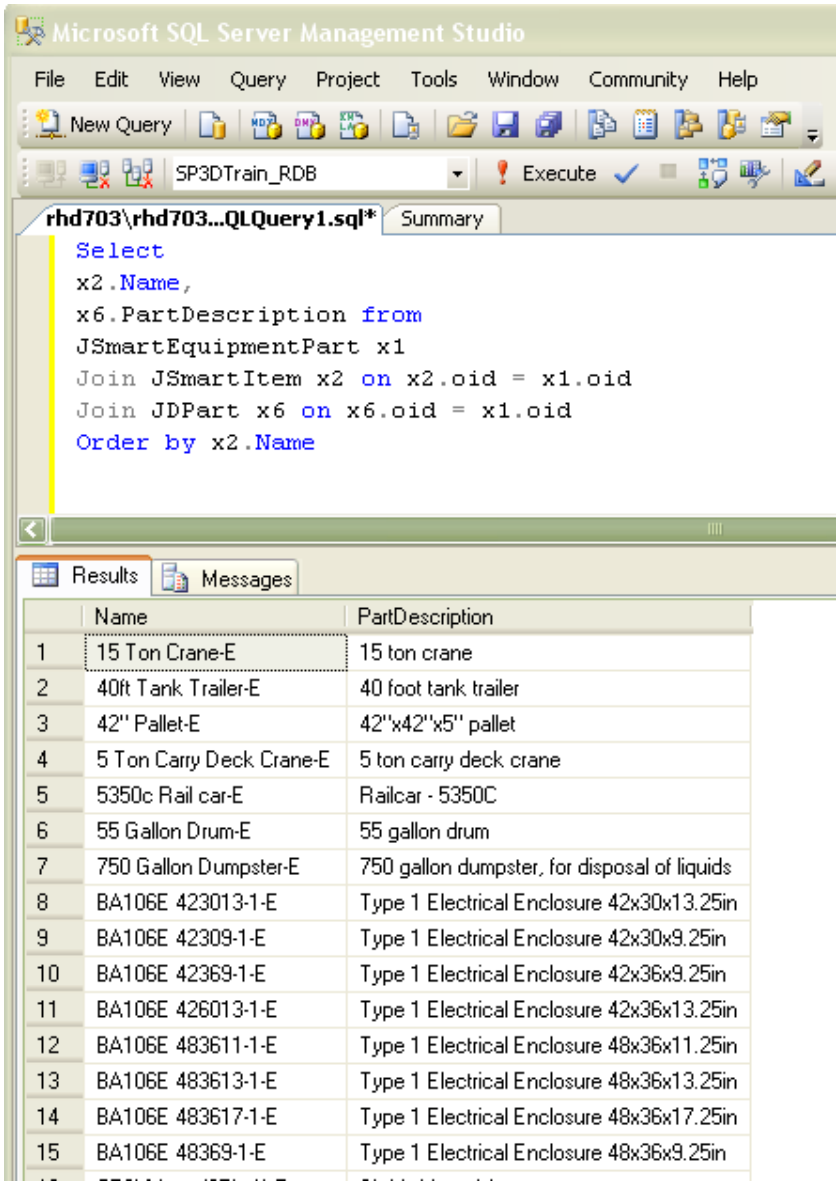
   Select * from JSmartEquipmentPart

3. We are also interested to get the description and the name of the equipment part.
4. This is done by using the "SQL JOIN" clause on the views that return the equipment name and the equipment description. Use the "SQL Order by" clause to sort the equipment parts by their name.

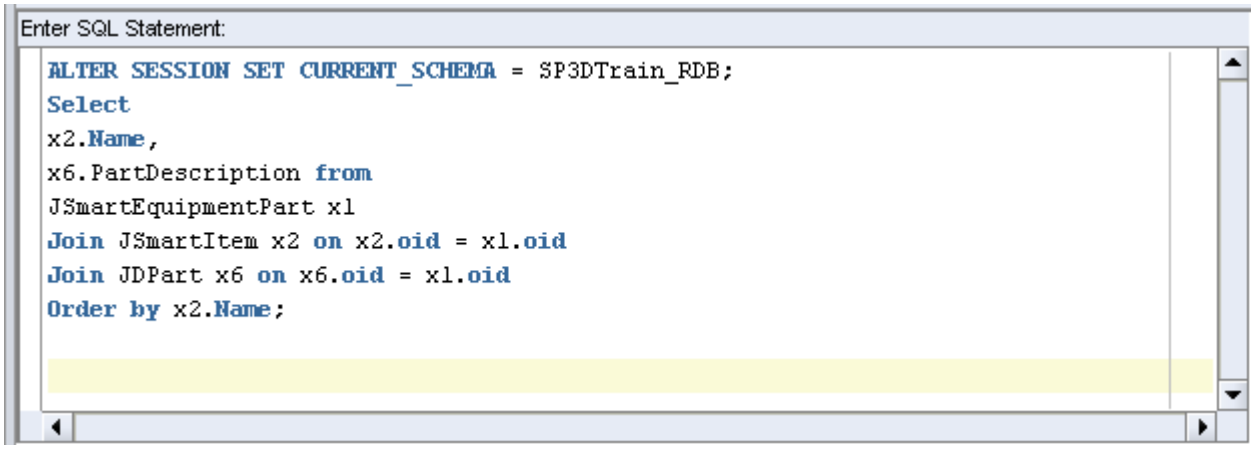


5. Using Microsoft SQL Server Management Studio, the SELECT query is as follows:
   :

```
Select
x2.Name,
x6.PartDescription from
JSmartEquipmentPart x1
Join JSmartItem x2 on x2.oid = x1.oid
Join JDPart x6 on x6.oid = x1.oid
Order by x2.Name
```



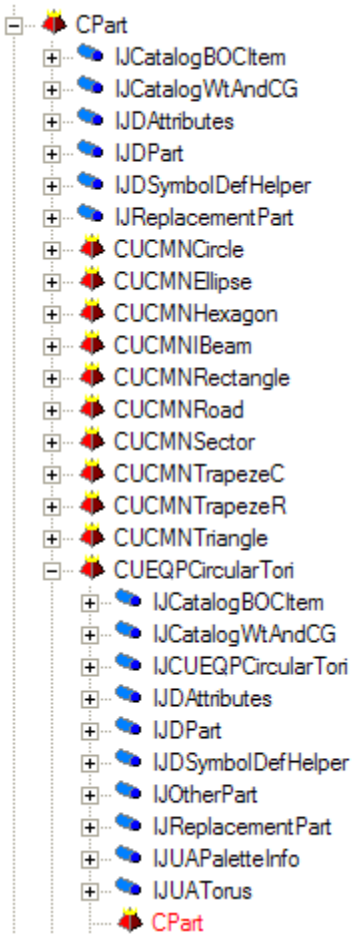
Using Oracle SQL Plus or Oracle SQL Developer, the SELECT query is as follows:



Lab 4: List all equipment shapes located in the palette

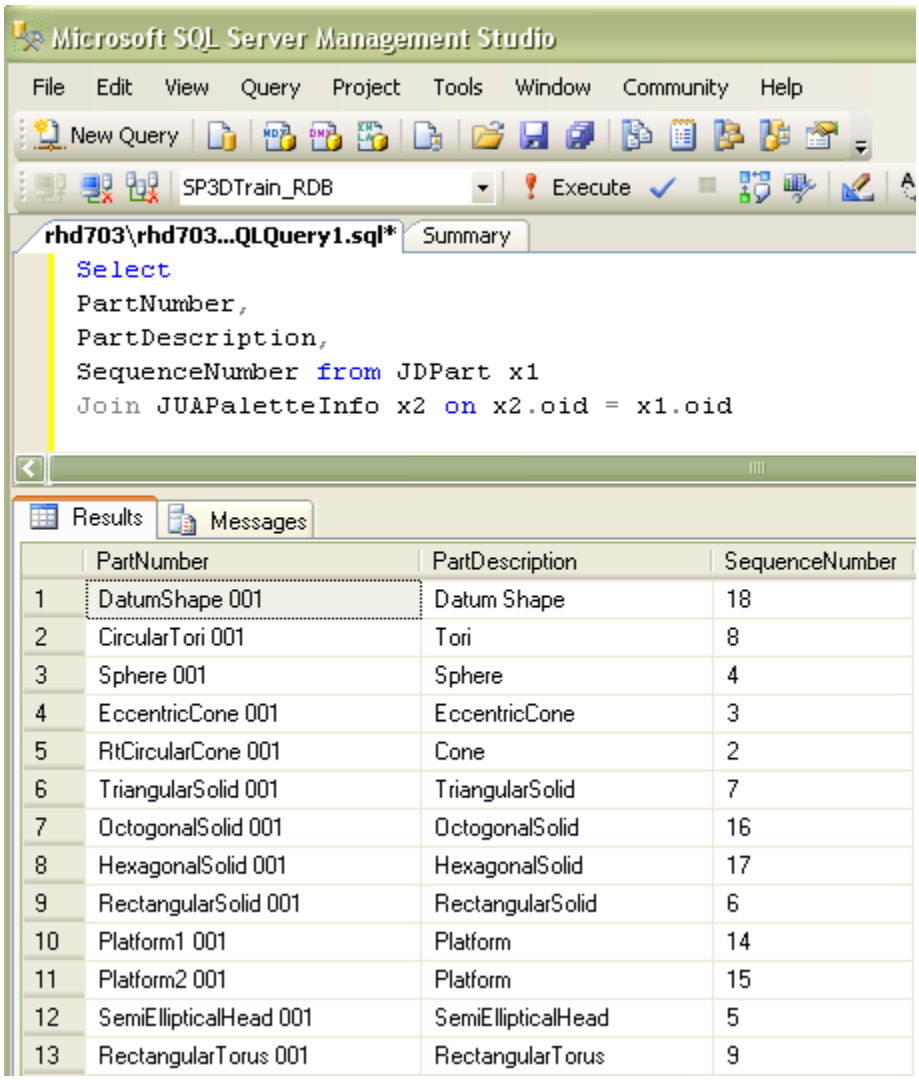
1. We are interested in query equipment shapes defined in the catalog, thus we must start our navigation at Ref Data Business Services. Equipment shapes are parts in the catalog. Thus, we must begin our hunt under the CPart folder.

2. Expand CPart node. The tool shows a list of Equipment shape part classes. Expand one of them and notice that if a part class is located in the palette, then it must implement the IJUAPaletteInfo



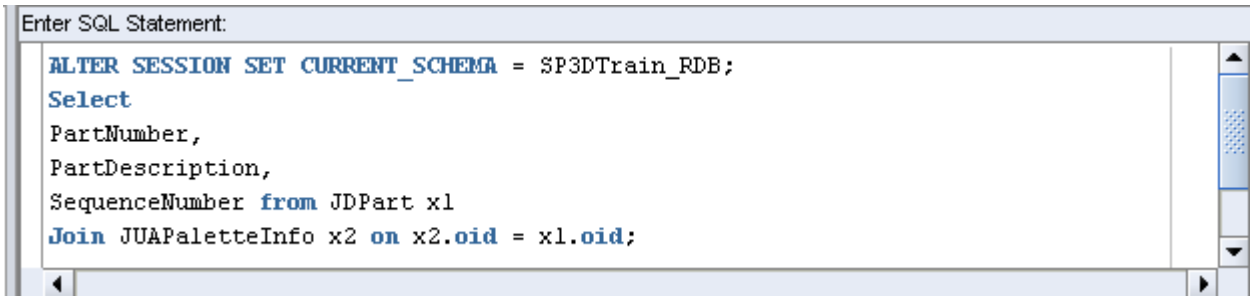
Using Microsoft SQL Server Management Studio, the SELECT query is as follows:

```
Select
PartNumber,
PartDescription,
SequenceNumber from JDPart x1
Join JUAPaletteInfo x2 on x2.oid = x1.oid
```



	PartNumber	PartDescription	SequenceNumber
1	DatumShape 001	Datum Shape	18
2	CircularTori 001	Tori	8
3	Sphere 001	Sphere	4
4	EccentricCone 001	EccentricCone	3
5	RtCircularCone 001	Cone	2
6	TriangularSolid 001	TriangularSolid	7
7	OctogonalSolid 001	OctogonalSolid	16
8	HexagonalSolid 001	HexagonalSolid	17
9	RectangularSolid 001	RectangularSolid	6
10	Platform1 001	Platform	14
11	Platform2 001	Platform	15
12	SemiEllipticalHead 001	SemiEllipticalHead	5
13	RectangularTorus 001	RectangularTorus	9

Using Oracle SQL Plus or Oracle SQL Developer, the SELECT query is as follows:

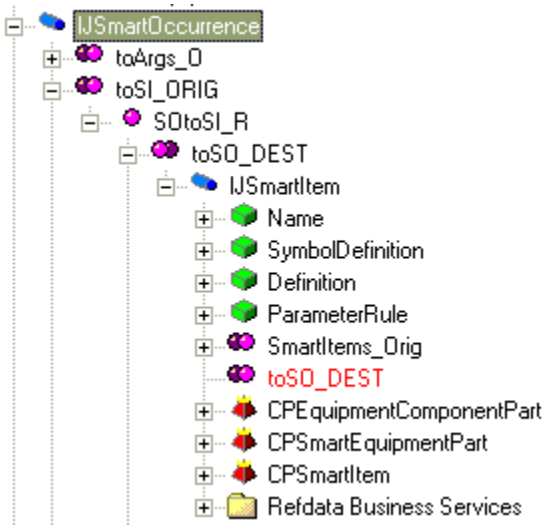


Lab 5: List all equipments located in the model with its corresponding part name from the catalog database

1. We are interested in query Smart Equipment occurrences located in the model, thus we must start our navigation at Equipment Business Services under the CPSmartEquipment folder.

2. Expand CPSmartEquipment node. The tool shows a list of interfaces that are implemented by Smart Equipment. Since we are looking for a relation to the catalog, let us expand IJSmartOccurrence (which is the interface implemented by all smart occurrences).

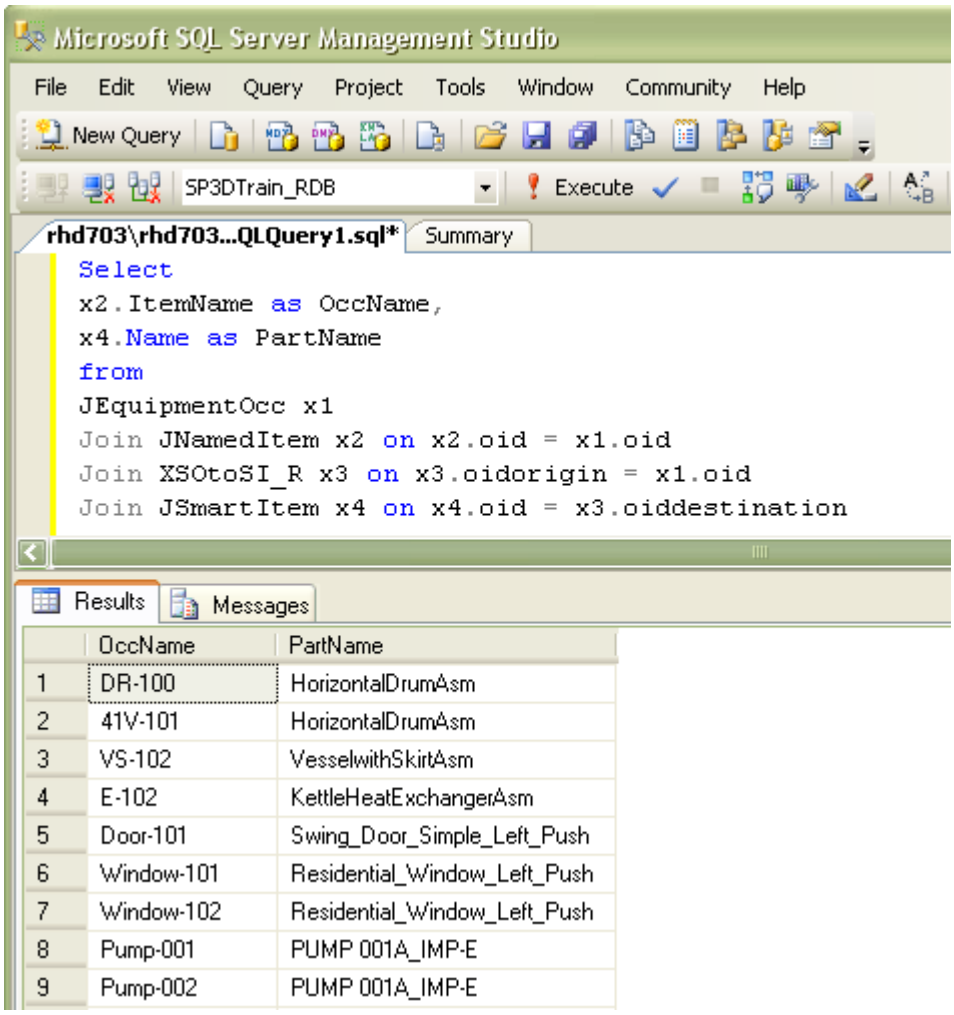
3. You will see a pink bubble that shows the toSI_ORIG relation collection. Expand the node further and you will find the property you are looking for on an interface at the other end of the relationship.



4. We are also interested to get the name of the smart equipment occurrence. We can use the IJNamedItem interface which provides the object name.

5. Using Microsoft SQL Server Management Studio, the SELECT query is as follows:

```
Select
x2.ItemName as OccName,
x4.Name as PartName
from
JEquipmentOcc x1
Join JNamedItem x2 on x2.oid = x1.oid
Join XSOtoSI_R x3 on x3.oidorigin = x1.oid
Join JSmartItem x4 on x4.oid = x3.oiddestination
```



Using Oracle SQL Plus or Oracle SQL Developer, the SELECT query is as follows:

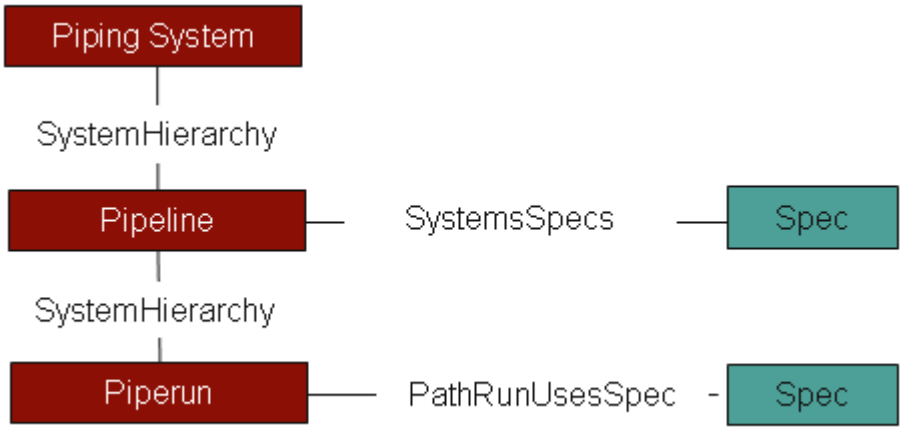
Enter SQL Statement:

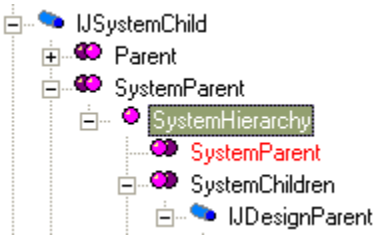
```
ALTER SESSION SET CURRENT_SCHEMA = SP3DTrain_RDB;
Select
x2.ItemName as OccName,
x4.Name as PartName
from
JEquipmentOcc x1
Join JNamedItem x2 on x2.oid = x1.oid
Join XSOtoSI_R x3 on x3.oidorigin = x1.oid
Join JSmartItem x4 on x4.oid = x3.oiddestination;
```

Lab 6: List all pipe runs and pipeline names located in the model database

Hints:

- We must begin our hunt under the Common Route Business Service folder
- Use the IJSystemChild to get the parent object. In order for an object to participate in the System Hierarchy, it must implement the IJSystemChild and establish a relationship to a design parent
- Find the JRtePipeRun in the Common Route Business Service folder





Solution:

Using Microsoft SQL Server Management Studio, the SELECT query is as follows:

```
Select
x4.ItemName as PipeRunName,
x3.ItemName as Parent_System
from JRtePipeRun x1
Join JNamedItem x4 on x4.oid = x1.oid
Join  XSystemHierarchy x2 on x2.oiddestination = x1.oid
Join JNamedItem x3 on x3.oid = x2.oidorigin
```

Microsoft SQL Server Management Studio

File Edit View Query Project Tools Window Community Help

New Query | SP3DTrain_RDB | Execute

rhd703\rhd703...QLQuery1.sql* | Summary

```
Select
x4.ItemName as PipeRunName,
x3.ItemName as Parent_System
from JRtePipeRun x1
Join JNamedItem x4 on x4.oid = x1.oid
Join  XSystemHierarchy x2 on x2.oiddestination = x1.oid
Join JNamedItem x3 on x3.oid = x2.oidorigin
```

Results | Messages

	PipeRunName	Parent_System
1	U01-4-P-0101-1C0101	1001-P
2	U01-4-P-0111-1C0101	1001-P
3	U01-10-P-0005-1C0031	1001-P
4	U01-8-P-0004-1C0031	1001-P
5	U01-8-P-0001-1C0031	1001-P
6	U01-10-P-0003-1C0031	1001-P
7	U03-0.75-W-0012-1C0031	300-W
8	U03-3-W-0011-1C0031	300-W
9	U03-10-W-0002-1C0031	300-W
10	U03-8-W-0001-1C0031	300-W
11	U03-1-W-0001-1C0031	301-W
12	U03-6-W-0001-1C0031	303-W
13	U01-6-P-0002-1C0031	1002-P
14	U01-6-P-0001-1C0031	1002-P

Using Oracle SQL Plus or Oracle SQL Developer, the SELECT query is as follows:

Enter SQL Statement:

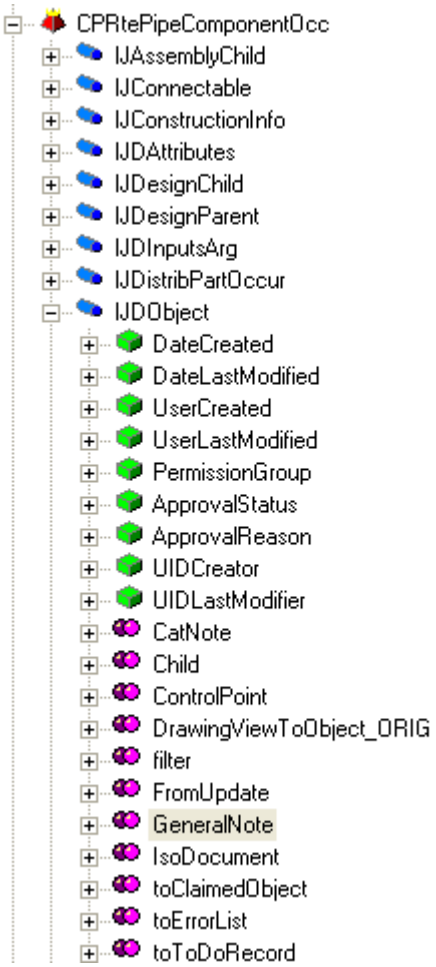
```
ALTER SESSION SET CURRENT_SCHEMA = SP3DTrain_RDB;
Select
x4.ItemName as PipeRunName,
x3.ItemName as Parent_System
from JRtePipeRun x1
Join JNamedItem x4 on x4.oid = x1.oid
Join  XSystemHierarchy x2 on x2.oiddestination = x1.oid
Join JNamedItem x3 on x3.oid = x2.oidorigin;
```

Lab 7: List all object with notes in the model database

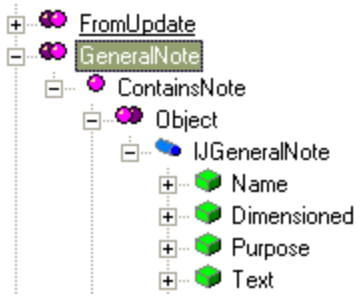
1. We must begin our hunt under the CommonRoute Business Services folder.

2. Expand Pipe Component occurrence node. The tool shows a list of interfaces that are implemented by pipe component occurrence. Since we are looking for object to note relation, let us expand IJDObject (which is the interface which defines that a Pipe component is an ‘object’).

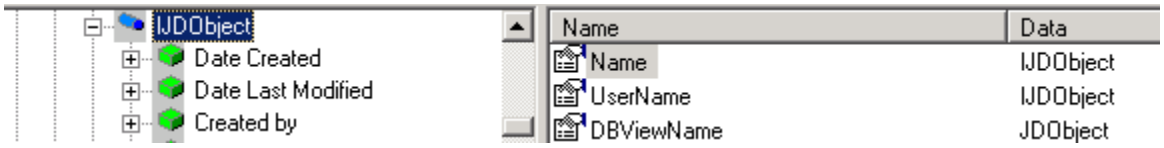
3. You will see a pink bubble that shows the GeneralNote relation collection.



4. Expand the node further and you will find the property you are looking for on an interface at the other end of the relationship.



5. Click on IJDObject to see that the DBViewName corresponding to it is JDObject in the detail view.

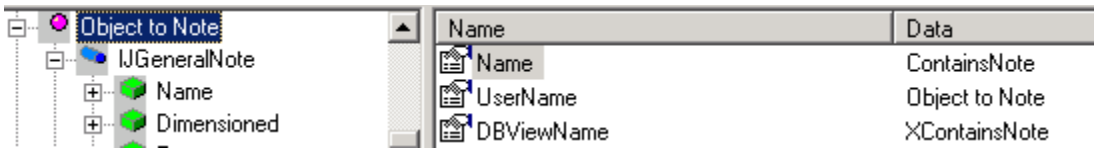


6. Thus to search for all ‘object’s in the database, we must execute a SQL query that searches for all entries in the view JDObject. We can do this using an SQL query on the Report database.

   Select * from JDObject

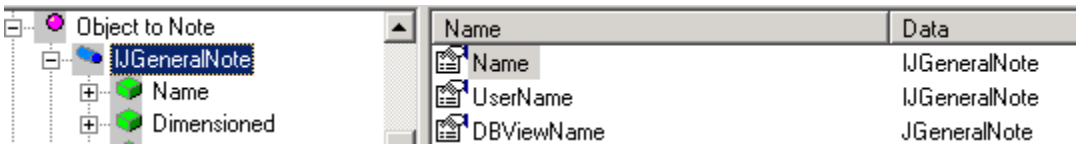
   This will return a list of all objects in the database.

7. However, we are interested in all objects that have a relationship with a note. Thus let us make a query for all relationships between objects and notes. This is done using the view corresponding to the relationship.



   Select * from XContainsNote

8. Finally we will search for all notes in the database using the following query



   Select * from JGeneralNote

9. To find the objects which are related to notes, we will make a join between the queries as follows

   Select * from JDObject
   Join XContainsNote on JDObject.oid = XcontainsNote.Oidorigin
   Join JGeneralNote on JGeneralNote.oid = XcontainsNote.OidDestination

10. Using the “SQL JOIN” clauses, we will get a list of all the objects (and only the objects) which has notes associated with them.

11. To simplify the query, we can use aliases for the view names

    Select * from JDObject x1
    Join XContainsNote x2 on x2.Oidorigin = x1.oid
    Join JGeneralNote x3 on x3.oid = x2.OidDestination

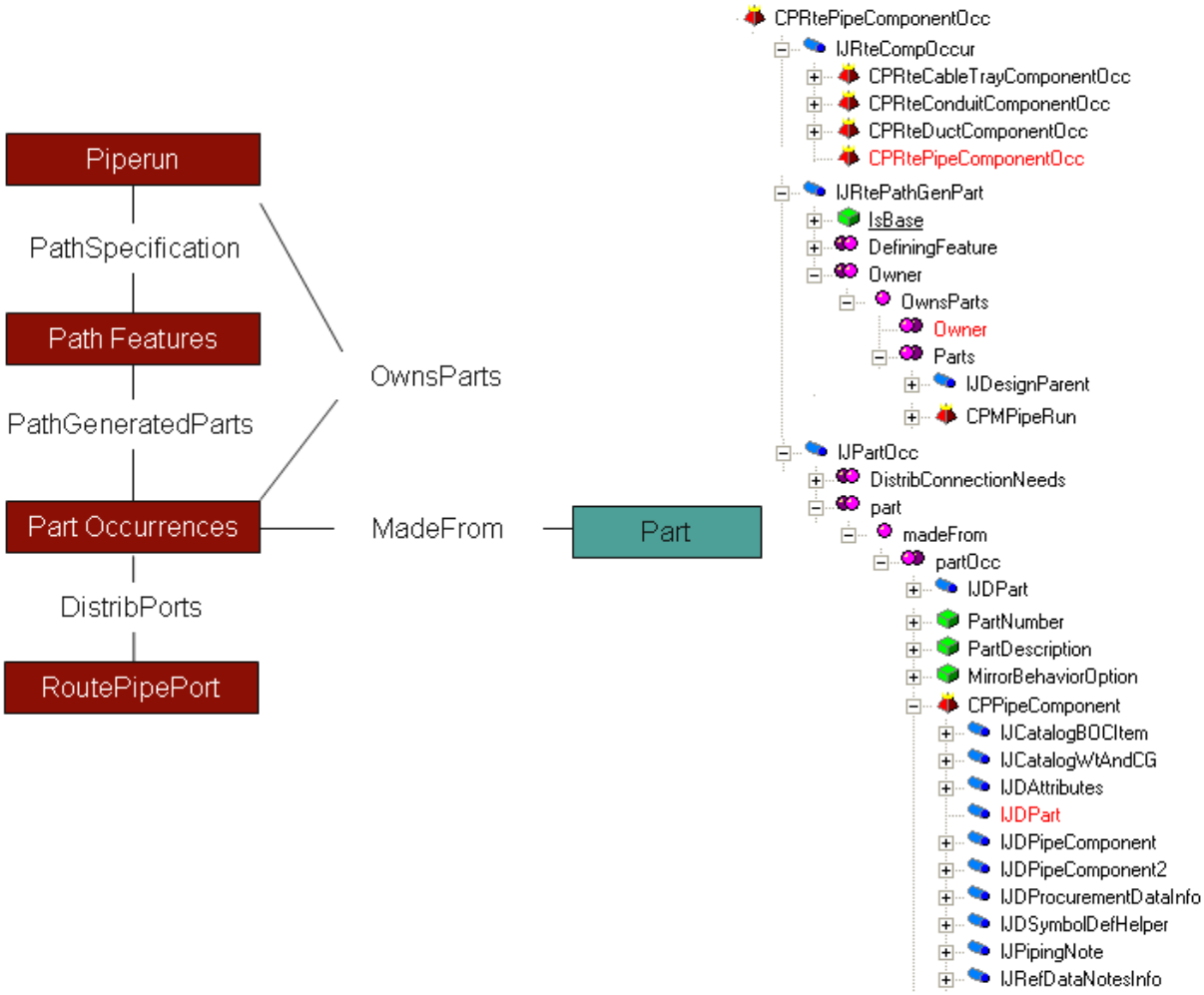
12. Change the query to return only the Note text column. Therefore, the SELECT query is as follows:

    Select x3.Text from JDObject x1
    Join XContainsNote x2 on x2.Oidorigin = x1.oid
    Join JGeneralNote x3 on x3.oid = x2.OidDestination

Lab 8: List all pipe component occurrences in the model database per PipeRun

Hints:

- We must begin our hunt under the Common Route Business Service folder
- Find the JRteCompOccur in the Common Route Business Service folder
- Use the MadeFrom relation to find the part in the catalog
- Use the IJDPipeComponent interface to get the Industry Commodity Code of the part occurrence
- Use the Run to Part (OwnParts) relation to get to the PipeRun object. This relation is provided by IJRtePathGenPart interface
- Use the "SQL Group by" clause and the aggregate function "Count(*)" to get the total number of part occurrences in the model database



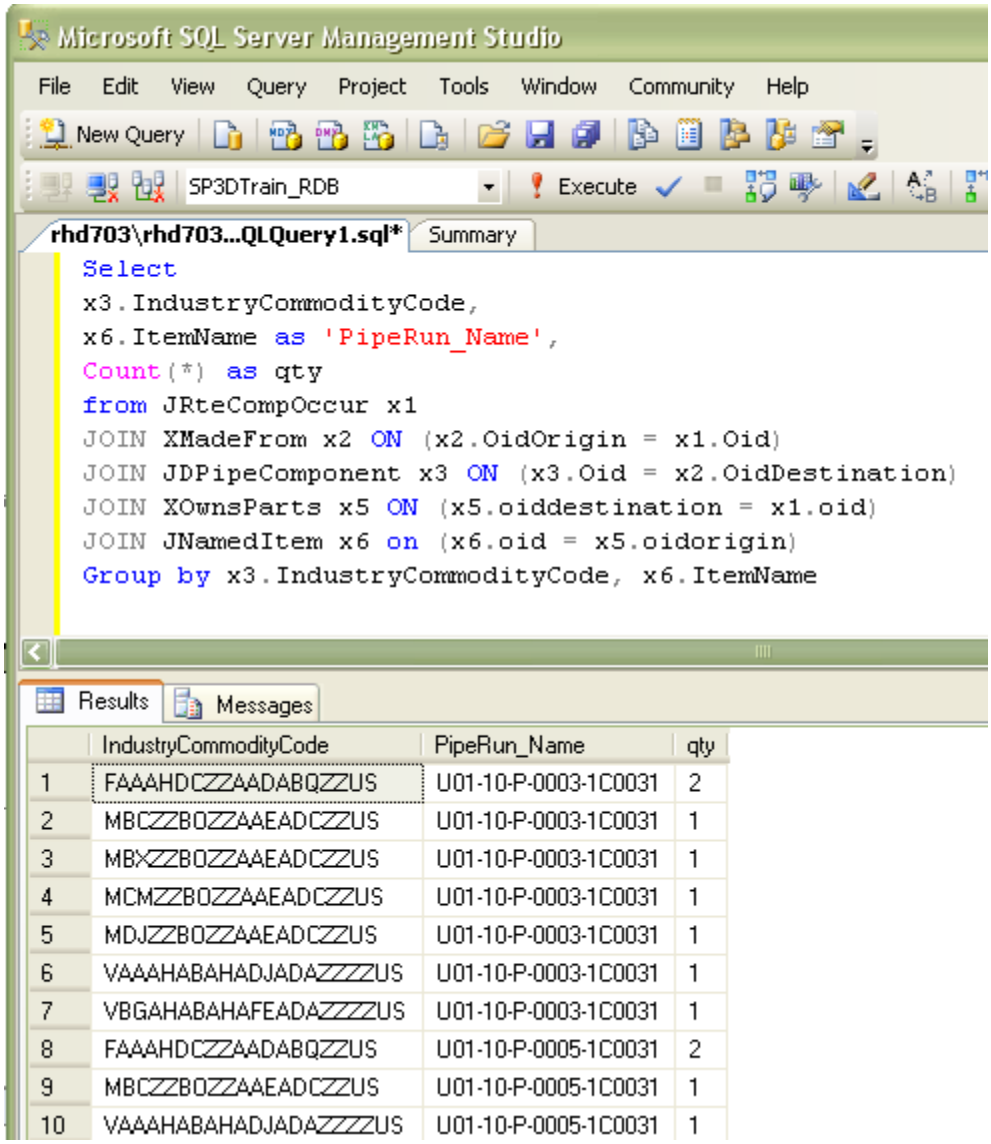
Results | Messages

	IndustryCommodityCode	PipeRun_Name	qty
1	FAAAHDCZZAADABQZZUS	U01-10-P-0003-1C0031	2
2	MBCZZBOZZAAEADCZZUS	U01-10-P-0003-1C0031	1
3	MBXZZBOZZAAEADCZZUS	U01-10-P-0003-1C0031	1
4	MCMZZBOZZAAEADCZZUS	U01-10-P-0003-1C0031	1
5	MDJZZBOZZAAEADCZZUS	U01-10-P-0003-1C0031	1
6	VAAAHABAHADJADAZZZZUS	U01-10-P-0003-1C0031	1
7	VBGAHABAHAFEADAZZZZUS	U01-10-P-0003-1C0031	1
8	FAAAHDCZZAADABQZZUS	U01-10-P-0005-1C0031	2
9	MBCZZBOZZAAEADCZZUS	U01-10-P-0005-1C0031	1
10	VAAAHABAHADJADAZZZZUS	U01-10-P-0005-1C0031	1
11	VBGAHABAHAFEADAZZZZUS	U01-10-P-0005-1C0031	1
12	MCMZZBOZZAAEADCZZUS	U01-6-P-0001-1C0031	3
13	MDJZZBOZZAAEADCZZUS	U01-6-P-0001-1C0031	1
14	FAAAHDCZZAADABQZZUS	U01-8-P-0001-1C0031	1
15	FAAAHDCZZAADABQZZUS	U01-8-P-0004-1C0031	1
16	MELAWDFZZAEYABQZZUM	U02-1-P-0003-1C0031	1
17	MFJAWBVZZALVABQZZUM	U02-1-P-0003-1C0031	1

Solution:

```
Select
x3.IndustryCommodityCode,
x6.ItemName as 'PipeRun_Name',
Count(*) as qty
from JRteCompOccur x1
JOIN XMadeFrom x2 ON (x2.OidOrigin = x1.Oid)
JOIN JDPipeComponent x3 ON (x3.Oid = x2.OidDestination)
JOIN XOwnsParts x5 ON (x5.oiddestination = x1.oid)
JOIN JNamedItem x6 on (x6.oid = x5.oidorigin)
Group by x3.IndustryCommodityCode, x6.ItemName
```

Using Microsoft SQL Server Management Studio, the SELECT query is as follows:



	IndustryCommodityCode	PipeRun_Name	qty
1	FAAAHDCZZAADABQZZUS	U01-10-P-0003-1C0031	2
2	MBCZZBOZZAAEADCZZUS	U01-10-P-0003-1C0031	1
3	MBXZZBOZZAAEADCZZUS	U01-10-P-0003-1C0031	1
4	MCMZZBOZZAAEADCZZUS	U01-10-P-0003-1C0031	1
5	MDJZZBOZZAAEADCZZUS	U01-10-P-0003-1C0031	1
6	VAAAHABAHADJADAZZZZUS	U01-10-P-0003-1C0031	1
7	VBGAHABAHAFEADAZZZZUS	U01-10-P-0003-1C0031	1
8	FAAAHDCZZAADABQZZUS	U01-10-P-0005-1C0031	2
9	MBCZZBOZZAAEADCZZUS	U01-10-P-0005-1C0031	1
10	VAAAHABAHADJADAZZZZUS	U01-10-P-0005-1C0031	1

Using Oracle SQL Plus or Oracle SQL Developer, the SELECT query is as follows:

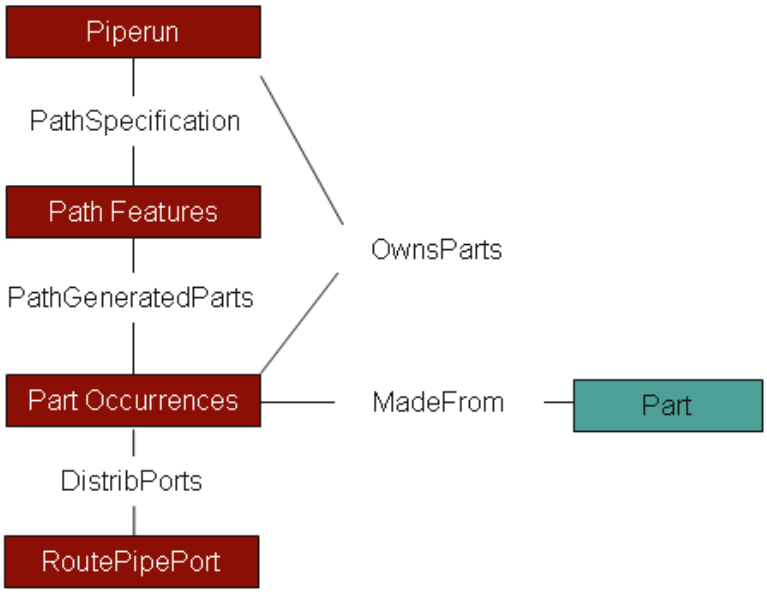
Enter SQL Statement:

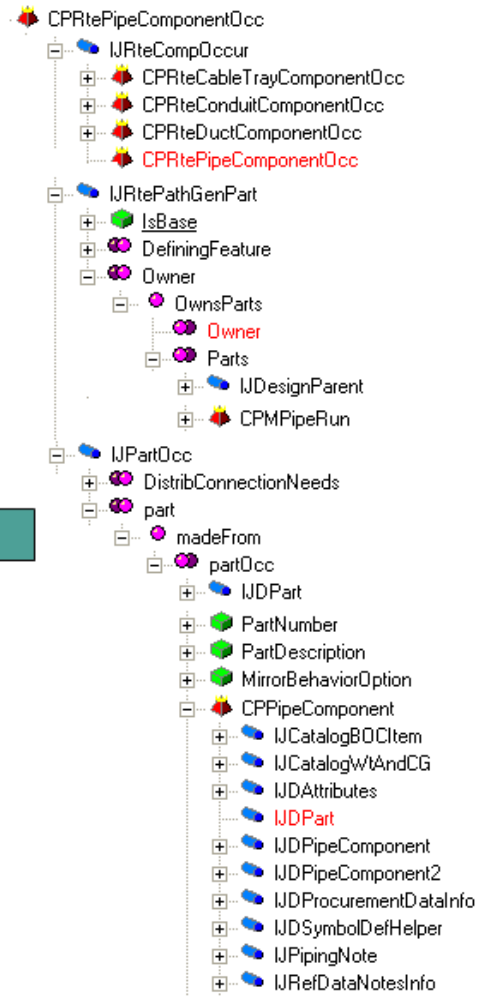
```
ALTER SESSION SET CURRENT_SCHEMA = SP3DTrain_RDB;
Select
x3.IndustryCommodityCode,
x6.ItemName as "PipeRun_Name",
Count(*) as qty
From JRteCompOccur x1
JOIN XMadeFrom x2 ON (x2.OidOrigin = x1.Oid)
JOIN JDPipeComponent x3 ON (x3.Oid = x2.OidDestination)
JOIN XOwnsParts x5 ON (x5.oiddestination = x1.oid)
JOIN JNamedItem x6 on (x6.oid = x5.oidorigin)
Group by x3.IndustryCommodityCode, x6.ItemName
```

Lab 9: List all valves occurrences located in the model per PipeRun

Hints:

- We must begin our hunt under the Common Route Business Service folder
- Use the MadeFrom relation to find the part in the catalog
- Use the IJDPipeComponent view to get the Industry Commodity Code and the Commodity Type of the part occurrence
- Use the Run to Part (OwnParts) relation to get to the PipeRun object. This relation is provided by IJRtePathGenPart interface





	IndustryCommodityCode	PipeRun_Name	CommodityType	qty
1	VAAAHABAHADJADAZZZZUS	U01-10-P-0003-1C0031	GAT	1
2	VAAAHABAHADJADAZZZZUS	U01-10-P-0005-1C0031	GAT	1
3	VAAAHABAHADJADAZZZZUS	U02-6-P-0002-1C0031	GAT	1
4	VAAAHABAHADJADAZZZZUS	U02-6-P-0004-1C0031	GAT	1
5	VAAAHABAHADJADAZZZZUS	U03-10-W-0002-1C0031	GAT	1
6	VAAAHABAHADJADAZZZZUS	U04-10-P-0002-1C0031	GAT	2
7	VAAAHABAHADJADAZZZZUS	U04-3-P-0005-1C0031	GAT	1

Solution:

```
Select
x3.IndustryCommodityCode,
x6.ItemName as 'PipeRun_Name',
x4.ShortStringValue as 'CommodityType',
count(*) as qty
from JRteCompOccur x1
Join XMadeFrom x2 ON (x2.OidOrigin = x1.Oid)
Join JDPipeComponent x3 ON (x3.Oid = x2.OidDestination)
Join CL_PipingCommodityType x4 ON (x4.ValueID = x3.CommodityType)
Join XOwnsParts x5 ON (x5.oiddestination = x1.oid)
Join JNamedItem x6 ON (x6.oid = x5.oidOrigin)
WHERE (x3.CommodityClass = 5)
Group by x3.IndustryCommodityCode, x6.ItemName, x4.ShortStringValue
```

Using Microsoft SQL Server Management Studio, the SELECT query is as follows:

Microsoft SQL Server Management Studio

File Edit View Query Project Tools Window Community Help

New Query | SP3DTrain_RDB | Execute

rhd703\rhd703...QLQuery1.sql* | Summary

```
Select
x3.IndustryCommodityCode,
x6.ItemName as 'PipeRun_Name',
x4.ShortStringValue as 'CommodityType',
count(*) as qty
from JRteCompOccur x1
Join XMadeFrom x2 ON (x2.OidOrigin = x1.Oid)
Join JDPipeComponent x3 ON (x3.Oid = x2.OidDestination)
Join CL_PipingCommodityType x4 ON (x4.ValueID = x3.CommodityType)
Join XOwnsParts x5 ON (x5.oiddestination = x1.oid)
Join JNamedItem x6 ON (x6.oid = x5.oidOrigin)
WHERE (x3.CommodityClass = 5)
Group by x3.IndustryCommodityCode, x6.ItemName, x4.ShortStringValue
```

Results | Messages

	IndustryCommodityCode	PipeRun_Name	CommodityType	qty
1	VAAAHABAHADJADAZZZZUS	U01-10-P-0003-1C0031	GAT	1
2	VAAAHABAHADJADAZZZZUS	U01-10-P-0005-1C0031	GAT	1
3	VAAAHABAHADJADAZZZZUS	U02-6-P-0002-1C0031	GAT	1
4	VAAAHABAHADJADAZZZZUS	U02-6-P-0004-1C0031	GAT	1
5	VAAAHABAHADJADAZZZZUS	U03-10-W-0002-1C0031	GAT	1
6	VAAAHABAHADJADAZZZZUS	U04-10-P-0002-1C0031	GAT	2
7	VAAAHABAHADJADAZZZZUS	U04-3-P-0005-1C0031	GAT	1
8	VAAAHABAHADJADAZZZZUS	U04-4-P-0002-1C0031	GAT	1
9	VAAAHABAHADJADAZZZZUS	U04-4-P-0004-1C0031	GAT	1

Using Oracle SQL Plus or Oracle SQL Developer, the SELECT query is as follows:

Enter SQL Statement:

```
ALTER SESSION SET CURRENT_SCHEMA = SP3DTrain_RDB;
Select
x3.IndustryCommodityCode,
x6.ItemName as "PipeRun_Name",
x4.ShortStringValue as "CommodityType",
count(*) as qty
from JRteCompOccur x1
Join XMadeFrom x2 ON (x2.OidOrigin = x1.Oid)
Join JDPipeComponent x3 ON (x3.Oid = x2.OidDestination)
Join CL_PipingCommodityType x4 ON (x4.ValueID = x3.CommodityType)
Join XOwnsParts x5 ON (x5.oiddestination = x1.oid)
Join JNamedItem x6 ON (x6.oid = x5.oidOrigin)
WHERE (x3.CommodityClass = 5)
Group by x3.IndustryCommodityCode, x6.ItemName, x4.ShortStringValue
```

Lab 10: Creating a Naming Rule for Pipeline Systems

Objectives

After completing this lab, you will be able to:

- Create a simple naming rule for the Pipeline System
- Implement the IJNameRule interface
- Use the Attribute Helper service to retrieve pipeline object properties
- Use Catalog Resource Manager to get a connection to the code list metadata
- Bulkload the Naming Rule into the Catalog database

This session will demonstrate an implementation of a naming rule for pipeline system objects. This component will generate a name for pipeline objects as shown here:

Pipeline Name = Fluid Code + Sequence Number

1. Create the following directories:

   c:\train\CustomNameRule

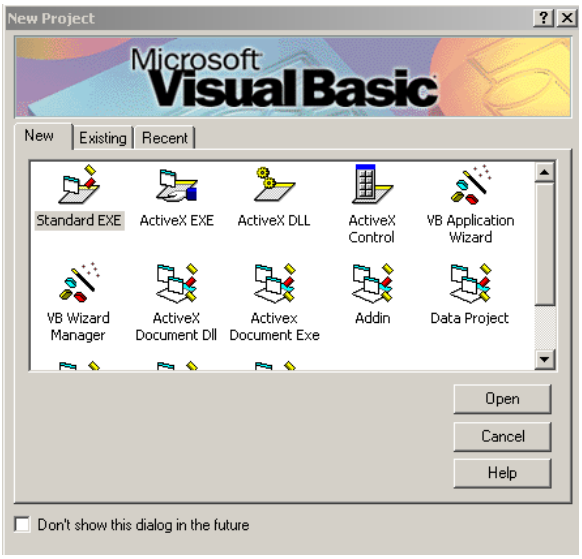
2. Copy the Naming Rule Visual Basic Template Project provided by the instructor to *c:\train\CustomNameRule\Template.*

   Note:
   - The Naming Rule template is delivered under [Installation]\Programming\ExampleCode\Symbols\NamingRuleTemplate

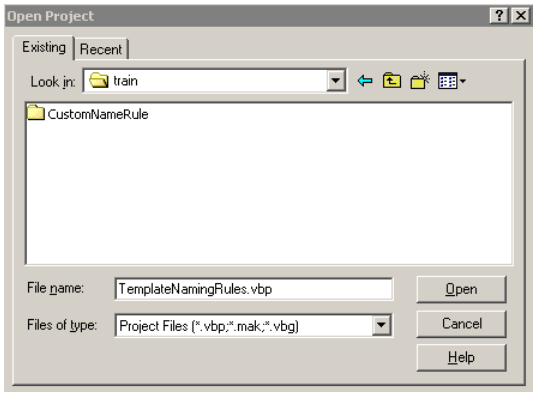
3. Create a directory called lab10 as shown here:

   c:\train\CustomNameRule\lab10

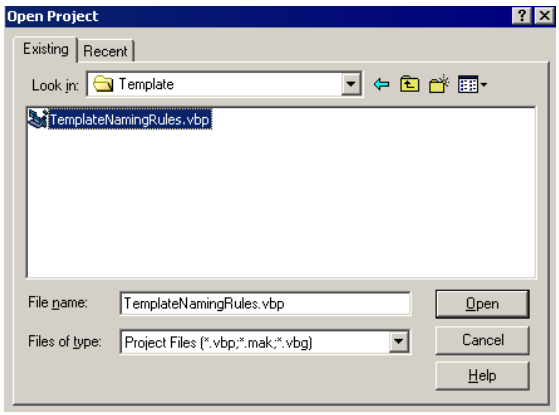
4. Run Microsoft Visual Basic 6.0.
5. Close the Microsoft New Project dialog box.



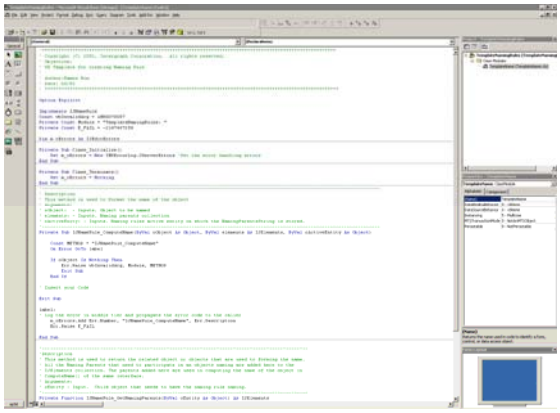
6. Select *File -> Open Project* option to open the Open Project Dialog box.



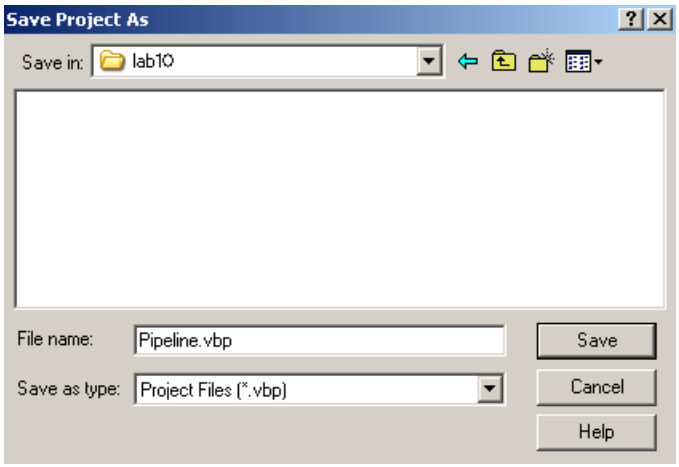
7. Navigate to c:\train\CustomNameRule\Template and open the Naming Rule Template project.



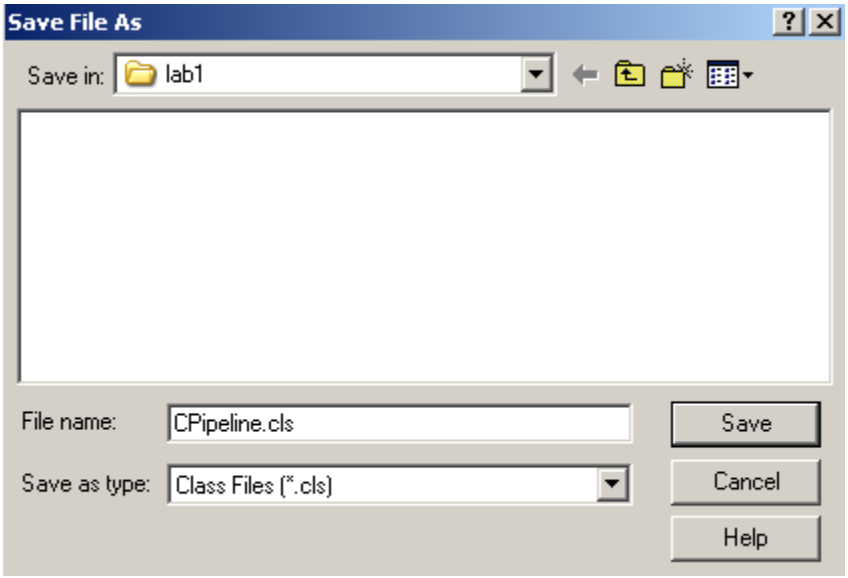
8. Setup the Visual Basic Development Environment as shown below:



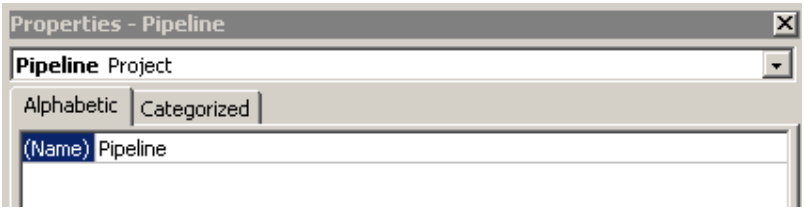
9. Go to the Visual Basic Explorer Window and select the Project node. Select *File -> Save Project As* option to save the project as Pipeline.vbp under the lab10 directory.

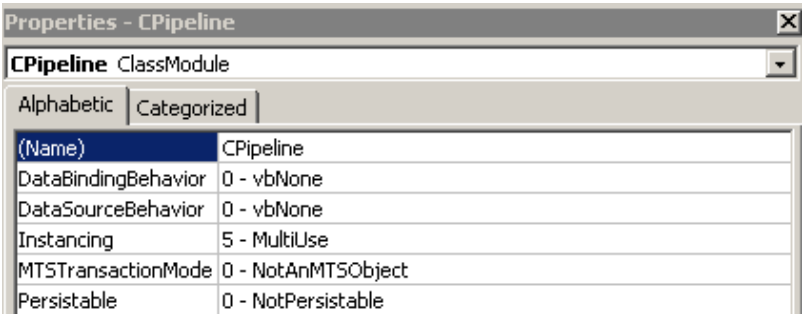


10. Go to the Visual Basic Explorer Window and select the TemplateName class node. Select *File -> Save TemplateName.cls As* option to save the class module as CPipeline.cls under lab10 directory.



11. Go to the Properties Window and change the name of the Project and ClassModule as shown here:





12. Go to the General Declarations section and change the value of the *Constant Module variable* from *"TemplateNamingRules:"* to *"Pipelines:"*

    Private Const Module = "Pipeline: "

13. Declare an object variable to hold the reference to the IJDCodeListMetaData.

    Private m_oCodeListMetadata As IJDCodeListMetaData

14. Access the subroutine ComputeName section by selecting IJNameRule in the Object List Box and select the ComputeName in the Procedure List Box.

15. Add lines to the body of the subroutine ComputeName method

    Hint:
    Declare an object variable to hold a reference to the IJNamedItem

```
Dim oChildNamedItem As IJNamedItem
Dim strChildName As String
Set oChildNamedItem = oObject
strChildName = vbNullString
```

16. Declare an object variable to hold a reference to the IJDAttributes

```
Dim oAttributes As IJDAttributes
Set oAttributes = oObject
```

17. Declare a variable of type String to store the sequence number.

```
Dim strSequenceNumber As String
```

18. Use IJDAttributes interface to get a collection of attributes of the selected item. Finally, Use the method value to get the object's attribute

```
strSequenceNumber =
oAttributes.CollectionOfAttributes("IJPipelineSystem").Item("SequenceNumber").Value
```

19. Declare local variables to hold the fluid codelist value and short description.

```
Dim FluidCodeID As Long
Dim strFluidCode As String
strFluidCode = vbNullString
```

20. Use IJDAttributes and IJDCodeListMetaData interfaces to get the fluid code short description.

```
Set m_oCodeListMetadata = GetCatalogResourceManager
FluidCodeID =   _
oAttributes.CollectionOfAttributes("IJPipelineSystem").Item("FluidCode").Value
strFluidCode = m_oCodeListMetadata.ShortStringValue("FluidCode", FluidCodeID)
```

21. Build the name of the pipeline:

```
strChildName = strFluidCode & "-" & strSequenceNumber
oChildNamedItem.Name = strChildName
```

22. Finally, remove the reference from all object variables.

```
Set oChildNamedItem = Nothing
Set oAttributes = Nothing
```

23. Insert into your existing project the following Private Function. Open the GetCatalog.txt file located in the template directory file and use Cut/Paste operation to insert the lines. The inserted lines should look like this:

```
'-----------------------------------------------------------------------------------------
'Description
' Function returns the CatalogResourceManager
'-----------------------------------------------------------------------------------------
Private Function GetCatalogResourceManager() As IUnknown
   Const METHOD = "GetCatalogResourceManager"
   On Error GoTo ErrHandler

   Dim oDBTypeConfig As IJDBTypeConfiguration
   Dim pConnMiddle As IJDConnectMiddle
   Dim pAccessMiddle As IJDAccessMiddle
   Dim jContext As IJContext
   Set jContext = GetJContext()
   Set oDBTypeConfig = jContext.GetService("DBTypeConfiguration")
   Set pConnMiddle = jContext.GetService("ConnectMiddle")
   Set pAccessMiddle = pConnMiddle

   Dim strCatlogDB As String
   strCatlogDB = oDBTypeConfig.get_DataBaseFromDBType("Catalog")
   Set GetCatalogResourceManager = pAccessMiddle.GetResourceManager(strCatlogDB)
   Set jContext = Nothing
   Set oDBTypeConfig = Nothing
   Set pConnMiddle = Nothing
   Set pAccessMiddle = Nothing
Exit Function
ErrHandler:
   m_oErrors.Add Err.Number, "GetCatalogResourceManager", Err.Description
   Err.Raise E_FAIL
End Function
```

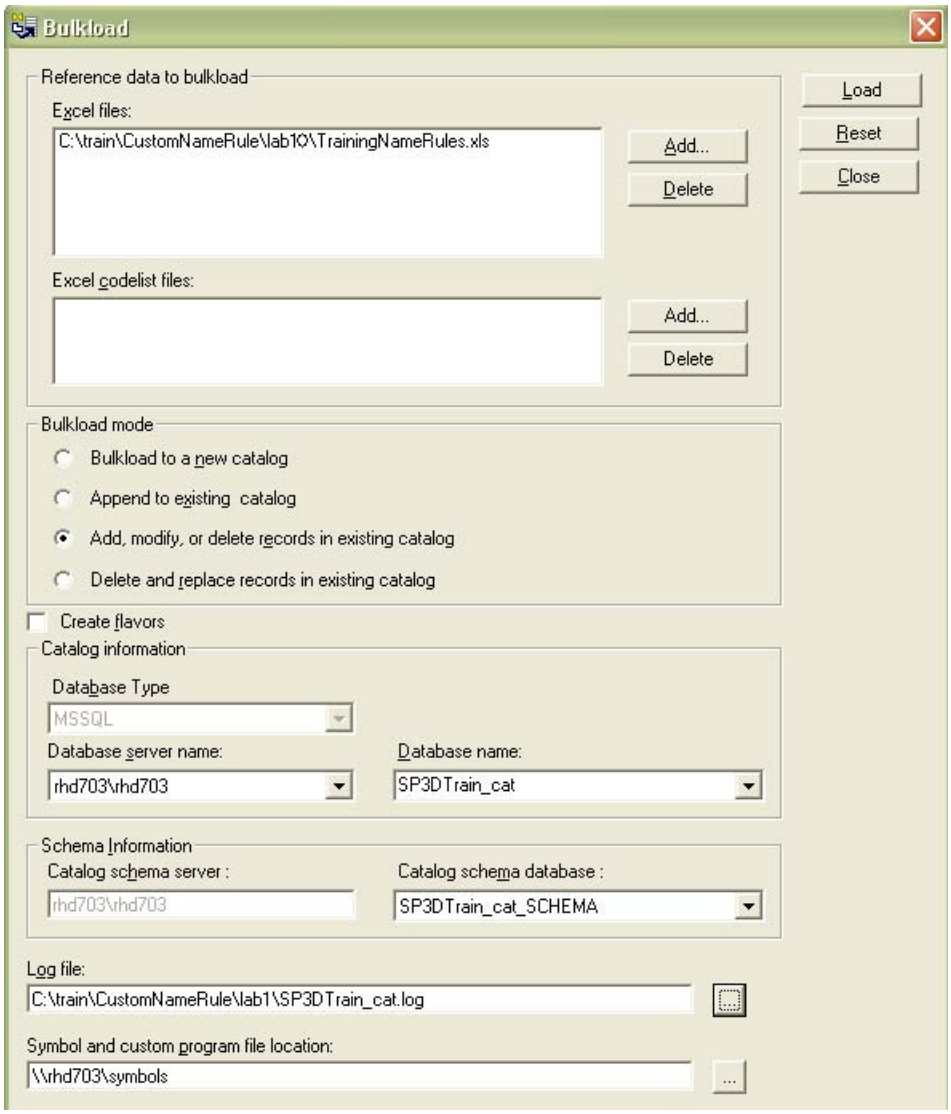
24. Go to the Subroutine Terminate method and add one line to remove the reference from object variable m_oCodeListMetadata.

```
   Set m_oCodeListMetadata = Nothing
```

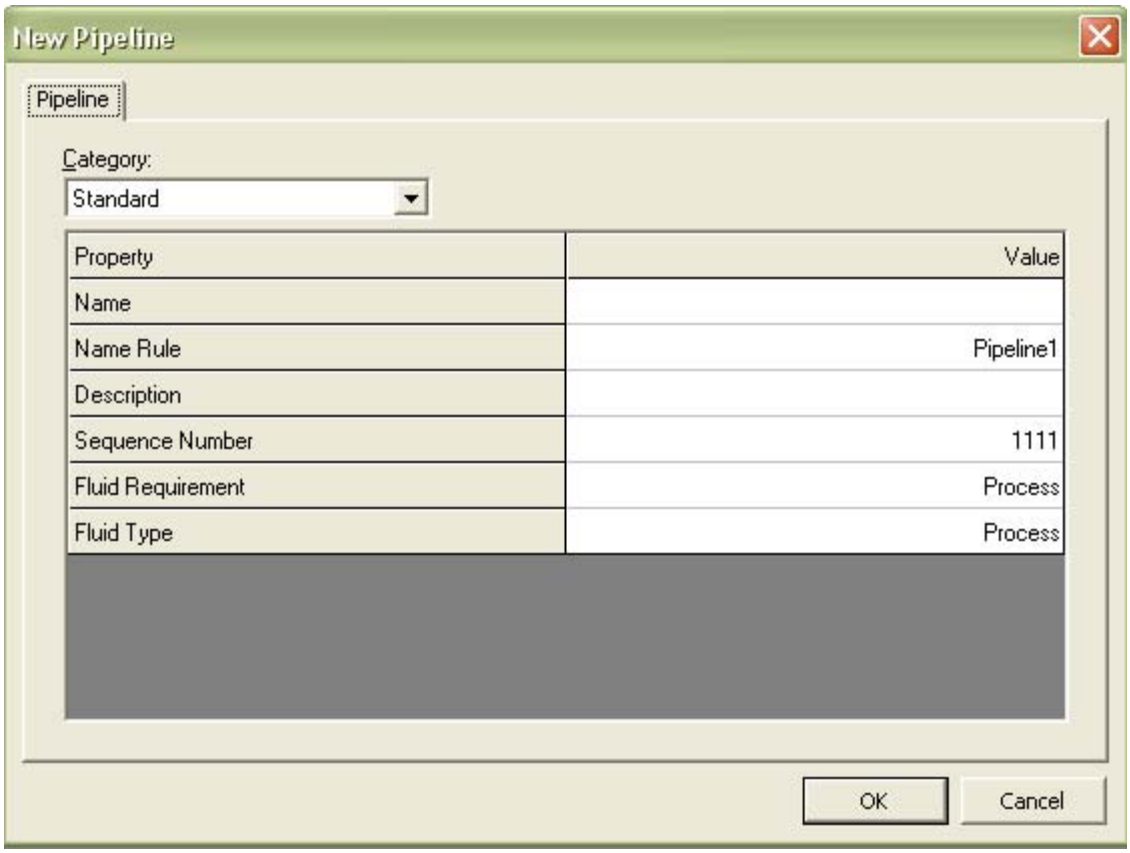
25. Compile the Visual Basic project and save the dll as pipeline.dll in the c:\train\lab10
26. Save and Exit the program.
27. Open the TemplateNamingRules.xls under C:\train\CustomNameRule\Templates
28. Add the name of the class object and the ProgID as follows:

Head	TypeName	Name	SolverProgID
!	Class Name of the object	GUI Name	ProgID(Vbprojectname.classmodulename)
Start			
a	CPPipelineSystem	Pipeline1	Pipeline.CPipeline
End			

29. Save the Excel sheet as TrainingNameRules.xls under c:\train and exit Excel.
30. Run Bulkload Utility (START Menu -> Intergraph SmartPlant 3D -> Database Tools -> Bulkload Reference Data)
31. Set the bulkload to A/M/D mode.
32. Select Load button to add the new naming rule into the training catalog.



33. Go to SP3D System & Specification Task and create a new pipeline system to test your naming rule. Select and Key in the following data in the New Pipeline dialog box.


New Pipeline
Pipeline
Category:
Standard
Property
Value
Name
Name Rule
Pipeline1
Description
Sequence Number
1111
Fluid Requirement
Process
Fluid Type
Process
OK
Cancel

Lab 11: Creating a Naming Rule for PipeRun objects

Objective

After completing this lab, you will be able to:

- Create a simple naming rule for the piperun objects
- Implement the IJNameRule interface
- Reference the appropriate libraries to build the object name
- Use the Attribute Helper service to retrieve piperun properties
- Use the Relation Helper service to obtain the Spec object
- Get the Parent Name System
- Bulk loading the Naming Rule into the Catalog database

This session will demonstrate an implementation of a naming rule for piperun objects. This component will generate a name for piperun objects as shown here:

PipeRun object:

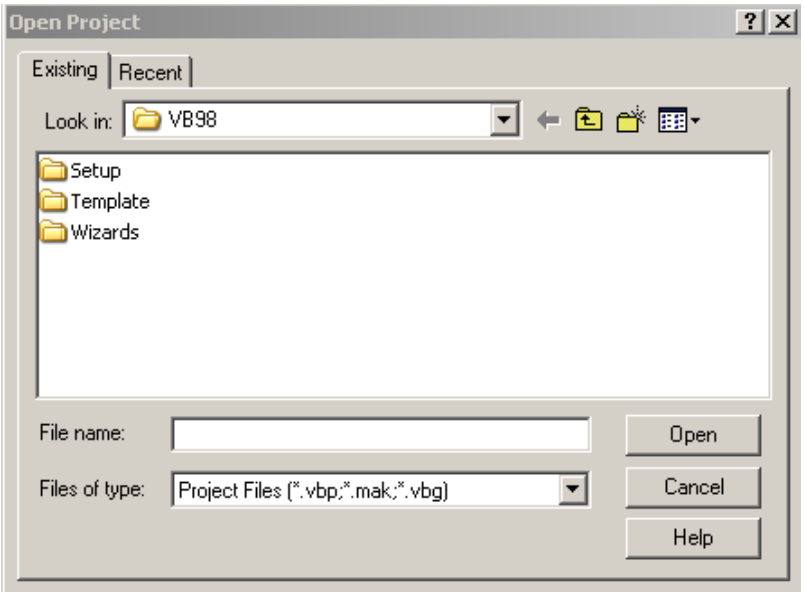
Pipe Runs:
NPD + NPD Units + Spec Name + Parent System

1. Create a directory called lab11 as shown here:

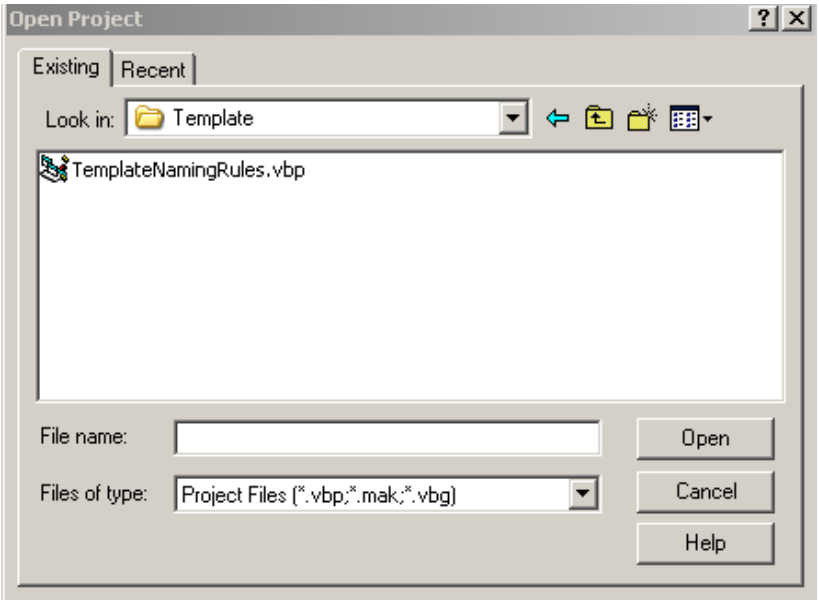
   c:\train\CustomNameRule\lab11

2. Run Microsoft Visual Basic 6.0.
3. Close the Microsoft New Project dialog box.

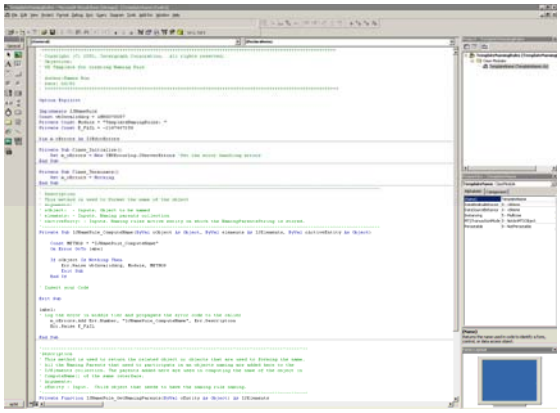
4. Select *File -> Open Project* option to open the Open Project Dialog box.



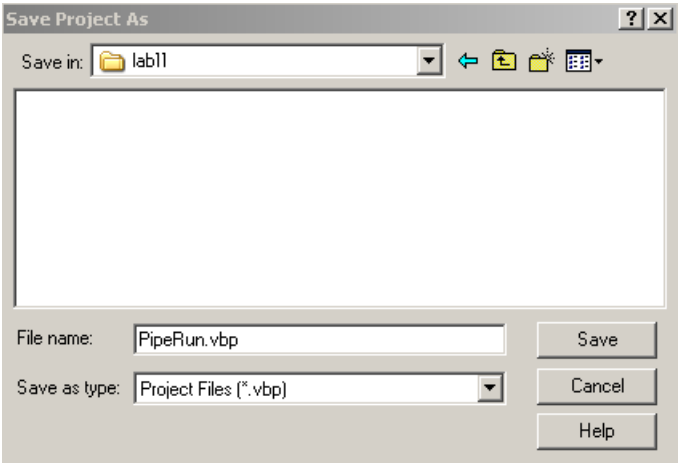
5. Navigate to c:\train\CustomNameRule\Template and open the Naming Rule Template project provided by the instructor.



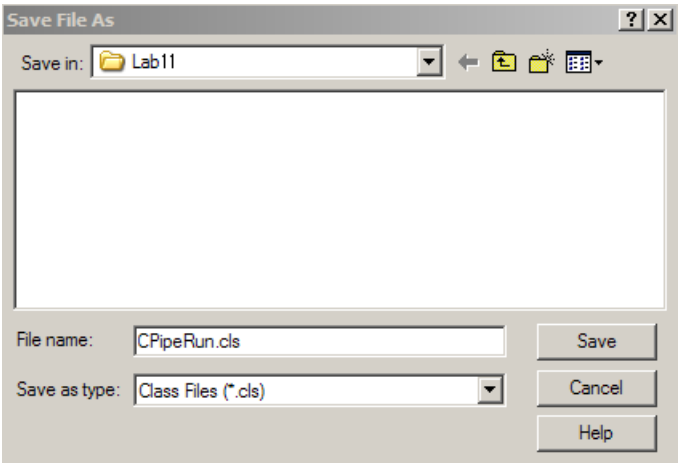
6. Setup the Visual Basic Development Environment as shown below:



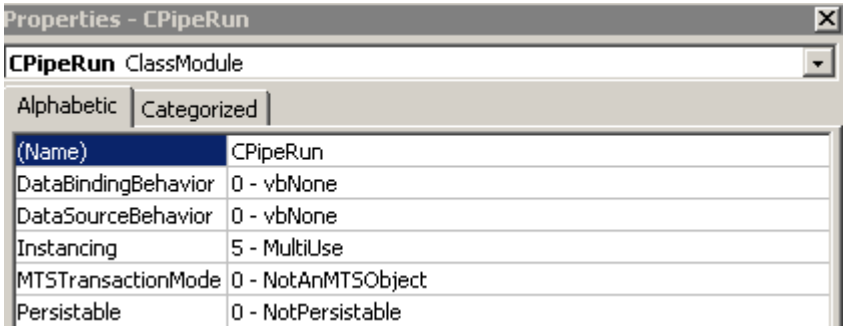
7. Go to the Visual Basic Explorer Window and select the Project node. Select *File -> Save Project As* option to save the project as PipeRun.vbp under the lab11 directory.



8. Go to the Visual Basic Explorer Window and select the TemplateName class node. Select *File -> Save TemplateName.cls As* option to save the class module as CPipeRun.cls under lab11 directory.



9. Go to the Properties Window and change the name of the Project and ClassModule as follows:



10. Go to the General Declarations section and change the value of the *Constant Module variable* from *"TemplateNamingRules:"* to *"PipeRun:"*

    Private Const Module = "PipeRun: "

11. Access the subroutine GetNamingParents section by selecting IJNameRule in the Object List Box and select the GetNamingParents in the Procedure List Box. Add code snippet to the body of the subroutine GetNamingParents. The lines should get all the parent objects that need to participate in the object naming. Add of the parent objects to the 'IJElements collection.

    Hints:
    Comment the following line:

    Set IJNameRule_GetNamingParents = Nothing

    Create the collection and declare an object variable to hold a reference to the IJSystemChild.

    Set IJNameRule_GetNamingParents = New IMSCoreCollections.JObjectCollection

    Dim oSysChild As IJSystemChild
    Set oSysChild = oEntity

    Declare an object variable to hold a reference to the IJSystem.

    Dim oSysParent As IJSystem
    Set oSysParent = oSysChild.GetParent

    Add the parent object into the collection using the method Add as shown here:

```
If Not (oSysParent Is Nothing) Then
   Call IJNameRule_GetNamingParents.Add(oSysParent)
End If
```

Add code snippet to remove the reference from object variables:

```
Set oSysChild = Nothing
Set oSysParent = Nothing
```

The resulting lines should look like this:

```
Set IJNameRule_GetNamingParents = New IMSCoreCollections.JObjectCollection
```

```
Dim oSysChild As IJSystemChild
Set oSysChild = oEntity
Dim oSysParent As IJSystem
Set oSysParent = oSysChild.GetParent
If Not (oSysParent Is Nothing) Then
   Call IJNameRule_GetNamingParents.Add(oSysParent)
End If

Set oSysChild = Nothing
Set oSysParent = Nothing
```

12. Access the subroutine ComputeName section by selecting IJNameRule in the Object List Box and select the ComputeName in the Procedure List Box.

13. Add code snippet to the body of the subroutine ComputeName. The lines should contain statements for formatting the object name. The object name consists of Parent System Name, NPD, NPD Unit and Piping Specification Name. For example,

    NPD + NPD Units + Spec Name + Parent System

14. Declare an object variable to hold a reference to the IJNamedItem.

```
Dim oChildNamedItem As IJNamedItem
Dim strChildName As String
Set oChildNamedItem = oObject
strChildName = vbNullString
```

15. Declare an object variable to hold a reference to the IJDAttributes.

```
Dim oAttributes As IJDAttributes
Set oAttributes = oObject
```

16. Declare variables strNPD and strNPDUnits to store the NPD of the PipeRun.

```
Dim strNPD As String
Dim strNPDUnitType As String
```

17. Use the attribute service to get the NPD and NPD Unit as follows:

```
strNPD = CStr(oAttributes.CollectionOfAttributes("IJRtePipeRun").Item("NPD").Value)
strNPDUnitType = oAttributes.CollectionOfAttributes("IJRtePipeRun").Item("NPDUnitType").Value

If strNPDUnitType = "in" Then
   strNPDUnitType = Chr(34)
End If
```

18. Declare object variables to hold a reference to the DRelationHelper and DCollectionHelper. Declare an object variable to hold a reference to the IJDSpec. Declare a variable strSpecName to store the Spec Name.

```
Dim oRelationHelper As IMSRelation.DRelationHelper
Dim oCollection As IMSRelation.DCollectionHelper
Set oRelationHelper = oObject
Dim oSpec As IJDSpec
Dim strSpecName As String
Set oCollection = oRelationHelper.CollectionRelations("IJRtePathRun", "Spec")
Set oSpec = oCollection.Item(1)
strSpecName = oSpec.SpecName
```

19. Add lines to get the Parent Name.

```
Dim oParentNamedItem As IJNamedItem
Dim strParentName As String
strParentName = vbNullString
Set oParentNamedItem = elements.Item(1)
strParentName = oParentNamedItem.Name
```

20. Build the name of the piperun.

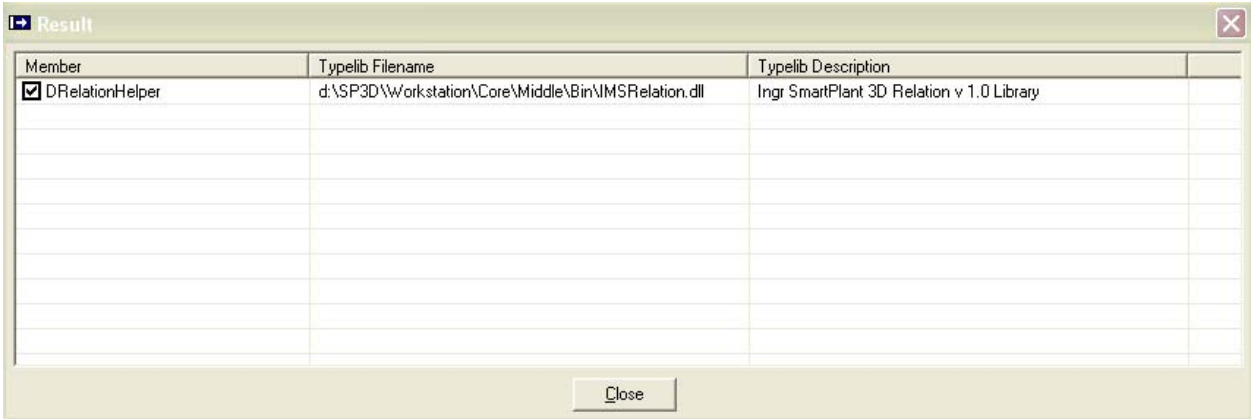
```
strChildName = strNPD & strNPDUnitType & "-" & strSpecName & "-" & strParentName
oChildNamedItem.Name = strChildName
```

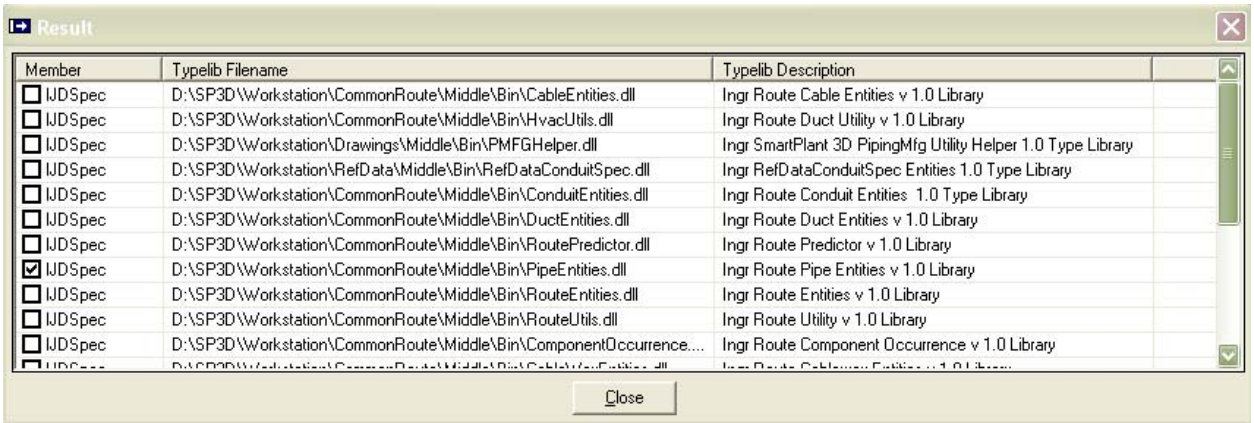
21. Add lines to remove the reference from object variables.

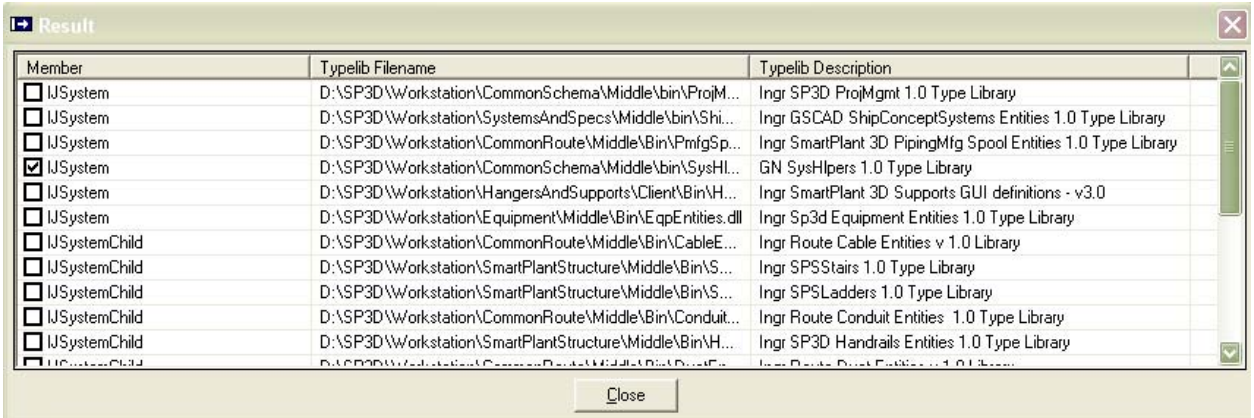
```
Set oChildNamedItem = Nothing
Set oAttributes = Nothing
Set oRelationHelper = Nothing
Set oCollection = Nothing
Set oSpec = Nothing
Set oParentNamedItem = Nothing
```

22. Compile the Visual Basic project and save the dll as PipeRun.dll in the c:\train\lab11
23. Save and Exit the program.

Note: You need to reference additional libraries using the SP3D Reference Tool. For example,



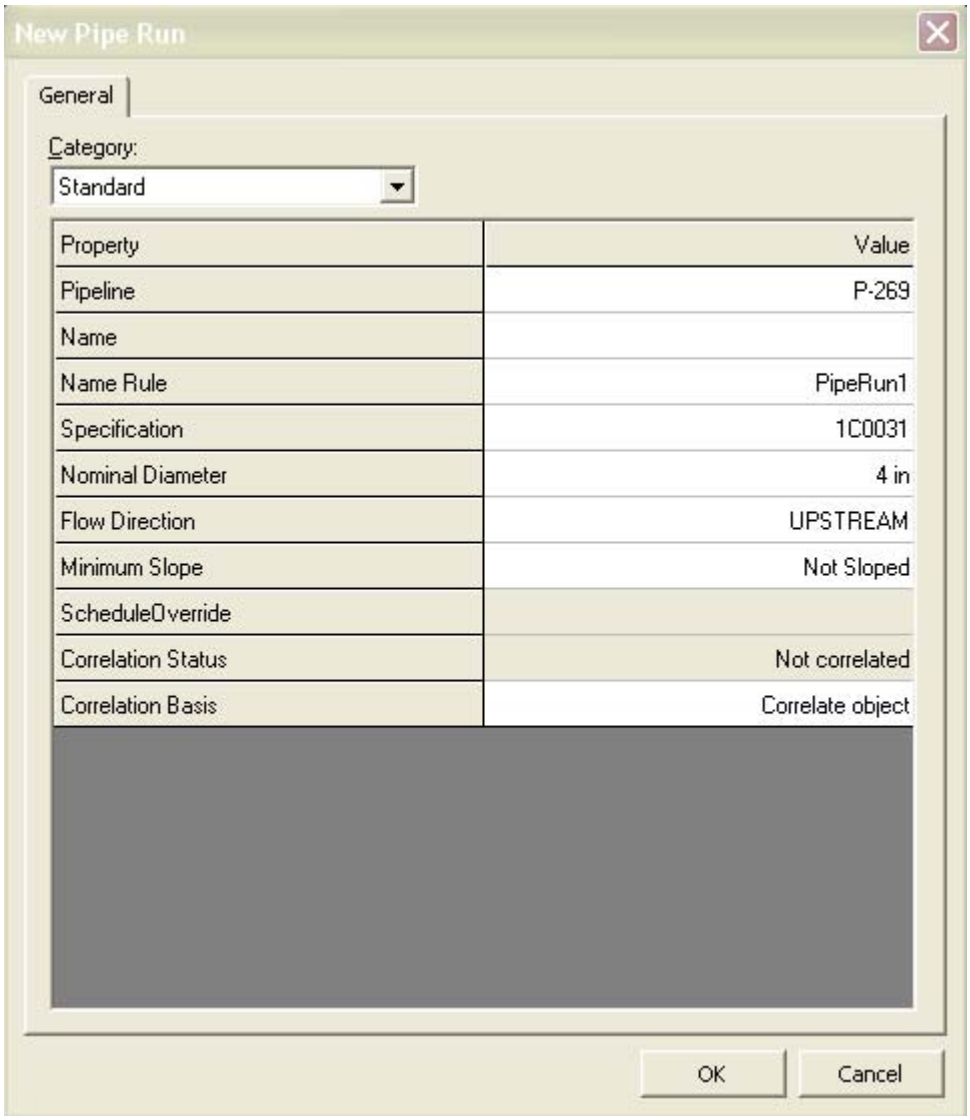




24. Open the c:\train\TrainingNameRules.xls saved in previous lab.
25. Add the name of the class object and the ProgID as follows:

Head	TypeName	Name	SolverProgID
!	Class Name of the object	GUI Name	ProgID(Vbprojectname.classmodulename)
Start			
	CPPipelineSystem	Pipeline1	Pipeline.CPipeline
a	CPMPipeRun	PipeRun1	PipeRun.CPipeRun
End			

26. Save and Exit Excel.
27. Run Bulkload Utility using the A/M/D mode to add the new naming rule into the training catalog.
28. Go to SP3D Piping Task and create a PipeRun to test your naming rule. Select and Key in the following data in the New PipeRun dialog box.



Lab 12: Creating a Naming Rule for Member Parts

Objective

After completing this lab, you will be able to:

- Create a simple naming rule for the Member Part
- Implement the IJNameRule interface
- Reference the appropriate libraries to build the object name
- Use the Attribute Helper service to retrieve Member Part properties
- Use the Relation Helper service to obtain the Cross Section object
- Use Catalog Resource Manager to get a connection to the Code List Meta Data
- Use Model Resource Manager to get a connection to the Model Database
- Use the Name Generator Service to get an unique counter
- Bulk loading the Naming Rule into the Catalog database

This session will demonstrate an implementation of a naming rule for the Member Part objects. This component will generate a name for Member Part objects as shown here:

Member Part Object:

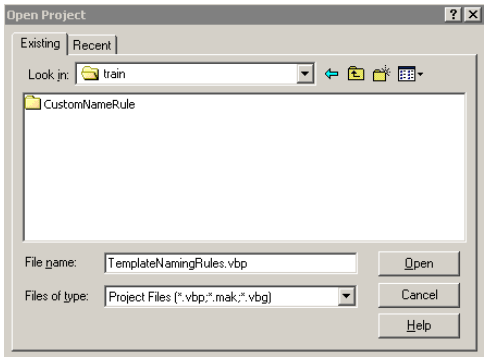
The Short Description of the Member Category Code List + Section Name + Location + IndexCounter

1. Create a directory called lab12 as follows:

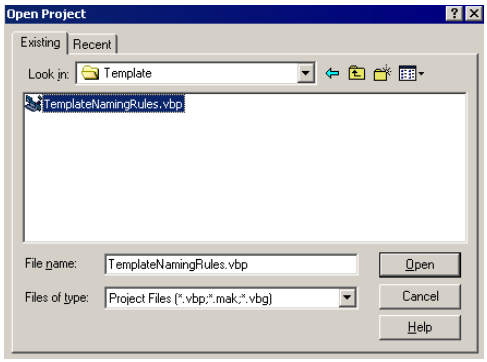
   c:\train\CustomNameRule\lab12

2. Run Microsoft Visual Basic 6.0.
3. Close the Microsoft New Project dialog box.

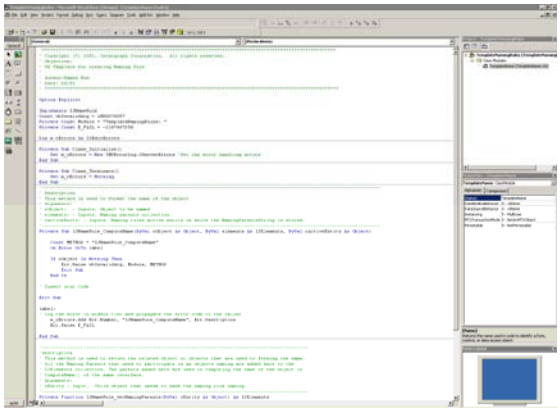
4. Select *File -> Open Project* option to open the Open Project Dialog box.



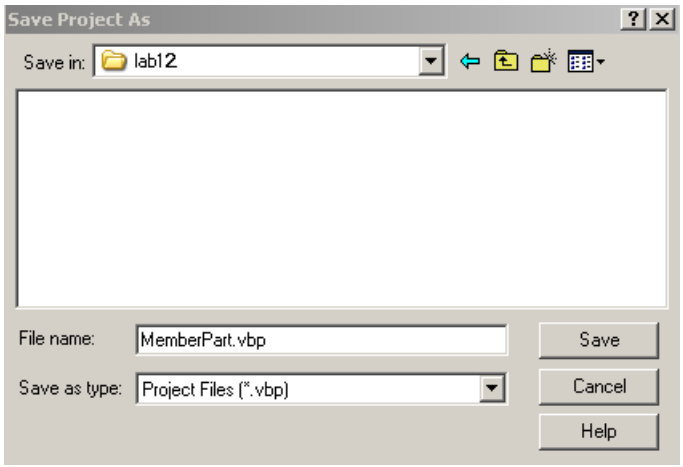
5. Navigate to c:\train\CustomNameRule\Template and open the Naming Rule Template project.



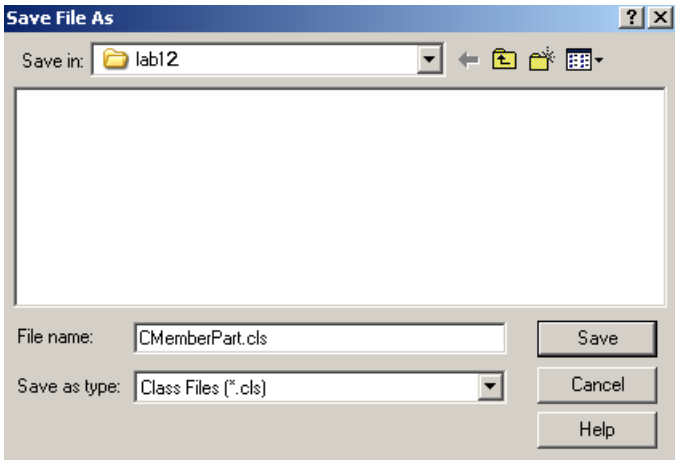
6. Setup the Visual Basic Development Environment as shown below:



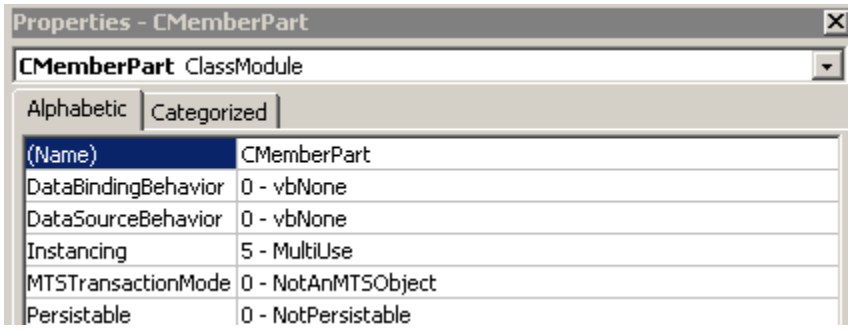
7. Go to the Visual Basic Explorer Window and select the Project node. Select *File -> Save Project As* option to save the project as MemberPart.vbp under the lab12 directory.



8. Go to the Visual Basic Explorer Window and select the TemplateName class node. Select *File -> Save TemplateName.cls As* option to save the class module as CMemberPart.cls under lab12 directory.



9. Go to the Properties Window and change the name of the Project and ClassModule as follows:



10. Go to the General Declarations section and change the value of the *Constant Module variable* from *"TemplateNamingRules:"* to *"MemberPart:"*

    Private Const Module = "MemberPart: "

11. Use the SP3D Reference tool to reference the following libraries or use the Project >References command. Go to *Project -> References* option to open the References dialog box. Select the *Browser* button and pick the following libraries:

    Ingr SPSMembers Entities 1.0 Type Library
    [Install Product]\SmartPlantStructure\Middle\Bin\SPSMembers.dll

    Ingr Sp3d NameGenerator 1.0 Type Library
    [Install Product]\CommonApp\Middle\Bin\NameGenerator.dll

Ingr SmartPlant3D Relation 1.0 Type Library
[Install Product]\Core\Middle\Bin\IMSRelation.dll

12. Insert into your existing project the following Private Functions. Open the GetCatalog.txt file and GetModel.txt located in the template directory file and use Cut/Paste operation to insert the code snippet. The inserted code snippet should look like this:

```
'-----------------------------------------------------------------------------------------
'Description
' Function returns the CatalogResourceManager
'-----------------------------------------------------------------------------------------
Private Function GetCatalogResourceManager() As IUnknown
   Const METHOD = "GetCatalogResourceManager"
   On Error GoTo ErrHandler

   Dim oDBTypeConfig As IJDBTypeConfiguration
   Dim pConnMiddle As IJDConnectMiddle
   Dim pAccessMiddle As IJDAccessMiddle
   Dim jContext As IJContext
   Set jContext = GetJContext()
   Set oDBTypeConfig = jContext.GetService("DBTypeConfiguration")
   Set pConnMiddle = jContext.GetService("ConnectMiddle")
   Set pAccessMiddle = pConnMiddle

   Dim strCatlogDB As String
   strCatlogDB = oDBTypeConfig.get_DataBaseFromDBType("Catalog")
   Set GetCatalogResourceManager = pAccessMiddle.GetResourceManager(strCatlogDB)
   Set jContext = Nothing
   Set oDBTypeConfig = Nothing
   Set pConnMiddle = Nothing
   Set pAccessMiddle = Nothing
Exit Function
ErrHandler:
   m_oErrors.Add Err.Number, "GetCatalogResourceManager", Err.Description
   Err.Raise E_FAIL
End Function

'-----------------------------------------------------------------------------------------
'Description
' Function returns the ModelResource Manager
'-----------------------------------------------------------------------------------------
Private Function GetModelResourceManager() As IUnknown
   Const METHOD = "GetModelResourceManager"
   On Error GoTo ErrHandler

   Dim jContext As IJContext
   Dim oDBTypeConfig As IJDBTypeConfiguration
   Dim oConnectMiddle As IJDAccessMiddle
   Dim strModelDBID As String
   Set jContext = GetJContext()
   Set oDBTypeConfig = jContext.GetService("DBTypeConfiguration")
   Set oConnectMiddle = jContext.GetService("ConnectMiddle")
   strModelDBID = oDBTypeConfig.get_DataBaseFromDBType("Model")
```

```
    Set GetModelResourceManager = oConnectMiddle.GetResourceManager(strModelDBID)

    Set jContext = Nothing
    Set oDBTypeConfig = Nothing
    Set oConnectMiddle = Nothing
Exit Function
ErrHandler:
    m_oErrors.Add Err.Number, "GetModelResourceManager", Err.Description
    Err.Raise E_FAIL
End Function
```

13. Go to the General Declarations section and declare object variables to hold the reference to the IJDCodeListMetaData and IUnknown interfaces.

```
Private m_oCodeListMetadata As IJDCodeListMetaData
Private m_oModelResourceMgr As IUnknown
```

14. Access the subroutine ComputeName section by selecting IJNameRule in the Object List Box and select the ComputeName in the Procedure List Box.

15. Add code snippet to the body of the subroutine ComputeName. The code snippet should contain statements for formatting the object name. The object name consists of a string to indicate the member category, a unique index counter and the section name. For example,

Member Part Object:

Short Description Member Category Code List + Section Name + Location + IndexCounter

Hint:
Declare an object variable to hold a reference to the IJNamedItem

```
Dim  oChildNamedItem As IJNamedItem
Set oChildNamedItem = oObject
```

Declare an object variable to hold a reference to the IJDAttributes

```
Dim oAttributes As IJDAttributes
 Set oAttributes = oObject
```

Declare a variable MemberTypeID to store the MemberType value.

```
Dim MemberTypeID As Long
```

Use the attribute service to get MemberTypeID. The resulting line should look like this:

```
MemberTypeID = oAttributes.CollectionOfAttributes("ISPSMemberType").Item("TypeCategory").Value
```

Declare variables to store the codelist table name and short description of the Member Type.

```
Dim strTableName As String
```

```
Dim strMemType As String
 strTableName = "StructuralMemberTypeCategory"
```

Add lines to get the member type short description and set the result to upper case. The resulting lines should look like this:

```
If m_oCodeListMetadata Is Nothing Then
   Set m_oCodeListMetadata = GetCatalogResourceManager
End If
strMemType = UCase(m_oCodeListMetadata.ShortStringValue(strTableName, MemberTypeID))
```

Use the relation service to get the name of the cross section.
Declare object variables to hold a reference to the DRelationHelper and DCollectionHelper. Declare an object variable to hold a reference to the IJCrossSection. Declare a variable strSectionName to store the Cross Section Name.
The resulting lines should look like this:

```
Dim oRelationHelper As IMSRelation.DRelationHelper
Dim oCollection As IMSRelation.DCollectionHelper
Set oRelationHelper = oObject

Set oCollection = oRelationHelper.CollectionRelations("ISPSMemberPartPrismatic", "Generation6_DEST")
Set oRelationHelper = oCollection.Item(1)
Set oCollection = Nothing
Set oCollection = oRelationHelper.CollectionRelations("ISPSPartPrismaticDesign", "Definition_ORIG")

Dim oMembCrossSection As IJCrossSection
Dim strSectionName As String
Set oMembCrossSection = oCollection.Item(1)
Set oAttributes = oCollection.Item(1)
strSectionName = oAttributes.CollectionOfAttributes("IStructCrossSection").Item("SectionName").Value

Dim strChildName As String
strChildName = strMemType
strChildName = strChildName + "-" + strSectionName
```

Use the Name Generator Service to generate a counter based on the Member Type Category. Store the formatted name in oChildNamedItem.Name. Declare an object variable to hold a reference to the IJNameCounter.

```
Dim oNameCounter As IJNameCounter
Set oNameCounter = New GSCADNameGenerator.NameGeneratorService
```

The resulting lines should look like this:

```
Dim strLocation As String
strLocation = vbNullString

Dim nCount As Long
Set m_oModelResourceMgr = GetModelResourceManager

nCount = oNameCounter.GetCountEx(m_oModelResourceMgr, strChildName, strLocation)
```

```
If Not (strLocation = vbNullString) Then
    strChildName = strChildName + "-" + strLocation + "-" + CStr(nCount)
Else
    strChildName = strChildName + "-" + CStr(nCount)
End If

oChildNamedItem.Name = strChildName
```

16. Add lines to remove the reference from object variables.

    Go to the subroutine ComputeName() method:

```
Set oNameCounter = Nothing
Set oChildNamedItem = Nothing
Set oCollection = Nothing
Set oRelationHelper = Nothing
Set oAttributes = Nothing
Set oMembCrossSection = Nothing
```

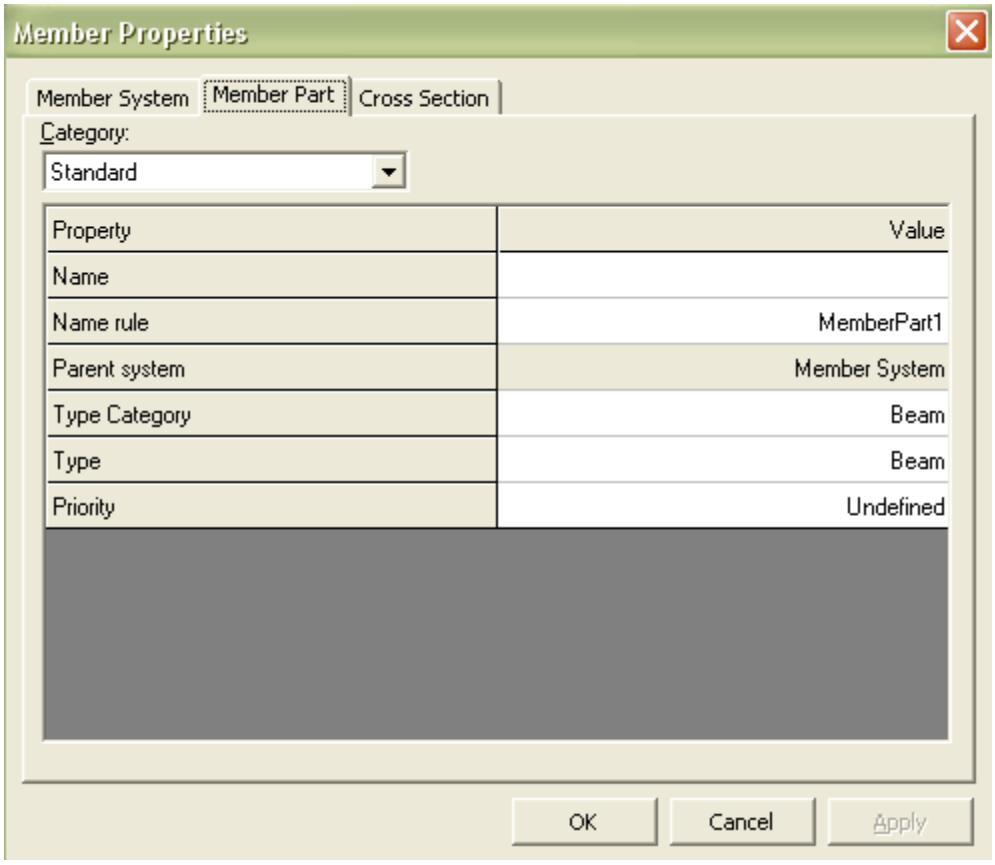
    Go to the Subroutine Terminate() method:

```
Set m_oCodeListMetadata = Nothing
Set m_oModelResourceMgr = Nothing
```

17. Compile the Visual Basic project and save the dll as MemberPart.dll in the c:\train\lab12
18. Save and Exit the program.
19. Open c:\train\TrainingNameRules.xls.
20. Add the name of the class object and the ProgID as follows:

Head	TypeName	Name	SolverProgID
!	Class Name of the object	GUI Name	ProgID(Vbprojectname.classmodulename)
Start			
	CPPipelineSystem	Pipeline1	Pipeline.CPipeline
	CPMPipeRun	PipeRun1	PipeRun.CPipeRun
a	CSPSMemberPartPrismatic	MemberPart1	MemberPart.CMemberPart
End			

20. Save and Exit Excel.
21. Run Bulkload Utility using the A/M/D mode and add the new naming rule into the training catalog.
22. Go to SP3D Structure task and run the Place Member Command to test your naming rule. Select and key in the following data in the Member properties dialog box.


Member Properties
Member System
Member Part
Cross Section
Category:
Standard
Property
Value
Name
Name rule
MemberPart1
Parent system
Member System
Type Category
Beam
Type
Beam
Priority
Undefined
OK
Cancel
Apply

Lab 13: Interference Check Post-Processing Rule

Objectives

After completing this lab, you will be able to:

- Understand the post-processing interference checking rule
- Assign the interference object to a permission group

1. Create the following directories:
   c:\train\IFCRule

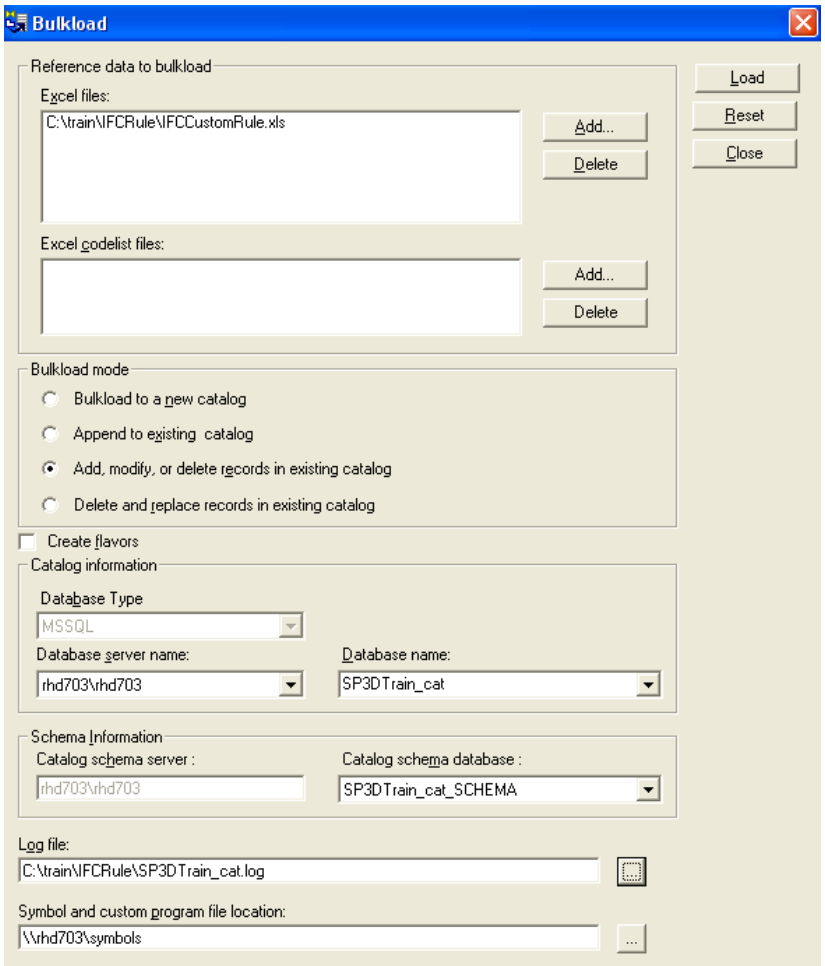
2. Copy the delivered IFC Post-Processing Visual Basic files to
   c:\train\IFCRule

   Note:
   - The IFC Post-Processing Visual Basic project is delivered under [Installation]\Programming\ExampleCode\Rules\InterferenceRules

3. Open the IFCRule.xls under [Installation]\CatalogData\BulkLoad\Datafiles
4. Remember to delete the existing record and add the letter A to the new record.
5. Add a new IFC rule name and the ProgID as shown here:

Head	RuleName	RuleProgID
Start		
d	Processor Rule_1	IFCRule.ProcessorRule
a	Processor Rule_1	IFCCustomRule.ProcessorRule
End		

6. Save the Excel sheet as IFCCustomRule.xls under c:\train\IFCRule. Exit Excel.
7. Run Bulkload Utility (START Menu -> Intergraph SmartPlant 3D -> Database Tools -> Bulkload Reference Data)
8. Set the bulkload to A/M/D mode.
9. Select Load button to add the new IFC rule into the training catalog.



10. Place/route some structures members, pipelines, cabletrays and standard equipments in the model such that they interfere with one-another as shown below:

    a) Piping against structure
    b) Equipment against structure
    c) Piping against cabletray
    d) Structure against cabletray

11. Navigate to *c:\train\IFCRule* and remove the Read-only flag from all files.

12. Open the IFCRule.vbp project.

13. Go to the Properties Window and change the name of the Project as shown here:

14. Go to the Visual Basic Explorer Window and select the Project node. Select *File -> Save Project As* option to save the project as IFCCustomRule.vbp under the *c:\train\IFCRule* directory.

15. Open the PostProcessorRule.cls class and navigate to the Class_Initialize subroutine.

16. Note the names of the permission groups:

```
m_strPermissionGroups(0) = "IFC Supports"
m_strPermissionGroups(1) = "IFC Conduits"
m_strPermissionGroups(2) = "IFC Cableway"
m_strPermissionGroups(3) = "IFC HVAC"
m_strPermissionGroups(4) = "IFC Piping"
m_strPermissionGroups(5) = "IFC Structure"
m_strPermissionGroups(6) = "IFC Equipment"
m_strPermissionGroups(7) = "IFC Volumes"
```

17. These are the names of the permission groups that the interferences will be assigned to. You will need to create these permission groups using the Project Management task.

18. Notice that in the Get Permission Index subroutine defines the ranking on the basis of which permission groups will be assigned.

```
Select Case (strParentType)

Case "Pipe Supports", "Cable Tray Supports", "Duct Supports"
   GetPermissionGroupIndex = 0

Case "Conduit Components", "Conduits"
   GetPermissionGroupIndex = 1

Case "Cable Tray Components", "Cableway Along Leg", "Cableway Straight", _
         "Cable Trays", "Cableway Turn"
   GetPermissionGroupIndex = 2

Case "HVAC Components", "Ducts"
   GetPermissionGroupIndex = 3

Case "Pipes", "Piping Welds", "Piping Components", "Piping Instruments", _
         "Piping Specialty Items"
   GetPermissionGroupIndex = 4

Case "Member Part Linear", "Member Part Curve", "Slab", _
         "Equipment Foundation", "Footing", "Stairs", "Ladders", "Handrails"
   GetPermissionGroupIndex = 5

Case "Legacy Equipment", "Legacy Designed Equipment", "Equipment"
   GetPermissionGroupIndex = 6
```

```
Case "Interference Volumes"
   GetPermissionGroupIndex = 7

Case Default
   GetPermissionGroupIndex = -1
End Select
```

Note This is the hierarchy of object types. If an object that is lower in the hierarchy (lower permissiongroupindex) interferes with an object higher in the hierarchy (higher permissiongroupindex), the interference will be assigned to the permission group of the object lower in the hierarchy.

19. Go to the IJDInterferenceRule_CreateInterference subroutine and uncomment the following lines:

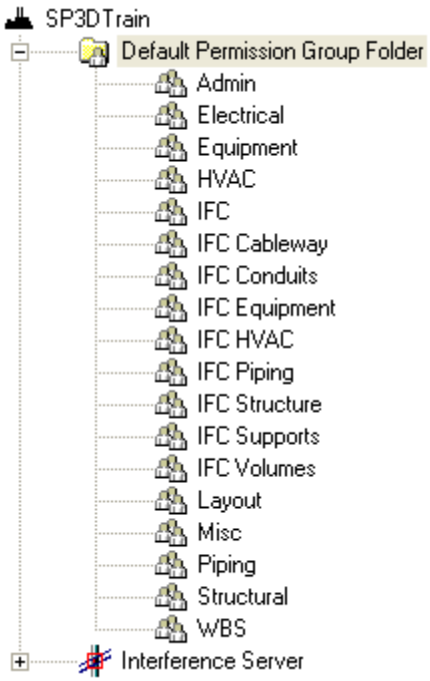
```
If IfcType = IfcServerInterference Then
'assign a permission group to the IFC object based on rule
   AssignIFCPermissionGroup pInterferenceObj, strParentType1, strParentType2
End If
```

20. Update the binary compatibility of your program to IFCCustomRule-ref.dll

    Note: One of the most important steps in Visual Basic programming is to preserve the binary compatibility of your program. Save the final version of your dll file to be binary compatibility in order to preserve the CLSID

21. Re-compile the program using File -> Make IFCCustomRule.dll

22. Start the Project Management task and make permission groups named "IFC Cableway", "IFC Piping" and "IFC Structure", etc. as shown here:



23. Start the interference check service.

24. Select the Plant and select permission group IFC as the group to assign interferences to.

25. Click Start to start the interference detection process. The process starts and begins running.

26. After 5 minutes, refresh/define workspace that includes objects placed in the above step. You should see interferences created between the objects.
27. Check the permission group of the interference object. You will see the following:

Interfering Objects	Permission Group
Piping against structure	IFC Piping
Equipment against structure	IFC Structure
Piping against cabletray	IFC Cableway
Structure against cabletray	IFC Cableway

28. Stop the interference detection process.

Lab 14: Interference object remark property

Objectives

After completing this lab, you will be able to:

- Add a note to the interference object based on the object type of the colliding objects

Add a note to the interference object remark property where the colliding objects are both structure objects (linear member part).

1. Open the IFCCustomRule.vbp project.

2. Go to the PostProcessorRule.cls

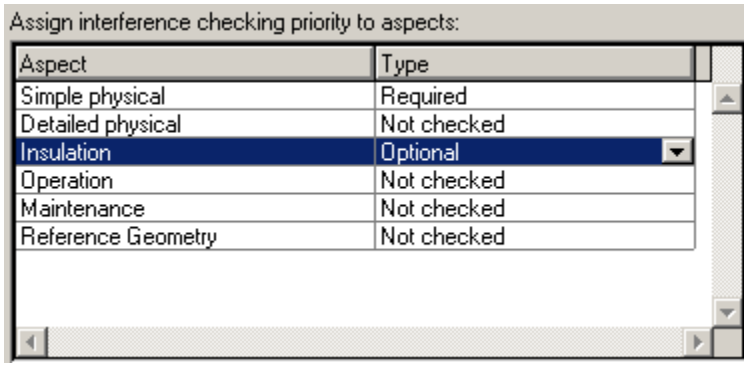
3. Go to the IJDInterferenceRule_CreateInterference subroutine and add the following lines

```
Dim strNotes As String
strNotes = ""
If IfcType = IfcServerInterference Then
   If strParentType1 Like "Member Part Linear" And _
      strParentType2 Like "Member Part Linear" Then
      strNotes = "Call Structure Design leader"
   pInterferenceObj.InterferenceRemark = strNotes
   End If
End If
```

4. Re-compile the program using File -> Make IFCCustomRule.dll

5. Start the interference check service.

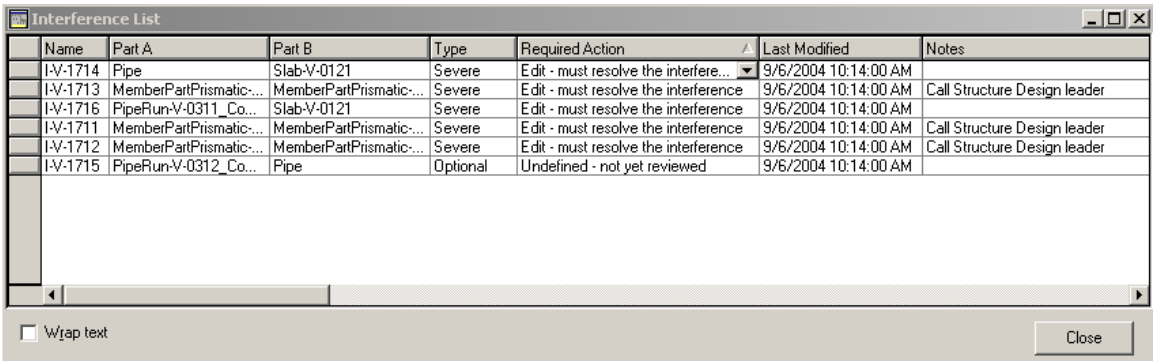
6. Change the interference checking process criteria by assigning the interference priority to the Insulation Aspect as Optional.



7. Click Start button to start the interference detection process. Select Yes to re-check the entire Model. Select OK button for the process starts and begins running.

8. After 5 minutes, refresh/define workspace. You should see interferences created between the objects. Stop the interference detection process.

9. Check the remark property of the interference objects as shown below:



Interference List

Name	Part A	Part B	Type	Required Action	Last Modified	Notes
I-V-1714	Pipe	Slab-V-0121	Severe	Edit - must resolve the interfere...	9/6/2004 10:14:00 AM	
I-V-1713	MemberPartPrismatic-...	MemberPartPrismatic-...	Severe	Edit - must resolve the interference	9/6/2004 10:14:00 AM	Call Structure Design leader
I-V-1716	PipeRun-V-0311_Co...	Slab-V-0121	Severe	Edit - must resolve the interference	9/6/2004 10:14:00 AM	
I-V-1711	MemberPartPrismatic-...	MemberPartPrismatic-...	Severe	Edit - must resolve the interference	9/6/2004 10:14:00 AM	Call Structure Design leader
I-V-1712	MemberPartPrismatic-...	MemberPartPrismatic-...	Severe	Edit - must resolve the interference	9/6/2004 10:14:00 AM	Call Structure Design leader
I-V-1715	PipeRun-V-0312_Co...	Pipe	Optional	Undefined - not yet reviewed	9/6/2004 10:14:00 AM	

Wrap text

Close

Lab 15: Interference rule for Handrails-to-Slab collisions

Objectives

After completing this lab, you will be able to:

- Add a rule to eliminate the creation of interference objects for Handrail-to-Slab collisions

Place handrails and grating slabs in the model such that they interfere with one-another.

1. Open the IFCCustomRule.vbp project.

2. Go to the PostProcessorRule.cls

3. Go to the IJDInterferenceRule_CreateInterference subroutine and add the following lines:

```
If IfcType = IfcServerInterference Then
    If HandrailClashGrating(strParentType1, strParentType2, pParent1, pParent2) Then
        IJDInterferenceRule_CreateInterference = False
        Exit Function
    End If
End If
```

4. Create a function called HandrailClashGrating() that return a Boolean value if there is an interference between handrail and slab of type grating.

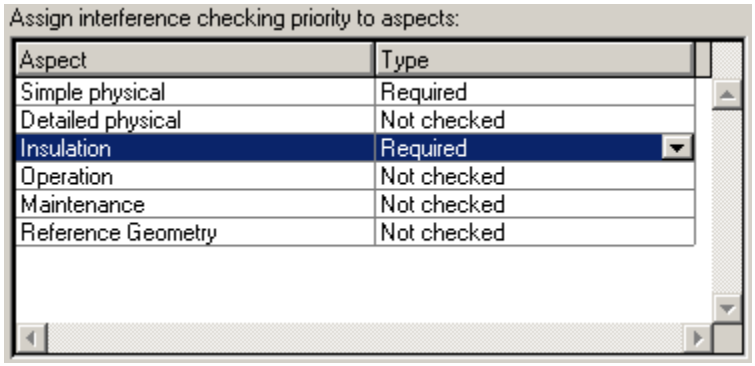
```
Private Function HandrailClashGrating(strObject1 As String _
                                      , strObject2 As String _
                                      , ByVal pParent1 As Object _
                                      , ByVal pParent2 As Object _
                                      ) As Boolean
On Error GoTo ErrHndlr

    HandrailClashGrating = False

    Dim oAttrbs As IJDAttributes
    Dim oRelationHelper As IMSRelation.DRelationHelper
    Dim oCollection As IMSRelation.DCollectionHelper
    Dim slabtype As String

    If (strObject1 Like "Slab" And strObject2 Like "Handrails") Then
    ' get the slab type from the object
        Set oRelationHelper = pParent1
        Set oCollection = oRelationHelper.CollectionRelations("ISPSSlabEntity",
"SlabEntityTypeReferenceRln_ORIG")
        If oCollection.count <> 0 Then
            Set oAttrbs = oCollection.Item(1)
```

```
            slabtype = oAttrbs.CollectionOfAttributes("IJDPart").Item("PartNumber").Value
            ' check if slab is of grating type
            If InStr(slabtype, "Grating") Then
                HandrailClashGrating = True
                Exit Function
            End If
        End If

    ElseIf (strObject1 Like "Handrails" And strObject2 Like "Slab") Then
        ' get the slab type from the object
        Set oRelationHelper = pParent2
        Set oCollection = oRelationHelper.CollectionRelations("ISPSSlabEntity",
"SlabEntityTypeReferenceRln_ORIG")
        If oCollection.count <> 0 Then
            Set oAttrbs = oCollection.Item(1)
            slabtype = oAttrbs.CollectionOfAttributes("IJDPart").Item("PartNumber").Value
            ' check if slab is of grating type
            If InStr(slabtype, "Grating") Then
                HandrailClashGrating = True
                Exit Function
            End If
        End If
    End If

Exit Function
ErrHndlr:
    Err.Clear
End Function
```

5. Re-compile the program using File -> Make IFCustomRule.dll

6. Start the interference check service.

7. Change the interference checking process criteria by assign the interference priority to the Insulation Aspect as Required.



8. Click Start button to start the interference detection process. Select Yes to re-check the entire Model. Select OK button for the process starts and begins running.

9. After 5 minutes, refresh/define workspace. You should see interferences created between the objects. The interference checking process should not create an interference object between handrails and slabs of type grating.

10. Stop the interference detection process.

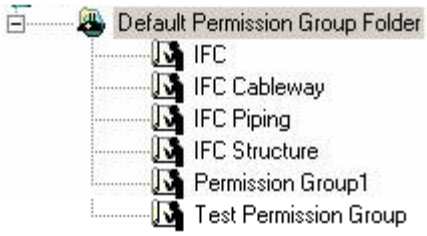
Lab 16: Interference Rule for objects belonging to a Test Permission Group

Objectives

After completing this lab, you will be able to:

- Add a rule to avoid creating interference objects where colliding objects belong to a test permission group

1. Start the Project Management task and create a permission group named "TESTPG”.



2. Go to the Equipment Task. Set the active permission group to “TESTPG”.
3. Place some standard equipments in the model such that they interfere with one-another.

4. Go to the Piping Task. Make sure the active permission group is set to “TESTPG”. Route some pipe runs in the model such that they interfere with any objects in the model.

5. Add a rule to avoid creating interference objects where colliding objects belong to “TESTPG”.

6. Open the IFCCustomRule.vbp project.

7. Go to the PostProcessorRule.cls

8. Go to the IJDInterferenceRule_CreateInterference subroutine and add the following lines:

```
If IfcType = IfcServerInterference Then
    If objectBelongToPG("TESTPG", pParent1, pParent2) Then
        IJDInterferenceRule_CreateInterference = False
        Exit Function
    End If
End If
```

9. Create a function called objectBelongToPG("TESTPG", pParent1, pParent2) that return a Boolean value if the colliding object belong to a Permission Group called “TESTPG”.

10. Create a Private Function with the following arguments:

```
Private Function objectBelongToPG(UPPERCASE_pgName_substring As String _
                                  , ByVal pParent1 As Object _
                                  , ByVal pParent2 As Object _
                                  ) As Boolean
```

11. Next, add an error handler statement

```
On Error GoTo ErrHndlr
```

12. Set the function to return a False Boolean value by default

```
objectBelongToPG = False
```

13. Declare and set the reference to the object variables:

```
Dim pNum1 As Long, pNum2 As Long
Dim strPgName1 As String, strPgName2 As String
Dim pObject1 As IJDObject
Dim pObject2 As IJDObject
Set pObject1 = pParent1
Set pObject2 = pParent2
```

14. Declare variables to store the Permission Group Name and Permission Group ID of the colliding objects.

```
Dim pNum1 As Long, pNum2 As Long
Dim strPgName1 As String, strPgName2 As String
```

15. Get the permission group ID from the two colliding objects

```
pNum1 = pObject1.PermissionGroup
pNum2 = pObject2.PermissionGroup
```

16. Get the permission group Name from the two colliding objects using a function called ConvertPGNumberToName(). The code for this function is documented later in this lab.

```
strPgName1 = ConvertPGNumberToName(pNum1)
strPgName2 = ConvertPGNumberToName(pNum2)
```

17. Write the test condition if the retrieved permission group name is “TESTPG”. If the condition is true then set the function to return a TRUE Boolean value.

```
   If InStr(UCase(strPgName1), UPPERCASE_pgName_substring) > 0 Or
InStr(UCase(strPgName2), UPPERCASE_pgName_substring) > 0 Then
      objectBelongToPG = True
      Exit Function
   End If
```

18. Add the exit statement to exit the function procedure

    Exit Function

19. When an error occurs at run time, add the following code to handle it:

```
Exit Function
ErrHndlr:
   Err.Clear
```

Note The function should look like this:

```
Private Function objectBelongToPG(UPPERCASE_pgName_substring As String _
                                  , ByVal pParent1 As Object _
                                  , ByVal pParent2 As Object _
                                  ) As Boolean
On Error GoTo ErrHndlr
   ' by default, foul will be created
   objectBelongToPG = False

   Dim pNum1 As Long, pNum2 As Long
   Dim strPgName1 As String, strPgName2 As String
   Dim pObject1 As IJDObject
   Dim pObject2 As IJDObject
   Set pObject1 = pParent1
   Set pObject2 = pParent2

   ' getting permission group name for parts
   pNum1 = pObject1.PermissionGroup
   pNum2 = pObject2.PermissionGroup
   strPgName1 = ConvertPGNumberToName(pNum1)
   strPgName2 = ConvertPGNumberToName(pNum2)

   If InStr(UCase(strPgName1), UPPERCASE_pgName_substring) > 0 Or InStr(UCase(strPgName2),
UPPERCASE_pgName_substring) > 0 Then
      objectBelongToPG = True
      Exit Function
   End If

Exit Function
ErrHndlr:
   Err.Clear
End Function
```

20. Next, write a Private Function that gets the permission group id and return the corresponding permission group name.

21. Create a Private Function with the following arguments:

    Private Function ConvertPGNameToNumber(ByVal strPFName As String) As Long

23. Next, add an error handler statement

```
On Error GoTo ErrHndlr
```

24. Declare variables to store temporary a Permission Group Name and a Permission Group ID. Also, declare a variable to store the total number of permission group (count) define in the model.

```
Dim CID As Long
Dim count As Long
Dim CIDName As String
Dim acc As Long
Dim i As Long
```

25. User the IJAccessControlConfiguration interface and the ApplicationContext service to retrieve the Access Control Configuration information for the current model. Declare and set the reference to the object variables:

```
Dim oMidCtx As IJMiddleContext
Dim oDBTypeConfig As IJDBTypeConfiguration
Dim oDataBaseConfig As IJDataBaseConfiguration
Dim oACConfig As IJAccessControlConfiguration
Dim oAccessControl As IJAccessControl
Dim bFound As Boolean

Set oDBTypeConfig = New DBTypeConfiguration
Set oDataBaseConfig = New DataBaseConfiguration
Set oACConfig = New AccessControlConfiguration

Set oMidCtx = New GSCADMiddleContext 'should come with initialzied one
oMidCtx.GetConfigurationTablesFromMiddle oDBTypeConfig, oDataBaseConfig, oACConfig
Set oAccessControl = oACConfig.AccessControl
count = oACConfig.NumberConditionIDs
```

26. Create a loop to go through the permission group list. Set a Boolean variable to TRUE if the permission group id is found in the list and return the corresponding permission group name.

```
For i = 1 To count
   oACConfig.GetConditionIDByIndex i, CIDName, CID
   oAccessControl.GetAccessRight CID, acc
   If ((acc And acUpdate) = acUpdate) Then
      If CIDName Like strPFName Then
         On Error Resume Next
         ConvertPGNameToNumber = CID
         bFound = True
         Exit For
      End If
```

```
        Else
            bFound = False
        End If
    Next i
```

27. If the permission group id is not found in the list, return No found string.

```
    'Could not find any permission group .. assingning 0
    If bFound = False Then
        ConvertPGNameToNumber = 0
    End If
```

28. Add the exit statement to exit the function procedure

```
    Exit Function
```

29. When an error occurs at run time, add the following code to handle it:

```
  ErrHndlr:
Err.Raise Err.Number
Debug.Assert False
```

Note The function should look like this:

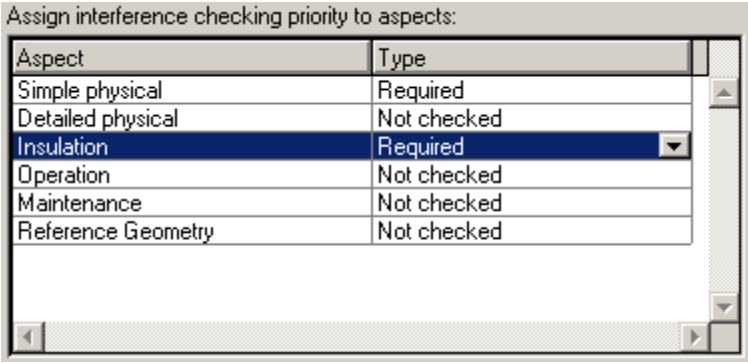
```
'Gets the PG number and return the corresponding permission group string
Private Function ConvertPGNumberToName(ByVal PGnum As Long) As String
On Error GoTo ErrHndlr
    Dim PGID As Long
    Dim count As Long
    Dim pgName As String
    Dim i As Long

    Dim oMidCtx As IJMiddleContext
    Dim oDBTypeConfig As IJDBTypeConfiguration
    Dim oDataBaseConfig As IJDataBaseConfiguration
    Dim oACConfig As IJAccessControlConfiguration
    Dim oAccessControl As IJAccessControl

    ' default
    ConvertPGNumberToName = ""

    Set oDBTypeConfig = New DBTypeConfiguration
    Set oDataBaseConfig = New DataBaseConfiguration
    Set oACConfig = New AccessControlConfiguration

    Set oMidCtx = New GSCADMiddleContext
    oMidCtx.GetConfigurationTablesFromMiddle oDBTypeConfig, oDataBaseConfig, oACConfig
    Set oAccessControl = oACConfig.AccessControl
    count = oACConfig.NumberConditionIDs
    For i = 1 To count
```

```
        oACConfig.GetConditionIDByIndex i, pgName, PGID
        If PGID = PGnum Then
            ConvertPGNumberToName = pgName
            Exit Function
        End If
    Next i

Exit Function
ErrHndlr:
    Err.Clear
    ConvertPGNumberToName = ""
End Function
```

30. Re-compile the program using File -> Make IFCCustomRule.dll

31. Start the Check interference check service.

32. Change the interference checking process criteria by assign the interference priority to the Insulation Aspect as Required.



33. Click Start button to start the interference detection process. Select Yes to re-check the entire Model. Select OK button for the process starts and begins running.

34. After 5 minutes, refresh/define workspace. You should see interferences created between the objects. The interference checking process should not create an interference object where colliding objects belong to “TESTPG”.

35. Stop the interference detection process.

Appendix

NamingRulesHelper Object

This is the helper object that implements the IJDNamingRulesHelper interface to query the naming rules for an object type, to create naming relations, and to query for the active naming rule. This is implemented in the middle tier so that both application commands and business objects can use this implementation.

References
Object Library: Ingr Sp3d Generic NamingRules Helper 1.0

Interfaces

Interface Name	lang	Description
IJDNamingRulesHelper	vb/c	This is the helper interface with the methods that can be used by application commands and business objects for defining naming rules for their objects.

IJDNamingRulesHelper
This is a helper interface that can be used to query the naming rules for an object type, to create naming relations, and to query for the active naming rule. The functionality of this interface is accessed by adding a project reference to the "Ingr Sp3d Generic NameRuleSemantics 1.0 Type Library".
This interface inherits from IDispatch.

When To Use
The Visual Basic® NamingRulesHelper Object implements all of the helper functions. This implementation can be used as long as the applications are using the generic naming rules semantic.

Methods

GetEntityNamingRulesGivenName (byval strEntityName as String) as IJElements

Description:	It returns a reference (as NamingRules) to the IJElements interface of the first object in a collection of the naming rules available in the catalog database for the given object name input.
Parameters:	
[in] strEntityName	Class(object) name(internal name).

GetEntityNamingRulesGivenProgID (byval strEntityProgID as String) as IJElements

Description:	It returns a reference (as NamingRules) to the IJElements interface of the first object in a collection of the naming rules available in the catalog database for the given object class ProgID input.
Parameters:	
[in] strEntityProgID	Object class ProgID.

AddNamingRelations (byval pDispEntity as Object , byval pNameRuleHolder as IJDNameRuleHolder) as IJNameRuleAE

Description:	Adds naming relations "NamedEntity" and "EntityNamingRule" after creating the Active Entity and returns a reference (as pActiveEntity) to the interface of the active entity object created. The method deletes the Active Entity if it is there before creating the new one so it can also be used to delete the relations. If nothing is sent as the pNameRuleHolder argument, the method deletes the existing relations.
Parameters:	
[in] pDispEntity	The IDispatch interface of the object to be named.
[in] pNameRuleHolder	The interface of the NamingRule.

GetActiveNamingRule (byval pDispEntity as Object) as IJDNameRuleHolder

Description:	This method returns a reference (as pNameRuleHolder) to the interface of the active naming rule that is being used for naming the input object from the relations. pNameRuleHldr will be nothing if there are no active naming rules on the object.
Parameters:	
[in] pDispEntity	The IDispatch interface of the named object.

IsGeneratedNameUnique (byval oEntity as LPDISPATCH , byval oFilter as IJSimpleFilter , byval strGenName as String , optional byval strIID as String , optional byval strAttributeName as String) as Boolean

Description:	This method returns a boolean value (as pVal) indicating whether the generated name is unique in the domain specified by the user through the oFilter. True indicates the name is unique. The optional arguments strIID and strAttribute Name are to be provided by the users of this function. They are provided so as to give an option to the user to specify the Interface and also the Attribute of the object on which the name uniqueness has to be ensured.
Parameters:	
[in] oEntity	The IDispatch interface of the named object.
[in] oFilter	The interface of the Filter to use in determining the uniqueness.
[in] strGenName	The generated name string.
[in] strIID	An optional IID as a string to help in making the determination. If the IID is provided then strAttributeName has to be provided. Default value is null string.
[in] strAttributeName	An optional AttributeName as a string to help in making the determination. Default value is null string.
Return error codes:	
E_FILTER_NOT_SPECIFIED	The Filter was not specified.

Attribute Helper service

CollectionHlp
The role of this object is to operate on one instantiated collection of attributes. A CollectionHlp object is returned by most of the methods of the IJDAttributes and IJAttributes interfaces. A collection of attributes maps to an interface definition, i.e., it gathers all the properties that belong to an interface.

References
Object Library: Ingr SmartPlant 3D Attributes 1.0 Type Library

Interfaces

Interface Name	lang	Description
IJDAttributesCol	vb/c	Visual Basic® Interface used to manipulate a collection of attributes.

IJDAttributesCol
This interface is used to get information from an item or items in a collection of attributes.
This interface inherits from IDispatch.

When To Use
Call this interface when you want to:
Access an item of a collection of attributes.
Access all the items of a collection of attributes.
Count the items of a collection.
Get the metadata about a collection of attributes.

Properties

Item (byval VItem as Variant) as IJDAttribute

Description:	Returns the IJDAttribute interface of the attribute as ppAttribute. Note that: The For Each loop is the preferred implementation to iterate through a collection instead of using a simple index because the DispatchID is NOT a sequential list (1, 2, 3, ...).
Modifiability:	Read Only
Parameters:	
[in] VItem	The VItem can be the DispatchID of the attribute or its name.
Return error codes:	
S_OK	Operation succeeded.
E_FAIL	Operation failed (no detail).

_EnumItem ( ) as LPUNKNOWN

Description:	Enumerates all the attributes of this collection by returning ppEnumUnk.
Modifiability:	Read Only
Return error codes:	
S_OK	Operation succeeded.
E_FAIL	Operation failed (no detail).

InterfaceInfo ( ) as IJDInterfaceInfo

Description:	Returns ppInfo, the IJDInterfaceInfo interface of an InterfaceInfo Object for this collection.
Modifiability:	Read Only
Return error codes:	
S_OK	Operation succeeded.
E_FAIL	Operation failed (no detail).

Count ( ) as Long

Description:	Returns the number of attributes of this Collection.
Modifiability:	Read Only
Return error codes:	
S_OK	Operation succeeded.
E_FAIL	Operation failed (no detail).

IJDAttributes

This interface is used to get a CollectionOfAttributes property. This interface is implemented by any component that is attributes-enabled and aggregates the AttributeHelper object.

When To Use

Call this interface when you want to access the CollectionOfAttributesproperty of an object.

Properties

CollectionOfAttributes(byval InterfaceType as Variant) as IJDAttributesCol

Description:	Returns a pointer (ppIAttributesCol) to the IJDAttributesCol interface of the CollectionHlp Object (collection of attributes). If the UserTypeCLSID property was set to an acceptable value, the method checks to see that this collection is allowed for this UserType according to the metadata. If UserTypeCLSID is set to CLSID_NULL, the method only checks to see that this collection/Interface is described in the metadata.
Modifiability:	Read Only
Parameters:	
[in] InterfaceType	The InterfaceType is a variant that contains a string with the formatted hexa value of the IID : "{24E1A26B-1275-11d2-A684-00A0C96F81B9}", or with the interface name IID : "IJGeometry", or a GUID structure.
Return error codes:	
S_OK	Operation succeeded.
E_FAIL	Operation failed (no detail).
E_NOINTERFACE	The interface is not implemented by the UserType class. The AttributesCol is set to NULL in this case.

Count ( ) as Long

Description:	Returns the number of collections of this object.
Modifiability:	Read Only
Return error codes:	
S_OK	Operation succeeded.
E_FAIL	Operation failed (no detail).

Attribute

The role of this object is to operate on one instantiated attribute. The Attribute object is returned by most of the methods of the IJDAttributesCol interface.

References

Object Library: Ingr SmartPlant 3D Attributes 1.0 Type Library

Interfaces

Interface Name	lang	Description
IJDAttribute	vb/c	Visual Basic® Interface used to manipulate an attribute

IJDAttribute

This interface is used to manipulate the value of an attribute.

This interface inherits from IDispatch.

When To Use
Call this interface when you want to:
Access the value of an attribute.
Get the metadata about an attribute.

Properties

Value ( ) as Variant

Description:	Allows you to get or set the value of an attribute. The method using this property is the generic way to access the value of an attribute. It is not responsible to check and see if the caller is allowed to write in this field. If one uses put_Value with Val.vt = VT_NULL or VT_EMPTY, the attribute is removed from the database. For Hierarchical Code Lists, if one uses put_Value with val.vt = VT_BSTR (implying that the ShortString value has been passed), it is automatically converted to the ValueID (val.vt = VT_I4). If one uses get_Value on a removed attribute, the returned variant will have its vt flag set to VT_EMPTY. This confusion of the VT_EMPTY and VT_NULL flag allows us to save database space. See the Specific Types Definition below for the definitions.
Modifiability:	Read/Write
Return error codes:	
S_OK	Operation succeeded.
E_FAIL	Operation failed (no detail).

AttributeInfo ( ) as IJDAttributeInfo

Description:	Returns the IJDAttributeInfo interface of an AttributeInfo object for this attribute.
Modifiability:	Read Only
Return error codes:	
S_OK	Operation succeeded.
E_FAIL	Operation failed (no detail).

Specific Types Definition

```
Enum tagSQLTypes
SQL_VB_CHAR = 1          // CHAR, VARCHAR, DECIMAL, NUMERIC = VT_BSTR =
                         SQL_C_CHAR = SQL_CHAR
SQL_VB_LONG = 4          // long int = VT_I4 = SQL_C_LONG = SQL_INTEGER
SQL_VB_SHORT = 5         // shrt int = VT_I2 = SQL_C_SHORT = SQL_SMALLINT
SQL_VB_FLOAT = 7         // float = VT_R4 = SQL_C_FLOAT = SQL_REAL
SQL_VB_DOUBLE = 8        // double = VT_R8 = SQL_C_DOUBLE = SQL_DOUBLE
SQL_VB_BIT = -7          // boolean = VT_BOOL = SQL_C_BIT
SQL_VB_DATE = 9          // date = VT_DATE = SQL_C_DATE
End Enum
```

Note about tagSQLTypes : The type of the attribute is defined in the METADATALib in terms of SQL_C_Types. The value of an attribute is a VARIANT. We use the correspondence table above. If the type of the VARIANT does not match the VT type, we try to coerce it using MS API VariantChangeType. If the attribute is hard coded, the coercion is done by the MS API invoke.

IJDCodeListMetaData
This interface is used to access the codelist metadata and is exported in the COM map of the business object that aggregates the attribute helper. The method calls are delegated to the POM.
This interface inherits from IDispatch.

When To Use
Call this interface when you want to access the metadata about a codelist.

Properties

ShortStringValue (byval TableName as String , byval ValueID as Long) as String

Description:	Gets the short string of a codelist.
Modifiability:	Read Only
Parameters:	
[in] TableName	Name of the table. Can be either Namespace:TableName (i.e., PackageName:TableName) or just TableName. When there are two tables with same name in different packages and no namespace is specified, an error will be returned.
[in] ValueID	Index of the codelist in the table.
Return error codes:	
S_OK	Operation succeeded, ShortString returned.
S_FALSE	Operation succeeded, no ShortString returned.
E_FAIL	(1) No TableName is provided; (2) Duplicated TableNames are found in Metadata database (need Namespce); (3) Operation failed for other reasons.
Note:	This API returns S_FALSE if the CodelistTable does not exist or the CodelistTable does not have ValueID as its value.

LongStringValue (byval TableName as String , byval ValueID as Long) as String

Description:	Gets the long text string of a codelist.
Modifiability:	Read Only
Parameters:	
[in] TableName	Name of the table. Can be either Namespace:TableName (i.e., PackageName:TableName) or just TableName. When there are two tables with same name in different packages and no namespace is specified, an error will be returned.
[in] ValueID	Index of the codelist in the table.
Return error codes:	
S_OK	Operation succeeded, longString returned.
S_FALSE	Operation succeeded, no longString returned.
E_FAIL	(1) No TableName is provided; (2) Duplicated TableNames are found in Metadata database (need Namespce); (3) Operation failed for other reasons.
Note:	This API returns S_FALSE if the CodelistTable does not exist or the CodelistTable does not have ValueID as its value.

ParentValueID (byval TableName as String , byval ValueID as Long) as Long

Description:	Gets the ParentValueID of a codelist. Returns -1 in case a valid ValueID does not have a ParentValueID.
Modifiability:	Read Only
Parameters:	
[in] TableName	Name of the table. Can be either Namespace:TableName (i.e., PackageName:TableName) or just TableName. When there are two tables with same name in different packages and no namespace is specified, an error will be returned.
[in] ValueID	Index of the codelist in the table.
Return error codes:	
S_OK	Operation succeeded, ParentValueID returned.
S_FALSE	Operation succeeded, no ParentValueID returned.
E_FAIL	(1) No TableName is provided; (2) Duplicated TableNames are found in Metadata database (need Namespce); (3) Operation failed for other reasons.
Note:	This API returns S_FALSE if the CodelistTable does not exist or the CodelistTable does not have ValueID as its value.

CodelistValueCollection (byval TableName as String) as IJDInfosCol

Description:	Returns (pEnumCodeList as RetVal) the IJDInfosCol interface of the first item of the collection of tables. The IJDInfosCol is a collection of IJDCodelistValue.
Modifiability:	Read Only
Parameters:	

[in] TableName	Name of the table. Can be either Namespace:TableName (i.e., PackageName:TableName) or just TableName. When there are two tables with same name in different packages and no namespace is specified, an error will be returned.
Return error codes:	
S_OK	Operation succeeded.
E_INVALIDARG	No TableName provided.
E_FAIL	(1) Duplicated TableNames are found in Metadata database (need Namespce); (2) Operation failed for other reasons.
Note:	This API returns a codelist value collection cotaining "Unidentified" if a non-existing Codelist table name is passed in.

ChildValueCollection (byval TableName as String , byval ValueID as Long) as IJDInfosCol

Description:	Returns (pEnumCodeList as RetVal) the IJDInfosCol interface of the first item of the collection of tables associated with a specific ValueID. The IJDInfosCol
Modifiability:	Read Only
Parameters:	
[in] TableName	Name of the table. Can be either Namespace:TableName (i.e., PackageName:TableName) or just TableName. When there are two tables with same name in different packages and no namespace is specified, an error will be returned.
[in] ValueID	Index of the codelist in the table.
Return error codes:	
S_OK	Operation succeeded.
S_FALSE	TableName does not have a ChildTable.
E_FAIL	(1) TableName has duplicates in Metadata; (2) Operation failed for other reasons (no detail).

ParentTable (byval TableName as String) as String

Description:	Gets ParentTable name of a given a codelist table.
Modifiability:	Read Only
Parameters:	
[in] TableName	Name of the table. Can be either Namespace:TableName (i.e., PackageName:TableName) or just TableName. When there are two tables with same name in different packages and no namespace is specified, an error will be returned.
Return error codes:	
S_OK	Operation succeeded, ParentTable returned.
S_FALSE	Operation succeeded, no ParentTable returned.
E_CL_TABLENAMEDUPLICATED	TableName has duplicates in Metadata database.
E_FAIL	More than one ParentTable name is found (require namespace); Operation failed (no detail).

ChildTable (byval TableName as String) as String

Description:	Gets ChildTable name of a given a codelist table.
Modifiability:	Read Only
Parameters:	
[in] TableName	Name of the table. Can be either Namespace:TableName (i.e., PackageName:TableName) or just TableName. When there are two tables with same name in different packages and no namespace is specified, an error will be returned.
Return error codes:	
S_OK	Operation succeeded, ChildTable returned.
S_FALSE	Operation succeeded, no ChildTable returned.
E_CL_TABLENAMEDUPLICATED	TableName has duplicates in Metadata database.
E_FAIL	More than one ChildTable name is found (require namespace); Operation failed (no detail).

TableDescription (byval TableName as String) as String

Description:	Gets the description of the codelist table.
Modifiability:	Read Only
Parameters:	
[in] TableName	Name of the table. Can be either Namespace:TableName (i.e., PackageName:TableName) or just TableName. When there are two tables with same name in different packages and no namespace is specified, an error will be returned.
Return error codes:	
S_OK	Operation succeeded, TableDescription returned.
S_FALSE	Operation succeeded, no TableDescription returned.
E_CL_TABLENAMEDUPLICATED	TableName has duplicates in Metadata database.
E_FAIL	More than one ChildTable name is found (require namespace); Operation failed (no detail).

TableCollection ( ) as Unknown

Description:	Returns (pEnumCodeList as RetVal) the IUnknown interface of the first item of the collection of tables. Gets an enumerated collection of CodeList tables.
Modifiability:	Read Only
Return error codes:	
S_OK	Operation succeeded.
E_FAIL	Operation failed (no detail).
Note:	This API returns S_OK no matter if a TableCollection is reurned or not.

ValueIDByShortString (byval TableName as String , byval ShortStringValue as String) as Long

Description:	Returns the ValueID of a codelist entry given the codelist TableName and the ShortStringValue of the entry.
Modifiability:	Read Only
Parameters:	
[in] TableName	Name of the table. Can be either Namespace:TableName (i.e., PackageName:TableName) or just TableName. When there are two tables with same name in different packages and no namespace is specified, an error will be returned.
[in] ShortStringValue	The short string value of a codelist.
Return error codes:	
S_OK	Operation succeeded, ValueId returned.
S_FALSE	Operation succeeded, no ValueId returned.
E_INVALIDARG	No TableName or ShortString is provided.
E_FAIL	More than one TableName is found in Metadata database (require namespace); Operation failed (no detail).

Relation Helper service

DRelationHelper

In the MS repository model of relationships, the Automation object CollectionHelper can be retrieved from any component that is relationships-enabled by getting the CollectionRelations property of the interface that the relationship is established to.

References
Object Library: Ingr SmartPlant 3D Relation 1.0 Type Library

Interfaces

Interface Name	lang	Description
IJDAssocRelation	vb/c	Visual Basic® Interface used to access a CollectionOfRelations property.
IJDTargetObjectCol	vb/c	Dual interface to manipulate the collection of target objects.
IJDRelationshipCol	vb/c	Dual interface to manipulate the collection of relationships.

IJDAssocRelation
This interface accesses the Collection of Relations in which a business object participates. It should be implemented by any business object that is relationship-enabled.
The relationship types are defined between interfaces of the two participant objects, and that relationships are gathered per homogenous collections. The Core uses this alternative accessor as an interface on the business object where both the interface and the property are input arguments when asking for the collection. This interface inherits from IDispatch.

When To Use
Call this interface when you want to access a collection of relationships on a business object.

Properties

CollectionRelations (byval InterfaceID as Variant , byval CollectionName as String) as Object

Description:	Returns the IDispatch interface of the Collection of relationships. This collection should implement the interfaces IJDRelationshipCol and IJDTargetObjectCol. If using the provided RelationHelper Object, the returned object is of the type CollectionHelper Object.
Modifiability:	Read Only
Parameters:	
[in] InterfaceID	IID that the collection is associated to. This variant contains a string with the formatted hexa value of the IID : "{24E1A26B-1275-11d2-A684-00A0C96F81B9}" or with the interface name IID : "IJGeometry", or a GUID structure.
[in] CollectionName	Name of the collection.
Return error codes:	
S_OK	Operation succeeded.
E_FAIL	Operation failed (no detail).

IJDRelationshipCol
This is one of the two basic interfaces that collections of relationships should implement.
This interface inherits from IDispatch.

When To Use
Use this interface to manage the relationships that belong to a particular relationship collection. This includes the set of relationships that:
Is of the same type.
Is attached to a particular source object.

Have objects playing the same role, have the same origin, or the same destination in the relationship.
With this interface, you can:
Get a count of the number of relationships in the collection.
Add and remove relationships to and from the collection.
If the collection is sequenced (which requires it to be an origin collection), place a relationship in a specific spot in the collection sequence or modify the sequencing of the collection.
Retrieve a specific relationship from the collection.
Obtain information about the collection and the relation to which it is associated.

Methods

Add (byval TargetObject as Unknown , byval Name as String) as IJDRelationship

Description:	Adds a relationship between the source object containing this collection of relationships and the given target object. Returns the IJDRelationship interface (CreatedRelationship) of the created relationship. If the business object is aggregating a RelationHelper Object, this object is a RelationshipHelper Object. Following the Repository API, if the relationship is of the ordered type, the added relationship is always added at the end of the existing ones.
Parameters:	
[in] TargetObject	Target Object to be connected.
[in] Name	Name of the relationship. This requires the relation to support naming.
Return error codes:	
S_OK	Operation succeeded.
S_FAIL	Operation failed (no detail).
E_OBJECTS_NOT_WITHIN_SAME_DB	The error is returned when DBContainment flag on relation metadata is WITHIN_DB and a relation is being created between objects belonging to different databases.

Insert (byval TargetObject as Unknown , byval Index as Long , byval Name as String) as IJDRelationship

Description:	Adds a relationship between the source object containing this collection of relationships and the given target object. Returns the IJDRelationship interface (CreatedRelationship) of the inserted relationship. If the business object is aggregating a RelationHelper Object, this object is a RelationshipHelper Object. This method can only be used when the origin side of the relation supports ordering.
Parameters:	
[in] TargetObject	Target object to be connected.
[in] Index	Index of the new relationship.
[in] Name	Name of the relationship.
Return error codes:	
S_OK	Operation succeeded.
S_FAIL	Operation failed (no detail).

IsSourceOrigin ( )

Description:	Returns if the source (i.e., the object that the collection has been retrieved from) is the origin of the relationships contained by the collection.
Return error codes:	
S_OK	Source is origin in the relationships.
S_FALSE	Source is destination in the relationships.

Remove (byval TargetItem as Variant)

Description:	Remove a relationship.
Parameters:	
[in] TargetItem	Identifies the Relationship to be removed by an index of type long or by a string (BSTR) when the relation supports unique naming and requires the collection to be the origin of the relation.

Return error codes:

S_OK	Operation succeeded.
E_FAIL	Operation failed (no detail).

Move (byval oldIndex as Long , byval newIndex as Long)

Description:	Move a relationship in a sequenced origin colelction.
Parameters:	
[in] oldIndex	Identifies the relationship to be moved by it's index.
[in] newIndex	Identifies the index to which the relation should be moved.
Return error codes:	
S_OK	Operation succeeded.
E_FAIL	Operation failed (no detail).

Refresh ( )

Description:	Refresh the collection with the current data from the database.
Return error codes:	
S_OK	Operation succeeded.
E_FAIL	Operation failed (no detail).
Note:	That method refreshs only a non associative collection. The method does nothing for an associative relation.

Properties

Count ( ) as Long

Description:	Returns the count of relationships.
Modifiability:	Read Only
Return error codes:	
S_OK	Operation succeeded.
S_FAIL	Operation failed (no detail).

Infos (InterfaceID as Variant , pCollectionName as String)

Description:	Returns the name of the collection and the interface that the collection is associated to.
Modifiability:	Read Only
Parameters:	
[out] InterfaceID	The IID of the interface with which the collection is associated.
[out] pCollectionName	The name of the collection.
Return error codes:	
S_OK	Operation succeeded.
S_FAIL	Operation failed (no detail).

Item (byval TargetItem as Variant) as IJDRelationship

Description:	Returns the IJDRelationship interface of an object describing the requested relationship. If using the provided helpers, this object is a RelationshipHelper.
Modifiability:	Read Only
Parameters:	
[in] TargetItem	Either the name or the index.
Return error codes:	
S_OK	Operation succeeded.
S_FAIL	Operation failed (no detail).
Note:	The TargetItem value identifies the relationship to be returned by a string (BSTR) when the relation supports unique naming and requires the collection to be origin of the relation or by an index of type long.

ItemByKey (byval Key as String) as IJDRelationship

Description:	Returns the IJDRelationship interface of an object describing the requested relationship. If using the provided helpers, this object is a RelationshipHelper.

Modifiability: Read Only
Parameters:
[in] Key The relation key relative to the origin collection.
Return error codes:
S_OK Operation succeeded.
S_FAIL Operation failed (no detail).
Note: This property requires the collection to be the origin of the relation.

Source ( ) as Unknown
Description: Returns the IUnknown interface of the source object. This is the object that the collection of relationships is associated to.
Modifiability: Read Only
Return error codes:
S_OK Operation succeeded.
S_FAIL Operation failed (no detail).

Type ( ) as Variant
Description: Returns the GUID identifying the relation to which the current collection is associated. Then the interface IJRelationMetaData on the source of the collection permits access to the complete meta-data information of this relation type.
Modifiability: Read Only
Return error codes:
S_OK Operation succeeded.
S_FAIL Operation failed (no detail).

IJDTargetObjectCol

This is one of the two basic interfaces that collections of relationships should implement.
With this interface, you can:

Get a count of the number of destinations in the collection.
Add and remove relationships to and from the collection.
If the collection is sequenced (which requires it to be an origin collection), place a relationship in a specific spot in the collection sequence, or modify the sequencing of the collection.
Retrieve a specific relationship from the collection.
Obtain information about the collection and the relation with which it is associated.

This interface inherits from IDispatch.

When To Use

Use this interface to manage the objects that are the destination of a particular relationship collection. This is the set of objects that are related to the source object (from which the current collection has been retrieved) by relationships:
of the same type.
attached to this particular source object.
where the objects in the relationship play the same role, origin, or destination.

<u>Methods</u>

Add (byval TargetObject as Unknown , byval Name as String , byval CreatedRelationship as IJDRelationship)
Description: Adds a relationship between the source object containing this collection of relationships and the given target object. Following the Repository API, if the relationship is of the ordered type, the added relationship is always added at the end of the existing ones.
Parameters:

[in] TargetObject	Target Object to be connected.
[in] Name	Name of the relationship.
[in] CreatedRelationship	Pointer to the created relationship. If the business object is aggregating a RelationHelper , this object is a RelationshipHelper.

Return error codes:

S_OK	Operation succeeded.
E_FAIL	Operation failed (no detail).
E_OBJECTS_NOT_WITHIN_SAME_DB	The error is returned when DBContainment flag on relation metadata is WITHIN_DB and a relation is being created between objects belonging to different databases.

Insert (byval TargetObject as Unknown , byval Index as Long , byval Name as String , byval CreatedRelationship as IJDRelationship)

Description: Adds a relationship between the source object containing this collection of relationships and the given target object. This method could only be used when the origin side of the relationship supports ordering.

Parameters:

[in] TargetObject	Target object to be connected.
[in] Index	Index of the new relationship.
[in] Name	Name of the relationship.
[in] CreatedRelationship	Pointer to the created relationship. If the business object is aggregating a RelationHelper, this object is a RelationshipHelper.

Return error codes:

S_OK	Operation succeeded.
E_FAIL	Operation failed (no detail).

IsSourceOrigin ( )

Description: Returns if the source (i.e., the object that the collection has been retrieved from) is the origin of the relationships contained by the collection.

Return error codes:

S_OK	Source is origin in the relationships.
S_FALSE	Source is destination in the relationships.

Move (byval ActualIndex as Long , byval NewIndex as Long)

Description: Moves the relationship to another location (for sequenced relations).

Parameters:

[in] ActualIndex	The index before the move where it actually is.
[in] NewIndex	The index to move it to.

Return error codes:

S_OK	Operation succeeded.
S_FAIL	Operation failed (no detail).

Remove (byval TargetItem as Variant)

Description: Removes a relationship.

Parameters:

[in] TargetItem	Identifies the Relationship to be removed by: - a string (BSTR) when the relation supports unique naming (requiring the collection to be the origin of the relation). - an index (long).

Return error codes:

S_OK	Operation succeeded.
S_FAIL	Operation failed (no detail).

EnumTargetMoniker (byval ppEnumMoniker as LPENUMMONIKER *)

Description: Enumerates monikers of target objects.

Parameters:

[in] ppEnumMoniker	Enumerates monikers of target objects. This enumeration will be sometimes useful in

avoiding binding all target objects. This enumeration can be used in VB also (see code example below).

Return error codes:

S_OK	Operation succeeded.
S_FAIL	Operation failed (no detail).

Properties

Count ( ) as Long

Description: Returns the count of target entities.

Modifiability: Read Only

Return error codes:

S_OK	Operation succeeded.
S_FAIL	Operation failed (no detail).

Infos (byval InterfaceID as Variant) as String

Description: Returns the name of the collection and the interface that the collection is associated to.

Modifiability: Read Only

Parameters:

[in] InterfaceID	The InterfaceID value passed out is the IID of the interface with which the collection is associated.

Return error codes:

S_OK	Operation succeeded.
S_FAIL	Operation failed (no detail).

Item (byval TargetItem as Variant) as Unknown

Description: Returns the IUnknown interface of a target object.

Modifiability: Read Only

Parameters:

[in] TargetItem	TargetItem value passed in identifies the Relationship to be removed by: - a string (BSTR) when the relation supports unique naming (requiring the collection to be the origin of the relation). - an index (long).

Return error codes:

S_OK	Operation succeeded.
E_ACCESSDENIED	Access to the target is denied.
E_FAIL	Operation failed (no detail).

Source ( ) as Unknown

Description: Returns the IUnknown interface of the source object.

Modifiability: Read Only

Return error codes:

S_OK	Operation succeeded.
E_FAIL	Operation failed (no detail).

Type ( ) as Variant

Description: Returns the GUID identifying the relationship with which the current collection is associated. Then use the interface IJRelationMetaData on the source of the collection to have access to the complete metadata information of this relation type.

Modifiability: Read Only

Return error codes:

S_OK	Operation succeeded.
E_FAIL	Operation failed (no detail).

SP3D References Tool

The software consists of hundreds of type libraries that provide the programmatic interfaces to the data model and its underlying data. These libraries consist of the data model's interfaces and their methods and properties.

The ability to integrate user-definable components into the environment is a key capability of the software. The mechanism of creating custom commands provides this extensibility.
To reference the available type libraries in Visual Basic:

- Click **Project > References**.

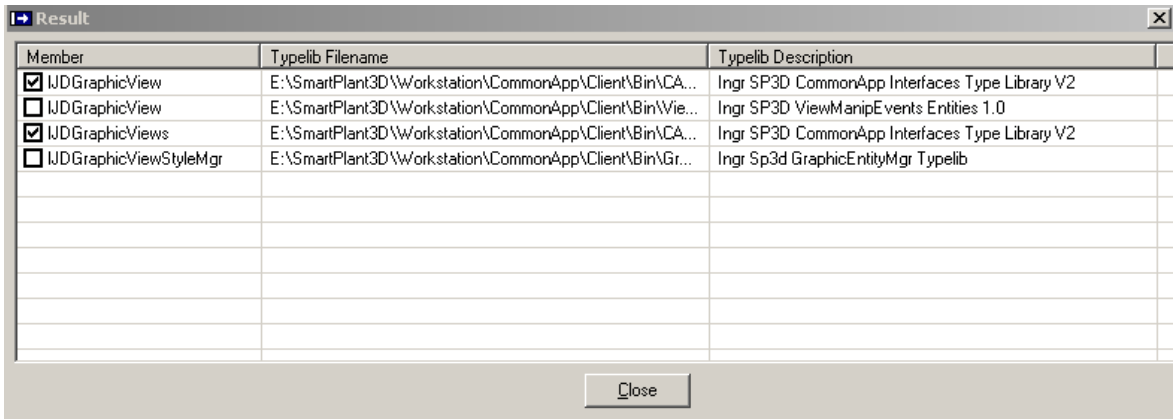
To perform the task of referencing your type libraries more quickly and efficiently:

- Click **Project > SP3D References**.

Using the SP3D References Tool
The SP3D References tool is a very useful utility that you can use to locate and reference type libraries quickly and easily. You only need to know the name of your class object or variable in which to perform a search.

1. Open Visual Basic.
2. Click **Add-Ins > Add-In Manager**....
3. Select **SP3D References** and make sure that the **Loaded/Unloaded** and **Load on Startup** boxes under **Load Behavior** are both checked.
4. Click **OK**.
5. Click **Project > SP3D References** to invoke the dialog.

6. Enter a class or variable name to search..
7. Click **Find**.



8. Check the appropriate type libraries.

Note: If this is the first time that you have invoked the tool, it begins reading your system to generate a data file that contains information about all existing registered type libraries.

Debugging Your Code

No matter how carefully you create your code, errors can occur. To handle these errors, you need to add error-handling code to your procedures.
You perform the process of locating and fixing bugs in applications by *debugging* the code. Visual Basic provides several tools to help analyze how your application operates. These debugging tools are useful in locating the source of bugs, but you can also use the tools to experiment with changes to your application or to learn how other applications work.

Note: You must add the TaskHost project to the integrated development environment (IDE) before you can debug your Visual Basic project.

Before you can use the TaskHost project, you must set new paths in your computer's environment variables. Click Start -> Settings -> Control Panel -> System. Select the Advanced tab and then click Environment Variables. Finally add the following path statements according to the location in which you installed the software:

PATH=[*Product Directory*]\Core\Runtime; [*Product Directory*]\GeometryTopology\Runtime

Adding the TaskHost Project to your Project

1. Open your Visual Basic .vbp project to debug.
2. Click File > Add Project.
3. Select the Existing tab.
4. Open SP3DTaskHost.vbp in the following path: ..\Debug\Container\Src\Host
5. In the Project window, right-click over SP3DTaskHost and then select Set as Start Up.
6. Right-click again on SP3DTaskHost and then select SP3DtaskHost Properties...
7. On the Project Properties dialog, change the Project Type to Standard EXE.
8. Set the breakpoint in your project to debug.
9. Click Run and wait for processing to begin. Your Visual Basic project becomes active when the breakpoint is reached.
10. Click to view <your project>, which returns you back to the code view. Then step through your code.

Important

Do not stop the debug process by clicking the End command. If you end processing this way, you will throw an exception, crash all the software that is running, and lose your changes.
To safely end processing, click File > Exit from the SmartPlant 3D TaskHost software.

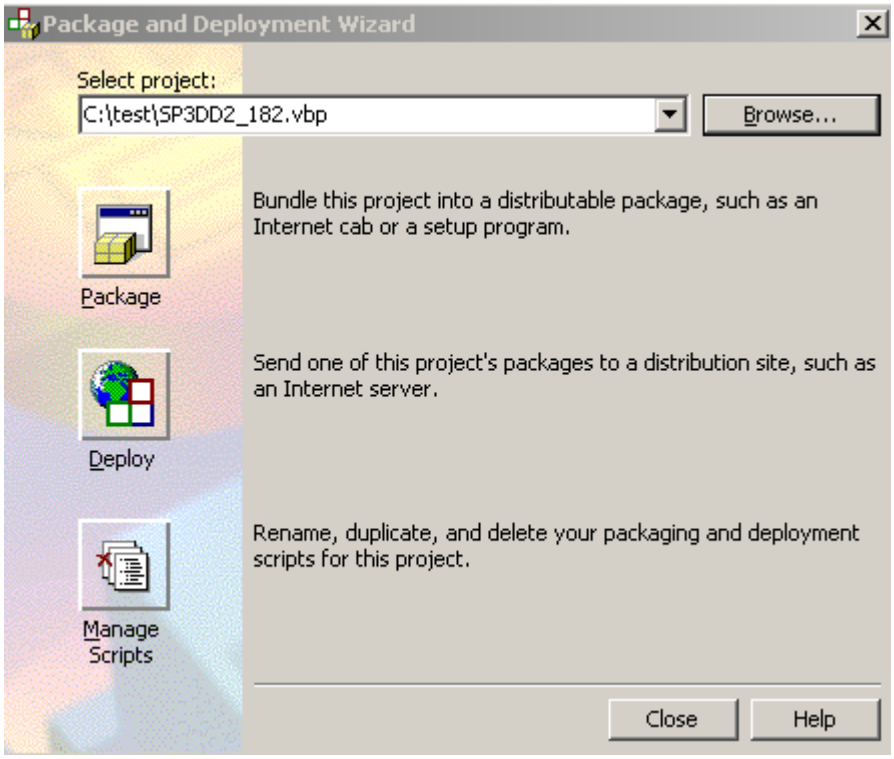
Creation of Cab Files

Introduction:

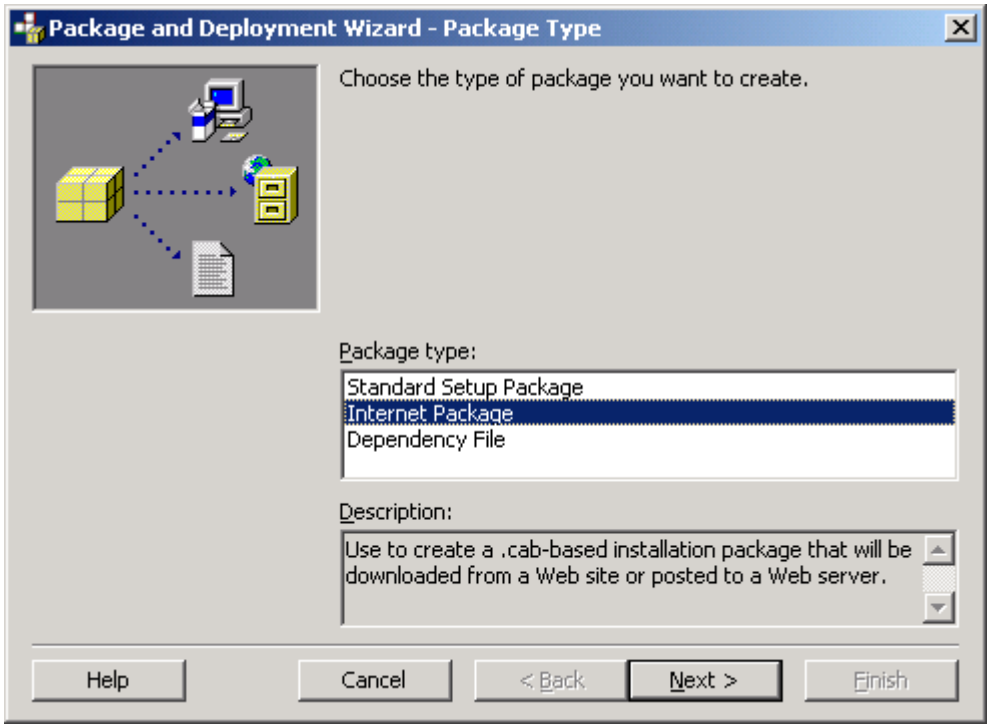
This document describes the step-by-step procedure for creating cab files

Procedure:

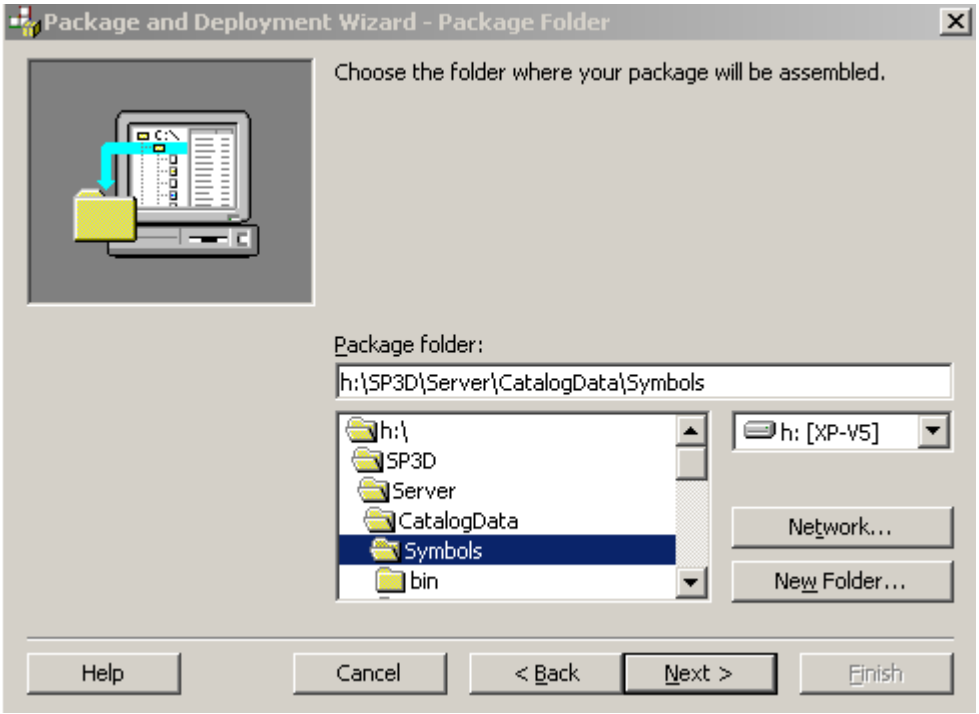
1. Start the “Package & Deployment Wizard” Under Programs ->Microsoft Visual Basic 6.0 -> Microsoft Visual Basic 6.0 Tools.
   Go to the “Select Project:” Click on the Browse button and navigate to the Symbol Project folder. Select the .vbp file of the symbol project
   Click on the Package Icon Button



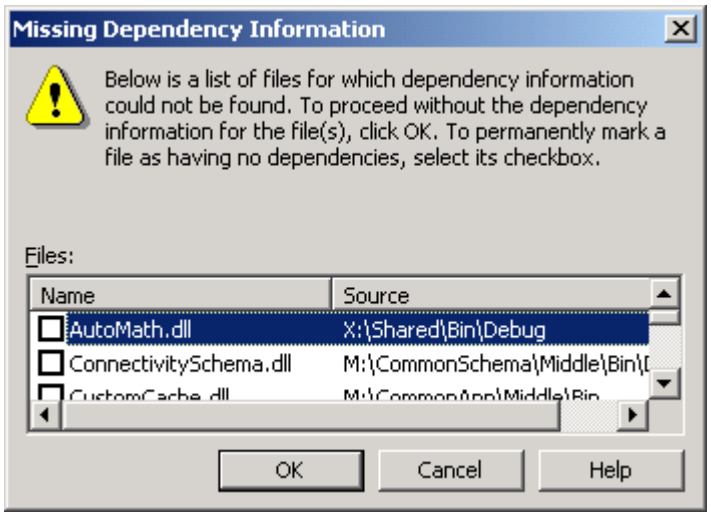
2. Next, select the “Package Type” as **Internet Package**. Click Next.



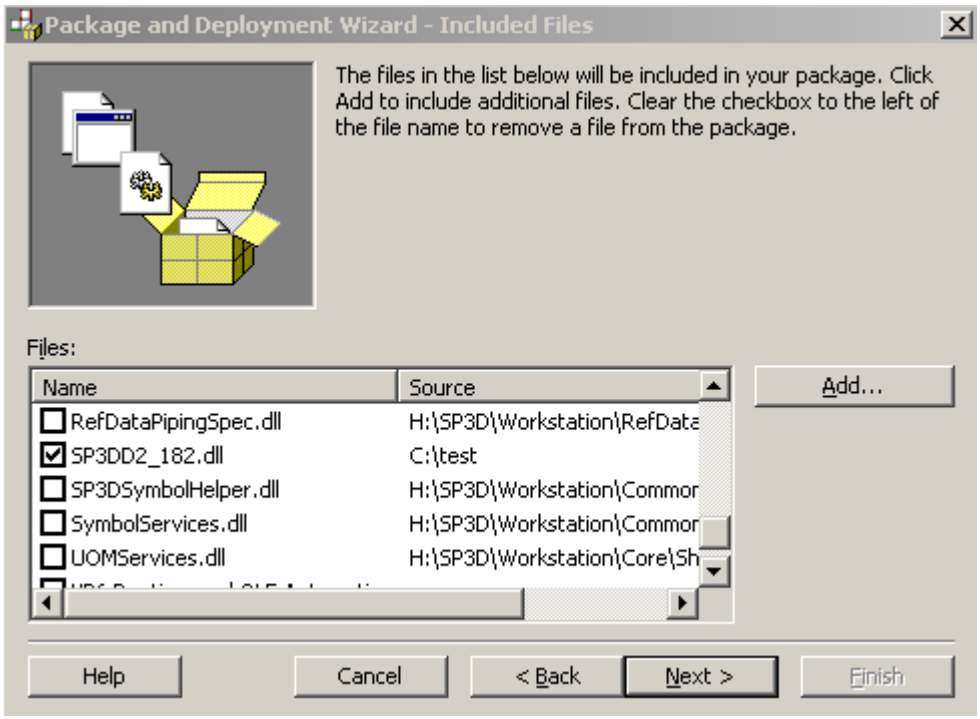
3. Select the Package Folder. Select the symbol share folder. (The Cab file must be created in the symbol share). Click **Yes** if it asks if we want to create the folder. Click Next.



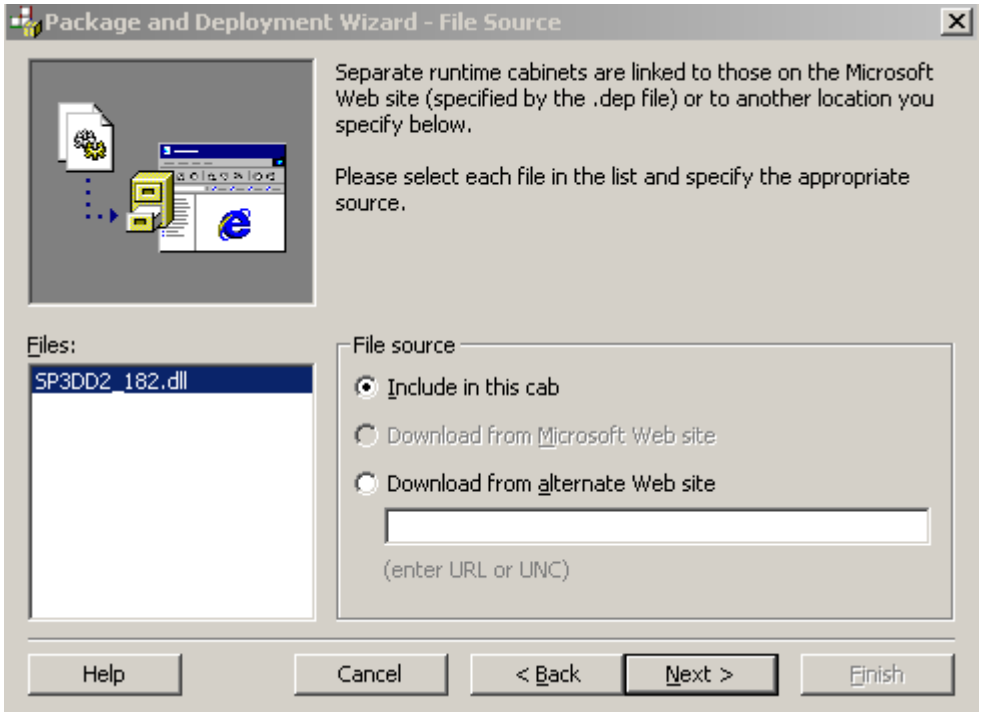
4. In the Missing Dependency Information dialog, do not check any of the dependency files. Click OK.



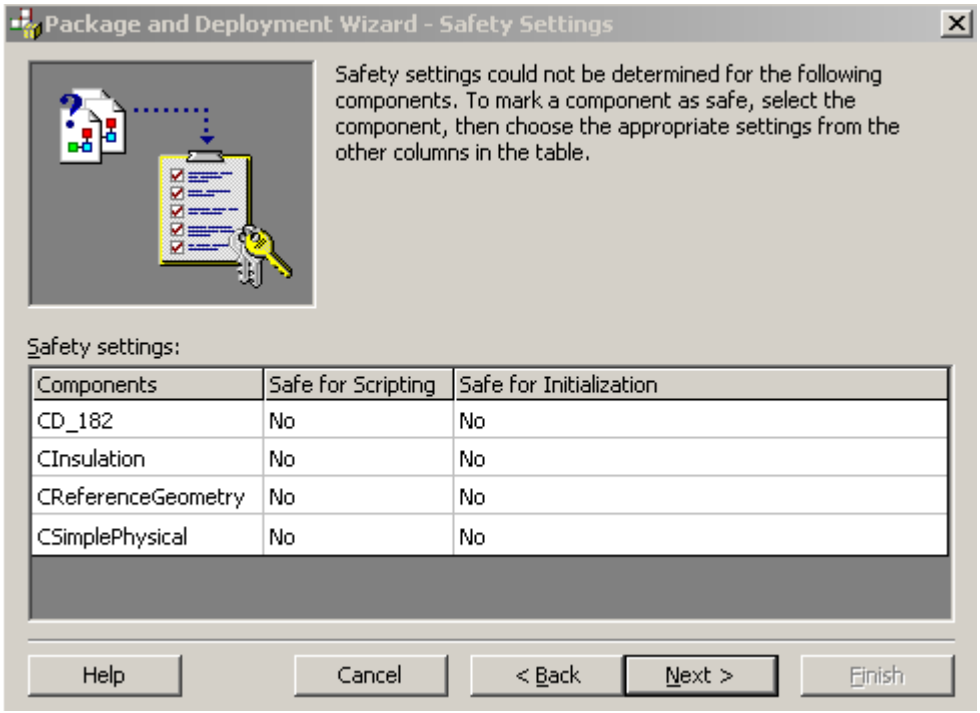
5. Next, uncheck all the files except the symbol dll file.



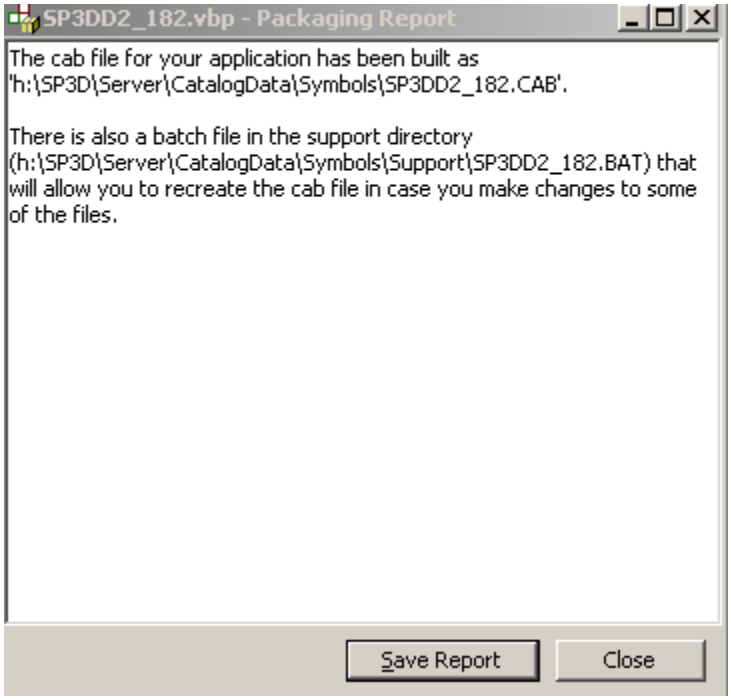
6. Next, let the File Source option be “Include in this cab”.



7. Retain the Safety Settings indicated. Click Next.



8. Click Finish. The cab file for the symbol gets built and a summary Report is displayed.



Hit close button.