SmartPlant 3D Programming I

Student Workbook

Process, Power & Marine

INTERGRAPH

Version 6.1 **May 2006** **N/A**

Copyright

Restricted Rights Legend

Intergraph Corporation
Huntsville, Alabama 35894-0001

Warranties and Liabilities

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

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

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

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

Trademarks

Table of Contents

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.

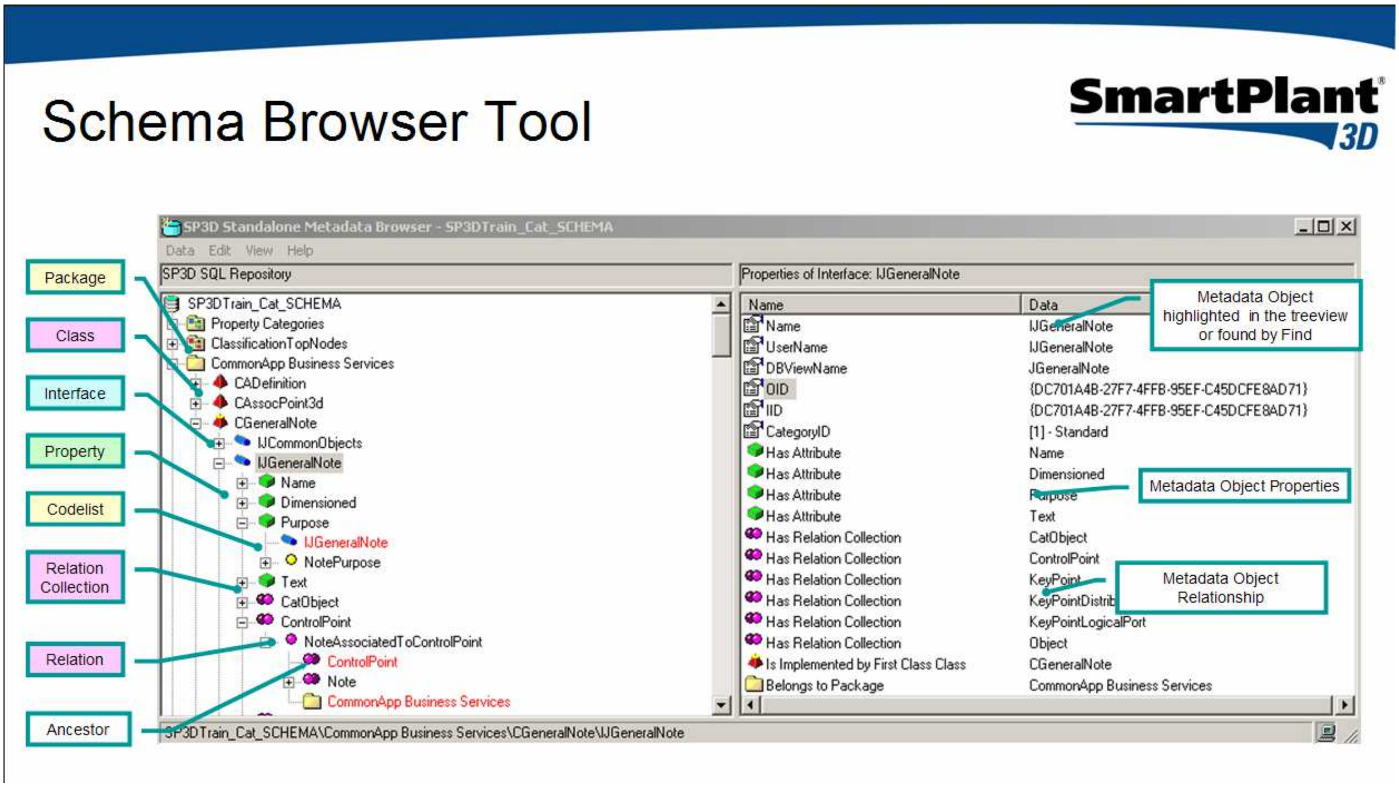
Course description

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

Course Reference Material

SmartPlant 3D Programmer’s Guide
SmartPlant 3D Symbols Reference Data Guide
SmartPlant 3D Reference Data Guide

Understanding Smart Plant 3D Data Model



Schema Browser Tool

SmartPlant® 3D

SP3D Standalone Metadata Browser - SP3DTrain_Cat_SCHEMA

SP3D SQL Repository

```
SP3DTrain_Cat_SCHEMA
  Property Categories
  ClassificationTopNodes
  CommonApp Business Services
    CADefinition
    CAssocPoint3d
    CGeneralNote
      IJCommonObjects
      IJGeneralNote
        Name
        Dimensioned
        Purpose
          IJGeneralNote
          NotePurpose
        Text
        CatObject
        ControlPoint
          NoteAssociatedToControlPoint
            ControlPoint
            Note
            CommonApp Business Services
```

SP3DTrain_Cat_SCHEMA\CommonApp Business Services\CGeneralNote\IJGeneralNote

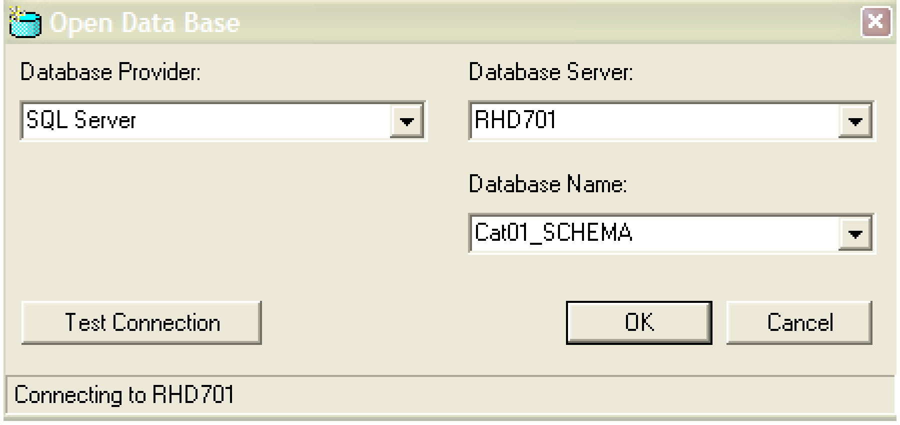
Properties of Interface: IJGeneralNote

Name	Data
Name	IJGeneralNote
UserName	IJGeneralNote
DBViewName	JGeneralNote
OID	{DC701A4B-27F7-4FFB-95EF-C45DCFE8AD71}
IID	{DC701A4B-27F7-4FFB-95EF-C45DCFE8AD71}
CategoryID	[1] - Standard
Has Attribute	Name
Has Attribute	Dimensioned
Has Attribute	Purpose
Has Attribute	Text
Has Relation Collection	CatObject
Has Relation Collection	ControlPoint
Has Relation Collection	KeyPoint
Has Relation Collection	KeyPointDistrib
Has Relation Collection	KeyPointLogicalPort
Has Relation Collection	Object
Is Implemented by First Class Class	CGeneralNote
Belongs to Package	CommonApp Business Services

Package
Class
Interface
Property
Codelist
Relation Collection
Relation
Ancestor
Metadata Object highlighted in the treeview or found by Find
Metadata Object Properties
Metadata Object Relationship

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.



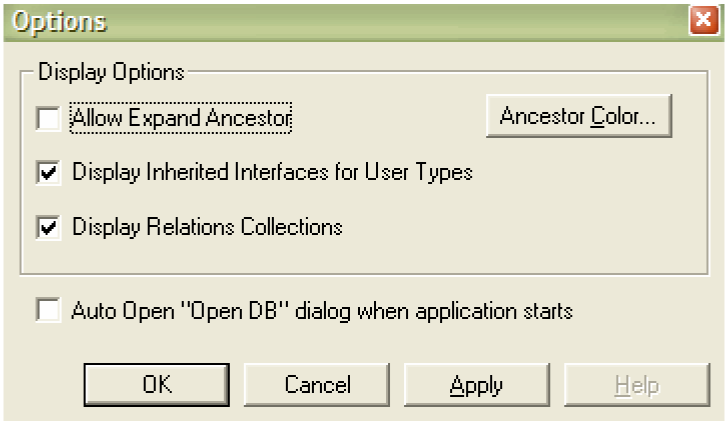
Database Provider: SQL Server

Database Server: RHD701

Database Name: Cat01_SCHEMA

Connecting to RHD701

2. Set the view menu -> Options to open the option dialog box. Enable the check box to displays Relations Collections.



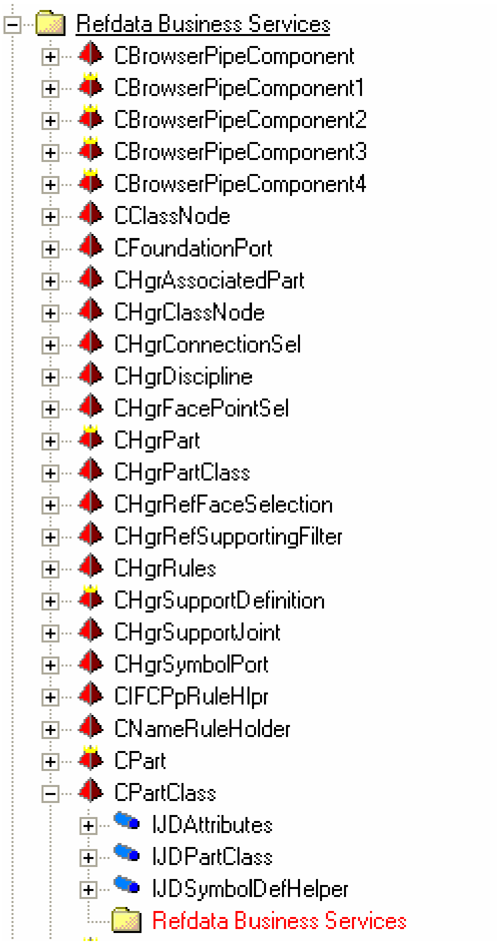
□ Allow Expand Ancestor

☑ Display Inherited Interfaces for User Types

☑ Display Relations Collections

□ Auto Open "Open DB" dialog when application starts

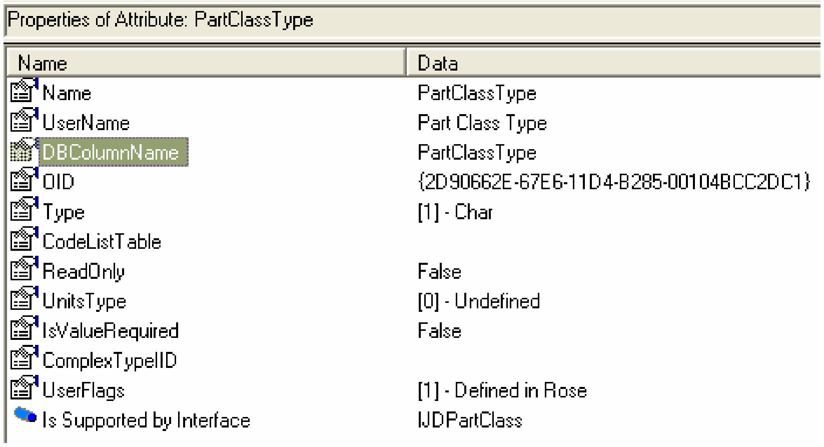
3. Exit the SP3D Schema Browser and re-open it to read the change. We are interested in query part classes, thus we must start our navigation at Ref Data Business Services.



- Refdata Business Services
  - CBrowserPipeComponent
  - CBrowserPipeComponent1
  - CBrowserPipeComponent2
  - CBrowserPipeComponent3
  - CBrowserPipeComponent4
  - CClassNode
  - CFoundationPort
  - CHgrAssociatedPart
  - CHgrClassNode
  - CHgrConnectionSel
  - CHgrDiscipline
  - CHgrFacePointSel
  - CHgrPart
  - CHgrPartClass
  - CHgrRefFaceSelection
  - CHgrRefSupportingFilter
  - CHgrRules
  - CHgrSupportDefinition
  - CHgrSupportJoint
  - CHgrSymbolPort
  - CIFCPpRuleHlpr
  - CNameRuleHolder
  - CPart
  - CPartClass
    - IJDAttributes
    - IJDPartClass
    - IJDSymbolDefHelper
    - Refdata Business Services

4. Expand CPartClass node. The tool shows a list of interfaces that are implemented by CPartClass. The interfaces describe both the attributes a class can have as well as the relationships. 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 the DBViewName corresponding to that entry in the detail view. The DBViewName corresponding to IJDPartClass is PartClassType.



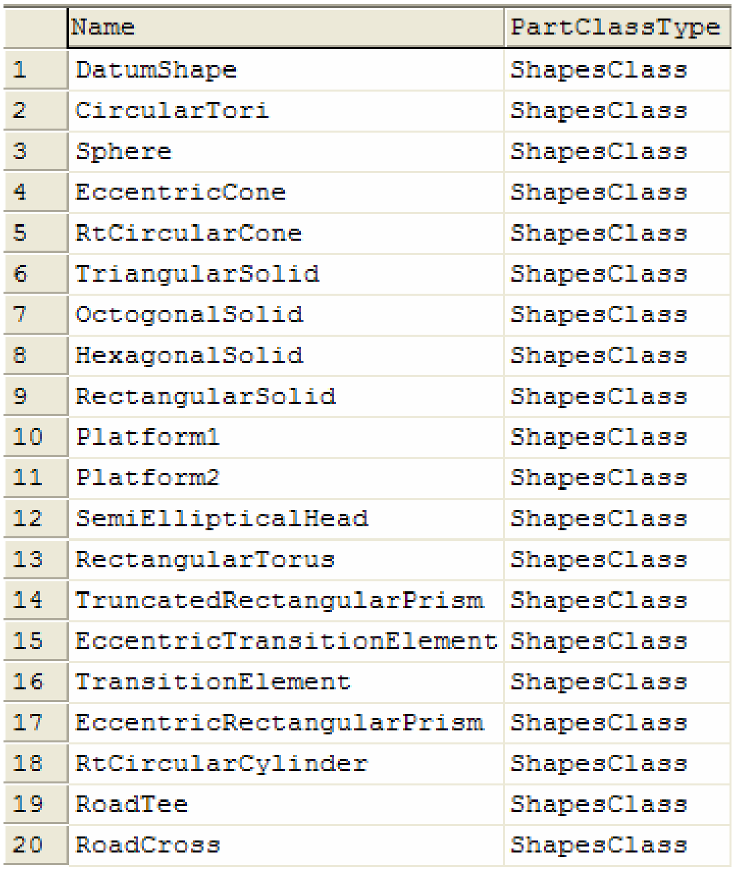
Properties of Attribute: PartClassType

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. Thus 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.

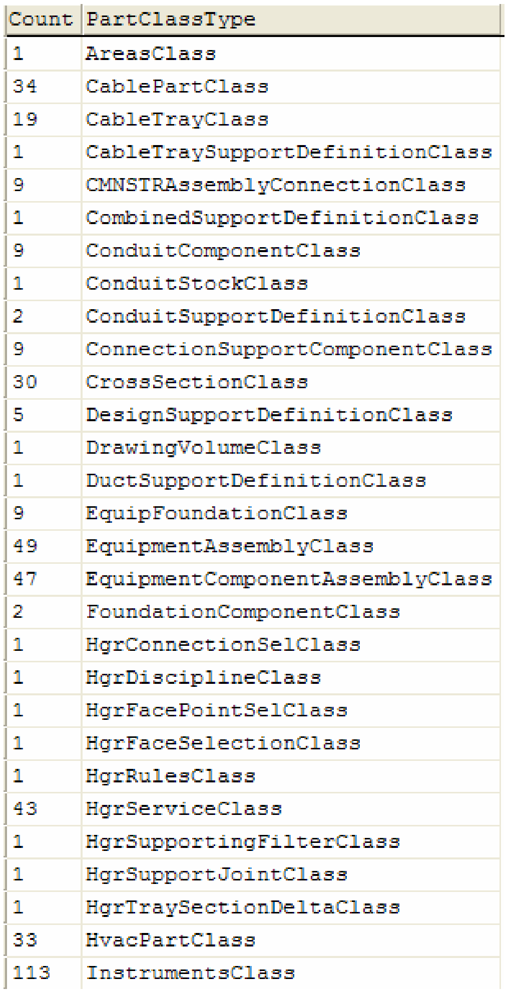


	Name	PartClassType
1	DatumShape	ShapesClass
2	CircularTori	ShapesClass
3	Sphere	ShapesClass
4	EccentricCone	ShapesClass
5	RtCircularCone	ShapesClass
6	TriangularSolid	ShapesClass
7	OctogonalSolid	ShapesClass
8	HexagonalSolid	ShapesClass
9	RectangularSolid	ShapesClass
10	Platform1	ShapesClass
11	Platform2	ShapesClass
12	SemiEllipticalHead	ShapesClass
13	RectangularTorus	ShapesClass
14	TruncatedRectangularPrism	ShapesClass
15	EccentricTransitionElement	ShapesClass
16	TransitionElement	ShapesClass
17	EccentricRectangularPrism	ShapesClass
18	RtCircularCylinder	ShapesClass
19	RoadTee	ShapesClass
20	RoadCross	ShapesClass

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

1. Use the Group 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:

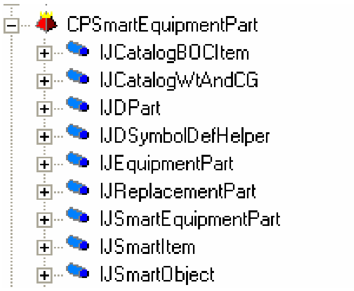
```
Select count(*) as Count,
PartClassType
from JDPartClass
Group by PartClassType
```



Count	PartClassType
1	AreasClass
34	CablePartClass
19	CableTrayClass
1	CableTraySupportDefinitionClass
9	CMNSTRAssemblyConnectionClass
1	CombinedSupportDefinitionClass
9	ConduitComponentClass
1	ConduitStockClass
2	ConduitSupportDefinitionClass
9	ConnectionSupportComponentClass
30	CrossSectionClass
5	DesignSupportDefinitionClass
1	DrawingVolumeClass
1	DuctSupportDefinitionClass
9	EquipFoundationClass
49	EquipmentAssemblyClass
47	EquipmentComponentAssemblyClass
2	FoundationComponentClass
1	HgrConnectionSelClass
1	HgrDisciplineClass
1	HgrFacePointSelClass
1	HgrFaceSelectionClass
1	HgrRulesClass
43	HgrServiceClass
1	HgrSupportingFilterClass
1	HgrSupportJointClass
1	HgrTraySectionDeltaClass
33	HvacPartClass
113	InstrumentsClass

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

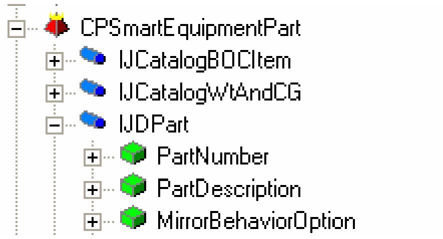
1. We are interested in query Smart Equipment parts, thus we must start our navigation at Ref Data Business Services.

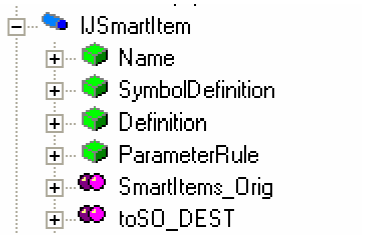


2. Expand Smart Equipment node. The tool shows a list of interfaces that are implemented by Smart Equipment. The interfaces describe both the attributes of a smart equipment can have as well as the relationships. Thus to search for all smart equipment parts in the catalog database, we must execute a 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
```

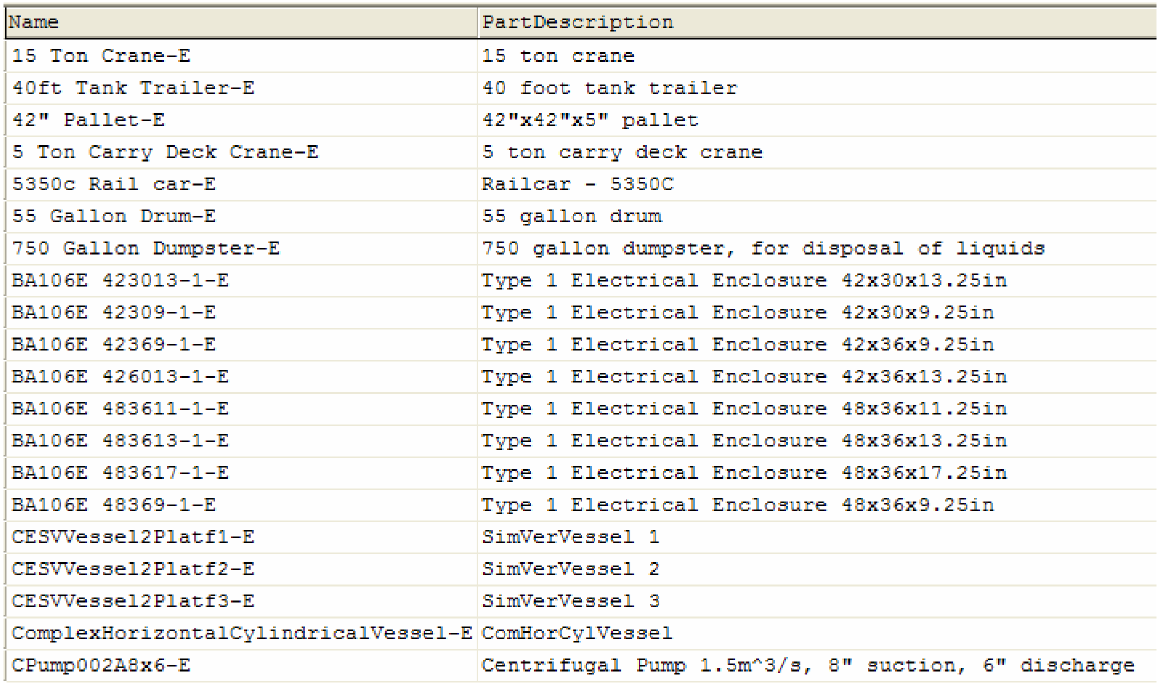
3. We are also interested to get the description and the name of the smart equipment.
4. This is done by performing the join operation on the views that return the equipment name and the equipment description. Use the Order by clause to sort the equipment parts by their name.





5. 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
```

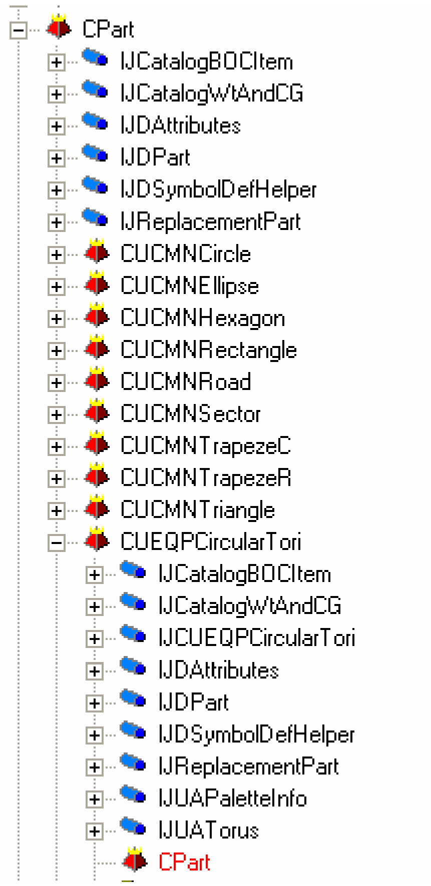


Name	PartDescription
15 Ton Crane-E	15 ton crane
40ft Tank Trailer-E	40 foot tank trailer
42" Pallet-E	42"x42"x5" pallet
5 Ton Carry Deck Crane-E	5 ton carry deck crane
5350c Rail car-E	Railcar - 5350C
55 Gallon Drum-E	55 gallon drum
750 Gallon Dumpster-E	750 gallon dumpster, for disposal of liquids
BA106E 423013-1-E	Type 1 Electrical Enclosure 42x30x13.25in
BA106E 42309-1-E	Type 1 Electrical Enclosure 42x30x9.25in
BA106E 42369-1-E	Type 1 Electrical Enclosure 42x36x9.25in
BA106E 426013-1-E	Type 1 Electrical Enclosure 42x36x13.25in
BA106E 483611-1-E	Type 1 Electrical Enclosure 48x36x11.25in
BA106E 483613-1-E	Type 1 Electrical Enclosure 48x36x13.25in
BA106E 483617-1-E	Type 1 Electrical Enclosure 48x36x17.25in
BA106E 48369-1-E	Type 1 Electrical Enclosure 48x36x9.25in
CESVVessel2Platf1-E	SimVerVessel 1
CESVVessel2Platf2-E	SimVerVessel 2
CESVVessel2Platf3-E	SimVerVessel 3
ComplexHorizontalCylindricalVessel-E	ComHorCylVessel
CPump002A8x6-E	Centrifugal Pump 1.5m^3/s, 8" suction, 6" discharge

Lab 4: List all the equipment shapes located in the palette

1. We are interested in query Equipment Shapes, 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.

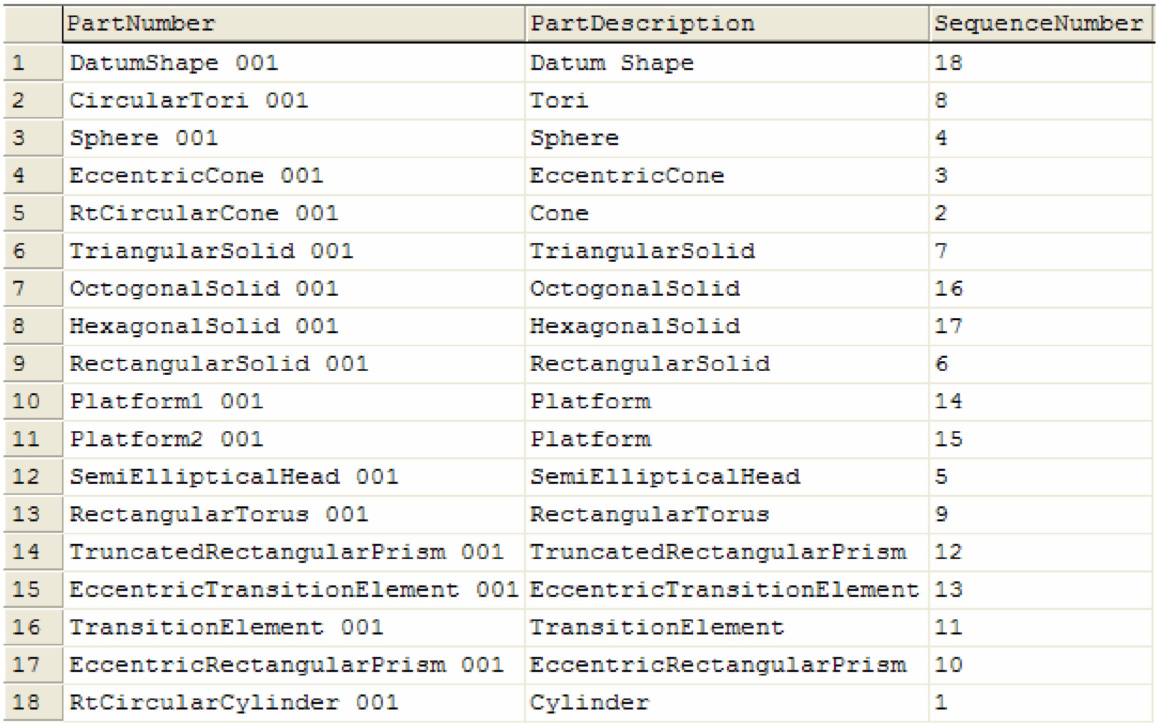
2. Expand CPart node. The tool shows a list of Equipment shape part classes. Expand one of them and notice that if a part class is located in the palette, then it must implement the IJUAPaletteInfo



- CPart
  - IJCatalogBOCItem
  - IJCatalogWtAndCG
  - IJDAttributes
  - IJDPart
  - IJDSymbolDefHelper
  - IJReplacementPart
  - CUCMNCircle
  - CUCMNEllipse
  - CUCMNHexagon
  - CUCMNRectangle
  - CUCMNRoad
  - CUCMNSector
  - CUCMNTrapezeC
  - CUCMNTrapezeR
  - CUCMNTriangle
  - CUEQPCircularTori
    - IJCatalogBOCItem
    - IJCatalogWtAndCG
    - IJCUEQPCircularTori
    - IJDAttributes
    - IJDPart
    - IJDSymbolDefHelper
    - IJReplacementPart
    - IJUAPaletteInfo
    - IJUATorus
    - CPart

Therefore, the SELECT query is as follows:

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

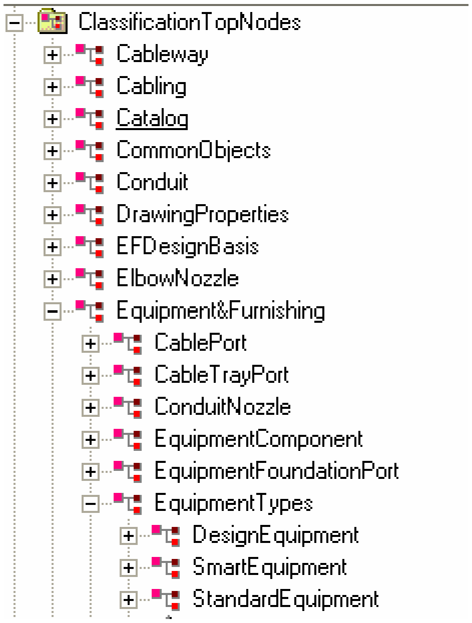


	PartNumber	PartDescription	SequenceNumber
1	DatumShape 001	Datum Shape	18
2	CircularTori 001	Tori	8
3	Sphere 001	Sphere	4
4	EccentricCone 001	EccentricCone	3
5	RtCircularCone 001	Cone	2
6	TriangularSolid 001	TriangularSolid	7
7	OctogonalSolid 001	OctogonalSolid	16
8	HexagonalSolid 001	HexagonalSolid	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
14	TruncatedRectangularPrism 001	TruncatedRectangularPrism	12
15	EccentricTransitionElement 001	EccentricTransitionElement	13
16	TransitionElement 001	TransitionElement	11
17	EccentricRectangularPrism 001	EccentricRectangularPrism	10
18	RtCircularCylinder 001	Cylinder	1

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

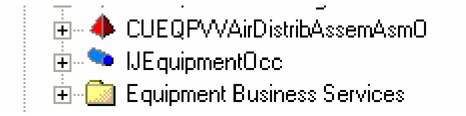
1. Go to the Classification TopNodes.

2. Expand Equipment & Furnisshing -> Equipment Type to find Smart Equipment.



- ClassificationTopNodes
  - Cableway
  - Cabling
  - Catalog
  - CommonObjects
  - Conduit
  - DrawingProperties
  - EFDesignBasis
  - ElbowNozzle
  - Equipment&Furnishing
    - CablePort
    - CableTrayPort
    - ConduitNozzle
    - EquipmentComponent
    - EquipmentFoundationPort
    - EquipmentTypes
      - DesignEquipment
      - SmartEquipment
      - StandardEquipment

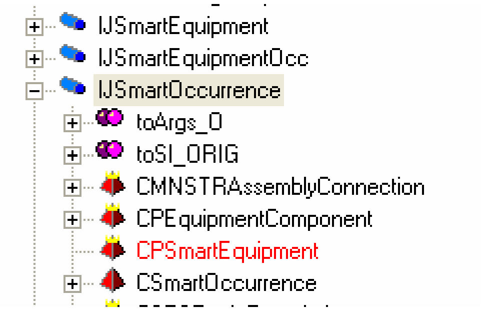
3. Scroll down the right pane, it will show us that Equipment belong to the Equipment Business Services.



- CUEQPVVAirDistribAssemAsm0
- IJEquipmentOcc
- Equipment Business Services

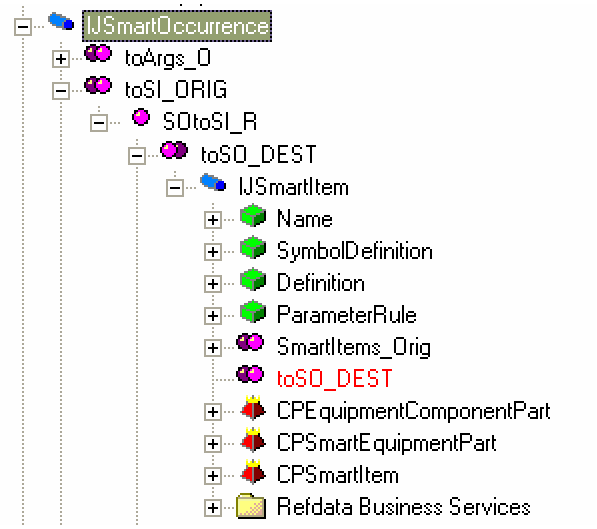
4. Thus, we must begin our hunt under the Equipment Business Service folder.

5. Expand CPSmartEquipment node. The tool shows a list of interfaces that are implemented by Smart Equipment. The interfaces describe both the attributes a smart equipment can have as well as the relationships. Since we are looking for a relation to the catalog, let us expand IJSmartOccurrence (which is the interface implemented by all smart occurrences).



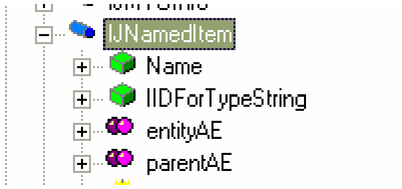
```
IJSmartEquipment
IJSmartEquipmentOcc
IJSmartOccurrence
    toArgs_O
    toSI_ORIG
    CMNSTRAssemblyConnection
    CPEquipmentComponent
    CPSmartEquipment
    CSmartOccurrence
```

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



```
IJSmartOccurrence
    toArgs_O
    toSI_ORIG
        SOtoSI_R
            toSO_DEST
                IJSmartItem
                    Name
                    SymbolDefinition
                    Definition
                    ParameterRule
                    SmartItems_Orig
                    toSO_DEST
                    CPEquipmentComponentPart
                    CPSmartEquipmentPart
                    CPSmartItem
                    Refdata Business Services
```

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

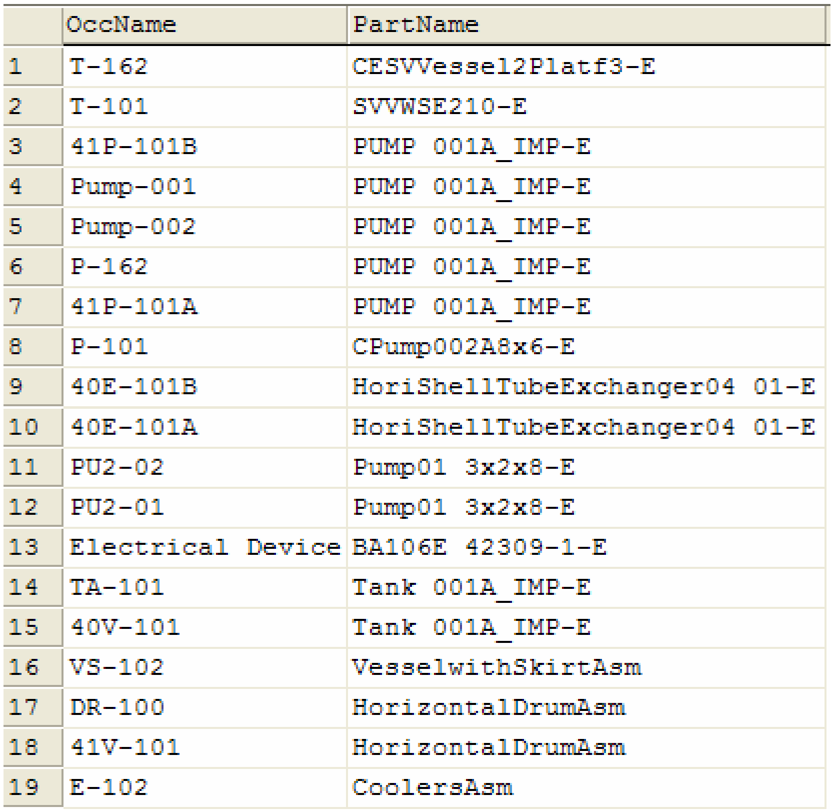


```
IJNamedItem
    Name
    IIDForTypeString
    entityAE
    parentAE
```

8. Therefore, 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
```

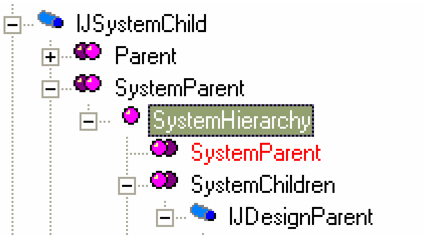


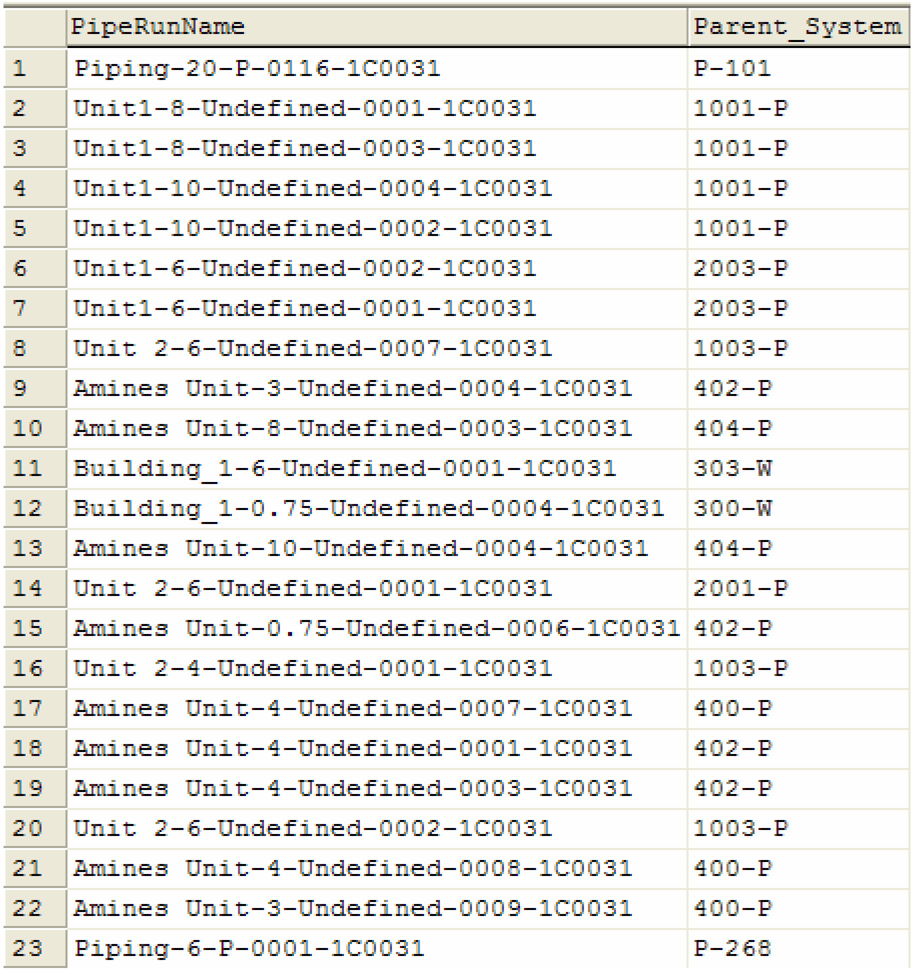
	OccName	PartName
1	T-162	CESVVessel2Platf3-E
2	T-101	SVVWSE210-E
3	41P-101B	PUMP 001A_IMP-E
4	Pump-001	PUMP 001A_IMP-E
5	Pump-002	PUMP 001A_IMP-E
6	P-162	PUMP 001A_IMP-E
7	41P-101A	PUMP 001A_IMP-E
8	P-101	CPump002A8x6-E
9	40E-101B	HoriShellTubeExchanger04 01-E
10	40E-101A	HoriShellTubeExchanger04 01-E
11	PU2-02	Pump01 3x2x8-E
12	PU2-01	Pump01 3x2x8-E
13	Electrical Device	BA106E 42309-1-E
14	TA-101	Tank 001A_IMP-E
15	40V-101	Tank 001A_IMP-E
16	VS-102	VesselwithSkirtAsm
17	DR-100	HorizontalDrumAsm
18	41V-101	HorizontalDrumAsm
19	E-102	CoolersAsm

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 either IJSystemChild and establish a relationship to a design parent.





	PipeRunName	Parent_System
1	Piping-20-P-0116-1C0031	P-101
2	Unit1-8-Undefined-0001-1C0031	1001-P
3	Unit1-8-Undefined-0003-1C0031	1001-P
4	Unit1-10-Undefined-0004-1C0031	1001-P
5	Unit1-10-Undefined-0002-1C0031	1001-P
6	Unit1-6-Undefined-0002-1C0031	2003-P
7	Unit1-6-Undefined-0001-1C0031	2003-P
8	Unit 2-6-Undefined-0007-1C0031	1003-P
9	Amines Unit-3-Undefined-0004-1C0031	402-P
10	Amines Unit-8-Undefined-0003-1C0031	404-P
11	Building_1-6-Undefined-0001-1C0031	303-W
12	Building_1-0.75-Undefined-0004-1C0031	300-W
13	Amines Unit-10-Undefined-0004-1C0031	404-P
14	Unit 2-6-Undefined-0001-1C0031	2001-P
15	Amines Unit-0.75-Undefined-0006-1C0031	402-P
16	Unit 2-4-Undefined-0001-1C0031	1003-P
17	Amines Unit-4-Undefined-0007-1C0031	400-P
18	Amines Unit-4-Undefined-0001-1C0031	402-P
19	Amines Unit-4-Undefined-0003-1C0031	402-P
20	Unit 2-6-Undefined-0002-1C0031	1003-P
21	Amines Unit-4-Undefined-0008-1C0031	400-P
22	Amines Unit-3-Undefined-0009-1C0031	400-P
23	Piping-6-P-0001-1C0031	P-268

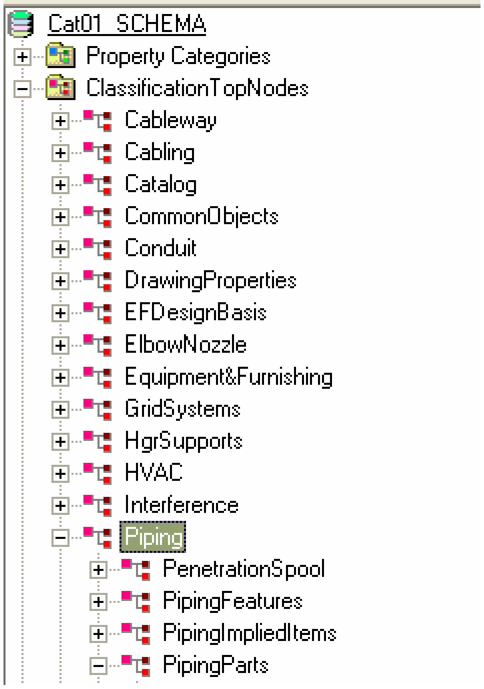
Solution:

```
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
```

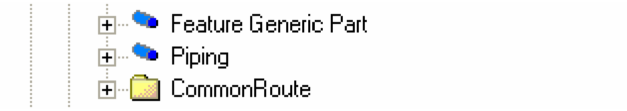
Lab 7: List all object with notes in the model database.

1. Go to the Classification TopNodes.

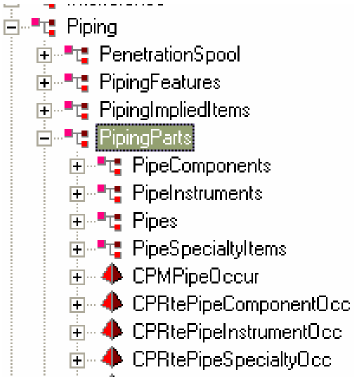
2. Expand Piping node to find Piping Parts and click on Piping Parts.



3. Scroll down the right pane, it will show us that Piping Parts belong to the CommonRoute Business Services.



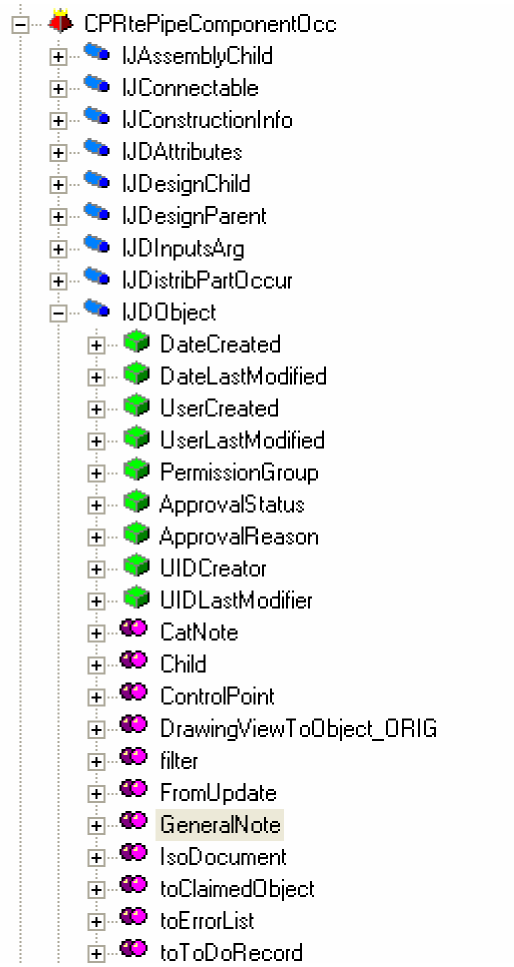
4. Also note in the tree view that specific kinds of piping parts are Pipe Components, Pipes, Pipe Instruments, etc.



5. Thus, we must begin our hunt under the CommonRoute Business Services folder.

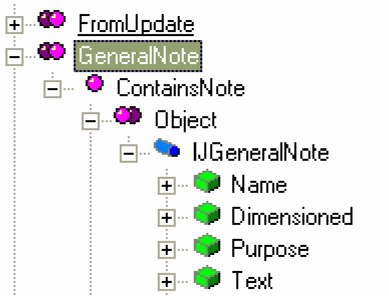
6. Expand Pipe Component occurrence node. The tool shows a list of interfaces that are implemented by Pipe Component occurrence. The interfaces describe both the attributes a pipe component can have as well as the relationships. Since we are looking for object to note relation, let us expand IJDObject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

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

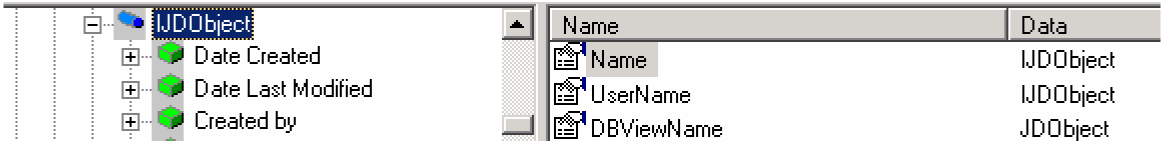


```
CPRtePipeComponentOcc
    IJAssemblyChild
    IJConnectable
    IJConstructionInfo
    IJDAttributes
    IJDesignChild
    IJDesignParent
    IJDInputsArg
    IJDistribPartOccur
    IJDObject
        DateCreated
        DateLastModified
        UserCreated
        UserLastModified
        PermissionGroup
        ApprovalStatus
        ApprovalReason
        UIDCreator
        UIDLastModifier
        CatNote
        Child
        ControlPoint
        DrawingViewToObject_ORIG
        filter
        FromUpdate
        GeneralNote
        IsoDocument
        toClaimedObject
        toErrorList
        toToDoRecord
```

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



9. Clicking on any of the entries in the tree view will show the DBViewName corresponding to that entry in the detail view. Click on IJDObject to see that the DBViewName corresponding to it is JDObject.



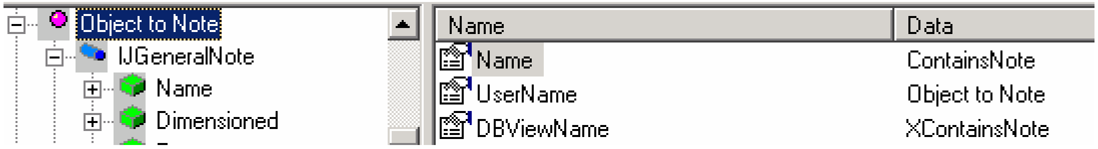
Name	Data
Name	IJDObject
UserName	IJDObject
DBViewName	JDObject

10. 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.

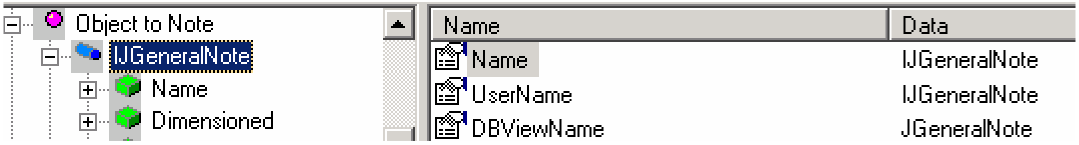
11. However, we are interested in all objects that are in a relationship with a note. Thus let us make a query for all relationships between objects and notes. This is done using the view corresponding to the relationship.



Name	Data
Name	ContainsNote
UserName	Object to Note
DBViewName	XContainsNote

    Select * from XContainsNote

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



Name	Data
Name	IJGeneralNote
UserName	IJGeneralNote
DBViewName	JGeneralNote

    Select * from JGeneralNote

13. 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
    ```

14. By virtue of the joins, we will get a list of all the objects (and only the objects) which has notes associated with them.

15. 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
    ```

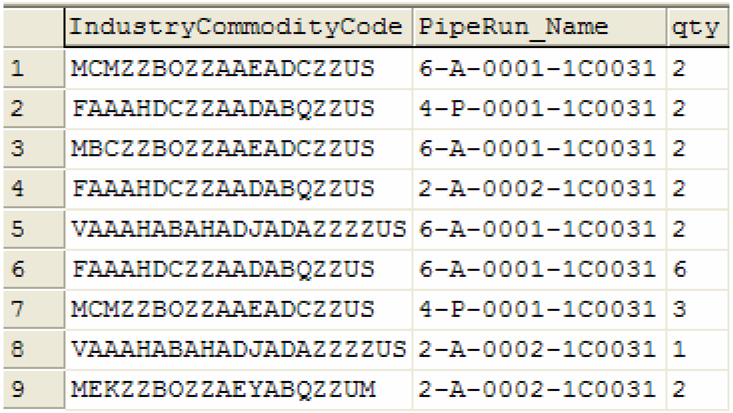
16. However this query gives us too much information, what we are interested in is the Note text. Therefore, the SELECT query is as follows:

    ```
    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 Group clause and the aggregate function count(*) to get the total number of part occurrences in the model database.



	IndustryCommodityCode	PipeRun_Name	qty
1	MCMZZBOZZAAEADCZZUS	6-A-0001-1C0031	2
2	FAAAHDCZZAADABQZZUS	4-P-0001-1C0031	2
3	MBCZZBOZZAAEADCZZUS	6-A-0001-1C0031	2
4	FAAAHDCZZAADABQZZUS	2-A-0002-1C0031	2
5	VAAAHABAHADJADAZZZZUS	6-A-0001-1C0031	2
6	FAAAHDCZZAADABQZZUS	6-A-0001-1C0031	6
7	MCMZZBOZZAAEADCZZUS	4-P-0001-1C0031	3
8	VAAAHABAHADJADAZZZZUS	2-A-0002-1C0031	1
9	MEKZZBOZZAEYABQZZUM	2-A-0002-1C0031	2

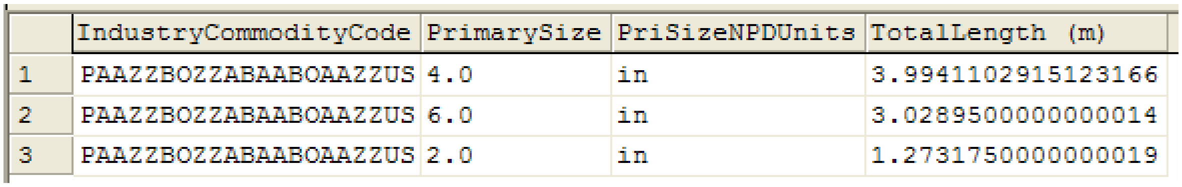
Solution:

```
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
```

Lab 9: List all pipe occurrences with the total length in the model database

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 interface to get the Industry Commodity Code of the part occurrence.
- Use the IJRteStockPartOccur interface to get the length of the pipe occurrence.



	IndustryCommodityCode	PrimarySize	PriSizeNPDUnits	TotalLength (m)
1	PAAZZBOZZABAABOAAZZUS	4.0	in	3.9941102915123166
2	PAAZZBOZZABAABOAAZZUS	6.0	in	3.0289500000000014
3	PAAZZBOZZABAABOAAZZUS	2.0	in	1.2731750000000019

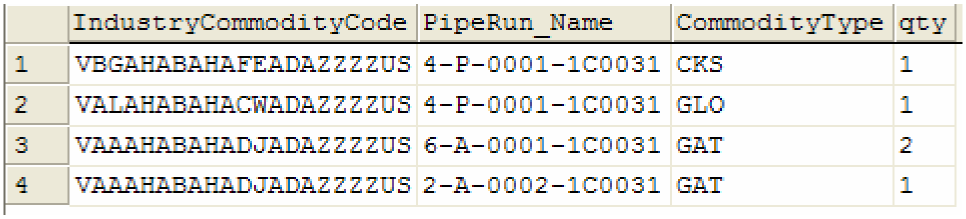
Solution:

```
Select
x3.IndustryCommodityCode,
x3.PrimarySize,
x3.PriSizeNPDUnits,
Sum (x1.length) as 'TotalLength (m)'
from JRteStockPartOccur x1
Join XmadeFrom x2 on x2.oidorigin = x1.oid
Join JDPipeComponent x3 on x3.oid = x2.oiddestination
Group by x3.IndustryCommodityCode, x3.PrimarySize, x3.PriSizeNPDUnits
```

Lab 10: 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	VBGAHABAHAFEADAZZZZUS	4-P-0001-1C0031	CKS	1
2	VALAHABAHACWADAZZZZUS	4-P-0001-1C0031	GLO	1
3	VAAAHABAHADJADAZZZZUS	6-A-0001-1C0031	GAT	2
4	VAAAHABAHADJADAZZZZUS	2-A-0002-1C0031	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
```

Lab 11: Creating a Naming Rule service for Pipeline Systems

Objectives

After completing this lab, you will be able to:

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

This Service will contain an implementation of a naming rule for the pipeline system objects. This component will implement a naming rule for pipeline objects as follows:

Pipeline Name = Fluid Code + Sequence Number

1. Create the following directories:

   c:\train\CustomNameRule

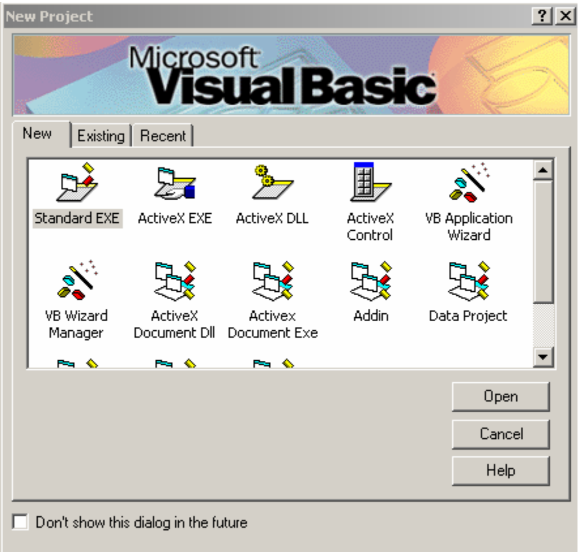
2. Copy the Naming Rule Visual Basic Template Project provided by the instructor to

   c:\train\CustomNameRule\Template

3. Create a directory called lab1 as follows:

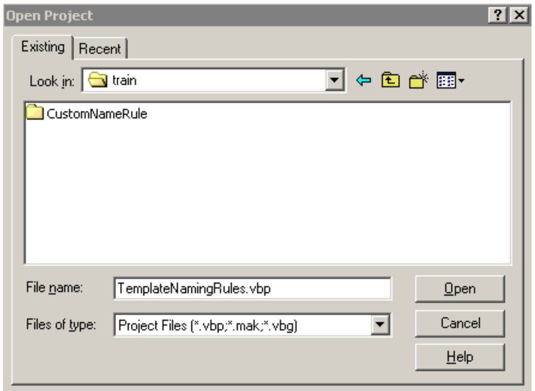
   c:\train\CustomNameRule\lab1

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



☐ Don't show this dialog in the future

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



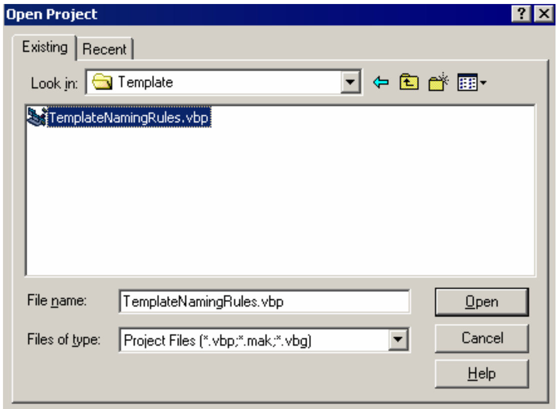
Look in: train

CustomNameRule

File name: TemplateNamingRules.vbp

Files of type: Project Files (*.vbp;*.mak;*.vbg)

7. Navigate the tree and open the Naming Rule Template project



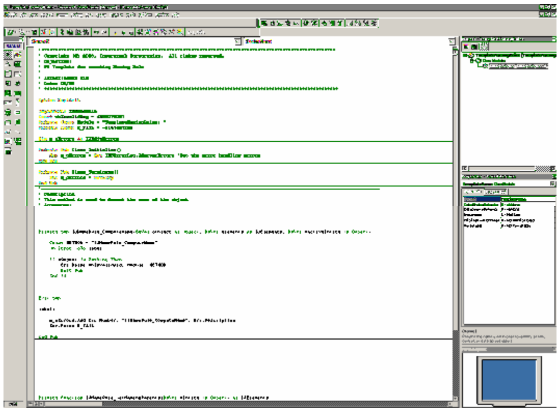
Look in: Template

TemplateNamingRules.vbp

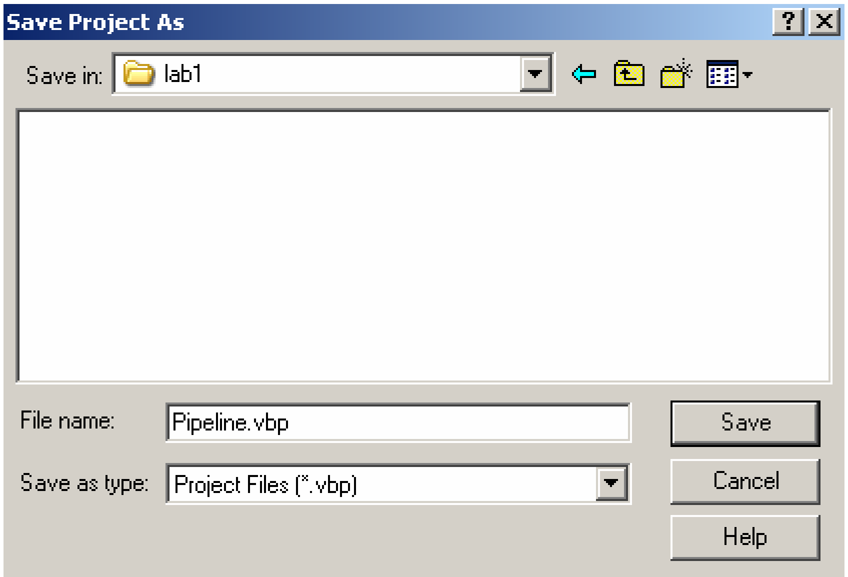
File name: TemplateNamingRules.vbp

Files of type: Project Files (*.vbp;*.mak;*.vbg)

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



9. Go to the Explorer Window and select the Project file in the tree. Select *File -> Save Project As* option to save the project as Pipeline.vbp under the lab1 directory



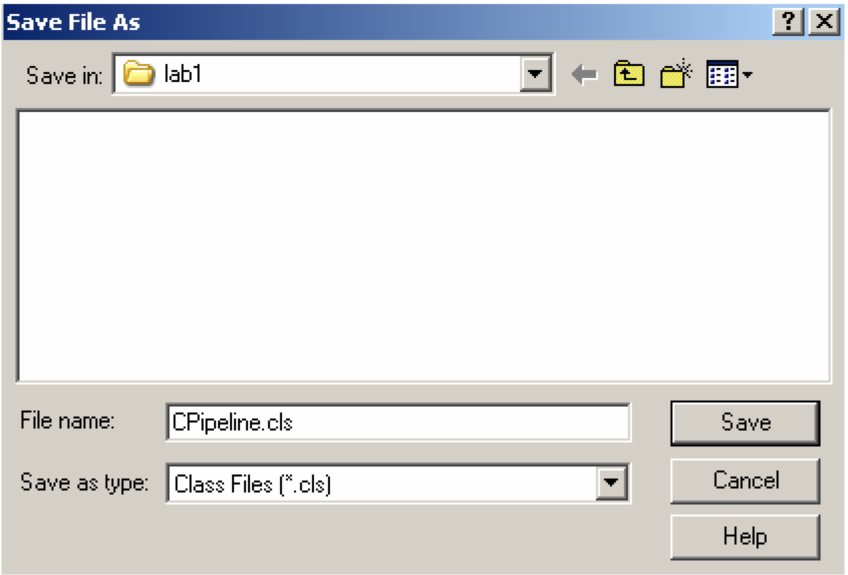
Save Project As

Save in: lab1

File name: Pipeline.vbp

Save as type: Project Files (*.vbp)

10. Go to the Explorer Window and select the TemplateName class file in the tree. Select *File -> Save TemplateName.cls As* option to save the class module as CPipeline.cls under lab1 directory

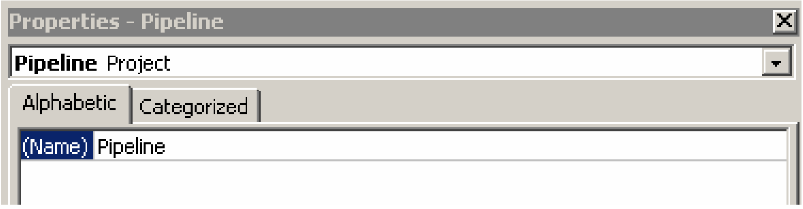


Save in: lab1

File name: CPipeline.cls

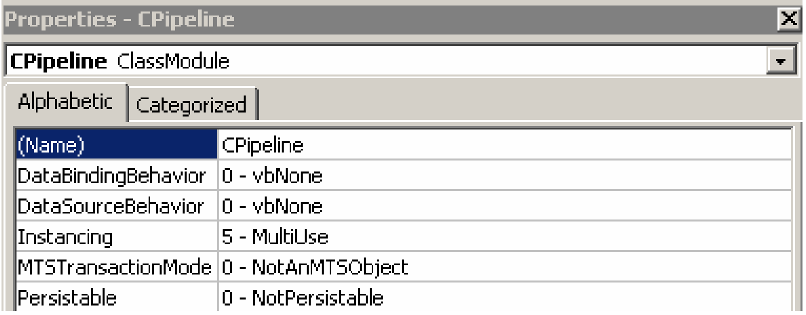
Save as type: Class Files (*.cls)

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



Pipeline Project

(Name)	Pipeline



CPipeline ClassModule

(Name)	CPipeline
DataBindingBehavior	0 - vbNone
DataSourceBehavior	0 - vbNone
Instancing	5 - MultiUse
MTSTransactionMode	0 - NotAnMTSObject
Persistable	0 - NotPersistable

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

14. Go to the General Declarations section and declare an object variable to hold the reference to the IJDCodeListMetaData.

```
Private m_oCodeListMetadata As IJDCodeListMetaData
```

15. Add code to the body of the subroutine ComputeName.

    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 property 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 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
    ```

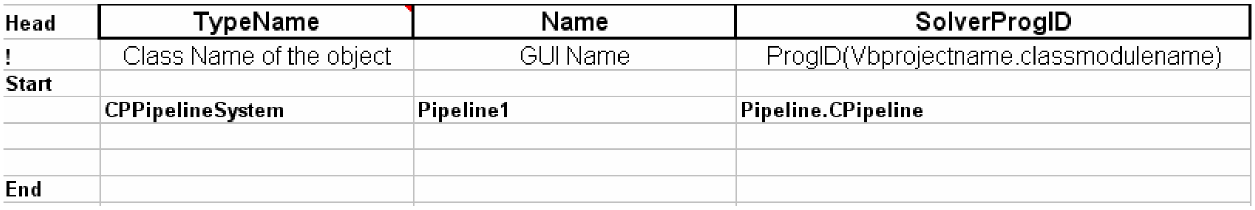
23. Insert into your existing project the following Private Function. Open the GetCatalog.txt file located in the template directory file and use Cut/Paste operation to insert the codes. The inserted codes should look like this:

```
'-------------------------------------------------------------------------------------
'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 and add code to remove the reference from object variable m_oCodeListMetadata.

```
    Set m_oCodeListMetadata = Nothing
```

25. Compile and Save the project.
26. Open the TemplateNamingRules.xls
27. 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
End			

28. Save the excel sheet as TrainingNameRules.xls and exit Excel.

29. Run Bulkload Utility using the A/M/D mode and add the new naming rule into the training catalog.
30. Go to SP3D System & Specifciation Task and place a pipeline system to test your naming rule.

Lab 12: Creating a Naming Rule service for PipeRun objects

Objective

After completing this lab, you will be able to:

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

This Service will contain an implementation of a naming rule for the piperun objects. This component will implement a naming rule for piperun objects as follows:

PipeRun object:

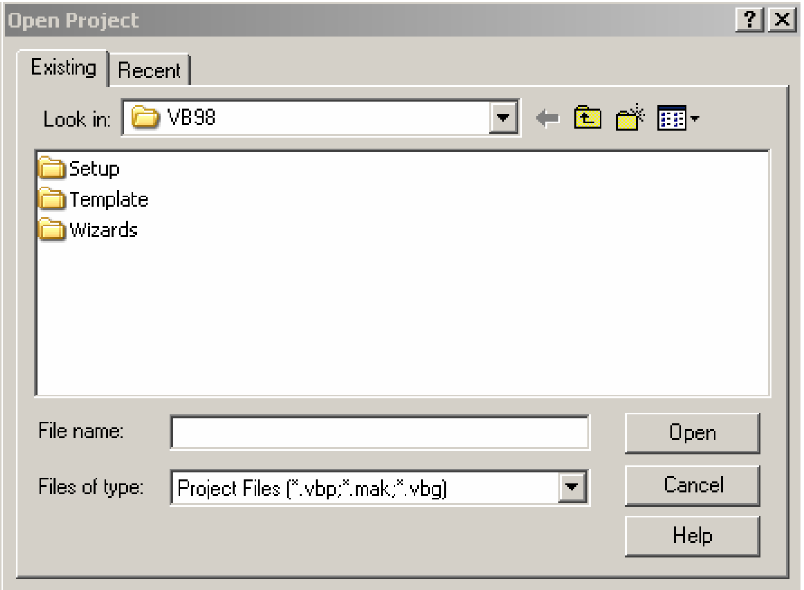
Pipe Runs:
NPD + NPD Units + Spec Name + Parent System

1. Create a directory called lab1 as follows:

   c:\train\CustomNameRule\lab2

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

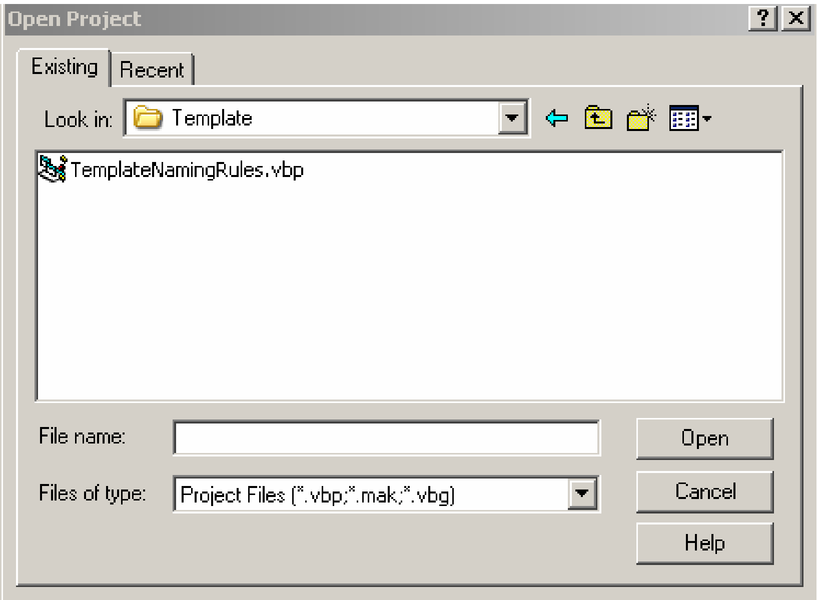


Look in: VB98

File name:

Files of type: Project Files (*.vbp;*.mak;*.vbg)

5. Navigate the tree and open the Naming Rule Template project provided by the instructor.

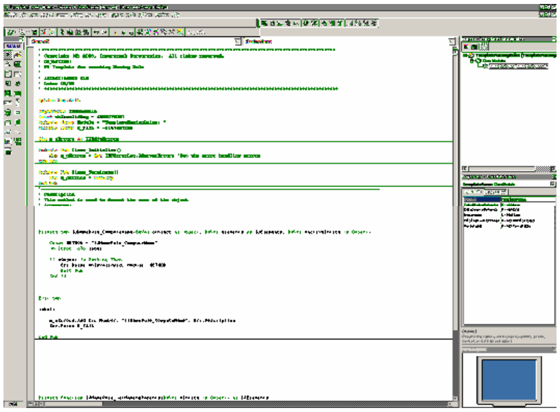


Look in: Template

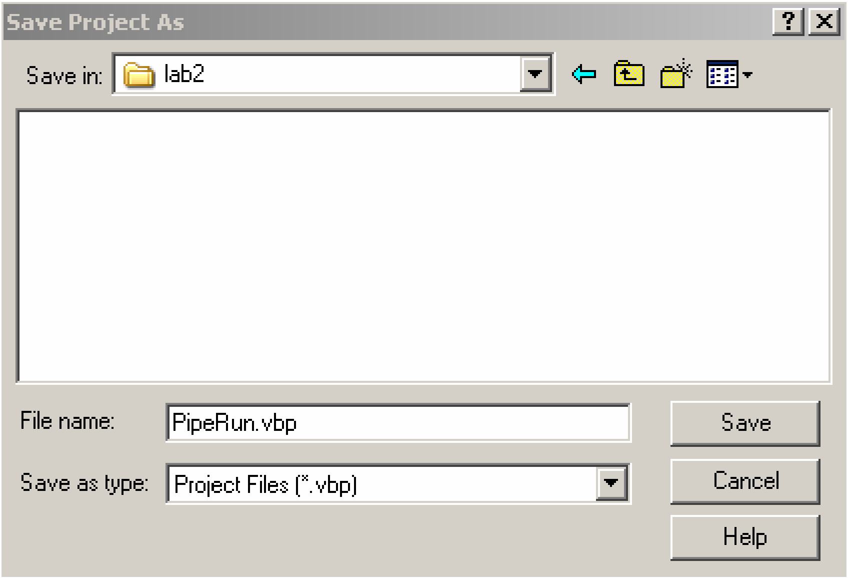
File name:

Files of type: Project Files (*.vbp;*.mak;*.vbg)

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



7. Go to the Explorer Window and select the Project file in the tree. Select *File -> Save Project As* option to save the project as PipeRun.vbp under the lab2 directory



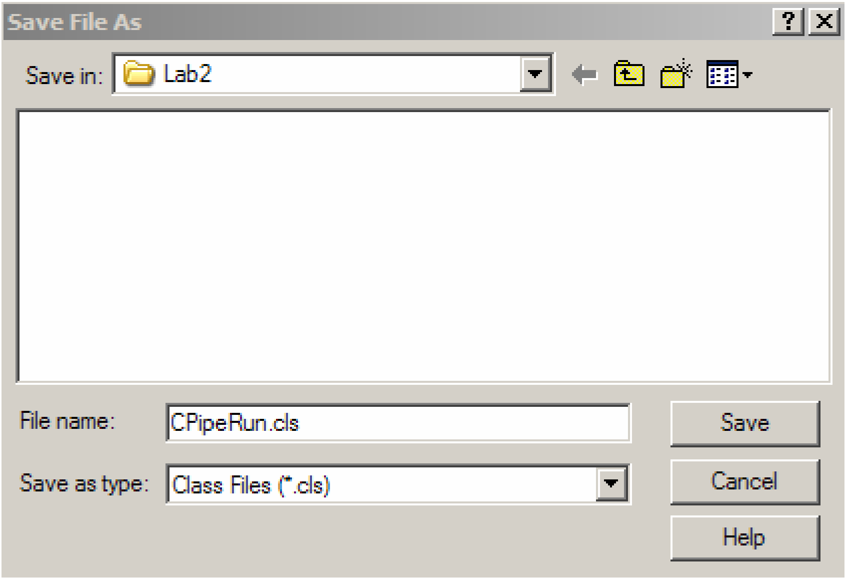
Save Project As

Save in: lab2

File name: PipeRun.vbp

Save as type: Project Files (*.vbp)

8. Go to the Explorer Window and select the TemplateName class file in the tree. Select *File -> Save TemplateName.cls As* option to save the class module as CPipeRun.cls under lab2 directory

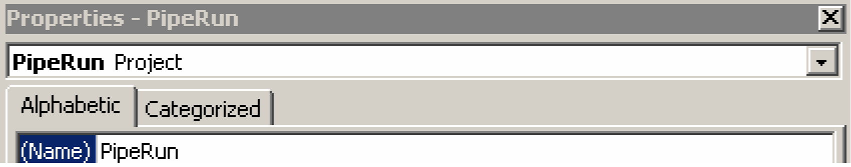


Save in: Lab2

File name: CPipeRun.cls

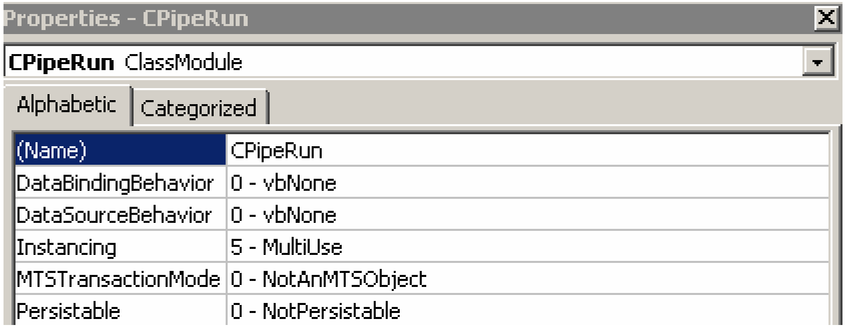
Save as type: Class Files (*.cls)

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



PipeRun Project

(Name) PipeRun



CPipeRun ClassModule

(Name)	CPipeRun
DataBindingBehavior	0 - vbNone
DataSourceBehavior	0 - vbNone
Instancing	5 - MultiUse
MTSTransactionMode	0 - NotAnMTSObject
Persistable	0 - NotPersistable

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 to the body of the subroutine GetNamingParents. The code should get all the Naming Parents that need to participate in the object naming and add them to the 'IJElements collection.

Hints:
Comment the following line:

```
Set IJNameRule_GetNamingParents = Nothing
```

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
```

Get the parent system using the method Add as shown below:

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

Add code to remove the reference from object variables:

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

The resulting code 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 to the body of the subroutine ComputeName. The code 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 code to get the Parent Name.

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

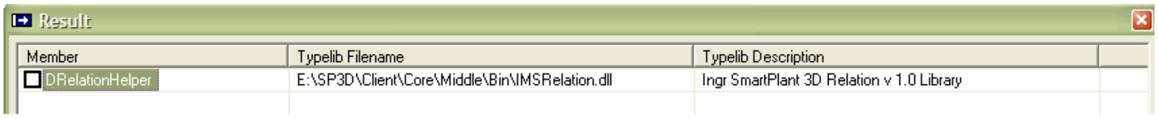
20. Build the name of the piperun.

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

21. Add code to remove the reference from object variables

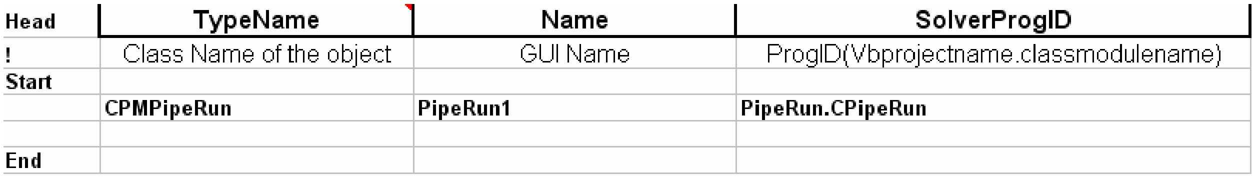
```
Set oChildNamedItem = Nothing
Set oAttributes = Nothing
Set oRelationHelper = Nothing
Set oCollection = Nothing
Set oSpec = Nothing
Set oParentNamedItem = Nothing
```

Note: Compile and Save the project. Note: You might need to reference additional libraries using the SP3D Reference Tool. For example,



Member	Typelib Filename	Typelib Description
☐ DRelationHelper	E:\SP3D\Client\Core\Middle\Bin\IMSRelation.dll	Ingr SmartPlant 3D Relation v 1.0 Library

22. Open the TrainingNameRules.xls saved in previous lab.
23. 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			
	CPMPipeRun	**PipeRun1**	**PipeRun.CPipeRun**
End			

25. Save the excel sheet and exit Excel.
26. Run Bulkload Utility using the A/M/D mode and add the new naming rule into the training catalog.
27. Go to SP3D Piping Task and create a PipeRun to test your naming rule.

Lab 13: Creating a Naming Rule service for Member Parts

Objective

After completing this lab, you will be able to:

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

This Service will contain an implementation of a naming rule for the Member Part objects. This Component will implement a naming rule for Member Part objects as follows:

Member Part object:

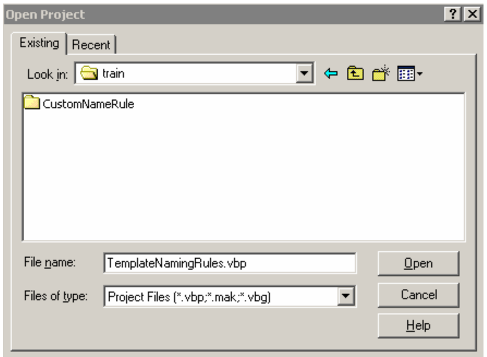
The Short Description of the Member Category Code List + Section Name + Location + IndexCounter

1. Create a directory called lab3 as follows:

   c:\train\CustomNameRule\lab3

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



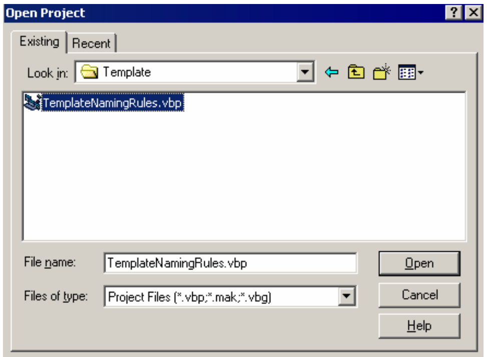
Look in: train

CustomNameRule

File name: TemplateNamingRules.vbp

Files of type: Project Files (*.vbp;*.mak;*.vbg)

5. Navigate the tree and open the Naming Rule Template project



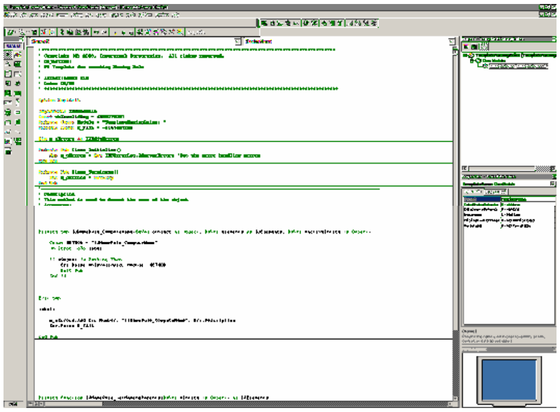
Look in: Template

TemplateNamingRules.vbp

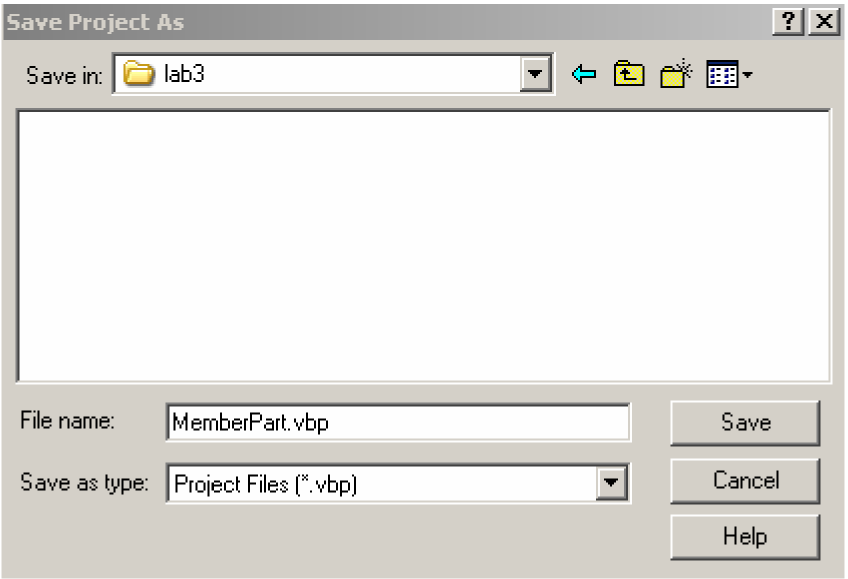
File name: TemplateNamingRules.vbp

Files of type: Project Files (*.vbp;*.mak;*.vbg)

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



7. Go to the Explorer Window and select the Project file in the tree. Select *File -> Save Project As* option to save the project as MemberPart.vbp under the lab3 directory.

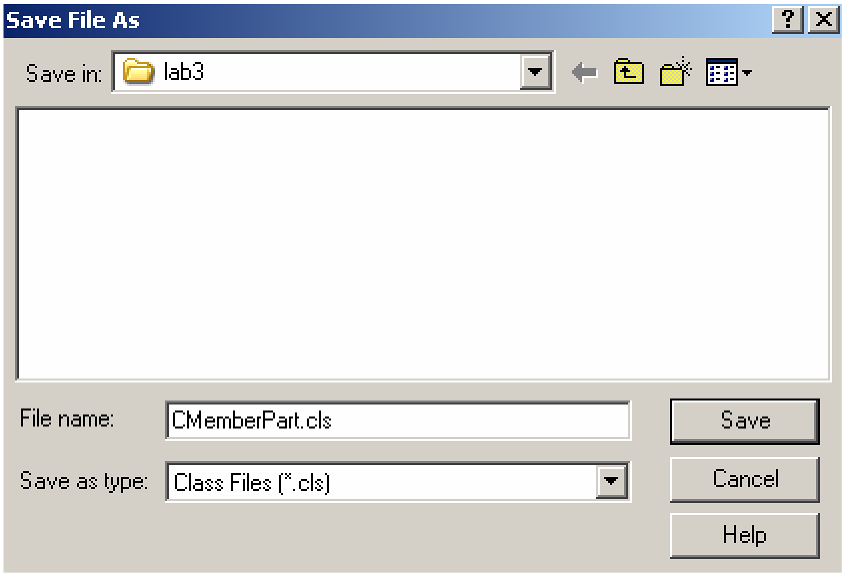


Save in: lab3

File name: MemberPart.vbp

Save as type: Project Files (*.vbp)

8. Go to the Explorer Window and select the TemplateName class file in the tree. Select *File -> Save TemplateName.cls As* option to save the class module as CMemberPart.cls under lab3 directory.

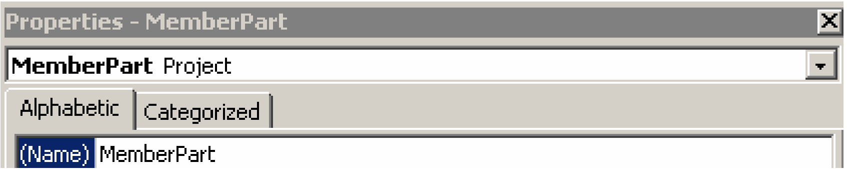


Save in: lab3

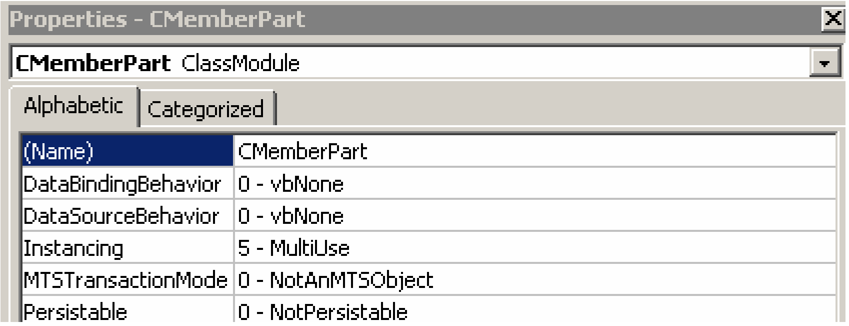
File name: CMemberPart.cls

Save as type: Class Files (*.cls)

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



(Name)	MemberPart



(Name)	CMemberPart
DataBindingBehavior	0 - vbNone
DataSourceBehavior	0 - vbNone
Instancing	5 - MultiUse
MTSTransactionMode	0 - NotAnMTSObject
Persistable	0 - NotPersistable

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 References dialog box. 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 codes. The inserted codes 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

```
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 to the body of the subroutine ComputeName. The code 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 code 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 code to get the member type short description and set the result to upper case. The resulting code 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 code 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 an unique 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 code 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
```

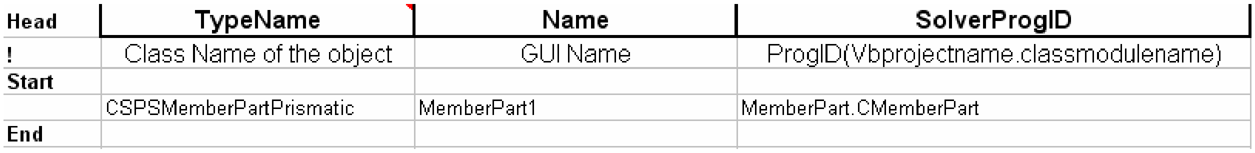
16. Add code to remove the reference from object variables.
Go to the Subroutine Terminate:

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

Go to the subroutine ComputeName:

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

17. Compile and Save the project.
18. Open the TrainingNameRules.xls
19. 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			
	CSPSMemberPartPrismatic	MemberPart1	MemberPart.CMemberPart
End			

20. Save the excel sheet 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.

Lab 14: Piping Component Symbol

Objective

After completing this lab, you will be able to:

- Create Piping Component symbols using the SmartPlant 3D Part Definition Wizard

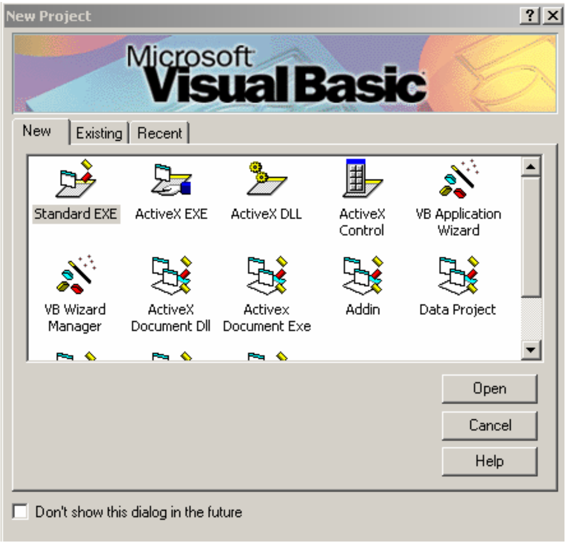
Create a simple weldneck flange symbol.

Skip the following lines (1-2) if the symbol wizard is installed on your machine.

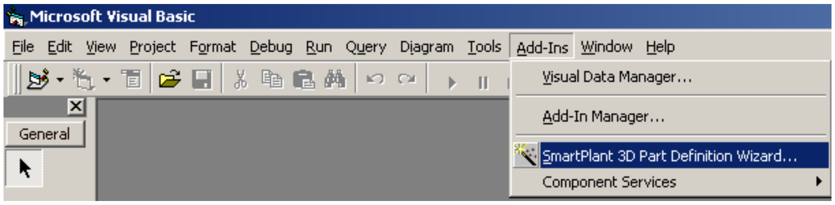
1. Go to [Install Directory]\Programming\Tools\SymbolWizard
2. Install SP3D VB Symbol Wizard in device c:\Program Files\ SP3D Symbol Wizard
3. Create the following directories:

   c:\train\SP3DFWN
   c:\train\IngrModules

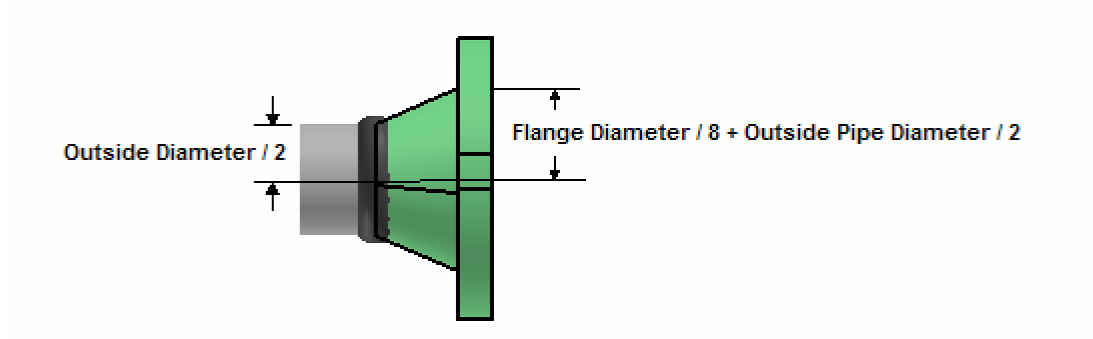
4. Run Microsoft Visual Basic 6.0
5. Close the Microsoft New Project dialog box.



6. Go to the Add-Ins Option and Select SmartPlant 3D Part Definition Wizard.

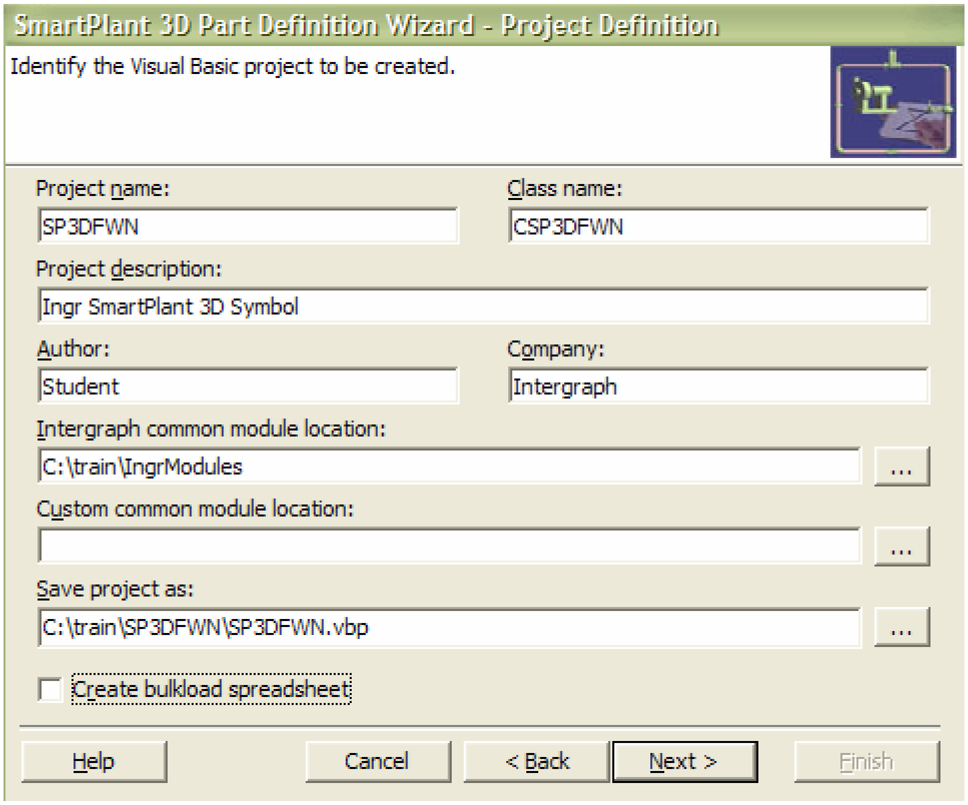

Microsoft Visual Basic
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
General
Visual Data Manager...
Add-In Manager...
SmartPlant 3D Part Definition Wizard...
Component Services

7. The Next step is to create the weldneck flange component symbol definition template using SP3D Part Definition Symbol Wizard.



8. In this page you define the VB project name. Key in the following information:

Project Name: *SP3DFWN*
Author: *Student*
Company: *Intergraph*
Intergraph Module location: *c:\Train\IngrModules*
Save the VB project as: *c:\Train\SP3DFWN*
Disable the create bulkload spreadsheet.



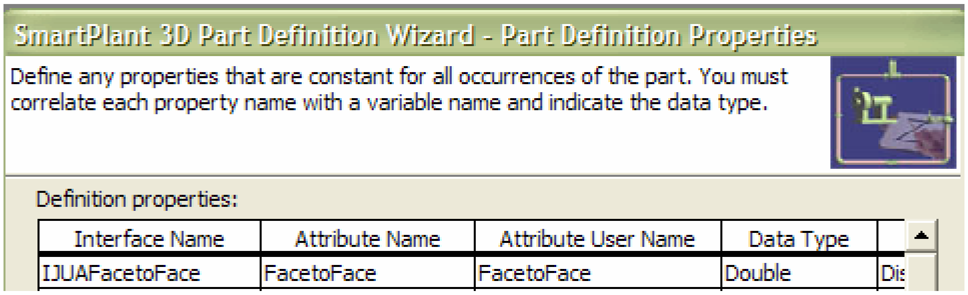
SmartPlant 3D Part Definition Wizard - Project Definition

Identify the Visual Basic project to be created.

Project name:	SP3DFWN
Class name:	CSP3DFWN
Project description:	Ingr SmartPlant 3D Symbol
Author:	Student
Company:	Intergraph
Intergraph common module location:	C:\train\IngrModules
Custom common module location:	
Save project as:	C:\train\SP3DFWN\SP3DFWN.vbp

☐ Create bulkload spreadsheet

9. Select Next button to go the next page. This page is to define any properties that are constant for all occurrences of the piping part. Key in the following data:

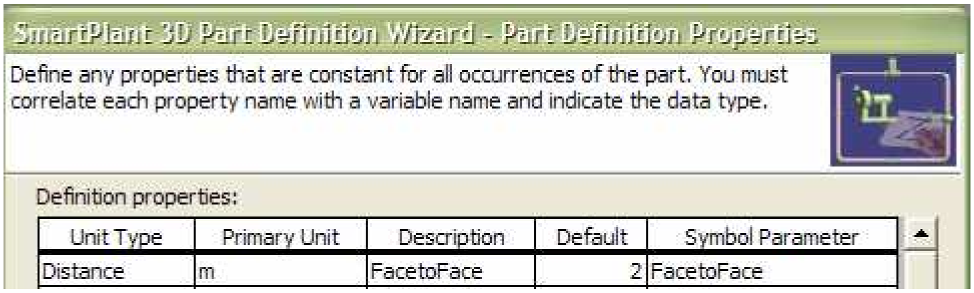


SmartPlant 3D Part Definition Wizard - Part Definition Properties

Define any properties that are constant for all occurrences of the part. You must correlate each property name with a variable name and indicate the data type.

Definition properties:

Interface Name	Attribute Name	Attribute User Name	Data Type	
IJUAFacetoFace	FacetoFace	FacetoFace	Double	Dis



SmartPlant 3D Part Definition Wizard - Part Definition Properties

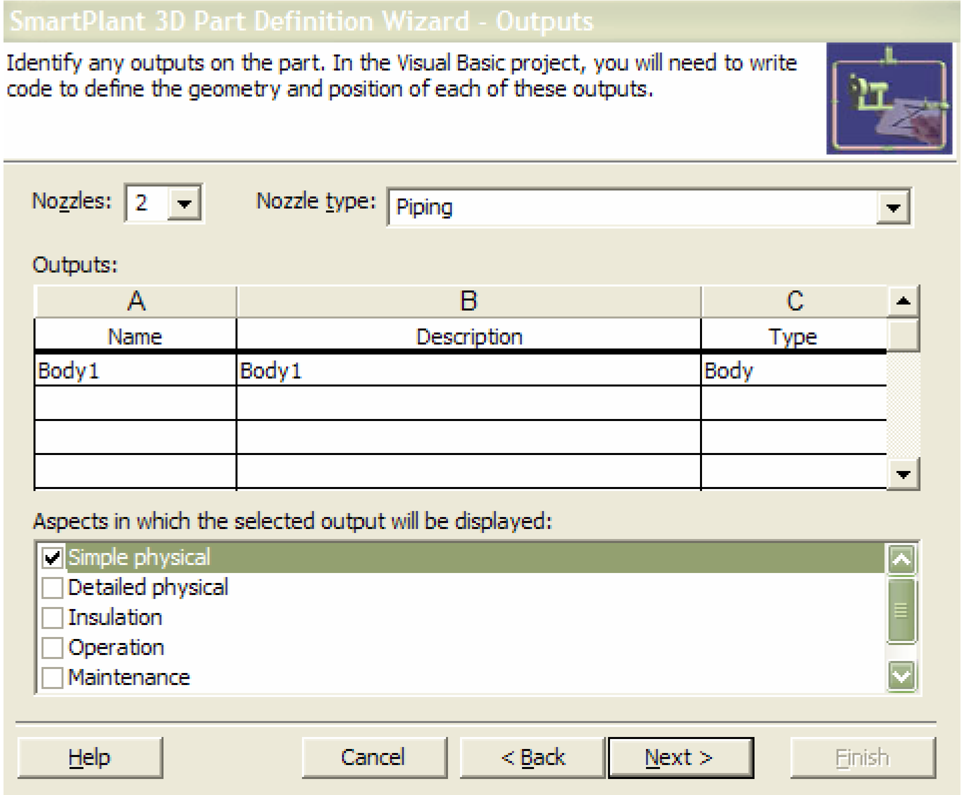
Define any properties that are constant for all occurrences of the part. You must correlate each property name with a variable name and indicate the data type.

Definition properties:

Unit Type	Primary Unit	Description	Default	Symbol Parameter
Distance	m	FacetoFace	2	FacetoFace

10. Select Next button to go the next page. This page defines all occurrence properties of the piping part. Select Next button to go the next page. This page identifies all the outputs of the

piping part. We are going to define three outputs: body and two piping nozzle for our weldneck flange. The Body output is in the Simple Physical aspect.



SmartPlant 3D Part Definition Wizard - Outputs

Identify any outputs on the part. In the Visual Basic project, you will need to write code to define the geometry and position of each of these outputs.

Nozzles: 2

Nozzle type: Piping

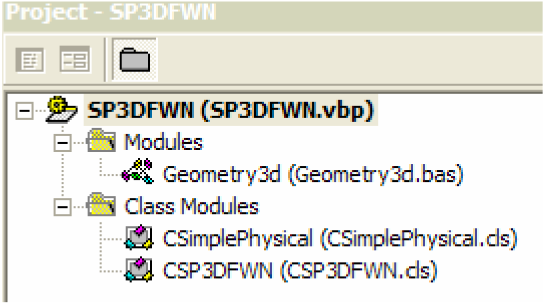
Outputs:

A	B	C
Name	Description	Type
Body1	Body1	Body

Aspects in which the selected output will be displayed:

☑ Simple physical
☐ Detailed physical
☐ Insulation
☐ Operation
☐ Maintenance

11. Press Next button and Finish button to create the SP3DFWN project template. The VB project consists of the following modules:



Project - SP3DFWN

- SP3DFWN (SP3DFWN.vbp)
  - Modules
    - Geometry3d (Geometry3d.bas)
  - Class Modules
    - CSimplePhysical (CSimplePhysical.cls)
    - CSP3DFWN (CSP3DFWN.cls)

12. Open the **CSP3DFWN Class** module. This Class contains several routines.
13. Go to the Class_Initialize() routine. Review the inputs and outputs section.

```
Private Sub Class_Initialize()
    Const METHOD = "Class_Initialize:"
    On Error GoTo Errx

    Set m_oSymbolHelper = New SymbolServices
    m_oSymbolHelper.ProjectName = "SP3DFWN"
    m_oSymbolHelper.ClassName = "CSP3DFWN"

' Inputs
    m_oSymbolHelper.NumInputs = 1
    m_oSymbolHelper.AddInputDef 1, "FacetoFace", "FacetoFace", 2

' Outputs
    m_oSymbolHelper.NumOutputs = 3
    m_oSymbolHelper.AddOutputDef 1, "Body1", "Body1", 1
    m_oSymbolHelper.AddOutputDef 2, "PipingNoz1", "Nozzle 1", 1
    m_oSymbolHelper.AddOutputDef 3, "PipingNoz2", "Nozzle 2", 1

' Aspects
    m_oSymbolHelper.NumAspects = 1
    m_oSymbolHelper.AddAspectDef 1, "SimplePhysical", "SimplePhysical", 1

    Exit Sub
Errx:
    Err.Raise Err.Number, Err.Source & " " & METHOD, Err.Description, _
        Err.HelpFile, Err.HelpContext
End Sub
```

14. Go to **CSimplePhysical Class** module and add your code to create the outputs:
15. Go to the *"Insert your code for output (Body1)"* section. The following code will use the m_oGeomHelper.CreateCone() routine to create a Cone for the Body. In addition, this code uses the RetrieveParameters function to retrieve the nozzle information from the generic data.

```
' Insert your code for output (Body1)

    RetrieveParameters 1, oPartFclt, m_OutputColl, pipeDiam, flangeThick, flangeDiam, cptOffset, depth

    Dim stPosition   As IJDPosition
    Dim enPosition   As IJDPosition

    Set stPosition = New DPosition
    Set enPosition = New DPosition

    stPosition.Set -parFacetoFace / 2 + flangeThick, 0, 0
    enPosition.Set parFacetoFace / 2, 0, 0

    iOutput = iOutput + 1
    Set ObjBody1 = m_oGeomHelper.CreateCone(arrayOfOutputs(iOutput), stPosition, enPosition, pipeDiam
+ flangeDiam / 4, pipeDiam )
```

Use the Set statement to remove references from all object variables.

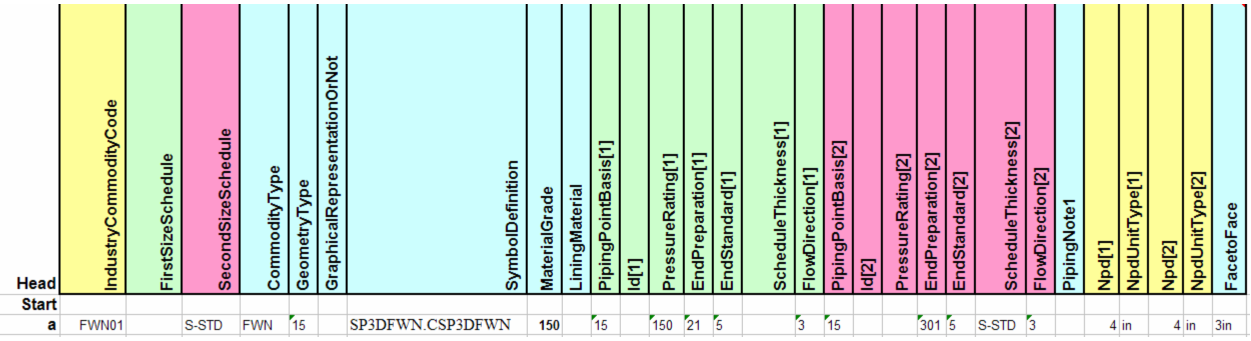
```
    Set ObjBody1 = Nothing
    Set stPosition = Nothing
    Set enPosition = Nothing
```

16. Compile the VB project and save the dll in the c:\train\SP3DFWN

17. Save the VB SP3DFWN project.

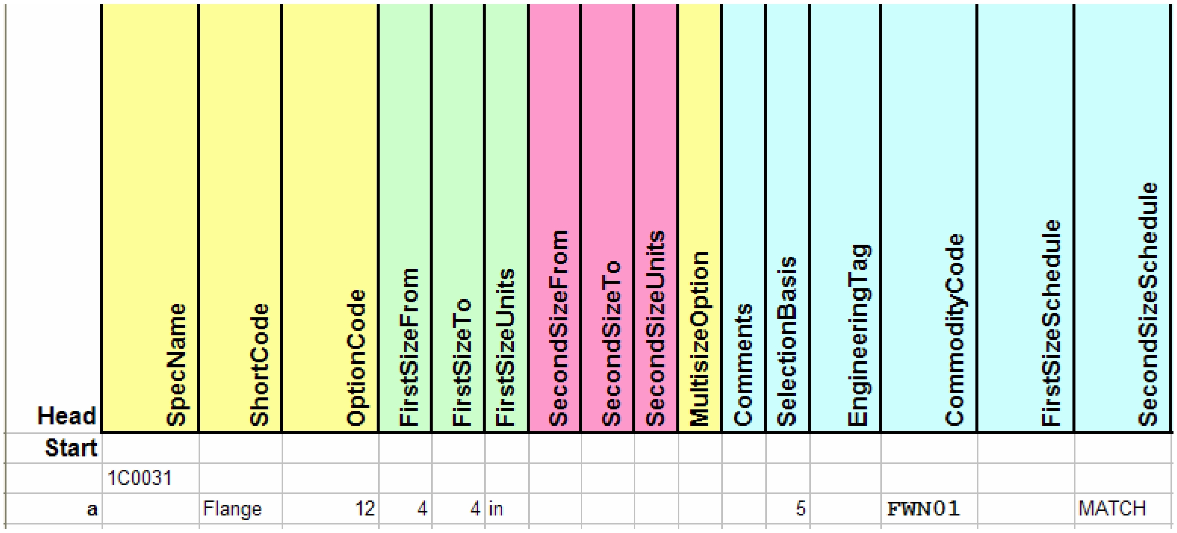
18. Open the [Install Product]\ CatalogData\BulkLoad\Datafiles\ Ten_Specs_CatalogData.xls. Make sure to remove the Read-Only setting on the file.
19. Find the WeldNeckFlange part class and and a new part using the new symbol definition SP3DFWN.CSP3DFWN

In the Part Section:



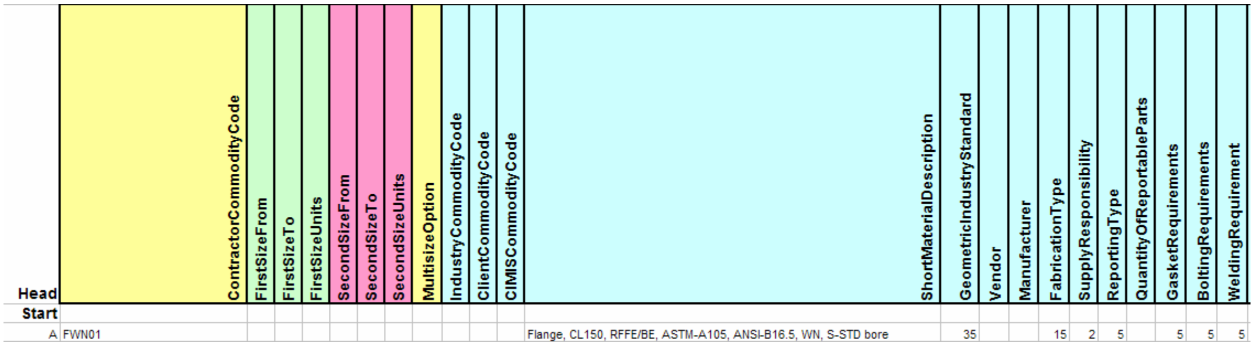
Head	IndustryCommodityCode	FirstSizeSchedule	SecondSizeSchedule	CommodityType	GeometryType	GraphicalRepresentationOrNot	SymbolDefinition	MaterialGrade	LiningMaterial	PipingPointBasis[1]	Id[1]	PressureRating[1]	EndPreparation[1]	EndStandard[1]	ScheduleThickness[1]	FlowDirection[1]	PipingPointBasis[2]	Id[2]	PressureRating[2]	EndPreparation[2]	EndStandard[2]	ScheduleThickness[2]	FlowDirection[2]	PipingNote1	Npd[1]	NpdUnitType[1]	Npd[2]	NpdUnitType[2]	FacetoFace
Start																													
a	FWN01		S-STD	FWN	15		SP3DFWN.CSP3DFWN	150		15		150	21	5		3	15			301	5	S-STD	3		4	in	4	in	3in

20. Save the file in c:\train\SP3DFWN\ Ten_Specs_CatalogData.xls.
21. Open the [Install Product]\ CatalogData\BulkLoad\Datafiles\ Ten_Specs_SpecificationData.xls. Make sure to remove the Read-Only setting on the file.
22. Add a new option for this weldneck flange in the piping commodity filter for spec 1c0031.



Head	SpecName	ShortCode	OptionCode	FirstSizeFrom	FirstSizeTo	FirstSizeUnits	SecondSizeFrom	SecondSizeTo	SecondSizeUnits	MultisizeOption	Comments	SelectionBasis	EngineeringTag	CommodityCode	FirstSizeSchedule	SecondSizeSchedule
Start																
	1C0031															
a		Flange	12	4	4	in						5		FWN01		MATCH

23. Add a new entry in the piping commodity material control data sheet.



Head	ContractorCommodityCode	FirstSizeFrom	FirstSizeTo	FirstSizeUnits	SecondSizeFrom	SecondSizeTo	SecondSizeUnits	MultisizeOption	IndustryCommodityCode	ClientCommodityCode	CIMISCommodityCode	ShortMaterialDescription	GeometricIndustryStandard	Vendor	Manufacturer	FabricationType	SupplyResponsibility	ReportingType	QuantityOfReportableParts	GasketRequirements	BoltingRequirements	WeldingRequirement
Start																						
A	FWN01											Flange, CL150, RFFE/BE, ASTM-A105, ANSI-B16.5, WN, S-STD bore	35			15	2	5		5	5	5

24. Save the file in c:\train\SP3DFWN\ Ten_Specs_SpecificationData.xls. Load the information into the Catalog using the A/M/D mode.
25. Go to the Piping Task and place this flange component using 1c0031 spec.

Lab 15: Piping Instrument Symbol

Objective

After completing this lab, you will be able to:

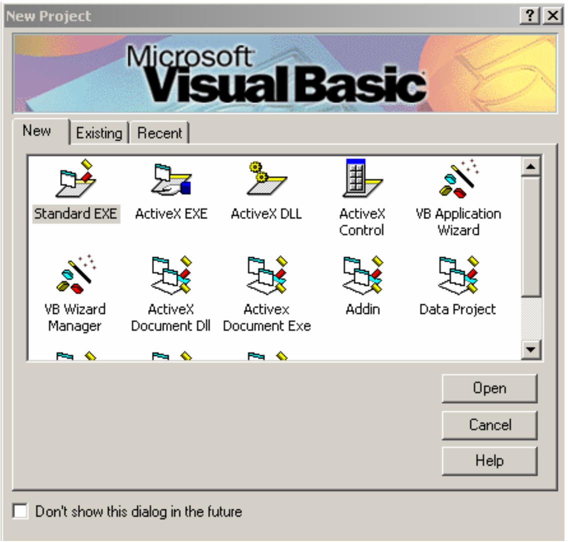
- Create Piping Instrument symbols using the SmartPlant 3D Part Definition Wizard

Create a simple instrument symbol.

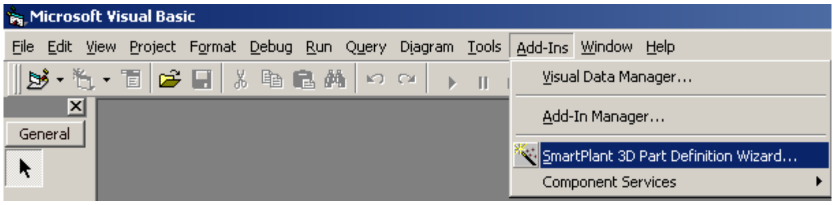
1. Create the following directories:

   c:\train\GenericComp
   c:\train\IngrModules

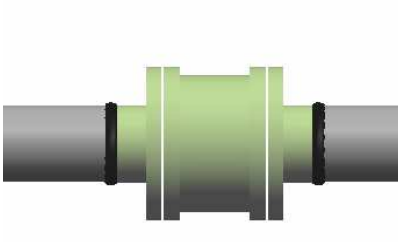
2. Run Microsoft Visual Basic 6.0
3. Close the Microsoft New Project dialog box.



4. Go to the Add-Ins Option and Select SmartPlant 3D Part Definition Wizard.

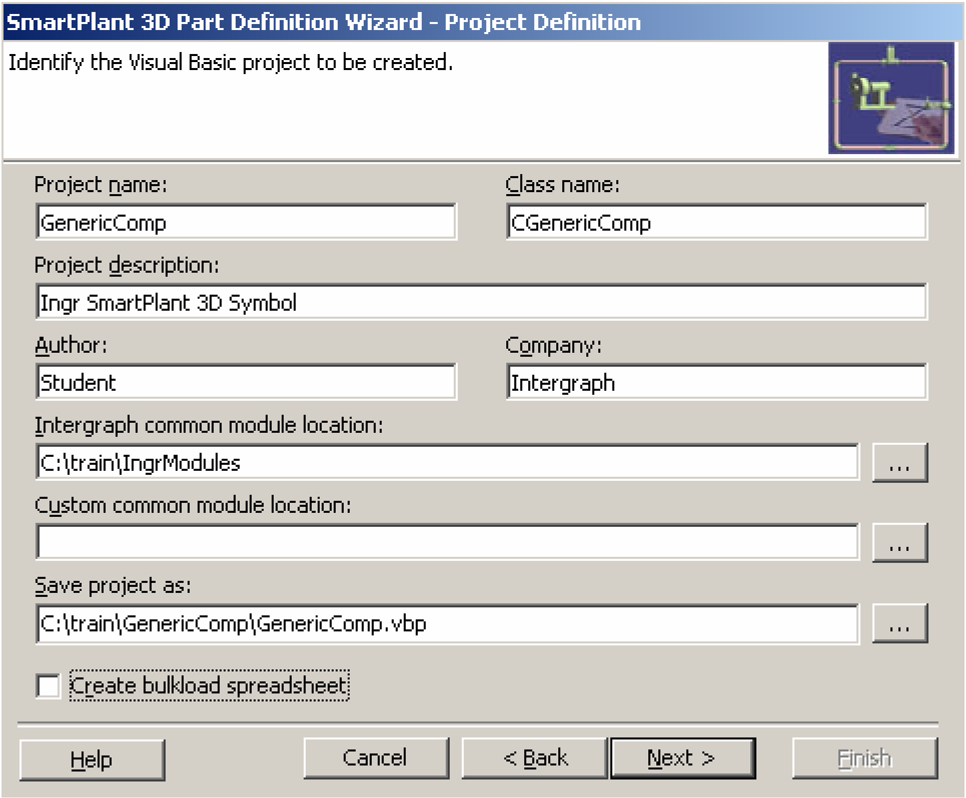

Microsoft Visual Basic
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
Visual Data Manager...
Add-In Manager...
SmartPlant 3D Part Definition Wizard...
Component Services
General

5. The Next step is to create the generic component symbol definition template using SP3D Part Definition Symbol Wizard.



6. In this page you define the VB project name. Key in the following information:

   Project Name: *GenericComp*
   Author: *Student*
   Company: *Intergraph*
   Intergraph Module location: *c:\Train\IngrModules*
   Save the VB project as: *c:\Train\GenericComp*
   Disable the create bulkload spreadsheet.



SmartPlant 3D Part Definition Wizard - Project Definition

Identify the Visual Basic project to be created.

Project name: GenericComp

Class name: CGenericComp

Project description: Ingr SmartPlant 3D Symbol

Author: Student

Company: Intergraph

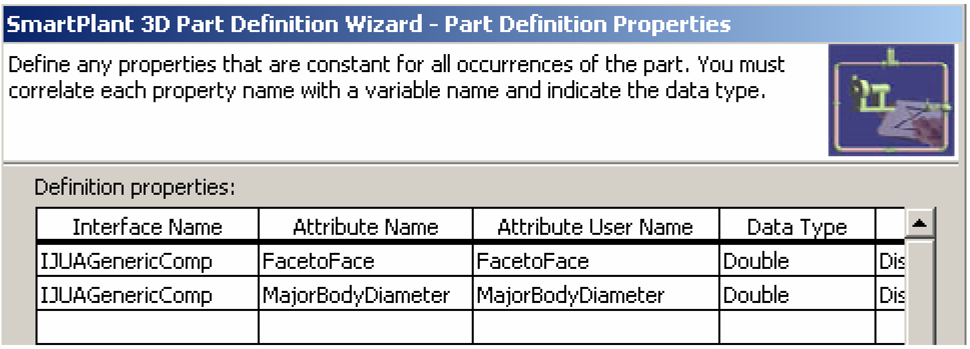
Intergraph common module location: C:\train\IngrModules

Custom common module location:

Save project as: C:\train\GenericComp\GenericComp.vbp

□ Create bulkload spreadsheet

7. Select Next button to go the next page. This page is to define any properties that are constant for all occurrences of the piping part. Key in the following data:

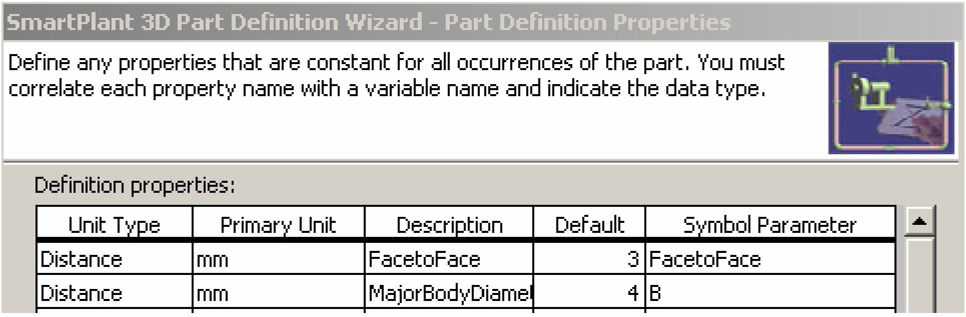


SmartPlant 3D Part Definition Wizard - Part Definition Properties

Define any properties that are constant for all occurrences of the part. You must correlate each property name with a variable name and indicate the data type.

Definition properties:

Interface Name	Attribute Name	Attribute User Name	Data Type	
IJUAGenericComp	FacetoFace	FacetoFace	Double	Dis
IJUAGenericComp	MajorBodyDiameter	MajorBodyDiameter	Double	Dis



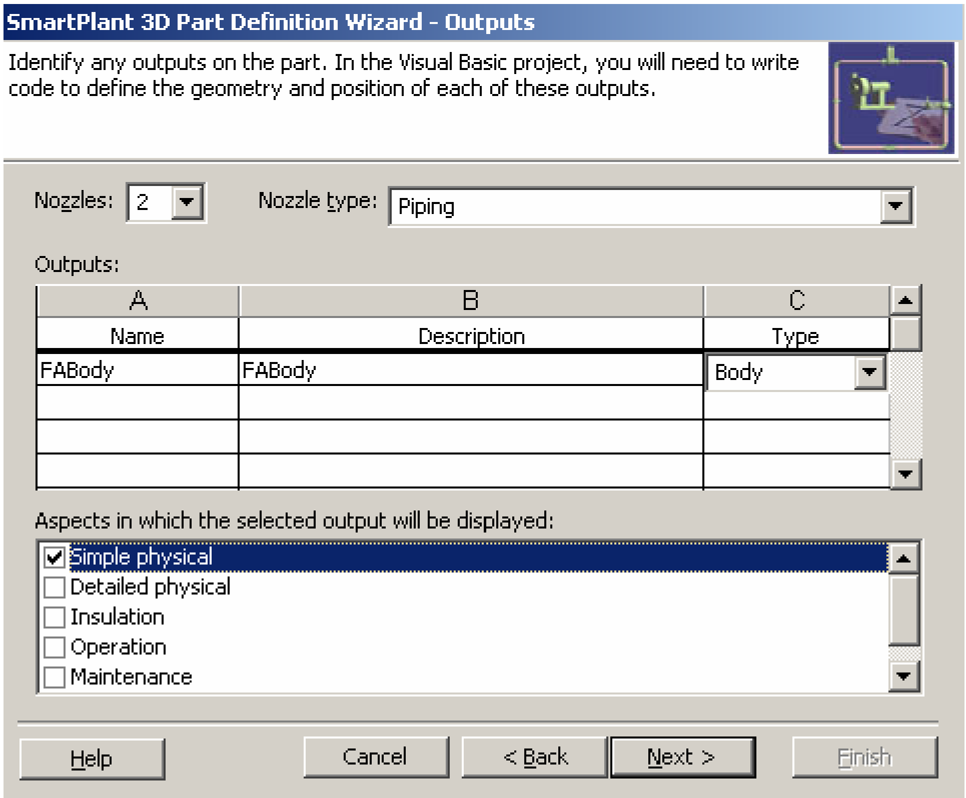
SmartPlant 3D Part Definition Wizard - Part Definition Properties

Define any properties that are constant for all occurrences of the part. You must correlate each property name with a variable name and indicate the data type.

Definition properties:

Unit Type	Primary Unit	Description	Default	Symbol Parameter
Distance	mm	FacetoFace	3	FacetoFace
Distance	mm	MajorBodyDiamel	4	B

8. Select Next button to go the next page. This page defines all occurrence properties of the piping part. Select Next button to go the next page. This page identifies all the outputs of the piping part. We are going to define three outputs: body and two piping nozzles for our GenericComp. The Body output is in the Simple Physical aspect.



SmartPlant 3D Part Definition Wizard - Outputs

Identify any outputs on the part. In the Visual Basic project, you will need to write code to define the geometry and position of each of these outputs.

Nozzles: 2 Nozzle type: Piping

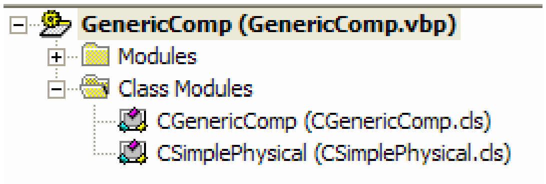
Outputs:

A	B	C
Name	Description	Type
FABody	FABody	Body

Aspects in which the selected output will be displayed:

☑ Simple physical
□ Detailed physical
□ Insulation
□ Operation
□ Maintenance

9. Press Next button and Finish button to create the GenericComp project template. The VB project consists of the following modules:



GenericComp (GenericComp.vbp)
Modules
Class Modules
CGenericComp (CGenericComp.cls)
CSimplePhysical (CSimplePhysical.cls)

10. Open the **CGenericComp Class** module. This Class contains several routines.
11. Go to the Class_Initialize() routine. Review the inputs and outputs section.
12. Make sure the MajorBodyDiameter is mapped to “B” for the second input.

```
Private Sub Class_Initialize()
    Const METHOD = "Class_Initialize:"
    On Error GoTo Errx

    Set m_oSymbolHelper = New SymbolServices
    m_oSymbolHelper.ProjectName = "GenericComp"
    m_oSymbolHelper.ClassName = "CGenericComp"

' Inputs
    m_oSymbolHelper.NumInputs = 2
    m_oSymbolHelper.AddInputDef 1, "FacetoFace", "FacetoFace", 3
    m_oSymbolHelper.AddInputDef 2, "B", "MajorBodyDiameter", 4

' Outputs
    m_oSymbolHelper.NumOutputs = 3
    m_oSymbolHelper.AddOutputDef 1, "FABody", "FABody", 1
    m_oSymbolHelper.AddOutputDef 2, "PipingNoz1", "Nozzle 1", 1
    m_oSymbolHelper.AddOutputDef 3, "PipingNoz2", "Nozzle 2", 1

' Aspects
    m_oSymbolHelper.NumAspects = 1
    m_oSymbolHelper.AddAspectDef 1, "SimplePhysical", "SimplePhysical", 1

    Exit Sub
Errx:
    Err.Raise Err.Number, Err.Source & " " & METHOD, Err.Description, _
        Err.HelpFile, Err.HelpContext
End Sub
```

13. Go to **CSimplePhysical Class** module and add your code to create the outputs:
14. Go to the “*Insert your code for output (FA Body)*” section. The following code will use the m_oGeomHelper.CreateCylinder() routine to create a Cylinder for the Body. The PlaceCylinder routine is defined in the geometry helper service. This function creates persistent projection of a circle based on two points and diameter.

In addition, this code uses the RetrieveParameters function to retrieve the nozzle information from the generic data.

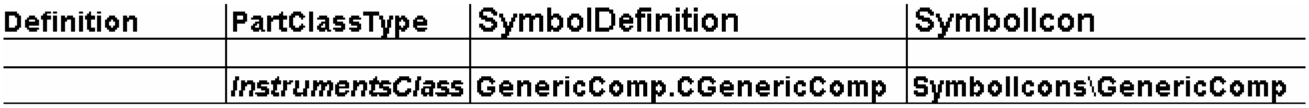
```
' Insert your code for output (FABody)

    RetrieveParameters 1, oPartFclt, m_OutputColl, pipeDiam, flangeThick, flangeDiam, cptOffset, depth
    Dim stPoint   As IJDPosition
    Dim enPoint   As IJDPosition
    Set stPoint = New DPosition
    Set enPoint = New DPosition
    stPoint.Set -parFacetoFace / 2 + flangeThick, 0, 0
    enPoint.Set parFacetoFace / 2 - flangeThick, 0, 0

' Set the output
    iOutput = iOutput + 1
    Set ObjFABody = m_oGeomHelper.CreateCylinder(arrayOfOutputs(iOutput), stPoint, enPoint,
parMajorBodyDiameter)
    Set stPoint = Nothing
    Set enPoint = Nothing
```

Use the Set statement to remove references from all object variables.

```
    Set objNozzle = Nothing
    Set CenterPos = Nothing
    Set oPlacePoint = Nothing
    Set oDir = Nothing
    Set ObjFABody = Nothing
```

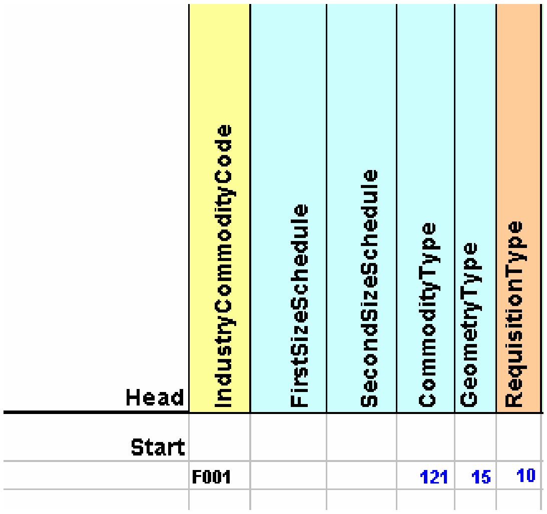
15. Compile the VB project and save the dll in the c:\train\GenericComp
16. Save the VB GenericComp project.
17. Open the [Install Product]\ CatalogData\BulkLoad\Datafiles\Instrument Data.xls. Make sure to remove the Read-Only setting on the file.
18. Create a New Part Class called GenericComp with the following data:
    Hint: Use the ANG sheet as a template

In the Definition Section:



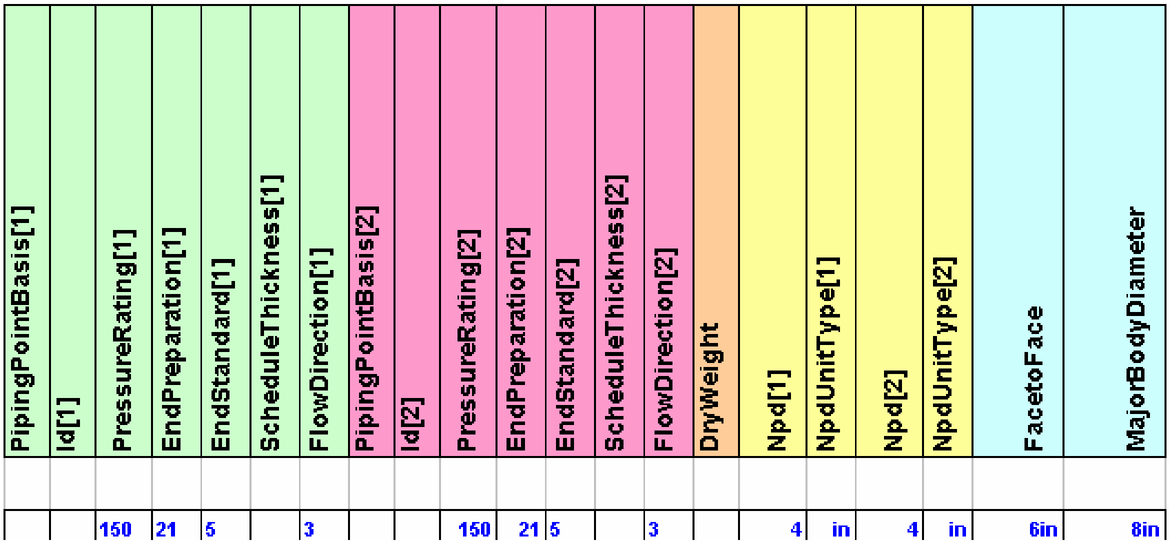
Definition	PartClassType	SymbolDefinition	SymbolIcon
	InstrumentsClass	GenericComp.CGenericComp	SymbolIcons\GenericComp

In the Part Section:
System attributes:



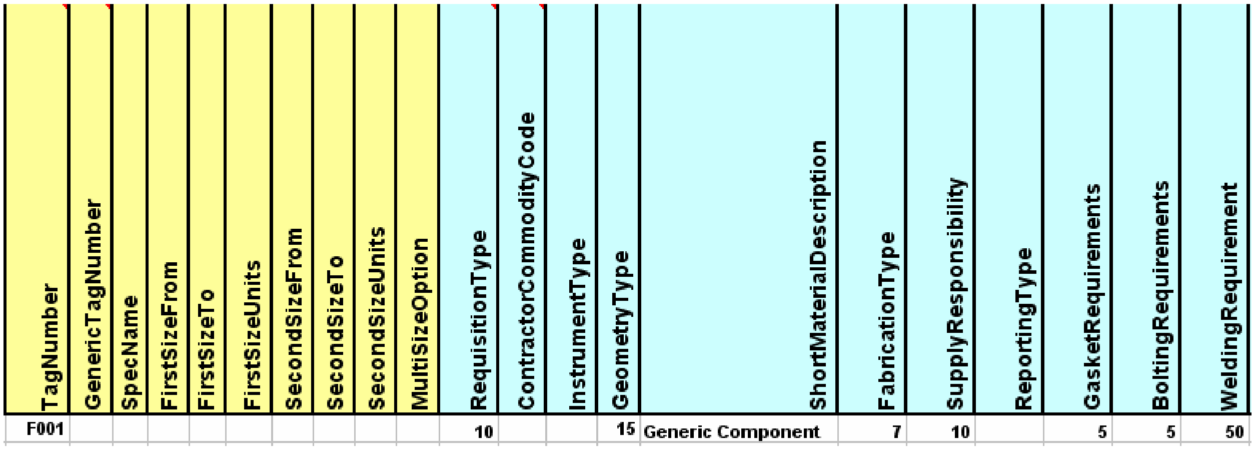
	IndustryCommodityCode	FirstSizeSchedule	SecondSizeSchedule	CommodityType	GeometryType	RequisitionType
Head						
Start						
	F001			121	15	10

Ports and generic component body data:



PipingPointBasis[1]	Id[1]	PressureRating[1]	EndPreparation[1]	EndStandard[1]	ScheduleThickness[1]	FlowDirection[1]	PipingPointBasis[2]	Id[2]	PressureRating[2]	EndPreparation[2]	EndStandard[2]	ScheduleThickness[2]	FlowDirection[2]	DryWeight	Npd[1]	NpdUnitType[1]	Npd[2]	NpdUnitType[2]	FacetoFace	MajorBodyDiameter
		150	21	5		3			150	21	5		3		4	in	4	in	6in	8in

19. Go to the InstrumentClassData sheet and add the following data:



TagNumber	GenericTagNumber	SpecName	FirstSizeFrom	FirstSizeTo	FirstSizeUnits	SecondSizeFrom	SecondSizeTo	SecondSizeUnits	MultiSizeOption	RequisitionType	ContractorCommodityCode	InstrumentType	GeometryType	ShortMaterialDescription	FabricationType	SupplyResponsibility	ReportingType	GasketRequirements	BoltingRequirements	WeldingRequirement
F001										10			15	Generic Component	7	10		5	5	50

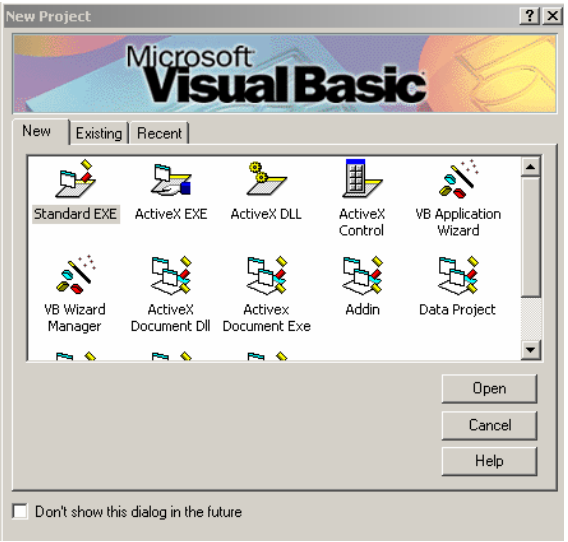
20. Save the file in c:\train\GenericComp\ Instrument Data.xls. Load the information into the Catalog using the Append mode. Once the bulkload process is complete, run the View Generator utility on the model to re-create the views in the model database. Finally, Re-generate the report databases.
21. Go to the Piping Task and place the Generic Component.

Lab 16: Valve Operator Symbol

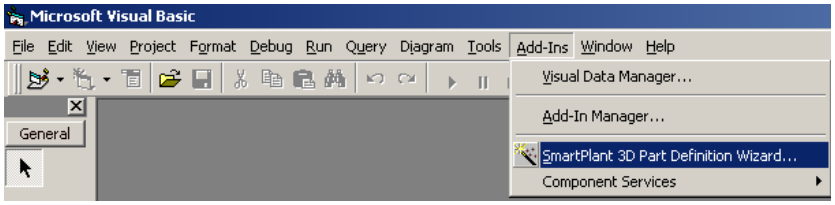
1. Create the following directory:

   c:\train\ SP3DOP_431

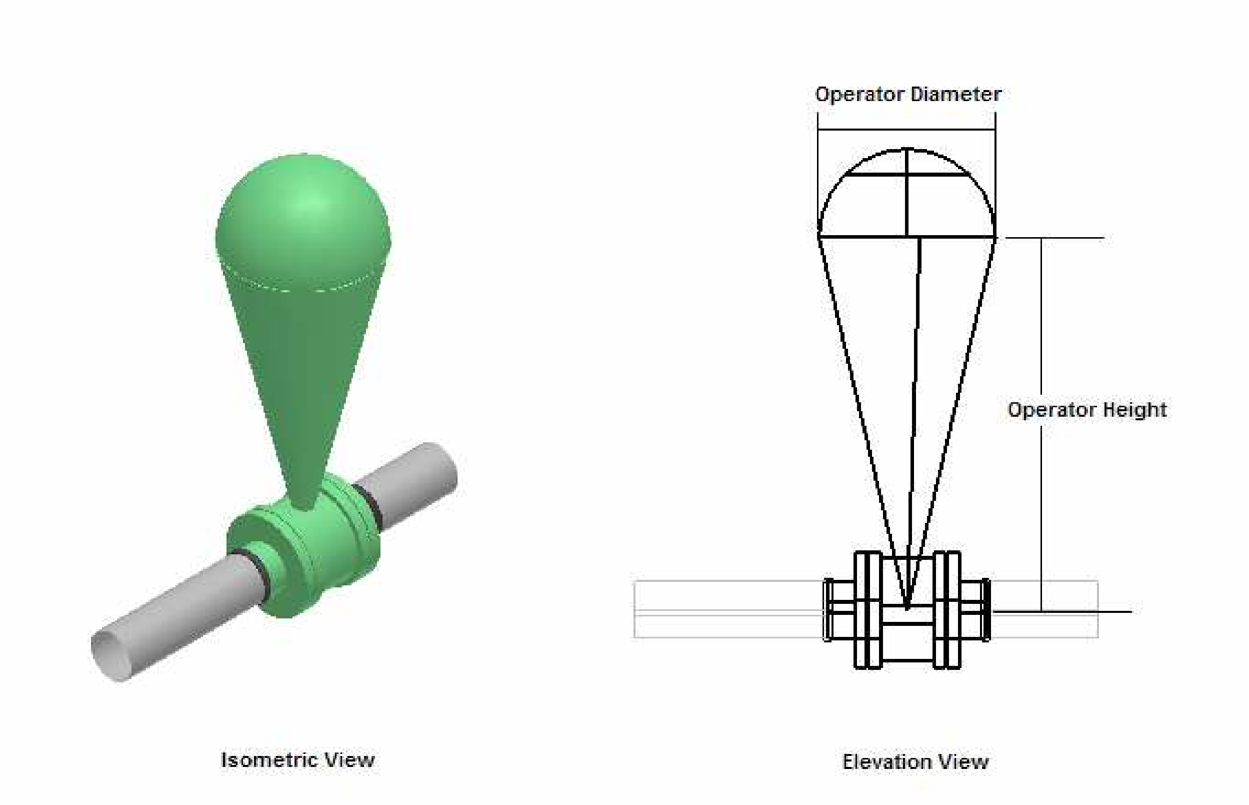
2. Run Microsoft Visual Basic 6.0
3. Close the Microsoft New Project dialog box.



4. Go to the Add-Ins Option and Select SmartPlant 3D Part Definition Wizard.

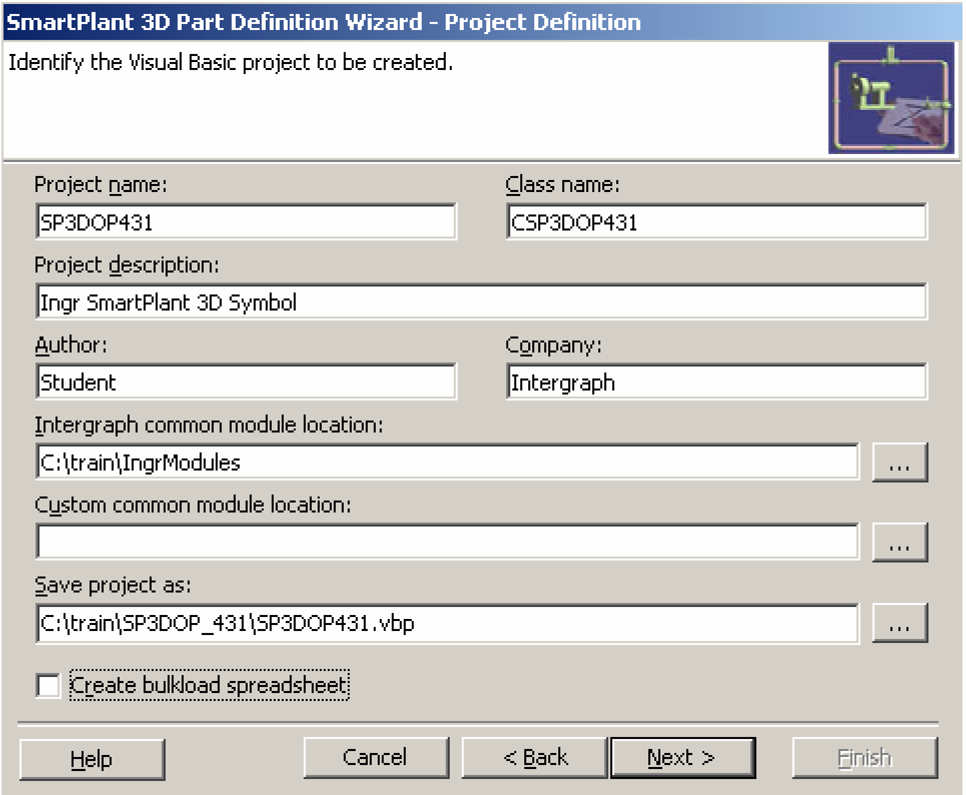

Microsoft Visual Basic
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
General
Visual Data Manager...
Add-In Manager...
SmartPlant 3D Part Definition Wizard...
Component Services

5. The Next step is to create the operator symbol definition template using SP3D Part Definition Symbol Wizard.



6. In this page you define the VB project name. Key in the following information:

   Project Name: *SP3DOP431*
   Author: *Student*
   Company: *Intergraph*
   Intergraph Module location: *c:\Train\IngrModules*
   Save the VB project as: *c:\Train\ SP3DOP_431*
   Disable the create bulkload spreadsheet.



SmartPlant 3D Part Definition Wizard - Project Definition

Identify the Visual Basic project to be created.

Project name: SP3DOP431

Class name: CSP3DOP431

Project description: Ingr SmartPlant 3D Symbol

Author: Student

Company: Intergraph

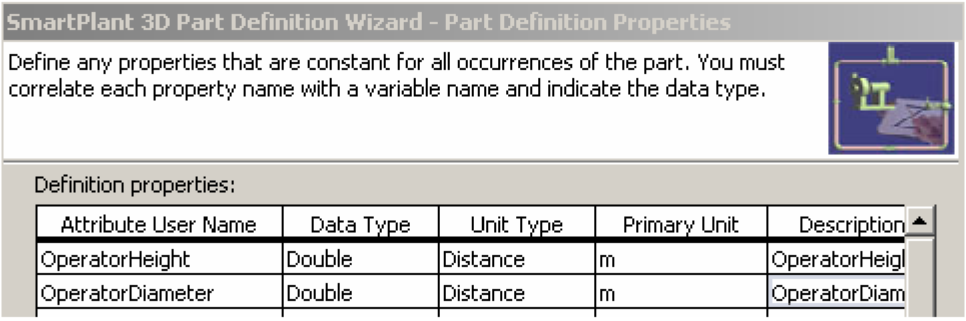
Intergraph common module location: C:\train\IngrModules

Custom common module location:

Save project as: C:\train\SP3DOP_431\SP3DOP431.vbp

☐ Create bulkload spreadsheet

7. Select Next button to go the next page. This page is to define any properties that are constant for all occurrences of the operator part. Key in the following data:

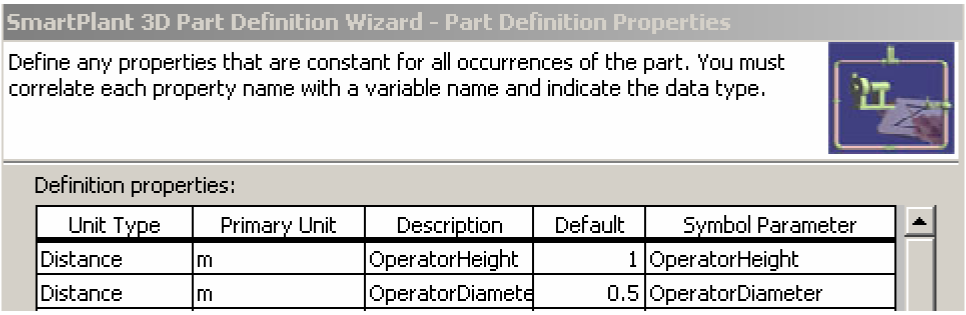


SmartPlant 3D Part Definition Wizard - Part Definition Properties

Define any properties that are constant for all occurrences of the part. You must correlate each property name with a variable name and indicate the data type.

Definition properties:

Attribute User Name	Data Type	Unit Type	Primary Unit	Description
OperatorHeight	Double	Distance	m	OperatorHeigl
OperatorDiameter	Double	Distance	m	OperatorDiam



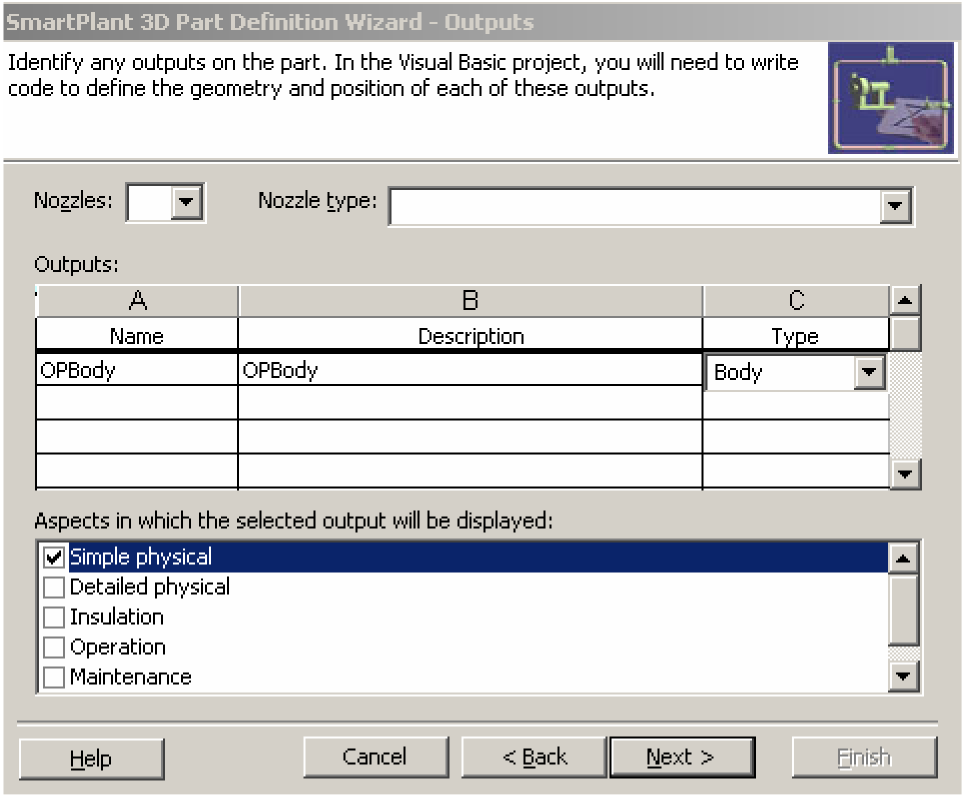
SmartPlant 3D Part Definition Wizard - Part Definition Properties

Define any properties that are constant for all occurrences of the part. You must correlate each property name with a variable name and indicate the data type.

Definition properties:

Unit Type	Primary Unit	Description	Default	Symbol Parameter
Distance	m	OperatorHeight	1	OperatorHeight
Distance	m	OperatorDiamete	0.5	OperatorDiameter

8. Select Next button to go the next page. This page defines all occurrence properties of the operator part. Select Next button to go the next page. This page identifies all the outputs of the operator part. We are going to define one output: Operator Body



SmartPlant 3D Part Definition Wizard - Outputs

Identify any outputs on the part. In the Visual Basic project, you will need to write code to define the geometry and position of each of these outputs.

Nozzles: Nozzle type:

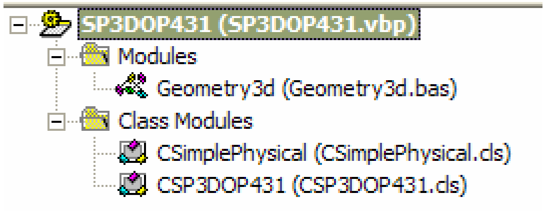
Outputs:

A	B	C
Name	Description	Type
OPBody	OPBody	Body

Aspects in which the selected output will be displayed:

☑ Simple physical
☐ Detailed physical
☐ Insulation
☐ Operation
☐ Maintenance

9. Press Next button and Finish button to create the Operator project template. The VB project consists of the following modules:



SP3DOP431 (SP3DOP431.vbp)
- Modules
  - Geometry3d (Geometry3d.bas)
- Class Modules
  - CSimplePhysical (CSimplePhysical.cls)
  - CSP3DOP431 (CSP3DOP431.cls)

10. Open the **CSP3DOP431 Class** module. This Class contains several routines.
11. Go to the Class_Initialize() routine. Review the inputs and outputs section. Add two additional outputs as shown below:

```
Private Sub Class_Initialize()
    Const METHOD = "Class_Initialize:"
    On Error GoTo Errx

    Set m_oSymbolHelper = New SymbolServices
    m_oSymbolHelper.ProjectName = "SP3DOP431"
```

```
   m_oSymbolHelper.ClassName = "CSP3DOP431"

' Inputs
   m_oSymbolHelper.NumInputs = 2
   m_oSymbolHelper.AddInputDef 1, "OperatorHeight", "OperatorHeight", 1
   m_oSymbolHelper.AddInputDef 2, "OperatorDiameter", "OperatorDiameter", 0.5

 ' Outputs
   m_oSymbolHelper.NumOutputs = 3
   m_oSymbolHelper.AddOutputDef 1, "OPBody1", "OPBody1", 1
   m_oSymbolHelper.AddOutputDef 2, "OPBody2", "OPBody2", 1
   m_oSymbolHelper.AddOutputDef 3, "OPBody3", "OPBody3", 1

' Aspects
   m_oSymbolHelper.NumAspects = 1
   m_oSymbolHelper.AddAspectDef 1, "SimplePhysical", "SimplePhysical", 1

   Exit Sub
Errx:
   Err.Raise Err.Number, Err.Source & " " & METHOD, Err.Description, _
      Err.HelpFile, Err.HelpContext
End Sub
```

12. Go to **CSimplePhysical Class** module and add your code to create the outputs:
13. Go to the "*Insert your code for output 1 (OPBody)*" section. The following code will use the m_oGeomHelper.CreateCone() routine to create a Cone for the Body. The PlaceCone routine is defined in the geometry helper service. This function creates a persistent object based on two points and two diameters.

```
' Insert your code for output (OPBody)
   Dim ConeCenterBasePt    As IJDPosition
   Dim ConeCenterTopPt     As IJDPosition
   Set ConeCenterBasePt = New DPosition
   Set ConeCenterTopPt = New Dposition

 '   A value of 0.0000001 is used to avoid symbol placement failure (gives assertion errors).

   ConeCenterTopPt.Set 0, parOperatorHeight, 0
   ConeCenterBasePt.Set 0, 0, 0.0000001

   Dim ObjOPBody  As IngrGeom3D.Cone3d
   iOutput = iOutput + 1

'   A value of 0.00001 is used to avoid symbol placement failure (gives assertion errors).

   Set ObjOPBody = m_oGeomHelper.CreateCone(arrayOfOutputs(iOutput),
ConeCenterBasePt,ConeCenterTopPt, 0.00001, parOperatorDiameter)
```

14. The following code will use the Geometry factory functions to create a dome for the top of the operator.

```
   Dim oGeomFactory As New IngrGeom3D.GeometryFactory
   Dim dCircleCenterX As Double, dCircleCenterY As Double, dCircleCenterZ As Double
```

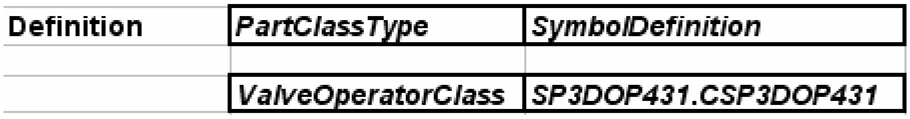
```
Dim oEllipticalArc As IngrGeom3D.EllipticalArc3d
Dim oRevolution As IngrGeom3D.Revolution3d
Dim PI As Double

Dim dRadius As Double

'Normal is North=  z-axis

PI = 4 * Atn(1)
dRadius = parOperatorDiameter / 2
Set oEllipticalArc = oGeomFactory.EllipticalArcs3d.CreateByCenterNormalMajAxisRatioAngle(Nothing, _
        0, parOperatorHeight, 0, 0, 0, 1, 0, dRadius, 0, _
        1, PI * 1.5, PI / 2)

Set oRevolution = oGeomFactory.Revolutions3d.CreateByCurve(m_OutputColl.ResourceManager, _
                oEllipticalArc, 0, 1, 0, 0, 0, 0, 2 * PI, False)

iOutput = iOutput + 1
m_OutputColl.AddOutput arrayOfOutputs(iOutput), oRevolution

Dim oCircle As IngrGeom3D.Circle3d
Set oCircle = oGeomFactory.Circles3d.CreateByCenterNormalRadius(m_OutputColl.ResourceManager, _
0, parOperatorHeight, 0, 0, 1, 0, dRadius)

iOutput = iOutput + 1
m_OutputColl.AddOutput arrayOfOutputs(iOutput), oCircle

Set oCircle = Nothing
Set oRevolution = Nothing
Set oEllipticalArc = Nothing
Set oGeomFactory = Nothing
Set ConeCenterBasePt = Nothing
Set ConeCenterTopPt = Nothing
```

Note: Go to the variable declaration and delete Dim ObjOPBody As Object

15. Compile the VB project and save the dll in the c:\Train\SP3DOP_431
16. Save the VB SP3DOP431 project.

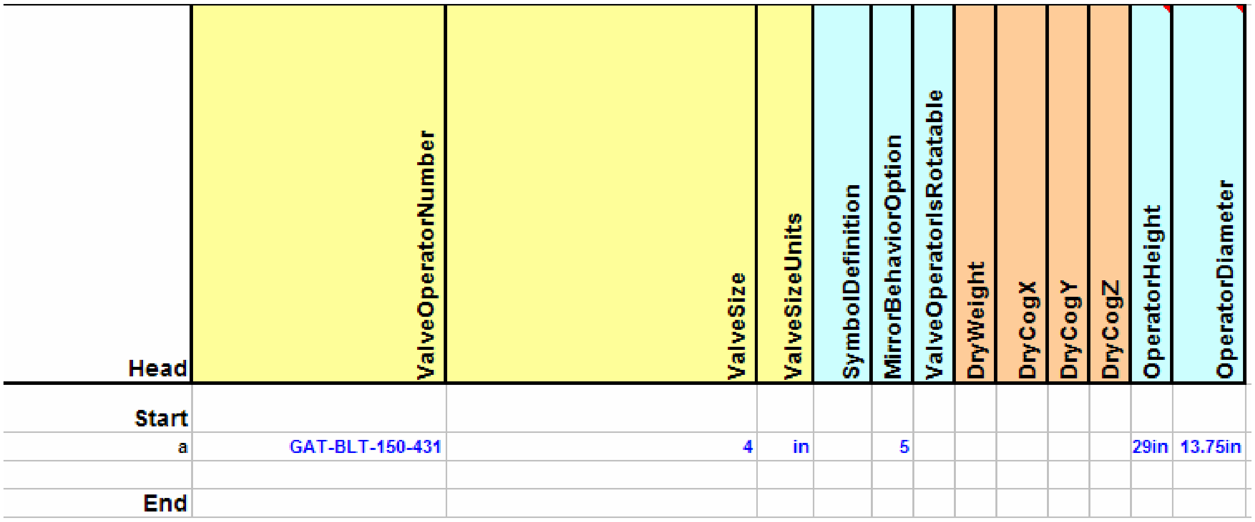
17. Open the Instrument Data.xls.
18. Create a New Part Class called Operator431 with the following data:
    Hint: Copy the Operator3 sheet from the Ten_Specs_CatalogData.xls

In the Definition Section:



Definition	*PartClassType*	*SymbolDefinition*
	ValveOperatorClass	*SP3DOP431.CSP3DOP431*

In the Part Section:

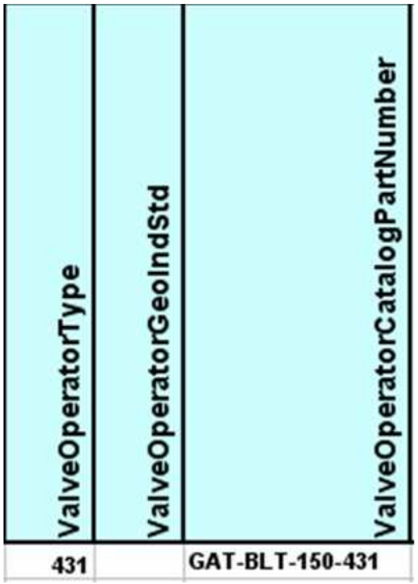


Head	ValveOperatorNumber	ValveSize	ValveSizeUnits	SymbolDefinition	MirrorBehaviorOption	ValveOperatorIsRotatable	DryWeight	DryCogX	DryCogY	DryCogZ	OperatorHeight	OperatorDiameter
Start												
a	GAT-BLT-150-431	4	in		5						29in	13.75in
End												

Add the letter A on the part class and part entries.

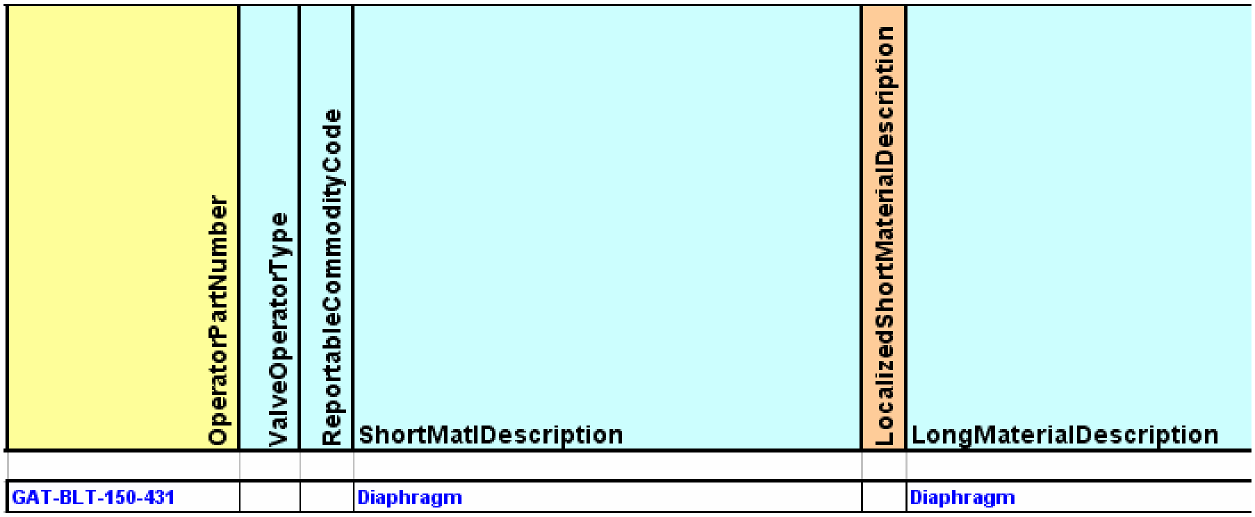
19. Go to the InstrumentClassData sheet and add the following data to F001 instrument. Remember to add the letter M.

    Valve Operator Data:



ValveOperatorType	ValveOperatorGeoIndStd	ValveOperatorCatalogPartNumber
431		GAT-BLT-150-431

20. Go to the ValveOperatorMatlControlData sheet located in Ten_Specs_SpecificationData.xls and add the following data:



OperatorPartNumber	ValveOperatorType	ReportableCommodityCode	ShortMatlDescription	LocalizedShortMaterialDescription	LongMaterialDescription
GAT-BLT-150-431			Diaphragm		Diaphragm

Modify the Instrument symbol GenericComp (done in previous lab) by adding the operator 431 as follows:

21. Open the GenericComp.vbp project.
22. Go to the output section and add the operator as shown below:

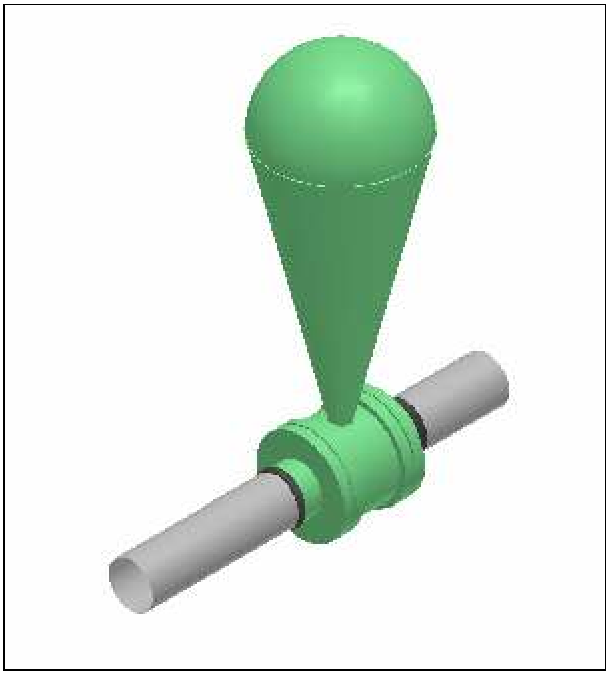
```
' Outputs
   m_oSymbolHelper.NumOutputs = 4
   m_oSymbolHelper.AddOutputDef 1, "FABody", "FABody", 1
   m_oSymbolHelper.AddOutputDef 2, "PipingNoz1", "Nozzle 1", 1
   m_oSymbolHelper.AddOutputDef 3, "PipingNoz2", "Nozzle 2", 1
   m_oSymbolHelper.AddOutputDef 4, "ValveOperator", "ValveOperator", 1
```

23. Go to **CSimplePhysical Class** module and add your code to reference the operator symbol:

```
   Dim oSymbolHelper As IJSymbolGeometryHelper
   Set oSymbolHelper = New SP3DSymbolHelper.SymbolServices
   oSymbolHelper.OutputCollection = m_OutputColl

   On Error Resume Next
   Dim oDirX As IJDVector
   Dim oDirY As IJDVector
   Dim oDirZ As IJDVector
   Set oDirX = New DVector
   Set oDirY = New DVector
   Set oDirZ = New DVector

   oDirX.Set 1, 0, 0
   oDirY.Set 0, 1, 0
   oDirZ.Set 0, 0, 1

   Dim oPipeComponent As IJDPipeComponent
   Set oPipeComponent = oPartFclt
   On Error GoTo ErrorLabel
   Dim oOperatorPart As IJDPart
   Dim oOperatorOcc   As IJPartOcc
```

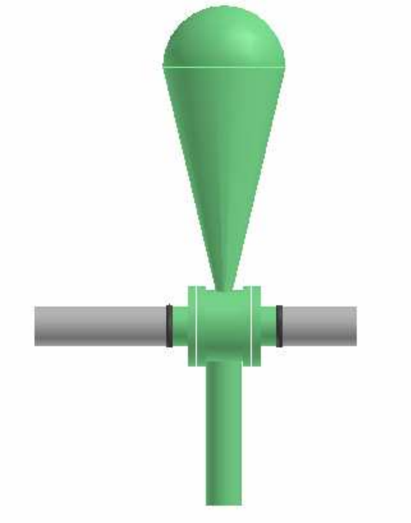
```
    If Not oPipeComponent Is Nothing Then
        Set oOperatorPart = oPipeComponent.GetValveOperatorPart
        If Not oOperatorPart Is Nothing Then
            Dim OpOrigin As IJDPosition
            Set OpOrigin = New DPosition
            OpOrigin.Set 0, 0, 0
            Set oOperatorOcc = oSymbolHelper.CreateChildPartOcc("ValveOperator", oOperatorPart, OpOrigin,
oDirX, oDirY, oDirZ)

        End If
    End If
    Set oSymbolHelper = Nothing
    Set oOperatorPart = Nothing
    Set oPipeComponent = Nothing
    Set oOperatorOcc = Nothing
```

24. Open the properties page of the vb project and increase the dll version number.
25. Compile the VB project and save the dll in the c:\Train\ *GenericComp*
26. Save the VB GenericComp project.
27. Load the information into the Catalog using the Add/Edit/Modify mode. Once the bulkload process is complete.
28. Go to Project Management task and run the synchronize model with the catalog database command. Finally, Re-generate the report databases.
29. Go to the Piping Task and place the Generic Component. Notice the system displays the operator.



Lab 17: Piping Symbol with Multiple Representations

Modify the instrument symbol (GenericComp) by adding another cylinder shape to represent a reserved space for operations



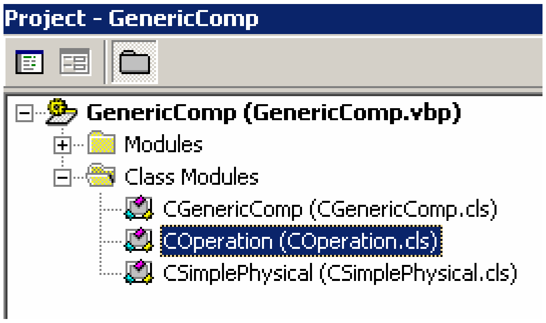
1. Open the GenericComp.vb program and add/edit the following entries in the output section:

```
' Outputs
   m_oSymbolHelper.NumOutputs = 5
   m_oSymbolHelper.AddOutputDef 1, "FABody", "FABody", 1
   m_oSymbolHelper.AddOutputDef 2, "PipingNoz1", "Nozzle 1", 1
   m_oSymbolHelper.AddOutputDef 3, "PipingNoz2", "Nozzle 2", 1
   m_oSymbolHelper.AddOutputDef 4, "ValveOperator", "ValveOperator", 1
   m_oSymbolHelper.AddOutputDef 5, "Shape1", "Shape1", 64
```

2. Add/edit the following entries in the Aspect section:

```
' Aspects
   m_oSymbolHelper.NumAspects = 2
   m_oSymbolHelper.AddAspectDef 1, "SimplePhysical", "SimplePhysical", 1
   m_oSymbolHelper.AddAspectDef 2, "Operation", "Operation", 64
```

3. Create a new class module called COperation


Project - GenericComp
GenericComp (GenericComp.vbp)
Modules
Class Modules
CGenericComp (CGenericComp.cls)
COperation (COperation.cls)
CSimplePhysical (CSimplePhysical.cls)

Add the following code in the COperation class module:

In the Declaration section:

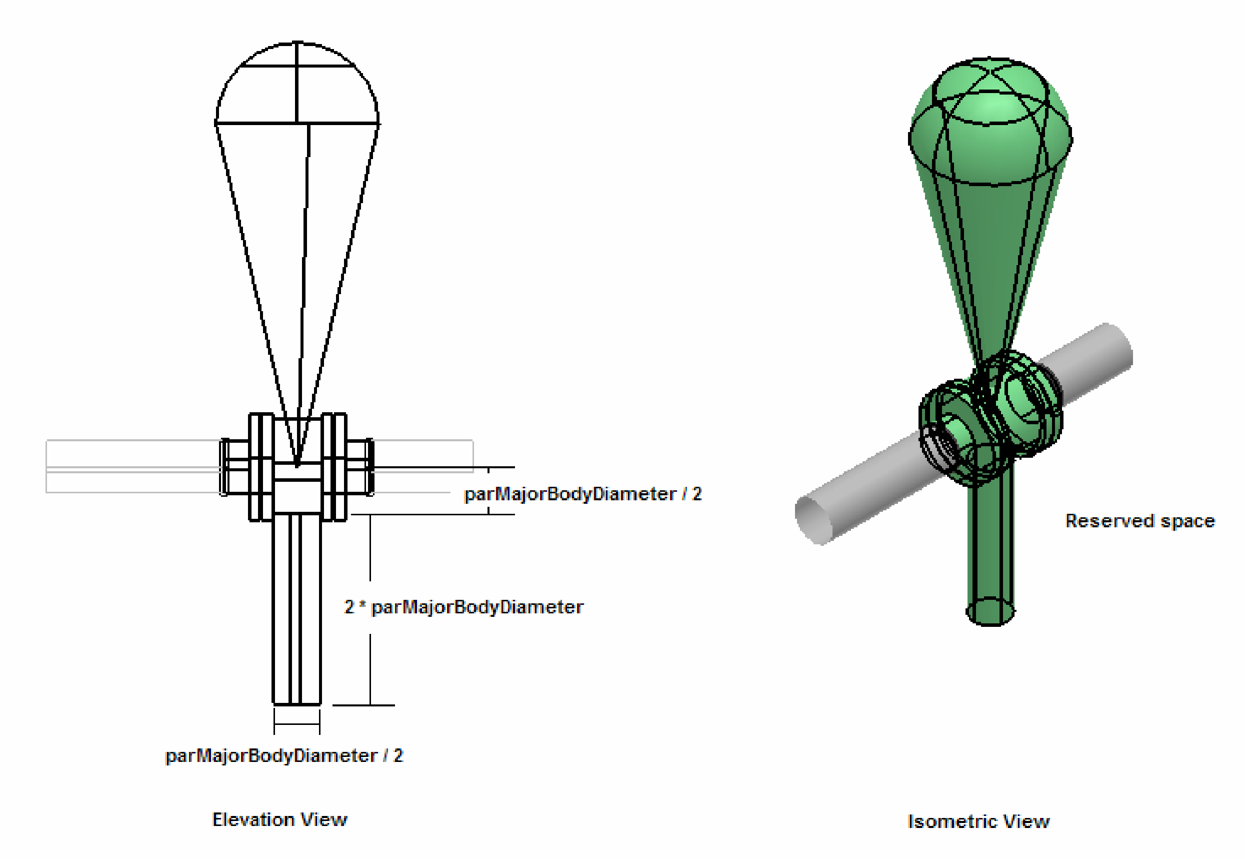
```
Option Explicit
Private Const MODULE = "CGenericComp" 'Used for error messages
Private m_oGeomHelper As IJSymbolGeometryHelper
Private Const E_FAIL = &H80004005
```

In the Class_Initialize subroutine:

```
Private Sub Class_Initialize()
  Const METHOD = "Class_Initialize"
  On Error GoTo Errx
  Set m_oGeomHelper = New SymbolServices
  Exit Sub

Errx:
  Err.Raise Err.Number, Err.Source & " " & METHOD, Err.Description, _
    Err.HelpFile, Err.HelpContext
End Sub
```

Create the Run() routine.
Hint: Copy the lines from the physical module and edit the appropriate lines.

Use the PlaceCylinder() routine to create a cylinder for the shape. The PlaceCylinder routine is defined in the geometry helper service. This function creates a persistent projection of a circle based on two points and diameter.



Elevation View

Isometric View

```
Public Sub run(ByVal m_OutputColl As Object, ByRef arrayOfInputs(), arrayOfOutputs() As String)

  Const METHOD = "run"
  On Error GoTo ErrorLabel

  Dim oPartFclt      As PartFacelets.IJDPart
  Dim pipeDiam       As Double
  Dim flangeThick    As Double
  Dim cptOffset      As Double
  Dim flangeDiam     As Double
  Dim depth          As Double

  Dim iOutput     As Double
  Dim ObjFABody As Object
  Dim parFacetoFace As Double
  Dim parMajorBodyDiameter As Double

' Inputs
  Set oPartFclt = arrayOfInputs(1)
  parFacetoFace = arrayOfInputs(2)
  parMajorBodyDiameter = arrayOfInputs(3)
  m_oGeomHelper.OutputCollection = m_OutputColl

  iOutput = 0
 ' Insert your code for output (Shape1)

  RetrieveParameters 1, oPartFclt, m_OutputColl, pipeDiam, flangeThick, flangeDiam, cptOffset, depth
```

```
    Dim stPoint   As IJDPosition
    Dim enPoint   As IJDPosition
    Set stPoint = New DPosition
    Set enPoint = New DPosition

    stPoint.Set 0, - parMajorBodyDiameter / 2, 0
    enPoint.Set 0, - parMajorBodyDiameter / 2 - 2 * parMajorBodyDiameter, 0

    iOutput = iOutput + 1
    Set ObjFABody = m_oGeomHelper.CreateCylinder(arrayOfOutputs(iOutput), stPoint, enPoint,
parMajorBodyDiameter / 2)
    Set stPoint = Nothing
    Set enPoint = Nothing
    Set ObjFABody = Nothing
    Exit Sub

ErrorLabel:
    Err.Raise Err.Number, Err.Source & " " & METHOD, Err.Description, _
       Err.HelpFile, Err.HelpContext
    Resume Next
EndSub
```

In the Class_Terminate subroutine:

```
Private Sub Class_Terminate()
    Set m_oGeomHelper = Nothing
End Sub
```

4. Open the properties page of the vb project and increase the dll version number.
5. Compile the VB project and save the dll in the c:\Train\ *GenericComp*
6. Save the VB GenericComp project.
7. Go to the instrument part class and add the letter M to the part class definition and part.
8. Update the part class instrument using the bulkload utility.
9. Go to Project Management task and run the synchronize model with the catalog database command.
10. Go to the Piping Task and place the Generic Component. Turn on the Operation Aspect and notice the system displays the new shape.

Lab 18: 90 deg vertical outside cabletray fitting Symbol (Optional)

Objective

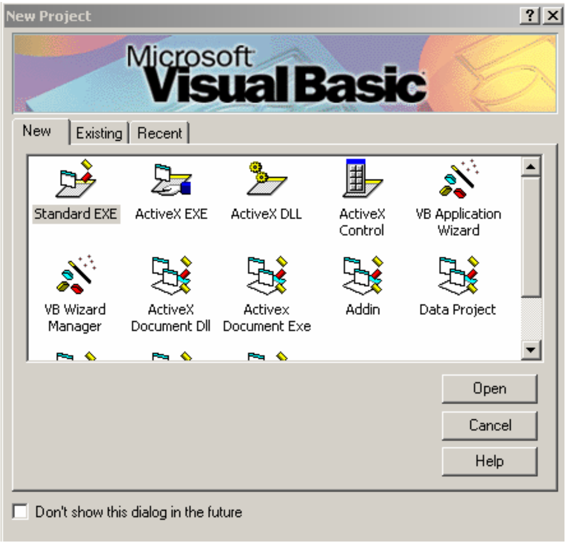
After completing this lab, you will be able to:

- Create cabletray component symbols using the SmartPlant 3D Part Definition Wizard

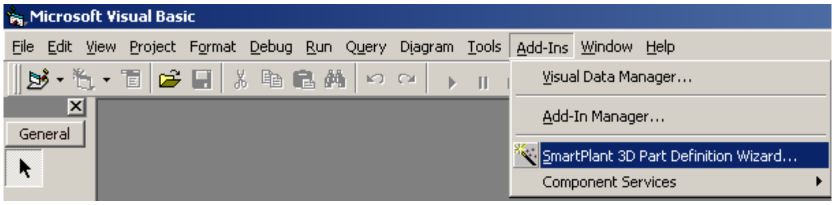
1. Create the following directories:

   c:\train\SP3D90VTrayOutside

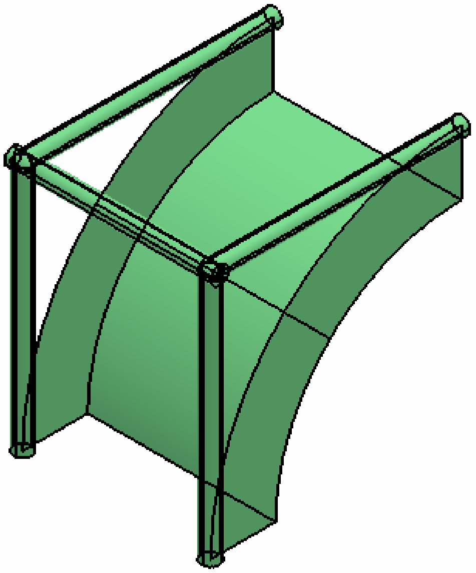
2. Run Microsoft Visual Basic 6.0
3. Close the Microsoft New Project dialog box.



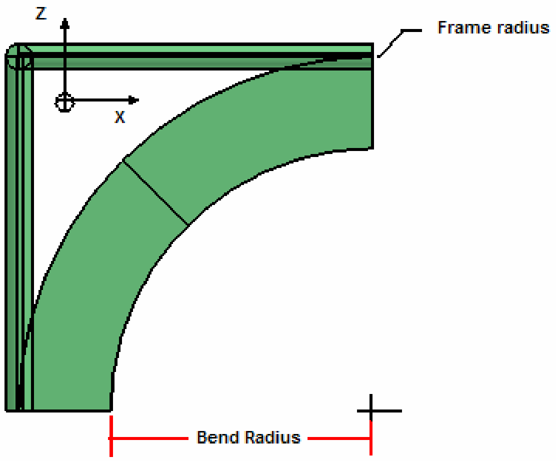
4. Go to the Add-Ins Option and Select SmartPlant 3D Part Definition Wizard.



5. The Next step is to create the 90 deg vertical outside cabletray symbol definition template using SP3D Part Definition Symbol Wizard.



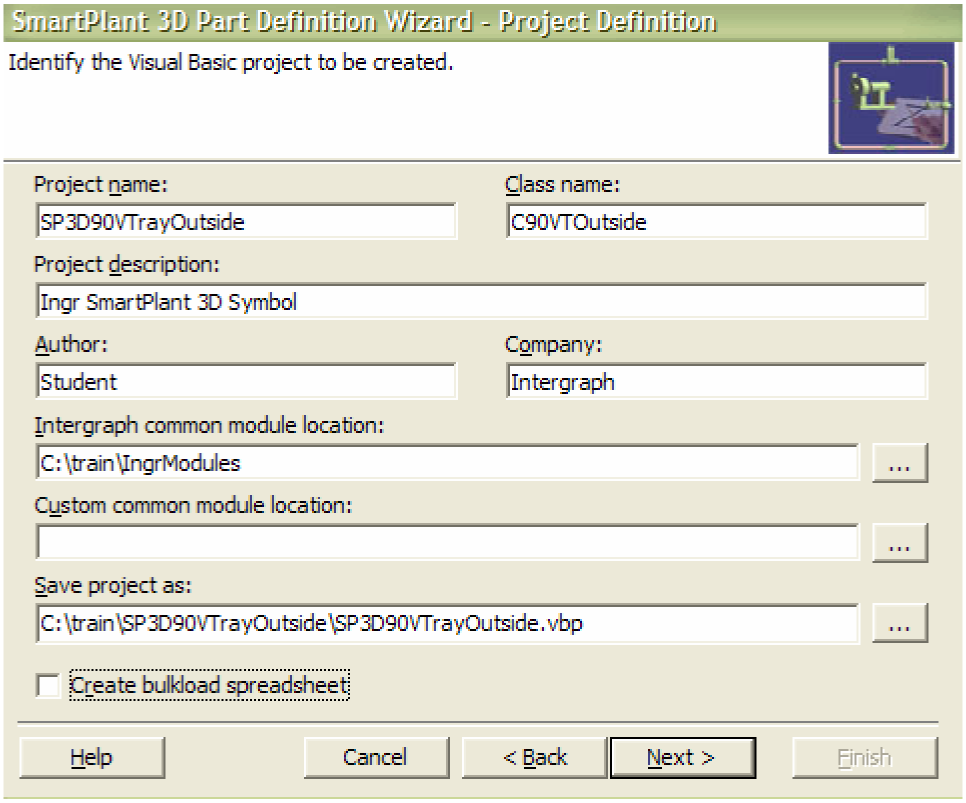
Isometric View



Elevation View

6. In this page you define the VB project name. Key in the following information:

   Project Name: *SP3D90VTrayOutside*
   Author: *Student*
   Company: *Intergraph*
   Intergraph Module location: *c:\Train\IngrModules*
   Save the VB project as: *c:\Train\ SP3D90VTrayOutside*
   Disable the create bulkload spreadsheet.



SmartPlant 3D Part Definition Wizard - Project Definition

Identify the Visual Basic project to be created.

Project name: SP3D90VTrayOutside

Class name: C90VTOutside

Project description: Ingr SmartPlant 3D Symbol

Author: Student

Company: Intergraph

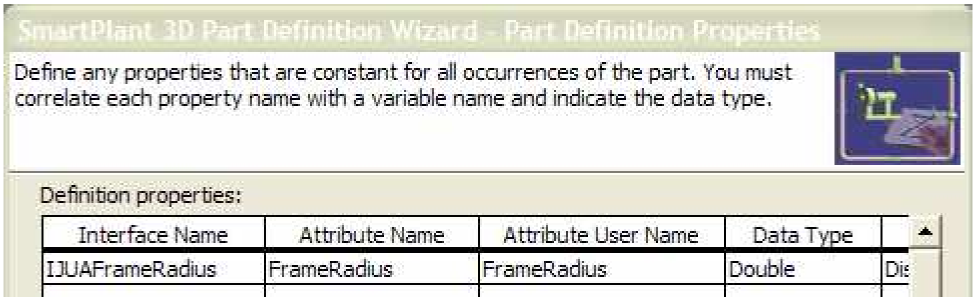
Intergraph common module location: C:\train\IngrModules

Custom common module location:

Save project as: C:\train\SP3D90VTrayOutside\SP3D90VTrayOutside.vbp

☐ Create bulkload spreadsheet

7. Select Next button to go the next page. This page is to define any properties that are constant for all occurrences of the cabletray part. Key in the following:

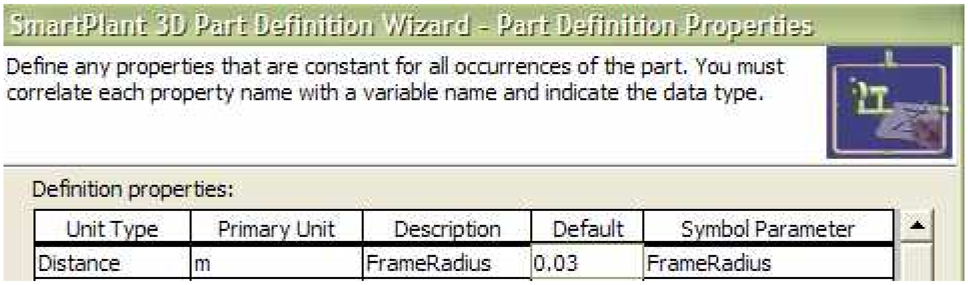


SmartPlant 3D Part Definition Wizard - Part Definition Properties

Define any properties that are constant for all occurrences of the part. You must correlate each property name with a variable name and indicate the data type.

Definition properties:

Interface Name	Attribute Name	Attribute User Name	Data Type	
IJUAFrameRadius	FrameRadius	FrameRadius	Double	Dis



SmartPlant 3D Part Definition Wizard - Part Definition Properties

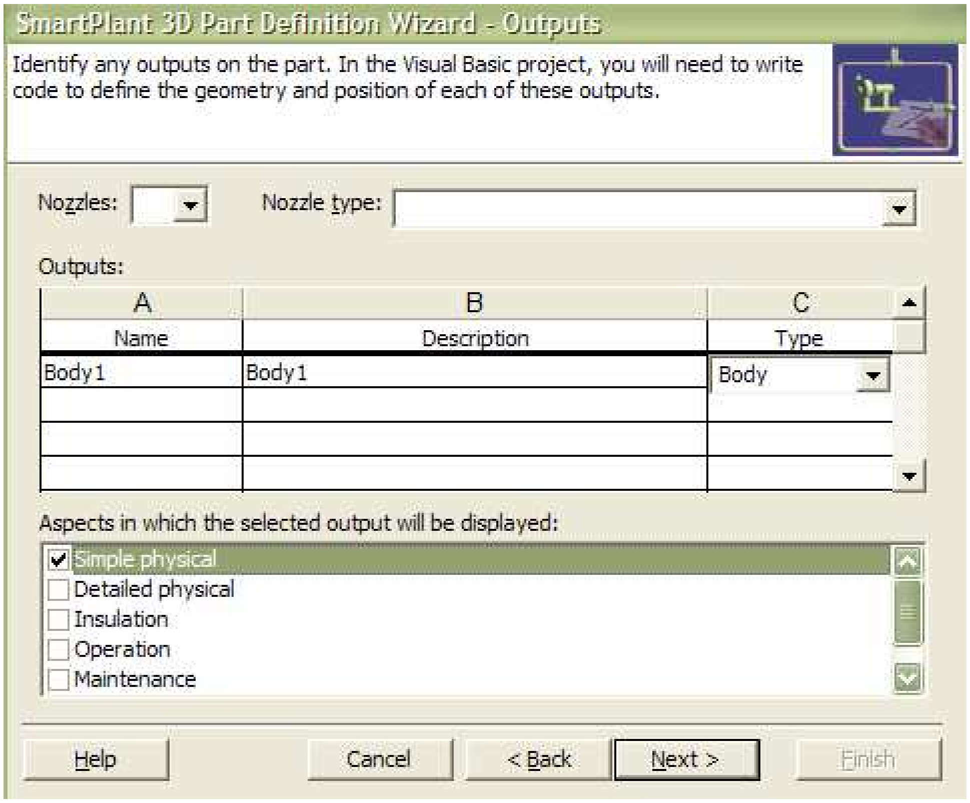
Define any properties that are constant for all occurrences of the part. You must correlate each property name with a variable name and indicate the data type.

Definition properties:

Unit Type	Primary Unit	Description	Default	Symbol Parameter
Distance	m	FrameRadius	0.03	FrameRadius

8. Select Next button to go the next page. This page defines all occurrence properties of the cabletray part. Select Next button to go the next page. This page identifies all the outputs of

the tray part. We are going to define 1 output. The Body output is in the Simple Physical aspect.



SmartPlant 3D Part Definition Wizard - Outputs

Identify any outputs on the part. In the Visual Basic project, you will need to write code to define the geometry and position of each of these outputs.

Nozzles:

Nozzle type:

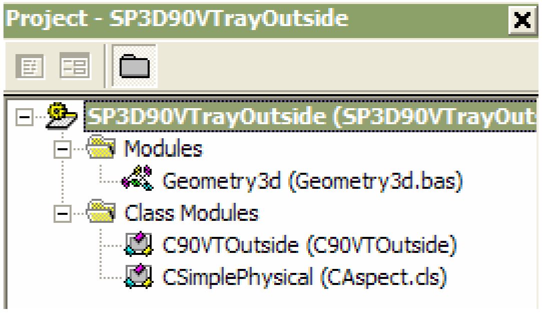
Outputs:

A Name	B Description	C Type
Body1	Body1	Body

Aspects in which the selected output will be displayed:

☑ Simple physical
□ Detailed physical
□ Insulation
□ Operation
□ Maintenance

9. Press Next button and Finish button to create the *SP3D90VTrayOutside* project template. The VB project consists of the following modules:



Project - SP3D90VTrayOutside

```
SP3D90VTrayOutside (SP3D90VTrayOut
  Modules
    Geometry3d (Geometry3d.bas)
  Class Modules
    C90VTOutside (C90VTOutside)
    CSimplePhysical (CAspect.cls)
```

10. Open the **CSP3D90VTrayOutside Class** module. This Class contains several routines.
11. Go to the Class_Initialize() routine. Review the inputs and outputs section. Add additional outputs as shown below:

```
Private Sub Class_Initialize()
   Const METHOD = "Class_Initialize:"
   On Error GoTo Errx
```

```
    Set m_oSymbolHelper = New SymbolServices
    m_oSymbolHelper.ProjectName = "SP3D90VTrayOutside"
    m_oSymbolHelper.ClassName = "C90VTOutside"

' Inputs

' Outputs
    m_oSymbolHelper.NumOutputs = 1
    m_oSymbolHelper.AddOutputDef 1, "Body1", "Body1", 1
    m_oSymbolHelper.AddOutputDef 2, "Body2", "Body2", 1
    m_oSymbolHelper.AddOutputDef 3, "Body3", "Body3", 1
    m_oSymbolHelper.AddOutputDef 4, "Body4", "Body4", 1
    m_oSymbolHelper.AddOutputDef 5, "Body5", "Body5", 1
    m_oSymbolHelper.AddOutputDef 6, "Body6", "Body6", 1
    m_oSymbolHelper.AddOutputDef 7, "port1", "port1", 1
    m_oSymbolHelper.AddOutputDef 8, "port2", "port2", 1

' Aspects
    m_oSymbolHelper.NumAspects = 1
    m_oSymbolHelper.AddAspectDef 1, "SimplePhysical", "SimplePhysical", 1

    Exit Sub
Errx:
    Err.Raise Err.Number, Err.Source & " " & METHOD, Err.Description, _
        Err.HelpFile, Err.HelpContext
End Sub
```

12. Go to CSimplePhysical Class module and declare all variables for your inputs and outputs

```
    Dim parActualWidth As Double
    Dim parActualDepth As Double
    Dim parBendRadius As Double
    Dim oPort1 As New AutoMath.DPosition
    Dim oPort2 As New AutoMath.DPosition
```

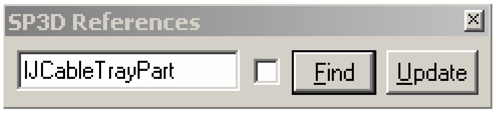
13. Go to **CSimplePhysical Class** module and add your code to create the outputs:
14. Go to the *"Insert your code for output (Body1)"* section. Use the RetrieveCableTrayPortProperties function to retrieve the port information from the part.

```
    Dim oTrayPart As IJCableTrayPart
    Set oTrayPart = oPartFclt
    parBendRadius = oTrayPart.BendRadius

  ' Insert your code for output 1
    Call RetrieveCableTrayPortProperties(1, oPartFclt, parActualWidth, parActualDepth)
    Dim CP As New AutoMath.DPosition 'arc center point
    Dim HalfDepth             As Double
    Dim HalfWidth             As Double

    Dim LineStrPoints(0 To 11)  As Double
    Dim Angle As Double
    Dim ProjVector        As New AutoMath.DVector

    Dim oLineString As IngrGeom3D.LineString3d
```

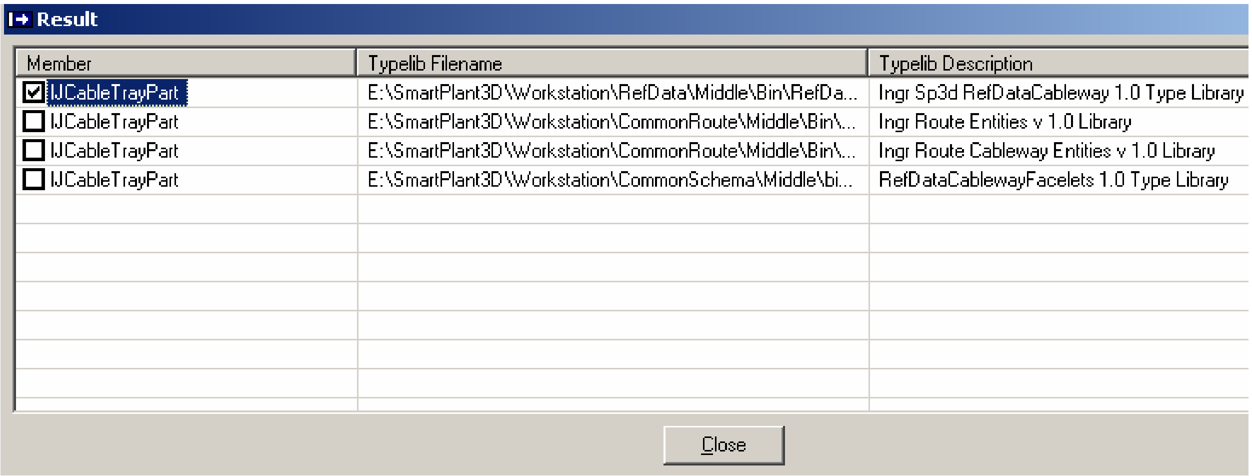
```
   Dim geomFactory     As IngrGeom3D.GeometryFactory
   Set geomFactory = New IngrGeom3D.GeometryFactory
   Angle = 2 * Atn(1)

   HalfDepth = parActualDepth / 2
   HalfWidth = parActualWidth / 2
   oPort1.Set 0, 0, -(parBendRadius + HalfDepth)
   oPort2.Set (parBendRadius + HalfDepth), 0, 0

   Dim LineStrCP As New AutoMath.DPosition
   LineStrCP.Set 0, 0, -(parBendRadius + HalfDepth)

   LineStrPoints(0) = LineStrCP.x - HalfDepth
   LineStrPoints(1) = LineStrCP.y - HalfWidth
   LineStrPoints(2) = LineStrCP.z

   LineStrPoints(3) = LineStrCP.x + HalfDepth
   LineStrPoints(4) = LineStrCP.y - HalfWidth
   LineStrPoints(5) = LineStrCP.z

   LineStrPoints(6) = LineStrCP.x + HalfDepth
   LineStrPoints(7) = LineStrCP.y + HalfWidth
   LineStrPoints(8) = LineStrCP.z

   LineStrPoints(9) = LineStrCP.x - HalfDepth
   LineStrPoints(10) = LineStrCP.y + HalfWidth
   LineStrPoints(11) = LineStrCP.z
   Set oLineString = geomFactory.LineStrings3d.CreateByPoints(Nothing, 4, LineStrPoints)
   ProjVector.Set 0, 1, 0
   CP.Set (parBendRadius + HalfDepth), 0, -(parBendRadius + HalfDepth)
   Set ObjBody1 = PlaceRevolution(m_OutputColl, oLineString, ProjVector, CP, Angle, False)
   Set oLineString = Nothing
' Set the output
   iOutput = iOutput + 1
   m_OutputColl.AddOutput arrayOfOutputs(iOutput), ObjBody1
   Set ObjBody1 = Nothing

'built the support

   Dim stpoint As IJDPosition
   Dim enpoint As IJDPosition

   Set stpoint = New DPosition
   Set enpoint = New DPosition

'support cylinder ----------
   stpoint.Set -HalfDepth, HalfWidth + parFrameRadius / 2, HalfDepth
   enpoint.Set -HalfDepth, -HalfWidth - parFrameRadius / 2, HalfDepth

   iOutput = iOutput + 1
   Set ObjBody1 = m_oGeomHelper.CreateCylinder(arrayOfOutputs(iOutput), stpoint, enpoint,
parFrameRadius)
   Set ObjBody1 = Nothing

'vertical cylinders----------
   stpoint.Set oPort1.x - HalfDepth, oPort1.y - HalfWidth, oPort1.z
   enpoint.Set -HalfDepth, -HalfWidth, HalfDepth

   iOutput = iOutput + 1
   Set ObjBody1 = m_oGeomHelper.CreateCylinder(arrayOfOutputs(iOutput), stpoint, enpoint,
parFrameRadius)
   Set ObjBody1 = Nothing

'-----------
   stpoint.Set oPort1.x - HalfDepth, oPort1.y + HalfWidth, oPort1.z
   enpoint.Set -HalfDepth, HalfWidth, HalfDepth

   iOutput = iOutput + 1
   Set ObjBody1 = m_oGeomHelper.CreateCylinder(arrayOfOutputs(iOutput), stpoint, enpoint,
parFrameRadius)
   Set ObjBody1 = Nothing

'horizontal cylinders-----------
   stpoint.Set oPort2.x, oPort2.y - HalfWidth, oPort2.z + HalfDepth
   enpoint.Set -HalfDepth, -HalfWidth, HalfDepth

   iOutput = iOutput + 1
   Set ObjBody1 = m_oGeomHelper.CreateCylinder(arrayOfOutputs(iOutput), stpoint, enpoint,
parFrameRadius)
   Set ObjBody1 = Nothing

'-----------
   stpoint.Set oPort2.x, oPort2.y + HalfWidth, oPort2.z + HalfDepth
   enpoint.Set -HalfDepth, HalfWidth, HalfDepth

   iOutput = iOutput + 1
   Set ObjBody1 = m_oGeomHelper.CreateCylinder(arrayOfOutputs(iOutput), stpoint, enpoint,
parFrameRadius)
   Set ObjBody1 = Nothing

   Set stpoint = Nothing
   Set enpoint = Nothing

' Place Nozzle 1
   Dim oDir        As AutoMath.DVector
   Dim oRadialOrient As AutoMath.DVector
   Dim objCableTrayPort   As GSCADNozzleEntities.IJCableTrayPortOcc

   Set oDir = New AutoMath.DVector
   Set oRadialOrient = New AutoMath.DVector

   oDir.Set 0, 0, -1
   oRadialOrient.Set -1, 0, 0
   Set objCableTrayPort = CreateCableTrayPort(oPartFclt, 1, oPort1, oDir, oRadialOrient, m_OutputColl)
' Set the output
   iOutput = iOutput + 1
   m_OutputColl.AddOutput arrayOfOutputs(iOutput), objCableTrayPort
   Set objCableTrayPort = Nothing
   Set oPort1 = Nothing
   Set oDir = Nothing
   Set oRadialOrient = Nothing
' Place Nozzle 2
```

```
    Set oDir = New AutoMath.DVector
    Set oRadialOrient = New AutoMath.DVector
    oDir.Set 1, 0, 0
    oRadialOrient.Set 0, 0, 1
    Set objCableTrayPort = CreateCableTrayPort(oPartFclt, 2, oPort2, oDir, oRadialOrient, m_OutputColl)
' Set the output
    iOutput = iOutput + 1
    m_OutputColl.AddOutput arrayOfOutputs(iOutput), objCableTrayPort
    Set objCableTrayPort = Nothing
    Set oPort2 = Nothing
    Set oDir = Nothing
    Set oRadialOrient = Nothing
```

15. Use the SP3D reference tool to find the library that reference IJCabletrayPart

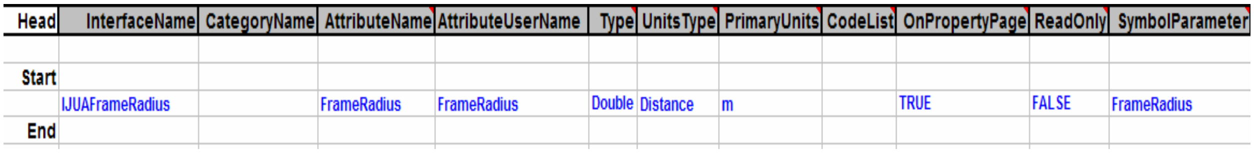


16. Select Ingr Sp3d RefDataCableway 1.0 Type Library.



Member	Typelib Filename	Typelib Description
☑ IJCableTrayPart	E:\SmartPlant3D\Workstation\RefData\Middle\Bin\RefDa...	Ingr Sp3d RefDataCableway 1.0 Type Library
□ IJCableTrayPart	E:\SmartPlant3D\Workstation\CommonRoute\Middle\Bin\...	Ingr Route Entities v 1.0 Library
□ IJCableTrayPart	E:\SmartPlant3D\Workstation\CommonRoute\Middle\Bin\...	Ingr Route Cableway Entities v 1.0 Library
□ IJCableTrayPart	E:\SmartPlant3D\Workstation\CommonSchema\Middle\bi...	RefDataCablewayFacelets 1.0 Type Library

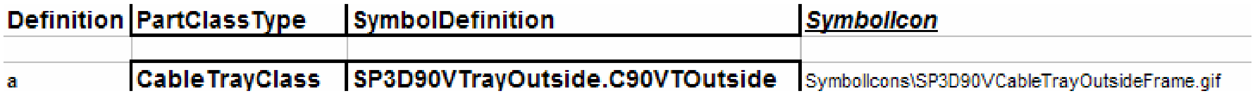
17. Compile the VB project and save the dll in the c:\train\ *SP3D90VTrayOutside*
18. Save the VB *SP3D90VTrayOutside* project.
19. Open the [Install Product]\ CatalogData\BulkLoad\Datafiles\CableTray.xls. Make sure to remove the Read-Only setting on the file.
20. Open the Cabletray.xls workbook. Go to the Custom Interface sheet and edit/add the following entries:



Head	InterfaceName	CategoryName	AttributeName	AttributeUserName	Type	UnitsType	PrimaryUnits	CodeList	OnPropertyPage	ReadOnly	SymbolParameter
Start											
	IJUAFrameRadius		FrameRadius	FrameRadius	Double	Distance	m		TRUE	FALSE	FrameRadius
End											

21. Create the CT90VOBendFrame Part Class ass follows:

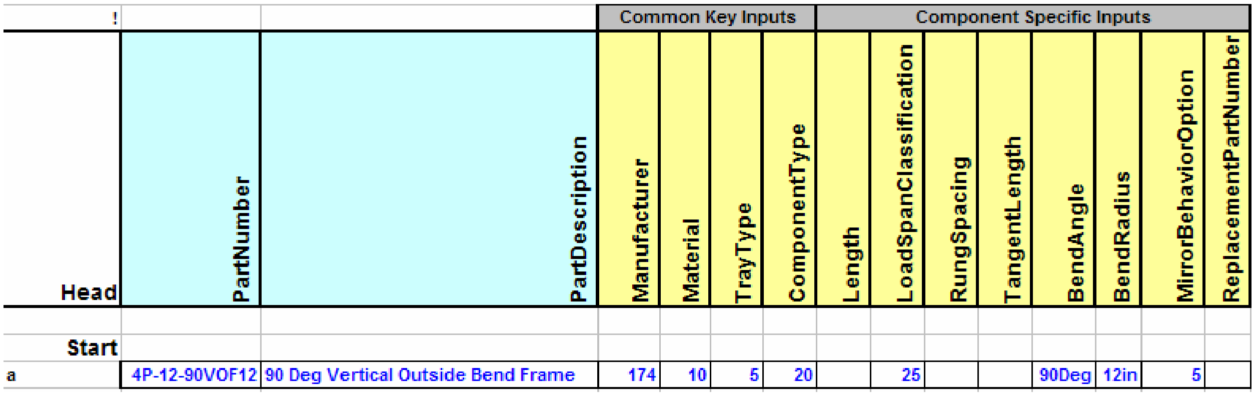
In the Definition Section:



Definition	PartClassType	SymbolDefinition	SymbolIcon
a	CableTrayClass	SP3D90VTrayOutside.C90VTOutside	SymbolIcons\SP3D90VCableTrayOutsideFrame.gif

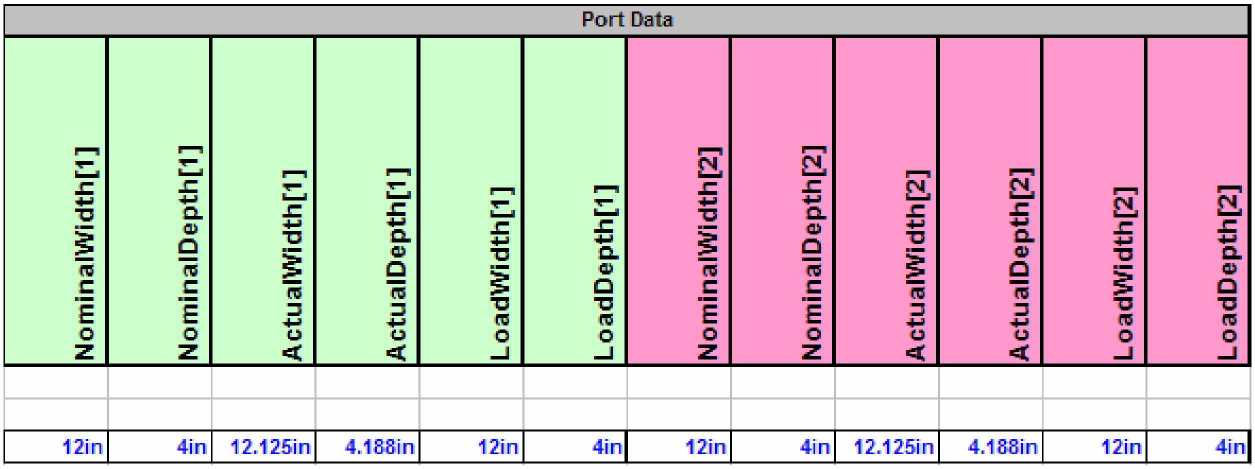
In the Part Section:

System attributes:



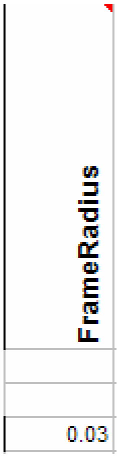
!			Common Key Inputs				Component Specific Inputs							
Head	PartNumber	PartDescription	Manufacturer	Material	TrayType	ComponentType	Length	LoadSpanClassification	RungSpacing	TangentLength	BendAngle	BendRadius	MirrorBehaviorOption	ReplacementPartNumber
Start														
a	4P-12-90VOF12	90 Deg Vertical Outside Bend Frame	174	10	5	20		25			90Deg	12in	5	

Port Information:



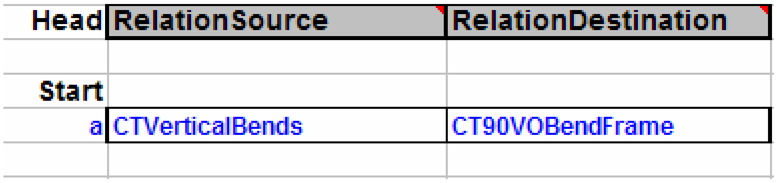
Port Data											
NominalWidth[1]	NominalDepth[1]	ActualWidth[1]	ActualDepth[1]	LoadWidth[1]	LoadDepth[1]	NominalWidth[2]	NominalDepth[2]	ActualWidth[2]	ActualDepth[2]	LoadWidth[2]	LoadDepth[2]
12in	4in	12.125in	4.188in	12in	4in	12in	4in	12.125in	4.188in	12in	4in

Dimensions:



FrameRadius
0.03

Go to R-ClassNodeDescribes sheet and add the following entry:



Head	RelationSource	RelationDestination
Start		
a	CTVerticalBends	CT90VOBendFrame

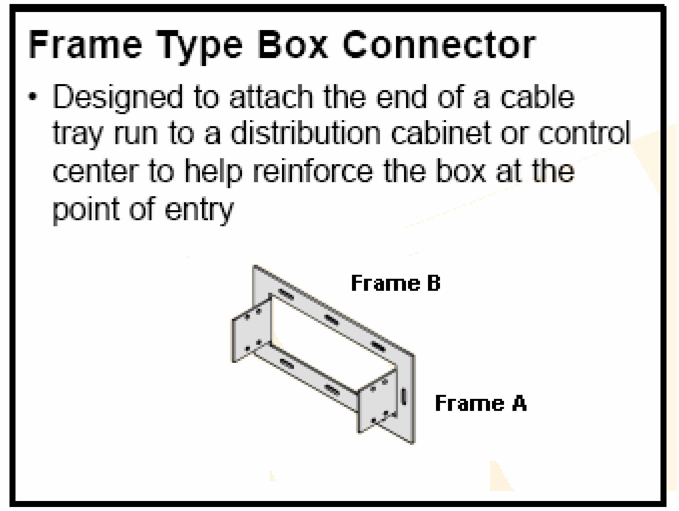
22. Load the information into the Catalog using the Append Mode. Once the bulkload process is complete, review the log file. Next, run the View Generator utility on the model to re-create the views in the model database. Finally, Re-generate the report databases.
23. Go to the Electrical Task and place the 90 deg vertical outside cabletray bend using CB-S1-L6-12B spec.

Lab 19: Electrical Box Connector Symbol (Optional)

Objective

After completing this lab, you will be able to:

- Create a Frame Box Connector using the SmartPlant 3D Part Definition VB Wizard



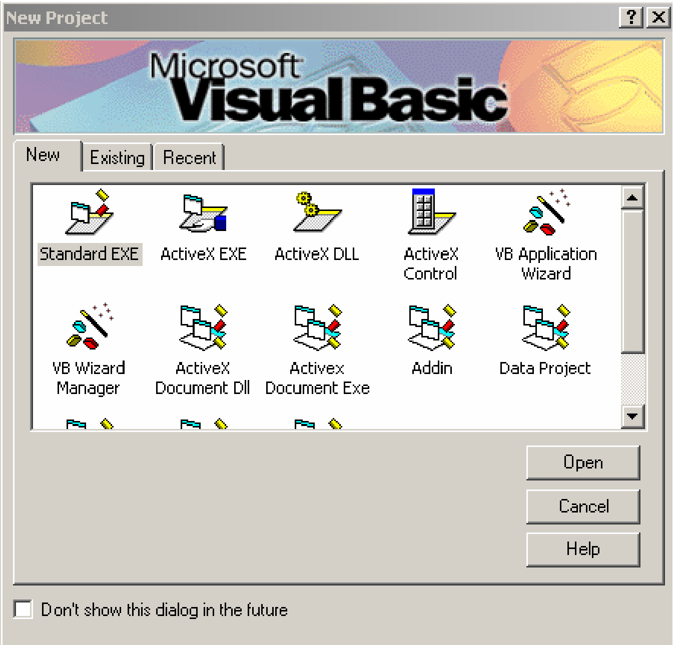
Frame Type Box Connector

- Designed to attach the end of a cable tray run to a distribution cabinet or control center to help reinforce the box at the point of entry

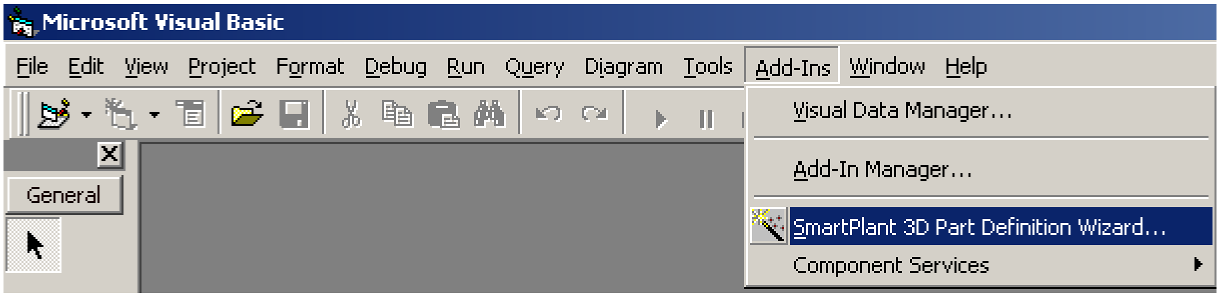
1. Create the following directory:

   c:\train\SP3DFrameBox

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



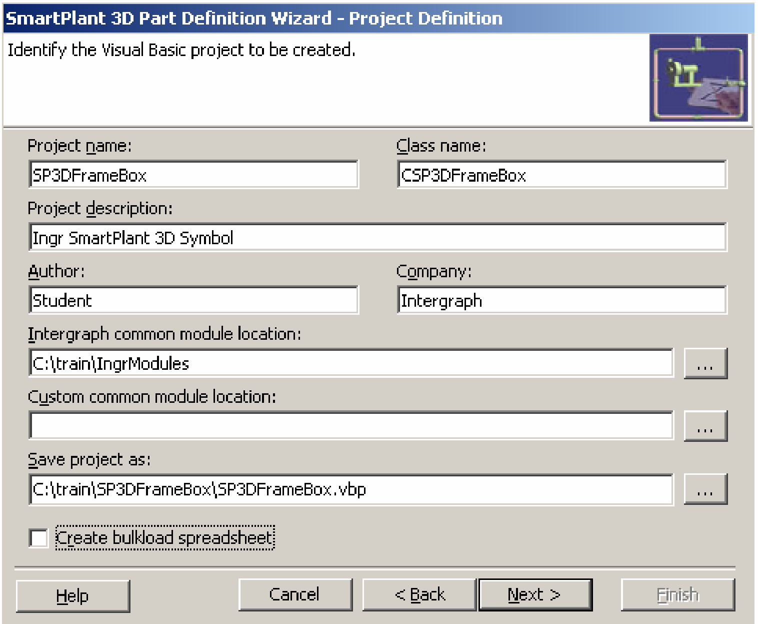
3. Go to the Add-Ins Option and Select SmartPlant 3D Part Definition Wizard.


Microsoft Visual Basic
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
General
Visual Data Manager...
Add-In Manager...
SmartPlant 3D Part Definition Wizard...
Component Services

4. Select Next button to skip the Introduction page. The Next step is to create the SP3DFrameBox symbol definition template using SP3D part Definition VB Symbol Wizard.

5. In this page you define the VB project name. Key in the following information:

   Project Name: SP3DFrameBox
   Author: Student
   Company: Intergraph
   Intergraph Module location: c:\Train\IngrModules
   Save the VB project as: c:\Train\SP3DFrameBox



SmartPlant 3D Part Definition Wizard - Project Definition

Identify the Visual Basic project to be created.

Project name: SP3DFrameBox

Class name: CSP3DFrameBox

Project description: Ingr SmartPlant 3D Symbol

Author: Student

Company: Intergraph

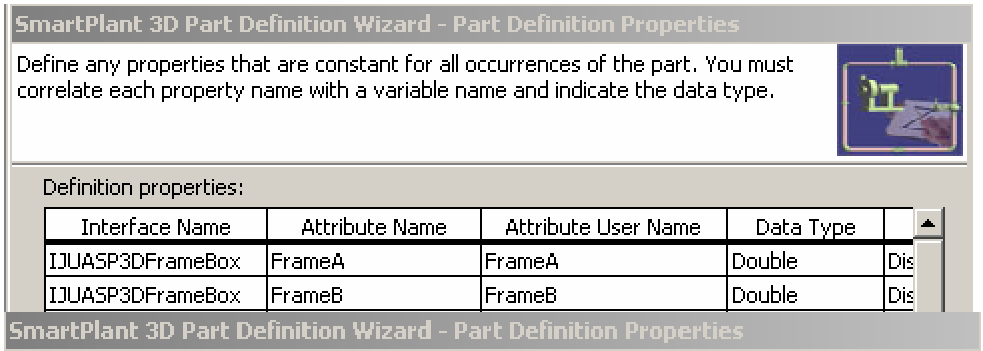
Intergraph common module location: C:\train\IngrModules

Custom common module location:

Save project as: C:\train\SP3DFrameBox\SP3DFrameBox.vbp

☐ Create bulkload spreadsheet

6. Select Next button to go the next page. This page is to define any input properties that are defined in the part class that are constant for all occurrences. We are going to define two attributes for our SP3DFrameBox. Key in the following data:

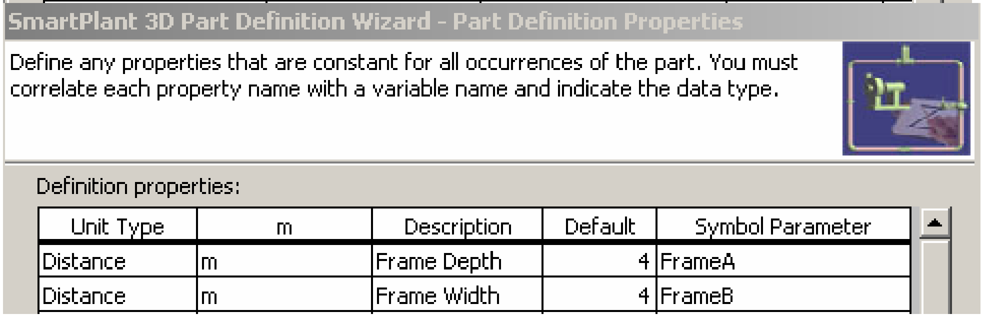


SmartPlant 3D Part Definition Wizard - Part Definition Properties

Define any properties that are constant for all occurrences of the part. You must correlate each property name with a variable name and indicate the data type.

Definition properties:

Interface Name	Attribute Name	Attribute User Name	Data Type	
IJUASP3DFrameBox	FrameA	FrameA	Double	Dis
IJUASP3DFrameBox	FrameB	FrameB	Double	Dis

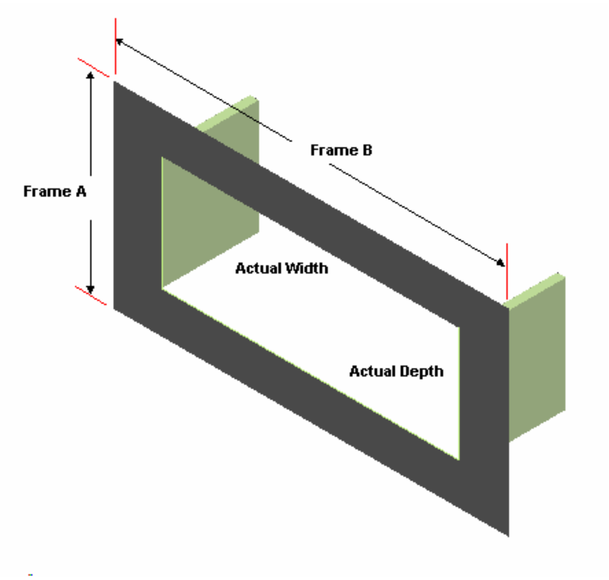


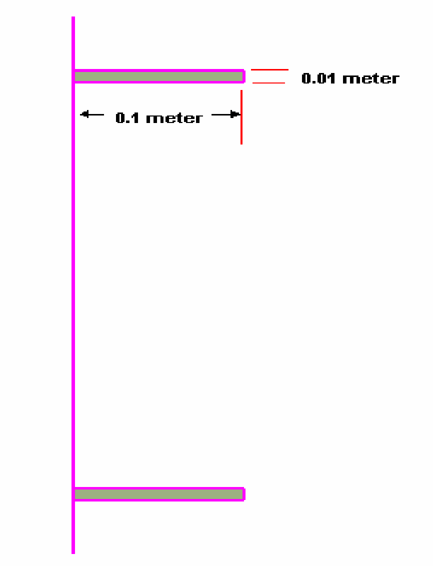
SmartPlant 3D Part Definition Wizard - Part Definition Properties

Define any properties that are constant for all occurrences of the part. You must correlate each property name with a variable name and indicate the data type.

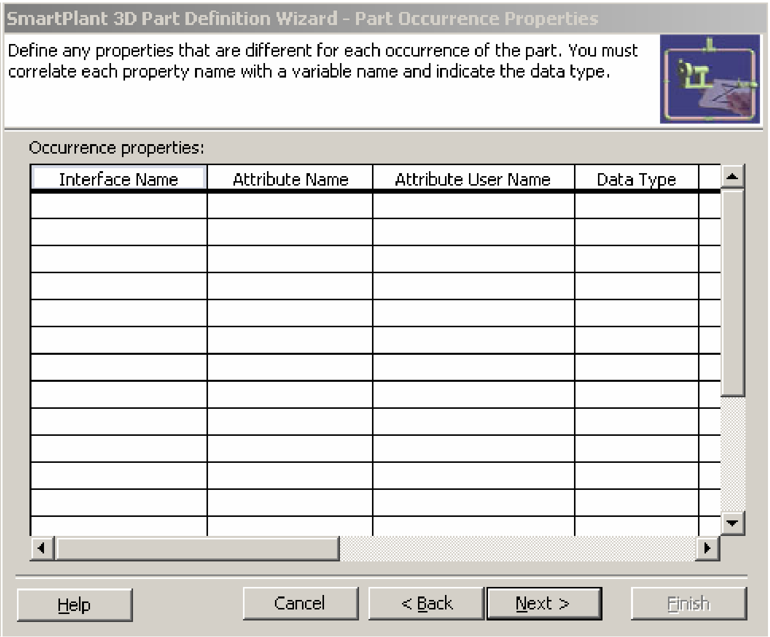
Definition properties:

Unit Type	m	Description	Default	Symbol Parameter
Distance	m	Frame Depth	4	FrameA
Distance	m	Frame Width	4	FrameB





7. Select Next button to go the next page. Skip this page because our cabletray part does not have occurrence attributes.



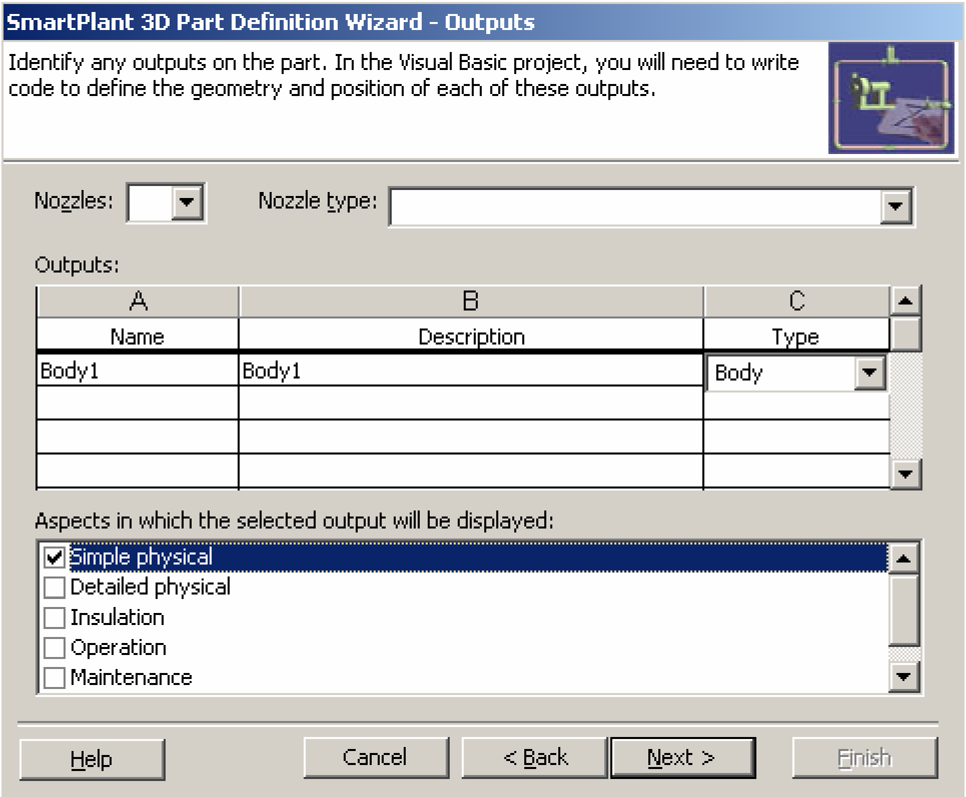
SmartPlant 3D Part Definition Wizard - Part Occurrence Properties

Define any properties that are different for each occurrence of the part. You must correlate each property name with a variable name and indicate the data type.

Occurrence properties:

Interface Name	Attribute Name	Attribute User Name	Data Type

8. Select Next button to go the next page. This page identifies all the outputs of the cabletray part. We are going to define one output and one port for our SP3DFrameBox. The output is in the simple Physical aspect.



SmartPlant 3D Part Definition Wizard - Outputs

Identify any outputs on the part. In the Visual Basic project, you will need to write code to define the geometry and position of each of these outputs.

Nozzles:

Nozzle type:

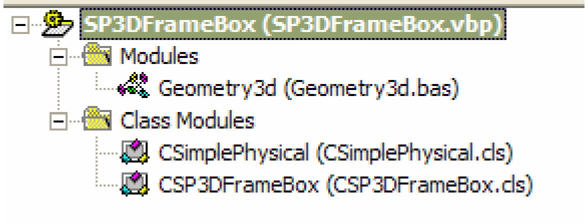
Outputs:

A	B	C
Name	Description	Type
Body1	Body1	Body

Aspects in which the selected output will be displayed:

☑ Simple physical
□ Detailed physical
□ Insulation
□ Operation
□ Maintenance

9. Hit Next button and Finish button to create the SP3DFrameBox project template. The VB project consists of the following modules:



SP3DFrameBox (SP3DFrameBox.vbp)
- Modules
  - Geometry3d (Geometry3d.bas)
- Class Modules
  - CSimplePhysical (CSimplePhysical.cls)
  - CSP3DFrameBox (CSP3DFrameBox.cls)

10. Open the **CSP3DFrameBox Class** module. This Class contains several routines.
11. Go to the Class_Initialize() routine in the input section. Review the inputs and add one output definition for the cabletray port as shown below.

```
Private Sub Class_Initialize()
    Const METHOD = "Class_Initialize:"
    On Error GoTo Errx

    Set m_oSymbolHelper = New SymbolServices
    m_oSymbolHelper.ProjectName = "SP3DFrameBox"
    m_oSymbolHelper.ClassName = "CSP3DFrameBox"

' Inputs
```

```
    m_oSymbolHelper.NumInputs = 2
    m_oSymbolHelper.AddInputDef 1, "FrameA", "Frame Depth", 4
    m_oSymbolHelper.AddInputDef 2, "FrameB", "Frame Width", 4

' Outputs
    m_oSymbolHelper.NumOutputs = 2
    m_oSymbolHelper.AddOutputDef 1, "Body1", "Body1", 1
    m_oSymbolHelper.AddOutputDef 2, "port1", "Port1", 1

' Aspects
    m_oSymbolHelper.NumAspects = 1
    m_oSymbolHelper.AddAspectDef 1, "SimplePhysical", "SimplePhysical", 1

    Exit Sub
Errx:
    Err.Raise Err.Number, Err.Source & " " & METHOD, Err.Description, _
        Err.HelpFile, Err.HelpContext
End Sub
```

12. Go to **CSimplePhysical Class** module and add your code to create the outputs:
13. Go to the Insert your code for output (Body1) section. The following code will use the 3D geometry factory to create a frame. Use the 3D geometry factory to create a 3D plane using the Frame A and Frame B dimensions.

```
    Dim oTrayPart As IJCableTrayPart
    Set oTrayPart = oPartFclt

    Dim Points(0 To 11) As Double
    Dim geomFactory As New IngrGeom3D.GeometryFactory
    Dim ObjBody1 As IngrGeom3D.Plane3d

    Points(0) = 0
    Points(1) = parFrameB / 2
    Points(2) = parFrameA / 2
    Points(3) = 0
    Points(4) = -parFrameB / 2
    Points(5) = parFrameA / 2
    Points(6) = 0
    Points(7) = -parFrameB / 2
    Points(8) = -parFrameA / 2
    Points(9) = 0
    Points(10) = parFrameB / 2
    Points(11) = -parFrameA / 2
    Set ObjBody1 = geomFactory.Planes3d.CreateByPoints(m_OutputColl.ResourceManager, 4, Points)

' Set the output
    iOutput = iOutput + 1
    m_OutputColl.AddOutput arrayOfOutputs(iOutput), ObjBody1
    Set ObjBody1 = Nothing
```

Note: Go to the declaration section and delete this statement Dim ObjBody1 As Object

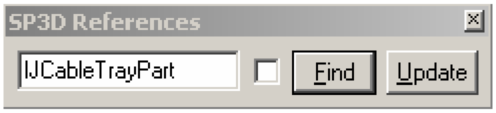
14. The following code will use the CreateCableTrayPort() method to create the cabletray port. The CreateCableTrayPort() routine is defined in the Geometry3d module.

```
    Dim oDir  As AutoMath.DVector
    Dim oRadialOrient As AutoMath.DVector
    Dim objCableTrayPort   As GSCADNozzleEntities.IJCableTrayPortOcc
    Set oDir = New AutoMath.DVector
    Set oRadialOrient = New AutoMath.DVector

    oDir.Set -1, 0, 0
    oRadialOrient.Set 0, 0, 1
    Set objCableTrayPort = CreateCableTrayPort(oPartFclt, 1, CenterPos, oDir, oRadialOrient,
m_OutputColl)

' Set the output
    iOutput = iOutput + 1
    m_OutputColl.AddOutput arrayOfOutputs(iOutput), objCableTrayPort
```

15. Use the Set statement to clear the references from all object variables.

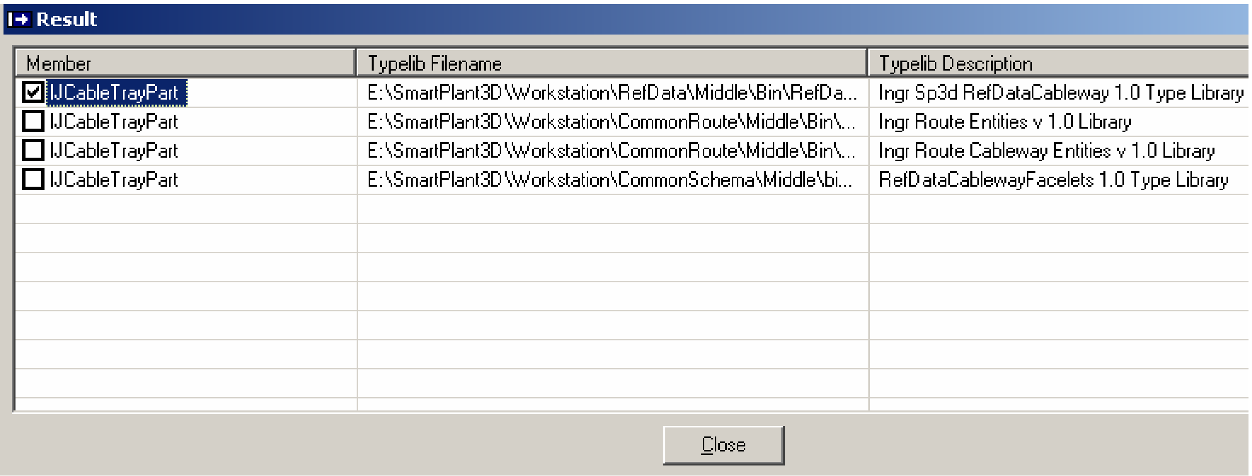
```
    Set objCableTrayPort = Nothing
    Set CenterPos = Nothing
    Set oDir = Nothing
    Set oRadialOrient = Nothing
    Set geomFactory = Nothing
```

16. Use the SP3D reference tool to find the library that reference IJCabletrayPart



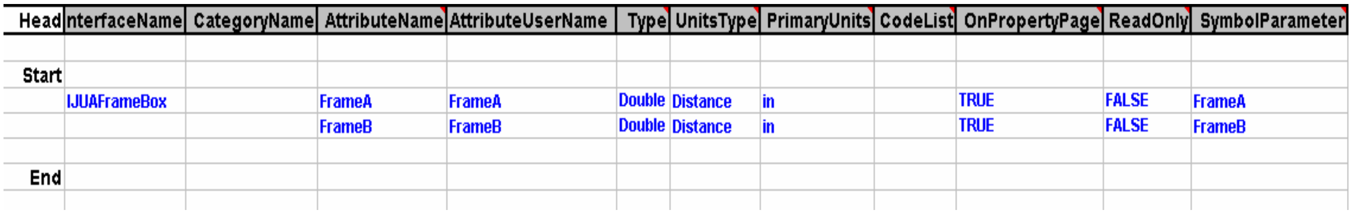
IJCableTrayPart

17. Select Ingr Sp3d RefDataCableway 1.0 Type Library.



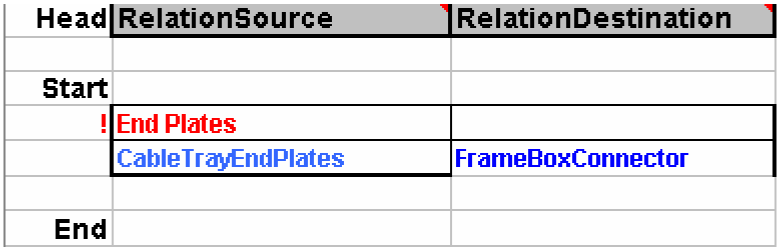
Member	Typelib Filename	Typelib Description
☑ IJCableTrayPart	E:\SmartPlant3D\Workstation\RefData\Middle\Bin\RefDa...	Ingr Sp3d RefDataCableway 1.0 Type Library
□ IJCableTrayPart	E:\SmartPlant3D\Workstation\CommonRoute\Middle\Bin\...	Ingr Route Entities v 1.0 Library
□ IJCableTrayPart	E:\SmartPlant3D\Workstation\CommonRoute\Middle\Bin\...	Ingr Route Cableway Entities v 1.0 Library
□ IJCableTrayPart	E:\SmartPlant3D\Workstation\CommonSchema\Middle\bi...	RefDataCablewayFacelets 1.0 Type Library

18. Hit close button. Compile the VB project and save the dll in the c:\train\ *SP3DFrameBox*
19. Save the VB SP3DFrameBox project.
20. Open the Cabletray.xls workbook. Go to the Custom Interface sheet and edit/add the following entries:



Head	nterfaceName	CategoryName	AttributeName	AttributeUserName	Type	UnitsType	PrimaryUnits	CodeList	OnPropertyPage	ReadOnly	SymbolParameter
Start											
	IJUAFrameBox		FrameA	FrameA	Double	Distance	in		TRUE	FALSE	FrameA
			FrameB	FrameB	Double	Distance	in		TRUE	FALSE	FrameB
End											

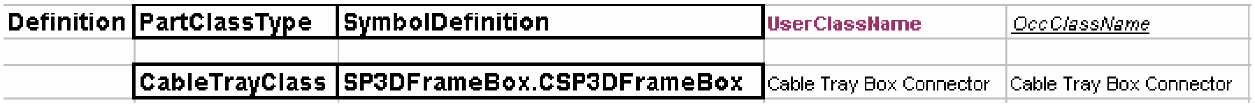
21. Go the R-ClassNodeDescribes sheet and add the following entry.



Head	RelationSource	RelationDestination
Start		
!	End Plates	
	CableTrayEndPlates	FrameBoxConnector
End		

22. Create the FrameBoxConnector Part Class ass follows:

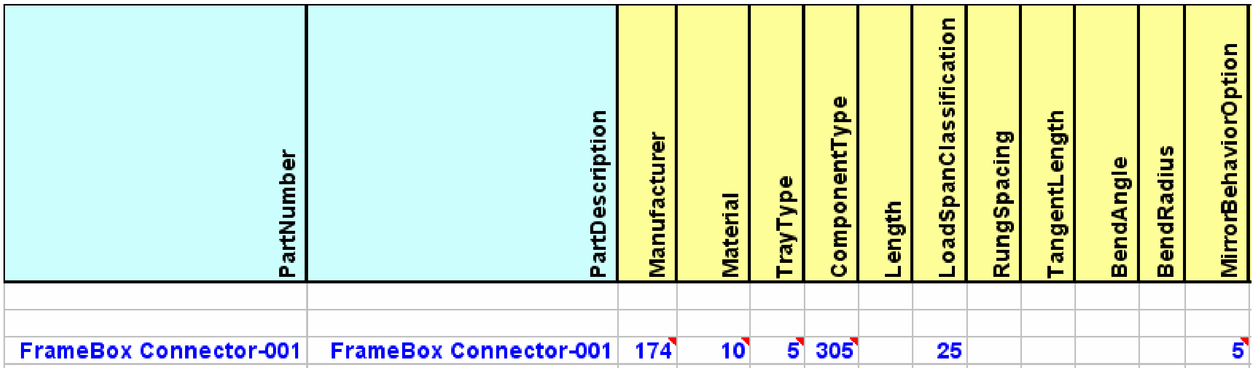
In the Definition Section:



Definition	PartClassType	SymbolDefinition	UserClassName	OccClassName
	CableTrayClass	SP3DFrameBox.CSP3DFrameBox	Cable Tray Box Connector	Cable Tray Box Connector

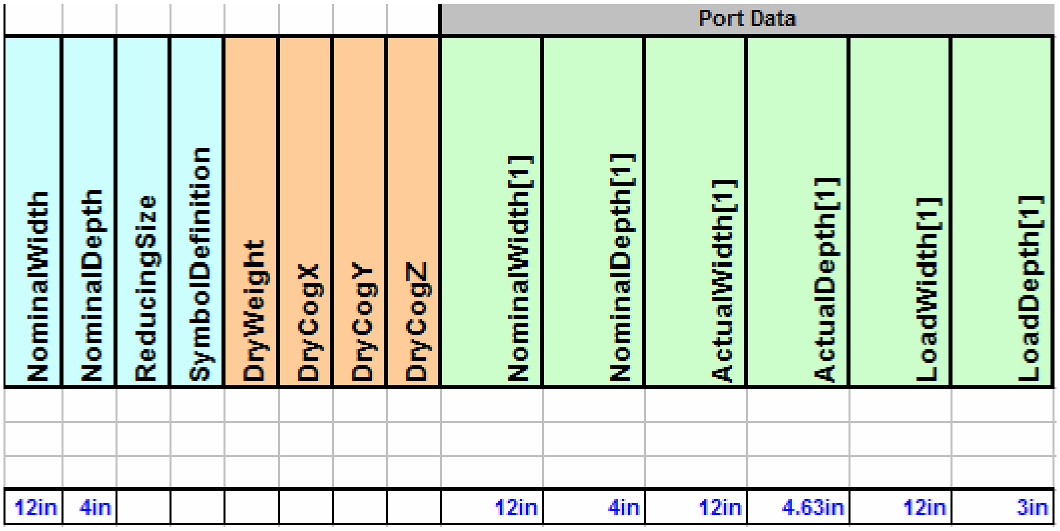
In the Part Section:

System attributes:



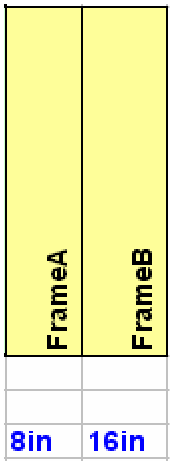
PartNumber	PartDescription	Manufacturer	Material	TrayType	ComponentType	Length	LoadSpanClassification	RungSpacing	TangentLength	BendAngle	BendRadius	MirrorBehaviorOption
FrameBox Connector-001	FrameBox Connector-001	174	10	5	305		25					5

Port Information:



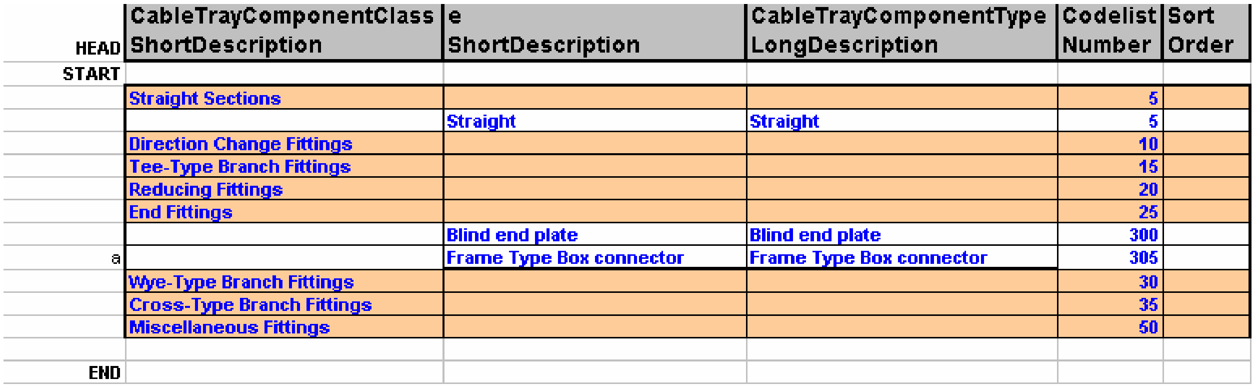
								Port Data					
NominalWidth	NominalDepth	ReducingSize	SymbolDefinition	DryWeight	DryCogX	DryCogY	DryCogZ	NominalWidth[1]	NominalDepth[1]	ActualWidth[1]	ActualDepth[1]	LoadWidth[1]	LoadDepth[1]
12in	4in							12in	4in	12in	4.63in	12in	3in

Dimensions:



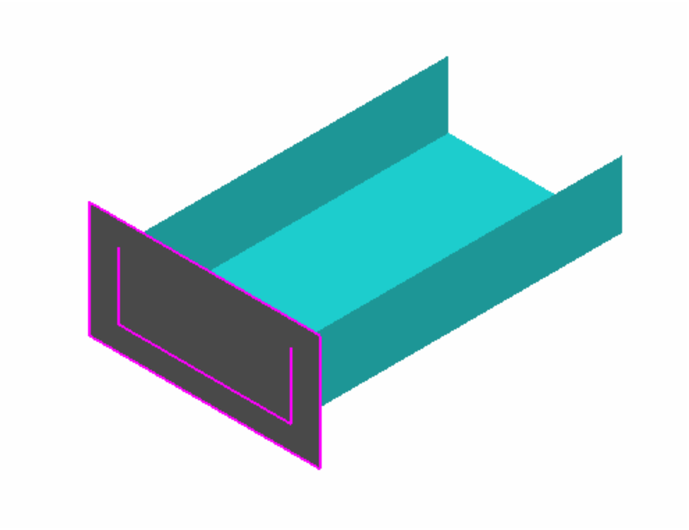
FrameA	FrameB
8in	16in

23. Open the AllCodeList.xls Excel Workbook.
24. Go to CableTrayComponentType sheet
25. Add a Frame Type Box connector (305) in the End Fitting Cable tray Component Class section as follows:



HEAD	CableTrayComponentClass ShortDescription	e ShortDescription	CableTrayComponentType LongDescription	Codelist Number	Sort Order
START					
	Straight Sections			5	
		Straight	Straight	5	
	Direction Change Fittings			10	
	Tee-Type Branch Fittings			15	
	Reducing Fittings			20	
	End Fittings			25	
		Blind end plate	Blind end plate	300	
a		Frame Type Box connector	Frame Type Box connector	305	
	Wye-Type Branch Fittings			30	
	Cross-Type Branch Fittings			35	
	Miscellaneous Fittings			50	
END					

26. Load the information into the Catalog using the Append Mode. Once the bulkload process is complete, review the log file. Next, run the View Generator utility on the model to re-create the views in the model database. Finally, Re-generate the report databases.
27. Go to the Electrical Task and place the Frame Box Connector.



Lab 20: Electrical Box Connector - Symbol Modification (Optional)

Modify the Frame Box connector symbol (SP3DFrameBox) by adding two plates and the hole.

1. Open the SP3DFrameBox.vb program and add the following entries in the output section:

```
' Outputs
   m_oSymbolHelper.NumOutputs = 4
   m_oSymbolHelper.AddOutputDef 1, "Body1", "Body1", 1
   m_oSymbolHelper.AddOutputDef 2, "Body2", "Body2", 1
   m_oSymbolHelper.AddOutputDef 3, "Body3", "Body3", 1
   m_oSymbolHelper.AddOutputDef 4, "port", "port", 1
```

2. Go to CSimplePhysical Class module and add your code to create the hole:

```
'------------------------------------
' Create the hole boundaries
'------------------------------------

   Dim parActualWidth As Double
   Dim parActualDepth As Double
   Dim HD As Double
   Dim HW As Double

   Call RetrieveCableTrayPortProperties(1, oPartFclt, parActualWidth, parActualDepth)
   HD = parActualDepth / 2
   HW = parActualWidth / 2

   Dim thickness1 As Double
   Dim thickness2 As Double
   thickness1 = (parFrameB - parActualWidth) / 2
   thickness2 = (parFrameA - parActualDepth) / 2

   Dim STPoint As IJDPosition
   Set STPoint = New DPosition
   STPoint.Set 0, Points(1) - thickness1, Points(2) - thickness2
   Dim lines As Collection
   Dim oline As IngrGeom3D.Line3d

   Set lines = New Collection
   Set oline = geomFactory.Lines3d.CreateBy2Points(Nothing, _
    0, Points(1) - thickness1, Points(2) - thickness2, _
    0, Points(4) + thickness1, Points(5) - thickness2)
   lines.Add oline

   Set oline = geomFactory.Lines3d.CreateBy2Points(Nothing, _
    0, Points(4) + thickness1, Points(5) - thickness2, _
    0, Points(7) + thickness1, Points(8) + thickness2)
   lines.Add oline

   Set oline = geomFactory.Lines3d.CreateBy2Points(Nothing, _
    0, Points(7) + thickness1, Points(8) + thickness2, _
```

```
     0, Points(10) - thickness1, Points(11) + thickness2)
    lines.Add oline

    Set oline = geomFactory.Lines3d.CreateBy2Points(Nothing, _
     0, Points(10) - thickness1, Points(11) + thickness2, _
     0, Points(1) - thickness1, Points(2) - thickness2)
    lines.Add oline

    Dim oContour As IngrGeom3D.ComplexString3d
    Set oContour = PlaceTrCString(STPoint, lines)
    ObjBody1.AddHole oContour

    Set oline = Nothing
    Dim iCount As Integer
    For iCount = 1 To lines.Count
        lines.Remove 1
    Next iCount
    Set lines = Nothing
    Set oContour = Nothing
    Set STPoint = Nothing

' Set the output
    iOutput = iOutput + 1
    m_OutputColl.AddOutput arrayOfOutputs(iOutput), ObjBody1
    Set ObjBody1 = Nothing
```

3. Add your code to create the right plate using the PlaceBox() routine.

```
'--- Right plate1
'           ^
'           |         |----| HD
'   +y      |   -HW   |    |
'    <------|-------->|____|-HD
'                  0.001  0.01
'
   Dim pPos1   As IJDPosition
   Dim pPos2   As IJDPosition
   Set pPos1 = New DPosition
   Set pPos2 = New DPosition

   Dim ObjBody2 As Object
   pPos1.Set -0.1, -HW - 0.01, -HD
   pPos2.Set 0, -HW - 0.001, HD
   Set ObjBody2 = PlaceBox(m_OutputColl, pPos1, pPos2)
   iOutput = iOutput + 1
   m_OutputColl.AddOutput arrayOfOutputs(iOutput), ObjBody2
```

4. Add code to create the left plate using the PlaceBox() routine.

```
'--- Left plate2
'                   ^
'HD |----|          |
'   |    |   HW +y  |  -y
'-HD|____| <-------------->
' 0.01  0.001
```

```
pPos1.Set -0.1, HW + 0.01, -HD
pPos2.Set 0, HW + 0.001, HD
Set ObjBody2 = PlaceBox(m_OutputColl, pPos1, pPos2)
iOutput = iOutput + 1
m_OutputColl.AddOutput arrayOfOutputs(iOutput), ObjBody2
Set ObjBody2 = Nothing
```

5. Use the Set statement to clear the references from all object variables.

   ```
   Set pPos1 = Nothing
   Set pPos2 = Nothing
   ```

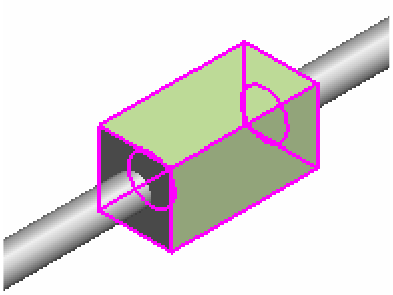
6. Go to Project->Properties to open the Project Properties Dialog box.
7. Go to the Make Tab and increase the major version number.
8. Compile the VB project and save the dll in the c:\\train\ *SP3DFrameBox*
9. Save the VB SP3DFrameBox project.
10. Open the Cabletray.xls
11. Add the letter M on the Part Class Definition and on the Part.
12. Load the information into the Catalog using the Modify Mode. Once the bulkload process is complete, review the log file. Next, synchronize the model with the catalog databases. Finally, Re-generate the report databases.
13. Go to the Electrical Task and review the Frame Box connector.

Lab 21: Electrical Junction Box Symbol (Optional)

Objective

After completing this lab, you will be able to:

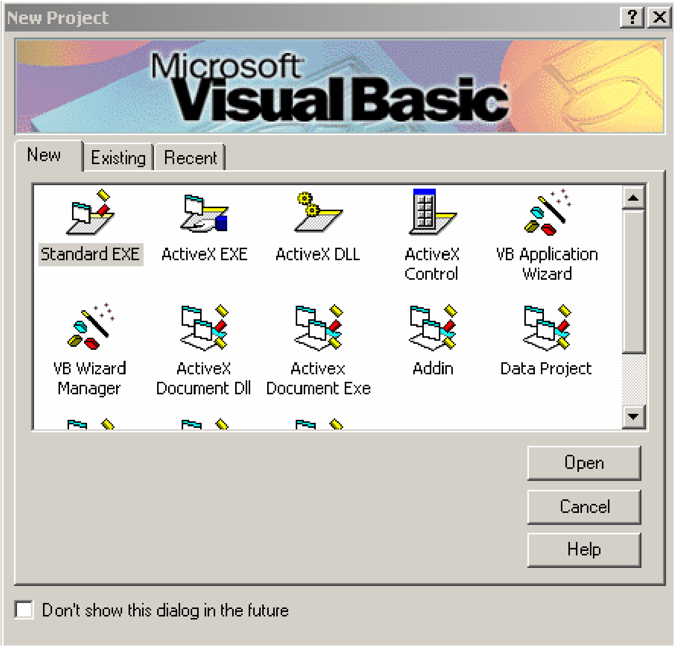
- Create a Junction Box using the SmartPlant 3D Part Definition VB Wizard



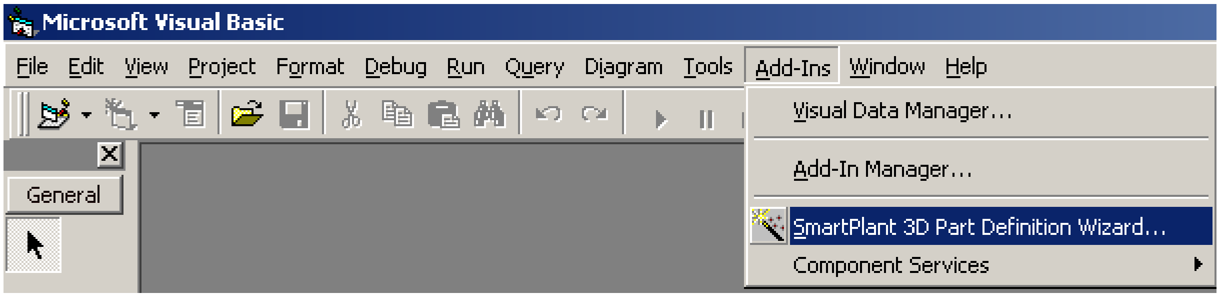
1. Create the following directory:

   c:\train\ SP3DJunctionBox

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



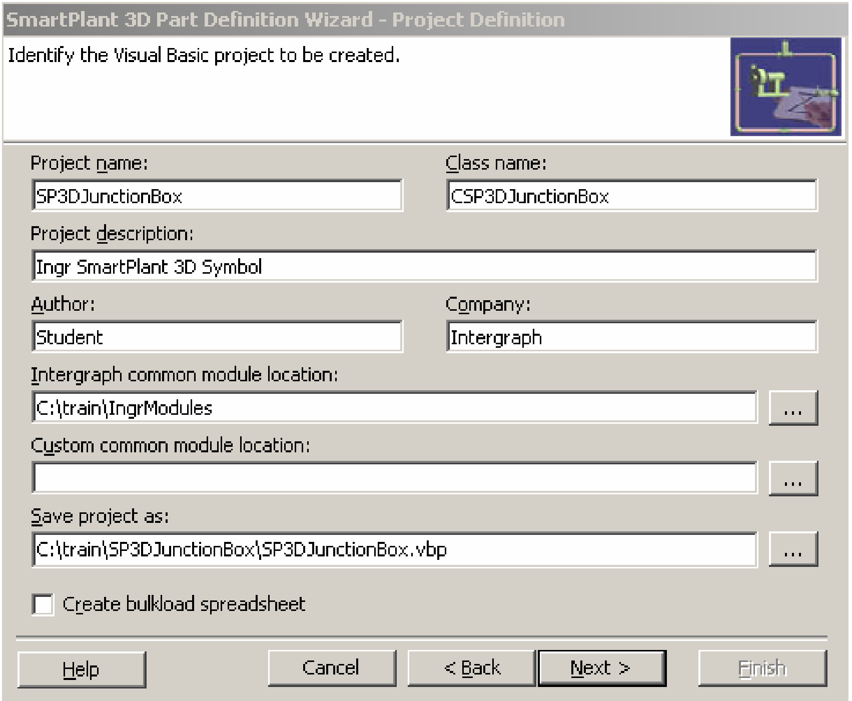
3. Go to the Add-Ins Option and Select SmartPlant 3D Part Definition Wizard.


Microsoft Visual Basic
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
General
Visual Data Manager...
Add-In Manager...
SmartPlant 3D Part Definition Wizard...
Component Services

4. Select Next button to skip the Introduction page. The Next step is to create the SP3DJunctionBox symbol definition template using SP3D part Definition VB Symbol Wizard.

5. In this page you define the VB project name. Key in the following information:

   Project Name: SP3DJunctionBox
   Author: Student
   Company: Intergraph
   Intergraph Module location: c:\Train\IngrModules
   Save the VB project as: c:\Train\ SP3DJunctionBox

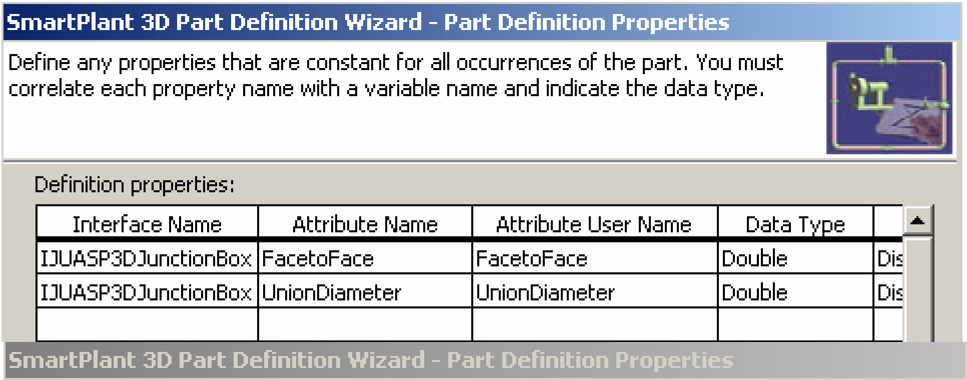

SmartPlant 3D Part Definition Wizard - Project Definition

Identify the Visual Basic project to be created.

Project name: SP3DJunctionBox
Class name: CSP3DJunctionBox
Project description: Ingr SmartPlant 3D Symbol
Author: Student
Company: Intergraph
Intergraph common module location: C:\train\IngrModules
Custom common module location:
Save project as: C:\train\SP3DJunctionBox\SP3DJunctionBox.vbp

☐ Create bulkload spreadsheet

6. Select Next button to go the next page. This page is to define any input properties that are defined in the part class that are constant for all occurrences. We are going to define two attributes for our SP3DJunctionBox. Key in the following data:

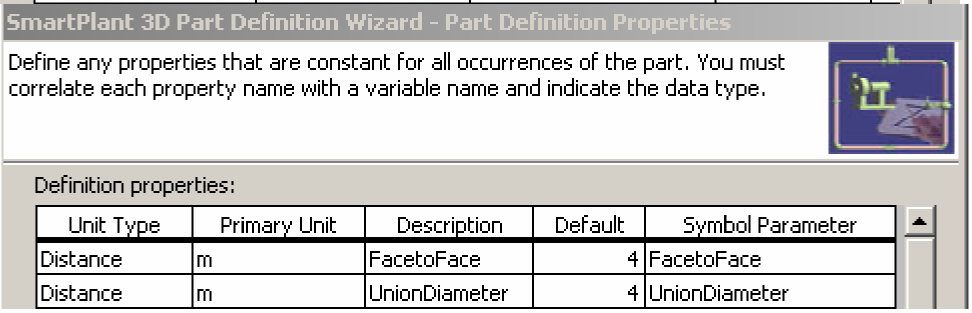


SmartPlant 3D Part Definition Wizard - Part Definition Properties

Define any properties that are constant for all occurrences of the part. You must correlate each property name with a variable name and indicate the data type.

Definition properties:

Interface Name	Attribute Name	Attribute User Name	Data Type	
IJUASP3DJunctionBox	FacetoFace	FacetoFace	Double	Dis
IJUASP3DJunctionBox	UnionDiameter	UnionDiameter	Double	Dis



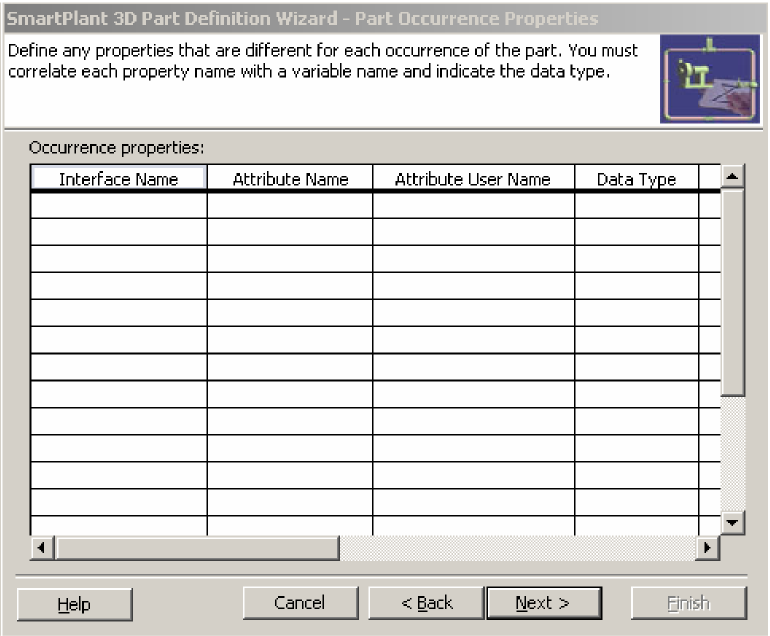
SmartPlant 3D Part Definition Wizard - Part Definition Properties

Define any properties that are constant for all occurrences of the part. You must correlate each property name with a variable name and indicate the data type.

Definition properties:

Unit Type	Primary Unit	Description	Default	Symbol Parameter
Distance	m	FacetoFace	4	FacetoFace
Distance	m	UnionDiameter	4	UnionDiameter

7. Select Next button to go the next page. Skip this page because our conduit part does not have occurrence attributes.



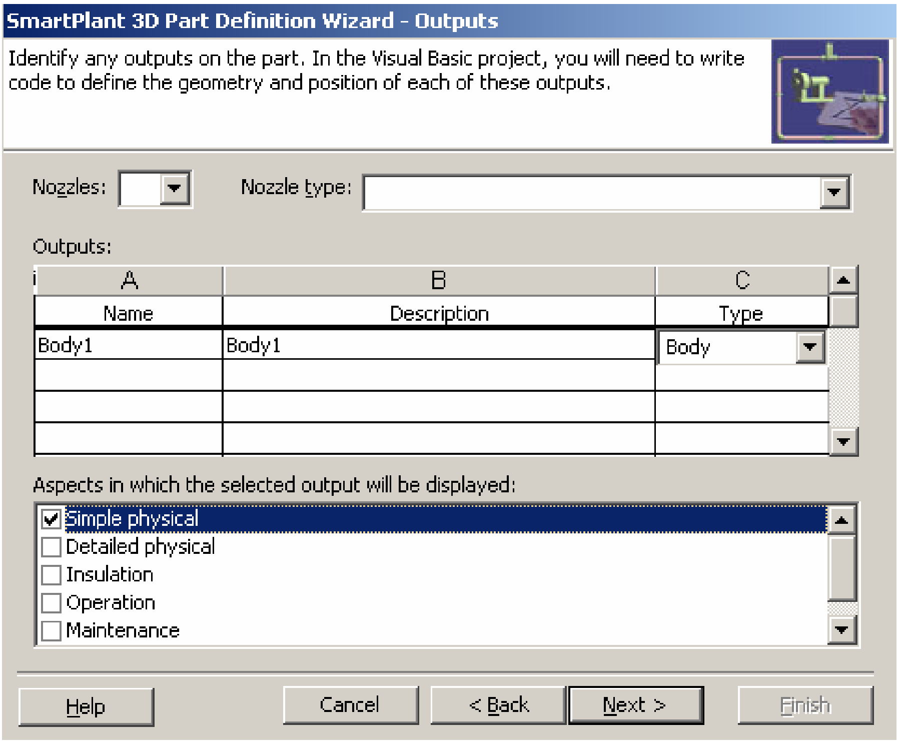
SmartPlant 3D Part Definition Wizard - Part Occurrence Properties

Define any properties that are different for each occurrence of the part. You must correlate each property name with a variable name and indicate the data type.

Occurrence properties:

Interface Name	Attribute Name	Attribute User Name	Data Type

8. Select Next button to go the next page. This page identifies all the outputs of the conduit part. We are going to define one output for our SP3DJunctionBox. The output is in the simple Physical aspect.



SmartPlant 3D Part Definition Wizard - Outputs

Identify any outputs on the part. In the Visual Basic project, you will need to write code to define the geometry and position of each of these outputs.

Nozzles:

Nozzle type:

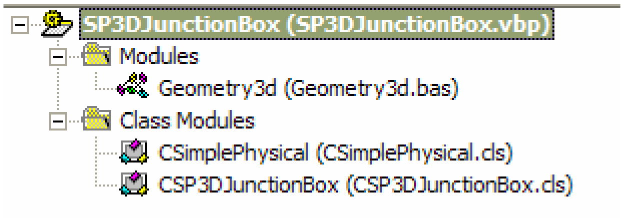
Outputs:

A	B	C
Name	Description	Type
Body1	Body1	Body

Aspects in which the selected output will be displayed:

☑ Simple physical
□ Detailed physical
□ Insulation
□ Operation
□ Maintenance

9. Hit Next button and Finish button to create the SP3DJunctionBox project template. The VB project consists of the following modules:



```
SP3DJunctionBox (SP3DJunctionBox.vbp)
    Modules
        Geometry3d (Geometry3d.bas)
    Class Modules
        CSimplePhysical (CSimplePhysical.cls)
        CSP3DJunctionBox (CSP3DJunctionBox.cls)
```

10. Open the **CSP3DJunctionBox Class** module. This Class contains several routines.
11. Go to the Class_Initialize() routine in the input section. Review the inputs and add two outputs definition for the conduit ports as shown below.

```
Private Sub Class_Initialize()
    Const METHOD = "Class_Initialize:"
    On Error GoTo Errx

    Set m_oSymbolHelper = New SymbolServices
    m_oSymbolHelper.ProjectName = "SP3DJunctionBox"
    m_oSymbolHelper.ClassName = "CSP3DJunctionBox"

' Inputs
    m_oSymbolHelper.NumInputs = 2
    m_oSymbolHelper.AddInputDef 1, "FacetoFace", "FacetoFace", 4
    m_oSymbolHelper.AddInputDef 2, "UnionDiameter", "UnionDiameter", 4
```

```
' Outputs
    m_oSymbolHelper.NumOutputs = 3
    m_oSymbolHelper.AddOutputDef 1, "Body1", "Body1", 1
    m_oSymbolHelper.AddOutputDef 2, "port1", "port1", 1
    m_oSymbolHelper.AddOutputDef 3, "port2", "port2", 1

' Aspects
    m_oSymbolHelper.NumAspects = 1
    m_oSymbolHelper.AddAspectDef 1, "SimplePhysical", "SimplePhysical", 1

    Exit Sub
Errx:
    Err.Raise Err.Number, Err.Source & " " & METHOD, Err.Description, _
        Err.HelpFile, Err.HelpContext
End Sub
```

12. Go to **CSimplePhysical Class** module and add your code to create the outputs:
13. Go to the Insert your code for output 1 (Body1) section. The following code will use the PlaceBox() routine to create a Box for the Junction box. The PlaceBox routine is defined in Geometry3d module. This function takes the two opposite corner points of the box as input parameters.

```
    Dim pPos1   As IJDPosition
    Dim pPos2   As IJDPosition
    Set pPos1 = New DPosition
    Set pPos2 = New DPosition

    pPos1.Set -parFacetoFace / 2, -parUnionDiameter / 2, -parUnionDiameter / 2
    pPos2.Set parFacetoFace / 2, parUnionDiameter / 2, parUnionDiameter / 2

    Set ObjBody1 = PlaceBox(m_OutputColl, pPos1, pPos2)

    iOutput = iOutput + 1
    m_OutputColl.AddOutput arrayOfOutputs(iOutput), ObjBody1
```

14. The following code will use the CreateConduitNozzle() method to create the conduit ports. The CreateConduitNozzle() routine is defined in the Geometry3d module.

```
     ' Place Nozzle 1

        Dim pipeDiam        As Double
        Dim flangeThick     As Double
        Dim sptOffset       As Double
        Dim flangeDiam      As Double
        Dim depth           As Double
        Dim ConduitOD       As Double

        RetrieveParameters 1, oPartFclt, m_OutputColl, ConduitOD, flangeThick, flangeDiam, sptOffset, depth

        Dim oPlacePoint As AutoMath.DPosition
        Dim oDir        As AutoMath.DVector
        Dim objNozzle   As GSCADNozzleEntities.IJConduitPortOcc
        Dim faceToFace  As Double
```

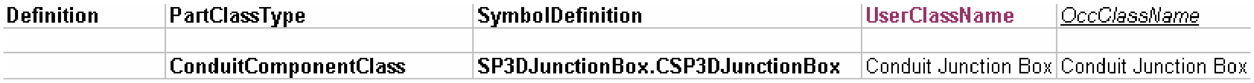
```
 Set oPlacePoint = New AutoMath.DPosition
 Set oDir = New AutoMath.DVector
 faceToFace = arrayOfInputs(2)
 oPlacePoint.Set -faceToFace / 2 - sptOffset + depth, 0, 0
 oDir.Set -1, 0, 0
 Set oPartFclt = arrayOfInputs(1)
 Set objNozzle = CreateConduitNozzle(oPlacePoint, oDir, m_OutputColl, oPartFclt, 1)
' Set the output
 iOutput = iOutput + 1
 m_OutputColl.AddOutput arrayOfOutputs(iOutput), objNozzle
 Set objNozzle = Nothing

' Place Nozzle 2
 RetrieveParameters 2, oPartFclt, m_OutputColl, ConduitOD, flangeThick, flangeDiam, sptOffset, depth
 oPlacePoint.Set faceToFace / 2 + sptOffset - depth, 0, 0
 oDir.Set 1, 0, 0
 Set objNozzle = CreateConduitNozzle(oPlacePoint, oDir, m_OutputColl, oPartFclt, 2)
' Set the output
 iOutput = iOutput + 1
 m_OutputColl.AddOutput arrayOfOutputs(iOutput), objNozzle
```

15. Use the Set statement to clear the references from all object variables.

```
 Set objNozzle = Nothing
 Set oPlacePoint = Nothing
 Set oDir = Nothing
 Set pPos1 = Nothing
 Set pPos2 = Nothing
 Set ObjBody1 = Nothing
```

16. Compile the VB project and save the dll in c:\train\ *SP3DJunctionBox*
17. Save the VB SP3DJunctionBox project.
18. Open the Conduit.xls workbook. Create the JuctionBox Part Class ass follows:

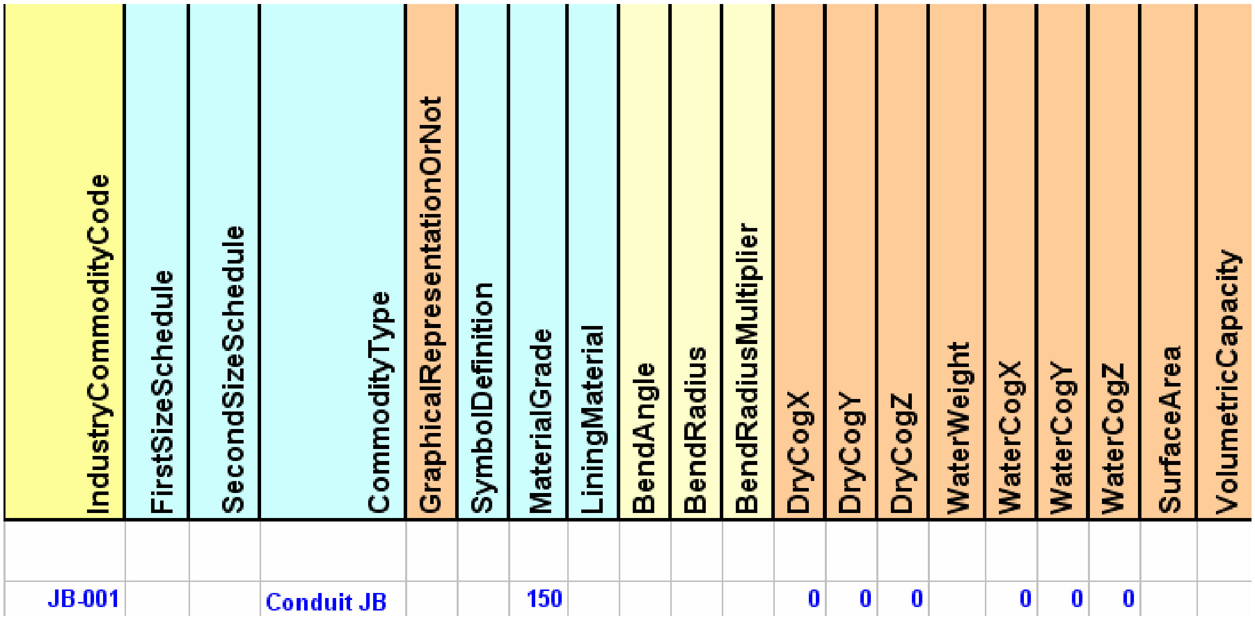
In the Definition Section:



Definition	PartClassType	SymbolDefinition	UserClassName	OccClassName
	ConduitComponentClass	SP3DJunctionBox.CSP3DJunctionBox	Conduit Junction Box	Conduit Junction Box

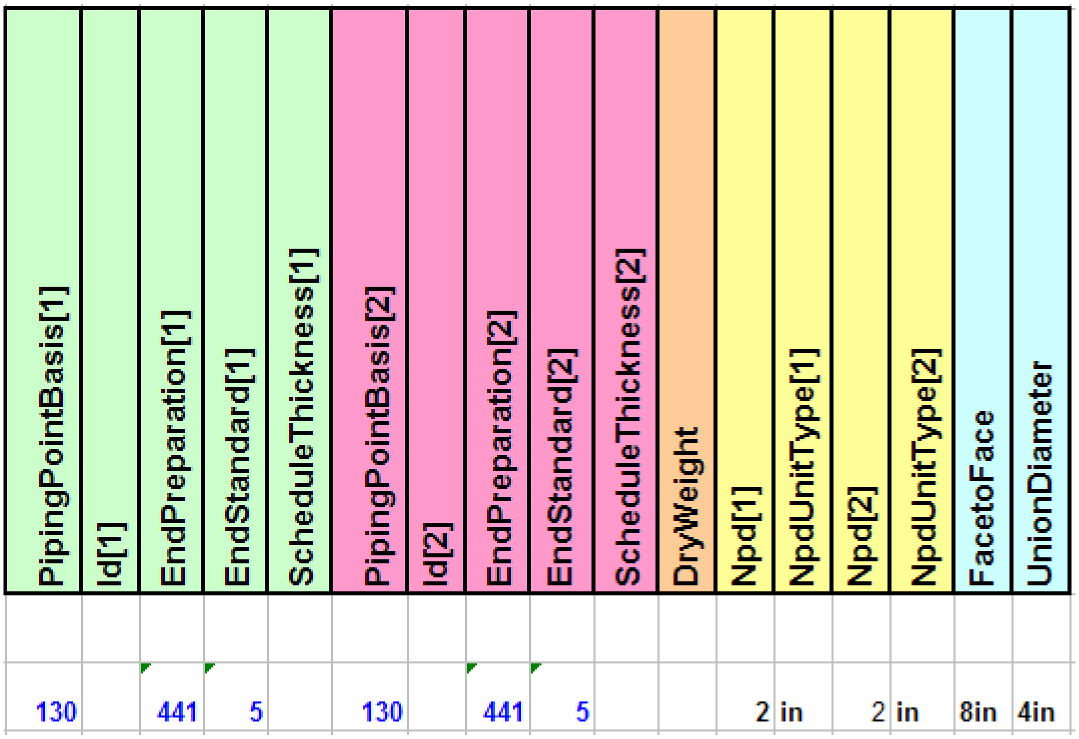
In the Part Section:

System attributes:



IndustryCommodityCode	FirstSizeSchedule	SecondSizeSchedule	CommodityType	GraphicalRepresentationOrNot	SymbolDefinition	MaterialGrade	LiningMaterial	BendAngle	BendRadius	BendRadiusMultiplier	DryCogX	DryCogY	DryCogZ	WaterWeight	WaterCogX	WaterCogY	WaterCogZ	SurfaceArea	VolumetricCapacity
JB-001			Conduit JB			150					0	0	0		0	0	0		

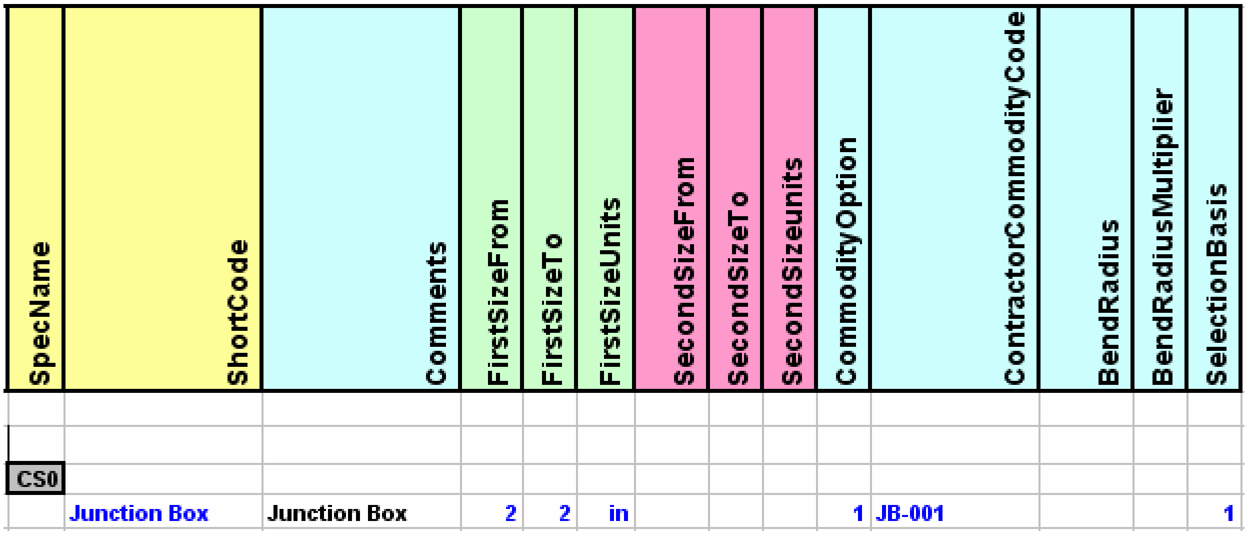
Port and Dimension Information:



PipingPointBasis[1]	Id[1]	EndPreparation[1]	EndStandard[1]	ScheduleThickness[1]	PipingPointBasis[2]	Id[2]	EndPreparation[2]	EndStandard[2]	ScheduleThickness[2]	DryWeight	Npd[1]	NpdUnitType[1]	Npd[2]	NpdUnitType[2]	FacetoFace	UnionDiameter
130		441	5		130		441	5			2	in	2	in	8in	4in

Conduit Filter Records

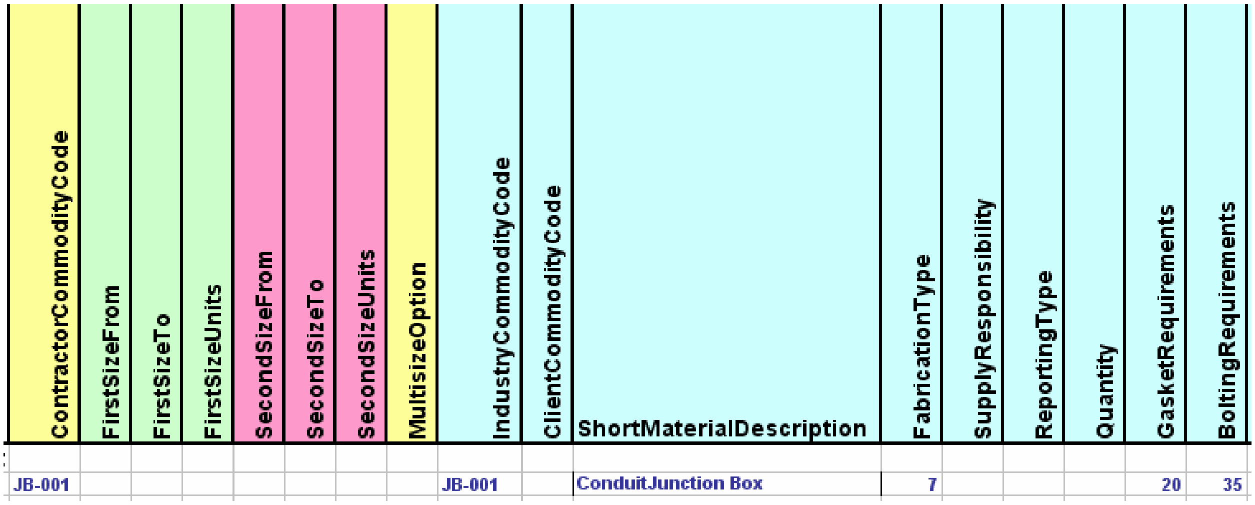
19. Go to the ConduitFilter worksheet.
20. Add record for the junction box as shown below:



SpecName	ShortCode	Comments	FirstSizeFrom	FirstSizeTo	FirstSizeUnits	SecondSizeFrom	SecondSizeTo	SecondSizeunits	CommodityOption	ContractorCommodityCode	BendRadius	BendRadiusMultiplier	SelectionBasis
CS0													
	Junction Box	Junction Box	2	2	in				1	JB-001			1

ConduitCommodityMatlControlData Data

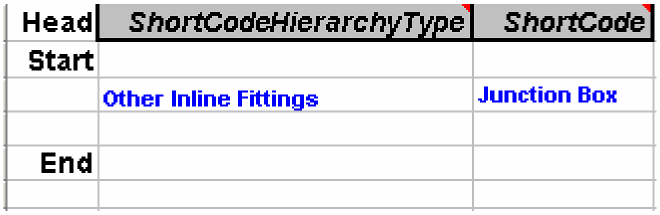
21. Go to the ConduitCommodityMatlControlData worksheet.
22. Add record for the junction box as shown below:



ContractorCommodityCode	FirstSizeFrom	FirstSizeTo	FirstSizeUnits	SecondSizeFrom	SecondSizeTo	SecondSizeUnits	MultisizeOption	IndustryCommodityCode	ClientCommodityCode	ShortMaterialDescription	FabricationType	SupplyResponsibility	ReportingType	Quantity	GasketRequirements	BoltingRequirements
JB-001								JB-001		ConduitJunction Box	7				20	35

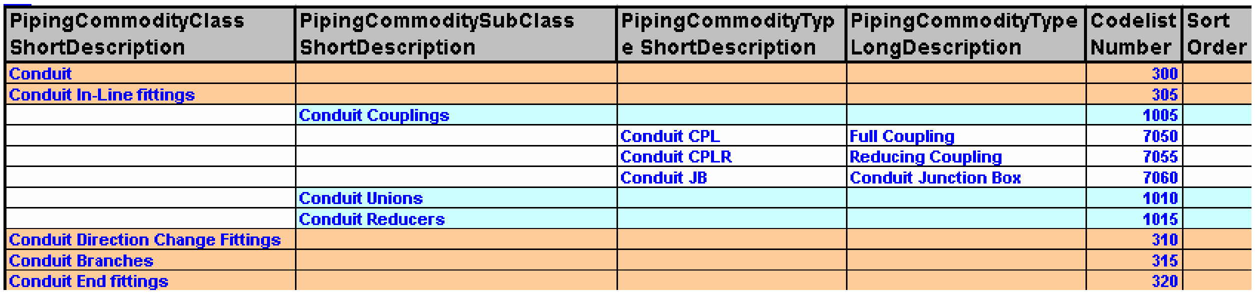
ShortCodeHierarchyRule Data

23. Create a sheet called ShortCodeHierarchyRule and add the appropriate records as shown below:



Head	ShortCodeHierarchyType	ShortCode
Start		
	Other Inline Fittings	Junction Box
End		

24. Save the workbook.
25. Open the AllCodeList.xls. Go to the PipingCommodityType worksheet.
26. Add record for the new Conduit Commodity Type as shown below:



PipingCommodityClass ShortDescription	PipingCommoditySubClass ShortDescription	PipingCommodityType ShortDescription	PipingCommodityType LongDescription	Codelist Number	Sort Order
Conduit				300	
Conduit In-Line fittings				305	
	Conduit Couplings			1005	
		Conduit CPL	Full Coupling	7050	
		Conduit CPLR	Reducing Coupling	7055	
		Conduit JB	Conduit Junction Box	7060	
	Conduit Unions			1010	
	Conduit Reducers			1015	
Conduit Direction Change Fittings				310	
Conduit Branches				315	
Conduit End fittings				320	

27. Save the workbook.
28. Select Start => Programs => Intergraph SmartPlant3D => Database Tools => Bulkload Reference Data.
29. Select the “Add” option under “Excel Files” and select Conduit.xls
30. Select the “Add” option under “Excel Codelist Files” and select Allcodelist.xls
31. Select an existing catalog.
32. Load the records into the database using the “Append” mode.
33. Once the bulkload process is complete, review the log file. Next, run the View Generator utility on the model to re-create the views in the model database. Finally, Re-generate the report databases.
34. Go to the Electrical Task and place the Junction Box.

Lab 22: Shape Symbol

Objectives

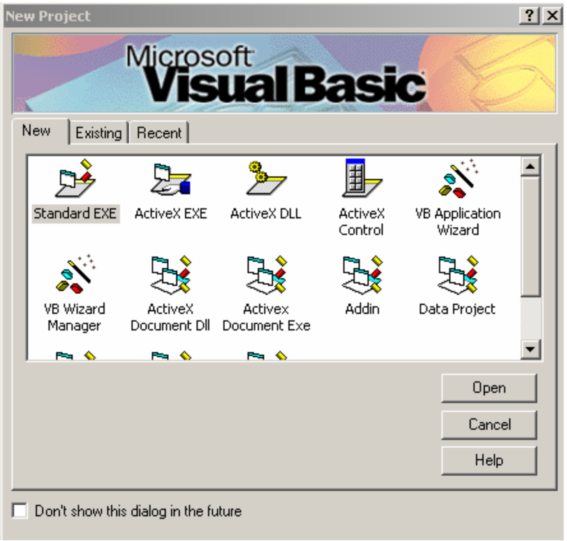
After completing this lab, you will be able to:

- Create shape symbol using the SmartPlant 3D Part Definition Wizard
- Learn to use the Symbol Helper service to create the symbol definition
- Learn to use the Geometry Helper service to create geometric entities for the symbol's output

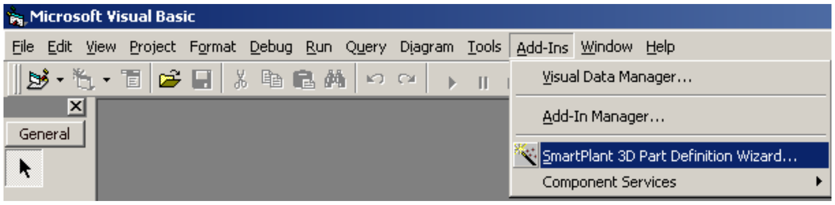
4. Create the following directory:

   c:\train\HollowCy

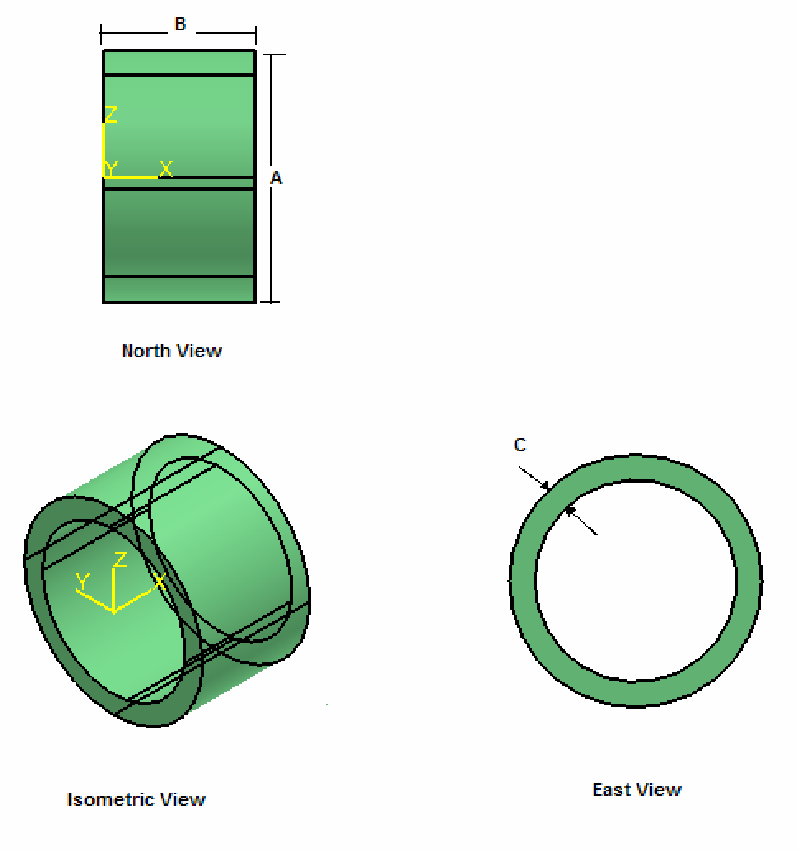
5. Run Microsoft Visual Basic 6.0
6. Close the Microsoft New Project dialog box.



12. Go to the Add-Ins Option and Select SmartPlant 3D Part Definition Wizard.

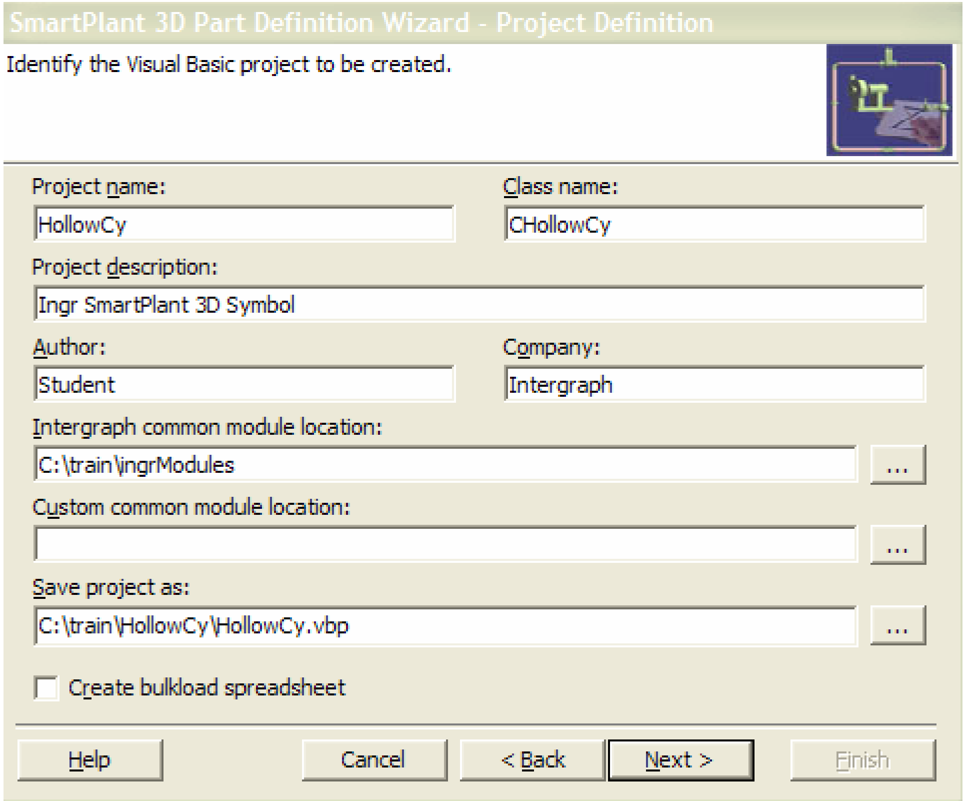


13. The Next step is to create the shape symbol definition template using SP3D Part Definition Symbol Wizard.



14. In this page you define the VB project name. Key in the following information:

Project Name: *HollowCy*
Author: *Student*
Company: *Intergraph*
Intergraph Module location: *c:\Train\IngrModules*
Save the VB project as: *c:\Train\ HollowCy*
Disable the create bulkload spreadsheet.



SmartPlant 3D Part Definition Wizard - Project Definition

Identify the Visual Basic project to be created.

Project name: HollowCy

Class name: CHollowCy

Project description: Ingr SmartPlant 3D Symbol

Author: Student

Company: Intergraph

Intergraph common module location: C:\train\ingrModules

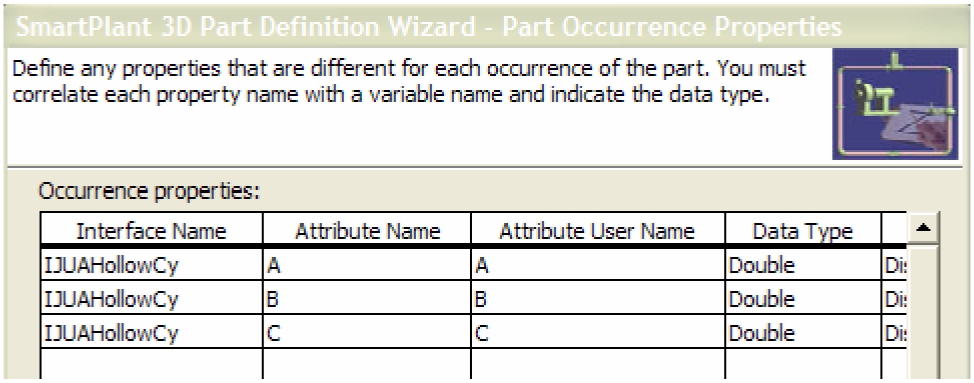
Custom common module location:

Save project as: C:\train\HollowCy\HollowCy.vbp

□ Create bulkload spreadsheet

15. Select Next button to go the next page. This page is to define any properties that are constant for all occurrences of the operator part. Select Next button to go the next page.

16. This page defines all occurrence properties of the shape part. Key in the following data:

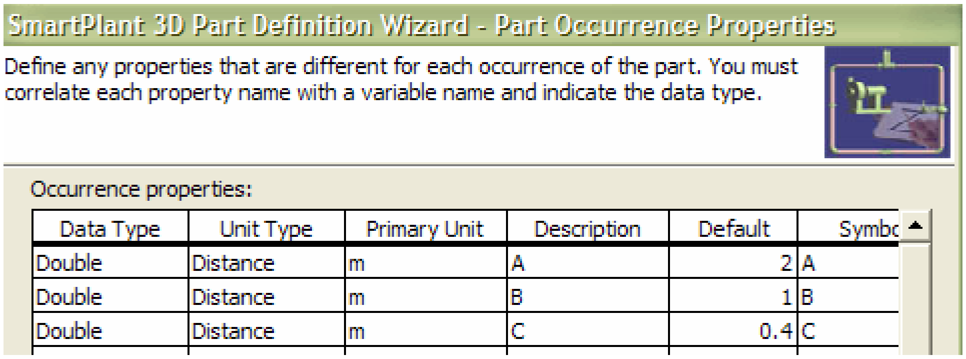


SmartPlant 3D Part Definition Wizard - Part Occurrence Properties

Define any properties that are different for each occurrence of the part. You must correlate each property name with a variable name and indicate the data type.

Occurrence properties:

Interface Name	Attribute Name	Attribute User Name	Data Type	
IJUAHollowCy	A	A	Double	Di
IJUAHollowCy	B	B	Double	Di
IJUAHollowCy	C	C	Double	Di



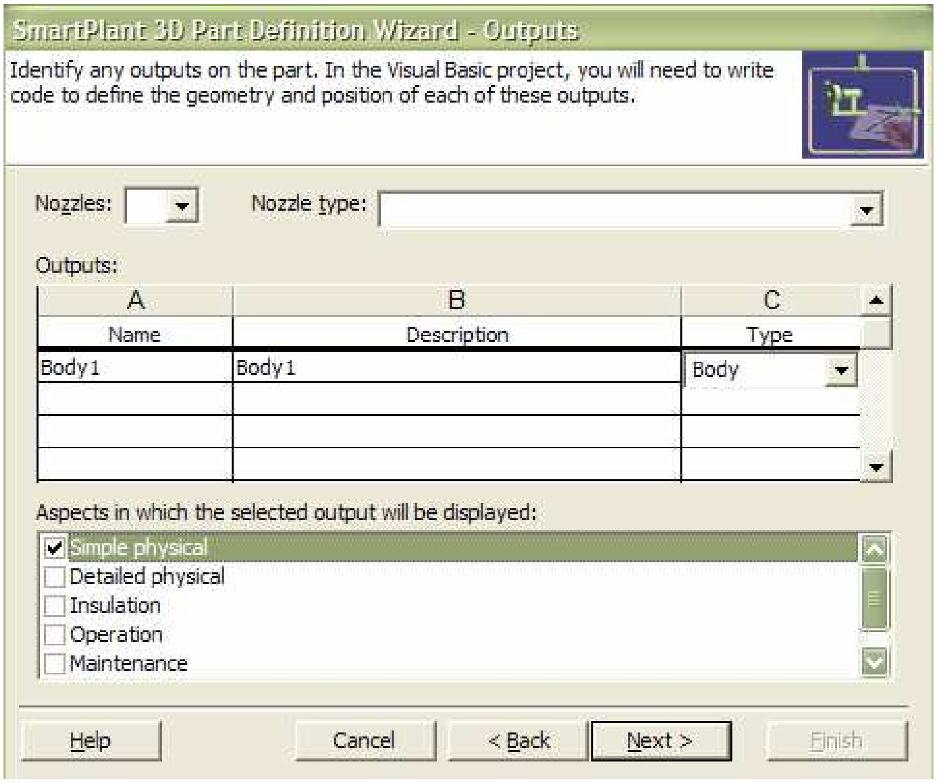
SmartPlant 3D Part Definition Wizard - Part Occurrence Properties

Define any properties that are different for each occurrence of the part. You must correlate each property name with a variable name and indicate the data type.

Occurrence properties:

Data Type	Unit Type	Primary Unit	Description	Default	Symbc
Double	Distance	m	A	2	A
Double	Distance	m	B	1	B
Double	Distance	m	C	0.4	C

17. Select Next button to go the next page. This page identifies all the outputs of the shape part. We are going to define one output: Body1



SmartPlant 3D Part Definition Wizard - Outputs

Identify any outputs on the part. In the Visual Basic project, you will need to write code to define the geometry and position of each of these outputs.

Nozzles:

Nozzle type:

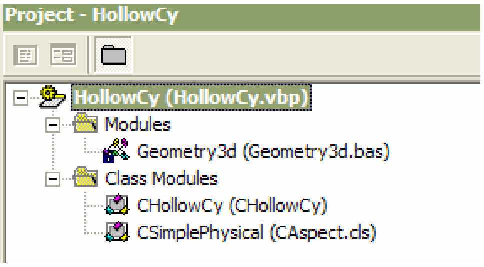
Outputs:

A	B	C
Name	Description	Type
Body1	Body1	Body

Aspects in which the selected output will be displayed:

☑ Simple physical
□ Detailed physical
□ Insulation
□ Operation
□ Maintenance

18. Press Next button and Finish button to create the shape project template. The VB project consists of the following modules:



Project - HollowCy

```
HollowCy (HollowCy.vbp)
    Modules
        Geometry3d (Geometry3d.bas)
    Class Modules
        CHollowCy (CHollowCy)
        CSimplePhysical (CAspect.cls)
```

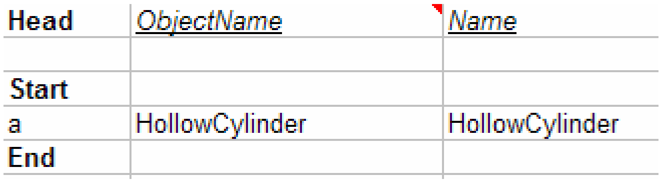
19. Open the **CHollowCy Class** module. This Class contains several routines.
20. Go to the Class_Initialize() routine. Review the inputs and outputs section. Add additional outputs as shown below:

```
Private Sub Class_Initialize()
    Const METHOD = "Class_Initialize:"
    On Error GoTo Errx

    Set m_oSymbolHelper = New SymbolServices
    m_oSymbolHelper.ProjectName = "HollowCy"
    m_oSymbolHelper.ClassName = "CHollowCy"

' Inputs
    m_oSymbolHelper.NumInputs = 3
    m_oSymbolHelper.AddInputDef 1, "A", "A", 2
    m_oSymbolHelper.AddInputDef 2, "B", "B", 1
    m_oSymbolHelper.AddInputDef 3, "C", "C", 0.4

' Outputs
    m_oSymbolHelper.NumOutputs = 8
    m_oSymbolHelper.AddOutputDef 1, "Body1", "Body1", 1
    m_oSymbolHelper.AddOutputDef 2, "Body2", "Body2", 1
    m_oSymbolHelper.AddOutputDef 3, "Body3", "Body3", 1
    m_oSymbolHelper.AddOutputDef 4, "Body4", "Body4", 1
    m_oSymbolHelper.AddOutputDef 5, "Body5", "Body5", 1
    m_oSymbolHelper.AddOutputDef 6, "Body6", "Body6", 1
    m_oSymbolHelper.AddOutputDef 7, "Body7", "Body7", 1
    m_oSymbolHelper.AddOutputDef 8, "Body8", "Body8", 1

' Aspects
    m_oSymbolHelper.NumAspects = 1
    m_oSymbolHelper.AddAspectDef 1, "SimplePhysical", "SimplePhysical", 1

    Exit Sub
Errx:
    Err.Raise Err.Number, Err.Source & " " & METHOD, Err.Description, _
        Err.HelpFile, Err.HelpContext
End Sub
```

13. Go to **CSimplePhysical Class** module and add your code to create the outputs:
14. Go to the “*Insert your code for output 1 (OPBody)*” section. The following code will use the Geometry Factory methods to create the graphic entities for the hollow cylinder.

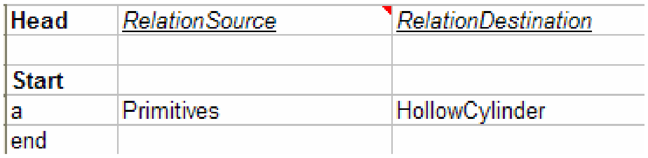
```
' Inputs
    Set oPartFclt = arrayOfInputs(1)
    parA = arrayOfInputs(2)
    parB = arrayOfInputs(3)
    parC = arrayOfInputs(4)
    m_oGeomHelper.OutputCollection = m_OutputColl

    iOutput = 0

        Dim oErrors As IJEditErrors
    Set oErrors = New JServerErrors
    If parA <= 0 Or parB <= 0 Or parC <= 0 Then
        oErrors.Add E_FAIL, "CSP3DHollowCy", "Shape Dimensions should be greater than zero",
"ZeroOrNegative"
        GoTo Errx:
    End If

Dim oGeomFactory As New GeometryFactory
    Dim oCircle(2) As Circle3d
    Dim oProjection As Projection3d
    Dim oDir As IJDVector
    Set oDir = New DVector
    oDir.Set 1, 0, 0

'create the cylinders

    Set oCircle(1) = oGeomFactory.Circles3d.CreateByCenterNormalRadius(m_OutputColl.ResourceManager,
0, 0, 0, 1, 0, 0, parA / 2)
    Set oCircle(2) = oGeomFactory.Circles3d.CreateByCenterNormalRadius(m_OutputColl.ResourceManager,
0, 0, 0, 1, 0, 0, parA / 2 - parC)

    iOutput = iOutput + 1
    m_OutputColl.AddOutput arrayOfOutputs(iOutput), oCircle(1)
    iOutput = iOutput + 1
    m_OutputColl.AddOutput arrayOfOutputs(iOutput), oCircle(2)

    Set oProjection = PlaceProjection(m_OutputColl, oCircle(1), oDir, parB, False)

    iOutput = iOutput + 1
    m_OutputColl.AddOutput arrayOfOutputs(iOutput), oProjection

    Set oProjection = PlaceProjection(m_OutputColl, oCircle(2), oDir, parB, False)

    iOutput = iOutput + 1
    m_OutputColl.AddOutput arrayOfOutputs(iOutput), oProjection

 'create the left face

    Dim oPlane  As IngrGeom3D.Plane3d
    Set oPlane = oGeomFactory.Planes3d.CreateByPointNormal(m_OutputColl.ResourceManager,  _
                                                                        0, 0, 0, 1, 0, 0)

    Dim oElements       As IJElements
    Dim objCStr         As IngrGeom3D.ComplexString3d
    Dim i As Integer

    Set oElements = New JObjectCollection

    For i = 1 To 2
        oElements.Add oCircle(i)
        Set objCStr = oGeomFactory.ComplexStrings3d.CreateByCurves(Nothing, oElements)
        oPlane.AddBoundary objCStr
        oElements.Clear
        objCStr.RemoveCurve True
    Next i
    iOutput = iOutput + 1
    m_OutputColl.AddOutput arrayOfOutputs(iOutput), oPlane

  'create the right face

    Set oCircle(1) = oGeomFactory.Circles3d.CreateByCenterNormalRadius(m_OutputColl.ResourceManager, parB, 0, 0, 1, 0, 0, parA / 2)
    Set oCircle(2) = oGeomFactory.Circles3d.CreateByCenterNormalRadius(m_OutputColl.ResourceManager, parB, 0, 0, 1, 0, 0, parA / 2 - parC)

    iOutput = iOutput + 1
    m_OutputColl.AddOutput arrayOfOutputs(iOutput), oCircle(1)
    iOutput = iOutput + 1
    m_OutputColl.AddOutput arrayOfOutputs(iOutput), oCircle(2)

    Set oPlane = oGeomFactory.Planes3d.CreateByPointNormal(m_OutputColl.ResourceManager, parB, 0, 0, 1, 0, 0)

    For i = 1 To 2
        oElements.Add oCircle(i)
        Set objCStr = oGeomFactory.ComplexStrings3d.CreateByCurves(Nothing, oElements)
        oPlane.AddBoundary objCStr
        oElements.Clear
        objCStr.RemoveCurve True
    Next i
    iOutput = iOutput + 1
    m_OutputColl.AddOutput arrayOfOutputs(iOutput), oPlane

    Set oProjection = Nothing
    Set oCircle(1) = Nothing
    Set oCircle(2) = Nothing
    Set oGeomFactory = Nothing
    Set oPlane = Nothing
    Set oDir = Nothing
    Set oElements = Nothing
    Set objCStr = Nothing
```

15. Compile the VB project and save the dll in the c:\Train\HollowCy
16. Save the VB HollowCy project.
17. Open the Shapes.xls located in [Install Product]\CatalogData\BulkLoad\DataFiles
18. Go the ClassNodeType sheet and add the following entry.



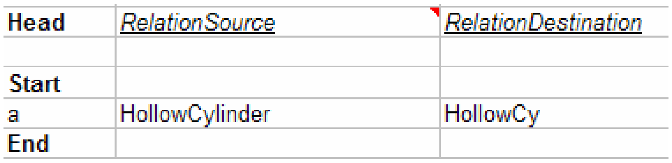
Head	ObjectName	Name
Start		
a	HollowCylinder	HollowCylinder
End		

19. Go the R-Hierarchy sheet and add the following entry.



Head	RelationSource	RelationDestination
Start		
a	Primitives	HollowCylinder
end		

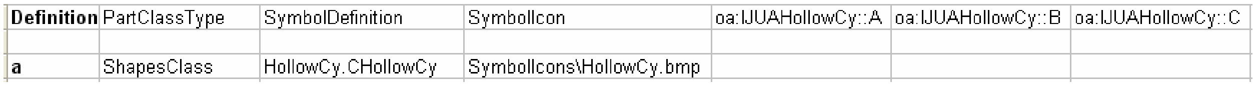
20. Go the R-ClassNodeDescribes sheet and add the following entry.



Head	RelationSource	RelationDestination
Start		
a	HollowCylinder	HollowCy
End		

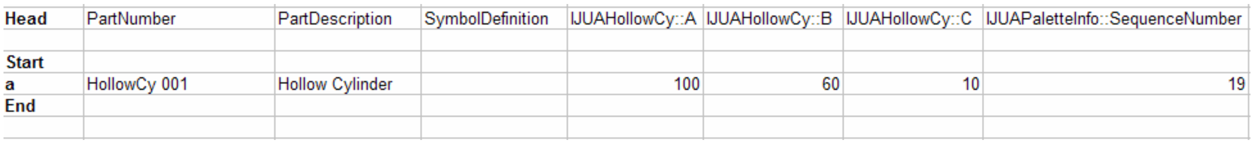
21. Create a New Part Class called HollowCy with the following data:

In the Definition Section:



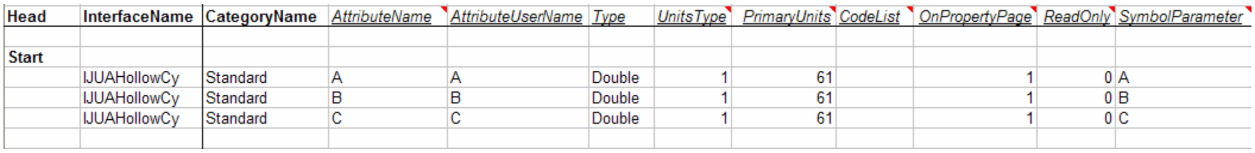
Definition	PartClassType	SymbolDefinition	SymbolIcon	oa:IJUAHollowCy::A	oa:IJUAHollowCy::B	oa:IJUAHollowCy::C
a	ShapesClass	HollowCy.CHollowCy	SymbolIcons\HollowCy.bmp			

In the Part Section:



Head	PartNumber	PartDescription	SymbolDefinition	IJUAHollowCy::A	IJUAHollowCy::B	IJUAHollowCy::C	IJUAPaletteInfo::SequenceNumber
Start							
a	HollowCy 001	Hollow Cylinder		100	60	10	19
End							

22. Create a new interface called IJUAHollowCy. Go to the Custom Interface sheet and add the following entries:



Head	InterfaceName	CategoryName	AttributeName	AttributeUserName	Type	UnitsType	PrimaryUnits	CodeList	OnPropertyPage	ReadOnly	SymbolParameter
Start											
	IJUAHollowCy	Standard	A	A	Double	1	61		1	0	A
	IJUAHollowCy	Standard	B	B	Double	1	61		1	0	B
	IJUAHollowCy	Standard	C	C	Double	1	61		1	0	C

23. Use Microsoft Paint and create a HollowCy.bmp and HollowCyicon.bmp. Place these files on your symbol share.
24. Go to the symbol share \\machine\\Symbols\ShapeTypes and open ShapeTypes.xml
25. Add the following lines in ShapeTypes.xml

```
<ShapeType name="HollowCy" picture= "HollowCyicon.bmp">
</ShapeType>
```

26. Load the information into the Catalog using the Append mode. Once the bulkload process is complete, run the View Generator utility on the model to re-create the views in the model database. Finally, Re-generate the report databases.
27. Go to the Equipment Task and place your shape.

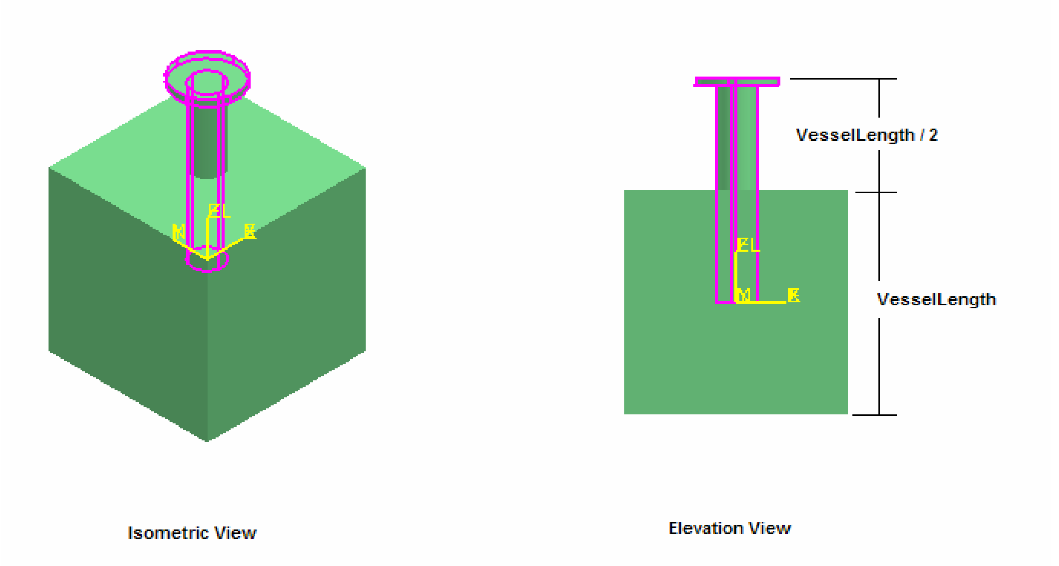
Lab 23: Equipment Symbol with pipe port created from a placeholder (Optional)

Objectives

After completing this lab, you will be able to:

- Create a simple equipment smart occurrence symbol with pipe nozzle created from a placeholder defined in the symbol
- Learn to use the Symbol Helper service to create the symbol definition
- Learn to use the Geometry Helper service to create simple geometric shapes for the symbol's output
- Use the Equipment CAD Helper to define the Custom Assembly Definition
- Learn to use the IJDAggregatorDescription , IJDMemberDescriptions, and IJDPropertyDescriptions to define the behaviors of the custom assembly occurrence (CAO)
- Use custom methods to create and manipulate the members within the CAD definition
- Use the IJDeletableMember interface to make the member deletable

In this lab, you will create an equipment symbol as shown below. You start by using the SP3DEqpTemplateAsm template provided by the instructor to create the symbol. This symbol consists of one geometric rectangular entity and a pipe nozzle to define the symbol's output. One input "VesselLength" is required to draw this symbol. Use the Equipment Custom Assembly Definition (CAD) Helper to create the pipe port from the placeholder defined in the symbol. The pipe port data is retrieved from the part at the given index. This type of creation is used when the position and the orientation of the pipe nozzle are driven totally or partially by the symbol.



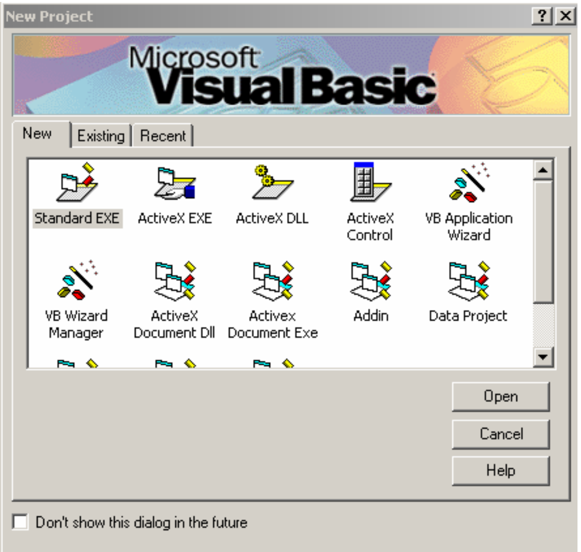
Isometric View

Elevation View

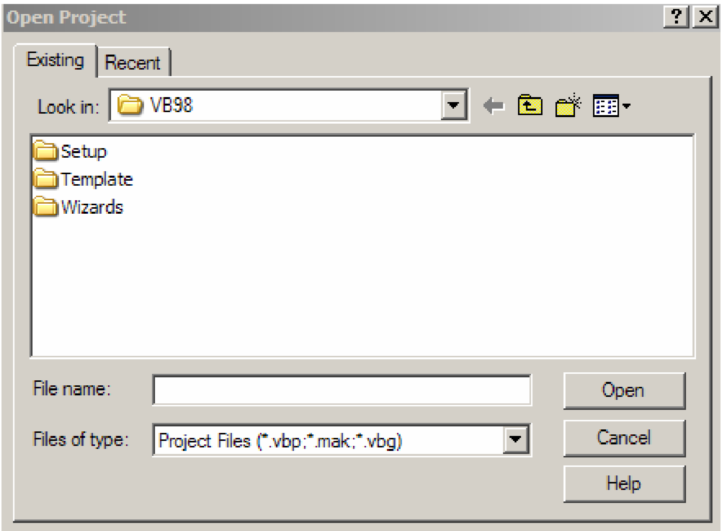
1. Create a directory called lab2 as follows:

   c:\train\lab2

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

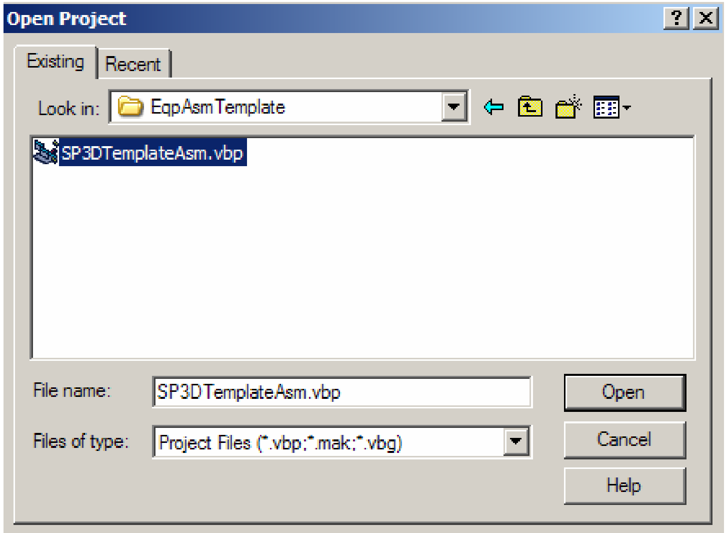


Look in: VB98

File name:

Files of type: Project Files (*.vbp;*.mak;*.vbg)

5. Navigate the tree and open the SP3DTemplateAsm Template project



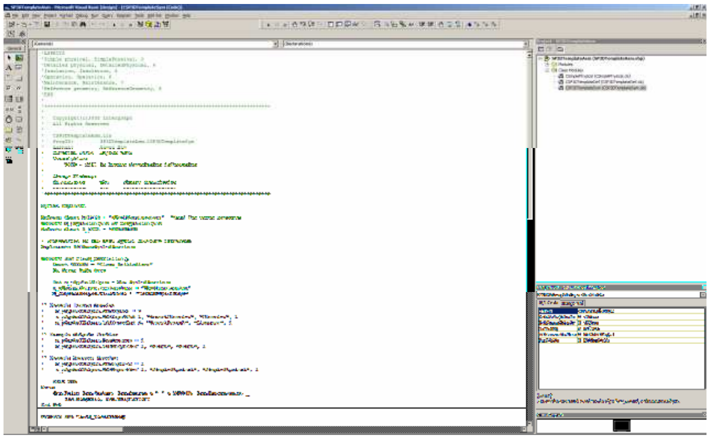
Look in: EqpAsmTemplate

SP3DTemplateAsm.vbp

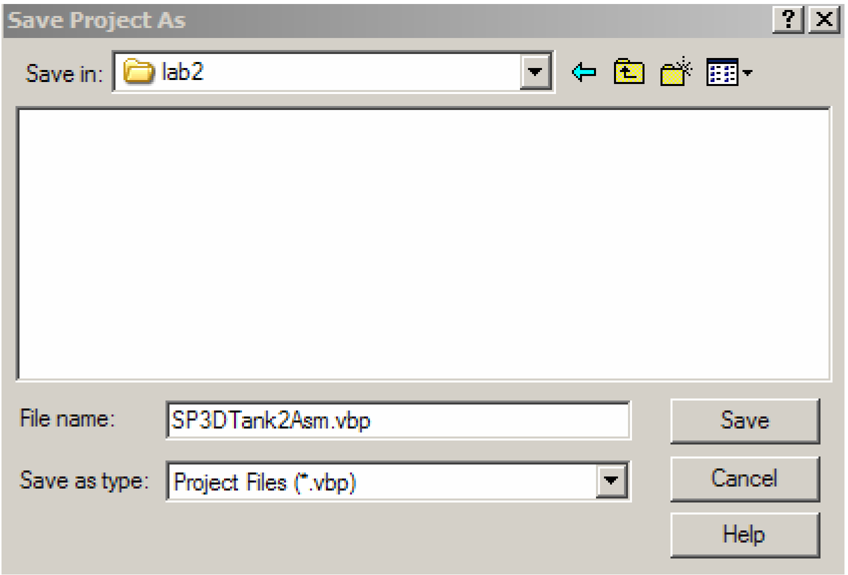
File name: SP3DTemplateAsm.vbp

Files of type: Project Files (*.vbp;*.mak;*.vbg)

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



7. Go to the Explorer Window and select the Project file in the tree. Select *File -> Save Project As* option to save the project as SP3DTank2Asm.vbp under the lab2 directory

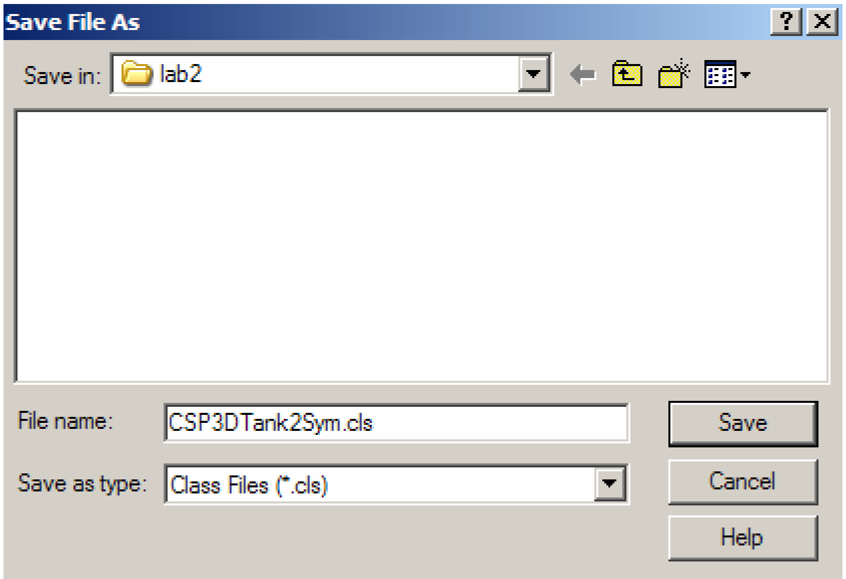


Save in: lab2

File name: SP3DTank2Asm.vbp

Save as type: Project Files (*.vbp)

8. Go to the Explorer Window and select the CSP3DTemplateSym class file in the tree. Select *File -> Save CSP3DTemplateSym.cls As* option to save the class module as CSP3DTank2Sym.cls under lab2 directory

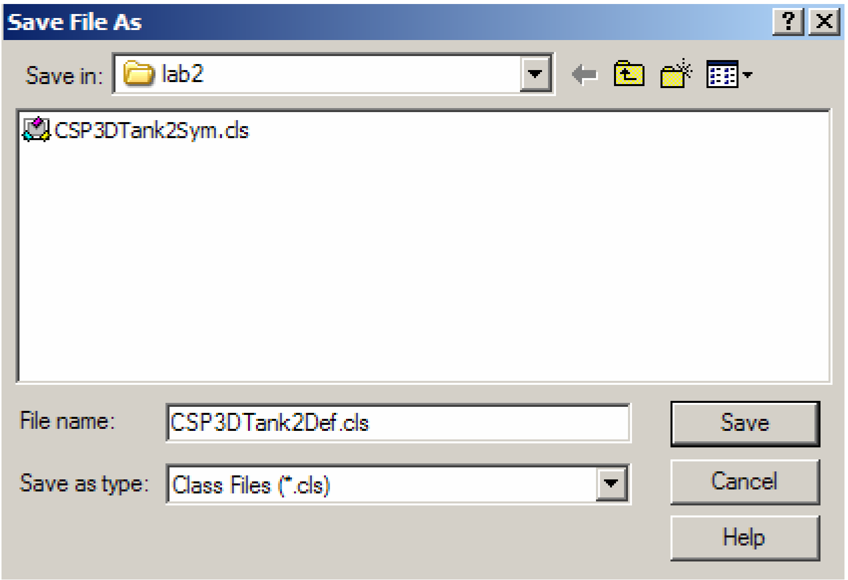


Save in: lab2

File name: CSP3DTank2Sym.cls

Save as type: Class Files (*.cls)

9. Go to the Explorer Window and select the CSP3DTemplateDef class file in the tree. Select *File -> Save CSP3DTemplateDef.cls As* option to save the class module as CSP3DTank2Def.cls under lab2 directory



Save in: lab2

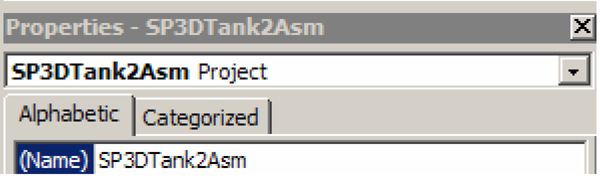
CSP3DTank2Sym.cls

File name: CSP3DTank2Def.cls

Save as type: Class Files (*.cls)

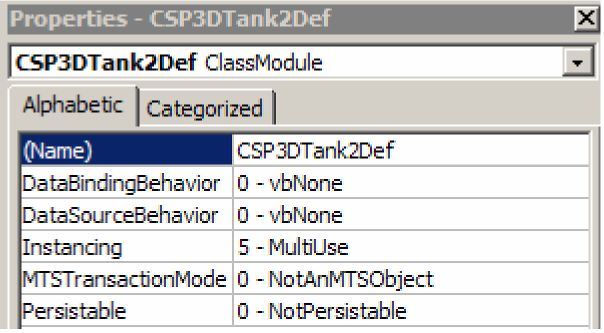
Go to the Explorer Window and select the CSimplePhysical class file in the tree. Select *File -> Save CSimplePhysical.cls As* option to save the class module as CSimplePhysical.cls under lab2 directory. Repeat the procedure for the two bas modules.

10. Go to the Properties Window and change the name of the Project and both Class Modules as follows:



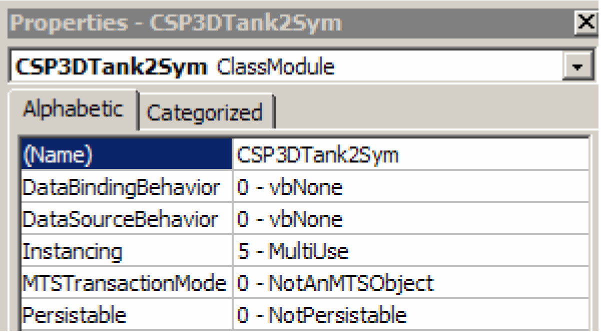
SP3DTank2Asm Project

(Name) SP3DTank2Asm



CSP3DTank2Def ClassModule

(Name)	CSP3DTank2Def
DataBindingBehavior	0 - vbNone
DataSourceBehavior	0 - vbNone
Instancing	5 - MultiUse
MTSTransactionMode	0 - NotAnMTSObject
Persistable	0 - NotPersistable



Properties - CSP3DTank2Sym

CSP3DTank2Sym ClassModule

(Name)	CSP3DTank2Sym
DataBindingBehavior	0 - vbNone
DataSourceBehavior	0 - vbNone
Instancing	5 - MultiUse
MTSTransactionMode	0 - NotAnMTSObject
Persistable	0 - NotPersistable

11. Go to the General Declarations section in CSP3DTank2Sym module and change the value of the *Constant Module variable* from *“CSP3DTemplateAsm:”* to *“CSP3DTank2Asm:”*

```
Private Const MODULE = "CSP3DTank2Asm:"  'Used for error messages
```

12. Go to the Class_Initialize() routine, rename the project name and class name as shown below:

```
    Set m_oSymbolHelper = New SymbolServices
    m_oSymbolHelper.ProjectName = "SP3DTank2Asm"
    m_oSymbolHelper.ClassName = "CSP3DTank2Sym"
```

13. In this Class_Initialize() routine, add the following code to define the inputs, outputs and aspects definition for this symbol.

```
'Inputs Section
    m_oSymbolHelper.NumInputs = 1
    m_oSymbolHelper.AddInputDef 1, "VesselLength", “VesselLength", 1
'
'Outputs Section
    m_oSymbolHelper.NumOutputs = 2
    m_oSymbolHelper.AddOutputDef 1, "Body1", "Body1", 1
    m_oSymbolHelper.AddOutputDef 2, "PipingNoz1", "PipingNoz1", 1

'Aspects Section
    m_oSymbolHelper.NumAspects = 1
    m_oSymbolHelper.AddAspectDef 1, "SimplePhysical", "SimplePhysical", 1
```

14. Go to the General Declarations section in CSP3DTank2Def module and change the value of the *Constant Module variable* from *"SP3DTemplateAsm:CSP3DTemplateDef"* to *"SP3DTank2Asm:CSP3DTank2Def"*

```
Private Const MODULE = "SP3DTank2Asm:CSP3DTank2Def"
```

15. Go to CSP3DTank2Def Class module and rename the project name and class name as shown below:

```
     m_oEquipCADHelper.ProjectName = "SP3DTank2Asm"
    m_oEquipCADHelper.ClassName = "CSP3DTank2Def"
```

16. Go to CSimplePhysical Class module and declare all variables for your inputs and outputs:

```
    Dim parLength As Double
    Dim ObjBody1 As Object

    Dim pipeDiam       As Double
    Dim flangeThick    As Double
    Dim cptOffset      As Double
    Dim flangeDiam     As Double
    Dim depth          As Double
```

17. Uses these variables to store the inputs as follows:

```
' Insert your code for inputs
    Set oPartFclt = arrayOfInputs(1)
    parLength = arrayOfInputs(2)
```

18. Go to the Insert your code for output (Body1) section. The following code will use the PlaceBox() routine to create a Box for the Body1. The PlaceBox routine is located at geometry3d.bas module. This function takes the two opposite corner points of the box as input parameters.

```
' Insert your code for output (Body1)
    Dim pPos1 As IJDPosition
    Dim pPos2 As IJDPosition

    Set pPos1 = New DPosition
    Set pPos2 = New DPosition

    pPos1.Set -parLength / 2, -parLength / 2, -parLength / 2
    pPos2.Set parLength / 2, parLength / 2, parLength / 2
    Set ObjBody1 = PlaceBox(m_OutputColl, pPos1, pPos2)
    iOutput = iOutput + 1
    m_OutputColl.AddOutput arrayOfOutputs(iOutput), ObjBody1
```

19. Declare the following variables to store the nozzle object, nozzle position and nozzle orientation.

```
    Dim oDir       As AutoMath.DVector
    Dim objNozzle  As IJDNozzle
    Set oDir = New AutoMath.DVector
    Dim CenterPos      As New AutoMath.DPosition

    RetrieveParameters 1, oPartFclt, m_OutputColl, pipeDiam, flangeThick, flangeDiam, cptOffset, depth

    CenterPos.Set 0, 0, parLength - depth + cptOffset
    oDir.Set 0, 0, 1
```

20. Use the CreateNozzlePHWithLength() to define the placeholder.

```
    Set objNozzle = CreateNozzlePHWithLength(1, oPartFclt, m_OutputColl, oDir, _
                                          CenterPos, parLength - depth + cptOffset)

' Set the output
    iOutput = iOutput + 1
```

```
m_OutputColl.AddOutput arrayOfOutputs(iOutput), objNozzle
```

21. Use the Set statement to clear the references from all object variables.

```
'Release BO 's
'
   Set ObjBody1 = Nothing
   Set pPos1 = Nothing
   Set pPos2 = Nothing
   Set objNozzle = Nothing
   Set CenterPos = Nothing
   Set oDir = Nothing
```

22. Go to CSP3DTank2Def Class module. Declare the appropriate custom methods to manage the pipe nozzle as follows

```
'Add your code here for the declaration of the Public Custom Methods used to manage new members
'Add new member(NozzleN1) to the definition

Set oMemberDescription = Nothing
Set oMemberDescription = oMemberDescriptions.AddMember("NozzleN1", 1, _
        "CMConstructNozzleN1", imsCOOKIE_ID_USS_LIB)
oMemberDescription.SetCMSetInputs imsCOOKIE_ID_USS_LIB, "CMSetInputsNozzleN1"
oMemberDescription.SetCMFinalConstruct imsCOOKIE_ID_USS_LIB, "CMFinalConstructNozzleN1"
oMemberDescription.SetCMConditional imsCOOKIE_ID_USS_LIB, "CMConditionalNozzleN1"
oMemberDescription.SetCMCount imsCOOKIE_ID_USS_LIB, "CMCountNozzleN1"
oMemberDescription.SetCMRelease imsCOOKIE_ID_USS_LIB, "CMReleaseNozzleN1"

'Add properties for (NozzleN1)
Set oPropertyDescriptions = Nothing
Set oPropertyDescriptions = oMemberDescription
oPropertyDescriptions.AddProperty "NozzleN1Properties", 1, _
       IID_IJDATTRIBUTES, "CMEvaluateNozzleN1", imsCOOKIE_ID_USS_LIB
oPropertyDescriptions.AddProperty "NozzleN1GeometryProperties", 2,  _
       IID_IJDGEOMETRY, "CMEvaluateGeometryNozzleN1", imsCOOKIE_ID_USS_LIB
```

23. Go to IJEquipUserAttrMgmt_OnAttributeChange function and add the following code. This OnAttributeChange method is called each time an attribute is changed. When the “Can be Deleted” property is changed, the MakeMemberDeletable should be called to make the member deletable.

```
Private Function IJEquipUserAttrMgmt_OnAttributeChange(ByVal pIJDAttrs As IJDAttributes, ByVal
CollAllDisplayedValues As Object, ByVal pAttrToChange As IJEquipAttrDescriptor, ByVal varNewAttrValue
As Variant) As String
   Const METHOD = "IJEquipUserAttrMgmt_OnAttributeChange"
   On Error GoTo ErrorHandler

' Add code here

   Dim oMemberDescription As IJDMemberDescription
   Set oMemberDescription = m_oEquipCADHelper.GetMemberDescriptionFromChild(pIJDAttrs)

   Select Case oMemberDescription.Name
      Case "NozzleN1"
```

```
         Select Case UCase(pAttrToChange.InterfaceName)
            Case "IJDELETABLEMEMBER"
               If UCase(pAttrToChange.AttrName) = "CANBEDELETED" Then
                           m_oEquipCADHelper.MakeMemberDeletable oMemberDescription, pIJDAttrs,  _
                              CBool(varNewAttrValue)
               End If
            Case Else
               '
         End Select
      Case Else
         '
   End Select

   Set oMemberDescription = Nothing
   IJEquipUserAttrMgmt_OnAttributeChange = ""

   Exit Function
ErrorHandler:
   IJEquipUserAttrMgmt_OnAttributeChange = "ERROR"
   HandleError MODULE, METHOD
End Function
```

24. Go to the end of CSP3DTank2Def Class module. Add the custom methods to manage the pipe nozzle as follows:

    Custom Method Construct:
    This method is in charge of the creation of the CAO member object (pipe nozzle). Use CreateNozzleFromPH() method to create a pipe nozzle from a nozzle place holder defined in the equipment symbol.

```
' Custom Methods for NozzleN1
Public Sub CMConstructNozzleN1(ByVal pMemberDescription As IJDMemberDescription, _
                         ByVal pResourceManager As IUnknown, _
                         ByRef pObject As Object)
   Const METHOD = "CMConstructNozzleN1"
   On Error GoTo ErrorHandler

   'Create Nozzle
   m_oEquipCADHelper.CreateNozzleFromPH pMemberDescription, pResourceManager, pObject, 1

   Exit Sub
ErrorHandler:
   HandleError MODULE, METHOD
End Sub
```

    Custom Method Final:
    There is no need to add any code for this custom method

```
Public Sub CMFinalConstructNozzleN1(ByVal pMemberDesc As IJDMemberDescription)
   Const METHOD = "CMFinalConstructNozzleN1"
   On Error GoTo ErrorHandler
   Exit Sub

ErrorHandler:
    HandleError MODULE, METHOD
End Sub
```

Custom method Inputs:

There is no need to add any code for this custom method

```
Public Sub CMSetInputsNozzleN1(ByVal pMemberDesc As IJDMemberDescription)
    Const METHOD = "CMSetInputsNozzleN1"
    On Error GoTo ErrorHandler

    Exit Sub
ErrorHandler:
    HandleError MODULE, METHOD
End Sub
```

Custom method Evaluate:

There is no need to add any code for this custom method

```
Public Sub CMEvaluateNozzleN1(ByVal oPropertyDescription As IJDPropertyDescription, pObject As Object)
    Const METHOD = "CMEvaluateNozzleN1"
    On Error GoTo ErrorHandler

    Exit Sub
ErrorHandler:
    HandleError MODULE, METHOD
End Sub
```

Custom method EvaluateGeometry:

This custom method will keep the nozzle’s position as the same as the nozzle placeholder in the symbol.

```
Public Sub CMEvaluateGeometryNozzleN1(ByVal oPropertyDescription As IJDPropertyDescription, pObject
As Object)
    Const METHOD = "CMEvaluateGeometryNozzleN1"
    On Error GoTo ErrorHandler

    'Transform the nozzle so that it behaves like a rigid body inside the equipment
    m_oEquipCADHelper.TransformNozzleWrtPH oPropertyDescription, pObject, 1

    Exit Sub
ErrorHandler:
    HandleError MODULE, METHOD
End Sub
```

Custom method Conditional:

This method checks if the member is conditional based on the CanBeDeleted flag. Remember, we added code to make a member deletable in the IJEquipUserAttrMgmt_OnAttributeChange function.When the property is changed, the MakeMemberDeletable is called to check the CanBeDeleted flag and whether or not to make the member deletetable.

```
Public Sub CMConditionalNozzleN1(ByVal pMemberDesc As IJDMemberDescription, ByRef IsNeeded As
Boolean)
    Const METHOD = "CMConditionalNozzleN1"
    On Error GoTo ErrorHandler

    IsNeeded = m_oEquipCADHelper.CheckMemberConditional(pMemberDesc)

    Exit Sub
ErrorHandler:
    HandleError MODULE, METHOD
End Sub
```

Custom method Count:
There is no need to add any code for this custom method

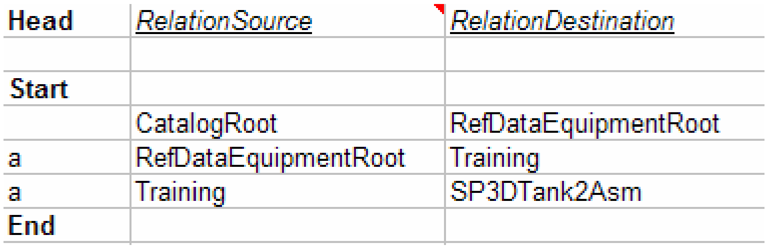
```
Public Sub CMCountNozzleN1(ByVal pMemberDesc As IJDMemberDescription, ByRef Count As Long)
    Const METHOD = "CMCountNozzleN1"
    On Error GoTo ErrorHandler

    Exit Sub
ErrorHandler:
    HandleError MODULE, METHOD
End Sub
```

Custom method Release:
There is no need to add any code for this custom method

```
Public Sub CMReleaseNozzleN1(ByVal pMemberDesc As IJDMemberDescription)
    Const METHOD = "CMReleaseNozzleN1"
    On Error GoTo ErrorHandler

    Exit Sub
ErrorHandler:
    HandleError MODULE, METHOD
End Sub
```

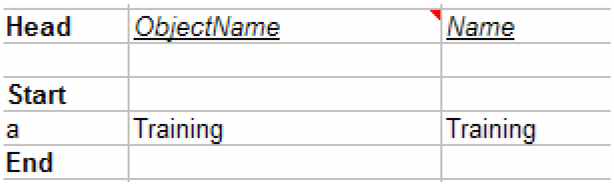
25. Compile the VB project and save the dll as SP3DTank2Asm.dll in the c:\Train\lab2
    One of the most important steps in Visual Basic programming is to preserve the binary compatibility of your program. Save the final version of your dll file to be binary compatibility in order to preserve the CLSID.

26. Save the VB SP3DTank2Asm project.
27. Open the SP3DTemplate.xls workbook. Go the R-Hierarchy sheet and add the following entry.



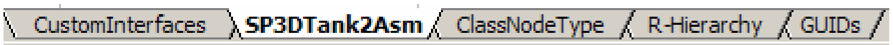
Head	RelationSource	RelationDestination
Start		
	CatalogRoot	RefDataEquipmentRoot
a	RefDataEquipmentRoot	Training
a	Training	SP3DTank2Asm
End		

28. Go the ClassNodeType sheet and add the following entry.



Head	ObjectName	Name
Start		
a	Training	Training
End		

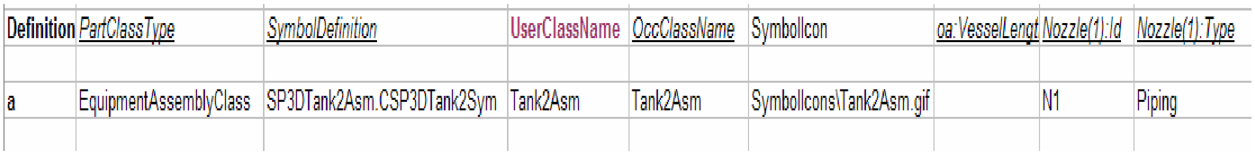
29. Go to the SP3DTemplateAsm sheet and rename it as SP3DTank2Asm.



CustomInterfaces | SP3DTank2Asm | ClassNodeType | R-Hierarchy | GUIDs

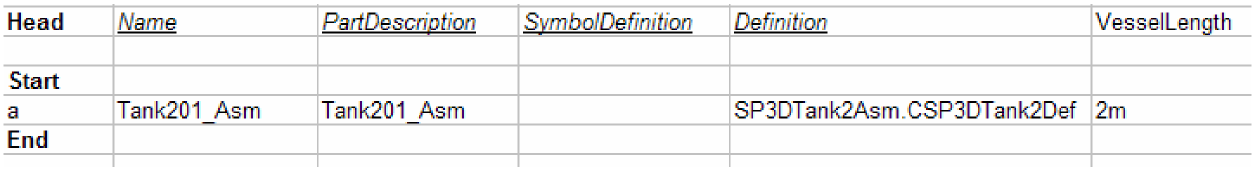
30. Go to the Class definition section and add/edit as follows:

In the Definition Section:

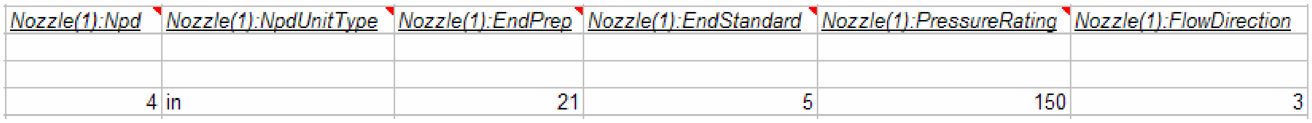


Definition	PartClassType	SymbolDefinition	UserClassName	OccClassName	SymbolIcon	oa:VesselLengt	Nozzle(1):Id	Nozzle(1):Type
a	EquipmentAssemblyClass	SP3DTank2Asm.CSP3DTank2Sym	Tank2Asm	Tank2Asm	SymbolIcons\Tank2Asm.gif		N1	Piping

In the Part Section:



Head	Name	PartDescription	SymbolDefinition	Definition	VesselLength
Start					
a	Tank201_Asm	Tank201_Asm		SP3DTank2Asm.CSP3DTank2Def	2m
End					



Nozzle(1):Npd	Nozzle(1):NpdUnitType	Nozzle(1):EndPrep	Nozzle(1):EndStandard	Nozzle(1):PressureRating	Nozzle(1):FlowDirection
4	in	21	5	150	3

31. Save the Excel workbook as SP3DTank2Asm.xls in the c:\Train\lab2.
32. Create the Tank2Asm.gif file and place it under \\<MachineName>\Symbols\SymbolIcons
33. Load the information into the Catalog using the A/M/D Mode. Once the bulkload process is completed, review the log file. Next, run the View Generator utility on the model to re-create the views in the model database. Finally, Re-generate the report databases.
34. Go to the Equipment Task and place the SP3DTank2Asm.

Appendix

Symbol Helper Reference

The Symbol Helper Reference provides documentation for symbol math functions and properties.

IJSymbolHelper

This interface provides methods to help in creating the definition of a VB symbol. It provides the implementation of the IJDUserSymbolServices interface as well as provides support for declaring the inputs and outputs of the symbol. Call this interface when you want to:

- Instantiate a symbol definition in a datastore.
- Update an existing symbol definition.
- Compute the symbol using a function.
- Edit the symbol occurrence.

Methods

AddInputDef(Count As Integer, Name As String, Description As String, DefaultValue As Double)	
Description:	Adds the input definition to the collection of inputs defined for the symbol
Parameters:	
[in] count	Index for the input parameter
[in] Name	Name of the input parameter
[in] Description	Description of the input parameter
[in] DefaultValue	Default value for the input parameter

AddOutputDef(Count As Integer, Name As String, Description As String, aspect as integer)	
Description:	Adds the output definition to the collection of outputs defined for the symbol
Parameters:	
[in] count	Index for the output parameter
[in] Name	Name of the output parameter
[in] Description	Description of the output parameter
[in] aspect	Aspect number for the output

AddAspectDef (Count As Integer, Name As String, Description As String, aspect as integer)	
Description:	Adds the aspect definition to the symbol
Parameters:	
[in] count	Index for the aspect
[in] Name	Name of the aspect
[in] Description	Description of the aspect
[in] aspect	Aspect number for the output

InstanciateDefinition (ByVal CodeBase As String, ByVal defParameters As Variant, ByVal ActiveConnection As Object)	
Description:	This method will create a symbol definition entity and initialize it. It will also set the progid and the code base values on the definition. It will take the same set of parameters as the method on the interface 'IJDUserSymbolServices'.
Parameters:	
[in] CodeBase	Specifies the URL (or UNC) of the .cab file that can provides the dll associated to the symbol definition object (ActiveX control packaging).
[in] defParameters	Definition parameters.
[in] ActiveConnection	Resource manager to which the symbol definition will be connected

InitializeSymbolDefinition(ByRef pSymbolDefinition As IJDSymbolDefinition)	
Description:	This method will define the inputs for the symbol definition, define the required number of representations and add the outputs defined to the correct representation. The input collection as well as the output collection can be made a 'VARIABLECOLLECTION' if required.
Parameters:	
pSymbolDefinition	Symbol definition passed by reference that will be initialized in this method.

InvokeRepresentation(ByVal sblOcc As Object, ByVal repName As String, ByVal outputcoll As Object, ByRef arrayOfInputs())	
Description:	This method will create the object that contains the implementation details for the required representation. The wizard follows a specific convention like so: ProjectName.<RepresentationName>. So the helper function can obtain the progid given this rule and create the object and then call the method 'Run' on the IDispatch interface of this object. This method will also take all the parameters in addition to an array of strings that contain the names of outputs belonging to that representation.
Parameters:	
[in] sblOcc	Symbol occurrence that calls the method.
[in] repName	Name of the representation requested on the symbol.
[in] outputcoll	Collection object to which the generated outputs will be attached.
[in] arrayOfInputs	A safearray of inputs defined as VARIANT.

Properties

NumInputs as Integer	
Description:	Number of inputs for the symbol
Modifiability:	Read/write

NumOutputs as Integer	
Description:	Number of outputs for the symbol.
Modifiability:	Read/write

NumAspects as Integer	
Description:	Number of aspects defined for the symbol
Modifiability:	Read/write

ProjectName as String	
Description:	Project Name for the symbol
Modifiability:	Read/write

ClassName as String	
Description:	Class name for the symbol
Modifiability:	Read/write

IJSymbolGeometryHelper

This interface provides methods to help in creating simple geometric primitives like Cylinder (given center, radius and length), Cone (given the 4 points), Sphere (center and radius), Torus (center, major radius, minor radius). The other geometric primitives are not yet implemented.

Methods

AddGeometry(Output As String, Aspect As Long, Geometry As Object)	
Description:	Adds the Geometry Object to the Output Collection.
Parameters:	
[in] Output	Required Output as string
[in] Aspect	Required long value
[in] Geometry	Required Object Geometry

CreateChildPartOcc(Output As String, ChildPart As Object, Position As IJDPosition, VecX As IJDVector, VecY As IJDVector, VecZ As IJDVector) As Object	
Description:	
Parameters:	
[in] Output	Required Output as string
[in] ChildPart	Required Object ChildPart
[in] Position	Required IJDPosition Position
[in] VecX	Required IJDVector VecX
[in] VecY	Required IJDVector VecY
[in] VecZ	Required IJDVector VecZ

CreateCone(Output As String, PosStart As IJDPosition, PosEnd As IJDPosition, diameterStart As Double, diameterEnd As Double, Optional Offset As Double = 0#) As Object	
Description:	Creates the Cone Object and adds it to the output collection
Parameters:	
[in] Output	Required Output as string
[in] PosStart	Required IJDPosition Start
[in] PosEnd	Required IJDPosition End
[in] diameterStart	Required double value
[in] diameterEnd	Required double value
[in, defaultvalue(0)] Offset	Optional double value – is an optional parameter

CreateCylinder(Output As String, PosStart As IJDPosition, PosEnd As IJDPosition, Diameter As Double) As Object	
Description:	Creates the Cylinder Object and adds it to the output collection
Parameters:	
[in] Output	Required Output as string
[in] PosStart	Required IJDPosition Start
[in] PosEnd	Required IJDPosition End
[in] Diameter	Required double value – diameter of the Cylinder

CreateMiteredTorus(Output As String, Origin As IJDPosition, NormalAxis As IJDVector, MajorAxis As IJDVector, Radius As Double, Angle As Double, Diameter As Double, NumberOfCuts As Long) As Object	
Description:	Creates the CreateMiteredTorus Object and adds it to the output collection
Parameters:	
[in] Output	Required Output as string
[in] Origin	Required IJDPosition Origin
[in] NormalAxis	Required IJDVector NormalAxis
[in] MajorAxis	Required IJDVector MajorAxis
[in] Radius	Required double value
[in] Angle	Required double value
[in] Diameter	Required double value
[in] NumberOfCuts	Required long value

CreatePolygon(Output As String, NumberOfSides As Long, SideLength As Double, Depth As Double, Object As Object)	
Description:	Creates the CreatePolygon Object and adds it to the output collection
Parameters:	
[in] Output	Required Output as string
[in] NumberOfSides	Required long value
[in] SideLength	Required double value
[in] Depth	Required double value

CreatePrism(Output As String, Width As Double, Depth As Double, Length As Double, Width2 As Double, Depth2 As Double, Optional Offset As Double = 0#) As Object	
Description:	Creates the CreatePrism Object and adds it to the output collection
Parameters:	
[in] Output	Required Output as string
[in] Width	Required double value
[in] Depth	Required double value
[in] Length	Required double value
[in] Width2	Required double value
[in] Depth2,	Required double value
[in, defaultvalue(0)] Offset	Optional double value

CreateProjectedRectangle(Output As String, PosStart As IJDPosition, PosEnd As IJDPosition, Axis As IJDVector, Width As Double, Depth As Double) As Object	
Description:	Creates the CreateProjectedRectangle Object and adds it to the output collection
Parameters:	
[in] Output	Required Output as string
[in] PosStart	Required IJDPosition Start
[in] PosEnd	Required IJDPosition End
[in] Axis	Required IJDVector Axis

[in] Width	Required double value
[in] Depth	Required double value

CreateProjectedShape(Output As String, Length As Double, Curve As Object) As Object	
Description:	Creates the CreateProjectedShape Object and adds it to the output collection
Parameters:	
[in] Output	Required Output as string
[in] Length	Required double value
[in] Curve	Required object curve

CreateProjectedShapeByPoints(Output As String, NumberOfPoints As Long, Length As Double, Points As IJElements) As Object	
Description:	Creates the CreateProjectedShapeByPoints Object and adds it to the output collection
Parameters:	
[in] Output	Required Output as string
[in] NumberOfPoints	Required long value
[in] Length	Required double value
[in] Points	Required point objects as IJElements collection

CreateProjectedTriangle(Output As String, PosStart As IJDPosition, PosEnd As IJDPosition, Axis As IJDVector, Width As Double, Depth As Double) As Object	
Description:	Creates the CreateProjectedTriangle Object and adds it to the output collection
Parameters:	
[in] Output	Required Output as string
[in] PosStart	Required IJDPosition start
[in] PosEnd	Required IJDPostion end
[in] Axis	Required IJDVector Axis
[in] Width	Required double value
[in] Depth	Required double value

CreateRectangularTorus(Output As String, Radius As Double, SweepAngle As Double, Width As Double, Depth As Double) As Object	
Description:	Creates the CreateRectangularTorus Object and adds it to the output collection
Parameters:	
[in] Output	Required Output as string
[in] Radius	Required double value
[in] SweepAngle	Required double value
[in] Width	Required double value
[in] Depth	Required double value

CreateSemiEllipsoid(Output As String, Origin As IJDPosition, NormalAxis As IJDVector, MajorAxis As IJDVector, AxisDiameter As Double, MinorAxisRadius As Long) As Object	
Description:	Creates the CreateSemiEllipsoid Object and adds it to the output collection
Parameters:	

[in] Output	Required Output as string
[in] Origin	Required IJDPosition Origin
[in] NormalAxis	Required IJDVector NormalAxis
[in] MajorAxis	Required IJDVector MajorAxis
[in] AxisDiameter	Required double value
[in] MinorAxisRadius	Required long value

CreateSphere(Output As String, Origin As IJDPosition, Radius As Double) As Object	
Description:	Creates the CreateSphere Object and adds it to the output collection
Parameters:	
[in] Output	Required Output as string
[in] Origin	Required IJDPosition Origin
[in] Radius	Required double value

CreateTorus(Output As String, Origin As IJDPosition, NormalAxis As IJDVector, MajorAxis As IJDVector, Radius As Double, Angle As Double, Diameter As Double) As Object	
Description:	Creates the CreateTorus Object and adds it to the output collection
Parameters:	
[in] Output	Required Output as string
[in] Origin	Required IJDPosition Origin
[in] NormalAxis	Required IJDVector NormalAxis
[in] MajorAxis	Required IJDVector MajorAxis
[in] Radius	Required double value
[in] Angle	Required double value
[in] Diameter	Required double value

CreateTransitionalElement(Output As String, Width As Double, Depth As Double, Length As Double, Radius As Double, Offset As Double) As Object	
Description:	Creates the CreateTransitionalElement Object and adds it to the output collection
Parameters:	
[in] Output	Required Output as string
[in] Width	Required double value
[in] Depth	Required double value
[in] Length	Required double value
[in] Radius	Required double value
[in] Offset	Required double value

Properties

AutoTransformUpdate() As Boolean	
Description:	Adding or getting the AutoTransformUpdate boolean value
Modifiability:	Read/write

OutputCollection() As IJDOutputCollection	
Description:	Adding or getting created output objects in the output collection

Modifiability:	Read/write

Transform() As IJDT4x4	
Description:	Adding or getting the transformation matrix IJDT4x4
Modifiability:	Read/write

Geometry Factory Programming Reference

The Geometry Factory Programming Reference provides documentation of Geom3d.dll, which includes the objects, methods, and properties for the geometry factory.

Description

The GeometryFactory object is the class factory for the creation of geometry entities. The factory implements properties that return "collection-like" interfaces for each of the geometry types. These interfaces have creation methods that the application programmer can use to create, initialize, and optionally specify a persistent database connection for the object.

If the objects are created with a NULL database connection, the object is created as a "transient." Transient objects can be displayed and added to the highlight system, but they do not participate in transactions or relationships.

IJGeometryFactory

Use this interface when you want to create transient or persistent geometry objects

Properties

Points3d ( ) as IPoints3d	
Description:	Returns a pointer (pVal) to the IPoints3d interface of the first element in the collection.
Modifiability:	Read Only

Lines3d ( ) as ILines3d	
Description:	Returns a pointer (pVal) to the ILines3d interface of the first element in the collection.
Modifiability:	Read Only

Arcs3d ( ) as IArcs3d	
Description:	Returns a pointer (pVal) to the IArcs3d interface of the first element in the collection.
Modifiability:	Read Only

Circles3d ( ) as ICircles3d	
Description:	Returns a pointer (pVal) to the ICircles3d interface of the first element in the collection.
Modifiability:	Read Only

Ellipses3d ( ) as IEllipses3d	
Description:	Returns a pointer (pVal) to the IEllipses3d interface of the first element in the collection.
Modifiability:	Read Only

EllipticalArcs3d ( ) as IEllipticalArcs3d	
Description:	Returns a pointer (pVal) to the IEllipticalArcs3d interface of the first element in the collection.
Modifiability:	Read Only

LineStrings3d ( ) as ILineStrings3d	
Description:	Returns a pointer (pVal) to the ILineStrings3d interface of the first element in the collection.
Modifiability:	Read Only

BSplineCurves3d ( ) as IBSplineCurves3d	
Description:	Returns a pointer (pVal) to the IBSplineCurves3d interface of the first element in the collection.
Modifiability:	Read Only

ComplexStrings3d ( ) as IComplexStrings3d	
Description:	Returns a pointer (pVal) to the IComplexStrings3d interface of the first element in the collection.
Modifiability:	Read Only

Planes3d ( ) as IPlanes3d	
Description:	Returns a pointer (pVal) to the IPlanes3d interface of the first element in the collection.
Modifiability:	Read Only

Cones3d ( ) as ICones3d	
Description:	Returns a pointer (pVal) to the ICones3d interface of the first element in the collection.
Modifiability:	Read Only

Projections3d ( ) as IProjections3d	
Description:	Returns a pointer (pVal) to the IProjections3d interface of the first element in the collection.
Modifiability:	Read Only

Revolutions3d ( ) as IRevolutions3d	
Description:	Returns a pointer (pVal) to the IRevolutions3d interface of the first element in the collection.
Modifiability:	Read Only

RuledSurfaces3d ( ) as IRuledSurfaces3d	
Description:	Returns a pointer (pVal) to the IRuledSurfaces3d interface of the first element in the collection.
Modifiability:	Read Only

Spheres3d ( ) as ISpheres3d	
Description:	Returns a pointer (pVal) to the ISpheres3d interface of the first element in the collection.
Modifiability:	Read Only

Tori3d ( ) as ITori3d	
Description:	Returns a pointer (pVal) to the ITori3d interface of the first element in the collection.
Modifiability:	Read Only

BSplineSurfaces3d ( ) as IBSplineSurfaces3d	
Description:	Returns a pointer (pVal) to the IBSplineSurfaces3d interface of the first element in the collection.
Modifiability:	Read Only

Methods:

CreateBSplineSurfaceByParametersWCaps Method

Description

The CreateBSplineSurfaceByParametersWCaps method creates and returns a BSplineSurface3d object based on a desired order, a set of poles, and optional caps. Weights and knots are optional and are set to NULL, or an empty array. The output will be the surface, then the caps.

If the order is equal to the number of poles, the curve evolves into the control polygon of a Bezier curve.

B-spline weights can be considered a gravitational type force with the magnitude of the weight equal to the pulling force. The weights are always normalized. If no weights are present, the curve is considered to be non-rational and may be NULL. Non-rational curves have weights with a value of 1.

The B-spline knots define the parameterization of the curve, and they may be periodic. Knots, also known as knot vectors, must be monotonic and strictly increasing. Monotonic refers to the successive terms as non-decreasing or non-increasing.

The Order property determines the relative accuracy of the poles with regard to the points that are entered to create the curve. The order returned evaluates as a polynomial degree plus one. For example, an order of 4 defines cubic. Since it is more efficient to use even-order b-spline curves, the number of poles (and knots) are maximized by increasing the order to the next even number.

Syntax

object.CreateBSplineSurfaceByParametersWCaps*(pConnection, uNumPoles, vNumPoles, Poles, Weights, uOrder, vOrder, uKnots, vKnots, uPeriodic, vPeriodic, ReverseNor, Solid, WCaps, numCaps)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
uNumPoles	long	Required. This argument is the number of poles in the u-direction. The type is long.
vNumPoles	long	Required. This argument is the number of poles in the v-direction. The type is long.
Poles	double	Required. This argument is a SAFEARRAY of poles. The type is double.
Weights	double	Required. This argument is a SAFEARRAY of weights. The type is double.
uOrder	long	Required. This argument is the order in the u-direction. The type is long.
vOrder	long	Required. This argument is the order in the v-direction. The type is long.
uKnots	double	Required. This argument is a SAFEARRAY of knots. The type is double.
vKnots	double	Required. This argument is a SAFEARRAY of Knots. The type is double.
uPeriodic	Boolean	Required. This argument is a Boolean flag that specifies whether the surface is periodic in u.
vPeriodic	Boolean	Required. This argument is a Boolean flag that specifies whether or not the surface is periodic in v.
ReverseNor	Boolean	Required. This argument specifies the outward normal. It is False when the outward normal is U X V. It is True when the outward normal is U (curve) cross V (proj vector). The type is Boolean.
Solid	Boolean	Required. This argument is a Boolean flag that specifies whether or not the object is solid. Possible values are: 0 - Set normal as hollow; 1 - Set normal as solid; 2 - Set normal according to input point; 3 - Just toggle the outward normal (no checks).
WCaps	Boolean	Required. This argument specifies whether or not the object has caps. If the value is False, the surface does not have caps; if the value is True, the surface has caps.
numCaps	Int	Required. This argument is the number of caps. The type is integer.

CreateBy2Points Method

Description

The CreateBy2Points method creates and returns a Line3d object defined by two points.

Syntax

object.CreateBy2Points*(pConnection, StartX, StartY, StartZ, EndX, EndY, EndZ)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
StartX	double	Required. This argument is the X-coordinate for the starting point. The type is double.
StartY	double	Required. This argument is the Y-coordinate for the starting point. The type is double.
StartZ	double	Required. This argument is the Z-coordinate for the starting point. The type is double.
EndX	double	Required. This argument is the X-coordinate for the ending point. The type is double.
EndY	double	Required. This argument is the Y-coordinate for the ending point. The type is double.
EndZ	double	Required. This argument is the Z-coordinate for the ending point. The type is double.

CreateBy3Points Method (IArcs3d)

Description

The CreateBy3Points method creates and returns an Arc3d object given three non-colinear points along the arc.

Syntax

object.CreateBy3Points*(pConnection, StartX, StartY, StartZ, AlongX, AlongY, AlongZ, EndX, EndY, EndZ)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
StartX	double	Required. This argument is the X-coordinate for the starting point on the arc. The type is double.
StartY	double	Required. This argument is the Y-coordinate for the starting point on the arc. The type is double.
StartZ	double	Required. This argument is the Z-coordinate for the starting point on the arc. The type is double.
AlongX	double	Required. This argument is the X-coordinate for the middle point on the arc. The type is double.
AlongY	double	Required. This argument is the Y-coordinate for the middle point on the arc. The type is double.
AlongZ	double	Required. This argument is the Z-coordinate for the middle point on the arc. The type is double.
EndX	double	Required. This argument is the X-coordinate for the ending point on the arc. The type is double.
EndY	double	Required. This argument is the Y-coordinate for the ending point on the arc. The type is double.
EndZ	double	Required. This argument is the Z-coordinate for the ending point on the arc. The type is double.

CreateBy3Points Method (ICircles3d)

Description

The CreateBy3Points method creates and returns a pointer (ppObj) to the IJCircle interface of a Circle3d object. This method uses three inscribed non-colinear points to create the circle.

Syntax

object.CreateBy3Points*(pConnection, X1, Y1, Z1, X2, Y2, Z2, X3, Y3, Z3)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
X1	double	Required. This argument is the first X-coordinate value. The type is double.
Y1	double	Required. This argument is the first Y-coordinate value. The type is double.
Z1	double	Required. This argument is the first Z-coordinate value. The type is double.
X2	double	Required. This argument is the second X-coordinate value. The type is double.
Y2	double	Required. This argument is the second Y-coordinate value. The type is double.
Z2	double	Required. This argument is the second Z-coordinate value. The type is double.
X3	double	Required. This argument is the third X-coordinate value. The type is double.
Y3	double	Required. This argument is the third Y-coordinate value. The type is double.
Z3	double	Required. This argument is the third Z-coordinate value. The type is double.

CreateBy4Pts Method

Description

The CreateBy4Pts method creates and returns a pointer (ppObj) to the IJCone interface of a full bounded Cone3d. This method takes as input a base center point, a top center point, a base starting point, and a top starting point. The axis runs through the top center point and base center point, and the cone follows the right-hand rule about the axis.

The base ellipse must not be degenerate, so the base center point cannot be the same as the base starting point.

To create a point cone, set the top center point to the top starting point.

Syntax

object.CreateBy4Pts*(pConnection, CenterBx, CenterBy, CenterBz, CenterTx, CenterTy, CenterTz, StartBx, StartBy, StartBz, StartTx, StartTy, StartTz, Solid)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
CenterBx	double	Required. This argument is the X-coordinate of the base center point. The type is double.
CenterBy	double	Required. This argument is the Y-coordinate of the base center point. The type is double.
CenterBz	double	Required. This argument is the Z-coordinate of the base center point. The type is double.
CenterTx	double	Required. This argument is the X-coordinate of the top center point. The type is double.
CenterTy	double	Required. This argument is the Y-coordinate of the top center point. The type is double.

CenterTz	double	Required. This argument is the Z-coordinate of the top center point. The type is double.
StartBx	double	Required. This argument is the X-coordinate of the base starting point. The type is double.
StartBy	double	Required. This argument is the Y-coordinate of the base starting point. The type is double.
StartBz	double	Required. This argument is the Z-coordinate of the base starting point. The type is double.
StartTx	double	Required. This argument is the X-coordinate of the top starting point. The type is double.
StartTy	double	Required. This argument is the Y-coordinate of the top starting point. The type is double.
StartTz	double	Required. This argument is the Z-coordinate of the top starting point. The type is double.
Solid	Boolean	Required. This argument is a Boolean flag indicating whether the cone is solid or not.

CreateByAxisMajorMinorRadius Method

Description
The CreateByAxisMajorMinor method creates and returns a pointer (ppObj) to the IJTorus interface of a Torus3d object. This method defines a torus by a point on the axis at the center of the torus, an axis vector, a vector toward the center of a minor circle (determining the origin of UV space), a major radius, and a minor radius. Set major radius = -major radius if the center of the torus is on the left-hand side of the axis, indicating the torus is a lemon shape.

Syntax
object.CreateByAxisMajorMinorRadius*(pConnection, AxisCenterX, AxisCenterY, AxisCenterZ, AxisVecX, AxisVecY, AxisVecZ, OriginDirX, OriginDirY, OriginDirZ, MajorRadius, MinorRadius, Solid)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
AxisCenterX	double	Required. This argument is the X-coordinate of the point on the center axis. The type is double.
AxisCenterY	double	Required. This argument is the Y-coordinate of the point on the center axis. The type is double.
AxisCenterZ	double	Required. This argument is the Z-coordinate of the point on the center axis. The type is double.
AxisVecX	double	Required. This argument is the X-coordinate of a point along the axis vector. The type is double.
AxisVecY	double	Required. This argument is the Y-coordinate of a point along the axis vector. The type is double.
AxisVecZ	double	Required. This argument is the Z-coordinate of a point along the axis vector. The type is double.
OriginDirX	double	Required. This argument is the X-coordinate of a point along the vector toward the center of the minor circle. The type is double.
OriginDirY	double	Required. This argument is the Y-coordinate of a point along the vector toward the center of the minor circle. The type is double.
OriginDirZ	double	Required. This argument is the Z-coordinate of a point along the vector toward the center of the minor circle. The type is double.
MajorRadius	double	Required. This argument is the length of the major radius. The type is double.
MinorRadius	double	Required. This argument is the length of the minor radius. The type is double.
Solid	Boolean	Required. This argument is a Boolean flag indicating whether or not the torus is solid.

CreateByAxisMajorMinorRadiusSweep Method

Description

The CreateByAxisMajorMinorRadiusSweep method creates and returns a pointer (ppObj) to the IJTorus interface of a Torus3d object. This method defines a partial torus by a point on the axis at the center of the torus, an axis vector, a vector toward the center of the minor circle (determining the origin of UV space), a major radius, a minor radius, and a sweep angle. Set the major radius = -major radius if the center of the torus is on the left-hand side of the axis, indicating the torus is a lemon shape.

Syntax

object.CreateByAxisMajorMinorRadiusSweep*(pConnection, AxisCenterX, AxisCenterY, AxisCenterZ, AxisVecX, AxisVecY, AxisVecZ, OriginDirX, OriginDirY, OriginDirZ, MajorRadius, MinorRadius, SwAngle, Solid)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
AxisCenterX	double	Required. This argument is the X-coordinate of a point on the center axis.The type is double.
AxisCenterY	double	Required. This argument is the Y-coordinate of a point on the center axis.The type is double.
AxisCenterZ	double	Required. This argument is the Z-coordinate of a point on the center axis.The type is double.
AxisVecX	double	Required. This argument is the X-coordinate of a point along the axis vector. The type is double.
AxisVecY	double	Required. This argument is the Y-coordinate of a point along the axis vector. The type is double.
AxisVecZ	double	Required. This argument is the Z-coordinate of a point along the axis vector. The type is double.
OriginDirX	double	Required. This argument is the X-coordinate of a point along the vector toward the center of the minor circle. The type is double.
OriginDirY	double	Required. This argument is the Y-coordinate of a point along the vector toward the center of the minor circle. The type is double.
OriginDirZ	double	Required. This argument is the Z-coordinate of a point along the vector toward the center of the minor circle. The type is double.
MajorRadius	double	Required. This argument is the length of the major radius. The type is double.
MinorRadius	double	Required. This argument is the length of the minor radius. The type is double.
SwAngle	double	Required. This argument is the sweep angle in radians. The type is double.
Solid	Boolean	Required. This argument is a Boolean flag indicating whether or not the torus is a solid.

CreateByCenterAxisRadEnds Method

Description

The CreateByCenterAxisRadEnds method creates and returns a pointer (ppObj) to the IJCone interface of a bounded partial Cone3d. This method takes as input a base center point, axis, base starting point, base ending point, and a top radius.

The cone follows the right-hand rule about the axis.

The axis vector must contain the height of the cylinder.

The base ellipse must not be degenerate, so the base center point cannot be the same as the base starting point.

To create a point cone, set the top radius length to zero.

Syntax

object.CreateByCenterAxisRadEnds*(pConnection, CenterBx, CenterBy, CenterBz, AxisVx, AxisVy, AxisVz, RadiusT, StartBx, StartBy, StartBz, EndBx, EndBy, EndBz, Solid)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
CenterBx	double	Required. This argument is the X-coordinate of the base center point. The type is double.
CenterBy	double	Required. This argument is the Y-coordinate of the base center point. The type is double.
CenterBz	double	Required. This argument is the Z-coordinate of the base center point. The type is double.
AxisVx	double	Required. This argument is the X-coordinate of a point on the axis vector. The type is double.
AxisVy	double	Required. This argument is the Y-coordinate of a point on the axis vector. The type is double.
AxisVz	double	Required. This argument is the Z-coordinate of a point on the axis vector. The type is double.
RadiusT	double	Required. This argument is the top radius value. The type is double.
StartBx	double	Required. This argument is the X-coordinate of the base starting point. The type is double.
StartBy	double	Required. This argument is the Y-coordinate of the base starting point. The type is double.
StartBz	double	Required. This argument is the Z-coordinate of the base starting point. The type is double.
EndBx	double	Required. This argument is the X-coordinate of the base ending point. The type is double.
EndBy	double	Required. This argument is the Y-coordinate of the base ending point. The type is double.
EndBz	double	Required. This argument is the Z-coordinate of the base ending point. The type is double.
Solid	Boolean	Required. This argument is a Boolean flag indicating whether the cone is solid or not.

CreateByCenterNormalMajAxisRatioAngle Method

Description
The CreateByCenterNormalMajAxisRatioAngle method creates and returns an EllipticalArc3d object given a center point, normal axis, major axis containing length, minor/major ratio, start angle, and sweep angle (angles in radians).

Syntax
object.CreateByCenterNormalMajAxisRatioAngle*(pConnection, CenterX, CenterY, CenterZ, NormalX, NormalY, NormalZ, MajorX, MajorY, MajorZ, MMRatio, StartAngle, SwAngle)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
CenterX	double	Required. This argument is the X-coordinate of the center point. The type is double.
CenterY	double	Required. This argument is the Y-coordinate of the center point. The type is double.
CenterZ	double	Required. This argument is the Z-coordinate of the center point. The type is double.
NormalX	double	Required. This argument is the X-coordinate of a point on the normal vector. The type is double.
NormalY	double	Required. This argument is the Y-coordinate of a point on the normal vector. The type is double.
NormalZ	double	Required. This argument is the Z-coordinate of a point on the normal vector. The type is double.

MajorX	double	Required. This argument is the X-coordinate of a point on the major axis vector. The type is double.
MajorY	double	Required. This argument is the Y-coordinate of a point on the major axis vector. The type is double.
MajorZ	double	Required. This argument is the Z-coordinate of a point on the major axis vector. The type is double.
MMRatio	double	Required. This argument is the minor axis to major axis ratio. The type is double.
StartAngle	double	Required. This argument is the start angle in radians. The type is double.
SwAngle	double	Required. This argument is the sweep angle in radians. The type is double.

CreateByCenterNormalRadius Method

Description

The CreateByCenterNormalRadius method creates and returns a pointer (ppObj) to an IJCircle interface of a Circle3d object, given the center, normal unit vector, and radius.

Syntax

object.CreateByCenterNormalRadius*(pConnection, CenterX, CenterY, CenterZ, NormalX, NormalY, NormalZ, Radius)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
CenterX	double	Required. This argument is the X-coordinate of the center of the circle. The type is double.
CenterY	double	Required. This argument is the Y-coordinate of the center of the circle. The type is double.
CenterZ	double	Required. This argument is the Z-coordinate of the center of the circle. The type is double.
NormalX	double	Required. This argument is the X-coordinate of a point on the normal vector. The type is double.
NormalY	double	Required. This argument is the Y-coordinate of a point on the normal vector. The type is double.
NormalZ	double	Required. This argument is the Z-coordinate of a point on the normal vector. The type is double.
Radius	double	Required. This argument is the radius of the circle. The type is double.

CreateByCenterNormMajAxisRatio Method

Description

The CreateByCenterNormMajAxisRatio method creates and returns a pointer (ppObj) to the IJEllipse interface of an Ellipse3d object, given a center point, normal axis, major axis containing length, and minor/major ratio.

Syntax

object.CreateByCenterNormMajAxisRatio*(pConnection, CenterX, CenterY, CenterZ, NormalX, NormalY, NormalZ, MajorX, MajorY, MajorZ, MMRatio)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
CenterX	double	Required. This argument is the X-coordinate of the center point. The type is double.
CenterY	double	Required. This argument is the Y-coordinate of the center point. The type is double.

CenterZ	double	Required. This argument is the Z-coordinate of the center point. The type is double.
NormalX	double	Required. This argument is the X-coordinate of a point on the normal vector. The type is double.
NormalY	double	Required. This argument is the Y-coordinate of a point on the normal vector. The type is double.
NormalZ	double	Required. This argument is the Z-coordinate of a point on the normal vector. The type is double.
MajorX	double	Required. This argument is the X-coordinate of a point on the major axis vector. The type is double.
MajorY	double	Required. This argument is the Y-coordinate of a point on the major axis vector. The type is double.
MajorZ	double	Required. This argument is the Z-coordinate of a point on the major axis vector. The type is double.
MMRatio	double	Required. This argument is the minor axis to major axis ratio. The type is double.

CreateByCenterRadius Method

Description
The CreateByCenterRadius method creates and returns a pointer (ppObj) to the IJSphere interface of a Sphere3d object, based on a center point and a radius.

Syntax
object.CreateByCenterRadius*(pConnection, CenterX, CenterY, CenterZ, Radius, Solid)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
CenterX	double	Required. This argument is the X-coordinate of the center point. The type is double.
CenterY	double	Required. This argument is the Y-coordinate of the center point. The type is double.
CenterZ	double	Required. This argument is the Z-coordinate of the center point. The type is double.
Radius	double	Required. This argument is the length of the radius. The type is double.
Solid	Boolean	Required. This argument is a Boolean flag indicating whether or not the sphere is solid.

CreateByCenterStartEnd Method

Description
The CreateByCenterStartEnd method creates an Arc3d object according to the specified inputs.
The center and start coordinates define the radius. A non-colinear ending point defines the sweep angle and plane (this returns an arc between 0 and P1).

Syntax
object.CreateByCenterStartEnd*(pConnection, CenterX, CenterY, CenterZ, StartX, StartY, StartZ, EndX, EndY, EndZ)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
CenterX	double	Required. This argument is the X-coordinate for the center point on the arc. The type is double.
CenterY	double	Required. This argument is the Y-coordinate for the center point on the arc. The type is double.

CenterZ	double	Required. This argument is the Z-coordinate for the center point on the arc. The type is double.
StartX	double	Required. This argument is the X-coordinate for the starting point on the arc. The type is double.
StartY	double	Required. This argument is the Y-coordinate for the starting point on the arc. The type is double.
StartZ	double	Required. This argument is the X-coordinate for the starting point on the arc. The type is double.
EndX	double	Required. This argument is the X-coordinate for the ending point on the arc. The type is double.
EndY	double	Required. This argument is the Y-coordinate for the ending point on the arc. The type is double.
EndZ	double	Required. This argument is the Z-coordinate for the ending point on the arc. The type is double.

CreateByComplexString Method

Description
The CreateByComplexString method creates and returns a pointer (ppObject) to the interface of a BSplineCurve3d object. This method works by converting an input complex string.

Syntax
object.CreateByComplexString*(pConnection, pCS)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
pCS	IJComplexString	Required. This argument is a pointer to IJComplexString.

CreateByCtrNormStartEnd Method

Description
The CreateByCtrNormStartEnd method creates and returns an Arc3d object given the center, normal vector, start and end points, radius, and direction.

Syntax
object.CreateByCtrNormStartEnd*(pConnection, CenterX, CenterY, CenterZ, NormalX, NormalY, NormalZ, StartX, StartY, StartZ, EndX, EndY, EndZ)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
CenterX	double	Required. This argument is the X-coordinate for the center point of the arc. The type is double.
CenterY	double	Required. This argument is the Y-coordinate for the center point of the arc. The type is double.
CenterZ	double	Required. This argument is the Z-coordinate for the center point of the arc. The type is double.
NormalX	double	Required. This argument is the X-coordinate for a point on the normal vector. The type is double.
NormalY	double	Required. This argument is the Y-coordinate for a point on the normal vector. The type is double.
NormalZ	double	Required. This argument is the Z-coordinate for a point on the normal vector. The type is double.
StartX	double	Required. This argument is the X-coordinate for the starting point on the arc.The type is double.
StartY	double	Required. This argument is the Y-coordinate for the starting point on the arc. The type is double.

StartZ	double	Required. This argument is the Z-coordinate for the starting point on the arc. The type is double.
EndX	double	Required. This argument is the X-coordinate for the ending point on the arc. The type is double.
EndY	double	Required. This argument is the Y-coordinate for the ending point on the arc. The type is double.
EndZ	double	Required. This argument is the Z-coordinate for the ending point on the arc. The type is double.

CreateByCurve Method (IProjections3d)

Description
The CreateByCurve method creates and returns a pointer (ppObj) to the IJProjection interface of a Projection3d object based on a planar curve, direction, and length.

Syntax
object.CreateByCurve*(pConnection, CurveObject, uvX, uvY, uvZ, Length, Capped)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
CurveObject	Object	Required. This argument is the IDispatch interface of the planar curve.
uvX	double	Required. This argument is the X-coordinate of the point along the curve in the plane. The type is double.
uvY	double	Required. This argument is the Y-coordinate of the point along the curve in the plane. The type is double.
uvZ	double	Required. This argument is the Z-coordinate of the point along the curve in the plane. The type is double.
Length	double	Required. This argument is the length of the projection in the direction of the point. The type is double.
Capped	Boolean	Required. This argument is a Boolean flag indicating whether or not the object is capped.

CreateByCurve Method (IRevolutions3d)

Description
The CreateByCurve method creates and returns a pointer (ppObj) to the IJRevolution interface of a Revolution3d object based on a curve to revolve, an axis vector, and a point on the axis.

Syntax
object.CreateByCurve*(pConnection, CurveObject, AxisX, AxisY, AxisZ, CenterX, CenterY, CenterZ, SwAngle, Capped)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
CurveObject	Object	Required. This argument is the IDispatch interface of the planar curve.
AxisX	double	Required. This argument is the X-coordinate of a point on the axis vector. The type is double.
AxisY	double	Required. This argument is the Y-coordinate of a point on the axis vector. The type is double.
AxisZ	double	Required. This argument is the Z-coordinate of a point on the axis vector. The type is double.

CenterX	double	Required. This argument is the X-coordinate of the center point on the axis. The type is double.
CenterY	double	Required. This argument is the Y-coordinate of the center point on the axis. The type is double.
CenterZ	double	Required. This argument is the Z-coordinate of the center point on the axis. The type is double.
SwAngle	double	Required. This argument is the sweep angle in radians. The type is double.
Capped	Boolean	Required. This argument is a Boolean flag indicating whether or not the object is capped. If capped, then the result is either a closed planar curve revolved partially or an open planar curve revolved fully.

CreateByCurves Method (IComplexStrings3d)

Description
The CreateByCurves method creates and returns a pointer (ppObj) to the IJComplexString interface of a ComplexString3d object. The input to this method is an array of Curves. Allowable open curve types include Line3d, Arc3d, EllipticalArc3d, LineString3d, ComplexString3d, and BsplineCurve3d.

Syntax
object.CreateByCurves*(pConnection, pIJCurveElements)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
pIJCurveElements	IJElements	Required. This argument is a pointer to the first element in an array of Curves.

CreateByCurves Method (IRuledSurfaces3d)

Description
The CreateByCurves method creates and returns a pointer (ppObj) to the IJRuled interface of a RuledSurface3d object based on a base curve and a top curve.

Syntax
object.CreateByCurves*(pConnection, CurveObjectBase, CurveObjectTop, Capped)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
CurveObjectBase	Object	Required. This argument is the IDispatch interface of the base planar curve.
CurveObjectTop	Object	Required. This argument is the IDispatch interface of the top planar curve. The type is double.
Capped	Boolean	Required. This argument is a Boolean flag indicating whether or not the object is capped. If capped, then the result is either two closed planar curves or one degenerate and the other closed and planar.

CreateByFitCurve Method

Description
The CreateByFitCurve method creates and returns a pointer (ppObj) to the interface of a BSplineCurve3d object. This method works by direct fitting a set of points.
The start and end tangent constraints are optional. These constraints should be set to 0.0 if they are not needed.
The Order property determines the relative accuracy of the poles with regard to the points that are entered to create the curve. The order returned evaluates as a polynomial degree plus one. For example, an order of 4 defines cubic.

Since it is more efficient to use even-order b-spline curves, the number of poles (and knots) are maximized by increasing the order to the next even number.

Syntax
object.CreateByFitCurve*(pConnection, Order, PointCount, Points, Start_vX, Start_vY, Start_vZ, End_vX, End_vY, End_vZ, Closed, periodic)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
Order	long	Required. This argument is the order of the curve. The type is long.
PointCount	long	Required. This argument is the number of points along the curve. The type is long.
Points	double	Required. This argument is a SAFEARRAY of points along the curve. The type is double.
Start_vX	double	Required. This argument is the X-coordinate for the starting point of the curve. The type is double.
Start_vY	double	Required. This argument is the Y-coordinate for the starting point of the curve. The type is double.
Start_vZ	double	Required. This argument is the Z-coordinate for the starting point of the curve. The type is double.
End_vX	double	Required. This argument is the X-coordinate for the ending point of the curve. The type is double.
End_vY	double	Required. This argument is the Y-coordinate for the ending point of the curve. The type is double.
End_vZ	double	Required. This argument is the Z-coordinate for the ending point of the curve. The type is double.
Closed	Boolean	Required. This argument is a Boolean flag that specifies whether or not the curve is closed.
periodic	Boolean	Required. This argument is a Boolean flag that specifies whether or not the curve is periodic.

CreateByFitSurface Method

Description
The CreateByFitSurface method creates and returns a pointer (ppObj) to an interface for a BSplineSurface3d object. This method does a direct fit of a B-spline surface through a set of points. The points are ordered (as surface poles are) in the u-direction by v-direction.

Syntax
object.CreateByFitSurface*(pConnection, vNumPoints, uNumPoints, Points, uOrder, vOrder, uClosedForm, vClosedForm)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
vNumPoints	long	Required. This argument is a SAFEARRAY of the v-number of points along the surface. The type is double.
uNumPoints	double	Required. This argument is a SAFEARRAY of the u-number of points along the surface. The type is double.
Points	double	Required. This argument is a SAFEARRAY of points along the surface. The type is double.
uOrder	long	Required. This argument is the u order of the surface, which must be greater than 1. The type is long.

vOrder	long	Required. This argument is the v-order of the surface, which must be greater than 1. The type is long.
uClosedForm	long	Required. This argument specifies the smoothness at the start and end of a closed B-spline surface in the u-direction. The type is long. If 0: no smoothness requirements, 1: closed with tangent continuity (no tangents input) (this value is not currently supported), 2: closed and periodic.
vClosedForm	long	Required. This argument specifies the smoothness at the start and end of a closed B-spline surface in the v-direction. The type is long. If 0: no smoothness requirements, 1: closed with tangent continuity (no tangents input) (this value is not currently supported), 2: closed and periodic.

CreateByLeastSquareFitCurve Method

Description

The CreateByLeastSquareFitCurve method creates and returns a pointer (ppObj) to the interface of a BSplineCurve3d object. This method fits a set of points using least squares.
The start and end tangent constraints are optional. You should set these constraints to 0.0 if they are not needed.

Syntax

object.CreateByLeastSquareFitCurve*(pConnection, Order, PointCount, Points, Start_vX, Start_vY, Start_vZ, End_vX, End_vY, End_vZ, Closed, periodic, opt, nseg, tol)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
Order	long	Required. This argument specifies the order of the curve. The type is long.
PointCount	long	Required. This argument is the number of points along the curve. The type is long.
Points	double	Required. This argument is a SAFEARRAY of points along the curve. The type is double.
Start_vX	double	Required. This argument is the X-coordinate for the starting point of the curve. The type is double.
Start_vY	double	Required. This argument is the Y-coordinate for the starting point of the curve. The type is double.
Start_vZ	double	Required. This argument is the Z-coordinate for the starting point of the curve. The type is double.
End_vX	double	Required. This argument is the X-coordinate for the ending point of the curve. The type is double.
End_vY	double	Required. This argument is the Y-coordinate for the ending point of the curve. The type is double.
End_vZ	double	Required. This argument is the Z-coordinate for the ending point of the curve. The type is double.
Closed	Boolean	Required. This argument is a Boolean flag that specifies whether or not the curve is closed.
periodic	Boolean	Required. This argument is a Boolean flag that specifies whether or not the curve is periodic.
opt	Boolean	Required. This argument is an option that specifies the fit of the curve. Its type is Boolean. If this option is 0, it means fit within the given tolerance; if it is 1, it means fit with the given number of segments.
nseg	long	Required. This argument is the number of segments used in the fitting, if opt=1. The type is long.
tol	double	Required. This argument is the tolerance used in the fitting, if opt = 0. The type is double.

CreateByLeastSquareFitSurface Method

Description
The CreateByLeastSquareFitSurface method creates and returns a pointer (ppObj) to an interface for a a BSplineSurface3d object. This method does a least square fit of a B-spline surface through a set of points. The points are ordered (as surface poles are) in the u-direction by v-direction.

Syntax
object.CreateByLeastSquareFitSurface*(pConnection, vNumPoints, uNumPoints, Points, uOrder, vOrder, uPeriodic, vPeriodic, uNseg, vNseg)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
vNumPoints	long	Required. This argument is a SAFEARRAY of the v-number of points along the surface. The type is double.
uNumPoints	double	Required. This argument is a SAFEARRAY of the u-number of points along the surface. The type is double.
Points	double	Required. This argument is a SAFEARRAY of points along the surface. The type is double.
uOrder	long	Required. This argument is the u-order of the surface, which must be greater than 1. The type is long.
vOrder	long	Required. This argument is the v-order of the surface, which must be greater than 1. The type is long.
uPeriodic	Boolean	Required. This argument is a Boolean flag that specifies whether or not the surface is periodic in u.
vPeriodic	Boolean	Required. This argument is a Boolean flag that specifies whether the surface is periodic in v.
uNseg	long	Required. This argument is the number of segments in u. The type is long.
vNseg	long	Required. This argument is the number of segments in v. The type is long.

CreateByOffset Method

Description
The CreateByOffset method creates and returns an offset curve.

Syntax
object.CreateByOffset*(pConnection, Obj, DPtx, DPty, DPtz, OffsetDist, code)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
Obj	Object	Required. This argument is the curve to offset. The type is Object.
DPtx	double	Required. This argument is the vector component in the X-direction. The type is double.
DPty	double	Required. This argument is the vector component in the Y-direction. The type is double.
DPtz	double	Required. This argument is the vector component in the Z-direction. The type is double.
OffsetDist	double	Required. This argument is the distance for the offset. The type is double.

code	Int	Required. This argument is an integer that describes the offset curve. Possible values are: 0 - extend; 1 - fillet.

CreateByOuterBdry Method

Description
The CreateByOuterBdry method creates and returns a pointer (ppObj) to the IJPlane interface of an infinite Plane3d object, based on a point and a normal.

Syntax
object.CreateByOuterBdry*(pConnection, CurveObject)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
CurveObject	Object	Required. This argument is the IDispatch interface of the planar curve.

CreateByParameters Method (IBSplineCurves3d)

Description
The CreateByParameters method creates and returns a pointer (ppObj) to the interface of a BSplineCurve3d object. This method uses order, poles, weights, and knots. The weights and knots are optional; they should be set to NULL if not needed.

If the order is equal to the number of poles, the curve evolves into the control polygon of a Bezier curve.

B-spline weights can be considered a gravitational type force with the magnitude of the weight equal to the pulling force. The weights are always normalized. If no weights are present, the curve is considered to be non-rational and may be NULL. Non-rational curves have weights with a value of 1.

The B-spline knots define the parameterization of the curve, and they may be periodic. Knots, also known as knot vectors, must be monotonic and strictly increasing. Monotonic refers to the successive terms as non-decreasing or non-increasing.

The Order property determines the relative accuracy of the poles with regard to the points that are entered to create the curve. The order returned evaluates as a polynomial degree plus one. For example, an order of 4 defines cubic. Since it is more efficient to use even-order b-spline curves, the number of poles (and knots) are maximized by increasing the order to the next even number.

Syntax
object.CreateByParameters*(pConnection, Order, PoleCount, Poles, Weights, Knots, periodic)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
Order	long	Required. This argument specifies the order of the curve. The type is long.
PoleCount	long	Required. This argument is the number of poles. The type is long.
Poles	double	Required. This argument is a SAFEARRAY of poles. The type is double.
Weights	double	Required. This argument is a SAFEARRAY of weights. The type is double.
Knots	double	Required. This argument is a SAFEARRAY of knots. The type is double. Generally, this value is the number of poles plus the order value.
periodic	Boolean	Required. This argument is a Boolean flag that specifies whether or not the curve is periodic.

CreateByParameters Method (IBSplineSurfaces3d)

Description

The CreateByParameters method creates and returns a pointer (ppObj) to an interface for a BSplineSurface3d object based on the desired order and a set of poles (weights and knots are optional).
If periodic knots are passed in, but periodic is set to False, the knots will be converted to multiple end knots.
The outward normal is generally U cross V, but if the reverse normal is desired, set ReverseNor to True.
The poles are ordered in the u-direction by v-direction. Weights and knots are optional. The number of poles (u or v) must be greater than or equal to the order in that direction.

Syntax

object.CreateByParameters*(pConnection, uNumPoles, vNumPoles, Poles, Weights, uOrder, vOrder, uKnots, vKnots, uPeriodic, vPeriodic, ReverseNor)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
uNumPoles	long	Required. This argument is the number of poles in the u-direction. The type is long.
vNumPoles	long	Required. This argument is the number of poles in the v-direction. The type is long.
Poles	double	Required. This argument is a SAFEARRAY of poles. The type is double.
Weights	double	Required. This argument is a SAFEARRAY of weights. The type is double.
uOrder	long	Required. This argument is the u-order of the surface, which must be greater than 1. The type is long.
vOrder	long	Required. This argument is the v-order of the surface, which must be greater than 1. The type is long.
uKnots	double	Required. This argument is a SAFEARRAY of knots. The type is double.
vKnots	double	Required. This argument is a SAFEARRAY of knots. The type is double.
uPeriodic	Boolean	Required. This argument is a Boolean flag that specifies whether the surface is periodic in u.
vPeriodic	Boolean	Required. This argument is a Boolean flag that specifies whether the surface is periodic in v.
ReverseNor	Boolean	Required. This argument is a Boolean flag that specifies whether or not the direction of the normal is reversed.

CreateByPartOfCurve Method

Description

The CreateByPartOfCurve method creates and returns a part of the input curve.
Note: It is possible to cross the seam.

Syntax

object.CreateByPartOfCurve*(pConnection, Obj, startPar, dirPar, endPar)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
Obj	Object	Required. This argument is the IDispatch interface of the top planar curve.
startPar	double	Required. This argument is the start of the part of the curve.

dirPar	double	Required. This argument is a point as the direction of the part of the curve that is returned.
endPar	double	Required. This argument is the end of the part of the curve.

CreateByPoint Method

Description
The CreateByPoint method creates and returns an interface for a Point3d object, given X-, Y- and Z-coordinates.

Syntax
object.CreateByPoint*(pConnection, x, y, z)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
x	double	Required. This argument is the X-coordinate. The type is double.
y	double	Required. This argument is the Y-coordinate. The type is double.
z	double	Required. This argument is the Z-coordinate. The type is double.

CreateByPointNormal Method

Description
The CreateByPointNormal method creates and returns a pointer (ppObj) to the IJPlane interface of an infinite Plane3d object, based on a point and a normal.

Syntax
object.CreateByPointNormal*(pConnection, PointX, PointY, PointZ, NormalX, NormalY, NormalZ)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
PointX	double	Required. This argument is the X-coordinate of the point. The type is double.
PointY	double	Required. This argument is the Y-coordinate of the point. The type is double.
PointZ	double	Required. This argument is the Z-coordinate of the point. The type is double.
NormalX	double	Required. This argument is the X-coordinate of a point on the normal. The type is double.
NormalY	double	Required. This argument is the Y-coordinate of a point on the normal. The type is double.
NormalZ	double	Required. This argument is the Z-coordinate of a point on the normal. The type is double.

CreateByPoints Method

Description
The CreateByPoints method creates and returns a pointer (ppObj) to the interface of a LineString3d object. This method takes as input an array of points. The array is a one-dimensional array of doubles containing the X-, Y-, and Z-coordinates of the vertex points.

Syntax
object.CreateByPoints*(pConnection, PointCount, Points)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
PointCount	long	Required. This argument is the number of points in the array. The type is long.
Points	double	Required. This argument is a SAFEARRAY of points. The type is double.

CreateByPtVectLength Method

Description
The CreateByPtVectLength method creates and returns a Line3d object, given the starting point, direction vector, and length.

Syntax
object.CreateByPtVectLength*(pConnection, StartX, StartY, StartZ, uvX, uvY, uvZ, Length)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
StartX	double	Required. This argument is the X-coordinate for the starting point. The type is double.
StartY	double	Required. This argument is the Y-coordinate for the starting point. The type is double.
StartZ	double	Required. This argument is the Z-coordinate for the starting point. The type is double.
uvX	double	Required. This argument is the X-coordinate for the ending point. The type is double.
uvY	double	Required. This argument is the Y-coordinate for the ending point. The type is double.
uvZ	double	Required. This argument is the Z-coordinate for the ending point. The type is double.
Length	double	Required. This argument is the length of the line from the starting point. The type is double.

CreateBySingleSweepWCaps Method

Description
The CreateBySingleSweepWCaps method creates a collection of swept surfaces with the option of caps. The output is surfaces, and then caps.

Syntax
object.CreateBySingleSweepWCaps*(pConnection, TrObj, CsObj, cornerOpt, BrkCv, StartOpt, StNorm, EdNorm, WCaps, numCaps)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
TrObj	Object	Required. This argument is the trace curve. The type is Object.
CsObj	Object	Required. This object is the cross section curve or curve to sweep. It can be one curve, or it can be a plane object that contains boundary curves, where the boundary curves are each swept to make a separate surface; the first boundary of the plane is always the region, and any following boundaries are holes. The type for CsObj is Object.
cornerOpt	SkinningCornerOptions	Required. This argument is an option on how to handle trace curves that are line strings. If the value is 0, the method averages the left/right tangent to get the plane for

		placing the cross section. If the value is 1, the method turns around the trace cusp with an arc.
BrkCv	SkinningBreakOptions	Required. This argument specifies whether or not the curves have breaks. Possible values include: 0 - No breaks. 1 - If the cross is a GComplexString, then break and create separate surfaces. 2 - If the trace is a GComplexString, then break and create separate surfaces. 3 - Break cross and trace.
StartOpt	SkinningCrossSectionStart	Required. This argument is the starting option. Possible values are: 0 - No breaks; 1 - If the cross is a GComplexString, then break and create separate surfaces; 2 - If the trace is a GComplexString, then break and create separate surfaces; 3 - Break cross and trace.
StNorm	double	Required. This argument specifies the starting normal. It is a SAFEARRAY of type double.
EdNorm	double	Required. This argument specifies the ending normal. It is a SAFEARRAY of type double.
WCaps	Boolean	Required. This argument is a Boolean flag that specifies whether or not the surfaces have caps. If the value is False, there are no caps; if the value is True, there are caps.
numCaps	Int	Required. This argument is the number of caps. The type is integer.

CreateBySkinning Method

Description
The CreateBySkinning method creates a skinned surface with the option of caps. The output is caps and the skin surface.

Syntax
object.CreateBySkinning*(pConnection, pTrElements, pCsElements, WCaps)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
pTrElements	IJElements	Required. This argument is a pointer to the trace curves (can be more than 1). If there is one trace only, the trace curve does not have to touch the cross section, but must cross the plane containing the cross section. If there is more than one trace, then the trace curves must touch the cross sections.
pCsElements	IJElements	Required. This argument is a pointer to the cross section curves The value can be more than 1. Cross sections are placed exactly how they are to be skinned.
WCaps	Int	Required. This argument is a Boolean flag that specifies whether or not the object has caps. If the value is False, there are no caps; if True, there are caps.

CreateConeBy4PtsWCaps Method

Description
The CreateConeBy4PtsWCaps method creates and returns a bounded Cone3d object based on four points **-** base center point, top center point, base starting point, and top starting point. Caps are optional. The output is the surface, and then caps.

Syntax
object.CreateConeBy4PtsWCaps*(pConnection, CenterBx, CenterBy, CenterBz, CenterTx, CenterTy, CenterTz, StartBx, StartBy, StartBz, StartTx, StartTy, StartTz, Solid, WCaps, numCaps)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
CenterBx	double	Required. This argument is the X-coordinate for the base ellipse center point. The type is double.
CenterBy	double	Required. This argument is the Y-coordinate for the base ellipse center point. The type is double.
CenterBz	double	Required. This argument is the Z-coordinate for the base ellipse center point. The type is double.
CenterTx	double	Required. This argument is the X-coordinate for the top ellipse center point. The type is double.
CenterTy	double	Required. This argument is the Y-coordinate for the top ellipse center point. The type is double.
CenterTz	double	Required. This argument is the Z-coordinate for the top ellipse center point. The type is double.
StartBx	double	Required. This argument is the X-coordinate for the base ellipse starting point. The type is double.
StartBy	double	Required. This argument is the Y-coordinate for the top ellipse starting point. The type is double.
StartBz	double	Required. This argument is the Z-coordinate for the base ellipse starting point. The type is double.
StartTx	double	Required. This argument is the X-coordinate for the top ellipse starting point. The type is double.
StartTy	double	Required. This argument is the Y-coordinate for the top ellipse starting point. The type is double.
StartTz	double	Required. This argument is the Z-coordinate for the top ellipse starting point. The type is double.
Solid	Boolean	Required. This argument is a Boolean flag that specifies whether or not the object is solid. Possible values are: 0 - Set normal as hollow; 1 - Set normal as solid; 2 - Set normal according to input point; 3 - Toggle the outward normal (no checks).
WCaps	Boolean	Required. This argument is a Boolean flag that specifies whether or not the object has caps. If the value is False, there are no caps; if True, there are caps.
numCaps	Int	Required. This argument is the number of caps. The type is integer.

CreateProjectionByCurveWCaps Method

Description
The CreateProjectionByCurveWCaps method creates a Projection3d object from a curve, direction, and length. Valid curves are Line, Arc, Circle, Ellipse, EllipticalArc, LineString, ComplexString, and BSplineCurve. Caps are optional. The output is the surface, and then the caps.

Syntax
object.CreateProjectionByCurveWCaps*(pConnection, CurveObject, uvX, uvY, uvZ, Length, Solid, WCaps, numCaps)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
CurveObject	Object	Required. This argument is the curve to project. The type is Object.
uvX	double	Required. This argument is the X-coordinate of the point that specifies the vector. The type is double.
uvY	double	Required. This argument is the Y-coordinate of the point that specifies the vector. The type is double.

uvZ	double	Required. This argument is the Z-coordinate of the point that specifies the vector. The type is double.
Length	double	Required. This argument is the projection distance. The type is double.
Solid	Boolean	Required. This argument is a Boolean flag that specifies whether or not the object is solid. Possible values are: 0 - Set normal as hollow; 1 - Set normal as solid; 2 - Set normal according to input point; 3 - Toggle the outward normal (no checks).
WCaps	Boolean	Required. This argument is a Boolean flag that specifies whether or not the object has caps. If the value is False, there are no caps; if True, there are caps.
numCaps	Int	Required. This argument is the number of caps. The type is integer.

CreateRevolutionByCurveWCaps Method

Description
The CreateRevolutionByCurveWCaps method creates a Revolution3d object from a curve, axis vector, point on axis, and sweep angle (radians). Valid curves are Line, Arc, Circle, Ellipse, EllipticalArc, LineString, ComplexString, and BSplineCurve. Caps are optional. Output is the surface, and then the caps.

Syntax
object.CreateRevolutionByCurveWCaps*(pConnection, CurveObject, AxisX, AxisY, AxisZ, CenterX, CenterY, CenterZ, SwAngle, Solid, WCaps, numCaps)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
CurveObject	Object	Required. This argument is the curve from which to create the revolution. The type is Object.
AxisX	double	Required. This argument is the X-coordinate of the point that specifies the axis direction. The type is double.
AxisY	double	Required. This argument is the Y-coordinate of the point that specifies the axis direction. The type is double.
AxisZ	double	Required. This argument is the Z-coordinate of the point that specifies the axis direction. The type is double.
CenterX	double	Required. This argument is the X-coordinate of the center point. The type is double.
CenterY	double	Required. This argument is the Y-coordinate of the center point. The type is double.
CenterZ	double	Required. This argument is the Z-coordinate of the center point. The type is double.
SwAngle	double	Required. This argument is the sweep angle. The type is double.
Solid	Boolean	Required. This argument is a Boolean flag that specifies whether or not the object is solid. Possible values are: 0 - Set normal as hollow; 1 - Set normal as solid; 2 - Set normal according to input point; 3 - Toggle the outward normal (no checks).
WCaps	Boolean	Required. This argument is a Boolean flag that specifies whether or not the object has caps. If the value is False, there are no caps; if True, there are caps.
numCaps	Int	Required. This argument is the number of caps. The type is integer.

CreateRuledByCurvesWCaps Method

Description
The CreateRuledByCurvesWCaps method creates a RuledSurface3d object from a base curve and a top curve. Valid curves are Line, Arc, Circle, Ellipse, EllipticalArc, LineString, ComplexString, and BSplineCurve. Caps are optional. The output is the surface, and then the caps.

Syntax
object.CreateRuledByCurvesWCaps*(pConnection, CurveObjectBase, CurveObjectTop, Solid, WCaps, numCaps)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
CurveObjectBase	Object	Required. This argument is the base curve.
CurveObjectTop	Object	Required. This argument is the top curve.
Solid	Boolean	Required. This argument is a Boolean flag that specifies whether or not the object is solid. Possible values are: 0 - Set normal as hollow; 1 - Set normal as solid; 2 - Set normal according to input point; 3 - Toggle the outward normal (no checks).
WCaps	Boolean	Required. This argument is a Boolean flag that specifies whether or not the object has caps. If the value is False, there are no caps; if True, there are caps.
numCaps	Int	Required. This argument is the number of caps. The type is integer.

CreateTorusByAxisMajorMinorRadiusSweepWCaps Method

Description
The CreateTorusByAxisMajorMinorRadiusSweepWCaps method creates and returns a Tori3d (torus) object based on an axis, a center point on the axis, the direction to the origin in UV space (orthogonal to the axis), a major radius, and a minor radius. Caps are optional. The output is the surface, and then the caps.

Syntax
object.CreateTorusByAxisMajorMinorRadiusSweepWCaps*(pConnection, AxisCenterX, AxisCenterY, AxisCenterZ, AxisVecX, AxisVecY, AxisVecZ, OriginDirX, OriginDirY, OriginDirZ, MajorRadius, MinorRadius, SwAngle, Solid, WCaps, numCaps)*

Parameter	Data Type	Description
pConnection	Unknown	Required. This argument is a pointer to IUnknown. It creates a transient object.
AxisCenterX	double	Required. This argument is the X-coordinate of the axis center point. The type is double.
AxisCenterY	double	Required. This argument is the Y-coordinate of the axis center point. The type is double.
AxisCenterZ	double	Required. This argument is the Z-coordinate of the axis center point. The type is double.
AxisVecX	double	Required. This argument is the X-coordinate of the point that specifies the axis direction. The type is double.
AxisVecY	double	Required. This argument is the Y-coordinate of the point that specifies the axis direction. The type is double.
AxisVecZ	double	Required. This argument is the Z-coordinate of the point that specifies the axis direction. The type is double.
OriginDirX	double	Required. This argument is the X-coordinate of the point that specifies the origin direction. The

		type is double.
OriginDirY	double	Required. This argument is the Y-coordinate of the point that specifies the origin direction. The type is double.
OriginDirZ	double	Required. This argument is the Z-coordinate of the point that specifies the origin direction. The type is double.
MajorRadius	double	Required. This argument is the major radius for the torus. The type is double.
MinorRadius	double	Required. This argument is the minor radius for the torus. The type is double.
SwAngle	double	Required. This argument is the sweep angle. The type is double.
Solid	Boolean	Required. This argument is a Boolean flag that specifies whether or not the object is solid. Possible values are: 0 - Set normal as hollow; 1 - Set normal as solid; 2 - Set normal according to input point; 3 - Toggle the outward normal (no checks).
WCaps	Boolean	Required. This argument is a Boolean flag that specifies whether or not the object has caps. If the value is False, there are no caps; if True, there are caps.
numCaps	Int	Required. This argument is the number of caps. The type is integer.

The following section shows some examples on how to create some geometry components:

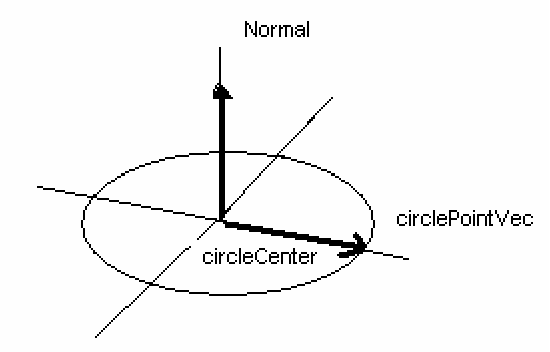
GeometryFactory.Ellipses3dCreateByCenterNormMajAxisRatio

Creates/returns an Ellipse given the center point, normal axis, major axis containing length, and minor/major ratio.

Function Ellipses3d.CreateByCenterNormMajAxisRatio(pConnection As Unknown, CenterX As Double, CenterY As Double, CenterZ As Double, NormalX As Double, NormalY As Double, NormalZ As Double, MajorX As Double, MajorY As Double, MajorZ As Double, MMRatio As Double) As Ellipse3d

Define the collection item:	m_outputColl.ResourceManager
Define the center of the ellipse:	CenterX, CenterY, CenterZ
Define the normal vector:	NormalX, NormalY, NormalZ
Define the major axis vector:	MajorPointVecX, MajorPointVecY, MajorPointVecZ
Define the axis ratio:	MMRatio

Example:



Dim ellipse As IngrGeom3D.Ellipse3d
Dim circlePointVecX As Double, circlePointVecY As Double, circlePointVecZ As Double

```
Dim circleNormalX As Double, circleNormalY As Double, circleNormalZ As Double
Dim projVecX As Double, projVecY As Double, projVecZ As Double

   circleCenterX = 0#
   circleCenterY = 0#
   circleCenterZ = 0#

   circleNormalX = 0#
   circleNormalY = 0#
   circleNormalZ = 1#

   circlePointVecX = 0#
   circlePointVecY = diameter * 0.5
   circlePointVecZ = 0#
  axesRation 1.0

Set ellipse = geomFactory.Ellipses3d.CreateByCenterNormMajAxisRatio(m_outputColl.ResourceManager, _
                        circleCenterX, circleCenterY, circleCenterZ, _
                        circleNormalX, circleNormalY, circleNormalZ, _
                        circlePointVecX, circlePointVecY, circlePointVecZ, _
                        axesRatio)
```

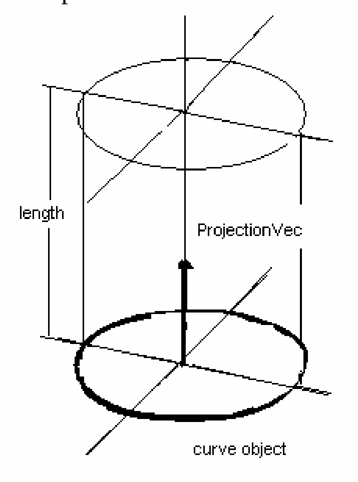
GeomFactory.Projections3d.CreateByCurve

Creates and returns a Projection3d based on a curve, direction and length. Valid curve objects are Line, Arc, Circle, Ellipse, EllipticalArc, LineString, ComplexString, and BSplineCurve.

Function Projections3d.CreateByCurve(pConnection As Unknown, CurveObject As Object, projvecX As Double, projvecY As Double, projvecZ As Double, Length As Double, Capped As Boolean) As Projection3d

Define the collection item:	m_outputColl.ResourceManager
Define the CurveObject to be projected:	CurveObject As Object
Define the projection vector:	projVecX, projVecY, projVecZ As Double
Define the projection sidtance:	Length As Double
Set the ends to be capped true or false:	Capped As Boolean

Example:



```
Dim projection As IngrGeom3D.Projection3d
Dim projVecX As Double, projVecY As Double, projVecZ As Double
Dim length As Double
```

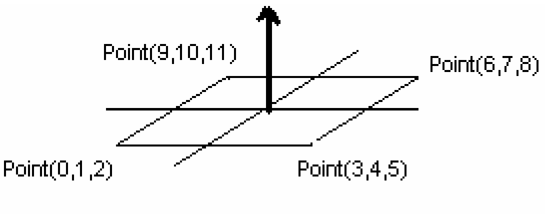
```
   projVecX = 0#
   projVecY = 0#
   projVecZ = 1#

Set projection = geomFactory.Projections3d.CreateByCurve(m_outputColl.ResourceManager, ellipse, _
                                   projVecX, projVecY, projVecZ, length, True)
```

GeomFactory.Planes3d.CreateByPoints

Creates and returns a bounded Plane3d based on 3 or more non-linear, coplanar points. The points must be oriented such that the orientation of the points defines the normal(follows the right hand rule).

Function Planes3d.CreateByPoints(pConnection As Unknown, PointCount As Long, Points() As Double) As Plane3d

Define the collection item:	m_outputColl.ResourceManager
Define the numbe of point in the collection:	PointCount As Long
Input an array of Points():	Points as Double



Example:

```
Dim plane As IngrGeom3D.Plane3d
Dim Points(0 To 11) As Double

   Points(0) = MinX
   Points(1) = MinY
   Points(2) = 0#
   Points(3) = MaxX
   Points(4) = MinY
   Points(5) = 0#
   Points(6) = MaxX
   Points(7) = MaxY
   Points(8) = 0#
   Points(9) = MinX
   Points(10) = MaxY1
   Points(11) = 0#

Set plane = geomFactory.Planes3d.CreateByPoints(m_outputColl.ResourceManager, 4, Points)
```

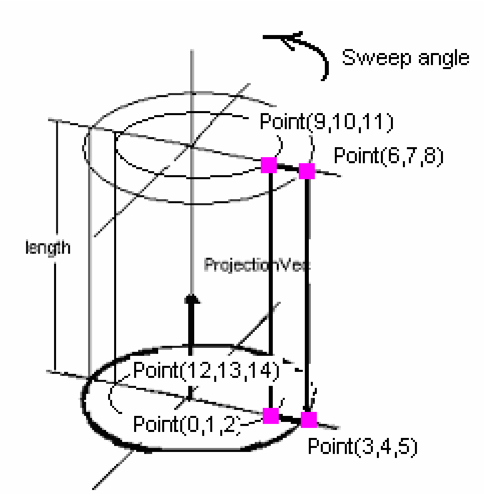
GeomFactory.Revolutions3d.CreateByCurve

Creates and returns a Revolution3d based on a curve to revolve, axis vector, point on axis and sweep angle(radians).Valid curve objects are Line, Arc, Circle, Ellipse, EllipticalArc, LineString, ComplexString, and BSplineCurve.

Function Revolutions3d .CreateByCurve(pConnection As Unknown, CurveObject As Object, AxisX As Double, AxisY As Double, AxisZ As Double, CenterX As Double, CenterY As Double, CenterZ As Double, SwAngle As Double, Capped As Boolean) As Revolution3d

Define the Projection vector to be revolved: AxisX, AxisY, AxisZ as Double
Define the point on axis: CenterX, CenterY, CenterZ as Double
Define sweep angle as Double
Set the ends to be capped true or false: Capped As Boolean

Example:



```
Dim axisCenterX As Double, axisCenterY As Double, axisCenterZ As Double
Dim axisVecX As Double, axisVecY As Double, axisVecZ As Double
Dim oRevolution As IngrGeom3D.Revolution3d

    axisCenterX = 0#
    axisCenterY = 0#
    axisCenterZ = 0#
    axisVecX = 0#
    axisVecY = 0#
    axisVecZ = 1#

Dim oLineString As IngrGeom3D.LineString3d
Dim planePoints(0 To 14) As Double

    planePoints(0) = diameter / 2
    planePoints(1) = 0
    planePoints(2) = 0
    planePoints(3) = diameter / 2 + dInsulationThickness
    planePoints(4) = 0
    planePoints(5) = 0
    planePoints(6) = diameter / 2 + dInsulationThickness
    planePoints(7) = 0
    planePoints(8) = length
```

```
   planePoints(9) = diameter / 2
   planePoints(10) = 0
   planePoints(11) = length
   planePoints(12) = diameter / 2
   planePoints(13) = 0
   planePoints(14) = 0

 Set oLineString = geomFactory.LineStrings3d.CreateByPoints(m_outputColl.ResourceManager, 5,_
                   planePoints)

Set oRevolution = geomFactory.Revolutions3d.CreateByCurve(m_outputColl.ResourceManager, _
                  oLineString,  axisVecX, axisVecY, axisVecZ, axisCenterX, axisCenterY, axisCenterZ, _
                  2 * PI, False)
```

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.

<u>When To Use</u>

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.

Methods

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:

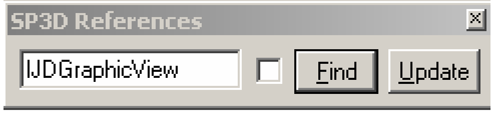
- Click **Project > References**.

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

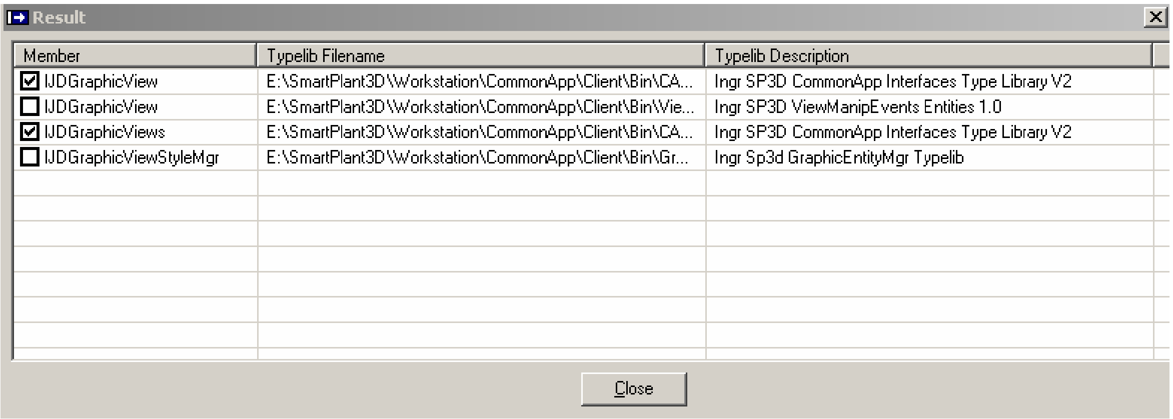
- Click **Project > SP3D References**.

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**.



Member	Typelib Filename	Typelib Description
☑ IJDGraphicView	E:\SmartPlant3D\Workstation\CommonApp\Client\Bin\CA...	Ingr SP3D CommonApp Interfaces Type Library V2
□ IJDGraphicView	E:\SmartPlant3D\Workstation\CommonApp\Client\Bin\Vie...	Ingr SP3D ViewManipEvents Entities 1.0
☑ IJDGraphicViews	E:\SmartPlant3D\Workstation\CommonApp\Client\Bin\CA...	Ingr SP3D CommonApp Interfaces Type Library V2
□ IJDGraphicViewStyleMgr	E:\SmartPlant3D\Workstation\CommonApp\Client\Bin\Gr...	Ingr Sp3d GraphicEntityMgr Typelib

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.

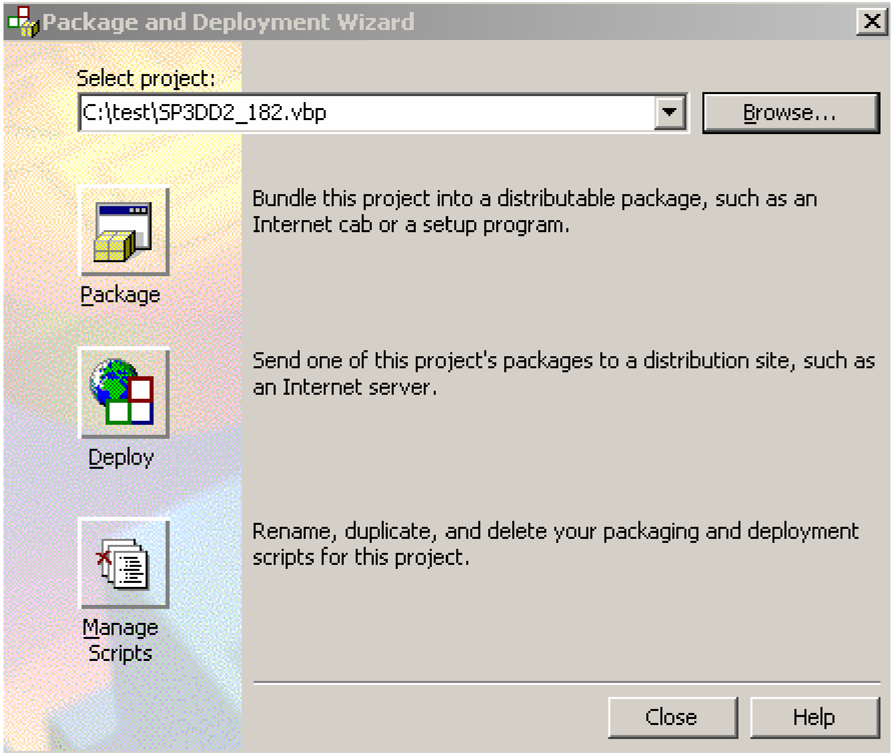
Creation of Cab Files

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



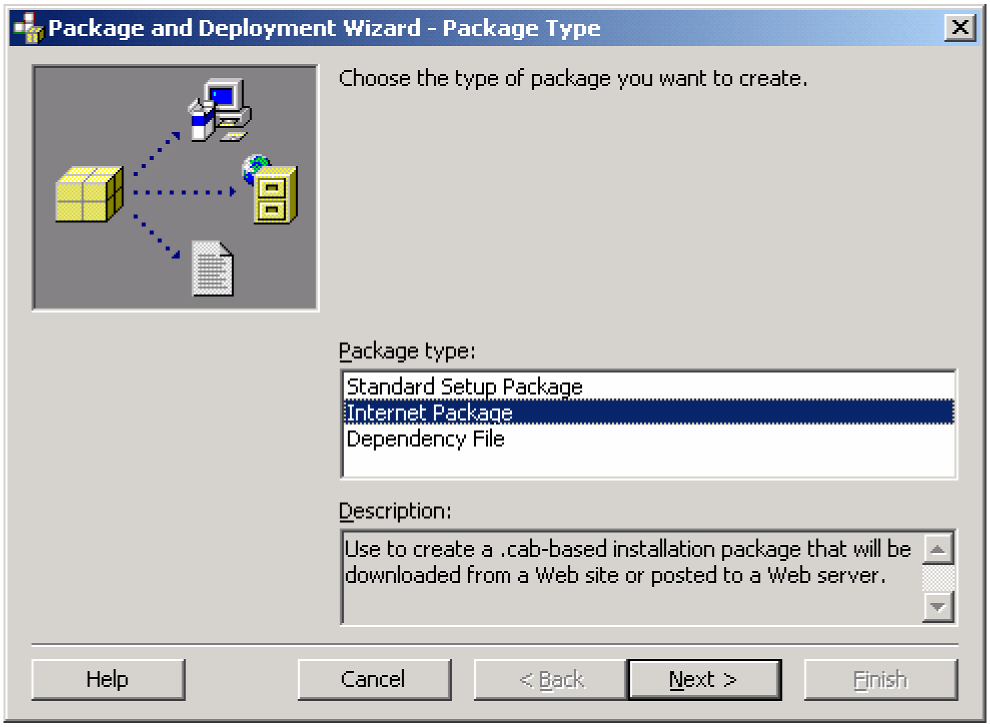
Select project: C:\test\SP3DD2_182.vbp

Bundle this project into a distributable package, such as an Internet cab or a setup program.

Send one of this project's packages to a distribution site, such as an Internet server.

Rename, duplicate, and delete your packaging and deployment scripts for this project.

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



Package and Deployment Wizard - Package Type

Choose the type of package you want to create.

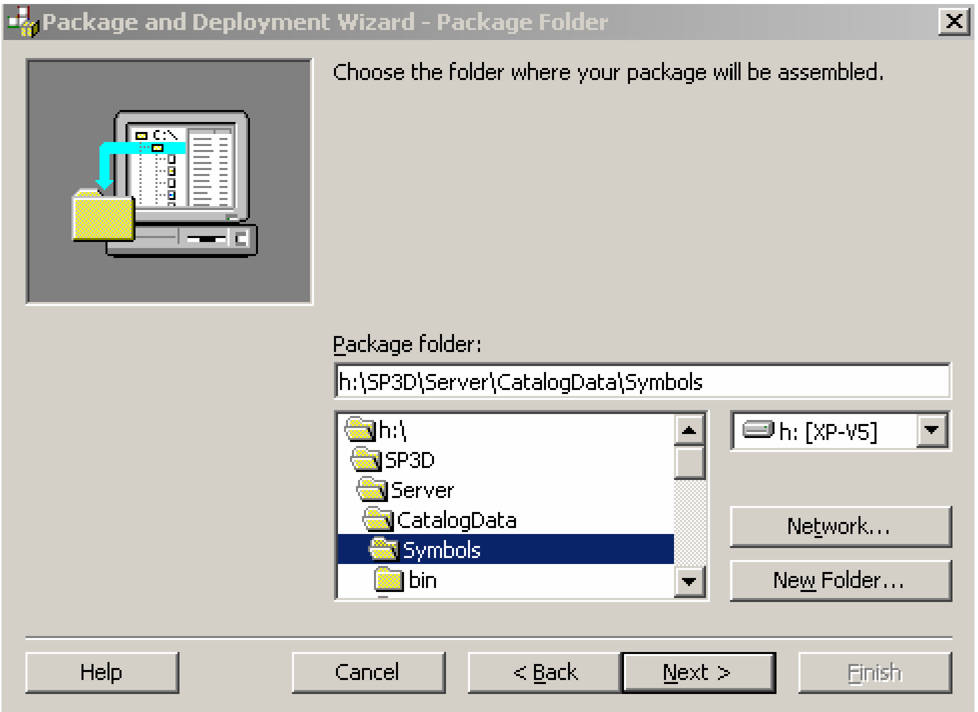
Package type:

- Standard Setup Package
- Internet Package
- Dependency File

Description:

Use to create a .cab-based installation package that will be downloaded from a Web site or posted to a Web server.

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.



Package and Deployment Wizard - Package Folder

Choose the folder where your package will be assembled.

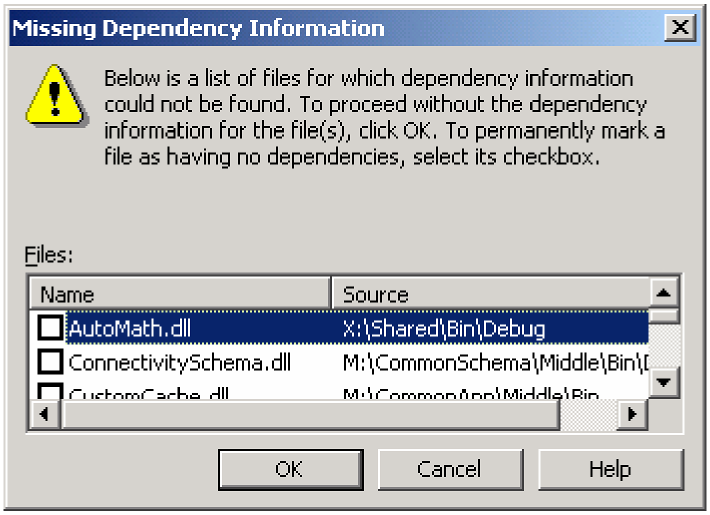
Package folder:

h:\SP3D\Server\CatalogData\Symbols

- h:\
- SP3D
- Server
- CatalogData
- Symbols
- bin

h: [XP-V5]

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



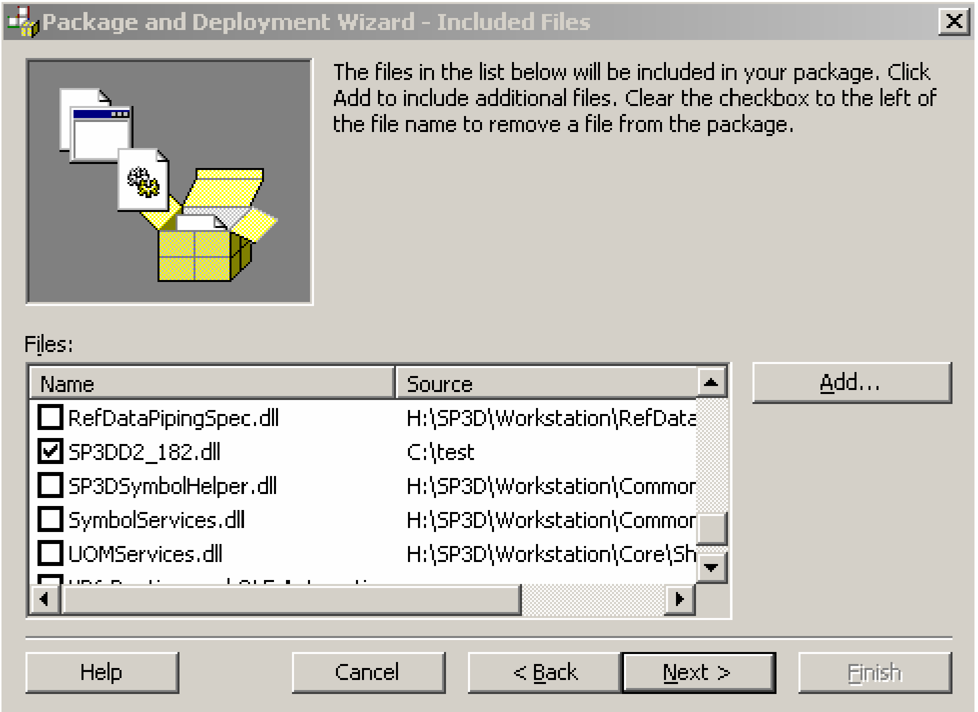
Missing Dependency Information

Below is a list of files for which dependency information could not be found. To proceed without the dependency information for the file(s), click OK. To permanently mark a file as having no dependencies, select its checkbox.

Files:

Name	Source
□ AutoMath.dll	X:\Shared\Bin\Debug
□ ConnectivitySchema.dll	M:\CommonSchema\Middle\Bin\
□ CustomCache.dll	M:\CommonApp\Middle\Bin

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



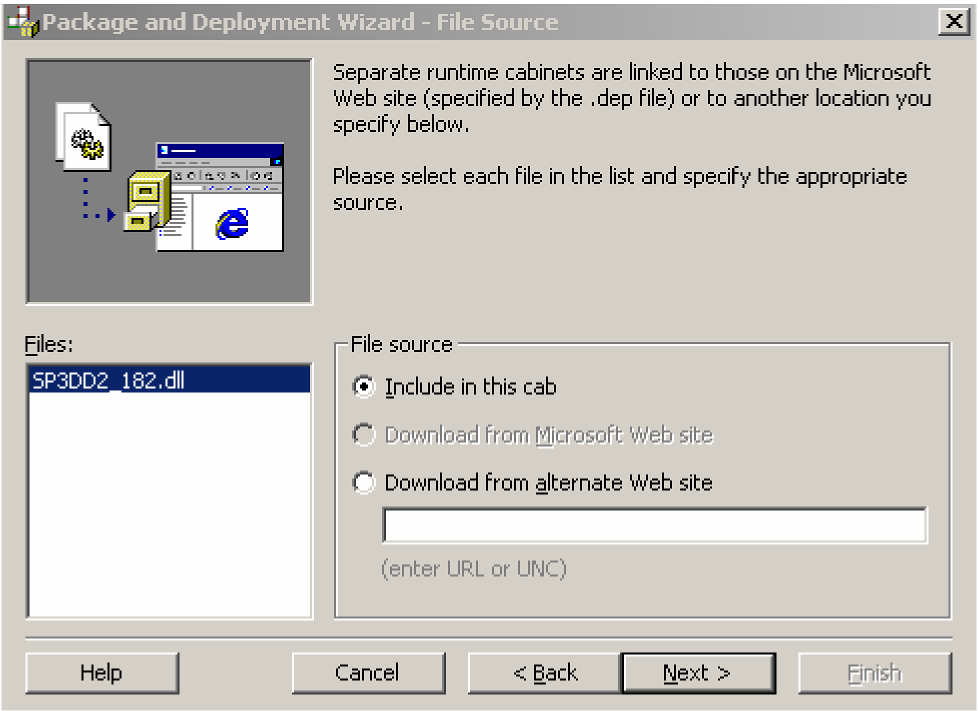
Package and Deployment Wizard - Included Files

The files in the list below will be included in your package. Click Add to include additional files. Clear the checkbox to the left of the file name to remove a file from the package.

Files:

Name	Source
□ RefDataPipingSpec.dll	H:\SP3D\Workstation\RefData
☑ SP3DD2_182.dll	C:\test
□ SP3DSymbolHelper.dll	H:\SP3D\Workstation\Commor
□ SymbolServices.dll	H:\SP3D\Workstation\Commor
□ UOMServices.dll	H:\SP3D\Workstation\Core\Sh

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



Package and Deployment Wizard - File Source

Separate runtime cabinets are linked to those on the Microsoft Web site (specified by the .dep file) or to another location you specify below.

Please select each file in the list and specify the appropriate source.

Files:

SP3DD2_182.dll

File source

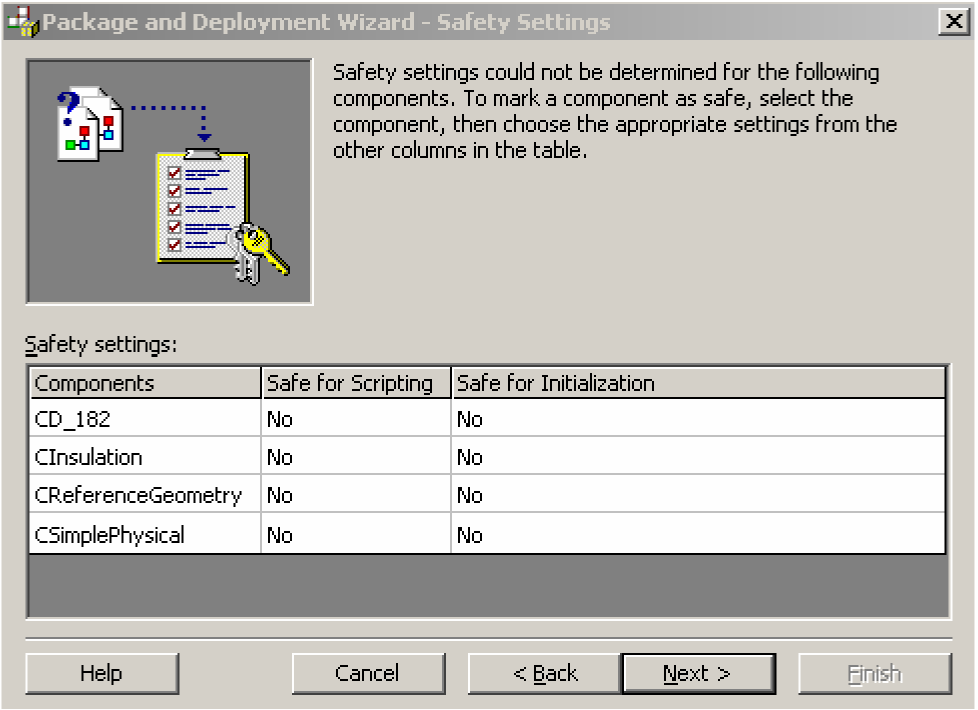
● Include in this cab

○ Download from Microsoft Web site

○ Download from alternate Web site

(enter URL or UNC)

7. Retain the Safety Settings indicated. Click Next.



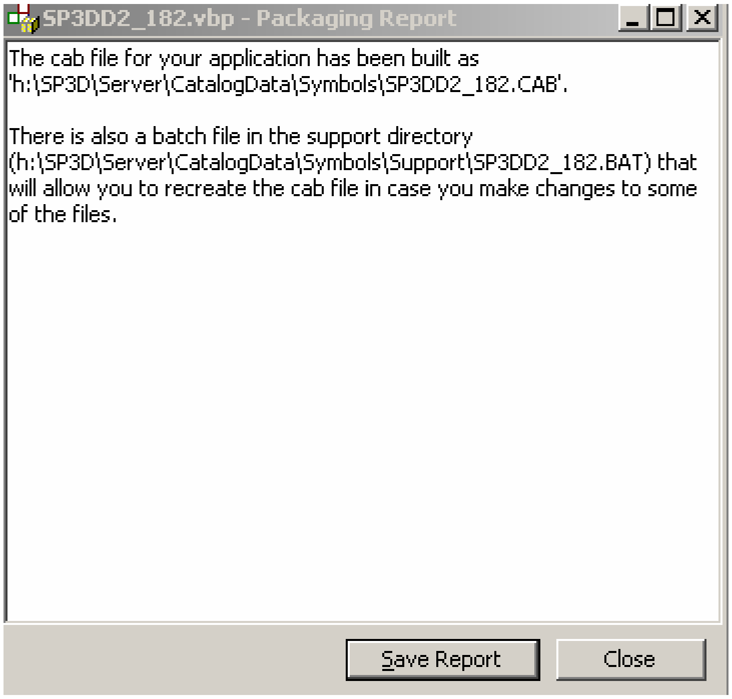
Package and Deployment Wizard - Safety Settings

Safety settings could not be determined for the following components. To mark a component as safe, select the component, then choose the appropriate settings from the other columns in the table.

Safety settings:

Components	Safe for Scripting	Safe for Initialization
CD_182	No	No
CInsulation	No	No
CReferenceGeometry	No	No
CSimplePhysical	No	No

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



The cab file for your application has been built as 'h:\SP3D\Server\CatalogData\Symbols\SP3DD2_182.CAB'.

There is also a batch file in the support directory (h:\SP3D\Server\CatalogData\Symbols\Support\SP3DD2_182.BAT) that will allow you to recreate the cab file in case you make changes to some of the files.

Hit close button.