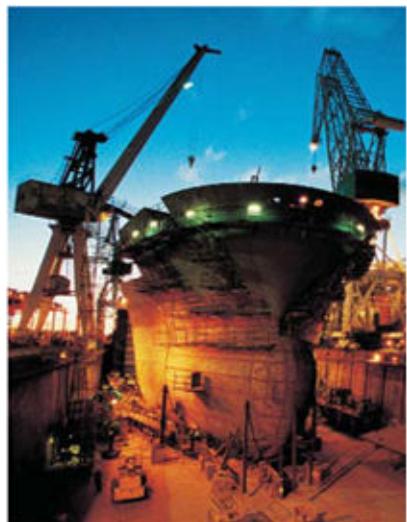


SmartPlant 3D

Reports Workshop Course Guide

Process, Power & Marine



INTERGRAPH

Copyright

Copyright © 2004 Intergraph Corporation. All Rights Reserved.

Including software, file formats, and audiovisual displays; may be used pursuant to applicable software license agreement; contains confidential and proprietary information of Intergraph and/or third parties which is protected by copyright law, trade secret law, and international treaty, and may not be provided or otherwise made available without proper authorization.

Restricted Rights Legend

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c) of the *Contractor Rights in Technical Data* clause at DFARS 252.227-7013, subparagraph (b) of the *Rights in Computer Software or Computer Software Documentation* clause at DFARS 252.227-7014, subparagraphs (b)(1) and (2) of the *License* clause at DFARS 252.227-7015, or subparagraphs (c) (1) and (2) of *Commercial Computer Software--Restricted Rights* at 48 CFR 52.227-19, as applicable.

Unpublished--rights reserved under the copyright laws of the United States.

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

Intergraph, the Intergraph logo, SmartSketch, FrameWorks, SmartPlant, INtools, MARIAN, and PDS are registered trademarks of Intergraph Corporation. Microsoft and Windows are registered trademarks of Microsoft Corporation. **MicroStation is a registered trademark of Bentley Systems, Inc. ISOGEN is a registered trademark of Alias Limited.** Other brands and product names are trademarks of their respective owners.

Table of Contents

Defining New Reports (Workflow Mechanics and User interface).....	5
Export of Report files (for bulk load later or to a different Catalog).....	16
Adding a Filter Based Query to a Report	19
Adding additional property queries.....	37
Allow Runtime Filter Selection (ARFS) - 1st Example	43
Allow Runtime Filter Selection (ARFS) - 2nd Example.....	49
Assigning queries to different sheets in the Report (Multi-sheet reports)	67
Metadata Browser/Data model.....	80
Intro to XML and Cooktop 2.5	80
XML Intro.....	80
CookTop 2.5	80
Understanding Report Templates	81
Report Definitions/Excel Data (Reports.xls).....	81
Report Templates (*rtp).....	83
Query (*rqe)	83
Filter based Query.....	84
SQL based Query.....	85
Filter based SQL Query	87
SQL Query With Parameters	90
Looking at the XMLs:	95
Query Parameters (*.rqp).....	98
Formatting (*.rfm).....	101
Coordinate Systems	101
Display Units & Precision	103
Layout (*.xls).....	104
Display Configuration (*.rdy)	106
Seed Data: Delivered Base Templates and Delivered Reports	107
Query Specific Settings	110
Grouping.....	110
Sorting	123
Macros	124
Differential Reporting in SmartPlant 3D	133
Hierachal Reporting	138
Labels.....	144
Tool Tip Labels	144
Embedded Labels.....	152
Special Formatting in Reports	160
Understanding the data model.....	171
Remarks or notes from objects	171
COM Query	171
SQL Query.....	183

Alternative method for those who cannot write SQL but who use MSSQL 2005: ..185

Advanced Queries**191**

1.	DestinationInterface = “CONSTANT”	191
2.	Virtual relationships.....	197
3.	Recursion with <i>Implements</i> exitcondition.....	205
4.	Using user interface/attributes	210
5.	Using embedded labels	211
6.	Filter First and Last (Optional)	218
7.	Recursion with Exit condition Depth (Optional).....	222
8.	Example of using recursion with Concatenate (Optional).....	233

Advanced Labels Formatting**243**

1.	Using a PHYSICAL token with UOM to format output in proper units	243
2.	Outputting codelisted values.....	252
3.	Using <i>ToParse</i> = yes for embedded labels	258
4.	Using a POINT object.....	259
5.	Conditional Formatting - I	264
	Part 1: Raw Values	264
	Part 2: Formatted Values	270
6.	Conditional formatting - II.....	272
7.	Using a VECTOR object for orientations (Optional)	280
8.	Using a FORMAT_EXPRESSION	285

Appendix A..........**289**

RTF_Label Schema Definition.....	289
<!-- CONDITION -->	291
<!-- VECTOR -->	291
<!-- ORIENTATION -->	293

Appendix B**294**

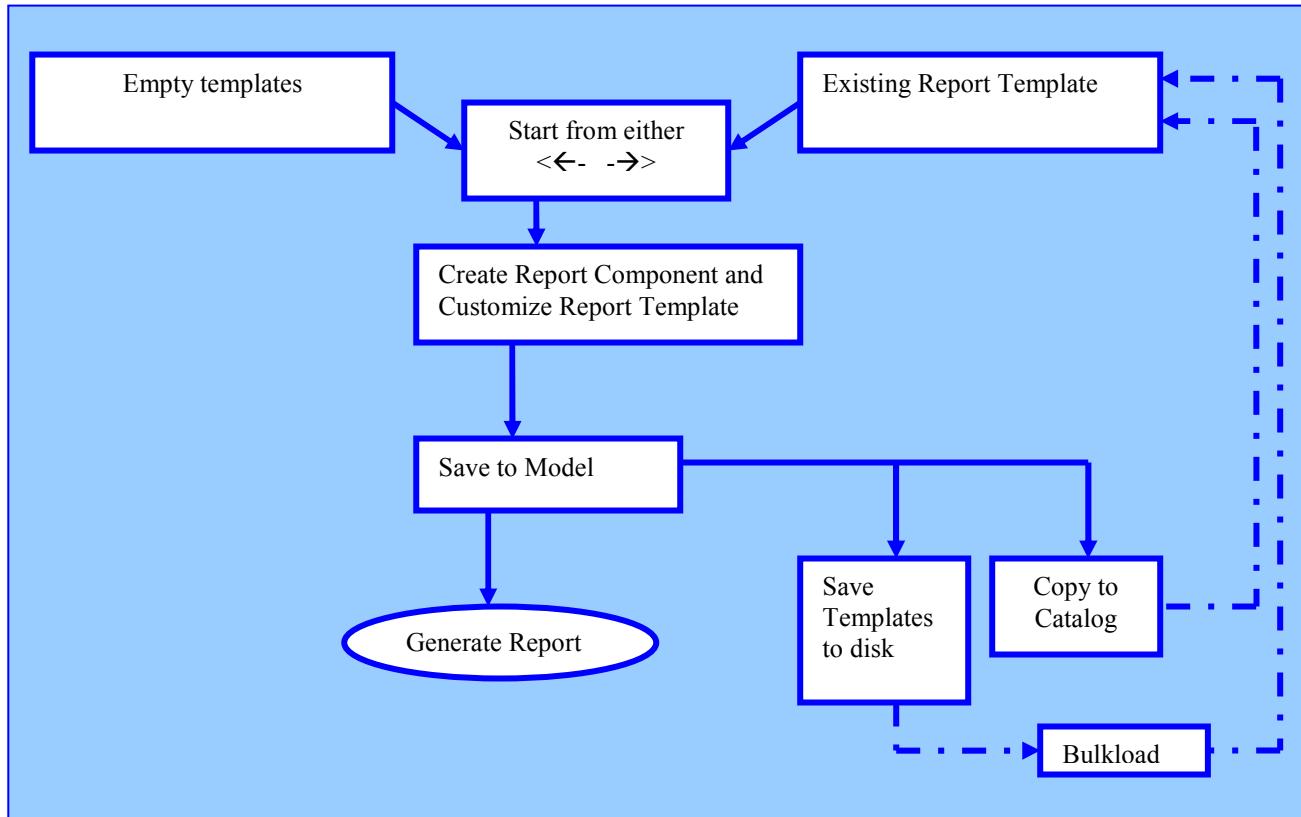
Visual Basic Format Functions.....	294
User-Defined String Formats (Format Function)	294
User-Defined Date/Time Formats (Format Function)	295
FormatCurrency Function.....	296
FormatDateTime Function.....	297

Appendix C**299**

Application specific data models.....	299
Equipment.....	299
Piping.....	300
Structure.....	301
Electrical Cable.....	302

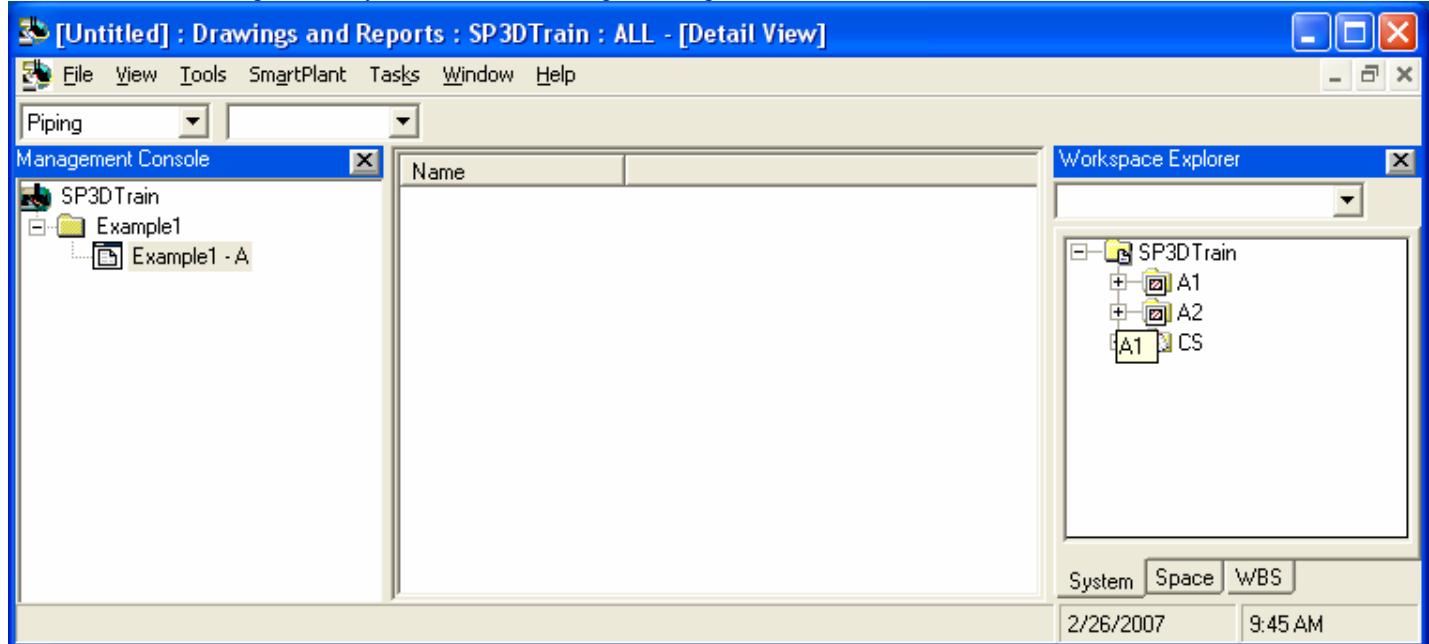
Defining New Reports (Workflow Mechanics and User interface)

The following sections are meant to familiarize you with the primary workflows for starting the creation of a new report, saving it to the model, saving it to the catalog, and realizing where the files are located in the even that you need to send them to another live catalog.

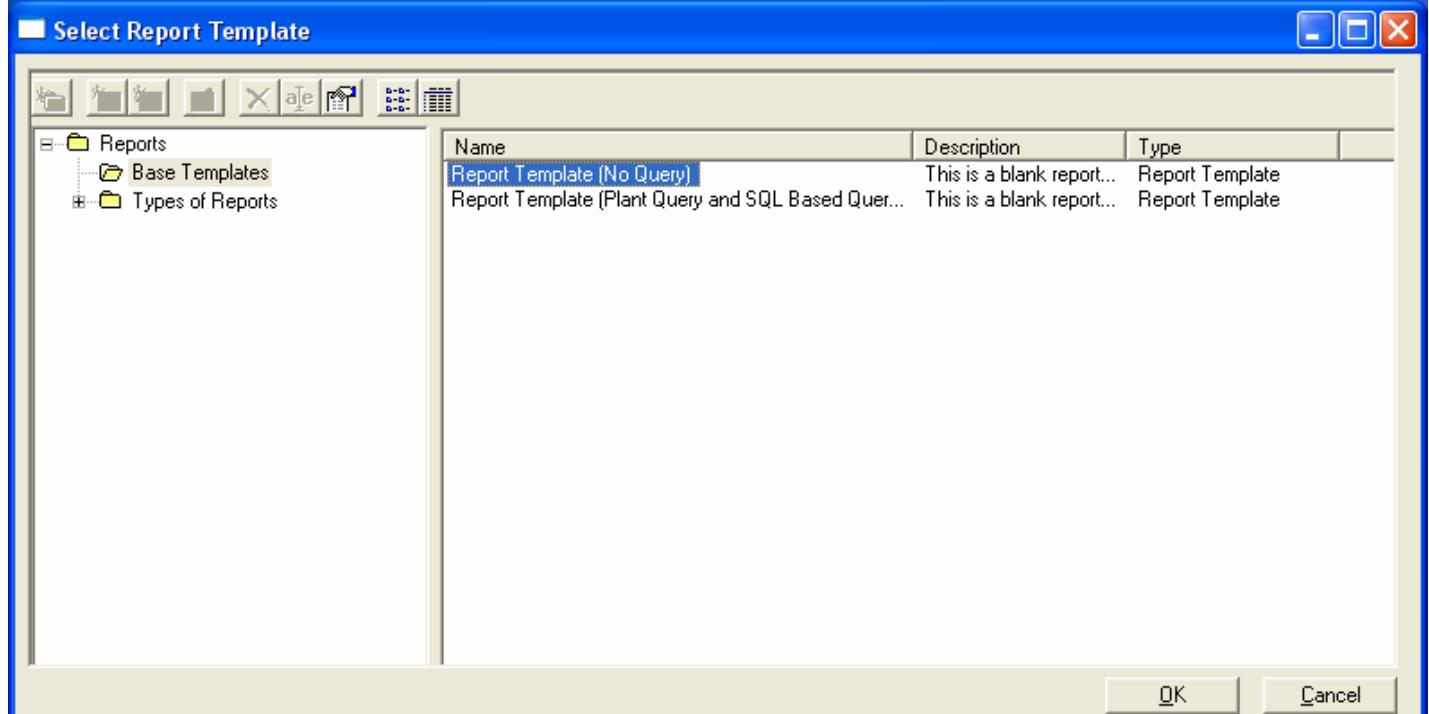


Creating Model Reports

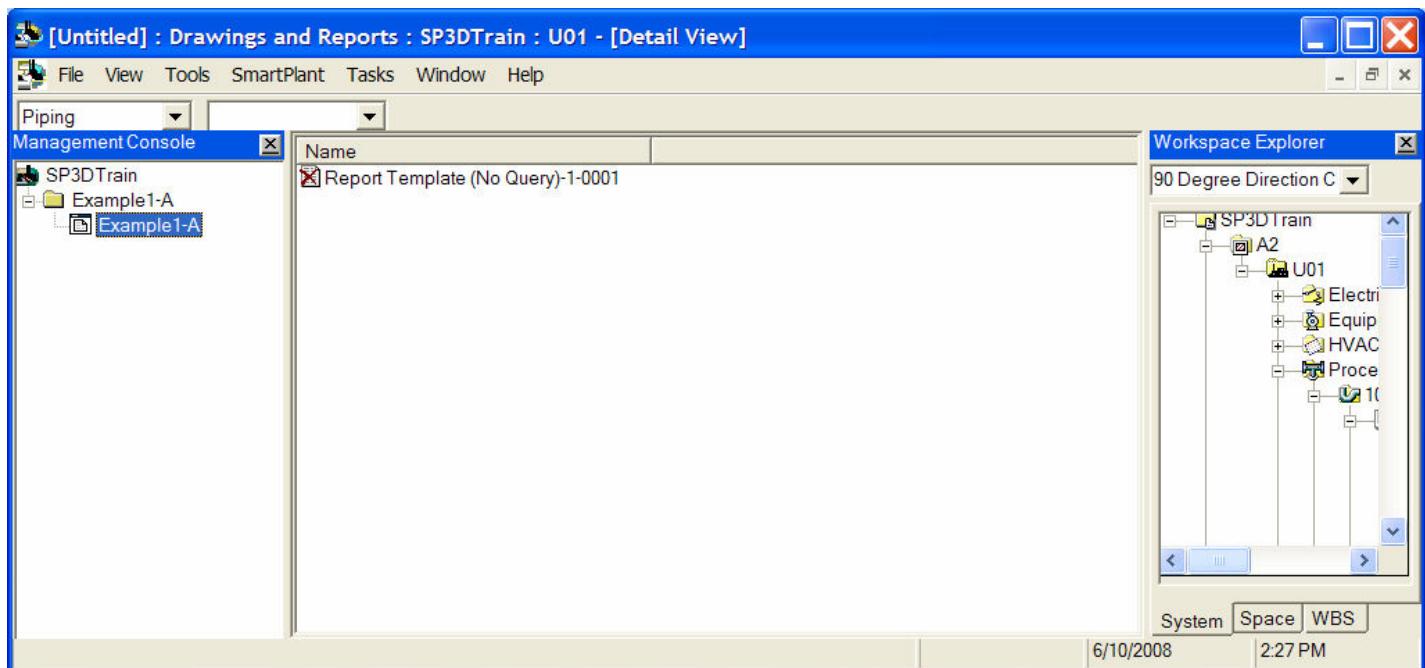
1. Build the following Hierarchy in the SP3D Drawings and Reports environment:



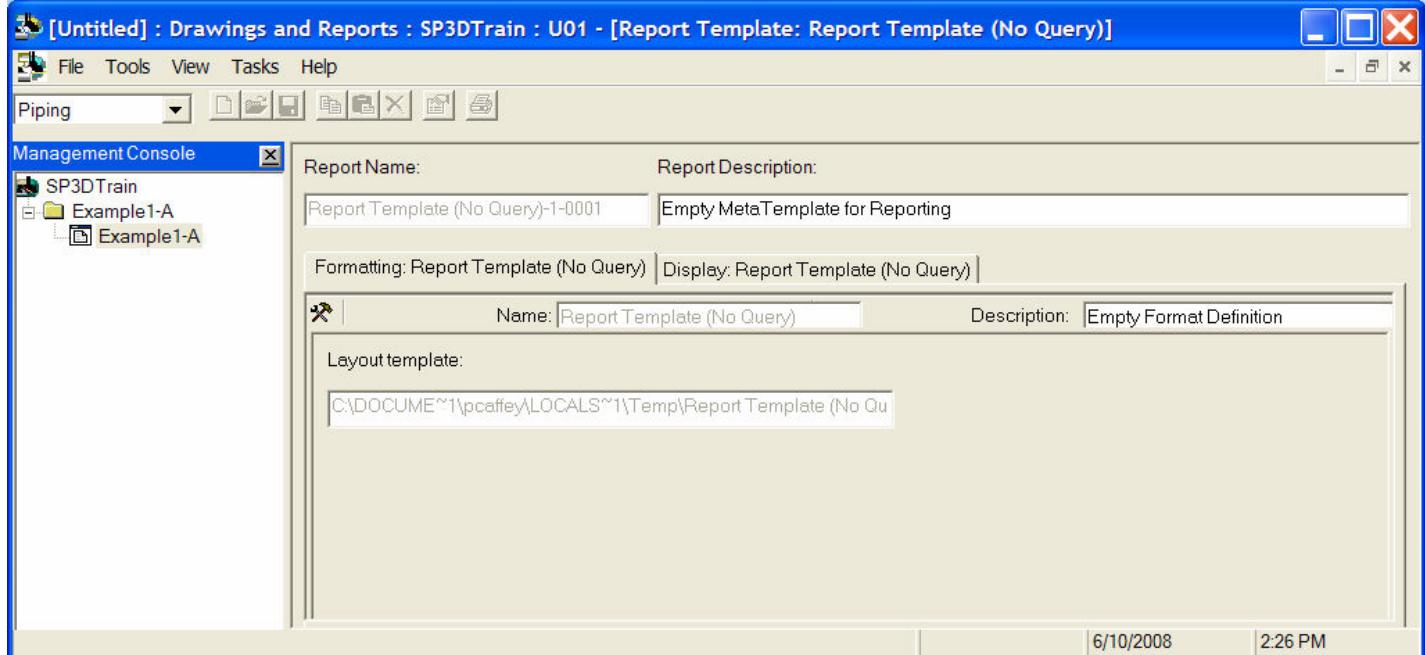
2. Right Mouse and Select Create Report



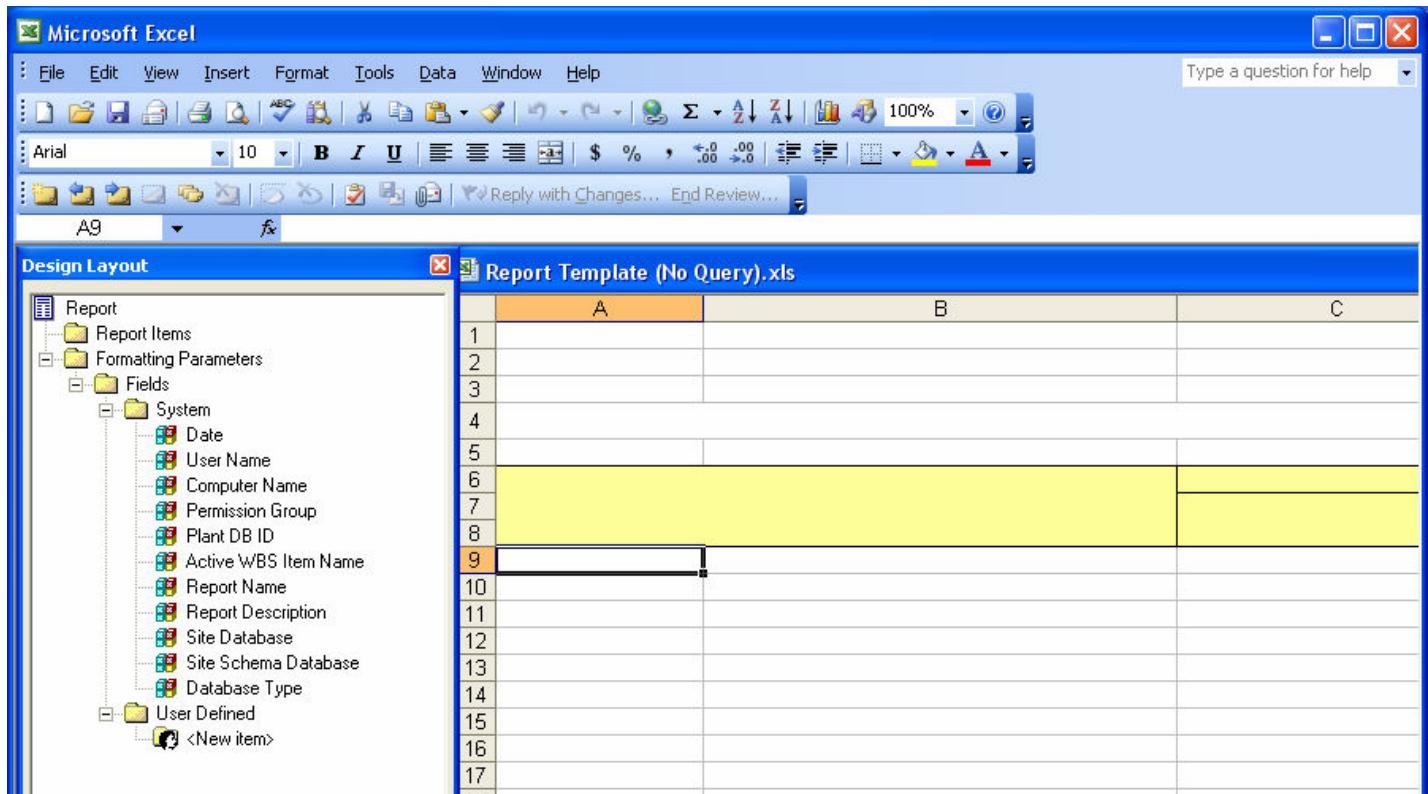
3. Select "Report Template (No Query) from the \Base Templates Catalog Node.



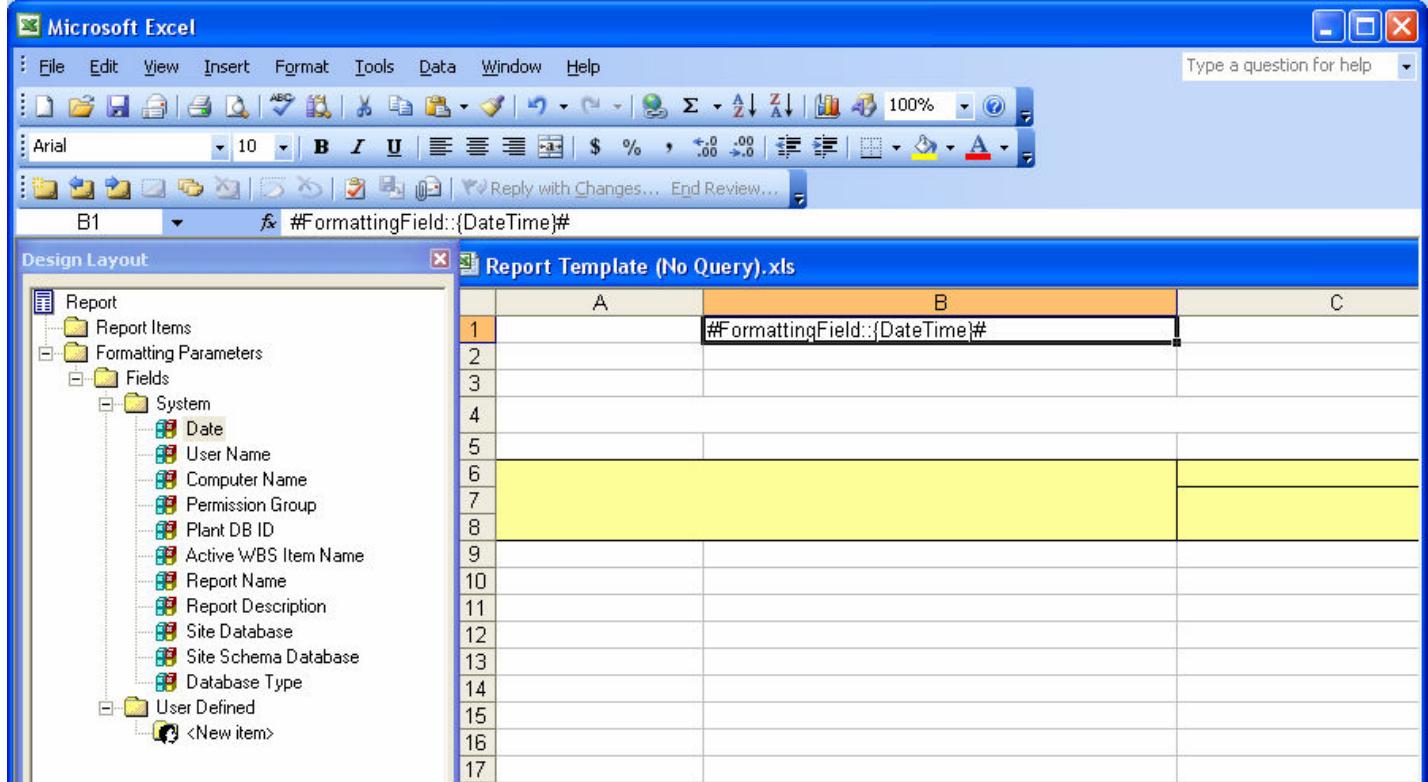
4. Right mouse on the Report Template in the middle pane of the Drawings and Report Environment and select Edit Template
5. Note: You can close the Workspace Explorer at this time and give yourself some more working space.



6. Note the two Tabs which are now displayed.
7. Select the Design Layout button, . Excel will start and display the Report Template (No Query).xls as well as the Design Layout browser:



8. The browser and Excel sheet interact primarily through drag and drop operations. Drag the **Date** to cell B1.

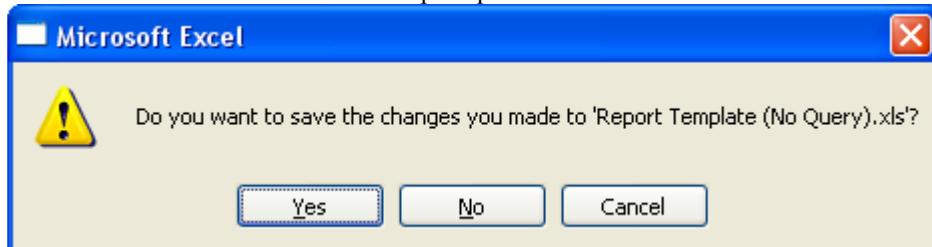


9. Notice after this operation is performed that cell B1 now contains data. This data will indicate to the software where to place the information on the final report when it is executed.

10. Complete Drag and Drop operations until your formats are as below. Also, you should key-in text in the A column cells as exemplified below to provide a title for the information displayed in Column B.

	A	B	C
1	Date:	#FormattingField:{DateTime}#	
2	User Name:	#FormattingField:{UserName}#	
3	Plant DB:	#FormattingField:{PlantName}#	
4	Report Name:	#FormattingField:{ReportName}#	
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			

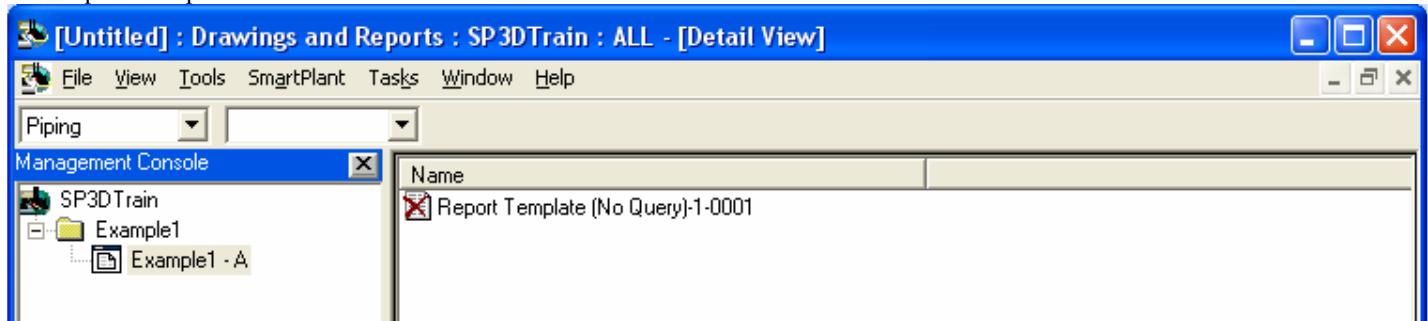
11. Close Excel and select save when prompted



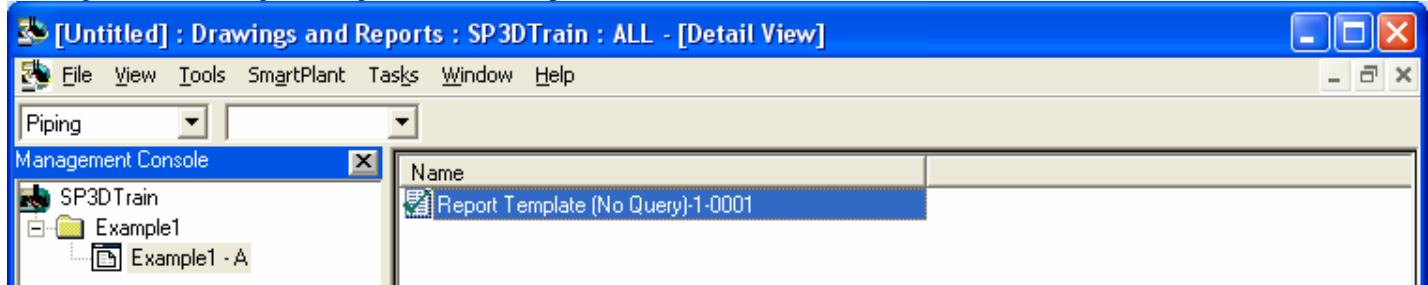
STOP. READ. DO.

Before clicking into the Drawings and Report Hierarchy, you should go to File -> Save Report Template. Until you have done this, these changes you have made are not fully committed and they will be LOST. If you run your report and it is completely empty, and when you return to the editing of the XLS sheet and it does not show your changes, more than likely this was the missing step in your workflow.

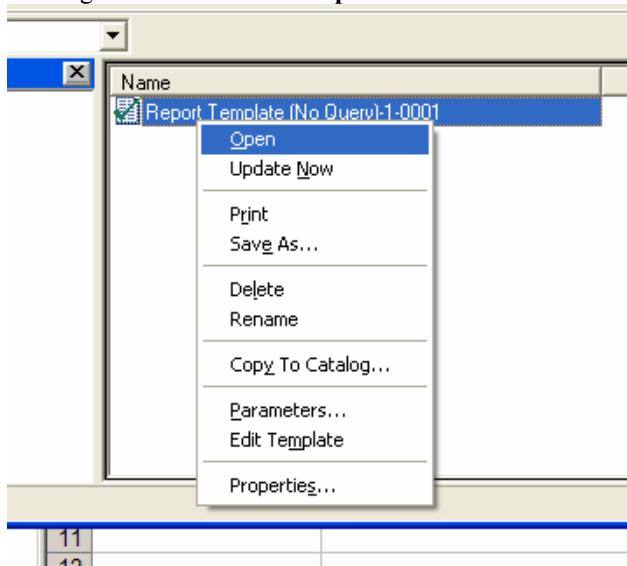
12. Left mouse on the Example 1 folder – this will exit out-of the Edit Template tabs, and then reselect the Example 1 – A Report Component.



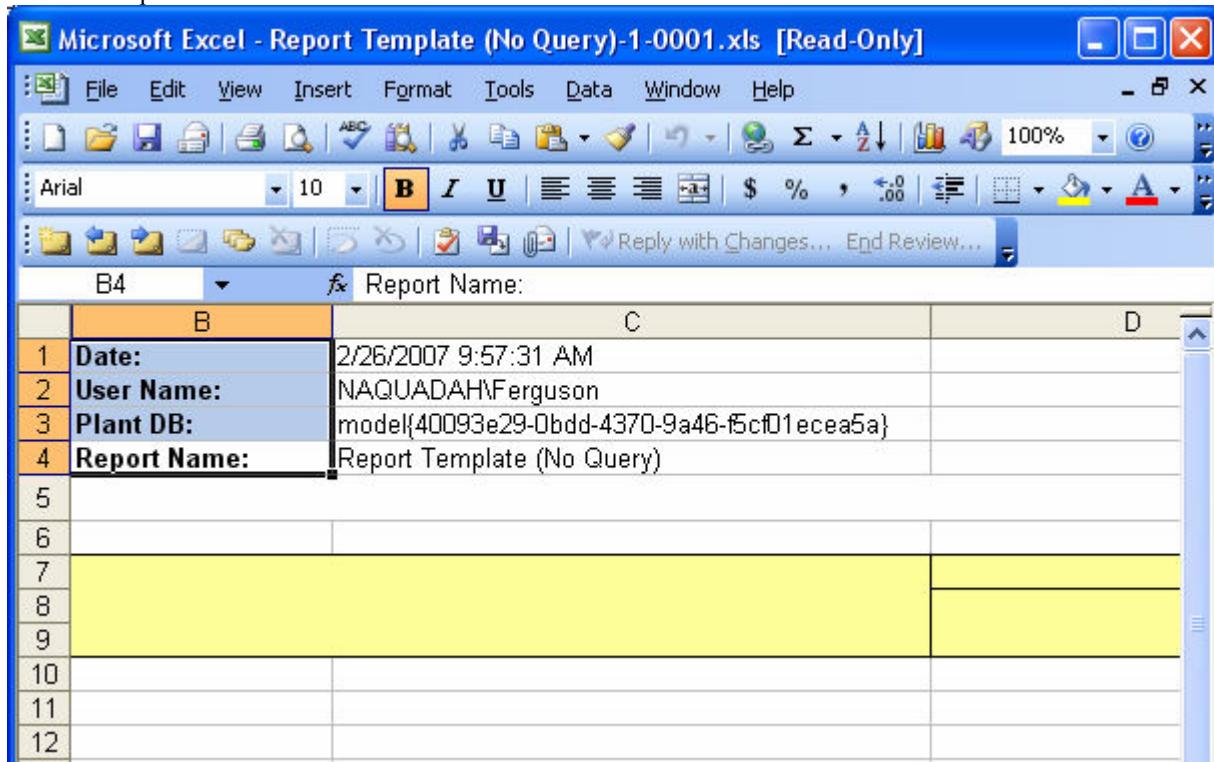
13. Right mouse on Report Template and select **Update Now**



14. Right mouse and Select **Open**



15. The Report will now look similar to this:

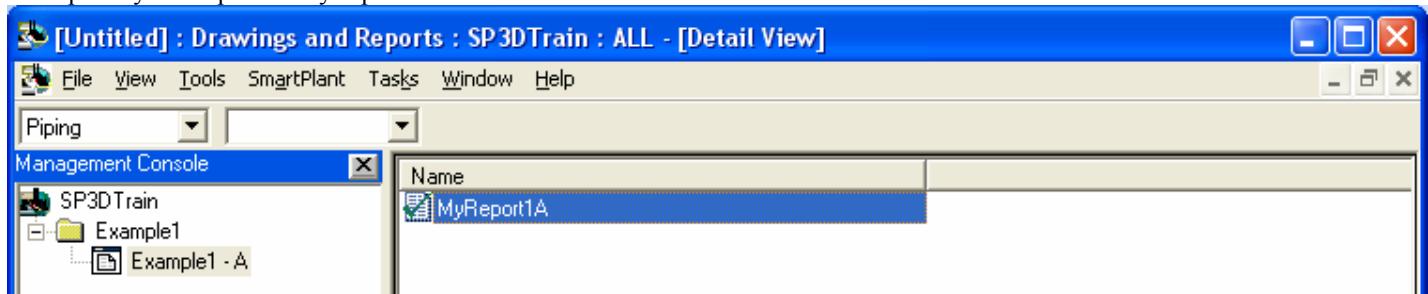


A screenshot of Microsoft Excel showing a report template. The window title is "Microsoft Excel - Report Template (No Query)-1-0001.xls [Read-Only]". The menu bar includes File, Edit, View, Insert, Format, Tools, Data, Window, and Help. The toolbar includes standard Excel icons for file operations, text styling, and data manipulation. The ribbon shows Arial font, size 10, bold style selected. The status bar shows "Reply with Changes... End Review...". The spreadsheet has four rows of data:

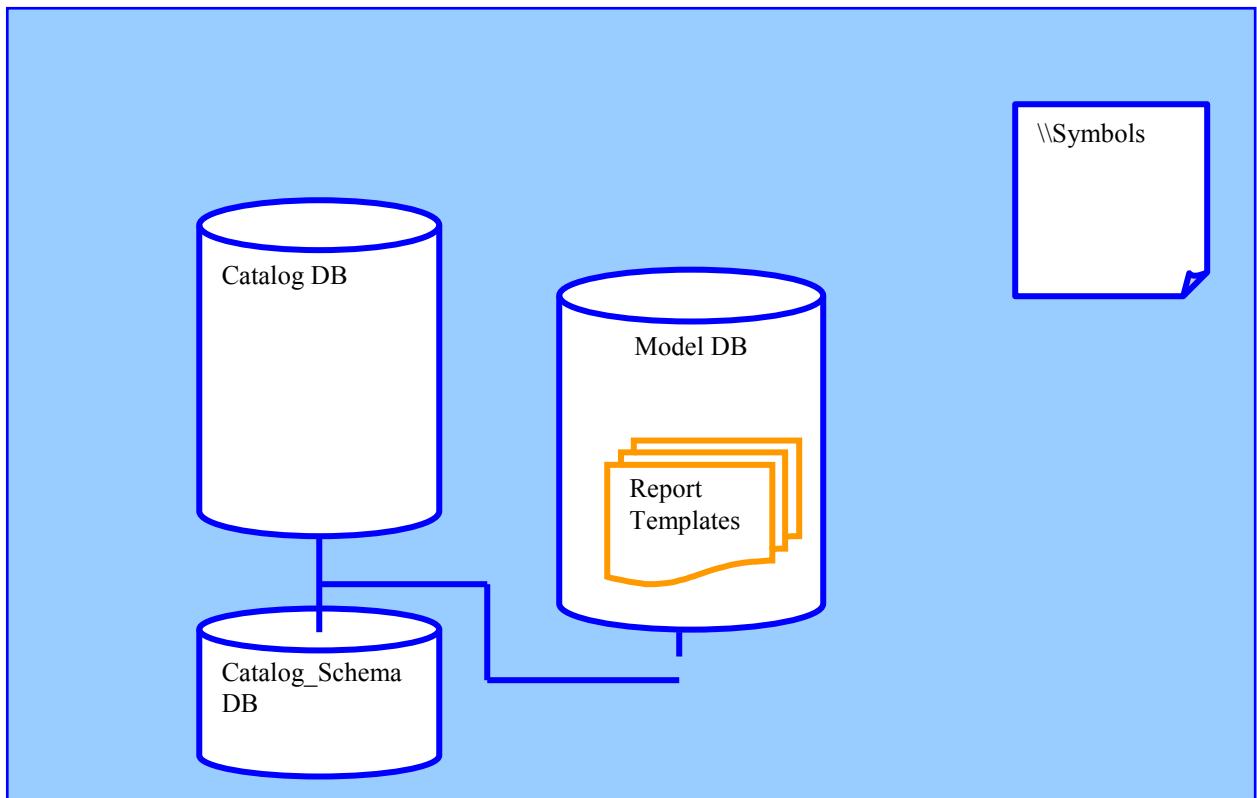
	B	C	D
1	Date:	2/26/2007 9:57:31 AM	
2	User Name:	NAQUADAH\Ferguson	
3	Plant DB:	model{40093e29-0bdd-4370-9a46-f5cf01eceae5a}	
4	Report Name:	Report Template (No Query)	
5			
6			
7			
8			
9			
10			
11			
12			

16. You have successfully created your first custom Report.

17. Now let us customize it further. Since "Report Template (No Query).xls" is no longer a meaningful name for our report, let us rename it to MyReport1A. This is accomplished by Right mousing on the Report and select Rename. Do this and update your Report to MyReport1A.

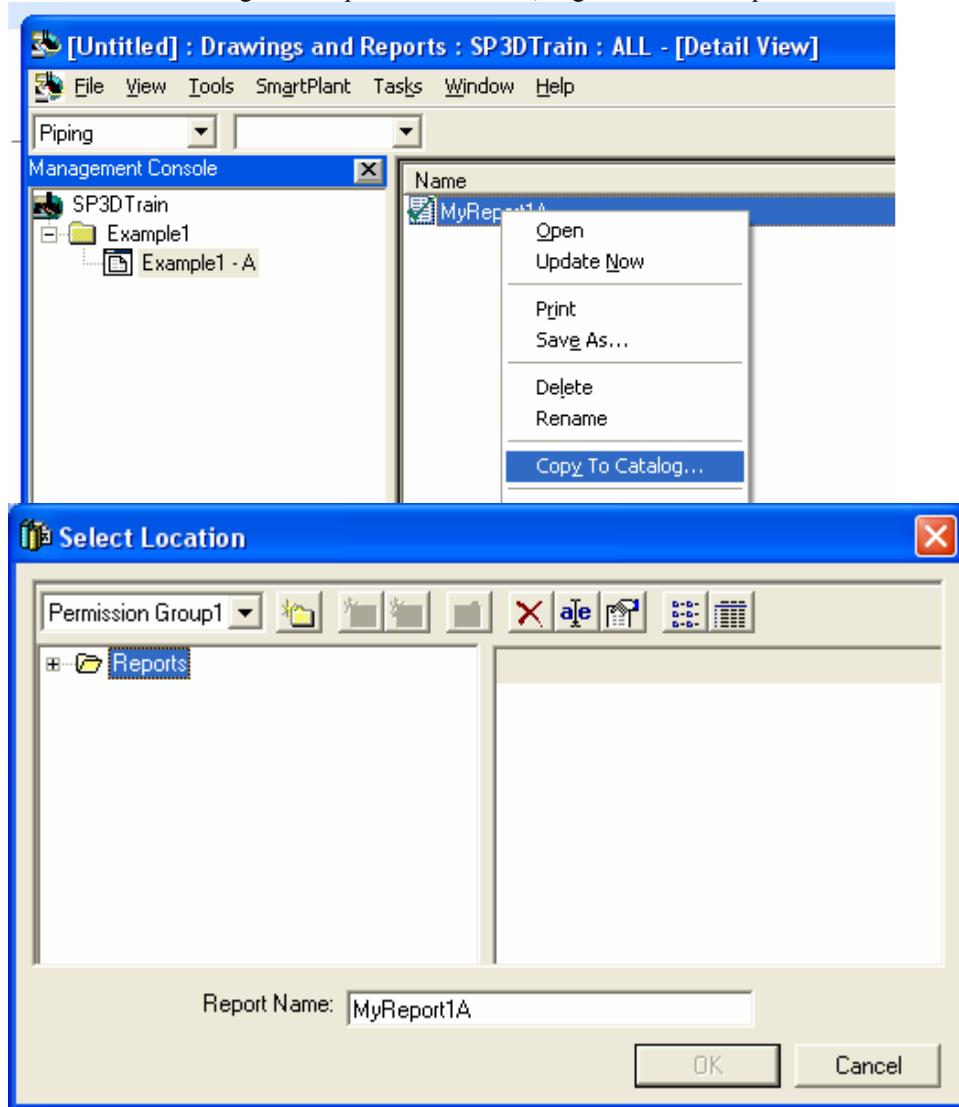


18. You have just completed the creation of a Report that has been written to the Model DB.
19. The files regarding this report are actually stored in the Model DB. When someone wishes to edit them through the GUI, they are moved locally (in temporary directories) and then re-copied back up into the model by the Save Report Template Command.

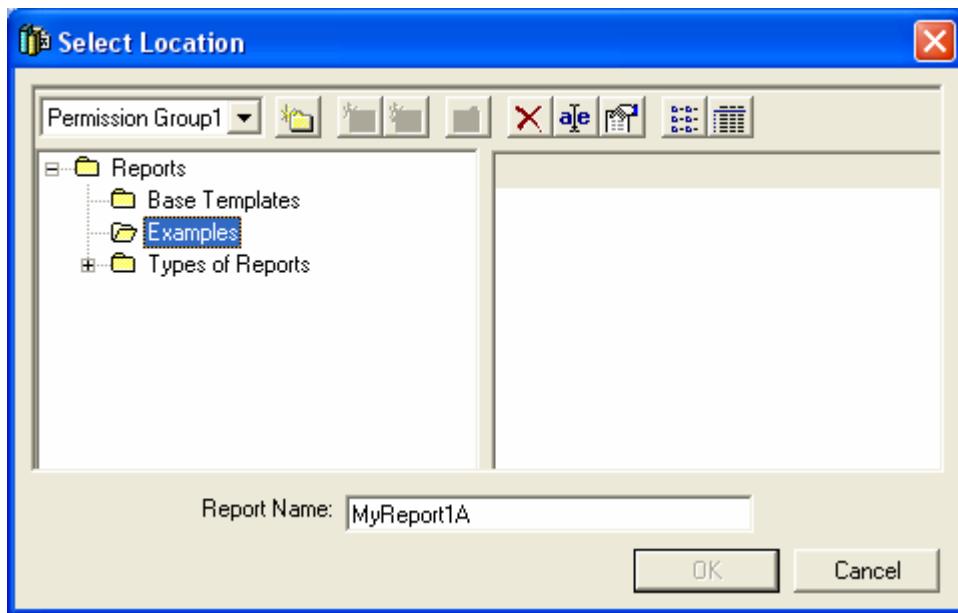


Save to Catalog Reports

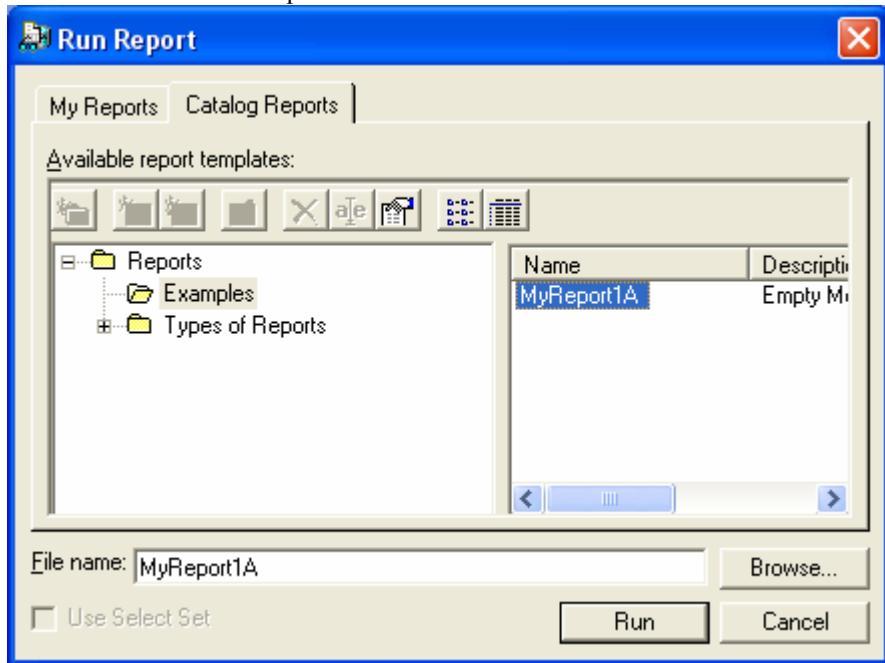
1. From the Drawings and Report Environment, Right mouse on Report1A and select **Copy to Catalog...**



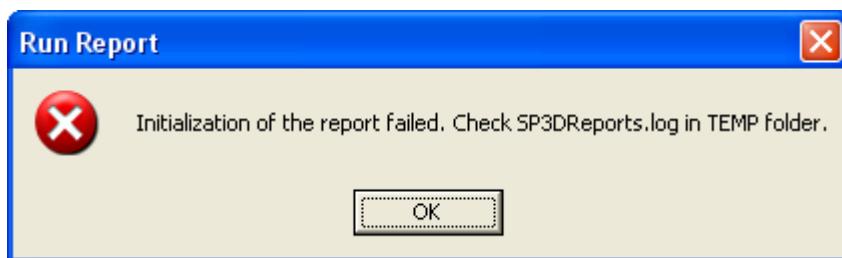
2. Expand the Reports Branch
3. Select the Reports node and then click the New Folder button
4. Create a folder call "Examples"



5. With the Examples folder created and selected, click the OK button to create MyReport1A in the Examples folder.
6. Note the form will close
7. Switch your task to Common
8. Go to Tools->Run Report....

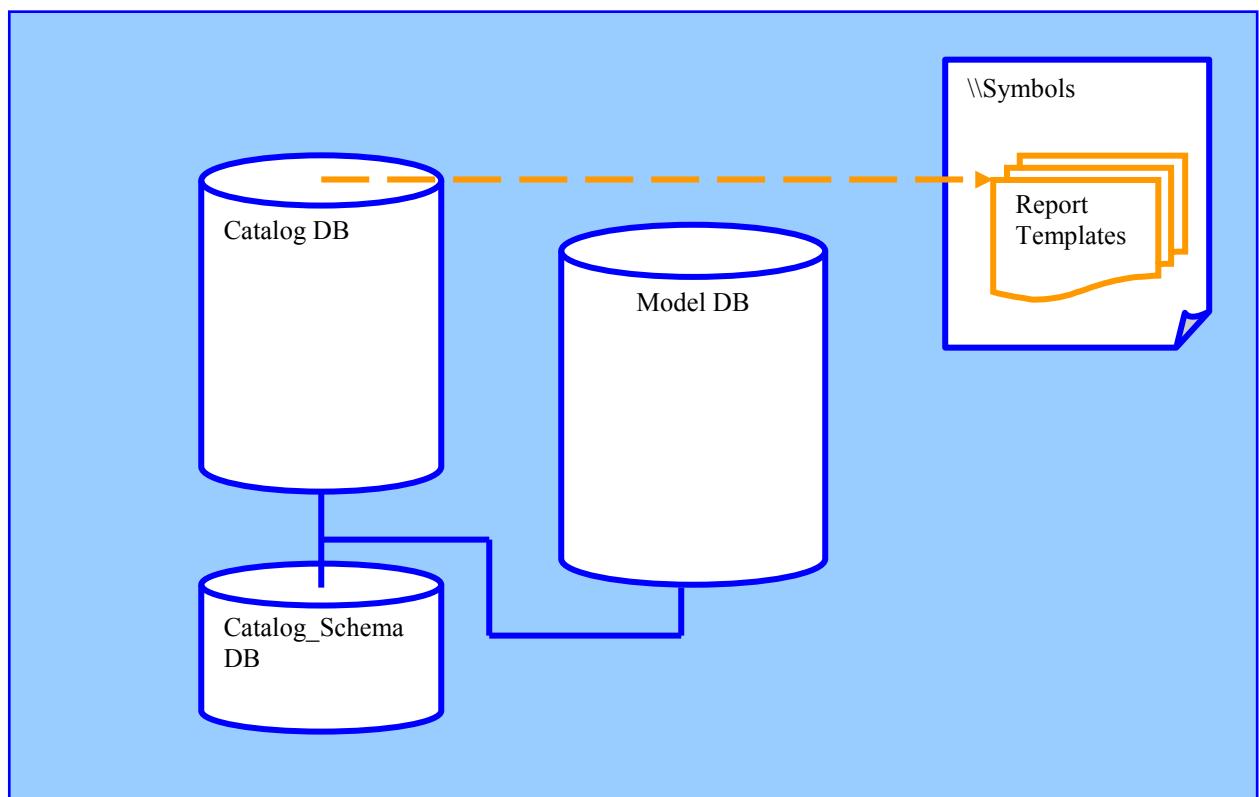


9. Locate the MyReport1A in the hierarchy.
10. Note that it is available for non-persistent report running. Because this particular report does not contain a query, the following mesg will be displayed:



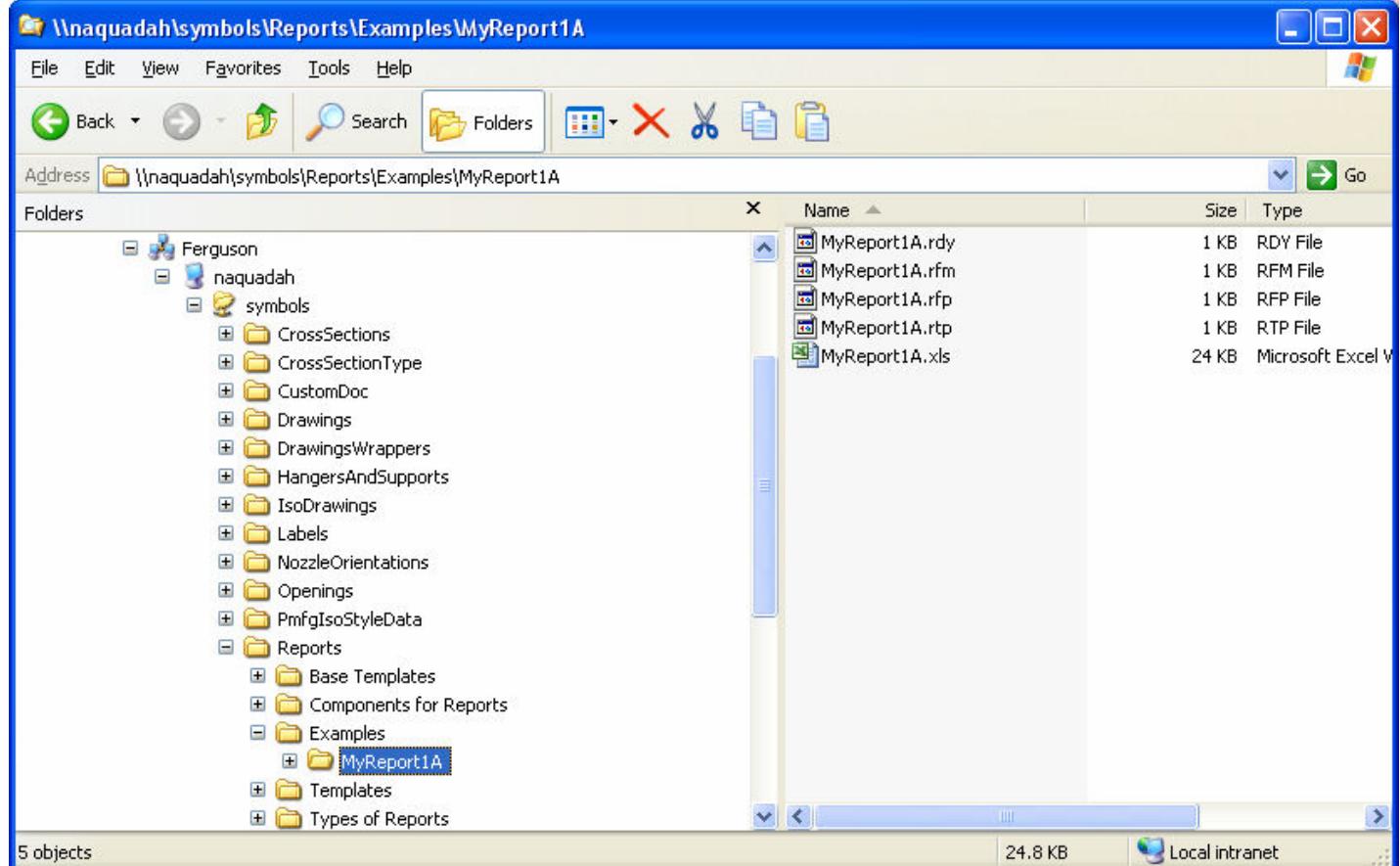
This message can be ignored as we do not have a query in the report. When we add a query we will repeat this process and see the report run successfully. For now, we are just verifying the availability of the report for all users.

11. With the Copy to Catalog command, the Report Template files have been stored on the \\symbols folder and Pointers exist in the Catalog to location of the files.

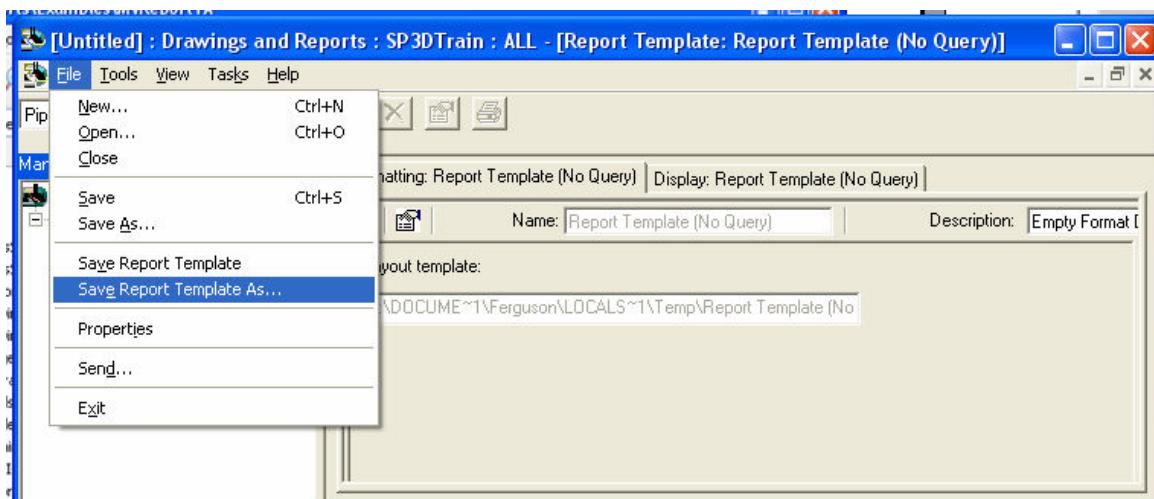


Export of Report files (for bulk load later or to a different Catalog)

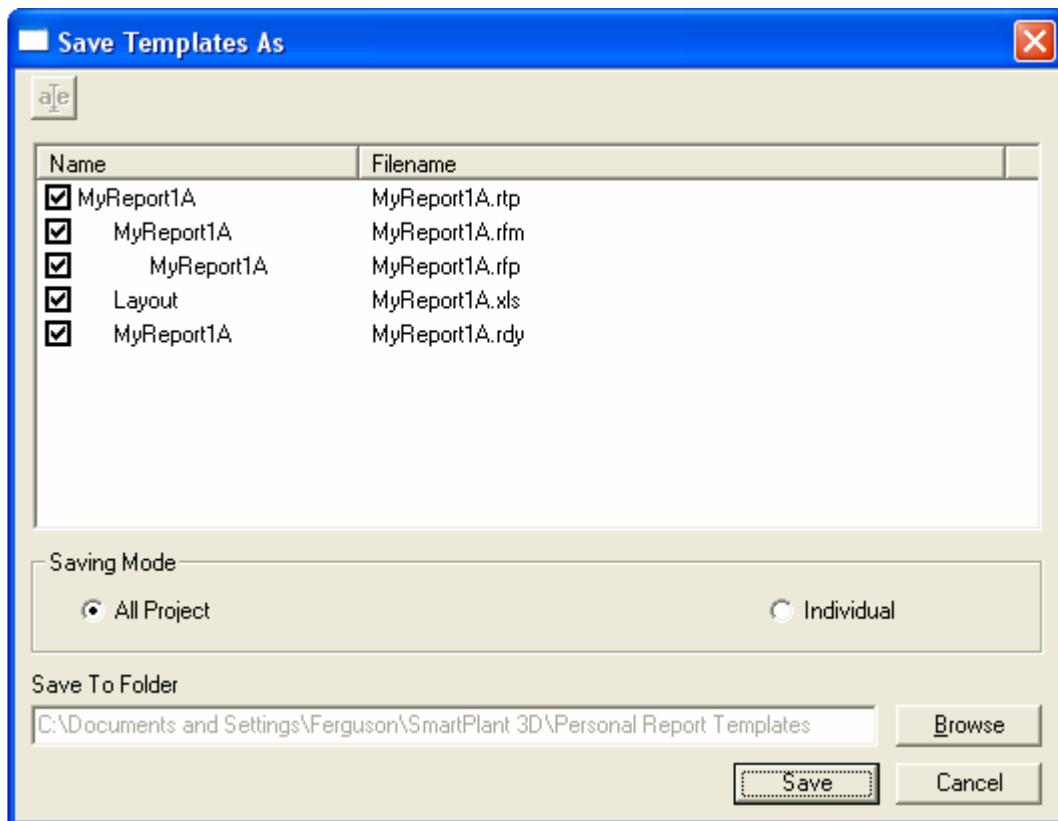
- When you have performed a saved to catalog operation, all of the file type relevant to the report template are automatically copied to the Symbols directory in V7 and are stored under a directory in the same hierarchy location (as seen in the Catalog browser) on the symbol share:
- Switch to your symbol share, and you will find (as screenshot below, all the relevant files pertaining to this report template.



- We will return to this and look at how we can modify the Reports Reference Data Excel workbook to load these into a different catalog. At this time, the live catalog we are working with is already loaded and ready to work with these files (skipping a bulk load step for the current dataset).
- Also while working in the Edit Template mode on the report, you can select File → “Save Report Template As....”



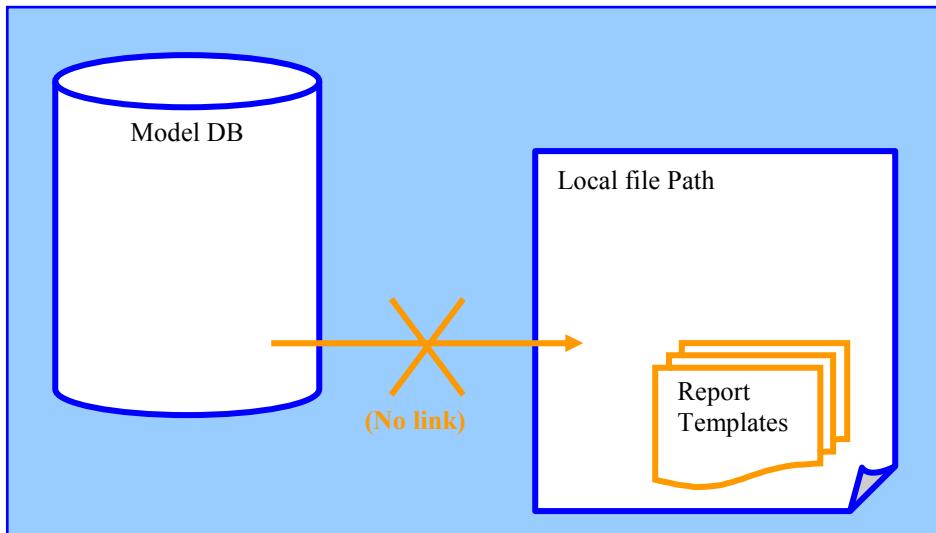
- From here you can pick and choose the files you wish to output with regard to this report. This method is most useful if the report being outputted has not been copied to the catalog and exists only as a model report.



- If your option is set to "All Project" and the de-select a file of importance then you will be warned with the following msg:

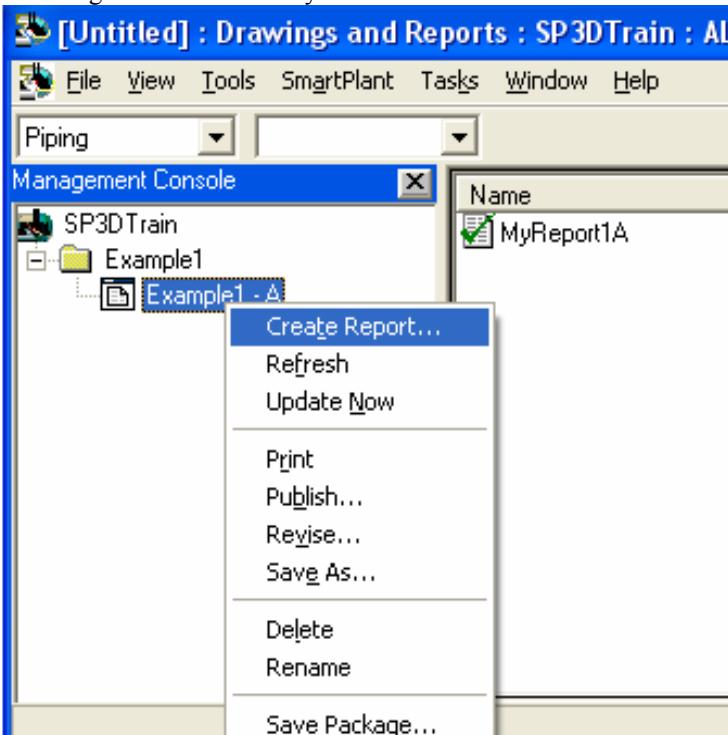


7. If you are saving single files and do not wish to be hassled by this mesg, you can select the “Individual” option. Likewise you can use Browse to set the location to save the files to and you can change the name of the file using the rename button near the top of the form.
8. Save them now just familiarize yourself with the process.

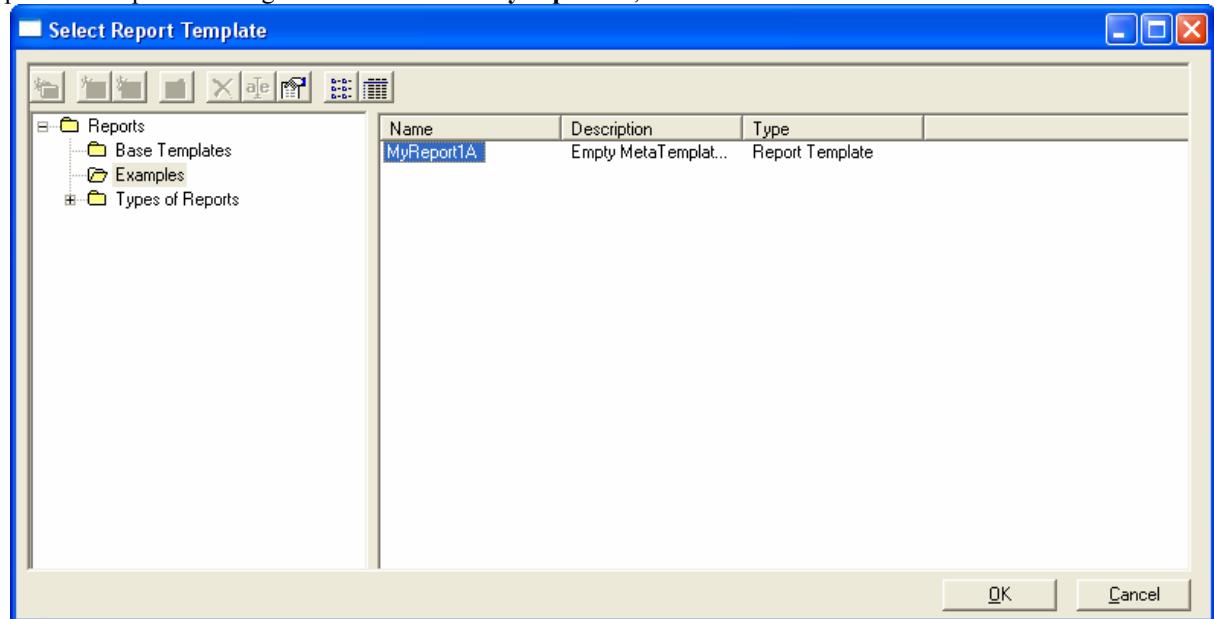


Adding a Filter Based Query to a Report

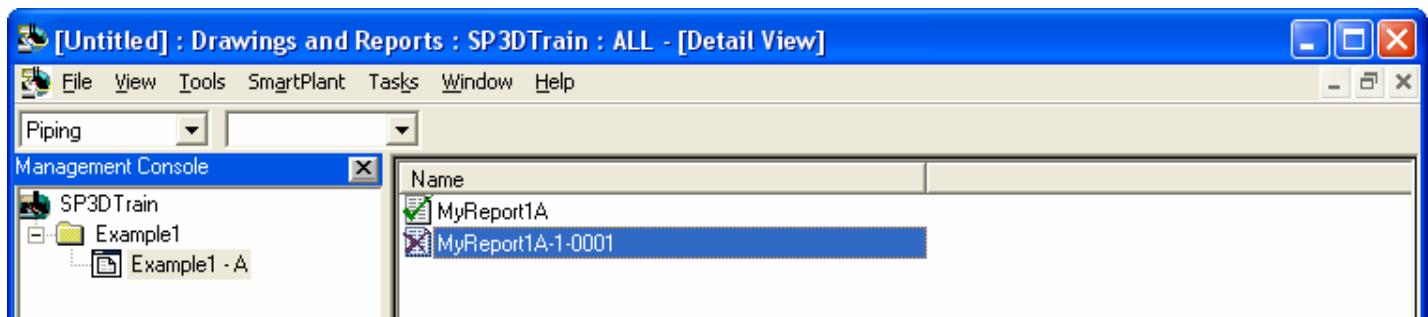
1. Select **Create Report** from the Right Mouse Button fly out menu.



2. Expand the Reports Catalog Branch and Select **MyReport1A**, then click **OK**

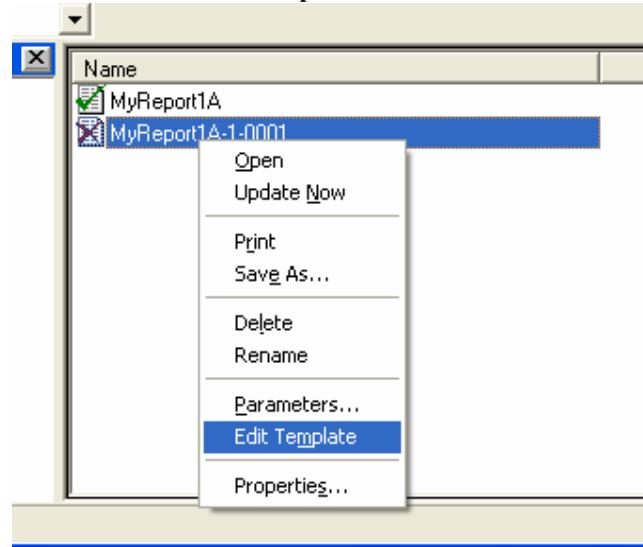


3. Note the addition of MyReport1A-0001

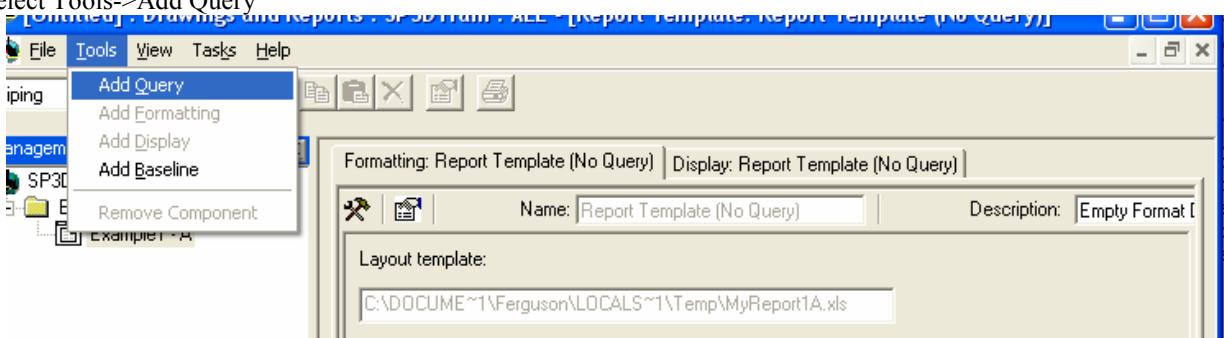


Note: In the next series of steps we will modify **MyReport1A-1-0001** to be more complex than **MyReport1A**.

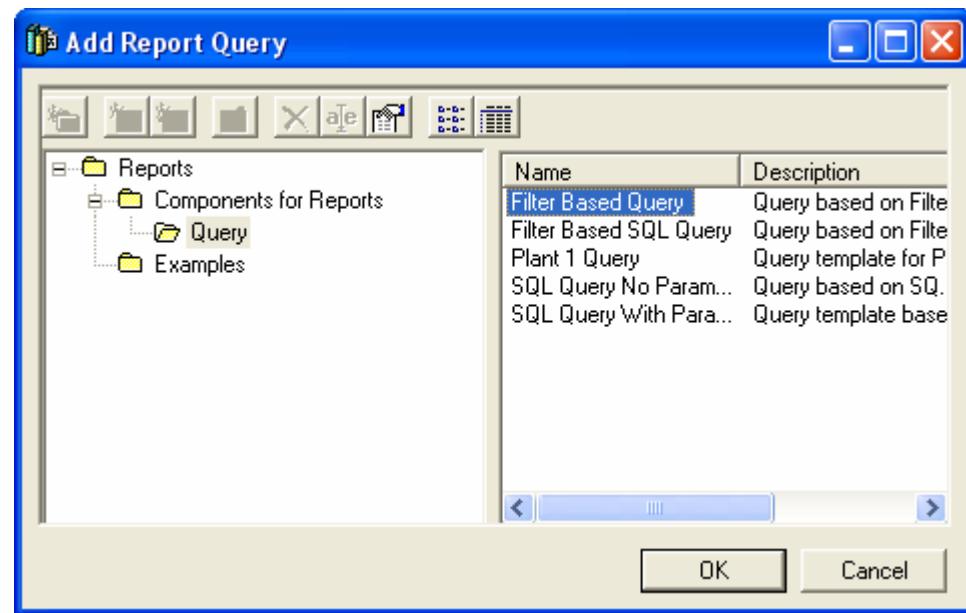
4. Right mouse on MyReport1A-0001 and select **Edit Template**



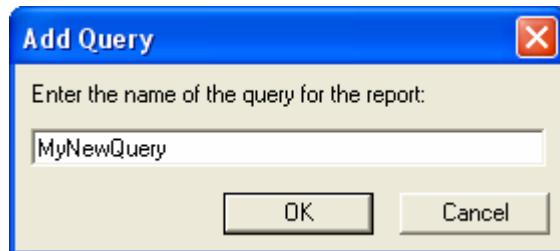
5. Select Tools->Add Query



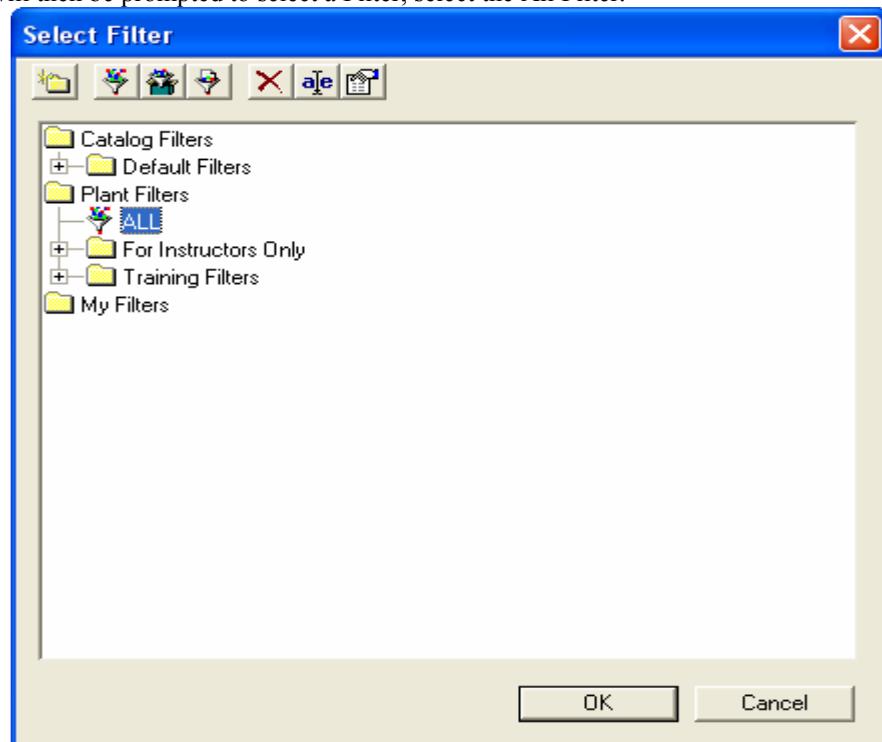
6. From the Add Report Query dialog select '\Reports\Components for Reports\Query\Filter Based Query'



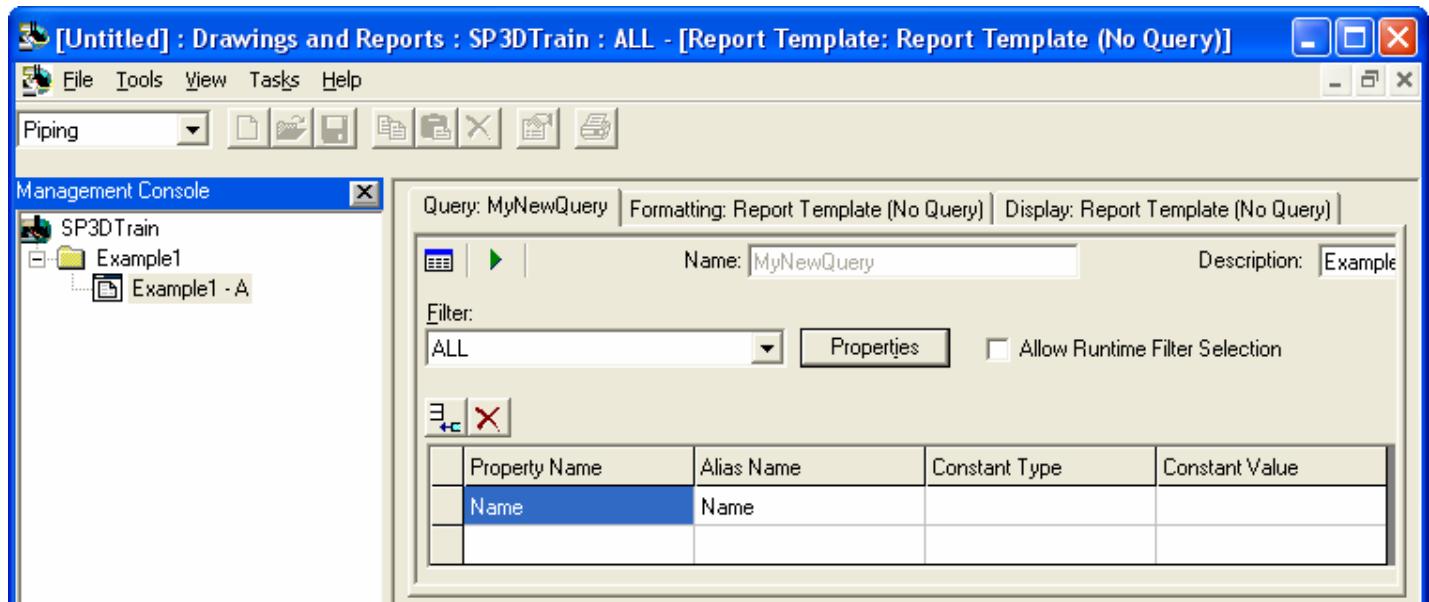
7. Click OK and name the Query:



8. Click OK – you will then be prompted to select a Filter, select the All Filter.

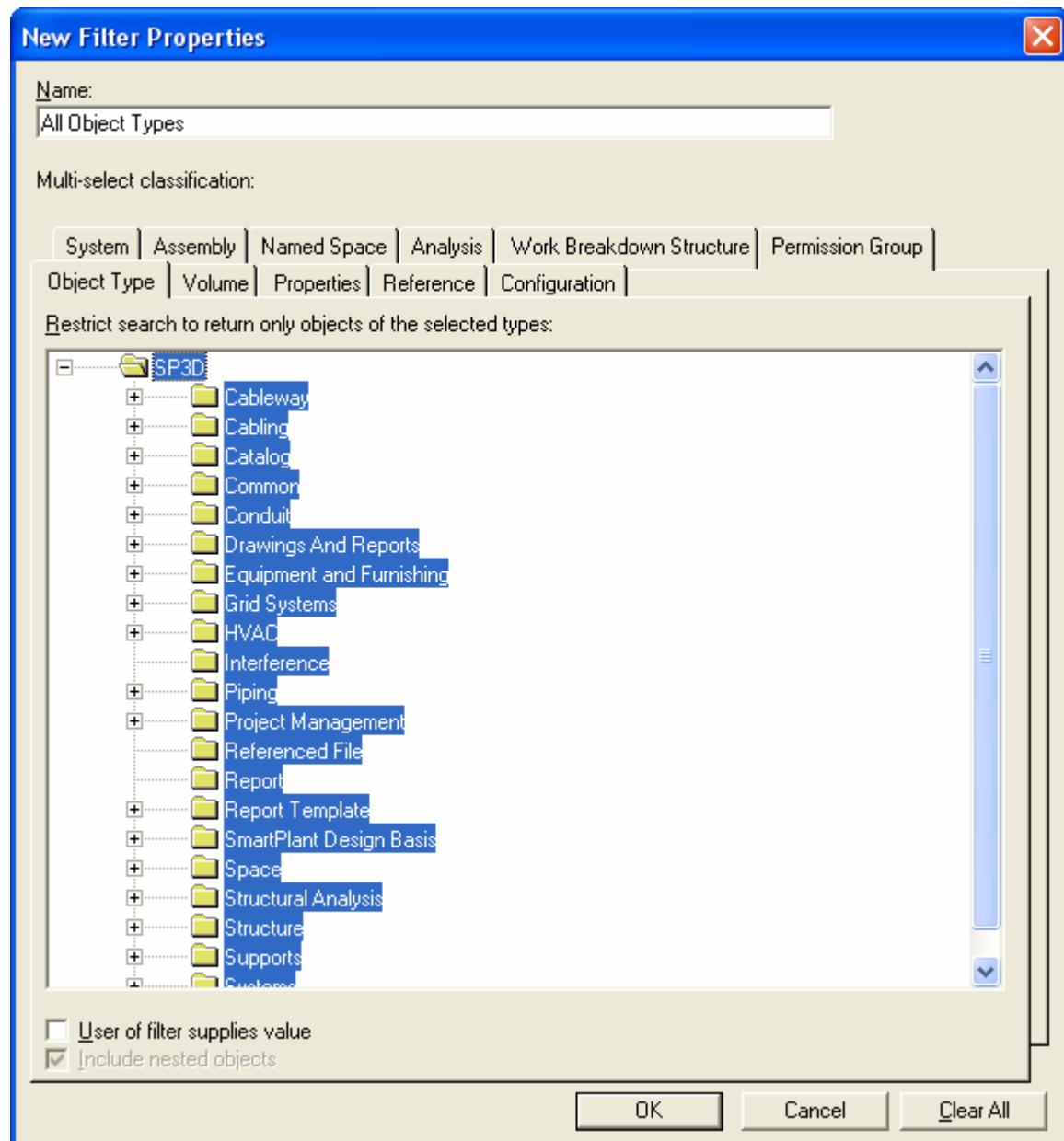


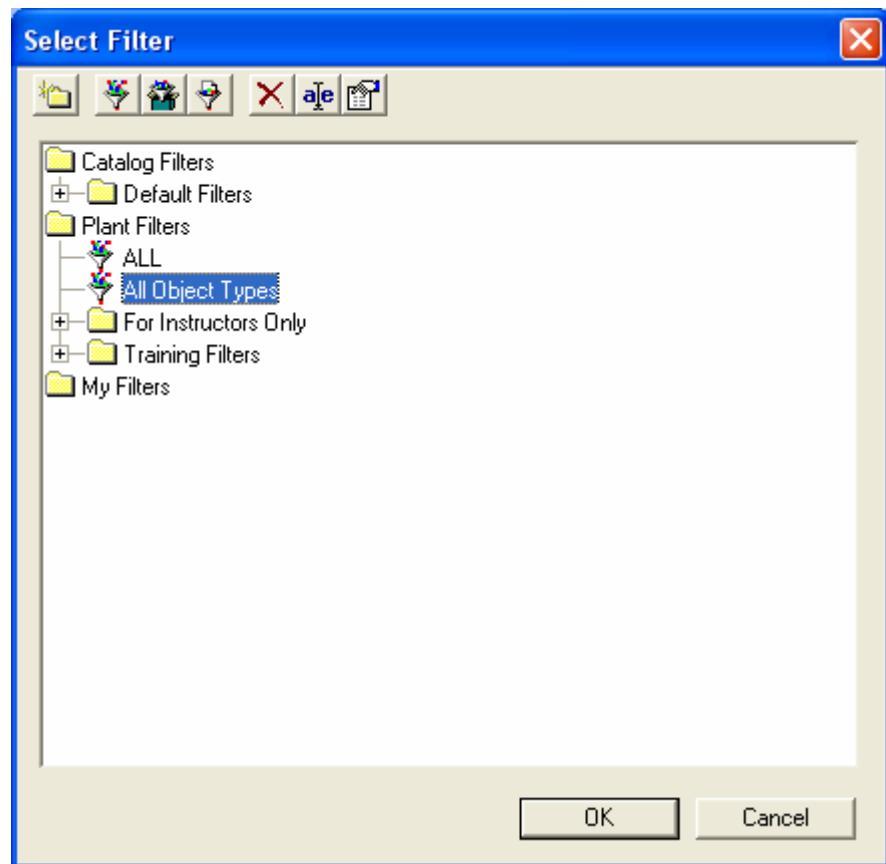
9. Note the addition of the Query:MyNewQuery tab:



10. The scope of the Report will be defined by the Filter drop down value (and the scope of the data that the filter selected represents). This opens the path for asking filter to make the final Reports more dynamic and less static.
11. Let us create a Filter based on all Object Type. So Select “More...” from the filter drop down, and use the filter dialog to create a filter in the Plant Filters folder. Name the filter “All Object Types”, and on the Object Type tab select the Root node in the tree.

Note: We are not using the “Create new filter....” In the drop down on this dialog because that would create the filter in the **My Filters** folder. You should always create Plant or Catalog filters for use with Reports (pick “More...”).





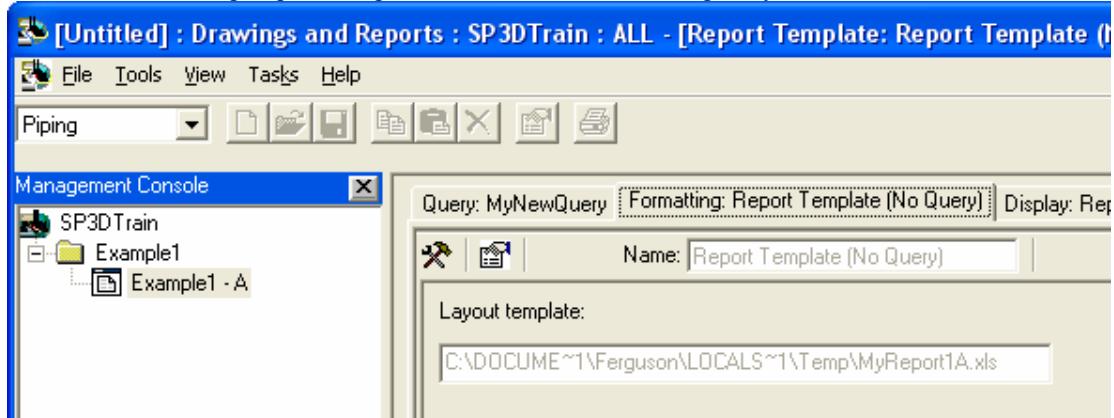
12. Click OK, notice that the filter drop down now displays “All Object Types”

Property Name	Alias Name	Constant Type	Constant Value
Name	Name		

13. By default you will see a Property Name = Name and Alias Name = Name row in the list. This is an automatically created query to pull back the Object Names matching the Filter Query.

14. Now, at this point the Filter exists but is not being utilized by the Report. If we were to run the report again at this point, no new information would appear on the Excel Workbook output. So we will need to add the Query (and define it's location) on the Excel Workbook.

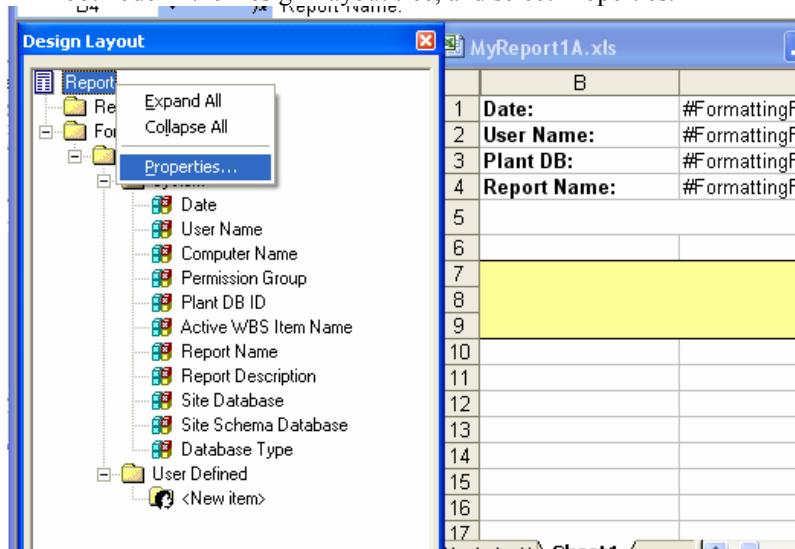
15. From the “Formatting Report Template....” Tab, select the “Design Layout” button.



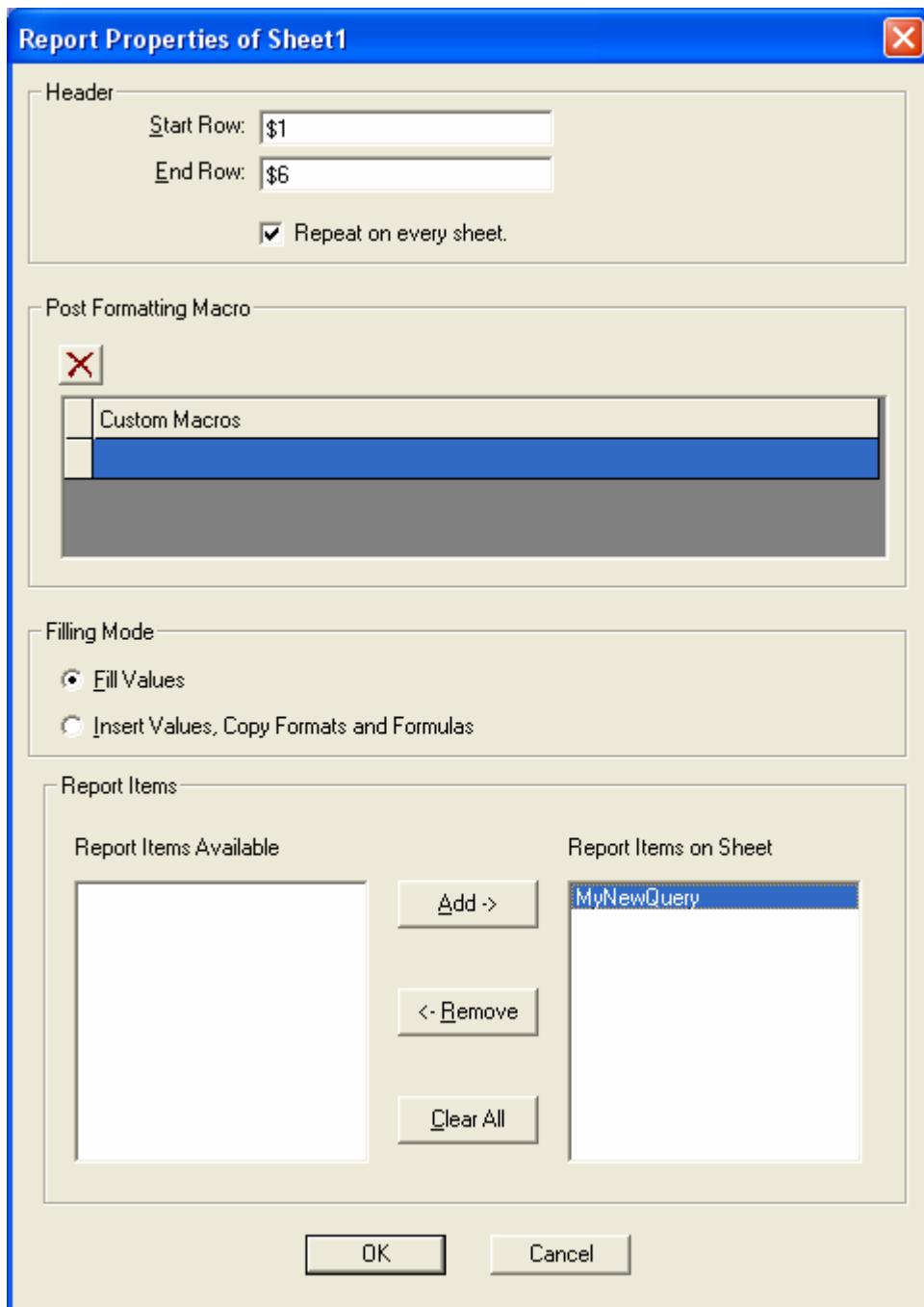
16. Excel will start.

	B	
1	Date:	#FormattingField::{D}
2	User Name:	#FormattingField::{U}
3	Plant DB:	#FormattingField::{P}
4	Report Name:	#FormattingField::{R}
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		

17. While we have added the Query to the template, we have not made it available for Sheet1. First, right mouse on the Report root node in the Design Layout tree, and select Properties.



18. From the Report Properties of Sheet1 form, select the MyNewQuery in the Report Items Available list, and then click Add→ until the form resembles the form below:



19. Click OK.

Note: The Attributes Branch has been added with nodes "Name" and "OID" prominently displayed.

Design Layout

- Report
- Report Items
 - MyNewQuery
 - Attributes
 - Name
 - OID
 - Parameters
 - FilterName
 - FilterLocation
 - FilterDescription
- Formatting Parameters
- Fields
 - System
 - Date
 - User Name
 - Computer Name
 - Permission Group
 - Plant DB ID
 - Active WBS Item Name
 - Report Name

MyReport1A.xls

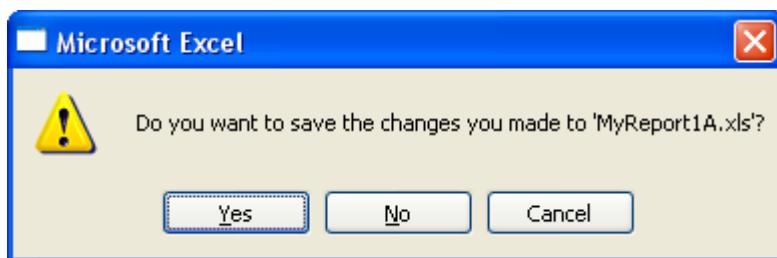
	B	
1	Date:	#FormattingField:{DateTime}#
2	User Name:	#FormattingField:{UserName}#
3	Plant DB:	#FormattingField:{PlantName}#
4	Report Name:	#FormattingField:{ReportName}#
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		

20. The branch Parameters has been added with nodes “FilterName” “FilterLocation” and “FilterDescription”
21. Each of these represent items that can be added to your report as a result of adding the MyNewQuery into this template set.
22. Update the Excel Template to including the following additional data holders:

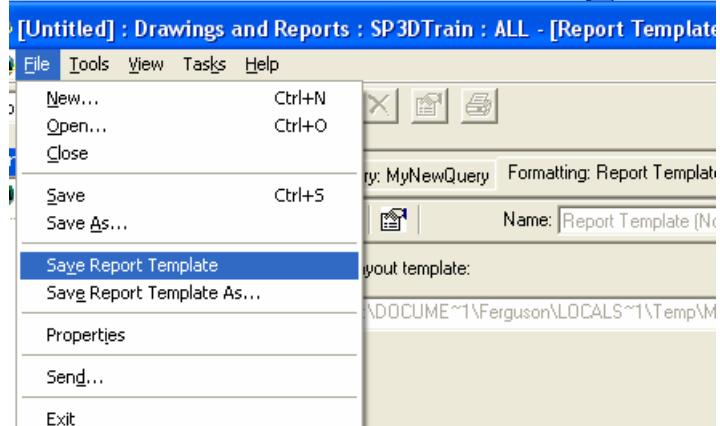
MyReport1A.xls

	B	C	D	E	F	G
1	Date: #FormattingField:{DateTime}#		Filtername: #MyNewQuery:{FilterName}#			
2	User Name: #FormattingField:{UserName}#		FilterLocation: #MyNewQuery:{FilterLocation}#			
3	Plant DB: #FormattingField:{PlantName}#		FilterDescription: #MyNewQuery:{FilterDescription}#			
4	Report Name: #FormattingField:{ReportName}#					
5						
6						
7						
8						
9						
0				#MyNewQuery:{Name}#	#MyNewQuery:{OID}#	
1						
2						
3						

23. Save the Excel changes



24. Save the changes to the Model Definition for this specific Report



25. Click into the left hand pain of the Drawings and Report environment to close out the Report Template Editing and then run an “Update Now...” on the “Example...” report component.

26. Open the report after it has completed running:

Microsoft Excel - MyReport1A-1-0001.xls [Read-Only]

The screenshot shows a Microsoft Excel spreadsheet with the title bar "Microsoft Excel - MyReport1A-1-0001.xls [Read-Only]". The menu bar includes File, Edit, View, Insert, Format, Tools, Data, Window, Help, and a "Type a question for help" search bar. The toolbar contains various icons for file operations like Open, Save, Print, and cell selection. The ribbon at the top has tabs for Home, Insert, Page Layout, Formulas, Data, Page Break Preview, and Review.

The spreadsheet contains several rows of data:

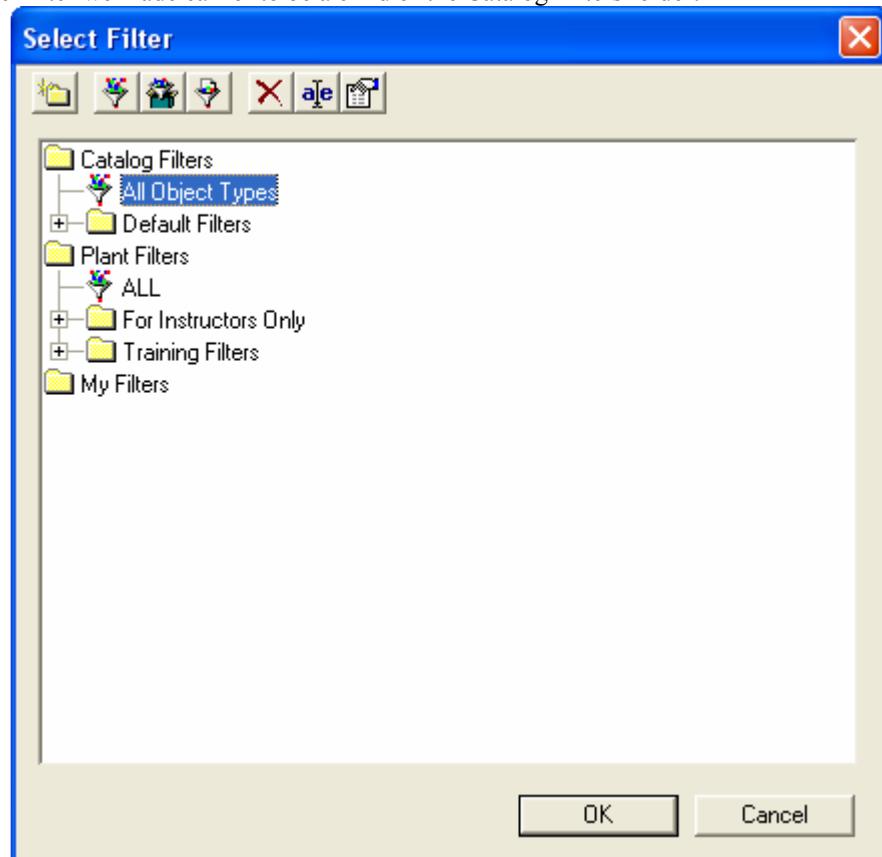
	B	C	D	E	F	G
1	Date:	2/26/2007 10:26:08 AM		Filtername:	All Object Types	
2	User Name:	NAQUADAHFerguson		FilterLocation:	model	
3	Plant DB:	model\{40093e29-0bdd-4370-9a46-f5cf01eceaa5a}		FilterDescription:	#####	
4	Report Name:	Report Template (No Query)				
5						
6						
7						
8				Name	OID	
9						
10				Pump-001	{00004E2E-0000-0000-0000-0D6FAF451D04}	
11				CT-ET-0005	{00013883-0000-0000-0004-0D6FAF451D04}	
12				90 Degree Direction Chang	{00013885-0000-0000-0019-0D6FAF451D04}	
13				Flange-0021	{00013885-0000-0000-011A-0D6FAF451D04}	
14				402-P	{000138AA-0000-0000-011D-0D6FAF451D04}	
15				CSys	{00046CD1-0000-0000-0200-0D6FAF451D04}	
16				DP1	{00004E27-0000-0000-0300-0D6FAF451D04}	
17				Cable Tray Part	{00013885-0000-0000-0305-0D6FAF451D04}	
18				U04-8-P-0001-1C0031	{0001388D-0000-0000-0318-0D6FAF451D04}	
19				Pipe	{0001388C-0000-0000-0412-0D6FAF451D04}	
20				Suction	{00004E23-0000-0000-0500-0D6FAF451D04}	
21				1001-P	{000138AA-0000-0000-050C-0D6FAF451D04}	
22				Flange-0007	{00013885-0000-0000-050D-0D6FAF451D04}	
23				Flange-0029	{00013885-0000-0000-051C-0D6FAF451D04}	
24				401-P	{000138AA-0000-0000-0813-0D6FAF451D04}	
25				Pipe	{0001388C-0000-0000-0819-0D6FAF451D04}	
26				404-P	{000138AA-0000-0000-0A1A-0D6FAF451D04}	
27				Discharge	{00004E23-0000-0000-0C00-0D6FAF451D04}	
28				401-P	{000138AA-0000-0000-0C13-0D6FAF451D04}	
29				Flange-0017	{00013885-0000-0000-0C18-0D6FAF451D04}	
30				1001-P	{000138AA-0000-0000-0E00-0D6FAF451D04}	
31				400-P	{000138AA-0000-0000-0E12-0D6FAF451D04}	
32				404-P	{000138AA-0000-0000-1019-0D6FAF451D04}	
33				404-P	{000138AA-0000-0000-101A-0D6FAF451D04}	
34				Cable Tray Part	{00013885-0000-0000-1104-0D6FAF451D04}	
35				404-P	{000138AA-0000-0000-1219-0D6FAF451D04}	
36				Tee-0002	{00013885-0000-0000-130F-0D6FAF451D04}	
37				Cable Tray Part	{00013885-0000-0000-1404-0D6FAF451D04}	
38				U01-6-P-0001-1C0031	{0001388D-0000-0000-140E-0D6FAF451D04}	
39				90 Degree Direction Chang	{00013885-0000-0000-1412-0D6FAF451D04}	
40				VG3-0004	{00013885-0000-0000-141A-0D6FAF451D04}	
41				Flange-0025	{00013885-0000-0000-141B-0D6FAF451D04}	
42				Pipe	{0001388C-0000-0000-141C-0D6FAF451D04}	
43				U04-10-P-0002-1C0031	{0001388D-0000-0000-1618-0D6FAF451D04}	
44				90 Degree Direction Chang	{00013885-0000-0000-170C-0D6FAF451D04}	

Sheet1 / Ready

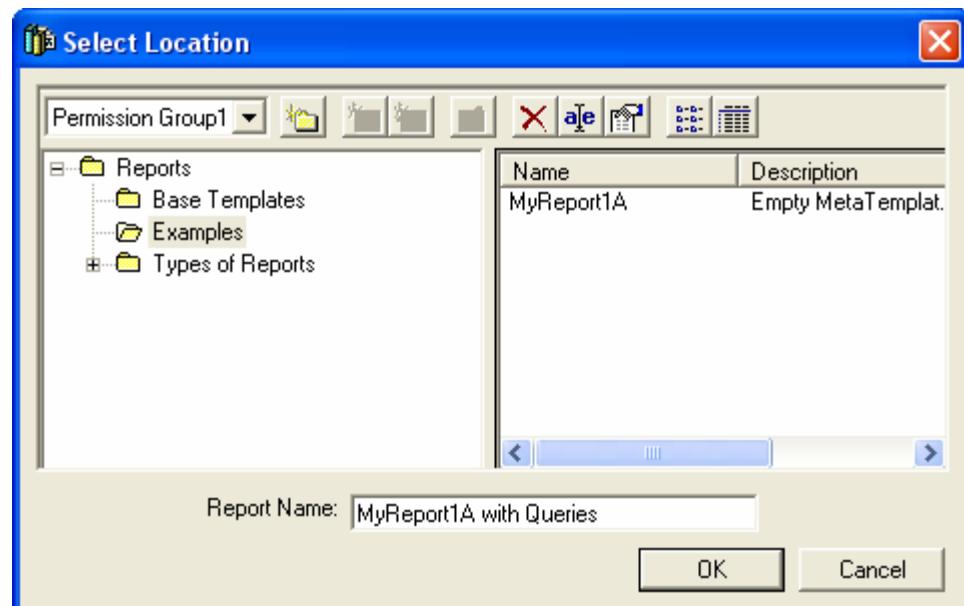
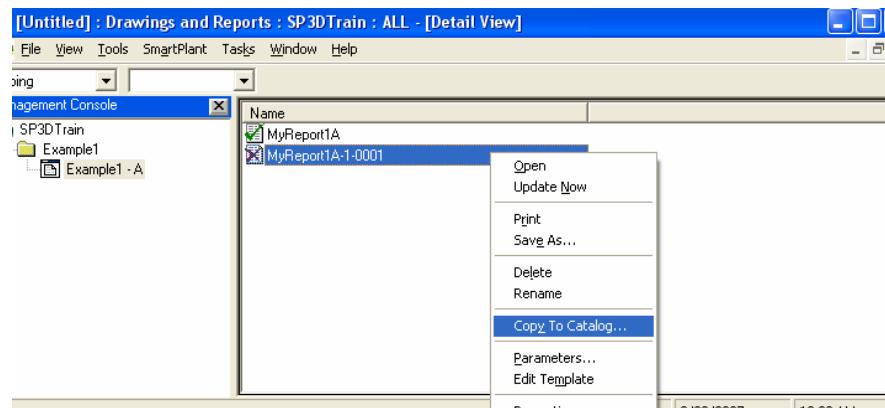
27. What you should be presented with is a list of names of all the Objects in the Plant as well as their OIDs.
28. Remember that at this point, these changes we have made are only specific to this single Report in the Model Database definition. To make these changes available Catalog – wide, we must run the Copy to Catalog command. Do this now.
29. Note that the following warning is displayed:



30. Click OK.
31. Right mouse on the Report and select Edit Template again.
32. On the query Tab, select the More... option from the Filter drop down.
33. Drag and Drop the Filter we made earlier to be a child of the Catalog Filters folder:

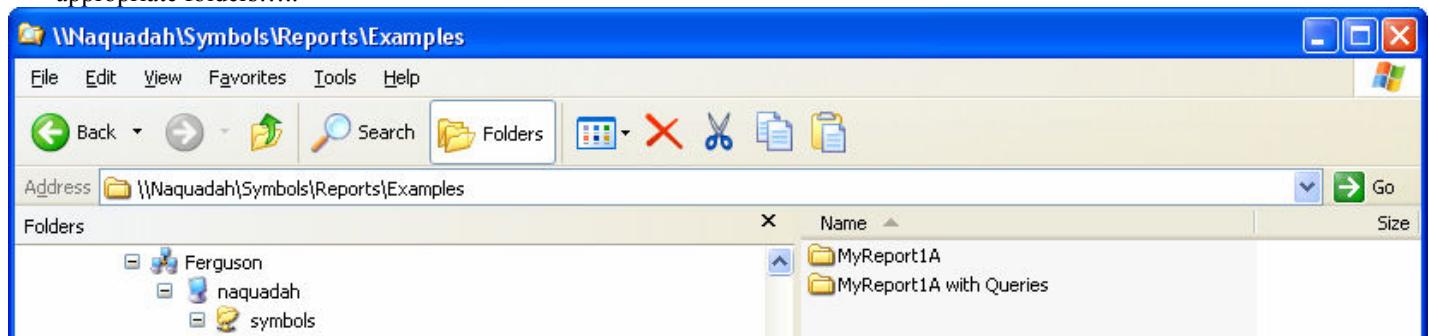


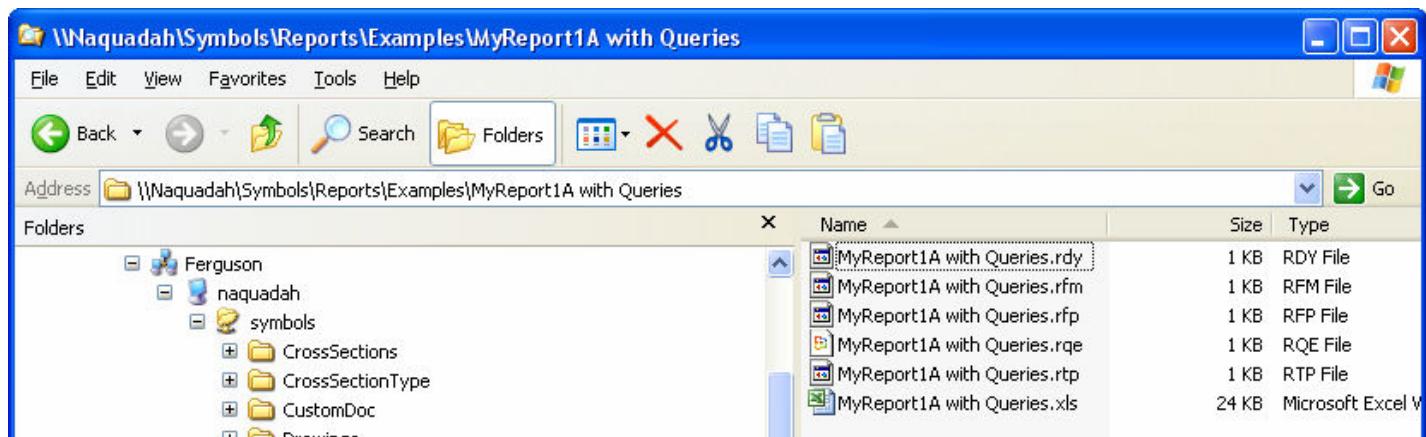
34. Click OK.
35. Save the Report Template.
36. Copy to Catalog again:



37. Provide it a new name so that a comparison can be performed later.

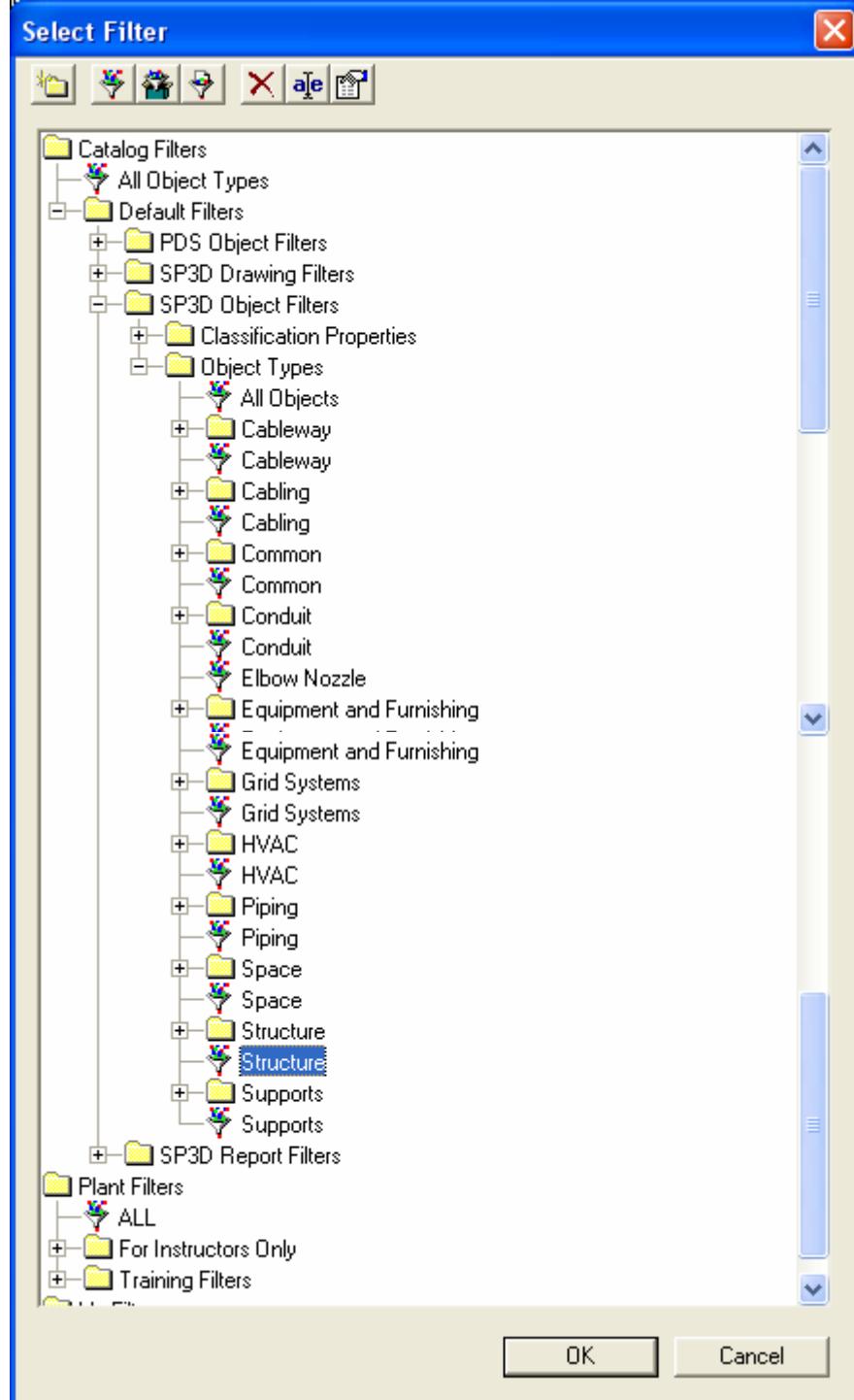
38. Notice that if you go to the Common environment and select Tools → Run Reports... that the additional Report has been added to the Catalog, and that the original still remains. Also, take a look at the Symbol share and you will now find the appropriate folders.....





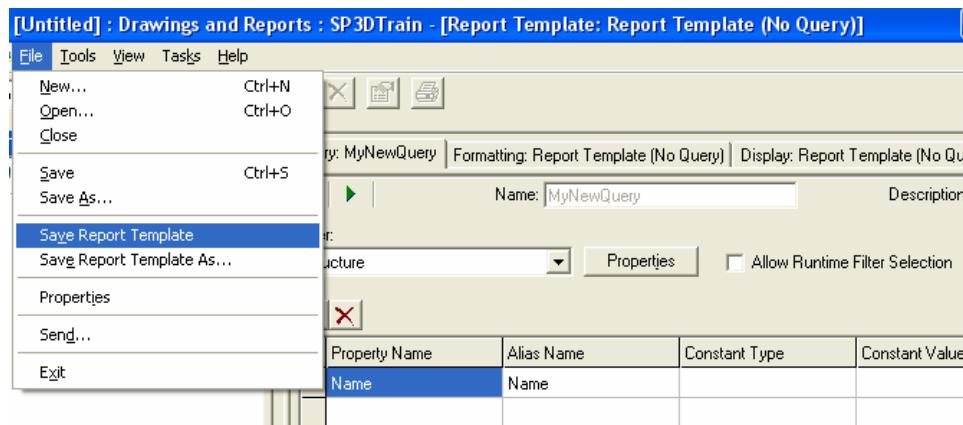
39. Note that if you compare the two directories (MyReport1A and MyReport1A with Queries) that you now have an additional file in the “...with Queries” directory. That file is an rqe and contains query information that we have defined through the GUI.
40. Switch to the Common Task and use the Tools→Run Report.. Functionality to execute this newly created report.
41. Let us return to the Drawings and Report environment and change this report to be only Structural Objects.
42. Right mouse click (again) on the “MyReport1A-0001” from within the Drawings and Reports environment and select Edit Template.

43. Let us now change the Filter that the Query is based on. Let us use the Catalog Filter "Structure"



Note: Path in above screenshot is /Catalog Filters/Default Filters/SP3D Object Filters/Object Types/

44. Now go File→Save Report Template

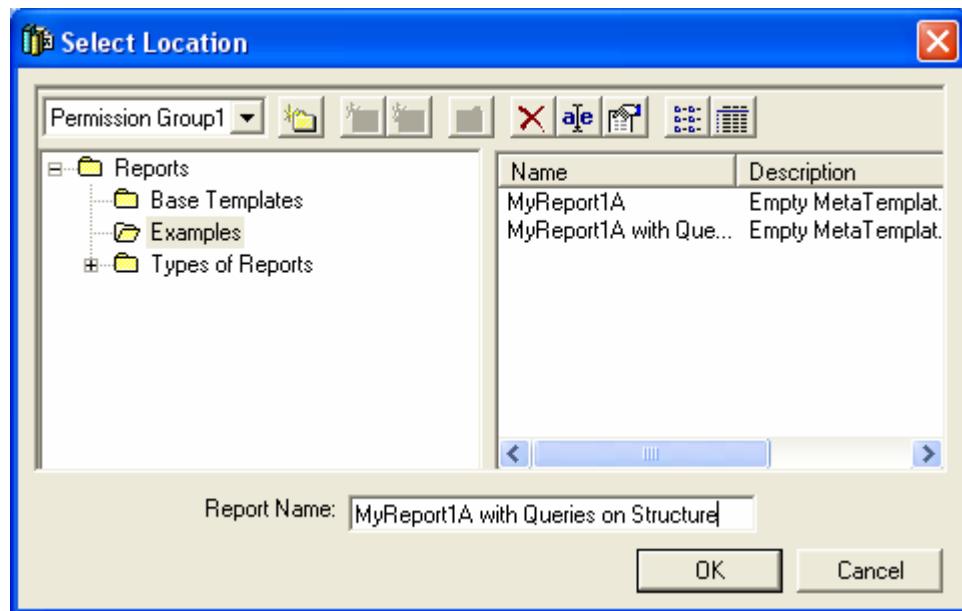


45. Now update the Report "Example 1-A" in the Drawings and Report environment and open the new reporting results.

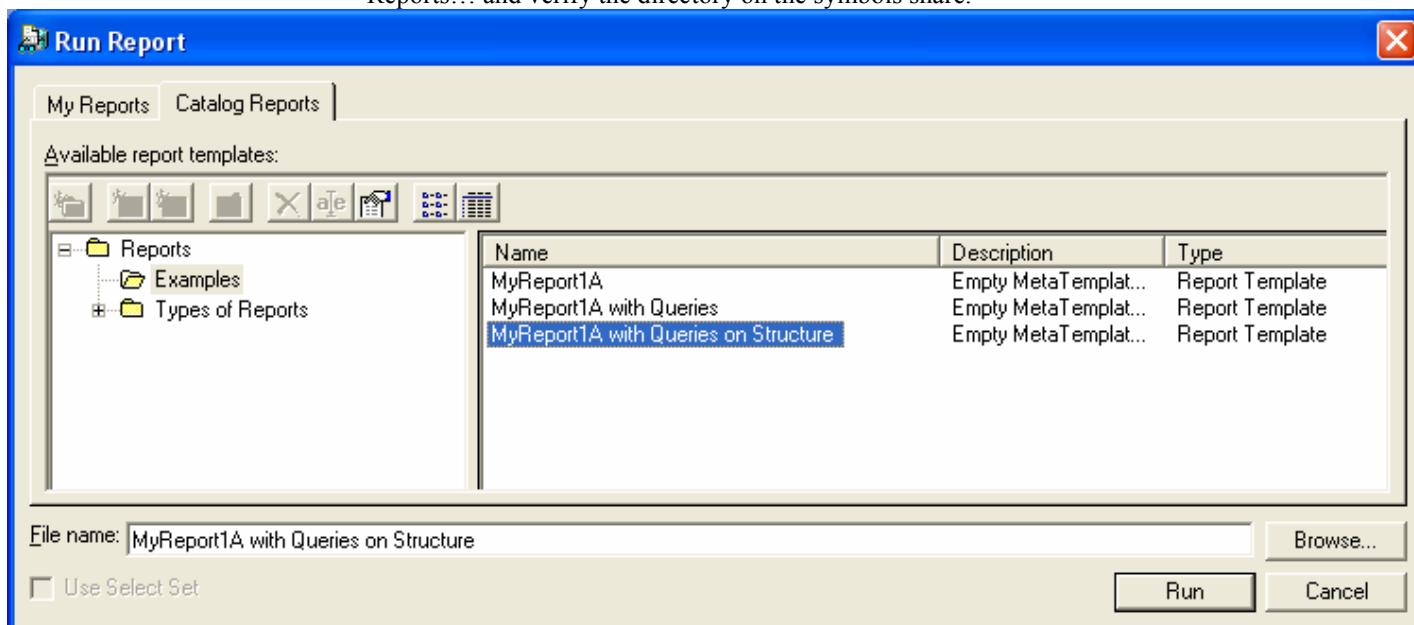
46. Observe that the new report output contains only structural objects.

B	C	D	E	F	G
Date:	2/26/2007 10:37:08 AM	Filtername:	Structure		
User Name:	NAQUADAHFerguson	FilterLocation:	Catalog		
Plant DB:	model(40093e29-0bdd-4370-9a46-f5cf01eceaa5a)	FilterDescription:	Object Type IN (Structure)		
Report Name:	Report Template (No Query)				
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					
21					
22					
23					
24					
25					
26					
27					

47. Once you are satisfied with the result we will Copy to Catalog the report once again, and name it “MyReport1A with Queries on Structure”

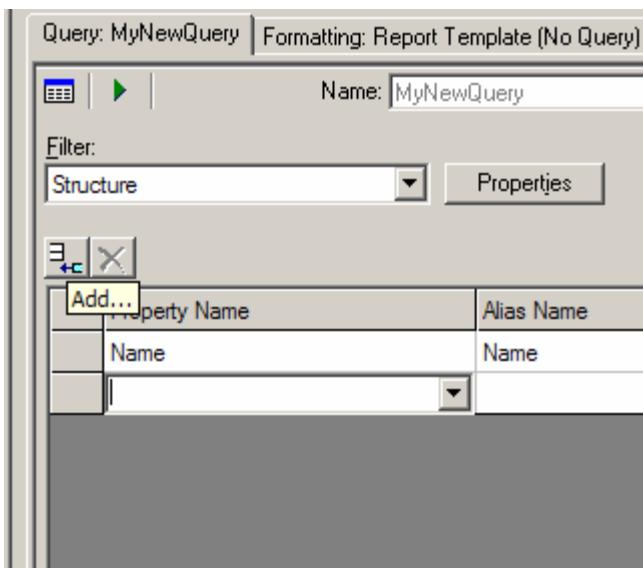


48. Again, check to see that you can locate it in the catalog hierarchy (as we did before by switching to Common, Tools→ Run Reports... and verify the directory on the symbols share).

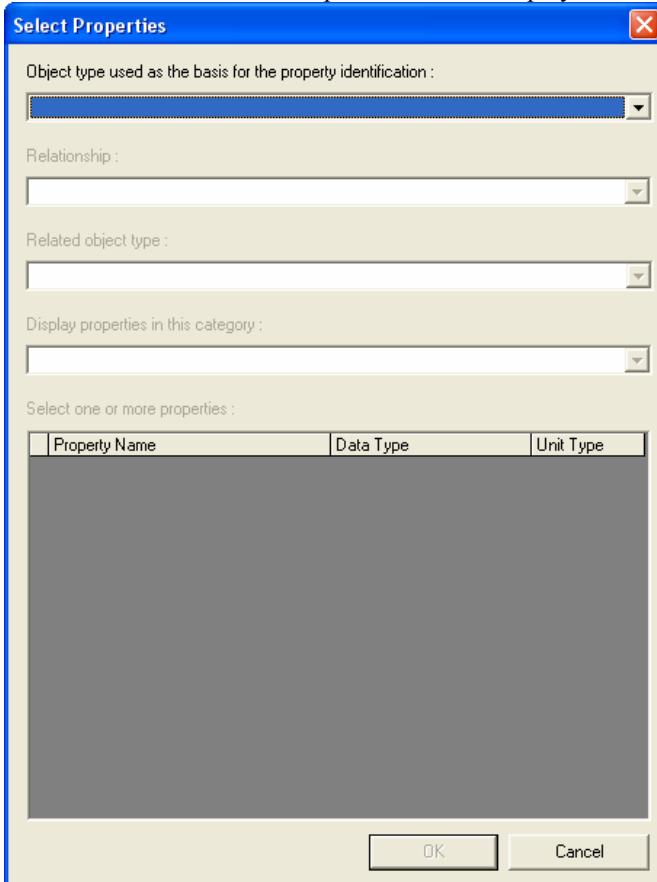


49. You can run this report from Tools→Run Report since it has a query.

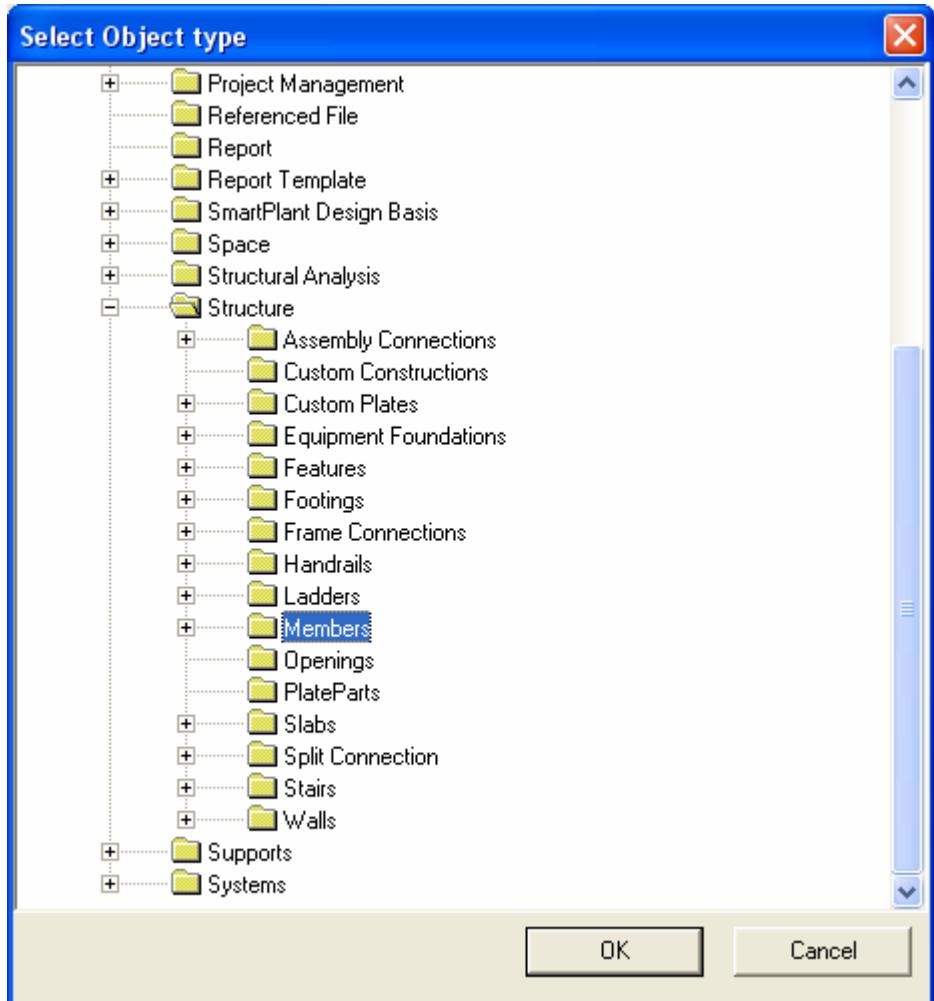
Adding additional property queries...



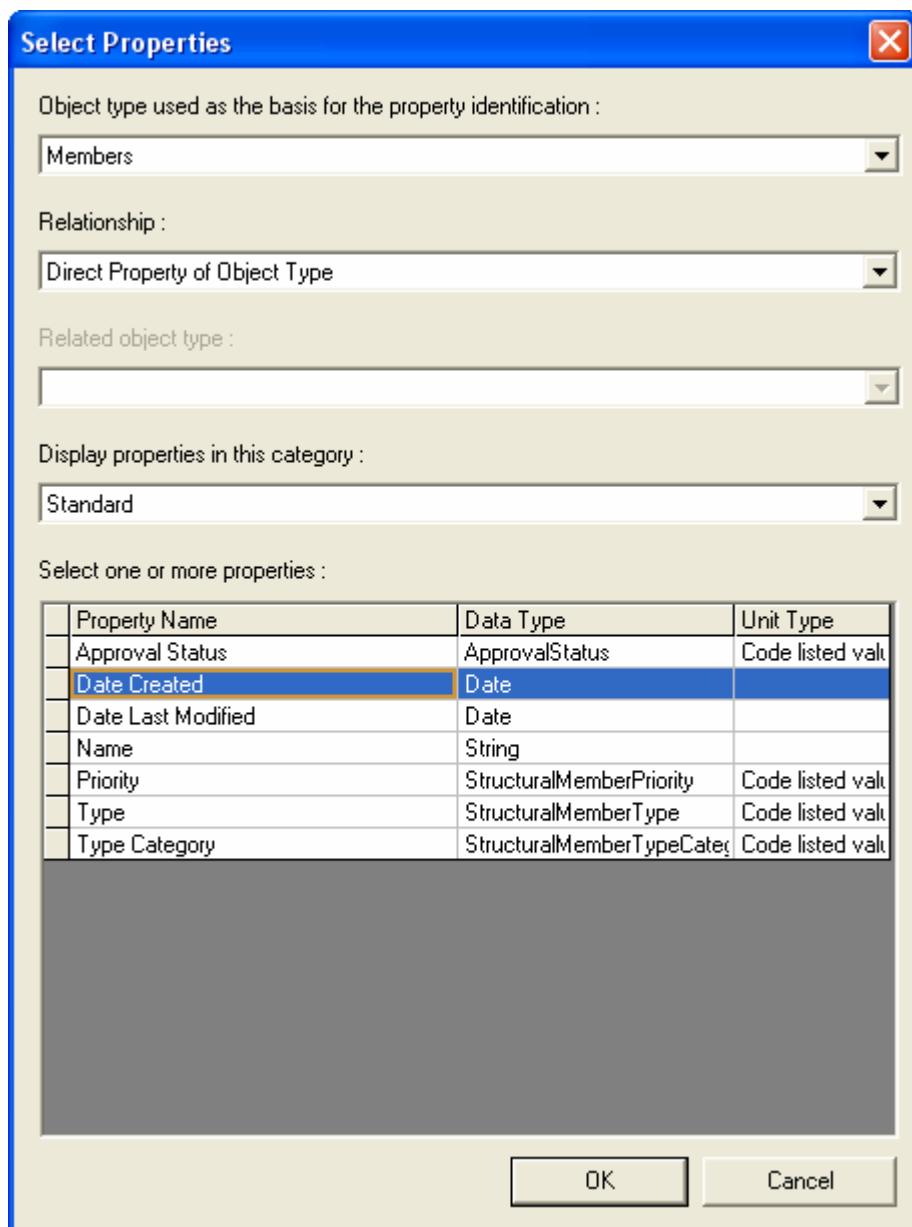
1. While in Edit Template mode on the Example 1A-0001 report, select an empty row and click the “Add...” button”
2. Note that the “Select Properties” form is displayed.



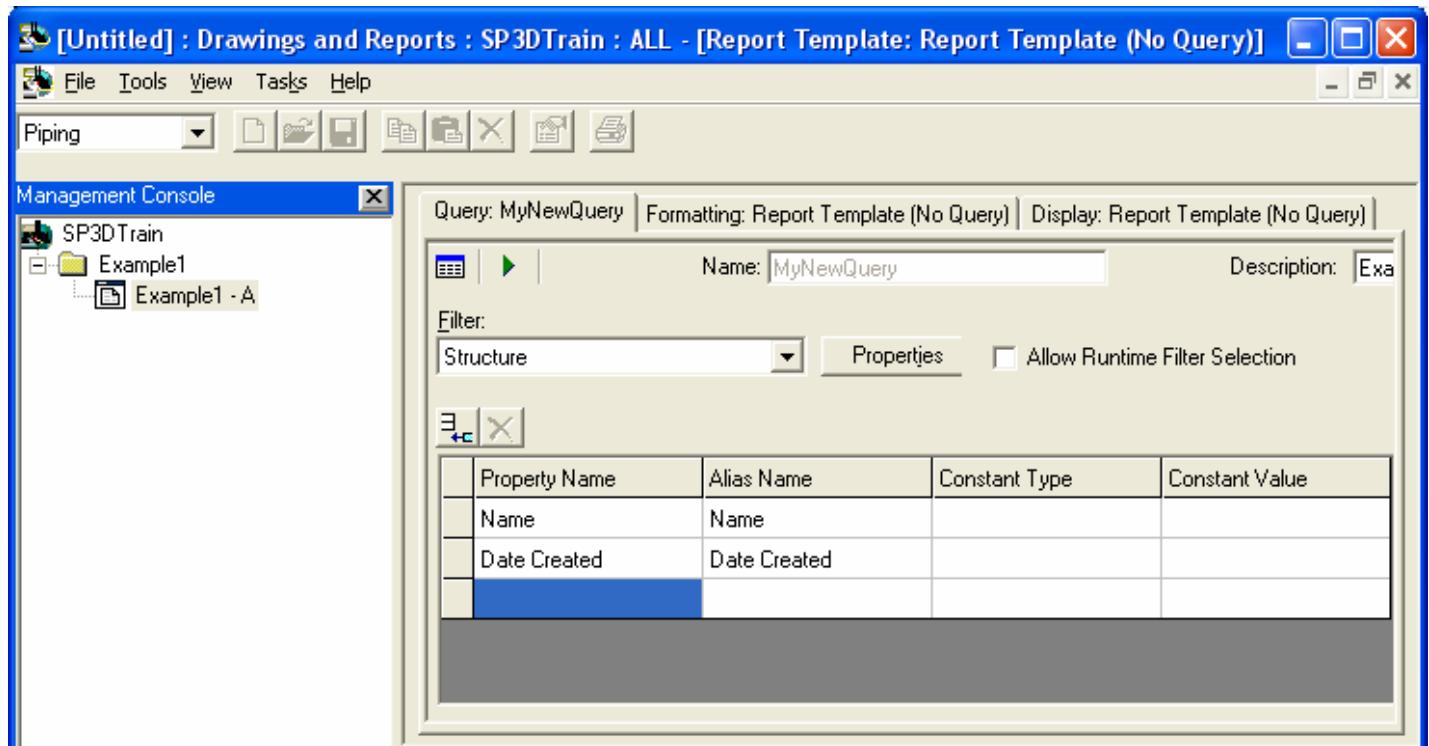
3. Users familiar with the Filter creation workflow will be familiar with this form as it is the same form used to define filters based on object properties. Likewise here, we will use it to identify properties that we wish to display on the Report.
4. Let us complete the form as follows, pick the first drop down and select the following item from the tree:



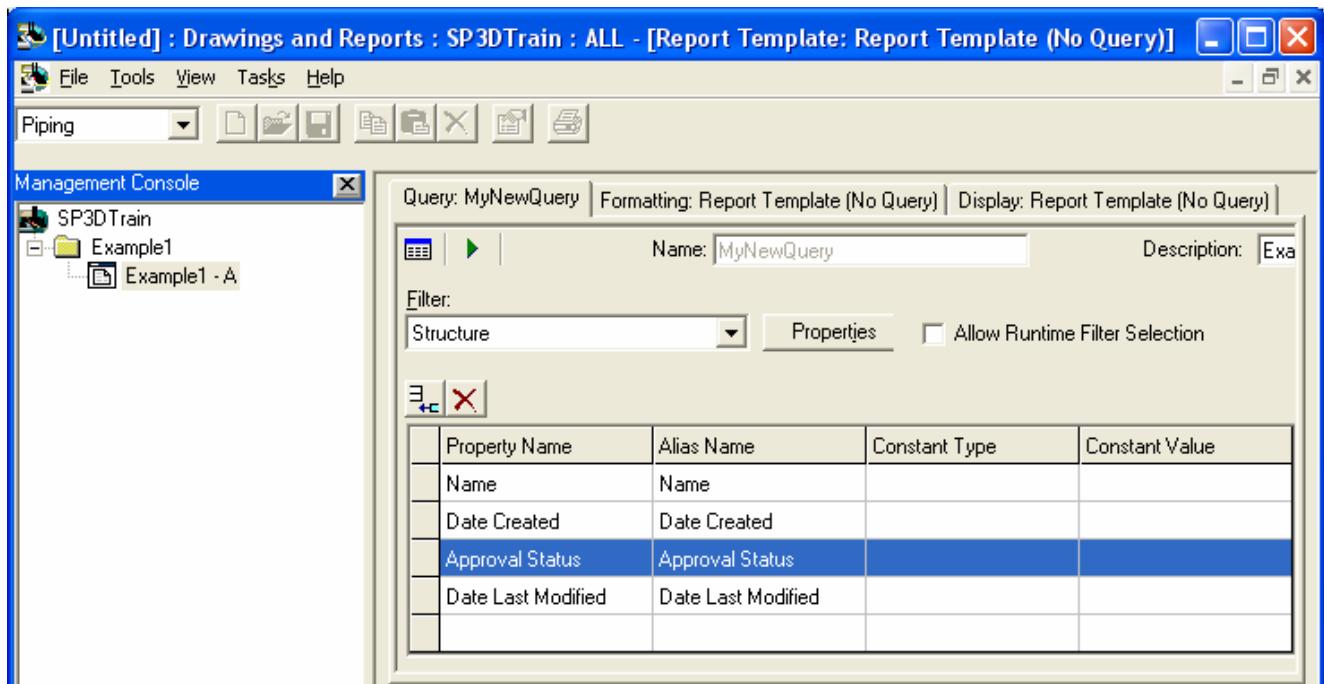
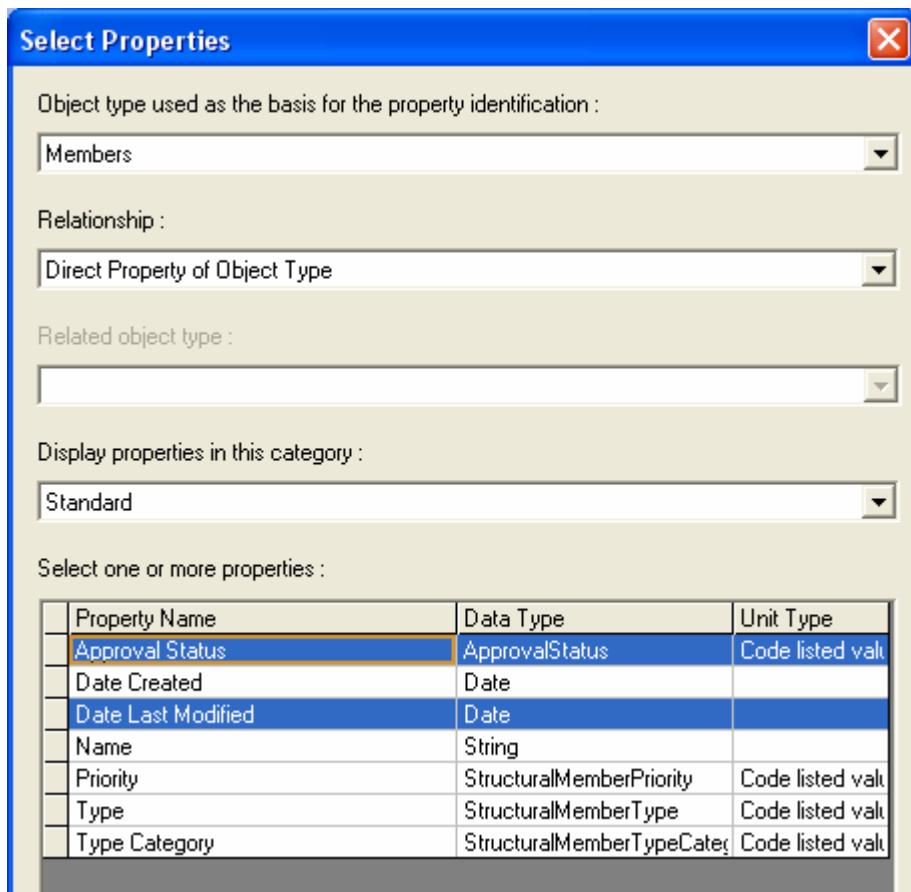
5. Click OK, and complete the form as follows:



6. Select the Date Created Property, and click OK.



7. Observe that Date Created is now listed.
8. Repeat the same workflow adding in “Approval Status” and “Date Last Modified”



- Place them as follows on the Excel Report (recall from earlier how this is done)

	Name	OID	Date Created	Last Modified	Status
#MyNewQuery::Name#	#MyNewQuery::OID#	#MyNewQuery::Date Created#	#MyNewQuery::Date Last Modified#	#MyNewQuery::Approval Status_LongString#	

10. Note that above we are placing in order from left to right:

#MyNewQuery::Date Created#	#MyNewQuery::Date Last Modified#	#MyNewQuery::Approval Status_LongString#
-----------------------------------	---	---

11. Save the XLS.
12. Save the Template.
13. Update the Report.
14. Verify results.

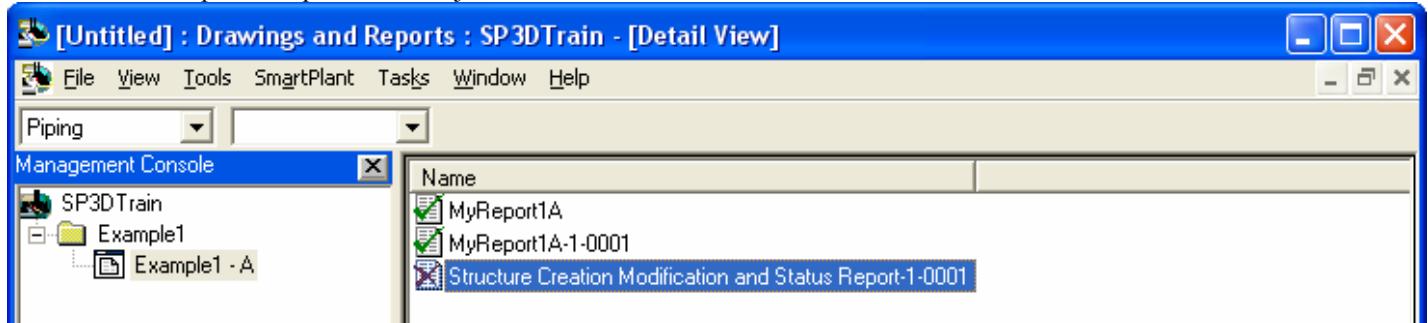
The screenshot shows an Excel spreadsheet with the following data:

	B	C	D	E	F	G	H	I
1	Date: 2/26/2007 10:54:30 AM		Filtername: Structure					
2	User Name: NAQUADAHFerguson		FilterLocation: Catalog					
3	Plant DB: model(40093e29-0bdd-4370-9a46-15cf01eceaa5a)		FilterDescription: Object Type IN (Structure)					
4	Report Name: Report Template (No Query)							
5								
6								
7								
8								
9								
10								
11								
12								
13								
14								
15								
16								
17								
18								
19								
20								
21								
22								
23								
24								
25								
26								
27								
28								
29								
30								
31								
32								
33								
34								
35								
36								
37								
38								
39								
40								
41								
42								
43								
44								

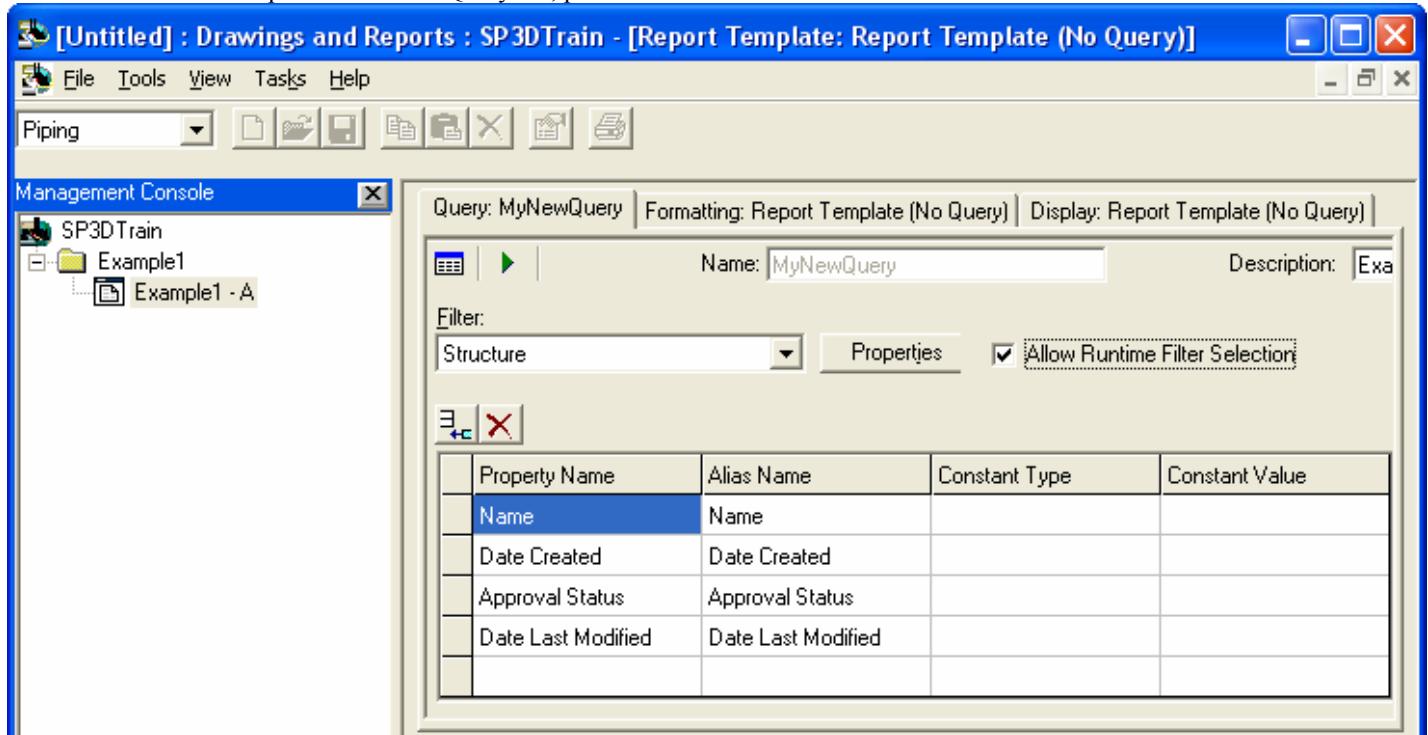
15. Save to Catalog, name it “Structure Creation Modification and Status Report” under the same Example folder.
16. Verify it is available in the Report hierarchy.
17. Verify the output file on the symbols share.

Allow Runtime Filter Selection (ARFS) - 1st Example

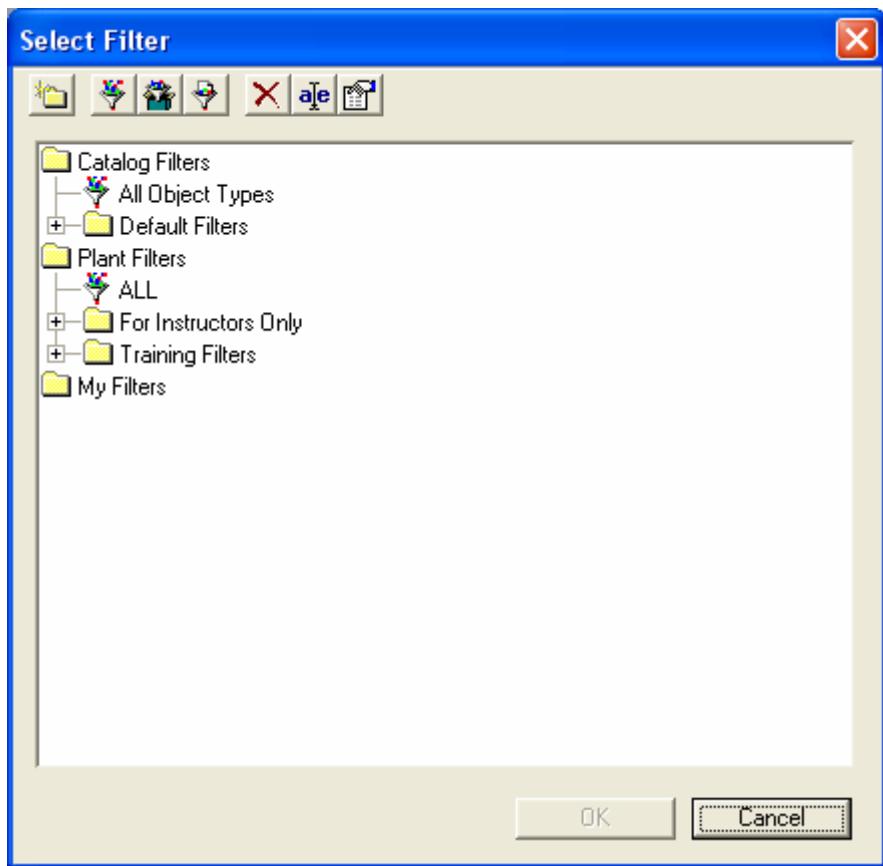
1. Create a new report in the drawings and report hierarchy (like we did before) based on the “Structure Creation Modification and Status Report” template that was just made:



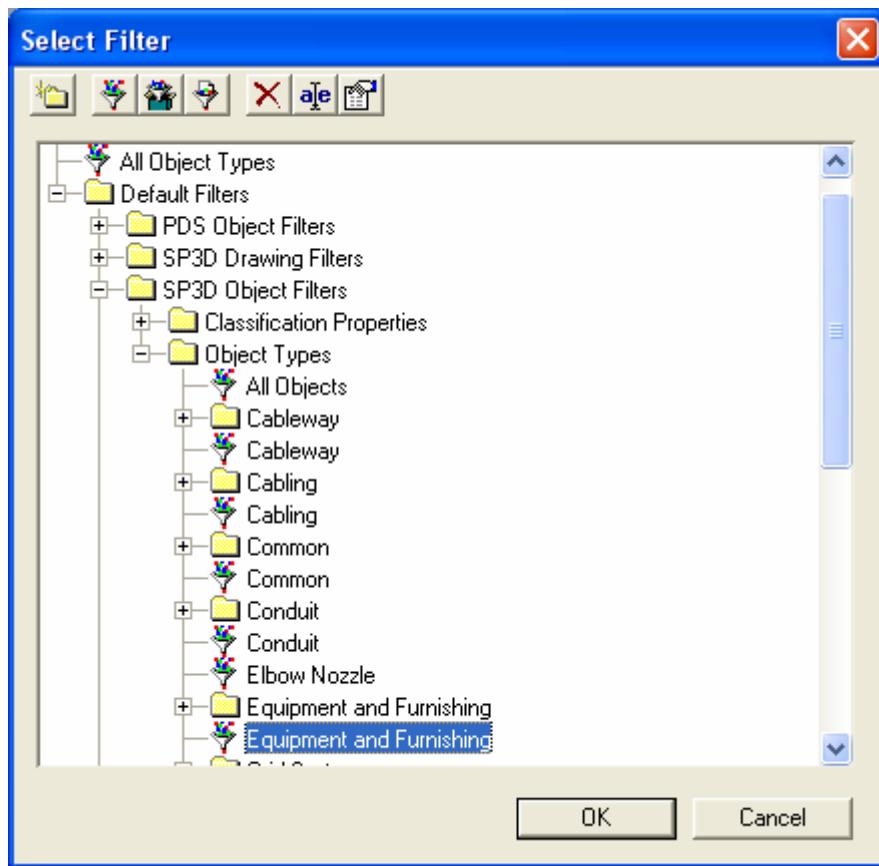
2. Return to Edit Template and on the Query tab, put a check in the box “Allow Runtime Filter Selection”



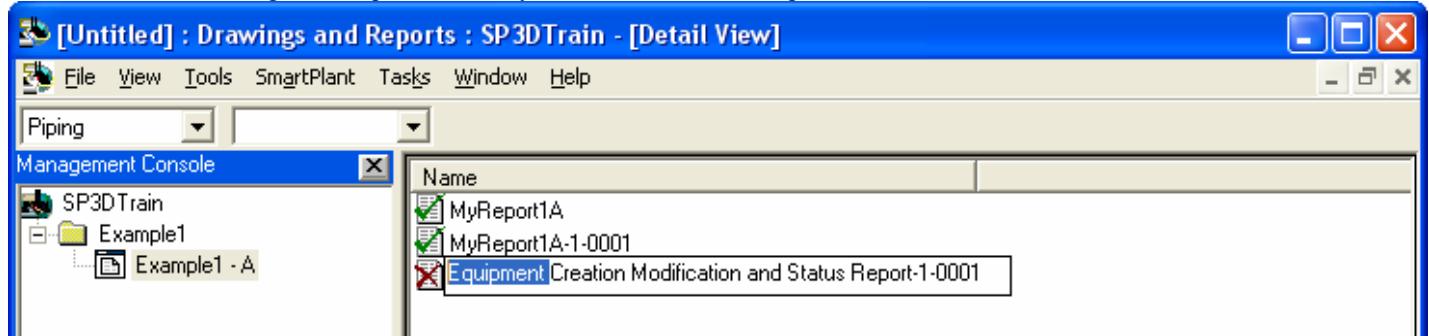
3. Save the Template.
4. Note that the following dialog will be presented and you will be asked to select a filter for this specific occurrence in the D&R hierarchy:



5. Pick the filter “Equipment and Furnishing”



6. Click OK
7. Click in the Drawings and Reports hierarchy, and then rename the report:

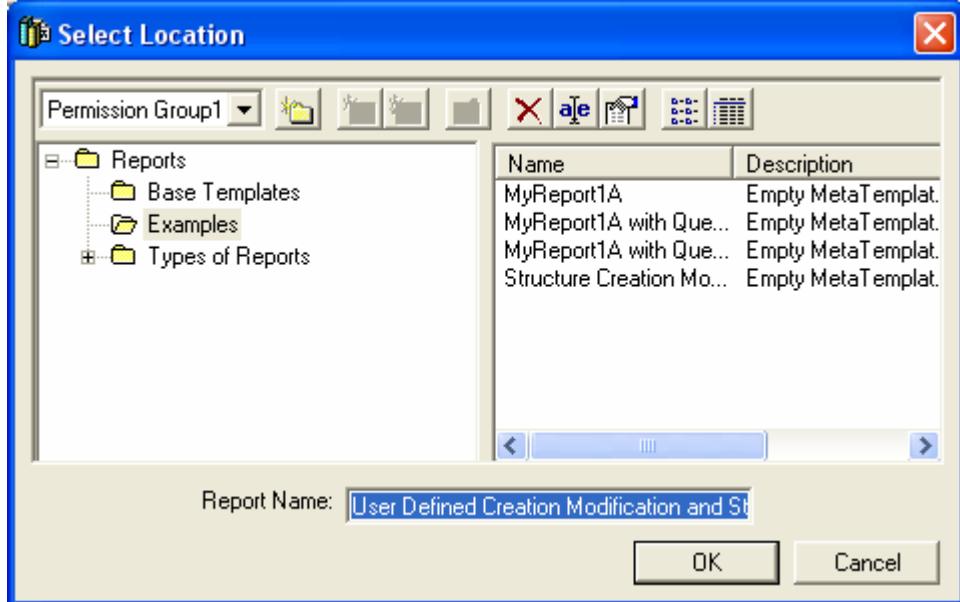


8. Update the Report.
9. Review the results:

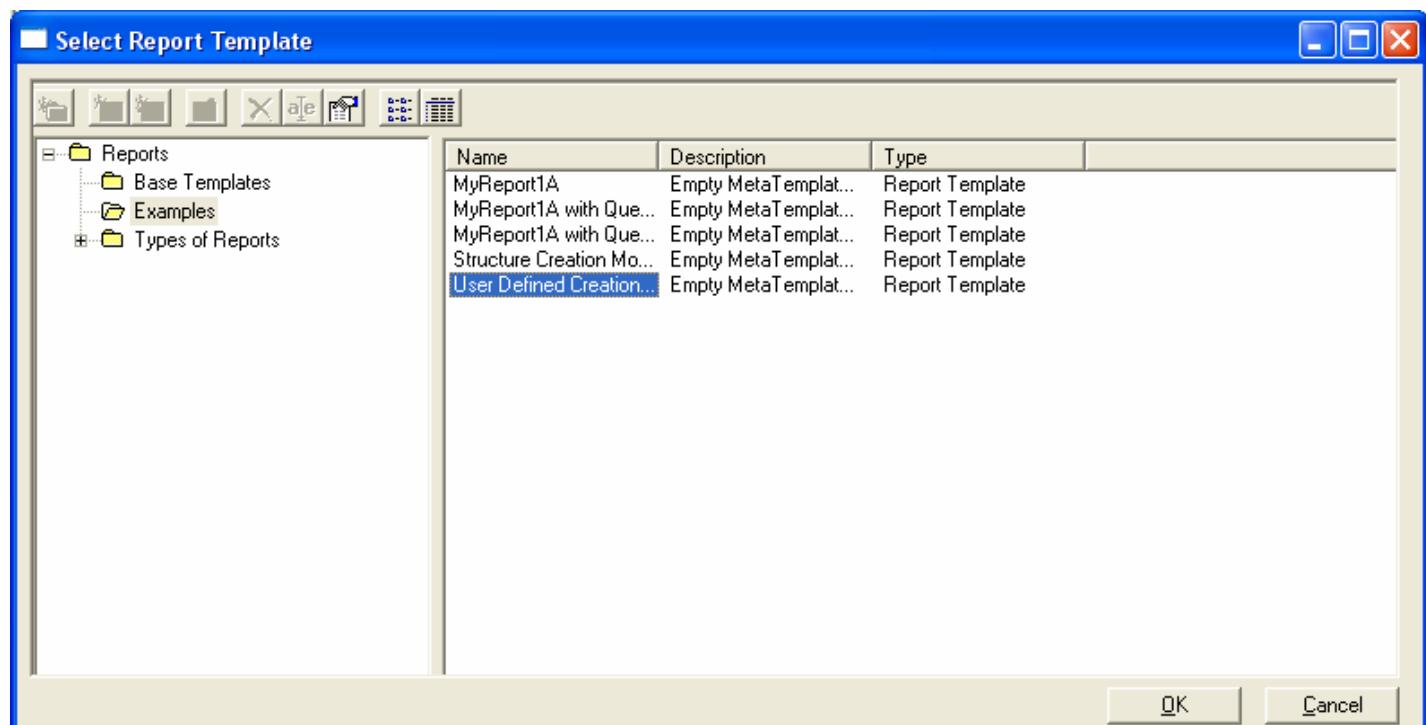
Microsoft Excel - Equipment Creation Modification and Status Report-1.0001.xls [Read-Only]

	B	C	D	E	F	G	H	I
1	Date:	2/26/2007 11:02:59 AM						
2	User Name:	NAQUADAHFerguson	Filtername:	Equipment and Furnishing				
3	Plant DB:	model40093e29-0bdd-4370-9a46-15cf01ece5a)	FilterLocation:	Catalog				
4	Report Name:	Report Template (No Query)	FilterDescription:	Object Type IN (Equipment&Furnishing)				
5								
6								
7								
8								
9								
10								
11								
12								
13								
14								
15								
16								
17								
18								
19								
20								
21								
22								
23								
24								
25								

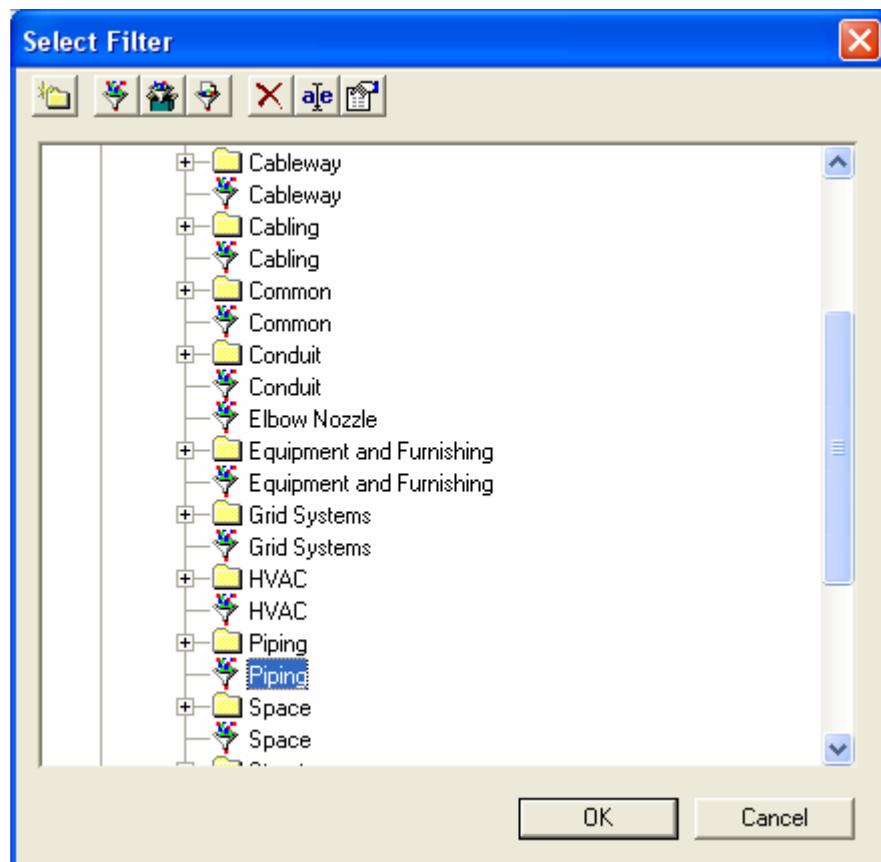
10. Note the results and the filter utilized.
11. Copy the report to the Catalog, giving it the name “User Defined Creation Modification and Status Report”



12. Create a new report based on this newly saved template:



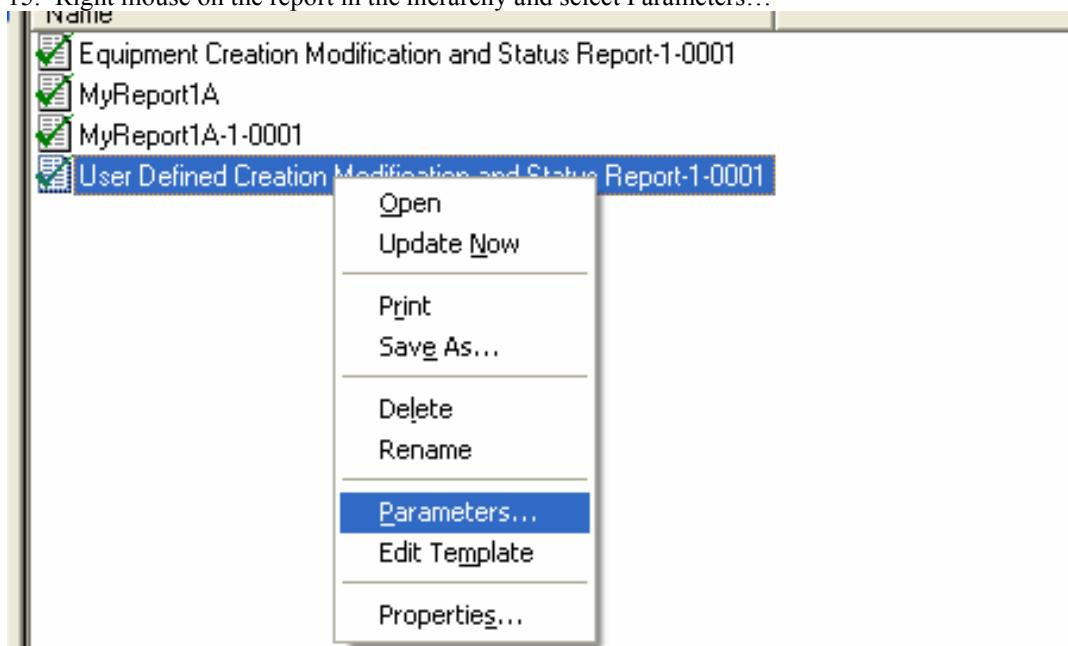
13. When prompted for filter, select the Piping Filter:



14. Update this report and observe the results:

Microsoft Excel - User Defined Creation Modification and Status Report-1-0001.xls [Read-Only]					
	B	C	D	E	F
1	Date:	2/26/2007 11:16:30 AM	Filtername:	Piping	
2	User Name:	NAQUADAHFerguson	FilterLocation:	Catalog	
3	Plant DB:	model40093e29-0b4d-4370-9a46-45cf01ece5a)	FilterDescription:	Object Type IN (Systems\PipelineSystems) AND (Piping)	
4	Report Name:	Report Template (No Query)			
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					
21					
22					
23					
24					
25					
26					
27					
28					
29					
30					
31					
32					
33					
34					
35					
36					
37					
38					
39					
40					
41					
42					
43					
44					

15. Right mouse on the report in the hierarchy and select Parameters...



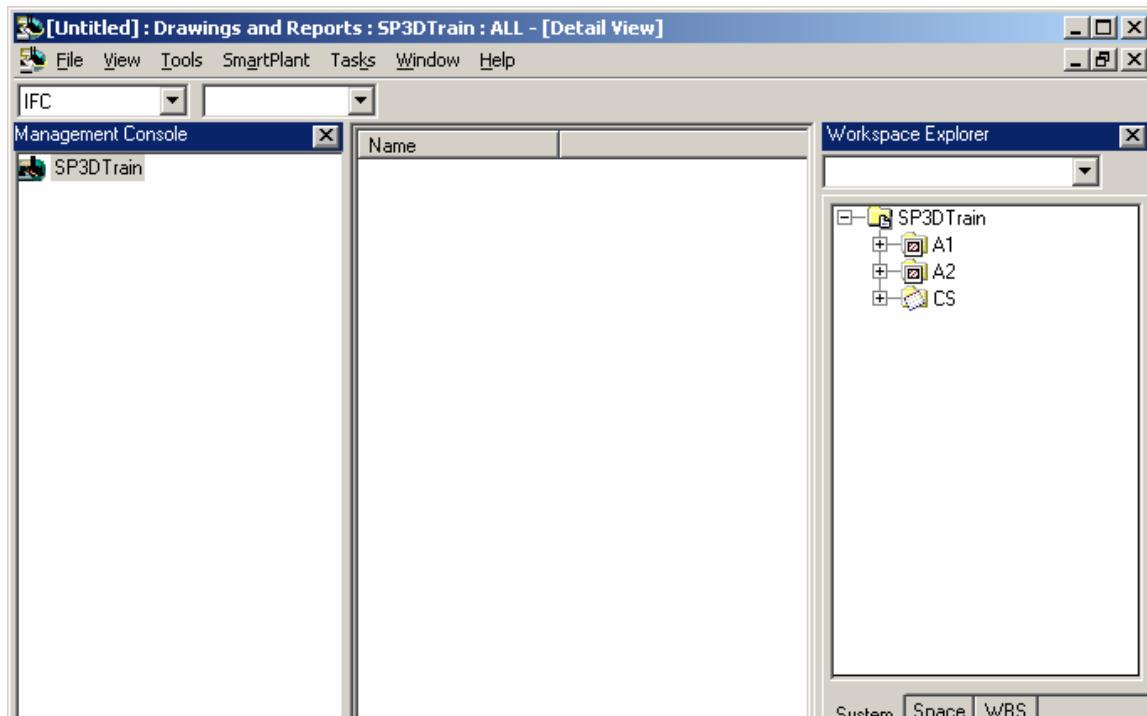
16. Because the Filter is considered a parameter (or variable) you will be prompted to pick the filter again, this time, select Cableway:

17. Then update the report again.

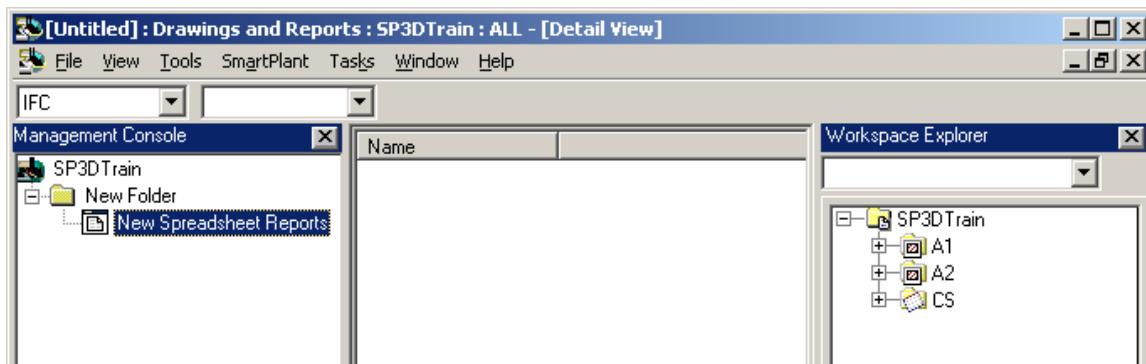
18. Return to the Common Task and use Tools → Run Report command to see how the behavior differs now that the option for the query has been set.

Allow Runtime Filter Selection (ARFS) - 2nd Example

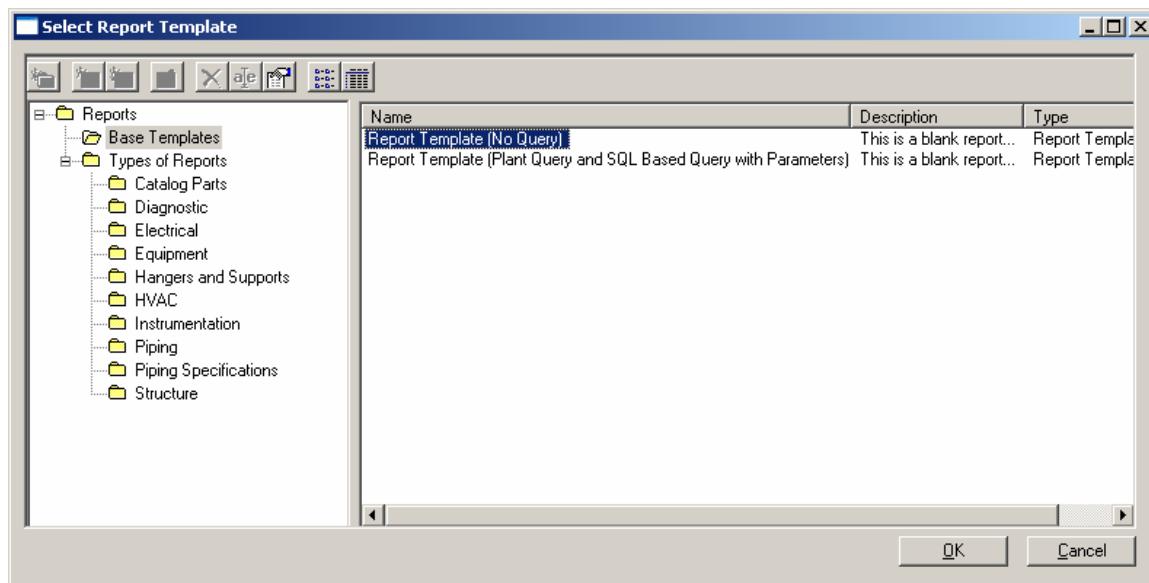
1. Start SP3D and connect to the Training Plant using the “All” Filter.
2. Enter the Drawings and Reports task.



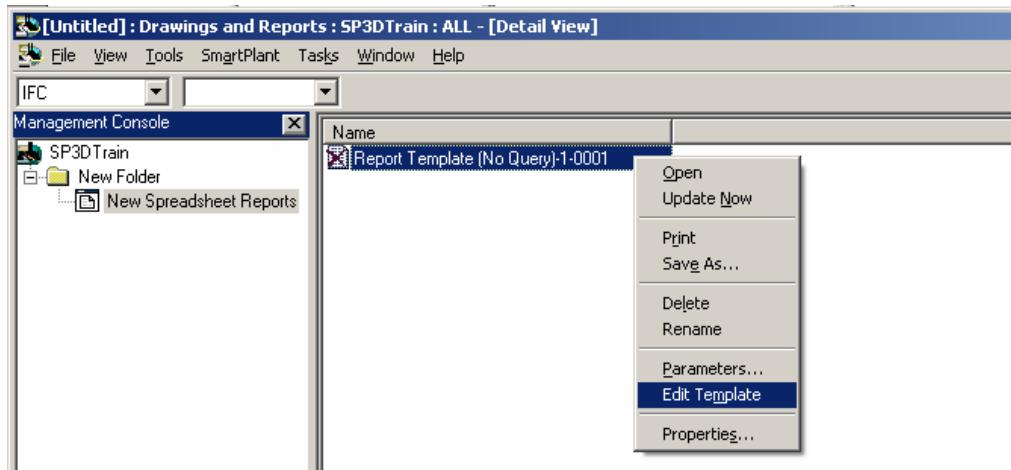
3. Create the following hierarchy by right mouse clicking on the SP3DTrain root node in the Management Console tree, and using the New... command.



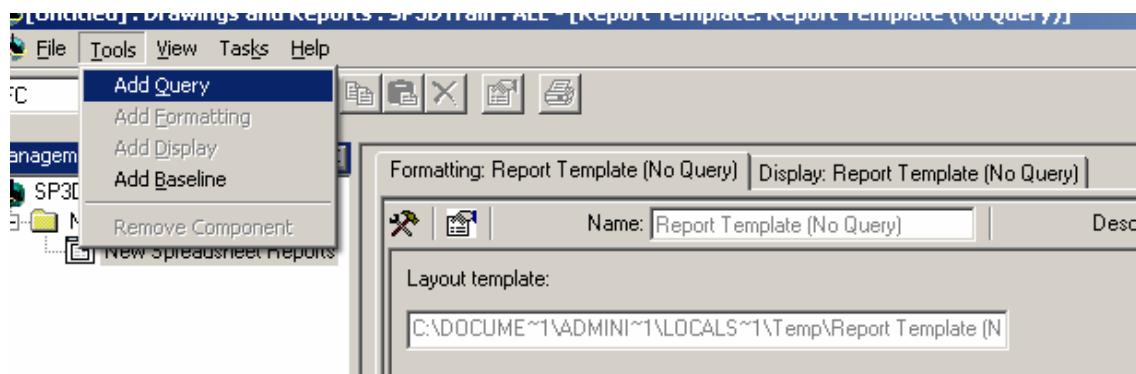
4. Right mouse on the component "New Spreadsheet Reports" and select "Create Report..."
 5. Select the base template exampled above and select OK.



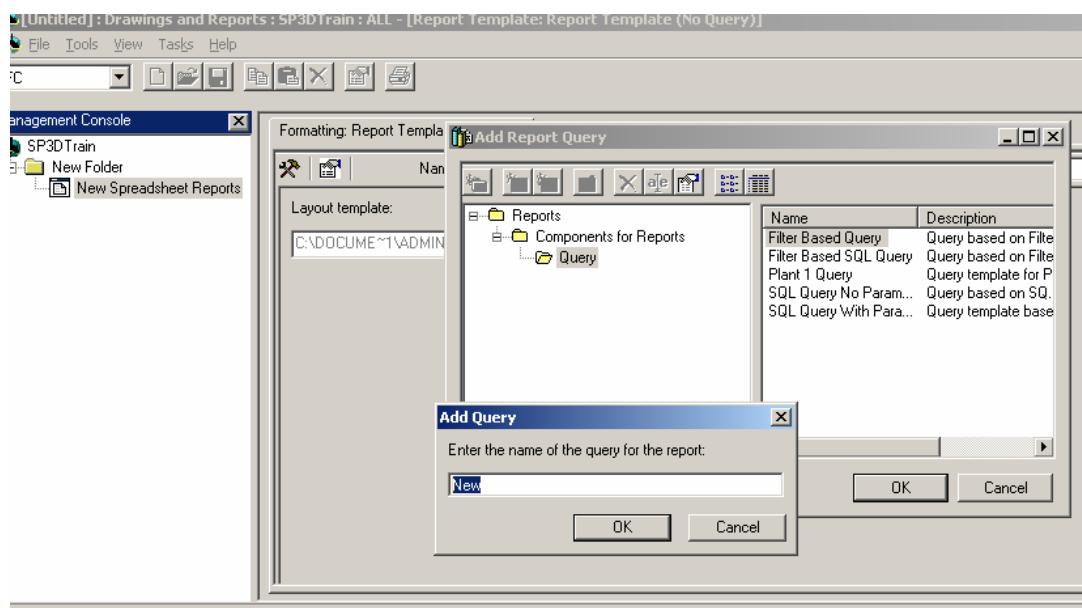
6. Right mouse on the report in the pane as shown below, and select Edit Template.



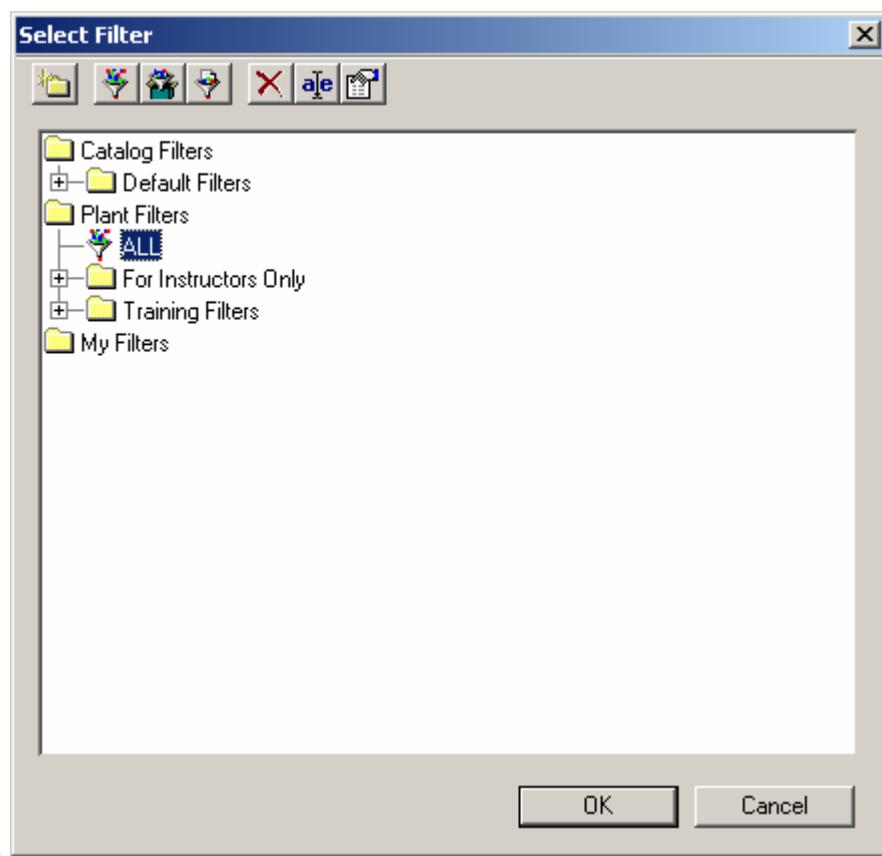
7. Once in the template editor, use the Tools→ Add Query command to add a new query to your report template.



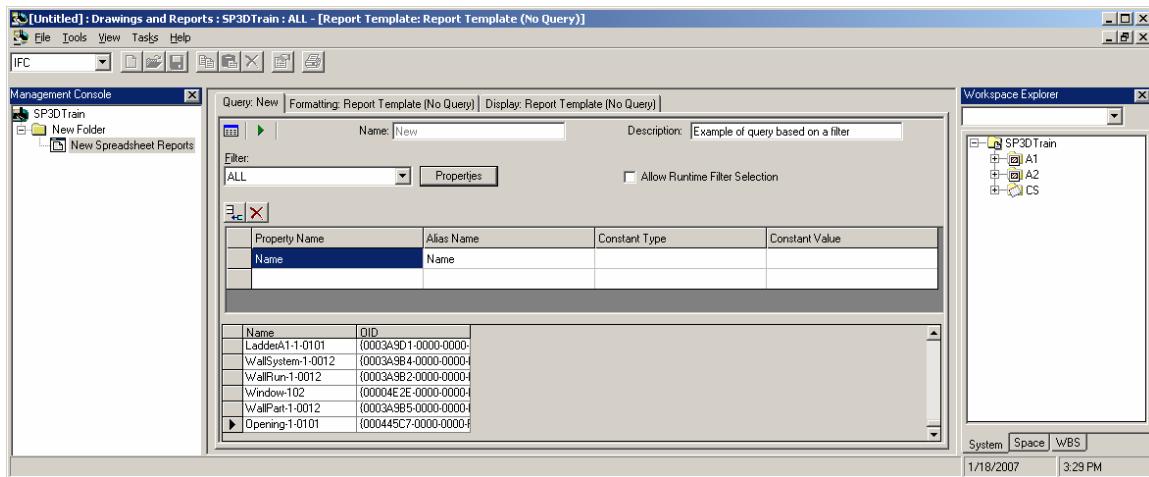
8. Select the Filter Based Query as shown below and click OK.
9. Then provide it a name “New” by click on the OK button of the follow-up form.



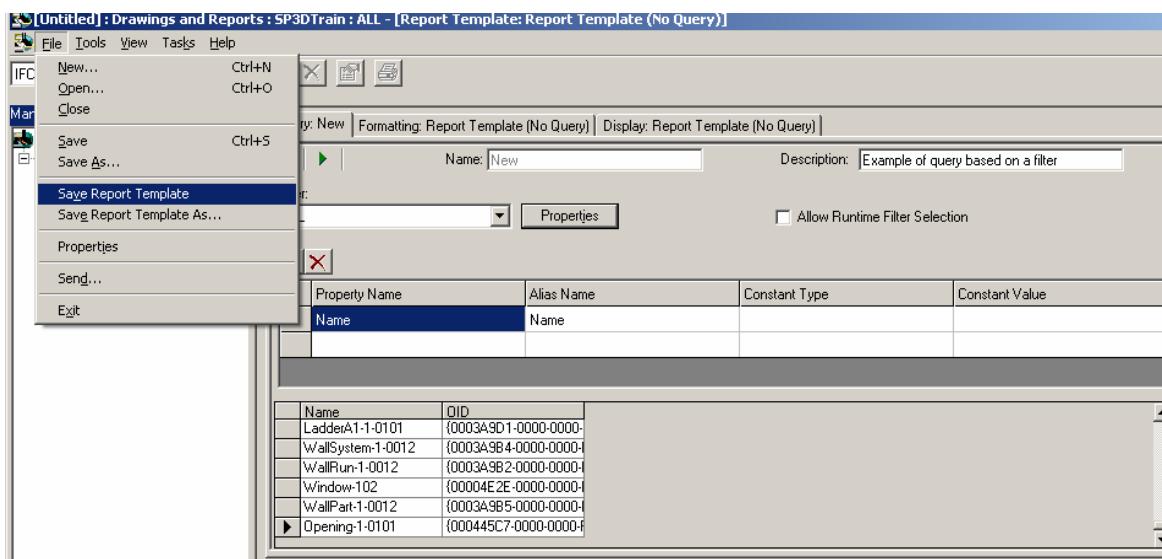
10. Because it is a filter based query you will need to select a filter, please use the All filter, you will see later why this may not be the best choice.



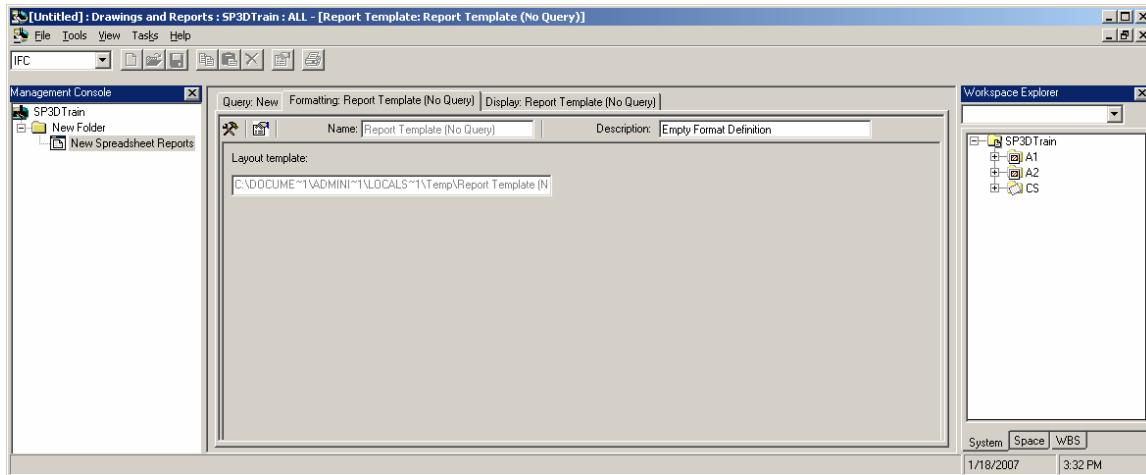
12. Use the green ‘play’ button (▶) to test the query. It should return you the name of the objects in the database.



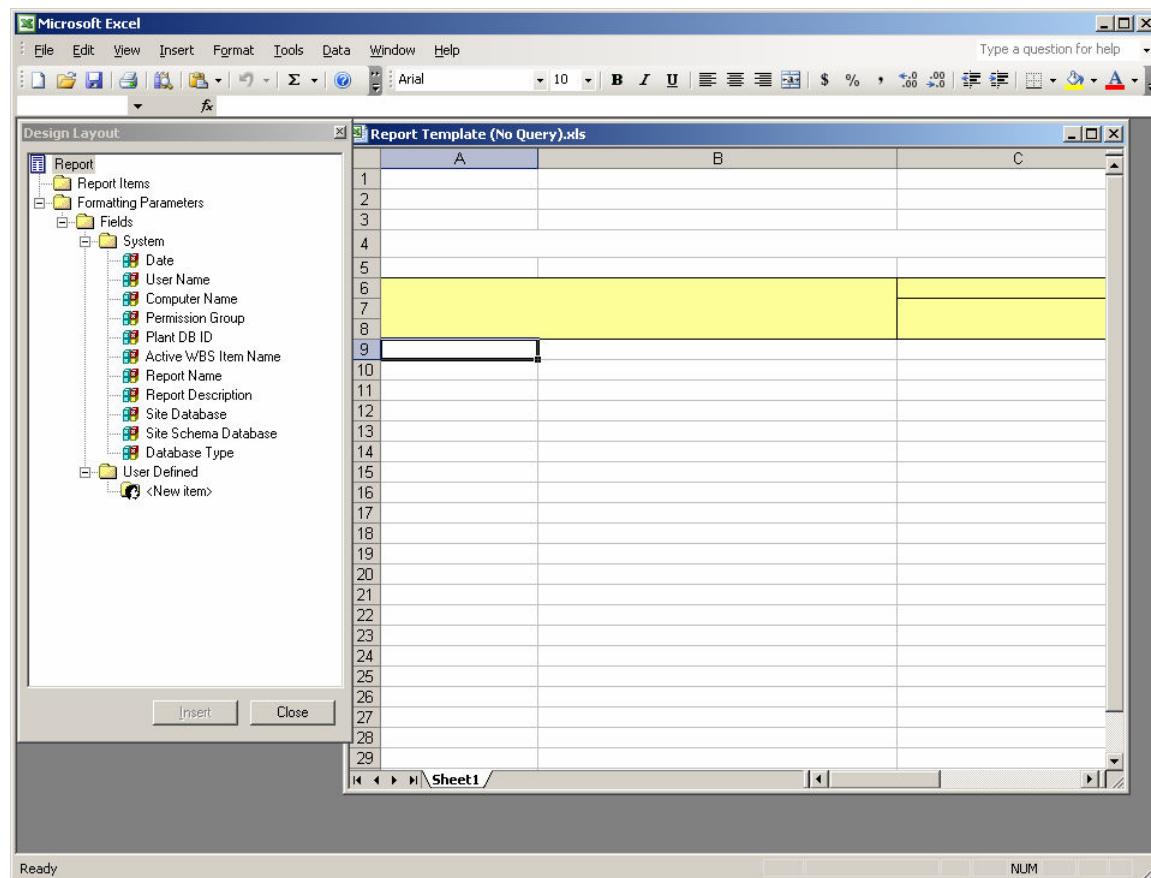
13. Save the report Template:



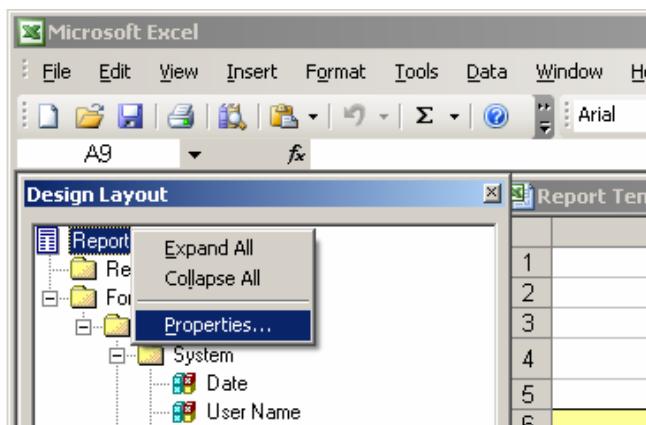
14. Switch to the Formatting tab, and select the Layout editor button ()



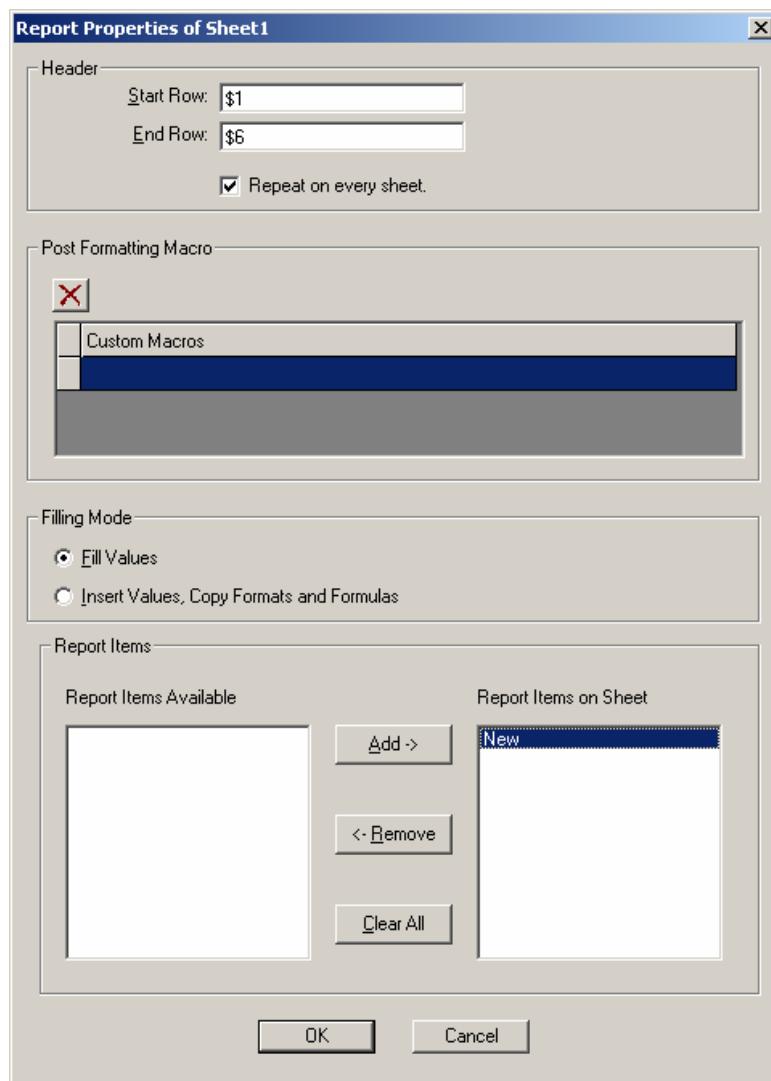
15. Observe that MS Excel starts, and that you are presented with an empty template. If you recall from prior versions of the software, you would expect to see the Query “New” as a branch in the Design Layout tree. However, it is missing.



16. Right mouse on the root of the Design Layout tree and select Properties:

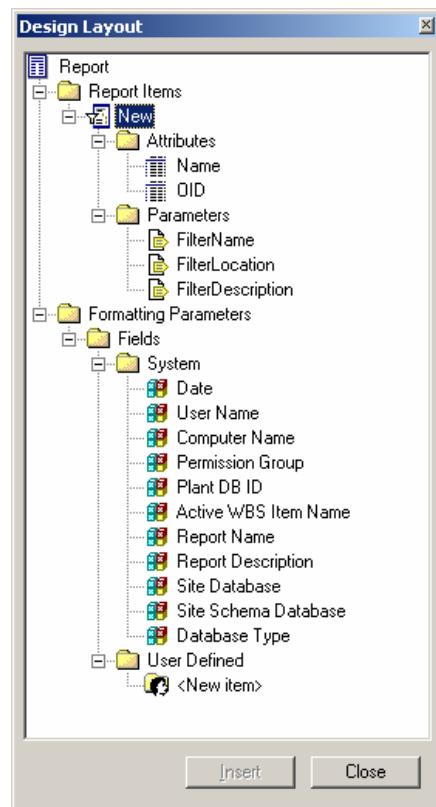


17. You will need to add the New query into the right hand column as depicted below (more on the uses of this in Lab2):



18. Click OK.

Note that you now have the Query branch you were expecting:

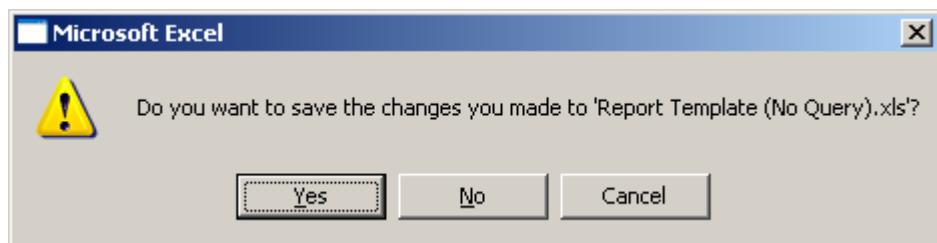


19. Drag and drop the Name and OID fields from the Design Layout window over onto the Excel Sheet as depicted below:

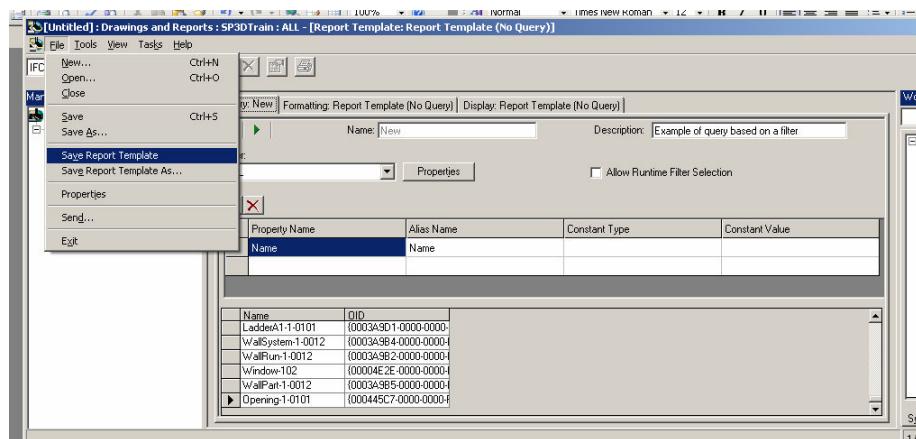
The screenshot shows two windows side-by-side:

- Microsoft Excel** window showing the 'Report Template (No Query).xls' sheet. Cell A9 contains the formula '#New::Name#' and cell B9 contains the formula '#New::OID#'. Both cells are highlighted in yellow.
- Design Layout** window showing the report structure. The 'Attributes' node under 'New' has 'Name' and 'OID' selected.

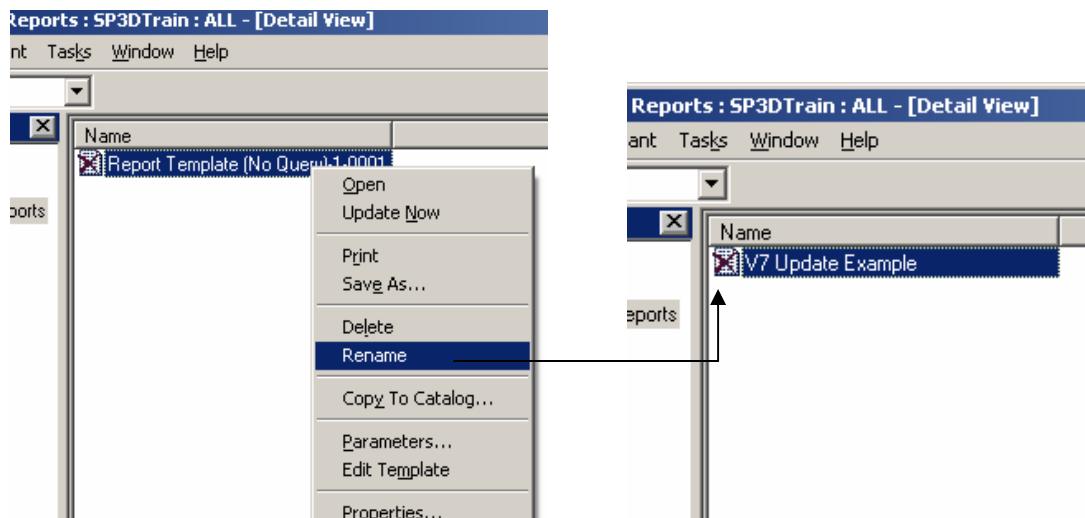
20. Close Excel and tell the software “Yes” when you are prompted to save:



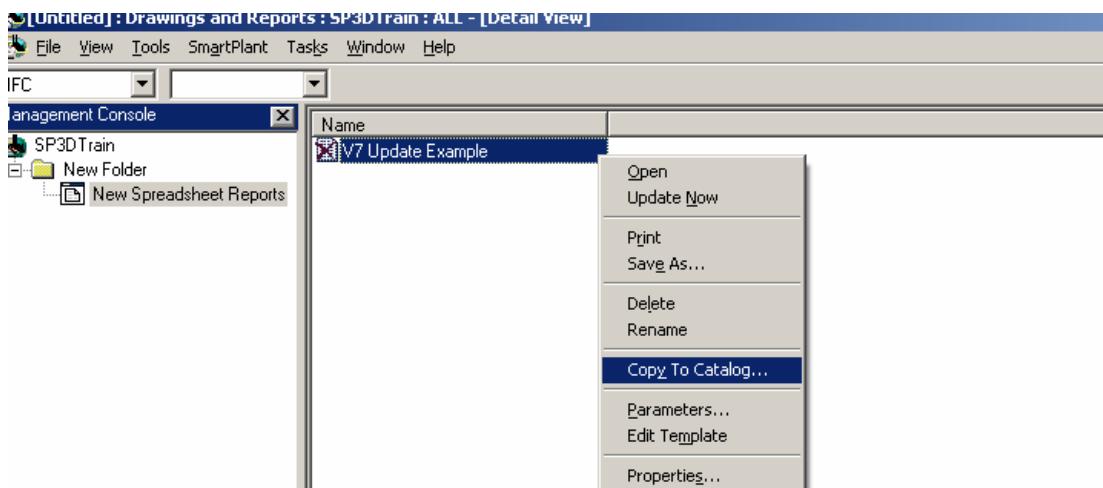
21. Save the Report Template again.



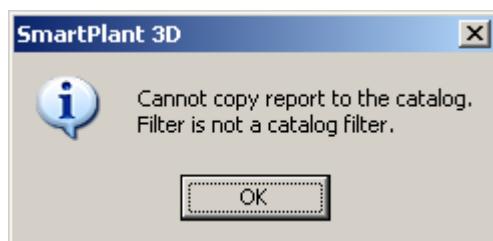
22. Rename the Report as shown below:



23. Use the Copy to Catalog... command to copy this report to the Catalog

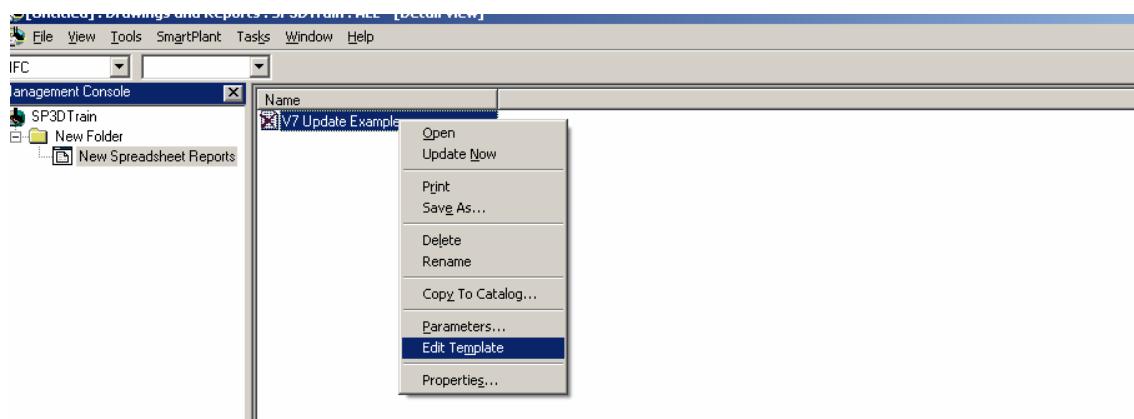


24. Note the restriction message:

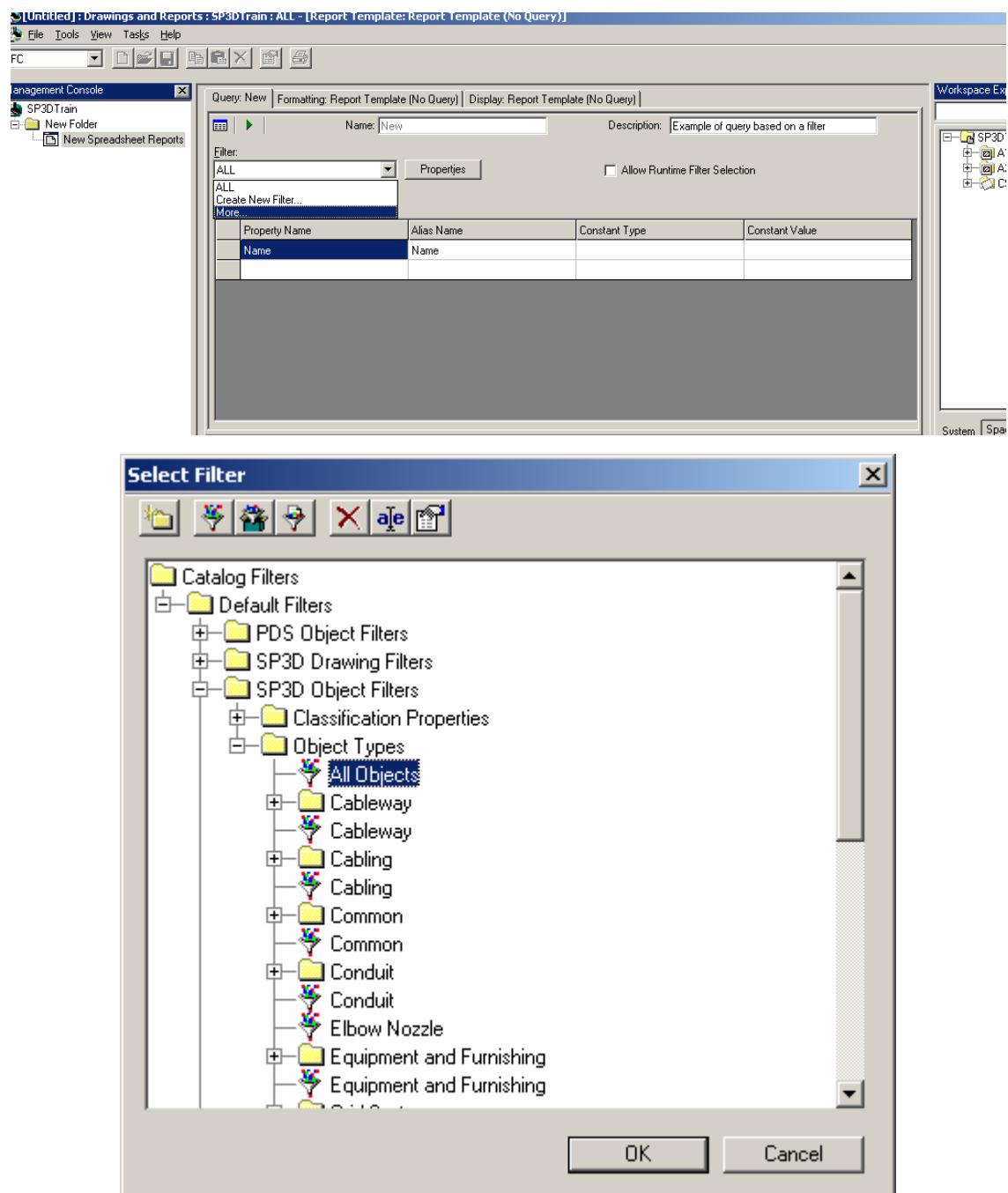


Recall that we said when choosing the All filter that it might not be our best choice. All reports that are copied to the Catalog must use Catalog filters for the Queries. We will see in a moment how to use ARFS to utilize Plant filters (in addition to the Catalog Filters) to control the report scope.

25. Use the Edit Template command to modify the query:



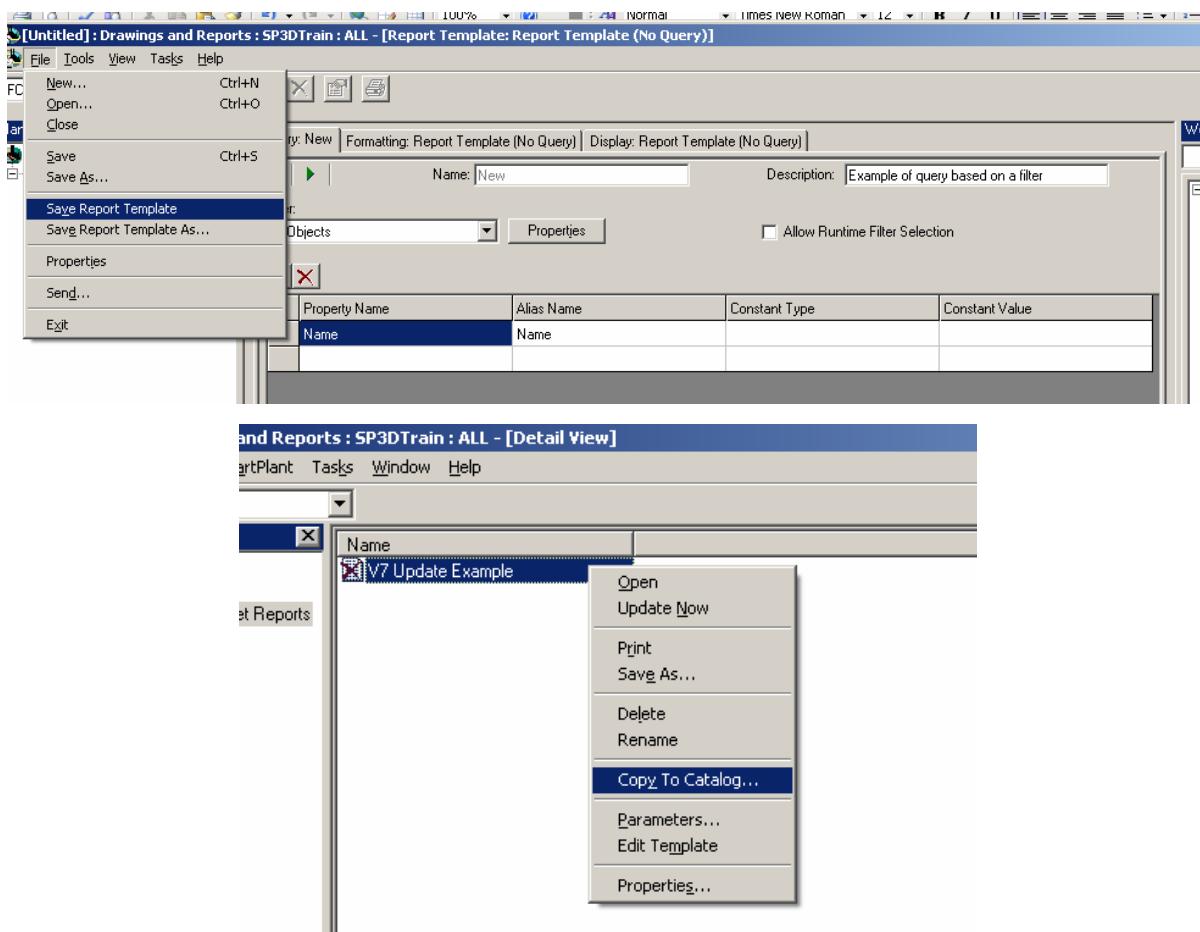
26. First, correct the Filter by selecting the More option as shown below, and selecting the “All Objects” filter from the Catalog.



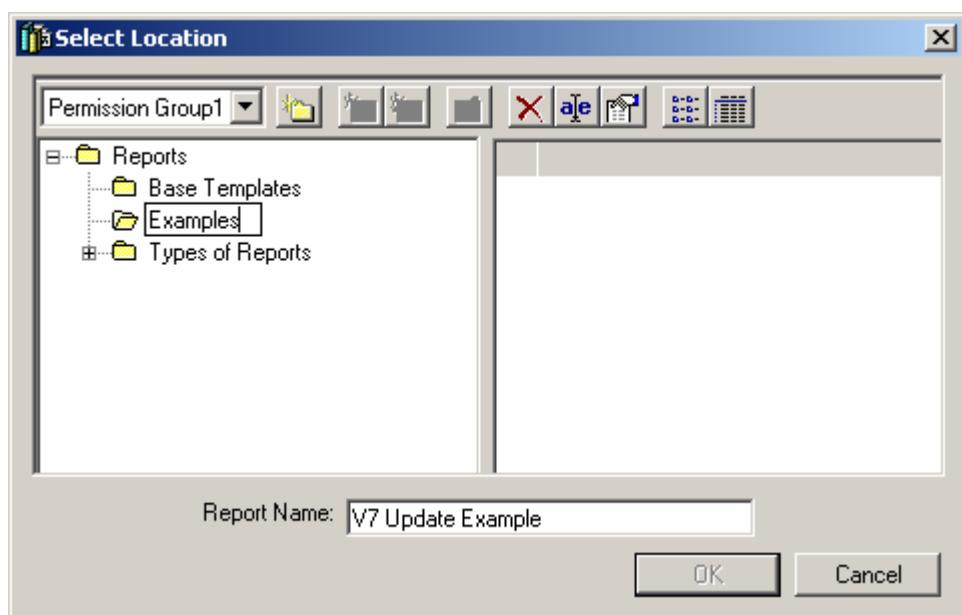
OK

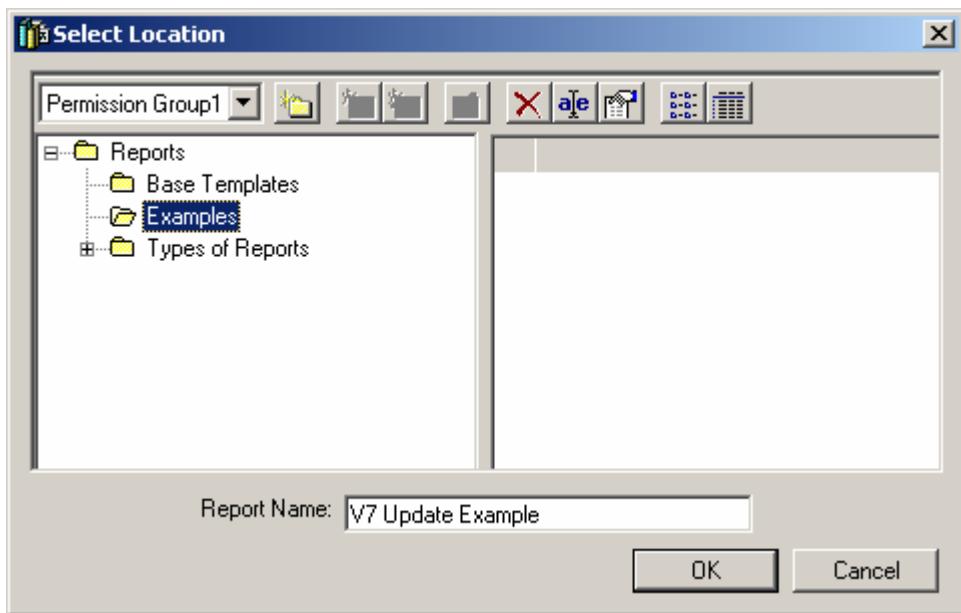
Cancel

27. Then save the Report Template and attempt to Copy to Catalog once again...



28. Create a new folder named Examples and keep the Report Name “V7 Update Example”

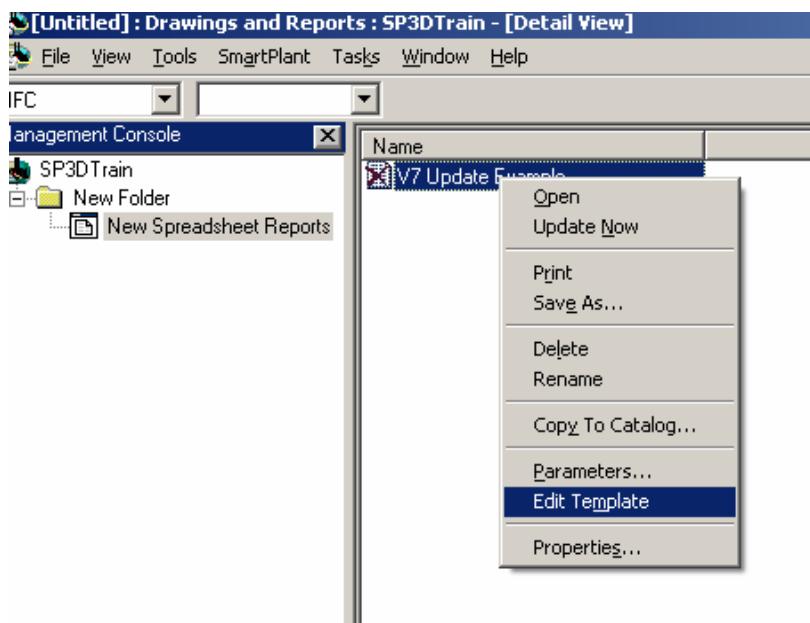




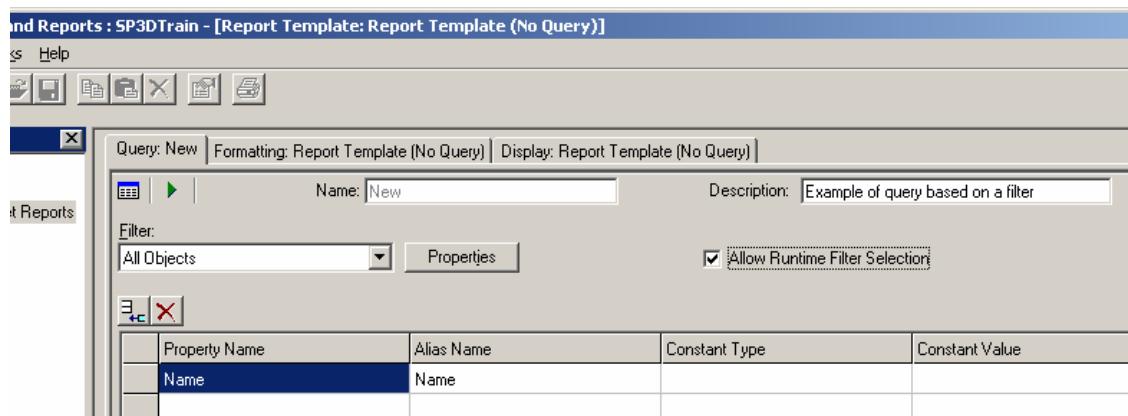
29. Click OK.

Note: This represents the report without ARFS. Now let us turn on ARFS and save another version of the same report.

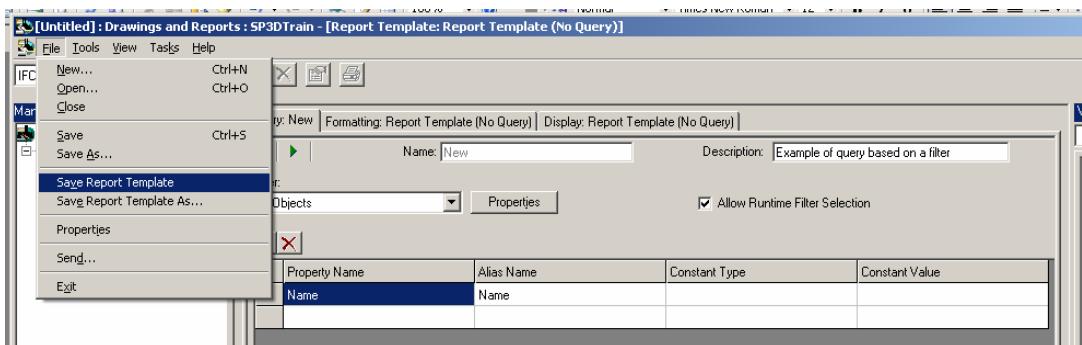
30. Use Edit Template to re-enter the report editor.



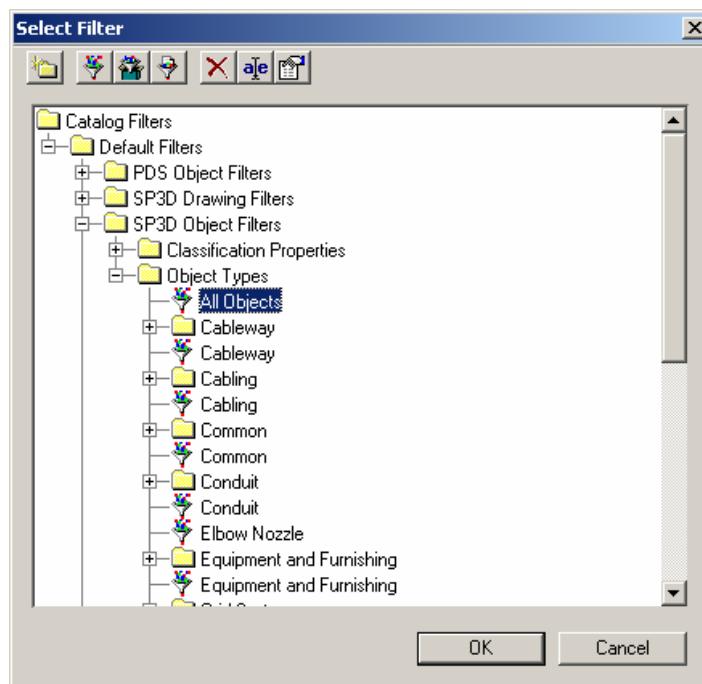
31. Place a Check in the box “Allow Runtime Filter Selection”



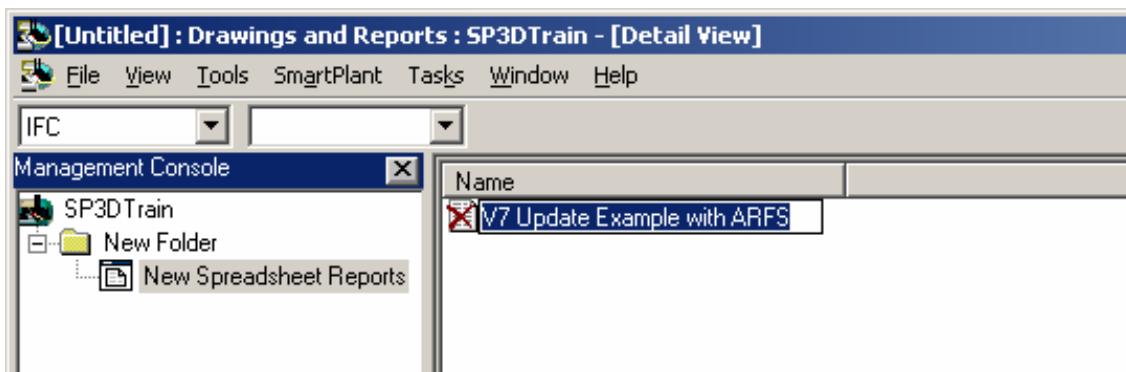
32. And Save the Report Template once again.



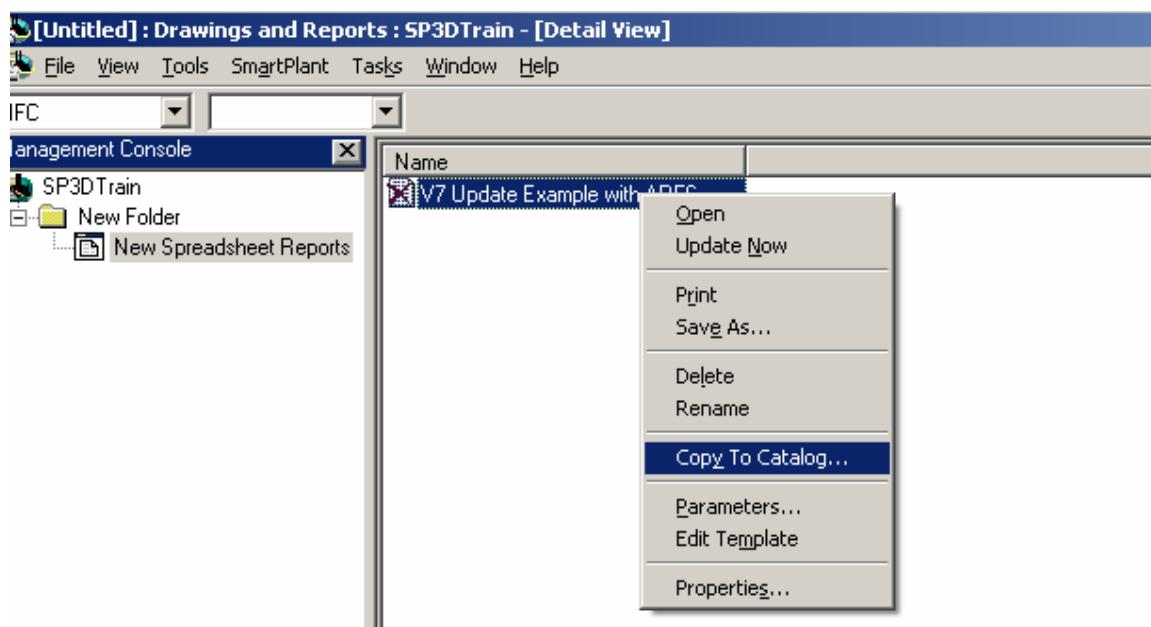
33. You may be prompted again to select the filter, if you are then select All Objects once again.

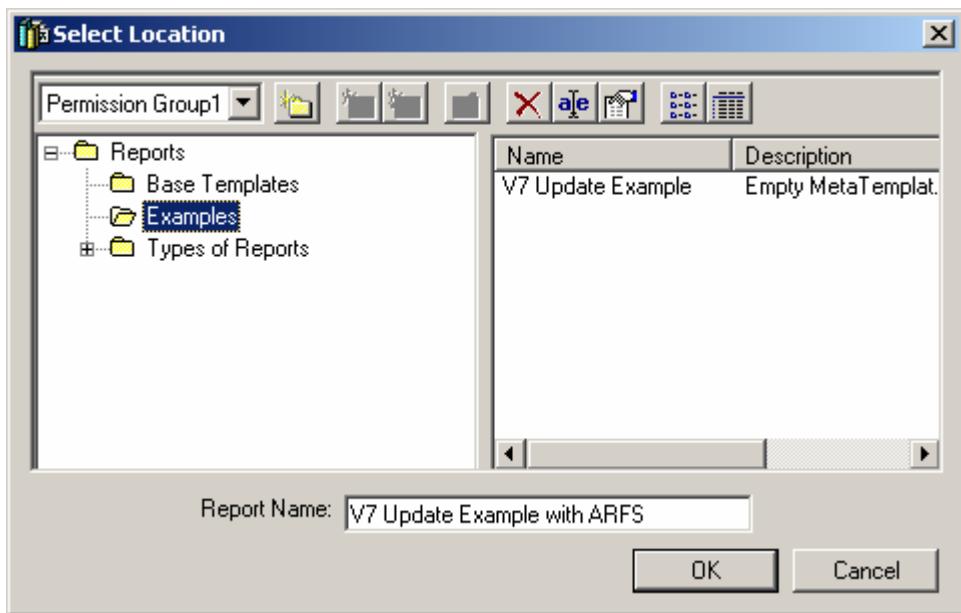


34. Rename the Report as shown below:



35. Repeat the Copy to Catalog Procedure:

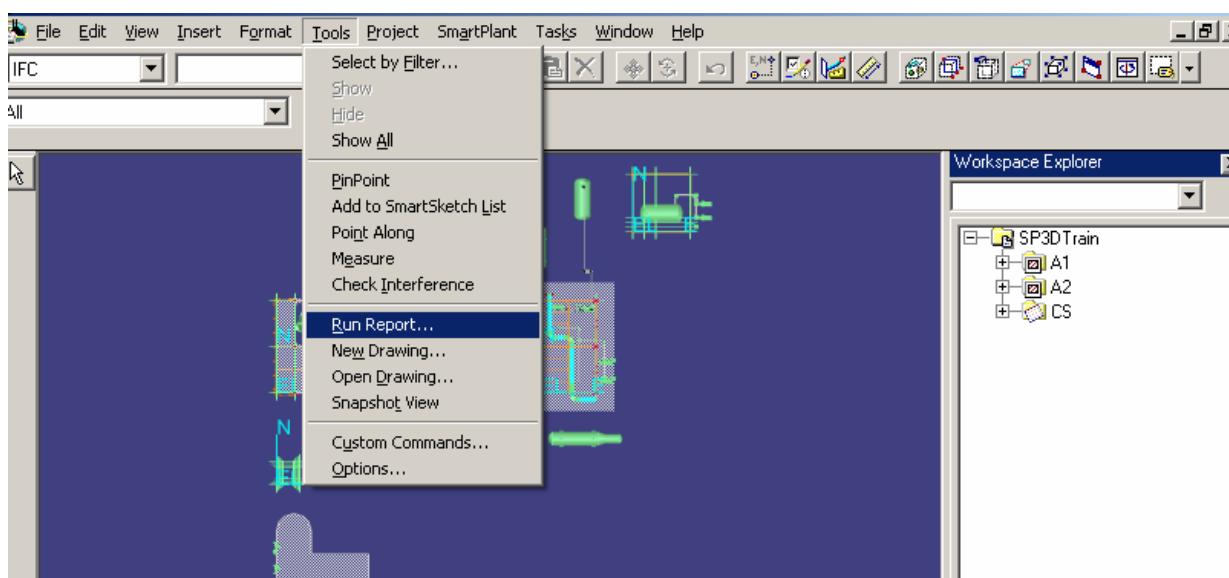




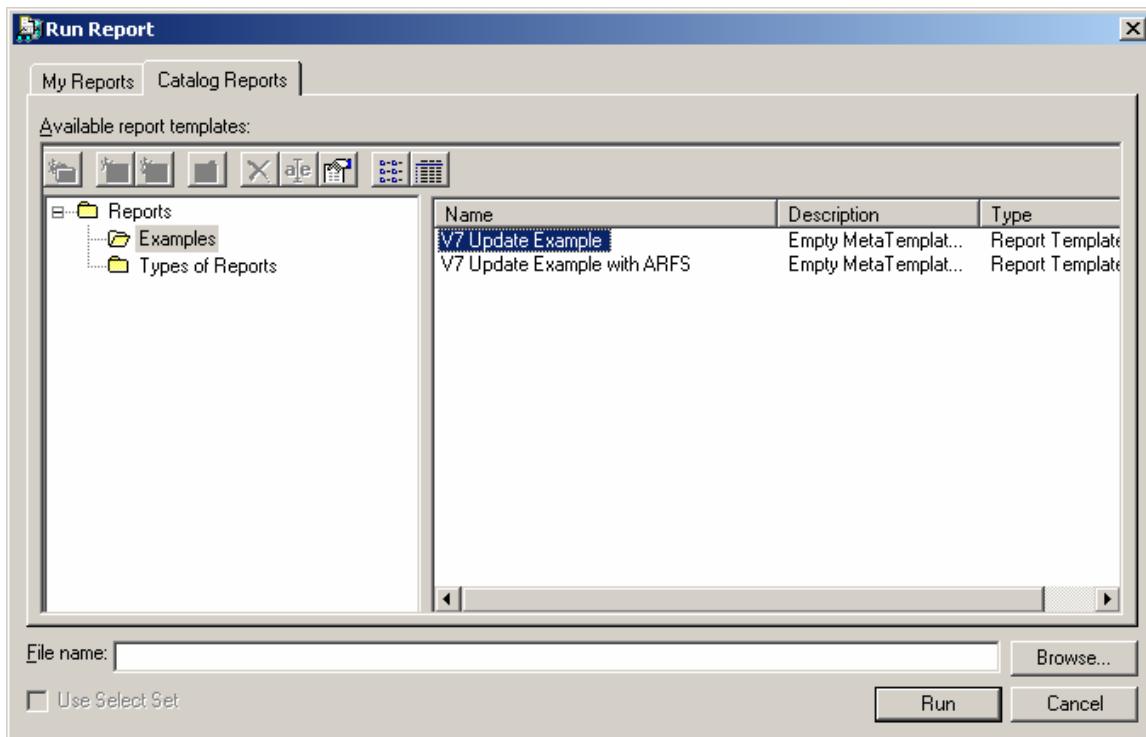
36. Click OK.

Note: We will run both reports and note the difference in functionality with and without ARFS.

37. Switch to the Common task and execute the Tools→Run Report.. command.



38. Choose the standard V7 Update Example.

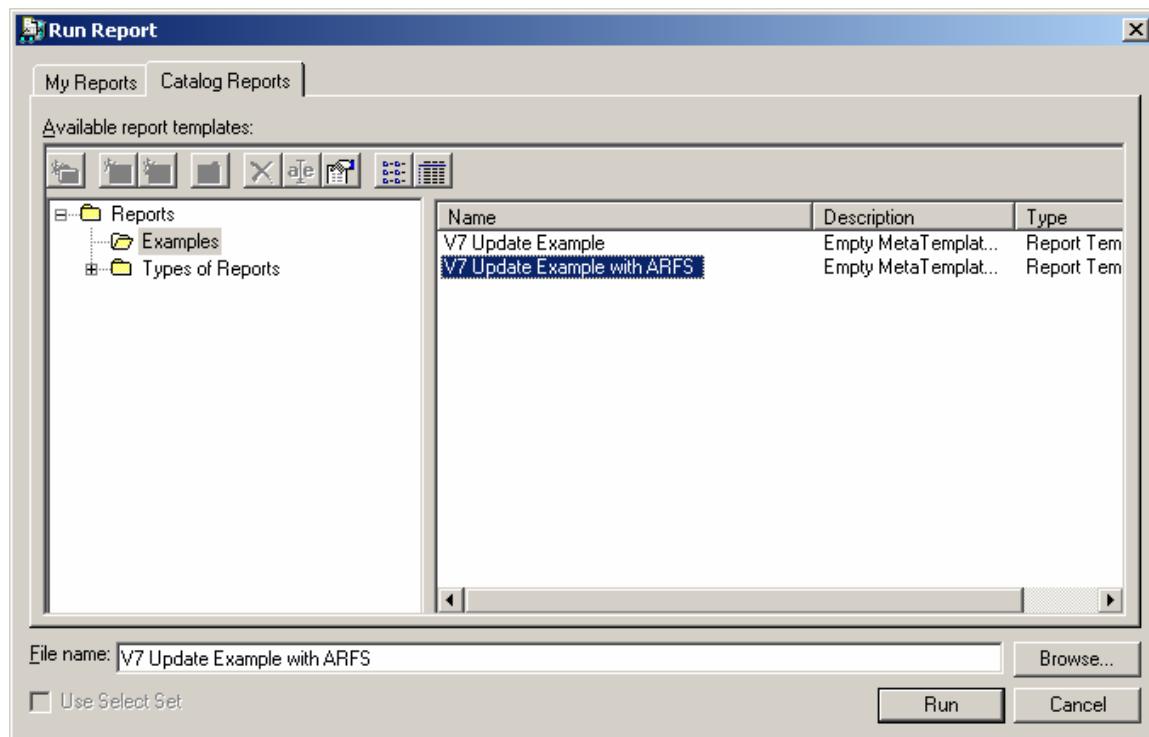


39. Note that it runs without prompting the user for an additional filter.

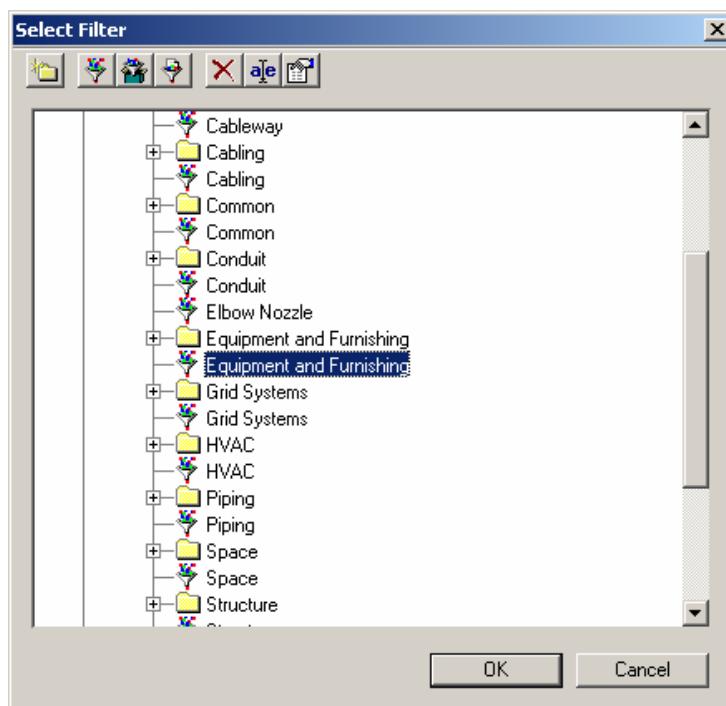
The screenshot shows an Excel spreadsheet titled 'Report Output.xls [Read-Only]'. The table has columns A, B, C, and D. The data consists of pairs of item names and their GUIDs. The first few rows are:

	B	C	D
1			
2			
3			
4			
5			
6			
7			
8			
9	As-Built	(0000274A-0000-0000-0700-3EAAA1451D04)	
10	Electrical	(0000274B-0000-0000-0A00-3EAAA1451D04)	
11	Mechanical	(0000274B-0000-0000-0B00-3EAAA1451D04)	
12	Piping	(0000274B-0000-0000-0C00-3EAAA1451D04)	
13	Structural	(0000274B-0000-0000-0D00-3EAAA1451D04)	
14	Project1	(0000274A-0000-0000-0E00-3EAAA1451D04)	
15	Contract1	(0000274B-0000-0000-1100-3EAAA1451D04)	
16	Contract2	(0000274B-0000-0000-1200-3EAAA1451D04)	
17	Contract3	(0000274B-0000-0000-1300-3EAAA1451D04)	
18	Contract4	(0000274B-0000-0000-1400-3EAAA1451D04)	
19	Design Area1	(0000274B-0000-0000-1500-3EAAA1451D04)	
20	Design Area2	(0000274B-0000-0000-1600-3EAAA1451D04)	
21	Nipple-0201	(00013885-0000-0000-4599AB452004)	
22	403-P	(000138AA-0000-0000-0107-4599AB452004)	
23	Pipe	(0001388C-0000-0000-010A-4599AB452004)	
24	Flange-0311	(00013885-0000-0000-0206-4599AB452004)	
25	301-VW	(000138AA-0000-0000-0313-4599AB452004)	
26	301-W	(000138AA-0000-0000-0513-4599AB452004)	
27	U04-4-P-0211-1C0031	(0001388D-0000-0000-0600-4599AB452004)	
28	Pipe	(0001388C-0000-0000-060B-4599AB452004)	
29	Flange-0324	(00013885-0000-0000-060C-4599AB452004)	

40. Now return to the Run Report command and run the 2nd report "...with ARFS"



41. Note that you are now prompted to pick a filter, for this example let us choose Equipment and Furnishing from the same branch in the Catalog that we find the All Objects filter (you can return and try plant filters as well on your own).



42. Note that when the report runs, the results are now restricted based on this additional filter selection.

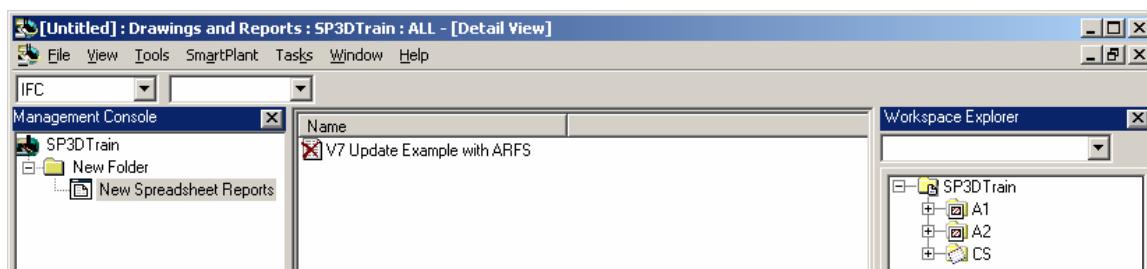
	B	C	D
1			
2			
3			
4			
5			
6			
7			
8			
9	1st Leg Support	{00004E27-0000-0000-1002-55D1AA451D04}	
10	T-101	{00004E2E-0000-0000-1400-55D1AA451D04}	
11	E-102-Shape-0108	{00004E27-0000-0000-1B03-55D1AA451D04}	
12	DP1	{00004E27-0000-0000-2F00-55D1AA451D04}	
13	DP2	{00004E27-0000-0000-3100-55D1AA451D04}	
14	B	{00004E23-0000-0000-3300-55D1AA451D04}	
15	C	{00004E23-0000-0000-3600-55D1AA451D04}	
16	D1	{00004E23-0000-0000-3900-55D1AA451D04}	
17	D2	{00004E23-0000-0000-3C00-55D1AA451D04}	
18	E2	{00004E23-0000-0000-3F00-55D1AA451D04}	
19	E1	{00004E23-0000-0000-4200-55D1AA451D04}	
20	A	{00004E23-0000-0000-4500-55D1AA451D04}	
21	2nd Leg Support	{00004E27-0000-0000-4B02-55D1AA451D04}	
22	VesselFoundationPort	{00004E26-0000-0000-4E00-55D1AA451D04}	
23	N1	{00004E23-0000-0000-5303-55D1AA451D04}	
24	P-101	{00004E2E-0000-0000-5600-55D1AA451D04}	
25	DP1	{00004E27-0000-0000-5900-55D1AA451D04}	
26	Suction	{00004E23-0000-0000-5B00-55D1AA451D04}	
27	Discharge	{00004E23-0000-0000-5E00-55D1AA451D04}	
28	40P-101B	{00004E2E-0000-0000-6501-55D1AA451D04}	
29	PumpFoundationPort	{00004E26-0000-0000-6600-55D1AA451D04}	

Assigning queries to different sheets in the Report (Multi-sheet reports)

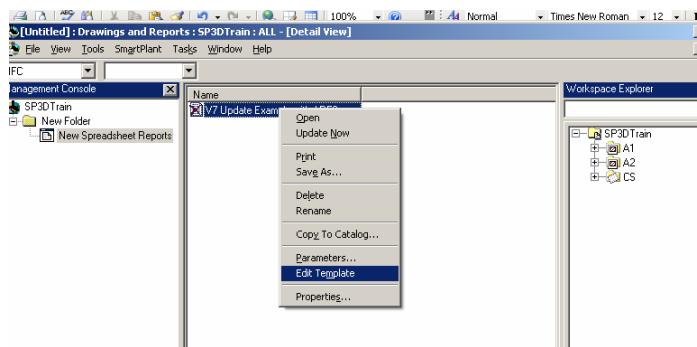
Objectives

After completing this lab, you will be able to:

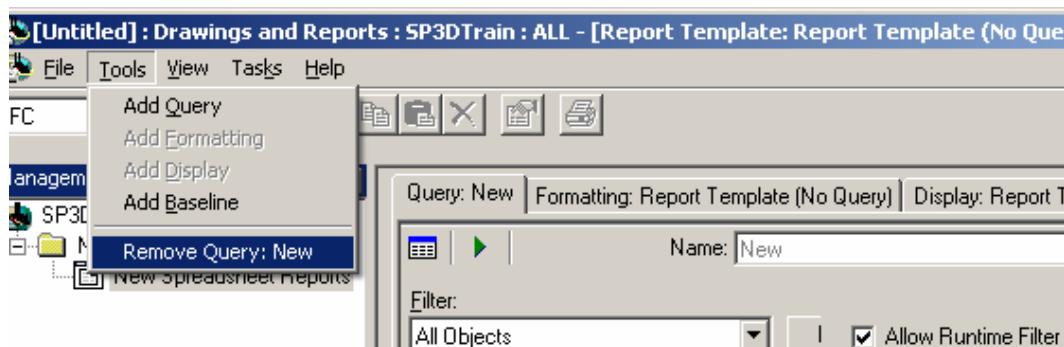
- Create a multi sheet report and assign individual queries to different sheets.
1. Let us re-use the template we created in Lab1.



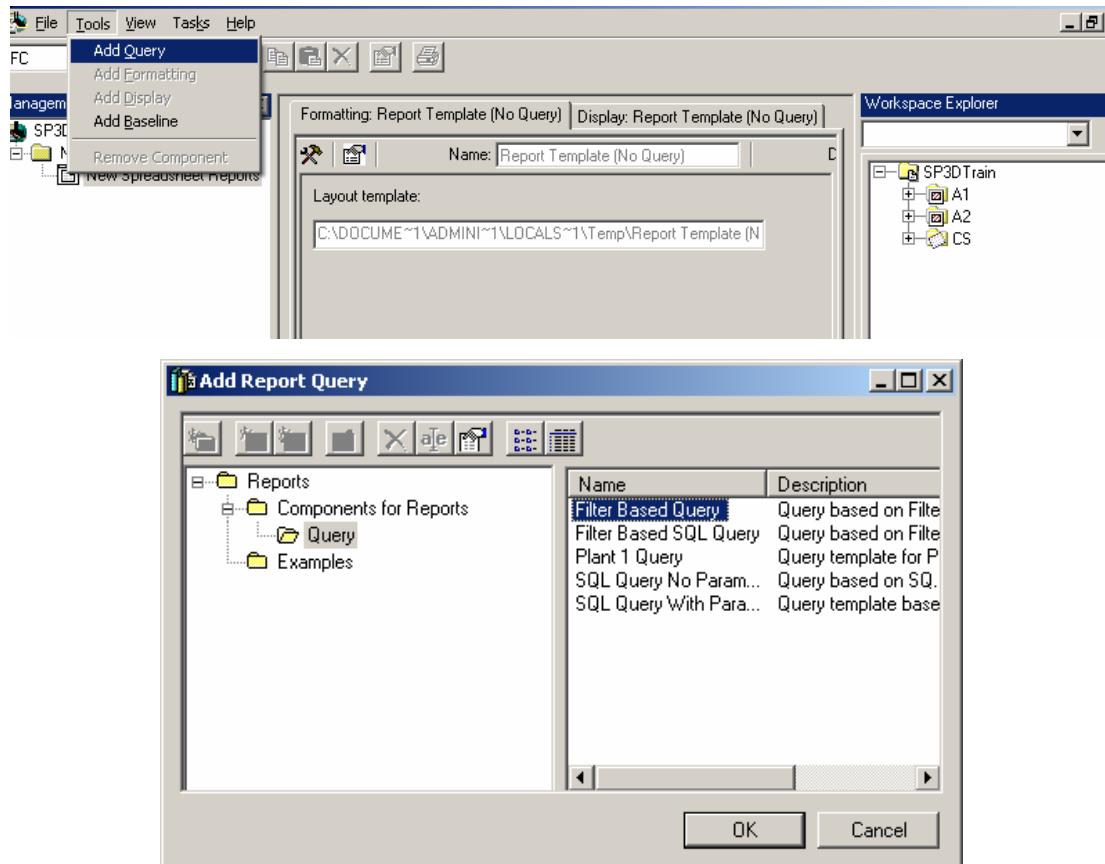
2. Enter the Template editor once again.

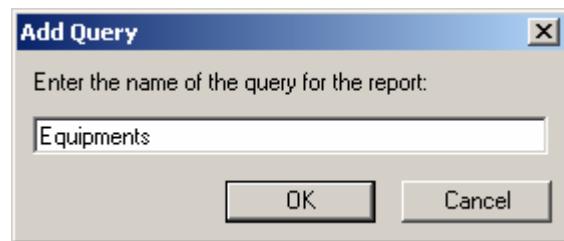


3. Use the Tools → Remove Query command to remove the old Query from the template.

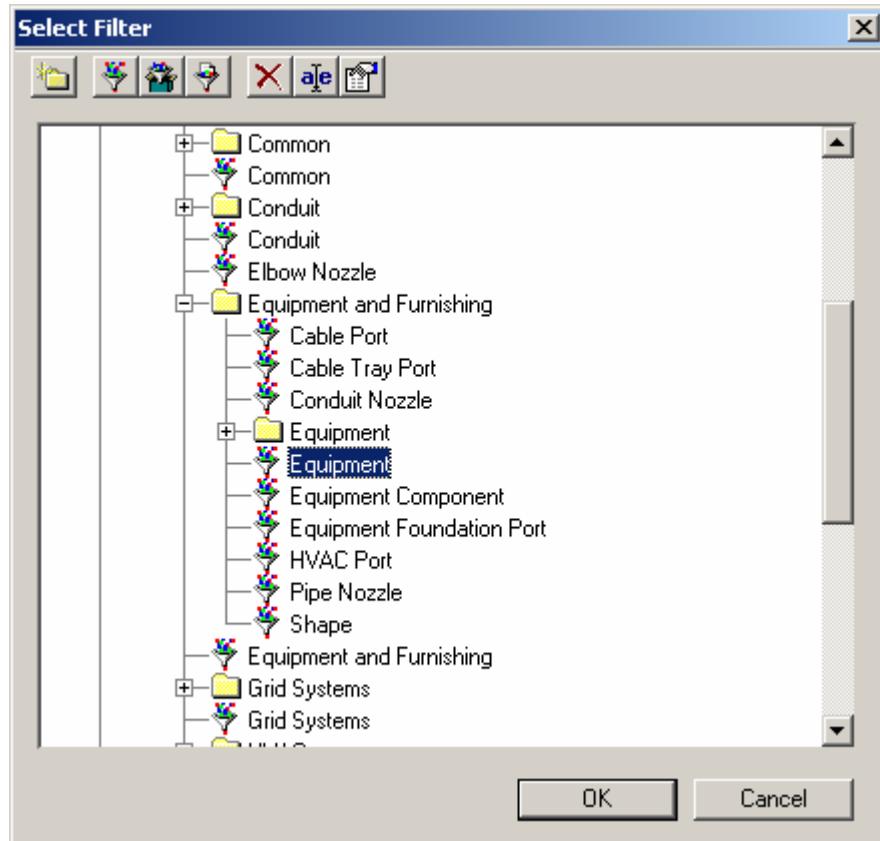


4. Use the Tools → Add Query command as we did in lab 1 to add a query for Equipments.

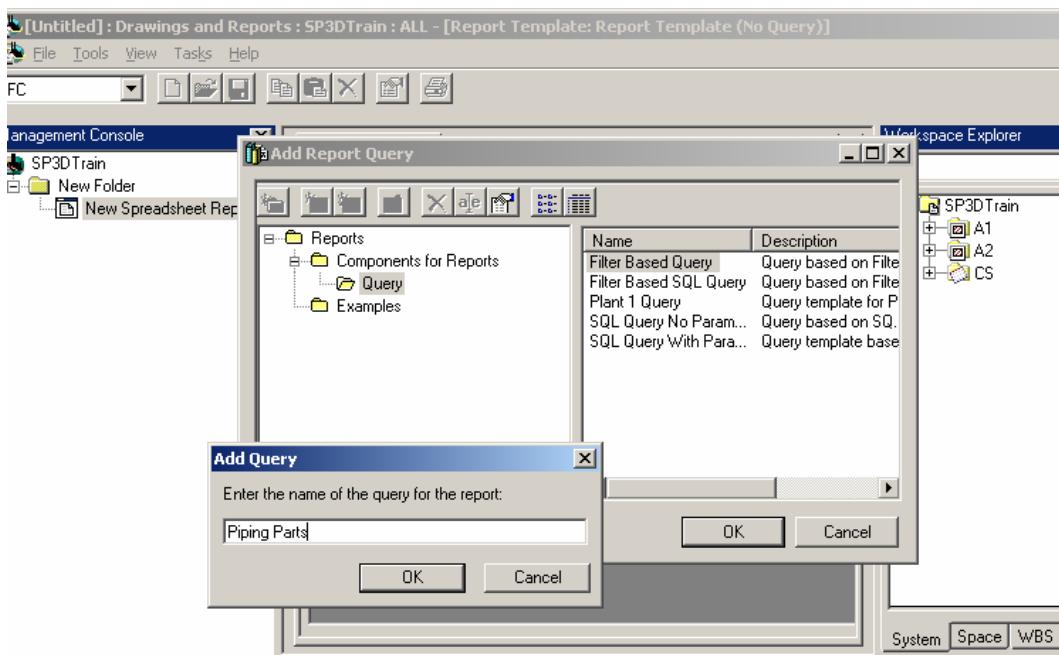




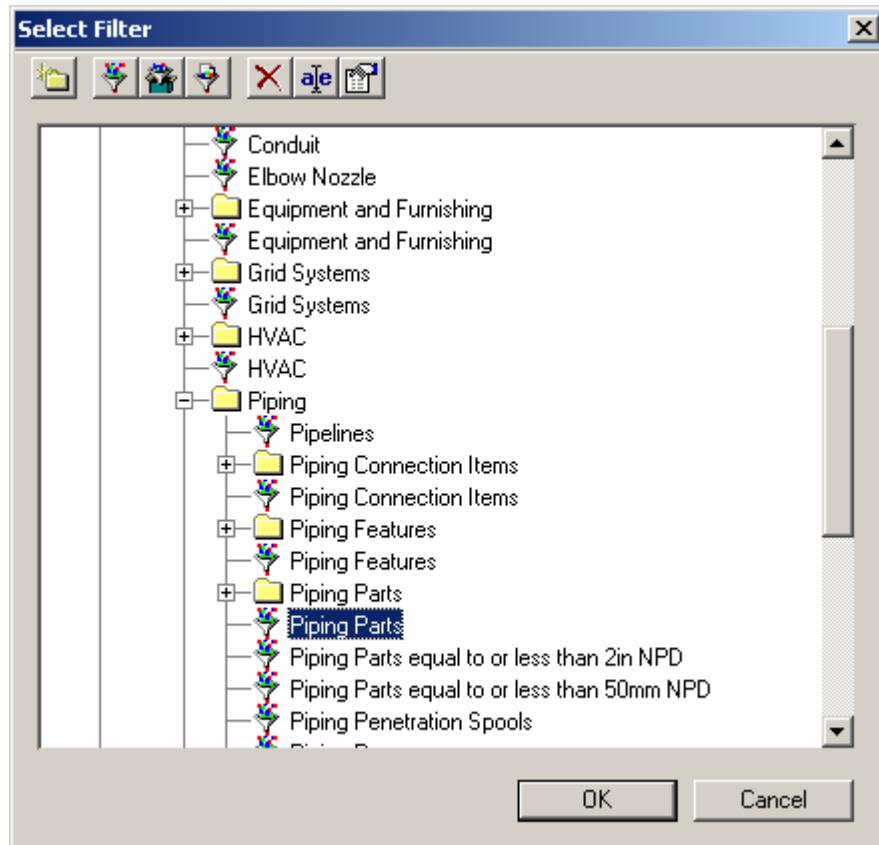
5. Choose the Catalog Object Filter “Equipment” as shown below:



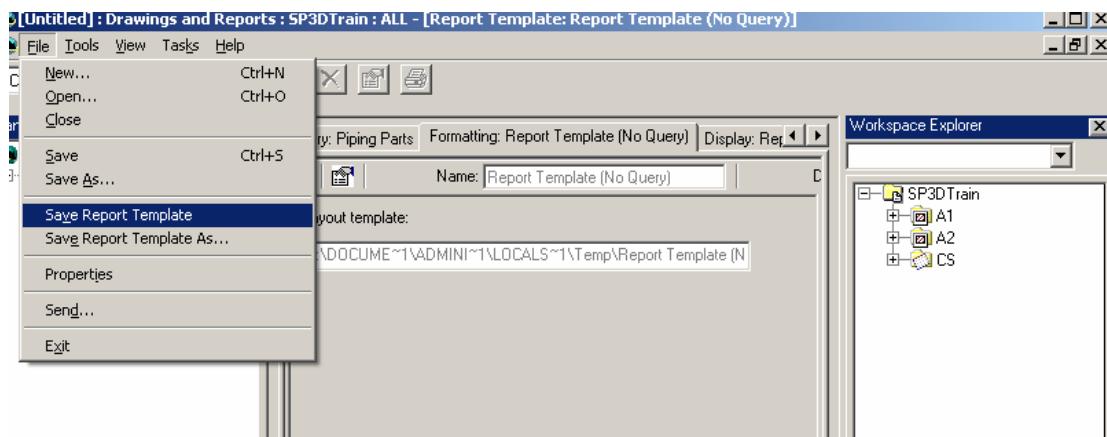
6. Repeat that same process again to add one for Piping.



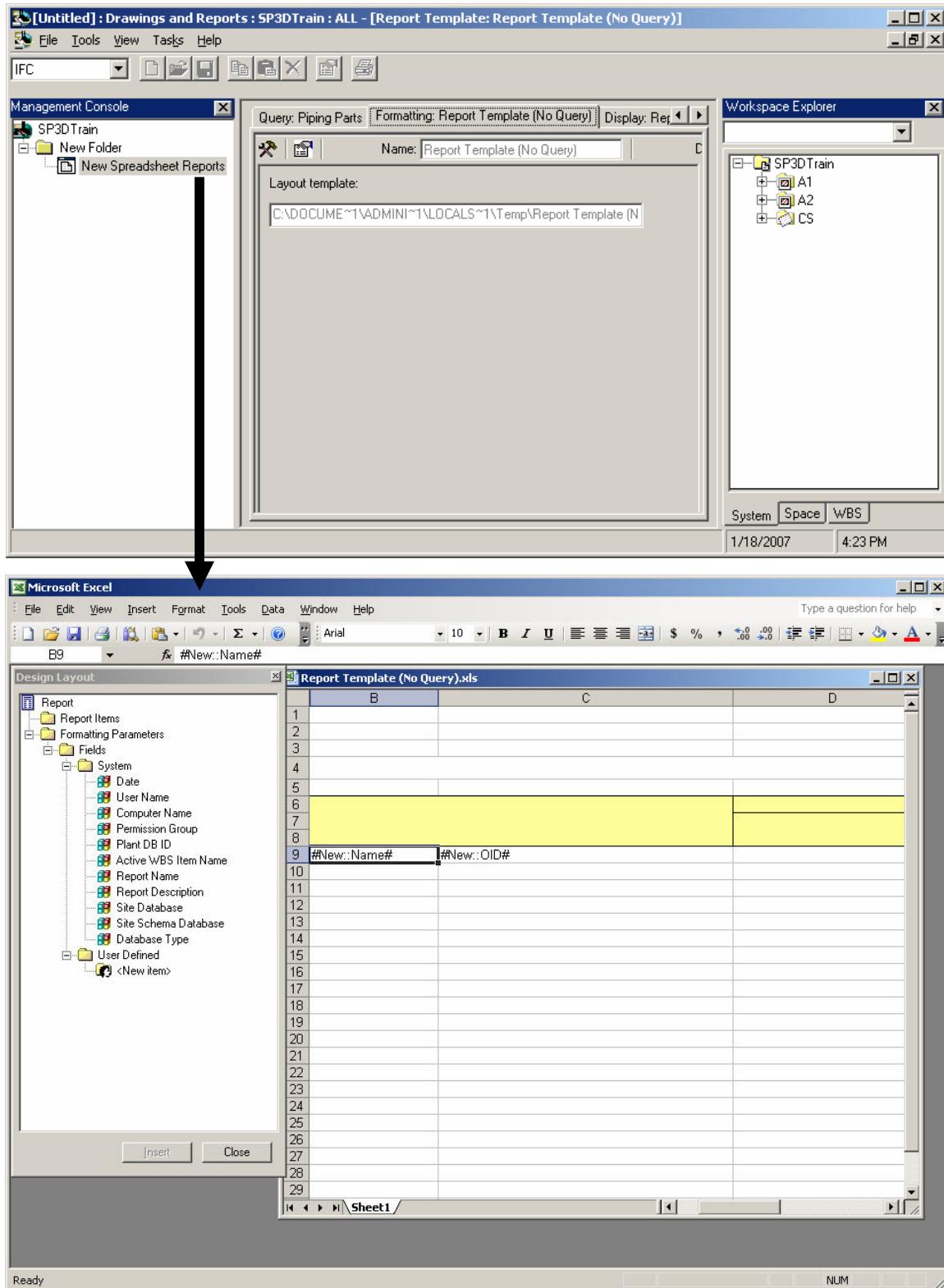
7. Choose the Catalog Object Type filter Piping Parts as shown below:



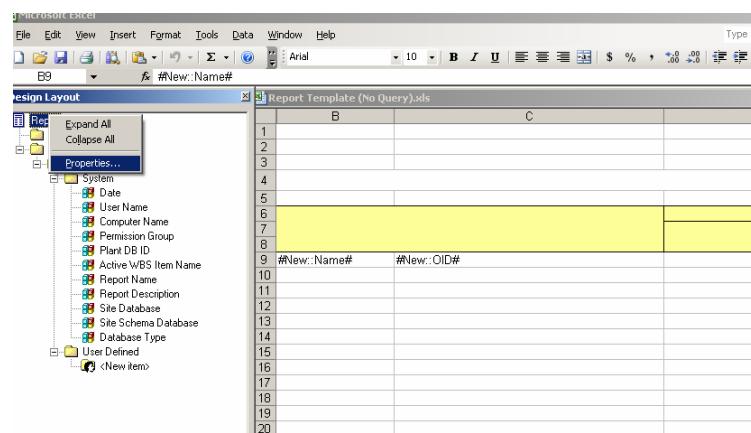
8. Save the Report Template.



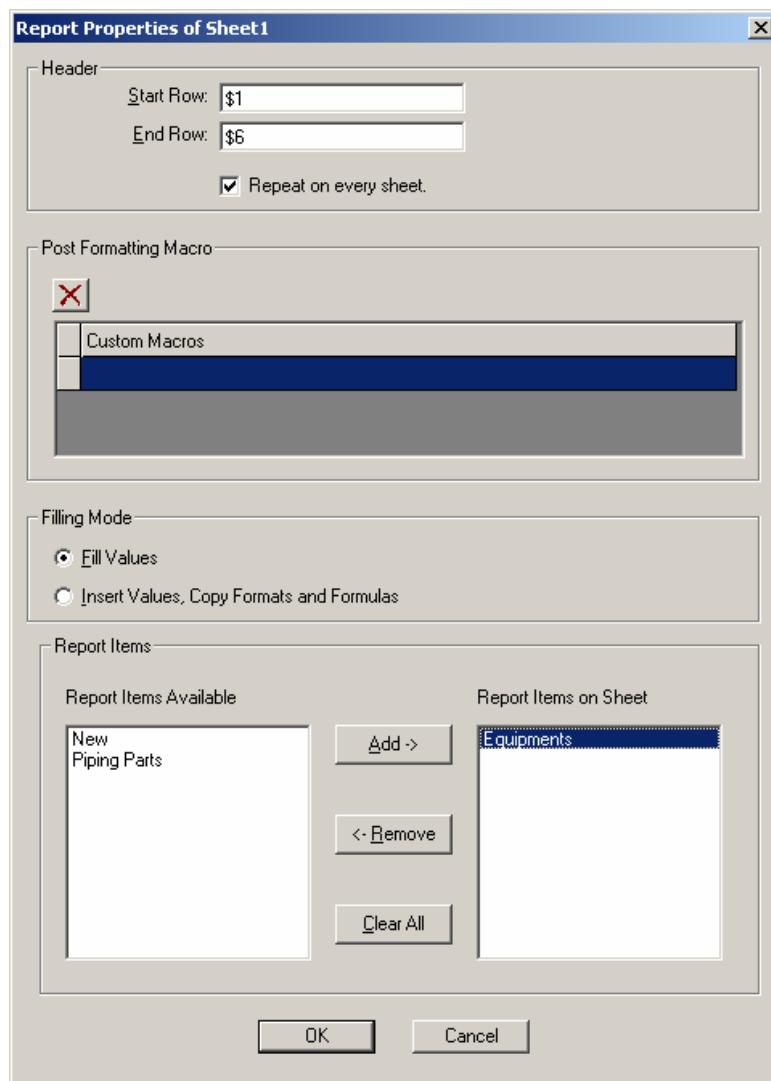
9. On the formatting tab, use the  button to enter Excel.



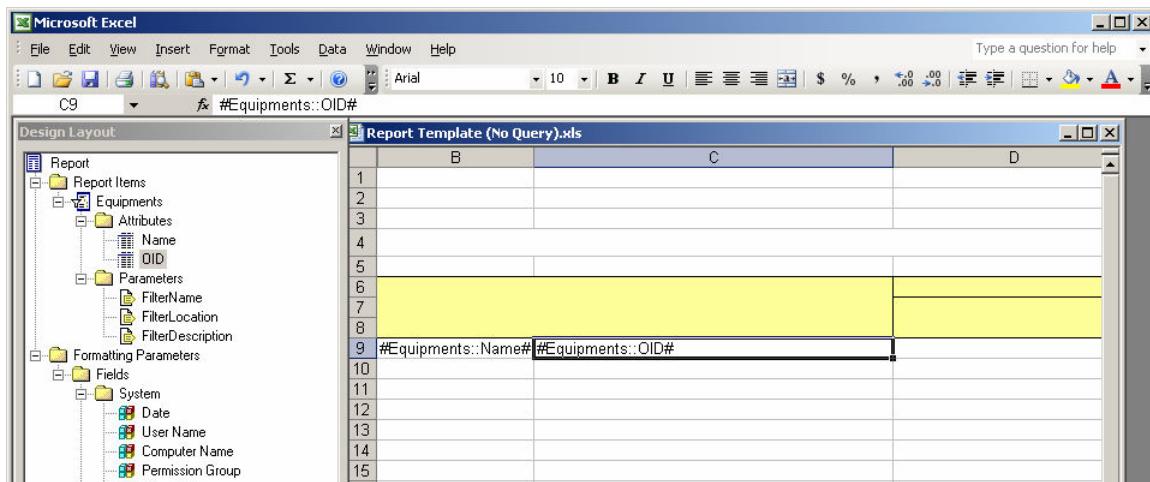
10. Use the properties button as before access the Report Properties form.



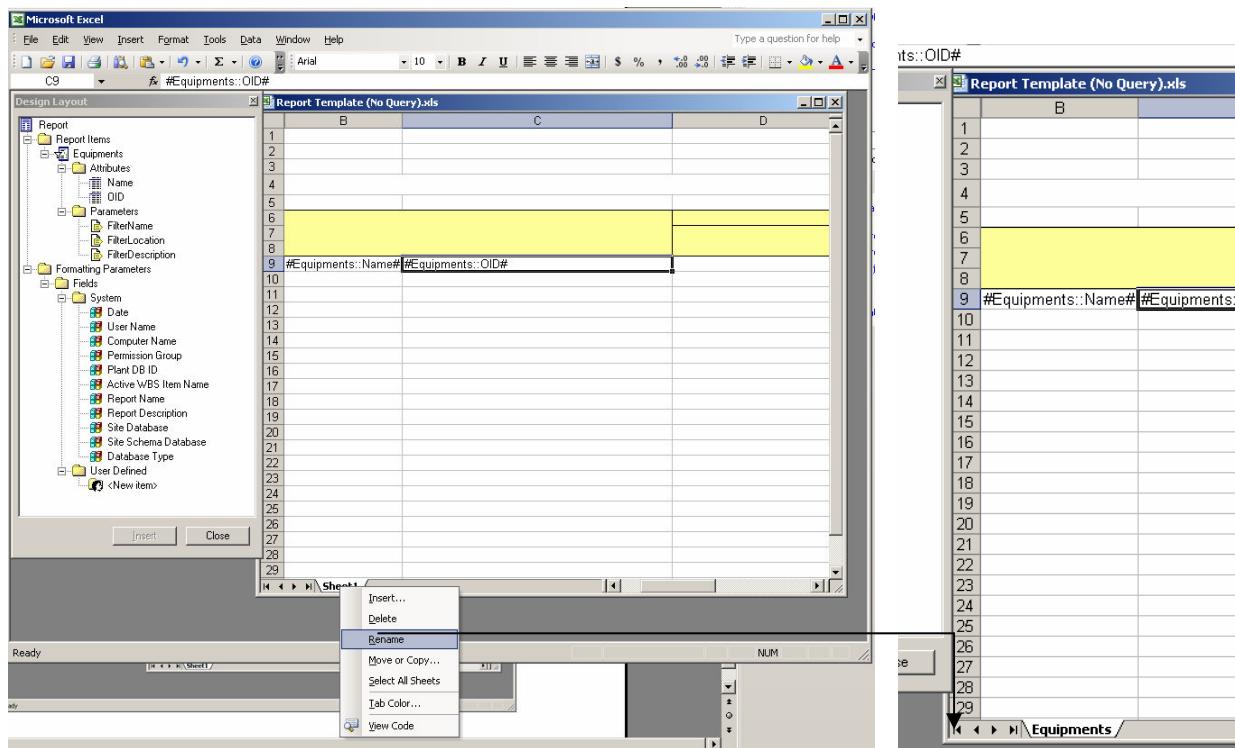
11. Add the Equipments query into the column as shown below. This will make this query set available on this sheet in the workbook.



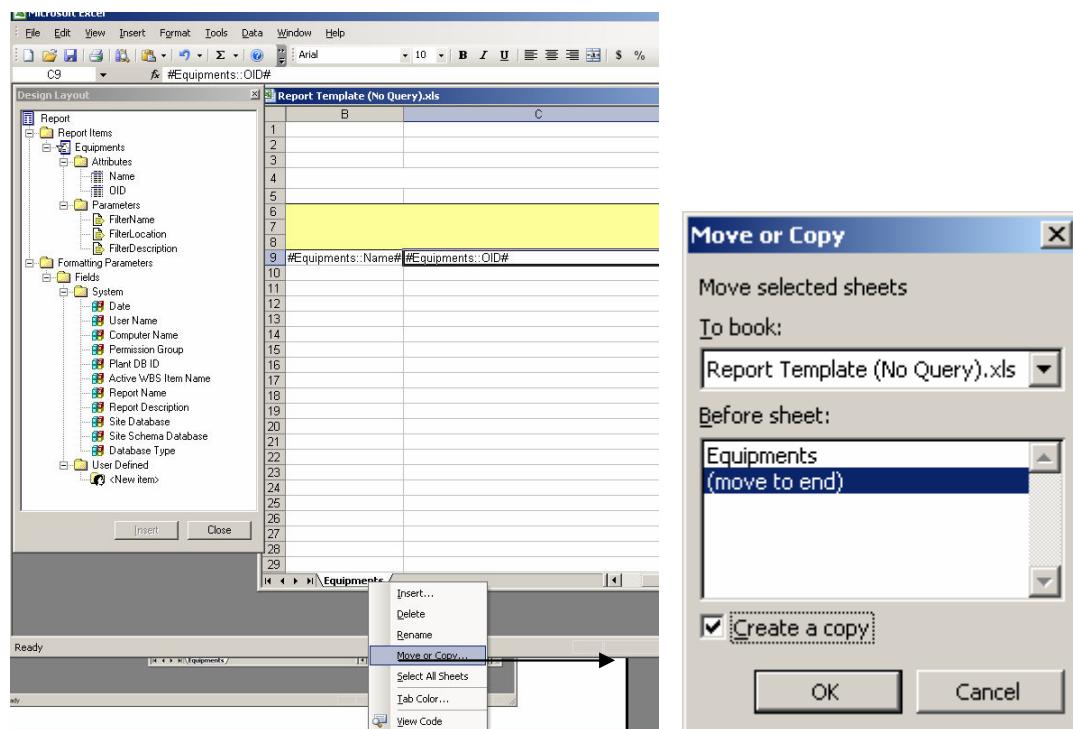
Note that the Report Query Equipments is exposed, drag and drop the Name and OID fields onto the Workbook.



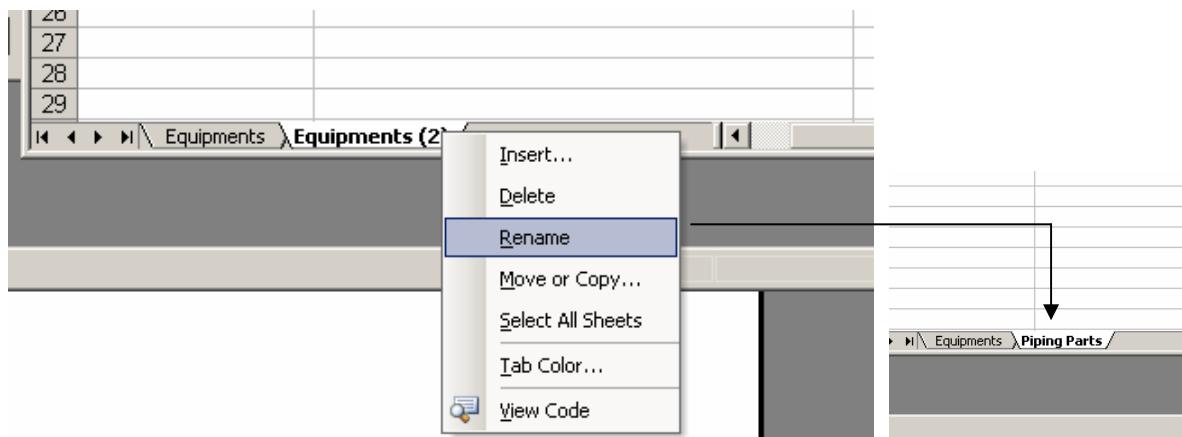
12. Rename the Excel sheet “Equipments”



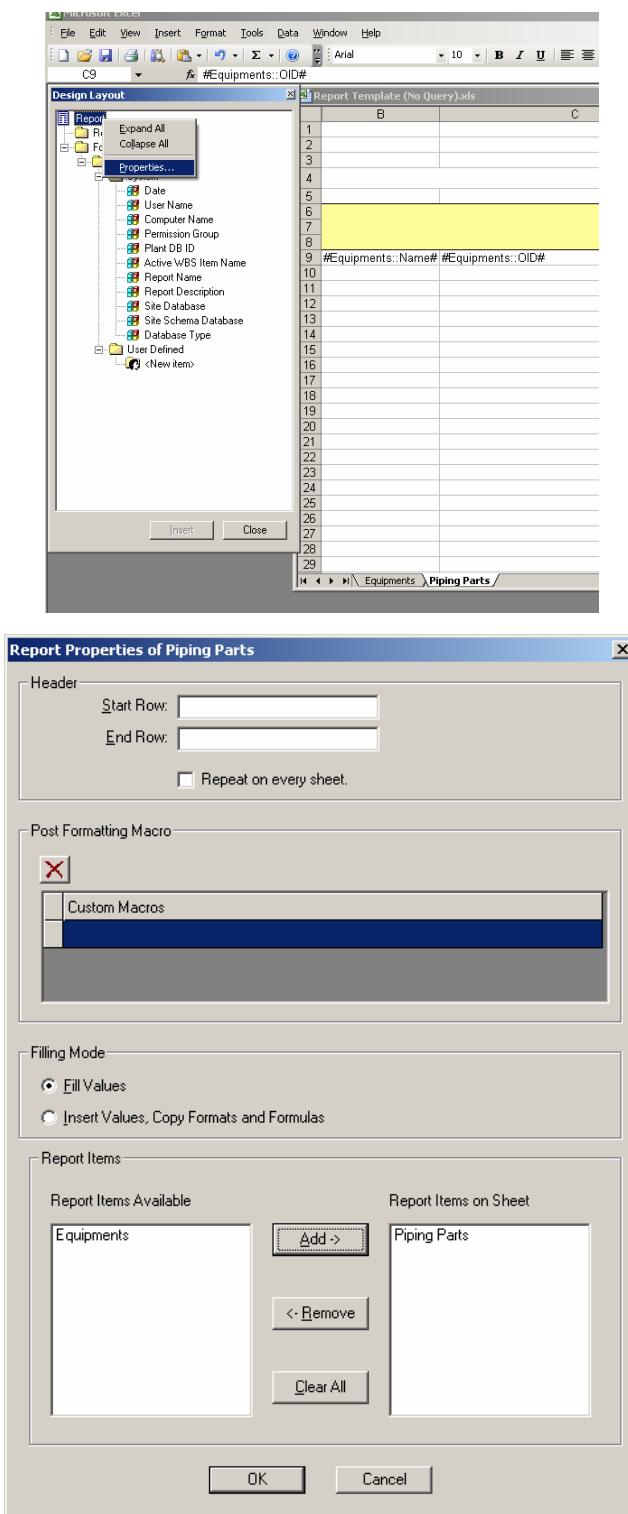
13. “Use the Move or Copy...” command to copy the existing Equipment Sheet to the end of the workbook.



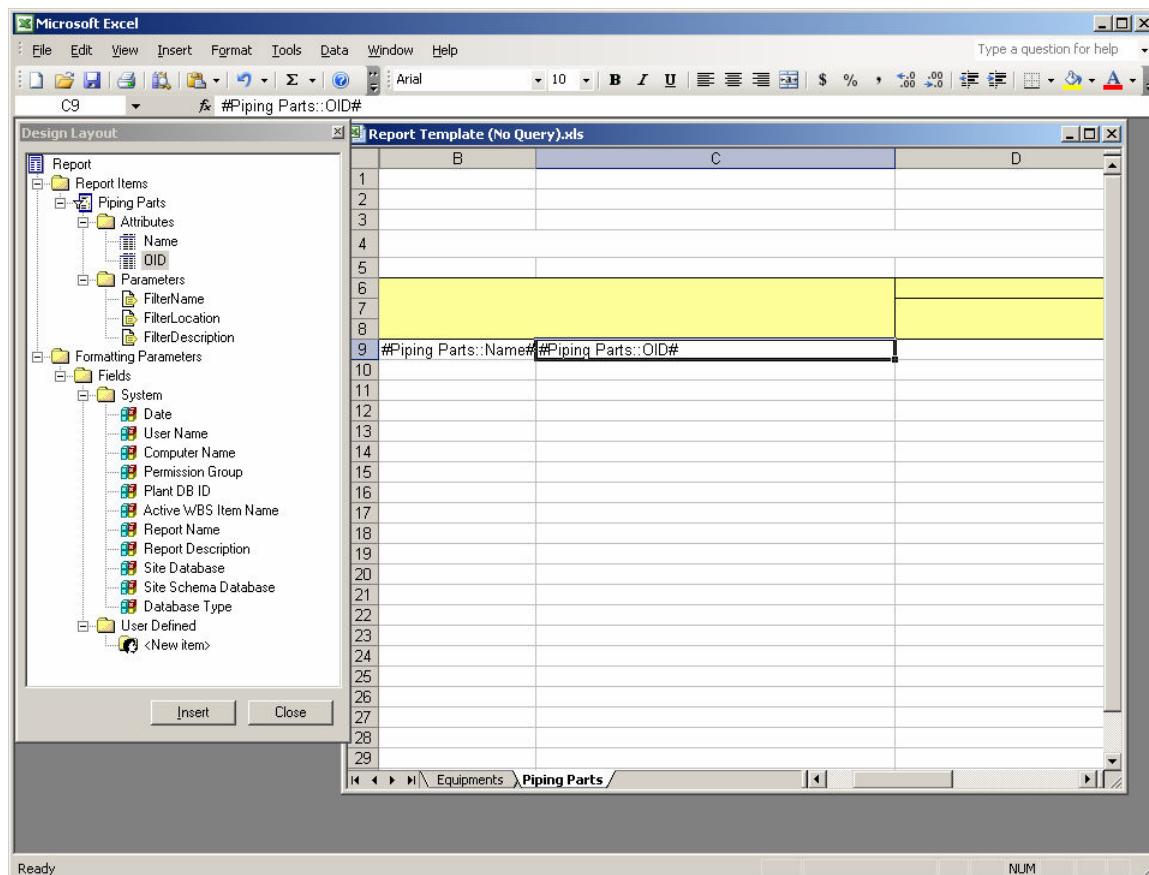
14. Then rename the sheet “Piping Parts”



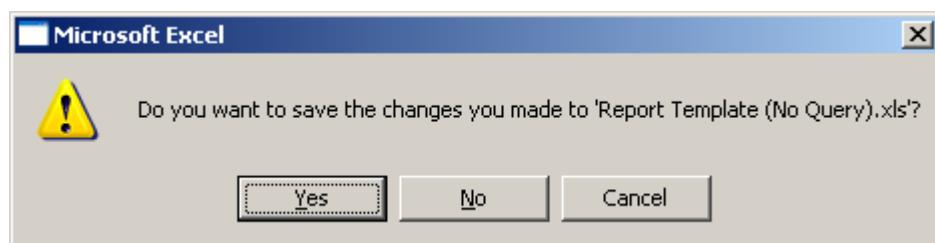
Note that no Queries are available for the new sheet, we will need to add in the Piping Parts query in the same manner as we did for the Equipments:



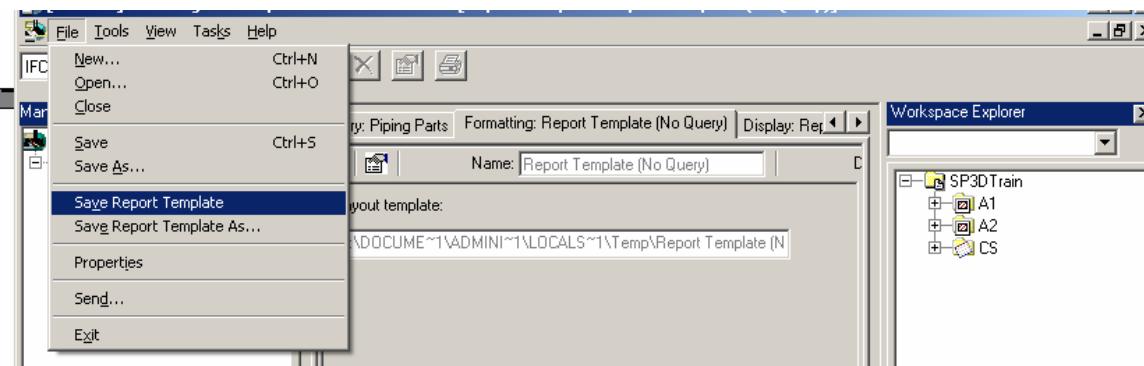
15. Drag and drop the Name and OID from the piping parts query onto the new sheet.



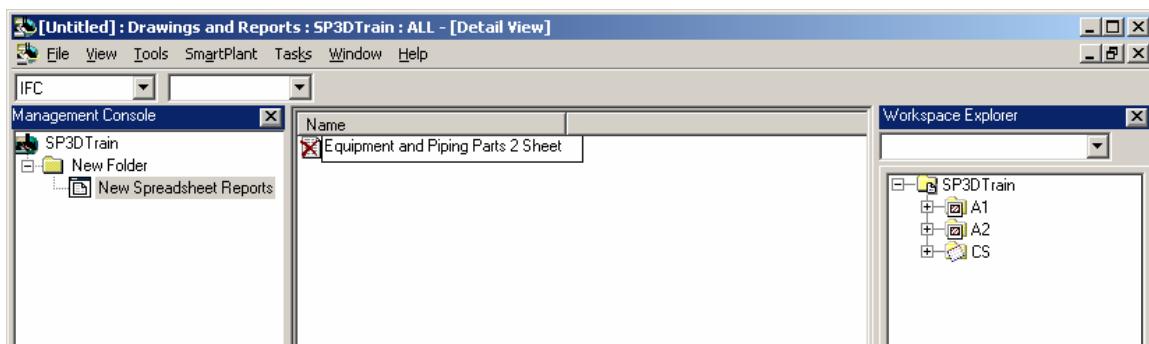
16. Close Excel and say Yes when prompted to save.



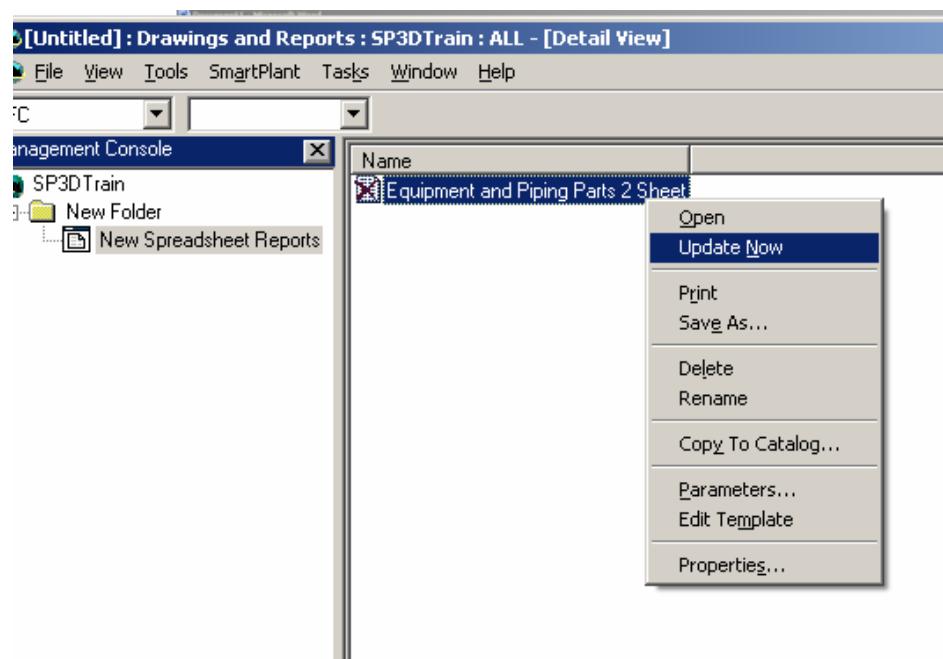
17. Save the Report Template.



18. Rename the Report.



19. Update the Report.



20. Review the two sheet results:

The image displays two separate Microsoft Excel windows side-by-side, both titled "Microsoft Excel - Equipment and Piping Parts 2 Sheet.xls [Read-Only]".

Left Window (Equipment List):

	B	C
1		
2		
3		
4		
5		
6		
7		
8		
9	T-101	{00004E2E-0000-0000-1400-55D1AA451D04}
10	P-101	{00004E2E-0000-0000-5600-55D1AA451D04}
11	40P-101B	{00004E2E-0000-0000-6501-55D1AA451D04}
12	40V-101	{00004E2E-0000-0000-7700-55D1AA451D04}
13	E-102	{00004E2E-0000-0000-9601-55D1AA451D04}
14	41V-101	{00004E2E-0000-0000-B000-55D1AA451D04}
15	DR-100	{00004E2E-0000-0000-B203-55D1AA451D04}
16	Tank	{00004E30-0000-0000-BA00-55D1AA451D04}
17	40P-101A	{00004E2E-0000-0000-E000-55D1AA451D04}
18	40E-101A	{00004E2E-0000-0000-0600-840CAB451D04}
19	40E-101B	{00004E2E-0000-0000-5400-840CAB451D04}
20	TA-101	{00004E2E-0000-0000-9400-840CAB451D04}
21	PU2-02	{00004E2E-0000-0000-CE01-840CAB451D04}
22	PU2-01	{00004E2E-0000-0000-FB00-840CAB451D04}
23	VS-102	{00004E2E-0000-0000-9A00-86FDAA451D04}
24	Vertical Tank	{00004E30-0000-0000-C400-86FDAA451D04}
25	Pump-002	{00004E2E-0000-0000-6115-89159D452004}
26	Electrical Device	{00004E2E-0000-0000-8D15-89159D452004}
27	Pump-001	{00004E2E-0000-0000-CA14-89159D452004}
28	Door-101	{00004E2E-0000-0000-4102-EBA1AA451D04}
29	Windows-101	{00004E2E-0000-0000-B302-EBA1AA451D04}
30	Window-102	{00004E2E-0000-0000-F802-EBA1AA451D04}
31		
32		
33		
34		

Right Window (Piping Parts List):

	B	C
1		
2		
3		
4		
5		
6		
7		
8		
9	Nipple-0201	{00013885-0000-0000-000E-4599AB452004}
10	Pipe	{0001388C-0000-0000-010A-4599AB452004}
11	Flange-0311	{00013885-0000-0000-0206-4599AB452004}
12	Pipe	{0001388C-0000-0000-060B-4599AB452004}
13	Flange-0324	{00013885-0000-0000-060C-4599AB452004}
14	90 Degree Direction C	{00013885-0000-0000-060D-4599AB452004}
15	90 Degree Direction C	{00013885-0000-0000-0E01-4599AB452004}
16	90 Degree Direction C	{00013885-0000-0000-0E06-4599AB452004}
17	Flange-0308	{00013885-0000-0000-0F04-4599AB452004}
18	Concentric Size Chan	{00013885-0000-0000-1000-4599AB452004}
19	VG3-0303	{00013885-0000-0000-130C-4599AB452004}
20	Tee-0302	{00013885-0000-0000-160B-4599AB452004}
21	Pipe	{0001388C-0000-0000-1701-4599AB452004}
22	90 Degree Direction C	{00013885-0000-0000-170A-4599AB452004}
23	Pipe	{0001388C-0000-0000-170D-4599AB452004}
24	Pipe	{0001388C-0000-0000-1800-4599AB452004}
25	90 Degree Direction C	{00013885-0000-0000-1804-4599AB452004}
26	Flange-0332	{00013885-0000-0000-180F-4599AB452004}
27	VG333-0201	{00013885-0000-0000-190E-4599AB452004}
28	90 Degree Direction C	{00013885-0000-0000-1C06-4599AB452004}
29	90 Degree Direction C	{00013885-0000-0000-1D10-4599AB452004}
30	Flange-0301	{00013885-0000-0000-1E00-4599AB452004}
31	Pipe	{0001388C-0000-0000-1E0B-4599AB452004}
32	Pipe	{0001388C-0000-0000-1F0A-4599AB452004}
33	Pipe	{0001388C-0000-0000-2104-4599AB452004}
34	Flange-0325	{0001388E-0000-0000-230C-4599AB452004}

Metadata Browser/Data model

The instructor will present the MetaDataBrowser and discuss some of the information regarding the Data Model for SmartPlant 3D.

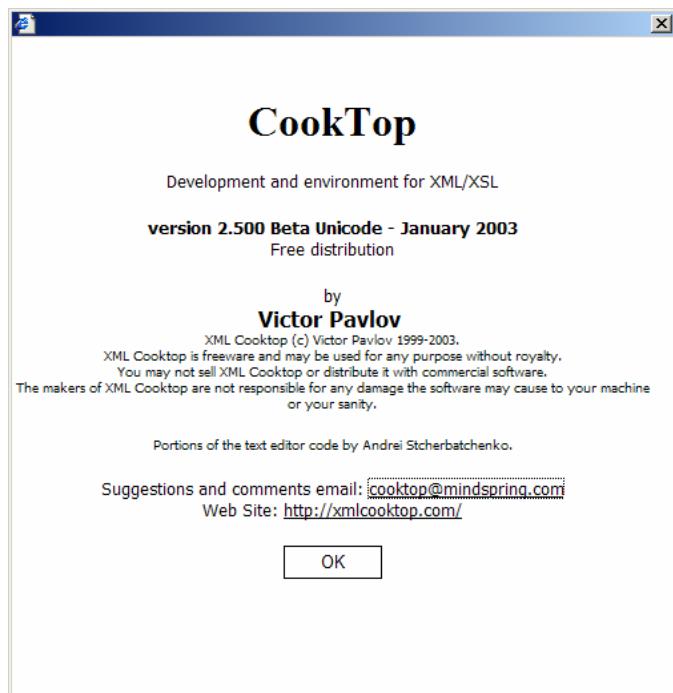
Intro to XML and Cooktop 2.5

XML Intro

The instructor will present SmartPlant 3D Reports Workshop – Intro to XML.ppt

CookTop 2.5

Many XML editors exist, but for the purpose of conformance in the class, we will be making use of Cooktop 2.5: a Free Distribution XML/XSL Development Environment copyrighted by Victor Pavlov. Please refer to the Help -> About within the CookTop program for legal information.



Locate the CookTop software shortcut on your desktop and double click it to launch.

The instructor will give provide a brief tour of CookTop's operation.

Understanding Report Templates

Report Definitions/Excel Data (Reports.xls)

You will find the Reports.xls on any machine which has had the reference data components installed as part of the project management options. To locate this Reports.xls, go to the installation directory for the product (for example, it may be C:\Program Files\SmartPlant\3D\WPM) and then locate the following directory: CatalogData\BulkLoad\DataFiles.

1. Open the Reports.xls file.

The Reports.xls file contains the following worksheets:

Revision History
Legend
Index
CustomInterfaces
CatalogRoot
Report
ClassNodeType
R-Hierarchy

2. Only the Report worksheet contains information not covered in the standard Reference Data class.

Recall:

The **CatalogRoot**, **ClassNodeType**, and **R-Hierarchy** sheets play a role in creating the folder structure in the Catalog Environment.

Revision History, **Legend** and **Index** are not processed by the Bulkload Utility, but are instead intended for user ease.

The **Custominterfaces** sheet allows for the defining of Interfaces and Properties/Attributes within the scope of SP3D Reference Data.

The Report Workbook is processed just as any other tabular Excel data in SmartPlant 3D: using the Bulkload Utility in any of its four specific Modes:

- 1) Create...
- 2) Append,
- 3) Add/Modify/Delete,
- 4) Delete and Replace.

The instructor will now cover the **Report** sheet and explain its structure and layout.

Valid Types:			Valid values for Types:	Description for valid values	Recomm
Undefined				1 Should not use	
Report Template				2 The Report master template. Contains references to other templates	rtp
Label Template				5 The Label master template (identical to a Report template). Exist for the purpose of classification	rtp
Report Executable				12 Not used for V4	rxe
Label Executable				15 Not used for V4	rxe
Report Execution Plan				22 Template used to persist the set of answers to parameters for a single instance of a report	rxp
Query Template				102 Describes user selections to build the SQL used to retrieve data.	rqe
Query Parameters Test				152 Describes the parameters required by the query.	rqp
Baseline Template				202 Describes the tools to get a baseline report, in the case of differential reports.	rbl
Baseline Parameters				252 Describes the parameters required by the baseline.	rpb
Formatting Template				302 Describes the tools used to format the reporting domain (result of the query/ies).	frm
Formatting Parameter				352 The parameters used in the formatting	rfp
Display Template				402 Describes the tools to display or publish the formatted result of a report	rdy
Display Parameters Test				452 Describe the parameters used by the display.	rdp
Type	Name	Description	Type	Description	
Start	REPORTS	Report Template	BASE TEMPLATES	Report Template	This is a blank report with no queries. It <Reports\Templates\Folder>\Reports\Base Templates\Report Template (No Query)
	Report Template (No Query)	Report Template	Report Template (Plant Query and SQL Bas Report Template	Report Template	This is a blank report with two example : <Reports\Templates\Folder>\Reports\Base Templates\Report Template (Plant Query and SQL Based Query
	COMPONENTS FOR REPORTS				
Baseline	Parameterized Baseline Selection	Baseline Template	Baseline Template	Select baseline by parameter1	<Reports\Templates\Folder>\Reports\Components for Reports\Baseline\SelectFileManual.rbl
	Display				

Valid Types and what these types represent

Head Name _____

REPORTS BASE TEMPI ATES

Report Template (No Q1

Report Template (Plant)

COMPONENTS FOR R

Baseline

Parameterized Baseline

Display

Salinity

10

100

10

100

Name of Object in

卷之三

三〇六

100

Type, Valid Types are listed above in the comment field, these are also listed as Code list values

Relative path to file based on <ReportsTemplateFolder> in \Symbols share. Software will often access these files when running a report (with the exception of reports saved only to the model (db))

Report Templates (*rtp)

Let us refer to the RTP we generated during our workflow labs.

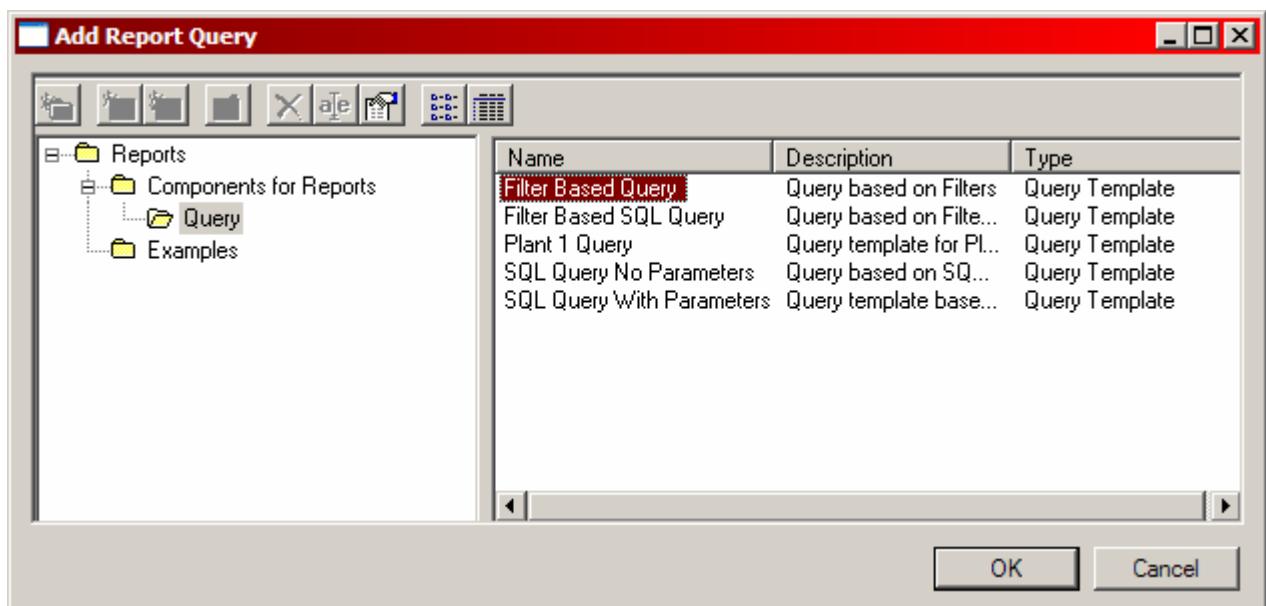
1. Locate on the Symbol Directory share the folder “<symbols>\Reports\Examples\Structure Creation Modification and Status Report”
2. Open the RTP in Cooktop.

```
<?xml version="1.0" encoding="UTF-8"?>
<REPORT_TEMPLATE Name="Report Template (No Query)" Description="Empty MetaTemplate for Reporting">
  <QUERIES>
    <QUERY Name="MyNewQuery" Site="User" Path="Structure Creation Modification and Status Report.rqe" />
  </QUERIES>
  <FORMATTING Name="Report Template (No Query)" site="User" Path="Structure Creation Modification and Status Report.rfm" />
  <DISPLAY Name="Report Template (No Query)" Site="User" Path="Structure Creation Modification and Status Report.rdy" />
</REPORT_TEMPLATE>
```

The *rtp identifies the other xml files that participate in this particular report template set.

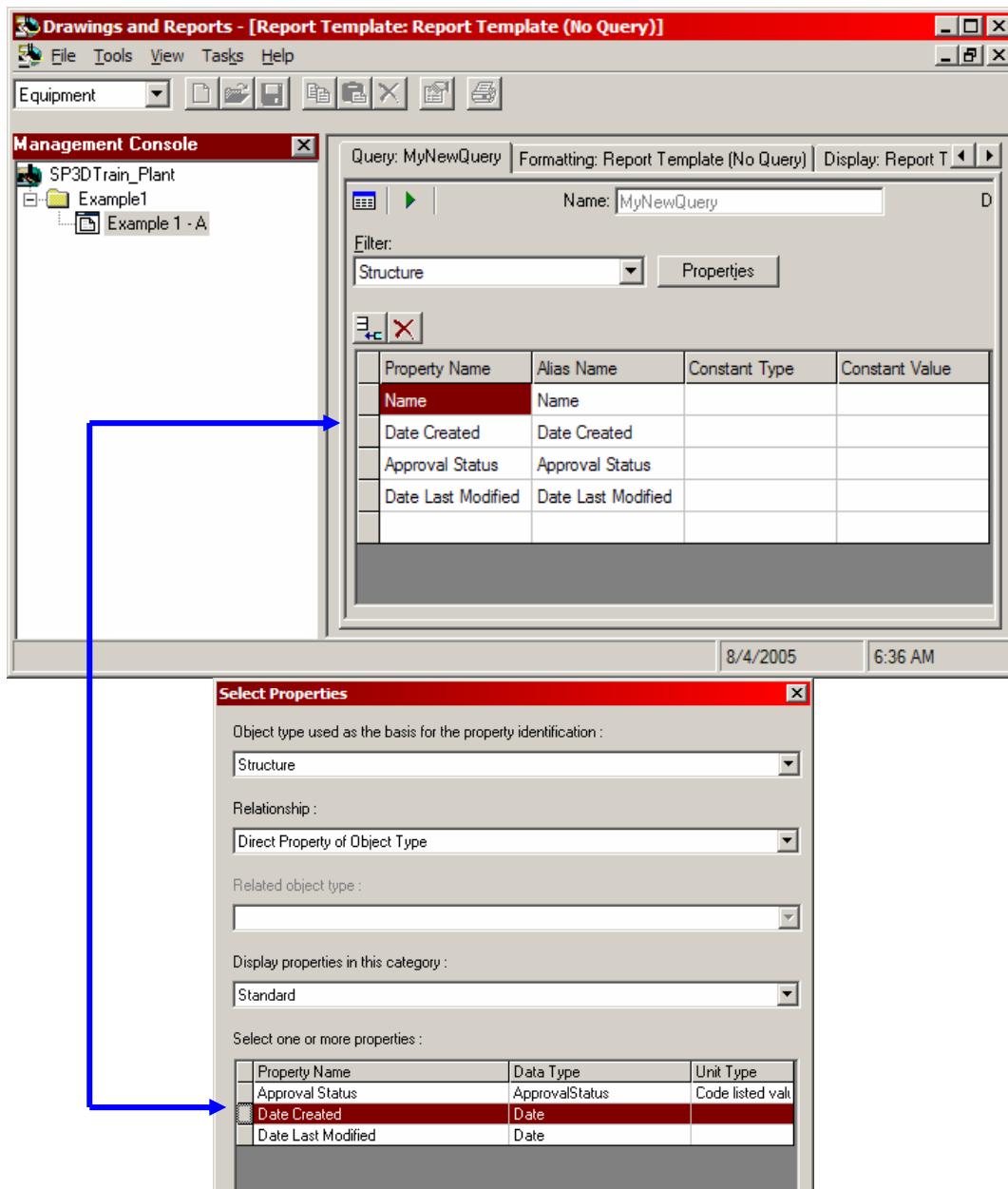
Query (*rqe)

The workflow to created a “Filter based Query” was exampled during the overall Workflow Section. Recall that while in the Drawings and Report Environment, you can Edit a Report Teamplate and then use the Tools → “Add Query” command to add a specific component query to meet your needs).



Filter based Query

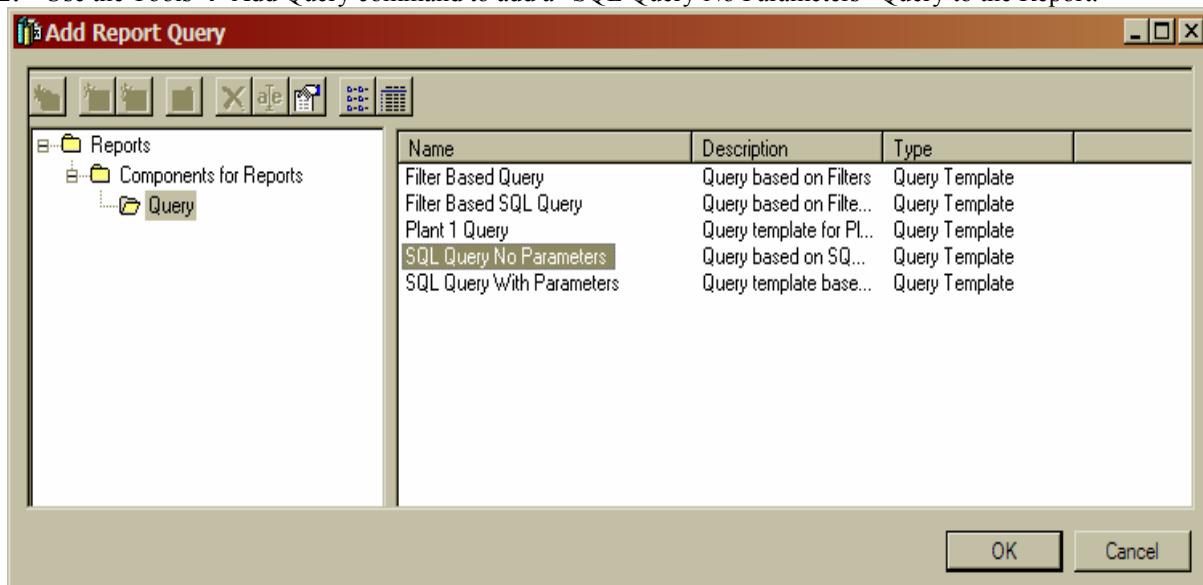
To make a Filter based Query through the GUI, pick the Filter Based Query item from the “Add Report Query” dialog. This will result in you being able to build queries based on the same mechanics you have available in the Filters property pages. Again, you should have already completed this earlier when we looked at the mechanics of save to Model, save to Catalog, and save to file. Please refer back to the creation of Example 1 – A.



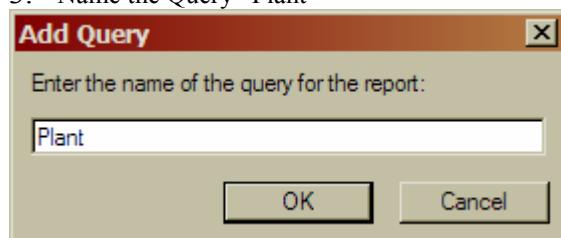
SQL based Query

The SQL based query selects the objects that will go into the report by executing an SQL statement against the Reports Database. We'll create a query that will give us the NPD, NPDUnits, and Approval Status of all PipeRuns from the model.

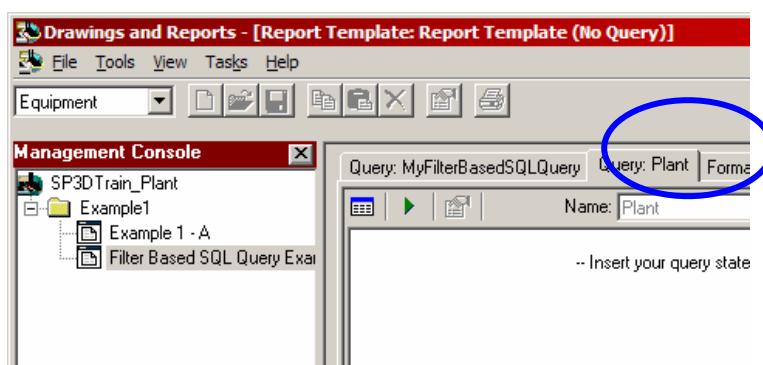
1. Create a new Report Component in the Drawings and Reports environment. Name it "SQL Query Example". [Recall how this operation and the following operations are done from the earlier labs.]
2. Use the Tools → Add Query command to add a "SQL Query No Parameters" Query to the Report.



3. Name the Query "Plant"



4. There should now be a new tab named "Query: Plant"



5. Input the following:

```
SELECT
pr.oid,
NPD,
NPDUnitType,
ShortStringValue
FROM JRtePipeRun pr
JOIN JDOObject o on o.oid = pr.oid
JOIN CL_ApprovalStatus aps on aps.ValueID = o.ApprovalStatus
```

6. Test the query.

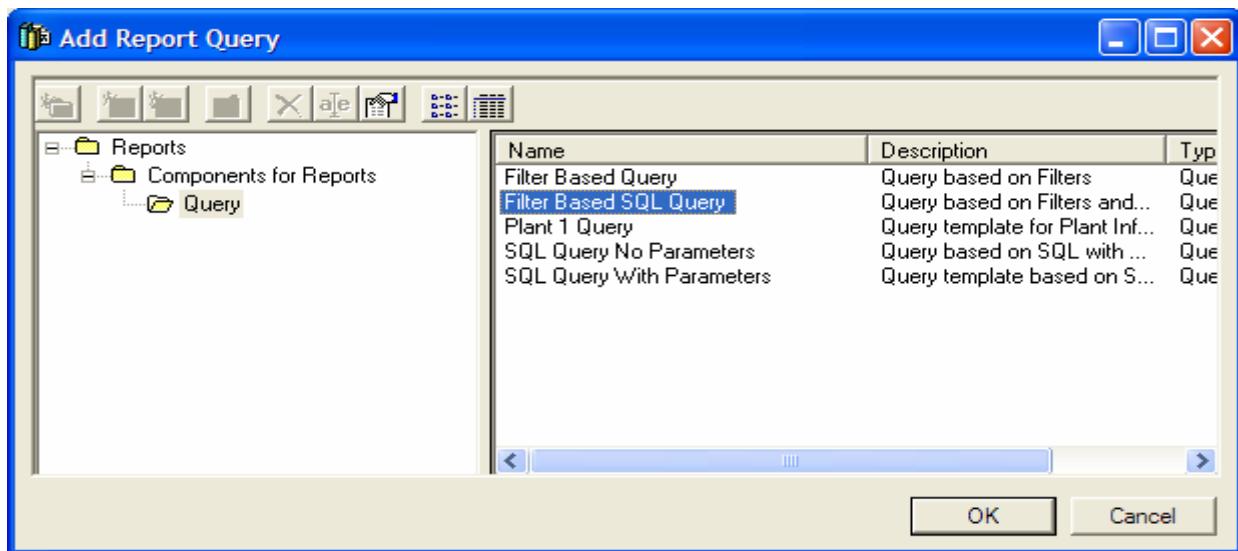
7. Remember to save the addition of this query to the template so as not to lose work.

Filter based SQL Query

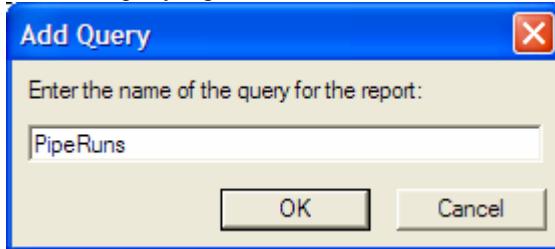
The Filter based SQL query selects the objects that will go into the report by executing an SQL statement NOT on the whole Reports DB, but only on the collection of objects returned from a filter. This means we need to supply both a filter and SQL statement.

Let's create a report that will return the NPD, NPDUnits, and Approval Status of all PipeRuns under a given system. One of the ways to achieve this is to create an asking filter where the user can select a system, and then run the SQL statement from the previous example.

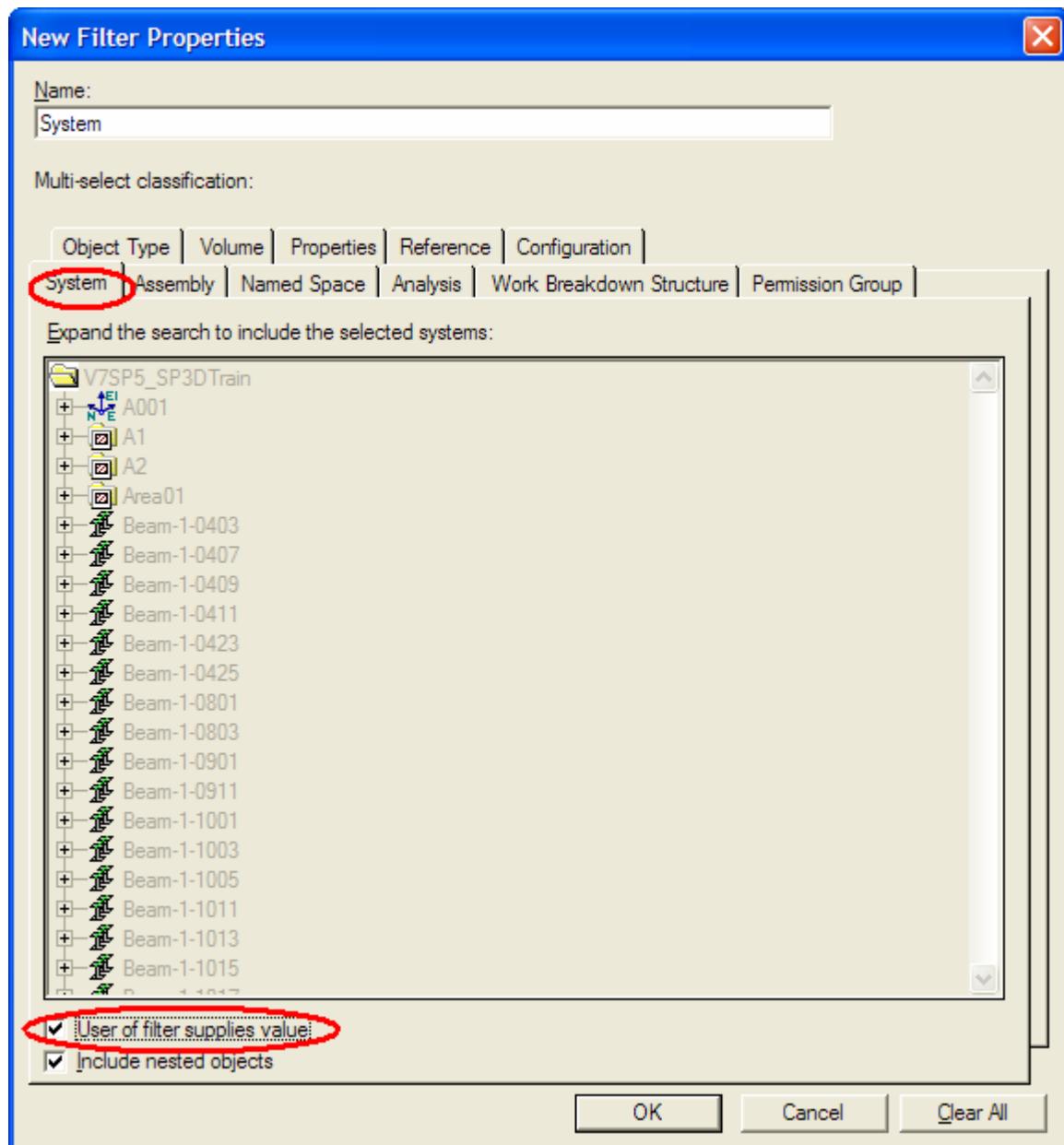
1. Create a new Report Component in the Drawings and Reports environment. Name it "Filter based SQL Query Example" and edit its template
2. To do a Filter Based SQL Query, from "Tools/Add Query" pick the "Filter Based SQL Query".



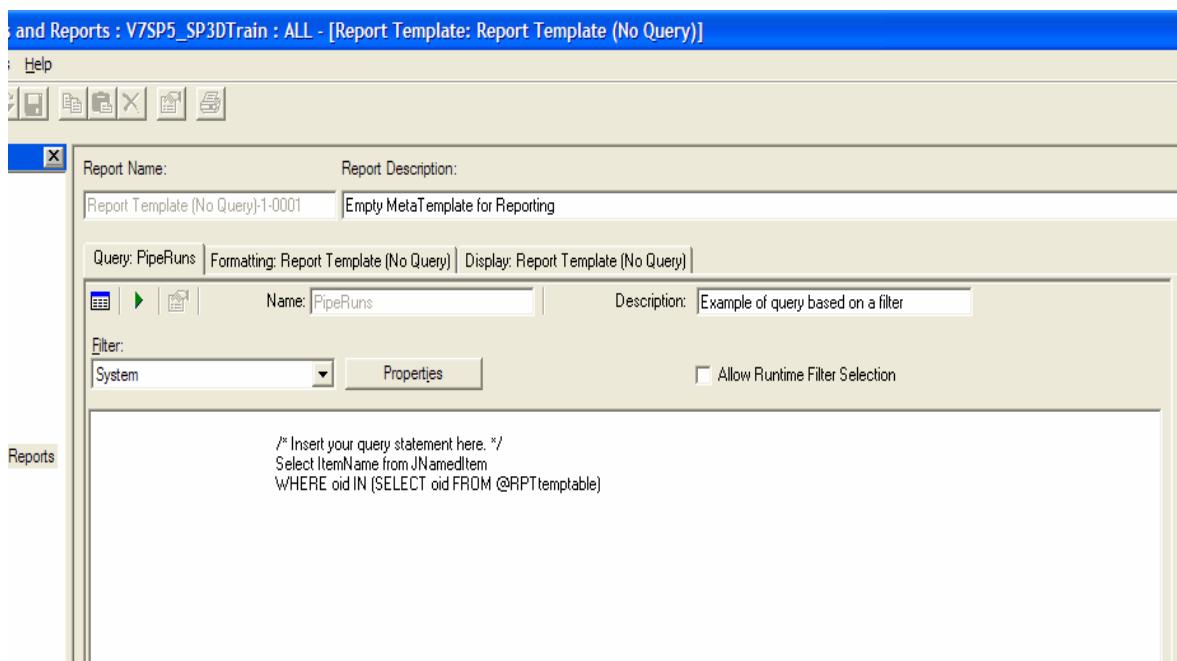
3. Name the query PipeRuns



4. In the "Select Filter" dialog create a new Catalog filter by selecting the "System" tab and clicking the "User of filter supplies value" check box. Make sure "Include nested objects" is also checked (this is the default). This creates a filter which will prompt the user to select a system and will then return all its system children.



5. Select that filter in the “Select Filter” dialog. You’ll see the following pane with a default SQL statement:



6. Change the SQL statement to:

```

SELECT
pr.oid,
NPD,
NPDUnitType,
ShortStringValue
FROM JRtePipeRun pr
JOIN JDObject o on o.oid = pr.oid
JOIN CL_ApprovalStatus aps on aps.ValueID = o.ApprovalStatus
WHERE pr.oid IN (SELECT oid FROM @RPTtemptable)

```

Note that the oid in the WHERE clause has to refer to the JRtePipeRun (alias "pr")

7. Test the report (). You will be prompted to select a system first, and then you'll see the results for only the pipe runs under that system.

Reports : V7SP5_SP3DTrain : ALL - [Report Template: Report Template (No Query)]

Report Name: Report Description:
Report Template (No Query)-1-0001 Empty MetaTemplate for Reporting

Query: PipeRuns | Formatting: Report Template (No Query) | Display: Report Template (No Query) |

Name: PipeRuns | Description: Example of query based on a filter

Filter: System Properties Allow Runtime Filter Selection

```

SELECT
pr.oid,
NPD,
NPDUntType,
ShortStringValue
FROM JRtePipeRun pr
JOIN JDObject o on o.oid = pr.oid
JOIN CL_ApprovalStatus aps on aps.ValueID = o.ApprovalStatus
WHERE pr.oid IN (SELECT oid FROM @RPTtemptable)

```

oid	NPD	NPDUntType	ShortStringValue
(0001388D-0000-0000-14	in	Working	
(0001388D-0000-0000-16	in	Working	
(0001388D-0000-0000-14	in	Working	
(0001388D-0000-0000-16	in	Working	
(0001388D-0000-0000-14	in	Working	
(0001388D-0000-0000-0.25	in	Working	

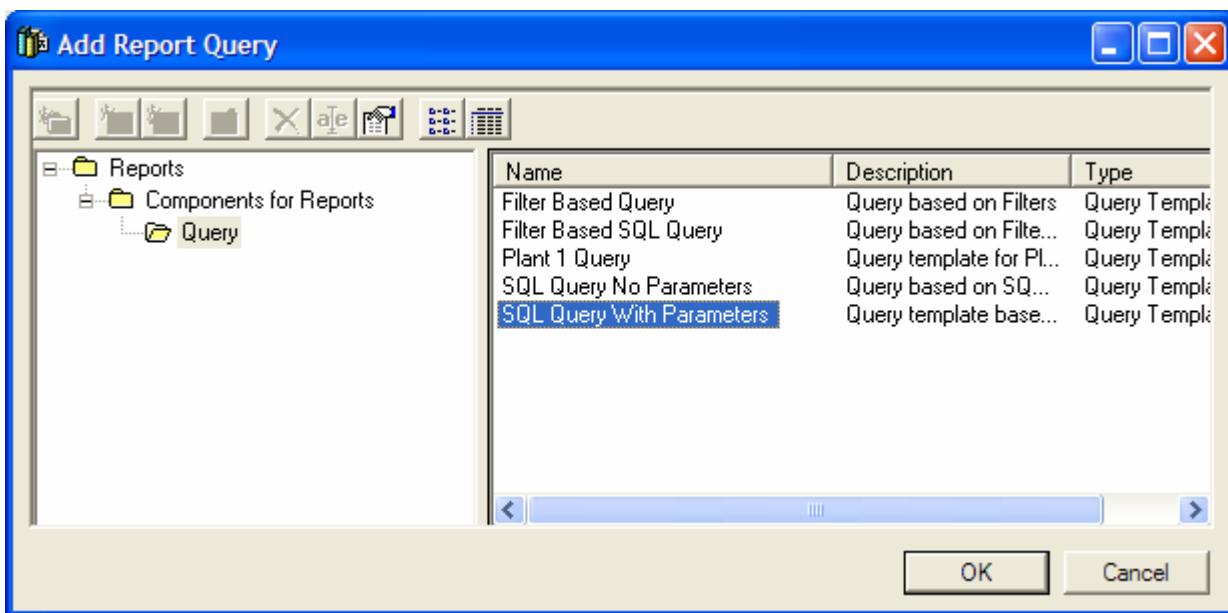
- Remember to always Save the changes to the Template in order to not lose work.

SQL Query With Parameters

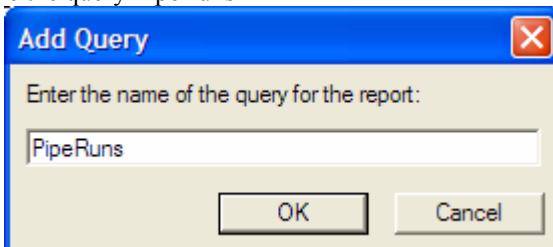
The SQL with parameters selects the objects that will go into the report by executing a parameterized SQL statement against the Reports Database – i.e. the user can change part of the query at run time.

Let's create a query that will give us the NPD, NPDUnts, and Approval Status of all PipeRuns from the model that have a given approval status. We'll let the user determine what that approval status should be at runtime.

- Create a new Report Component in the Drawings and Reports environment. Name it “SQL Query With Parameters Example”.
[Recall how this operation and the following operations are done from the earlier labs.]
- Use the Tools → Add Query command to add a “SQL Query With Parameters” Query to the Report.



3. Name the query PipeRuns



4. You should see the following:

Report Name: SQL Report Description: Empty MetaTemplate for Reporting

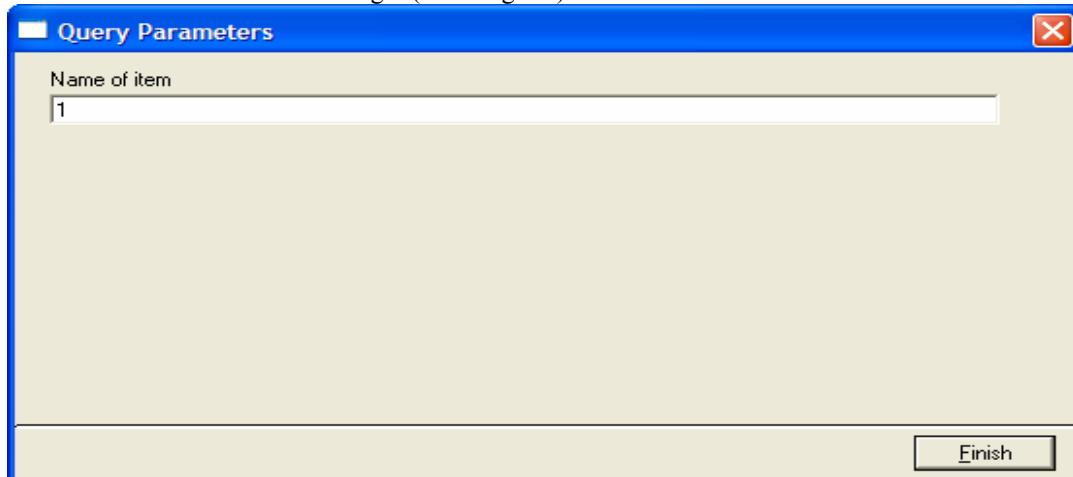
Query: PipeRuns Formatting: Report Template (No Query) Display: Report Template (No Query)

```
-- Insert your query statement here.
-- Your query can contain question mark place holders for parameters
SELECT j.oid, n.itemname as Name
FROM jobject j
join jnameditem n on n.oid = j.oid
WHERE n.itemname = ?
```

5. Input the following:

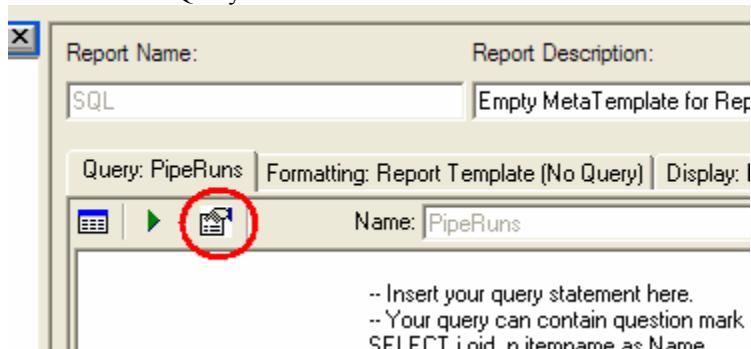
```
SELECT  
pr.oid,  
NPD,  
NPDUIType,  
ShortStringValue  
FROM JRtePipeRun pr  
JOIN JDOBJECT o on o.oid = pr.oid  
JOIN CL_ApprovalStatus aps on aps.ValueID = o.ApprovalStatus  
WHERE o.ApprovalStatus = ?
```

6. Since Approval Status is a code-listed value if we test the query now, we'll need to input the particular Approval Status we are interested in as an integer (Working = 1)

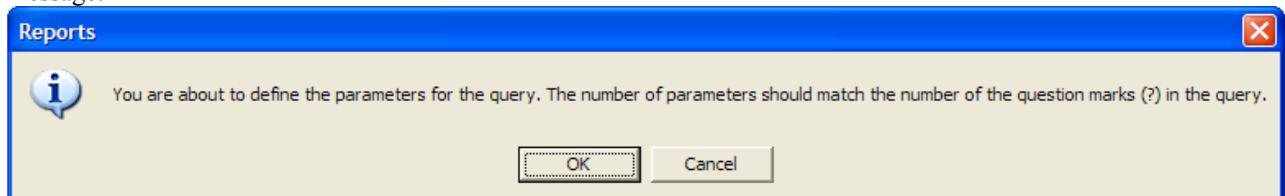


This is not particularly convenient, plus we want a more meaningful prompt.

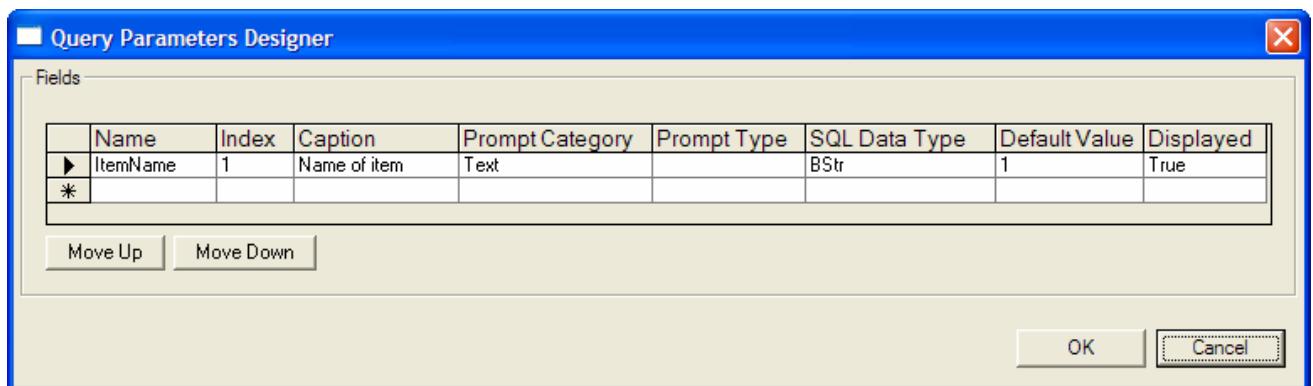
7. Click the “Edit Query Parameters” button:



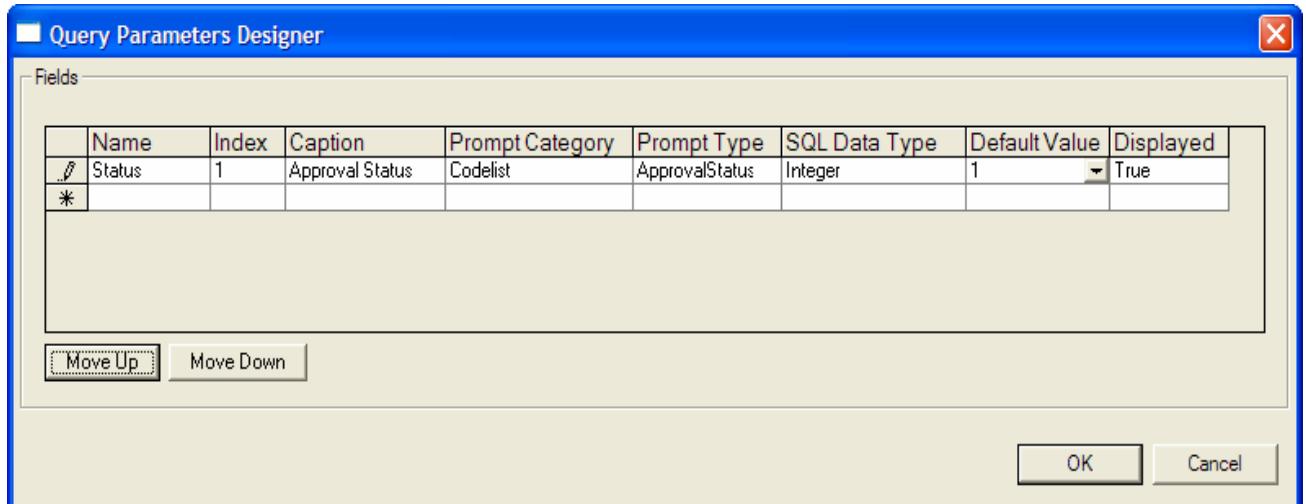
8. Click “OK” on the following warning message:



9. You will see the following dialog (you may need to resize it to see all fields at the same time)

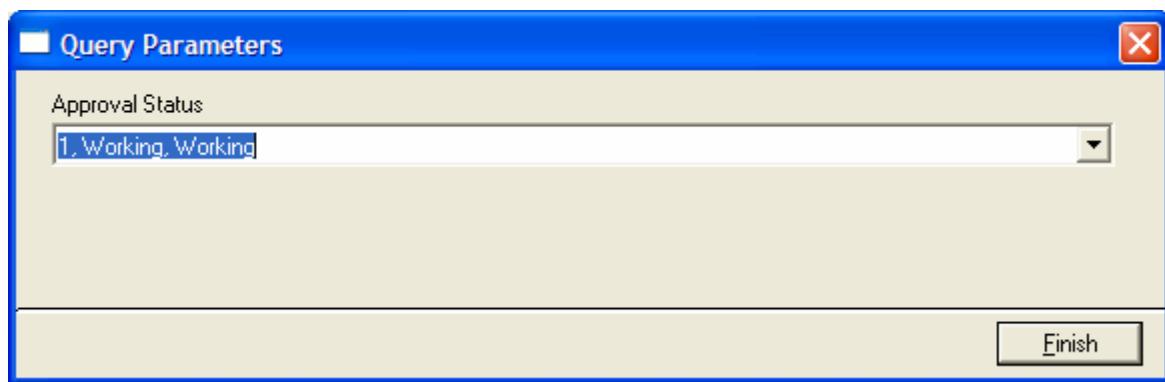


10. Fill in the table as follows:



Name -	internal name. You can give any name – the user will not see it.
Index -	used only if 2 parameters have the same Name.
Caption –	This is the prompt that the user will see
Prompt Category -	Select “Codelist” from the drop down combo since we are working with a code list
Prompt Type –	in our case this will be the particular codelist – ApprovalStatus
SQL Data Type –	this will be automatically populated with “integer” for codelists
Default Value –	the value that will be displayed to the user initially
Displayed –	should be always true if the user is to be able to select that parameter

11. Test the report – you’ll be presented with this prompt to select the query parameter Approval Status:



12. Selecting Working will give you only the pipe runs in the “Working” Approval Status.

oid	NPD	NPDUntType	ShortStringValue
{0001388D-0000-0000-14}	in		Working
{0001388D-0000-0000-16}	in		Working
{0001388D-0000-0000-14}	in		Working
{0001388D-0000-0000-16}	in		Working
{0001388D-0000-0000-14}	in		Working
{0001388D-0000-0000-11}	in		Working

13. Remember to save the addition of this query to the template so as not to lose work.

Looking at the XMLs:

1. Open the Symbol Share directory and go to the “<symbols>\Reports\Examples\MyReport1A with Queries” folder
2. Open the .rqe file in Cooktop
3. Use the Cooktop command XML → Format XML from the horizontal menu to make the information more legible.
4. You will be presented with something similar to this:

```
<?xml version="1.0" encoding="UTF-8"?>
<REPORT_QUERY Name="MyNewQuery" Description="Example of query based on a filter" RequiresFilter="Yes">
  <DESIGN_TIME Progid="SP3DReportsQueryBuilder.SFBQuery" Action="" Arg="" />
  <RUN_TIME Progid="SP3DRuntimeQuery.CQueryInterpreter" Action="" Arg="" />
  <FILTER FilterName="{0000272E-0000-0000-0300-D1758F422304}|model" RequiresBinding="No" />
  <RETURNED_PROPERTIES>
    <RETURNED_PROPERTY Name="Name" SQLType="BStr">
      <PATHS>
        <PATH SourceType="*" DestinationInterface="IJNamedItem" DestinationProperty="Name">
          <STROKES />
        </PATH>
      </PATHS>
    </RETURNED_PROPERTY>
  </RETURNED_PROPERTIES>
</REPORT_QUERY>
```

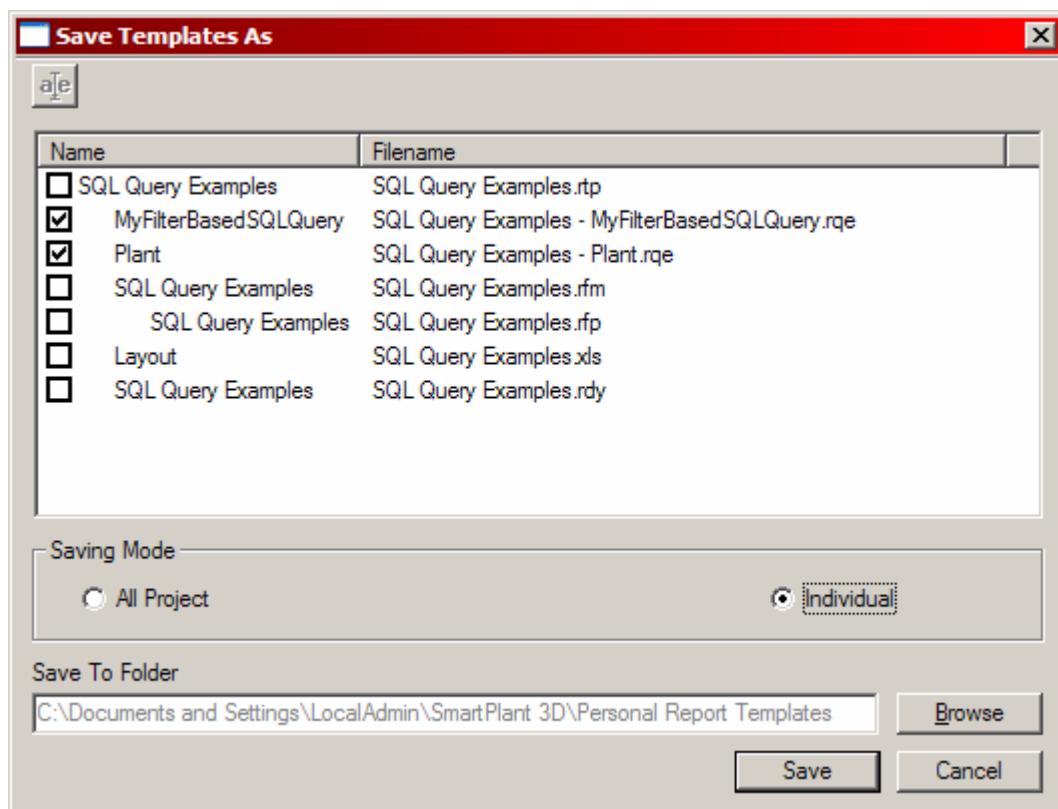
5. Recall that when we made this query, there was only one item being queried, that being the Name from IJNamedItem.
6. Change to the folder “<symbols>\Reports\Examples\Structure Creation Modification and Status Report”
7. Open the rqe from this folder in Cooktop:

```

<?xml version="1.0" encoding="UTF-8"?>
<REPORT_QUERY Name="MyNewQuery" Description="Example of query based on a filter" RequiresFilter="Yes">
    <DESIGN_TIME Progid="SP3DReportsQueryBuilder.SFBQuery" Action="" Arg="" />
    <RUN_TIME Progid="SP3DRuntimeQuery.CQueryInterpreter" Action="" Arg="" />
    <FILTER FilterName="Structure" RequiresBinding="No" />
    <RETURNED_PROPERTIES>
        <RETURNED_PROPERTY Name="Name" SQLType="BStr">
            <PATHS>
                <PATHSourceType="*" DestinationInterface="IJNamedItem" DestinationProperty="Name">
                    <STROKES />
                </PATH>
            </PATHS>
        </RETURNED_PROPERTY>
        <RETURNED_PROPERTY Name="Date Created" SQLType="DBTimeStamp">
            <PATHS>
                <PATHSourceType="*" DestinationInterface="IJDBObject" DestinationProperty="DateCreated">
                    <STROKES />
                </PATH>
            </PATHS>
        </RETURNED_PROPERTY>
        <RETURNED_PROPERTY Name="Approval status" SQLType="BStr">
            <PATHS>
                <PATHSourceType="*" DestinationInterface="IJDBObject" DestinationProperty="ApprovalStatus">
                    <STROKES />
                </PATH>
            </PATHS>
        </RETURNED_PROPERTY>
        <RETURNED_PROPERTY Name="Date Last Modified" SQLType="DBTimeStamp">
            <PATHS>
                <PATHSourceType="*" DestinationInterface="IJDBObject" DestinationProperty="DateLastModified">
                    <STROKES />
                </PATH>
            </PATHS>
        </RETURNED_PROPERTY>
    </RETURNED_PROPERTIES>
</REPORT_QUERY>

```

8. Notice how that you can quickly identify the FilterName in play (that being “Structure”) and each property being returned (Name, Date Created, Approval Status, Date Last Modified).
9. Use “File Save Report Template as” to just save the Plant SQLQueryNoParam.rqe file and the FilterBasedSQLQuery.rqe file.



10. Open the rqe using cooktop.

11. Notice how the SQL Query is wrapped in the SQLQueryNoParam.rqe:

```
<?xml version="1.0" encoding="UTF-8"?>
<REPORT_QUERY Name="Plant" Description="Example of non parameterized SQL query" RequiresFilter="No">
  <DESIGN_TIME Progid="SP3DReportsQueryBuilder.SQLQuery" Action="" Arg="" />
  <RUN_TIME Progid="SP3DRuntimeQuery.CQueryInterpreter" Action="" Arg="" />
  <SQL Timeout="900">
    <![CDATA[
      SELECT jcpr.oid,
             jni.ItemName AS Name,
             juap.Owner AS Owner,
             juap.Site AS Site
        FROM JNamedItem jni
       JOIN JConfigProjectRoot jcpr ON (jcpr.oid = jni.oid)
       JOIN XConfigurableProjectRoot r1 ON r1.oidorigin = jcpr.oid
       JOIN JDUAProjectRoot juap ON juap.oid = r1.oiddestination
      OPTION (MAXDOP 1)
    ]]>
  </SQL>
</REPORT_QUERY>
```

12. Notice how the SQL Query is wrapped in the FilterBasedSQLQuery.rqe:

```

<?xml version="1.0" encoding="UTF-8"?>
<REPORT_QUERY Name="MyFilterBasedSQLQuery" Description="Example of query based on a filter" RequiresFilter="Yes">
  <DESIGN_TIME Progid="SP3DReportsQueryBuilder.FSQLQuery" Action="" Arg="" />
  <RUN_TIME Progid="SP3DRuntimeQuery.CQueryInterpreter" Action="" Arg="" />
  <FILTER FilterName="{0000272E-0000-0000-0300-D1758F422304}|model" RequiresBinding="No" />
  <SQL Timeout="0">
    <! [CDATA[SELECT eqp.oid,
      jnim.ItemName AS 'Equipment',
      eqp.o0 AS X,
      eqp.o1 AS Y,
      eqp.o2 AS Z,
      gn.Text AS 'Description',
      cas.LongStringValue AS 'Status',
      jpg.name AS 'Project'
    FROM JEquipment eqp
    JOIN JNamedItem jnim ON (eqp.oid = jnim.oid)
    LEFT JOIN (
      SELECT rcn.oidorigin, jgn.Text
      FROM XContainsNote rcn
      JOIN JGeneralNote jgn ON (jgn.oid = rcn.oiddestination)
    ) AS gn ON (gn.oidorigin = eqp.oid)
    JOIN JDObject jobj ON (eqp.oid = jobj.oid)
    JOIN CL_ApprovalStatus cas ON (cas.valueid = jobj.Approvalstatus)
    JOIN JDPermissiongroup jpg ON (jpg.PermissionGroupID = jobj.ConditionID)
    WHERE eqp.oid IN (SELECT oid FROM @RPTtemptable)
    OPTION (MAXDOP 1)
  ]]>
  </SQL>
</REPORT_QUERY>

```

13. Note how the two are very similar in terms of structure, resultant, and wrapper handling.

Query Parameters (*.rqp)

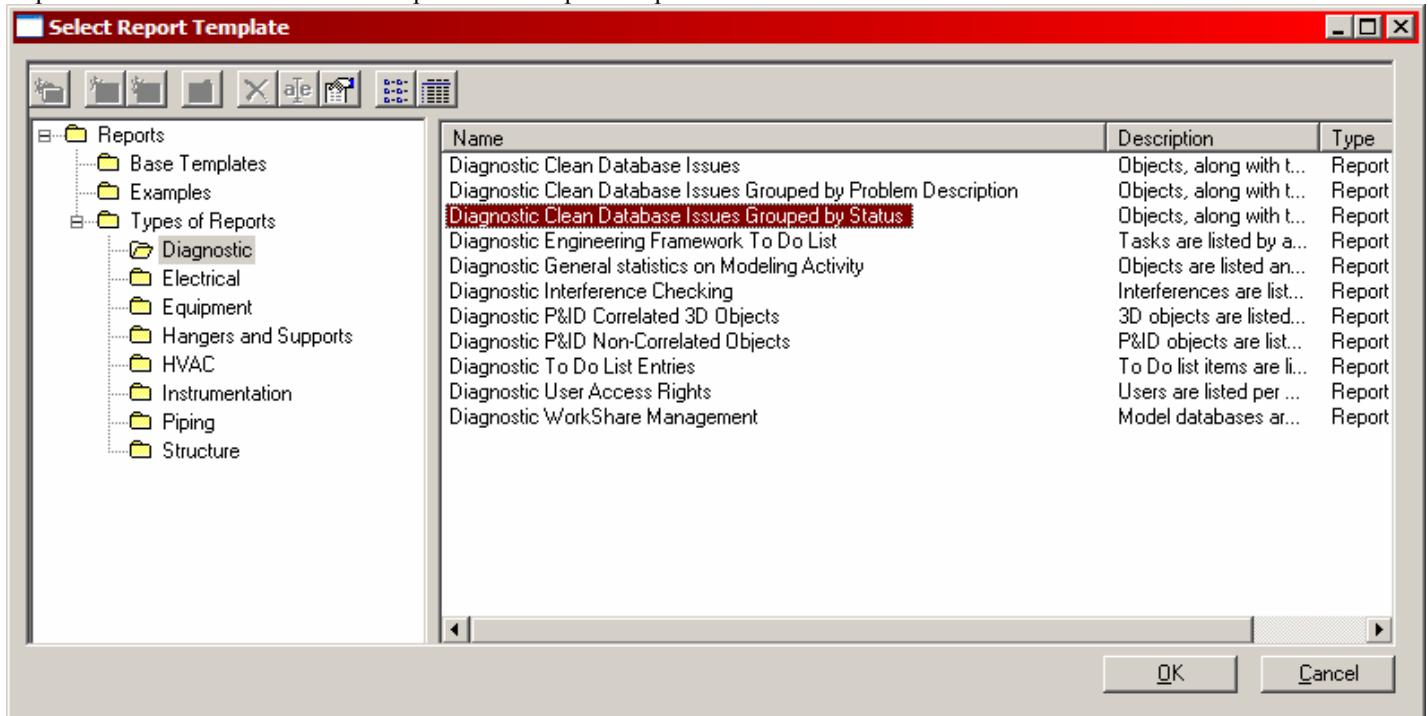
Let us take a look at one the delivered .rqp files. This one is the Database Corruption Issues Grouped by Status.rqp file that is found in <symbols>\Reports\Types of Reports\Diagnostic\Database Corruption Issues Grouped by Status.

```

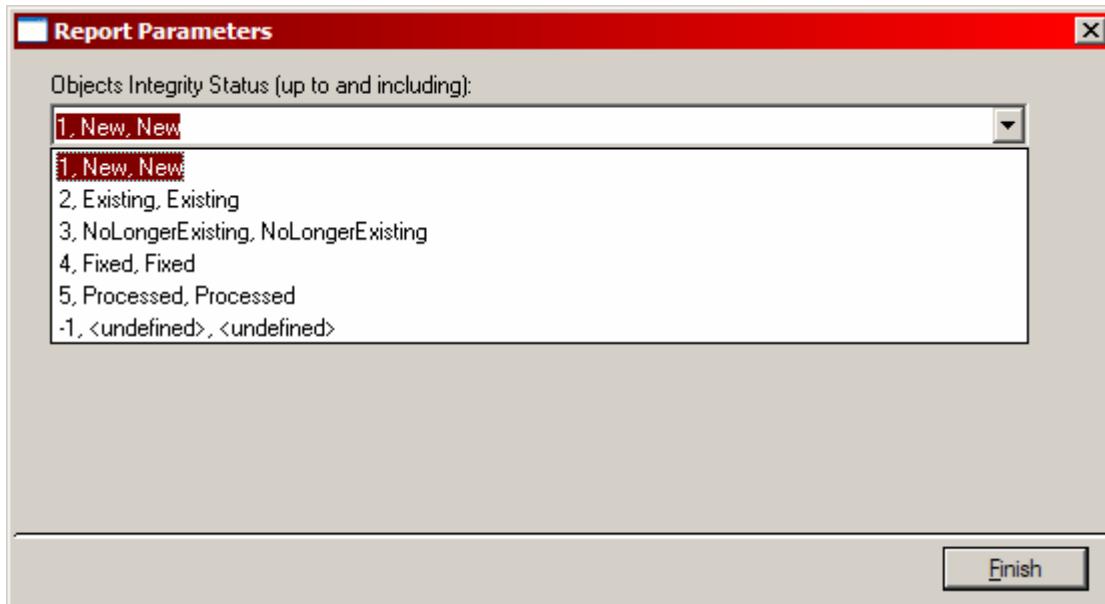
SOURCE(XML) | xpath console | stylesheets(xsl) | result | result(xml) |
<?xml version="1.0" encoding="windows-1252" ?>
<QUERY_PARAMETERS
  Name="Diagnostic Database Corruption Issues Grouped by Status"
  Description="Example of Parameters for DBIntegrity">
  <DESIGN_TIME
    Progid=""
    Action=""
    Arg="" />
  <RUN_TIME
    Progid="SP3DPrompts.ct1Tab"
    Action=""
    Arg="" />
  <FIELDS>
    <SQL
      DataType="objectIntegritystatus"
      LookUp="CodeList"
      Name="Integrity Status"
      Caption="Objects Integrity status (up to and including):"
      DefaultValue="2"
      SQLType="INTEGER" />
  </FIELDS>
  <!-- Possible values for SQLType are:
  Array,BigInt,Binary,Boolean,BSTR,Chapter,Char,Currency,Date,DBDate,DBTime,DBTimeStamp,
  Decimal,Double,Empty>Error,FileTime,GUID,Dispatch, Integer,IUnknown,LongVarBinary,Longvarchar,
  LongVarWChar,Numeric,PropVariant,Single,SmallInt,TinyInt,UnsignedBigInt,UnsignedInt,UnsignedSmall,
  Int,UnsignedTinyInt,UserDefined,VarBinary,VarChar,Variant,VarNumeric,VarwChar,wchar
  -->
</QUERY_PARAMETERS>

```

Go and add this “Diagnostic Clean Database Issues Grouped by Status” onto a Spreadsheet Report component in the Drawings and Report Environment. Start Edit Template on the report template.



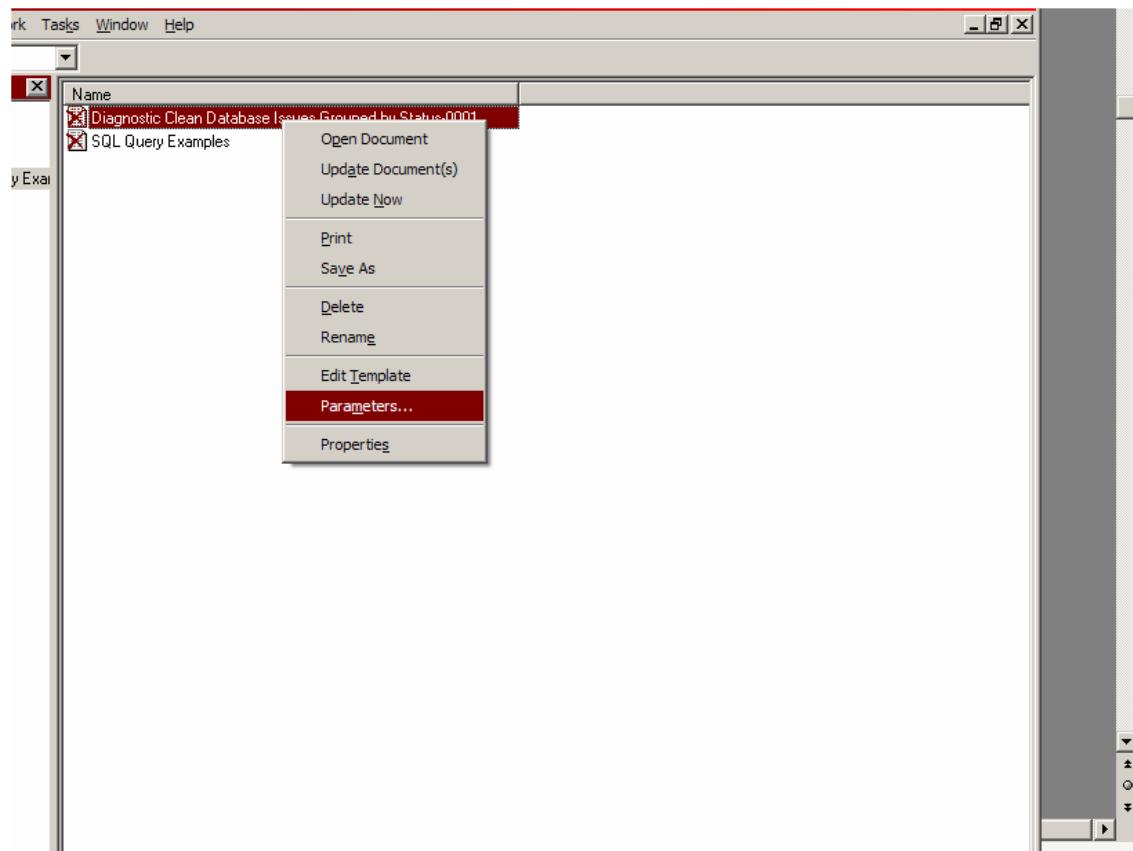
During the addition of the template you will be prompted with the following dialog:



The display of this form, the drop down, and the contents of the drop down are determined by the .rqp file.

You can pick an arbitrary value from the list and hit finish.

Also note, that if you later wish to access this again, you have the option of accessing it through the Drawings and Report Environment by right mouse clicking on the report, and selecting Parameters...



Formatting (*.rfm)

Open the MyReport1A.rfm with CookTop. The file is located at <symbols>\Reports\Examples\MyReport1A on the symbols share.

```
<?xml version="1.0" encoding="UTF-8"?>
<REPORT_FORMATTING Name="Report Template (No Query)" Description="Empty Format Definition">
  <DESIGN_TIME Progid="SP3DReportFormatDesigner.ExcelReport"
    Action="" Arg="" />

  <RUN_TIME Progid="SP3DEXCELFormat.SP3DReport"
    Action="DoFormatting" Arg="False" />

  <FORMATTING_PARAMETERS
    Name="Report Template (No Query)"
    Site="User"
    Path="MyReport1A.rfp" />

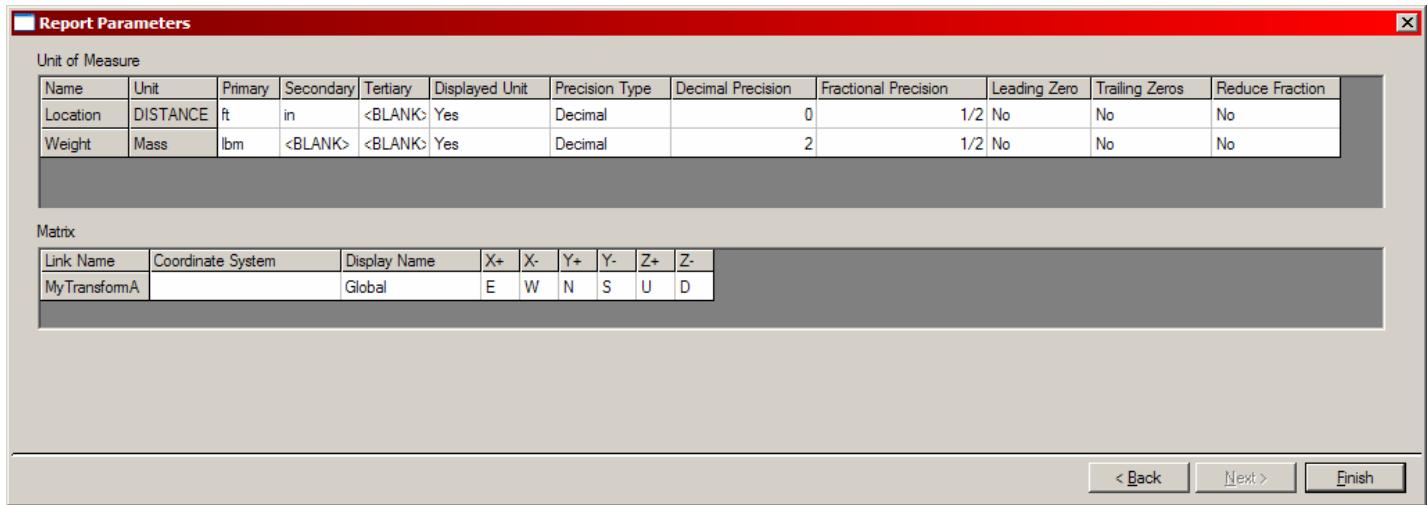
  <LAYOUT_TEMPLATE Type="External"
    Site="User" Path="MyReport1A.xls" />

</REPORT_FORMATTING>
```

Note above the identification of the .xls for the Layout Template, and the Formatting Parameters file (rfp).

Coordinate Systems

When working with the Equipment MTO report, a parameters form appears with the following inputs for Control:



The top table area controls UOM (the XML for UOM is discussed immediately after this section). The bottom table area controls the Coordinate System. The XML controlling this lower table area (named Matrix on the form) is in the Material Take-Off.rtp (located at "<symbols>\Reports\Types of Reports\Equipment\Material Take-Off").

```
        PrecisionType="Decimal"
        DecimalPrecision="2"
        FractionalPrecision=""
        LeadingZero=""
        TrailingZeros=""
        ReduceFraction="" />
    </UOMS>

<MATRIXES>
    <MATRIX
        LinkName="MyTransformA"
        DisplayName="Global"
        ModelName=""
        CSType="0"
        PosX="E "
        PosY="N "
        PosZ="U "
        NegX="W "
        NegY="S "
        NegZ="D "
        XYReferenceAxis="1"
        CanInherit="No" />
    </MATRIXES>
</FORMATTING_PARAMETERS>
```

Form default parameters
for Coordinate System

Display Units & Precision

If we look at the rfp (this example is taken from the “Member Weight and Length Sorted by Section Type.rfp” in the symbol share under <symbols>\Reports\Types of Reports\Structure\Member Weight and Length Sorted by Section Type\ we can see how this is captured.

```

<?xml version="1.0" encoding="UTF-8"?>
<FORMATTING_PARAMETERS
  Name="Structure Member weight and Length Sorted by Section Type"
  Description="Example of Parameters">
  <DESIGN_TIME
    Progid=""
    Action=""
    Arg="" />

  <RUN_TIME
    Progid="SP3DPrompts.ctlTab"
    Action=""
    Arg="" />

  <FIELDS>
  </FIELDS>

  <UOMS>
    <UOM
      Name="Length"
      CanInherit="No"
      Type="DISTANCE"
      Primary="ft"
      Secondary="in"
      Tertiary=""
      UnitsDisplayed="yes"
      PrecisionType="Fractional"
      DecimalPrecision="0"
      FractionalPrecision="1/32"
      Leadingzero=""
      Trailingzeros=""
      ReduceFraction="Yes" />

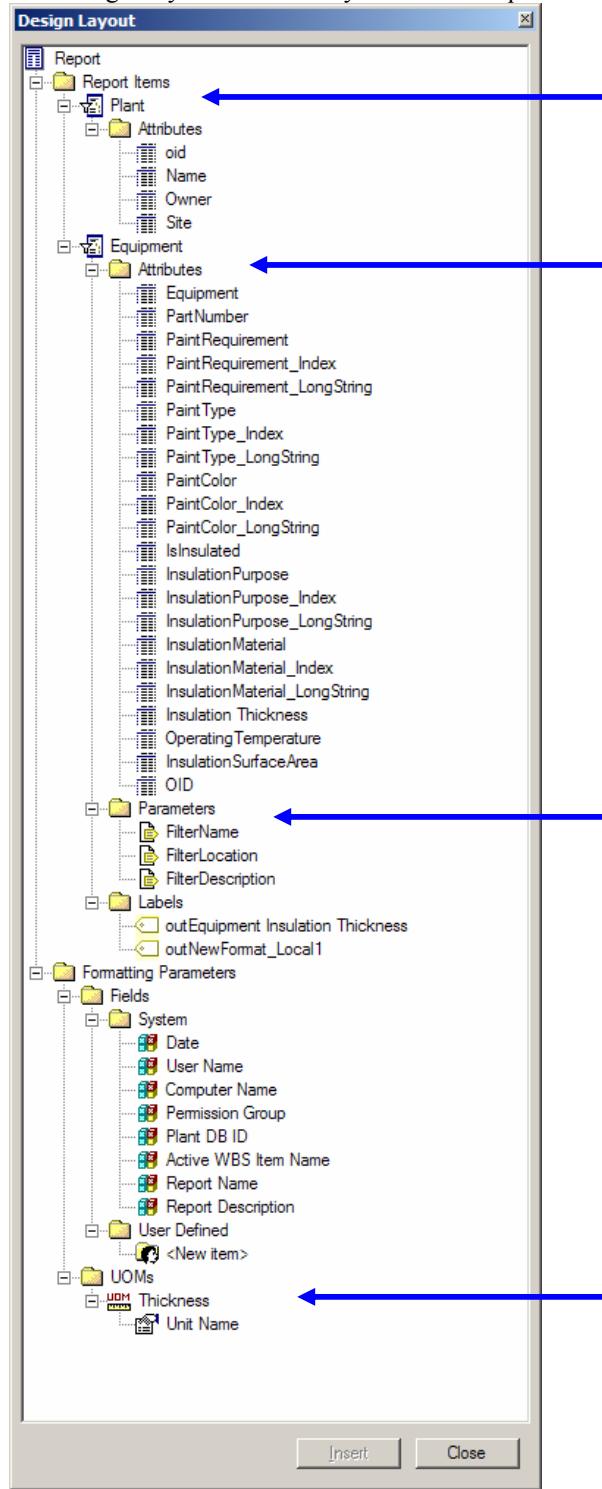
    <UOM
      Name="Weight"
      CanInherit="No"
      Type="Mass"
      Primary="lbf"
      Secondary=""
      Tertiary=""
      UnitsDisplayed="no"
      PrecisionType="Decimal"
      DecimalPrecision="2"
      FractionalPrecision=""
      LeadingZero=""
      TrailingZeros=""
      ReduceFraction="" />
  </UOMS>
</FORMATTING_PARAMETERS>

```

Precision Controls

Layout (*.xls)

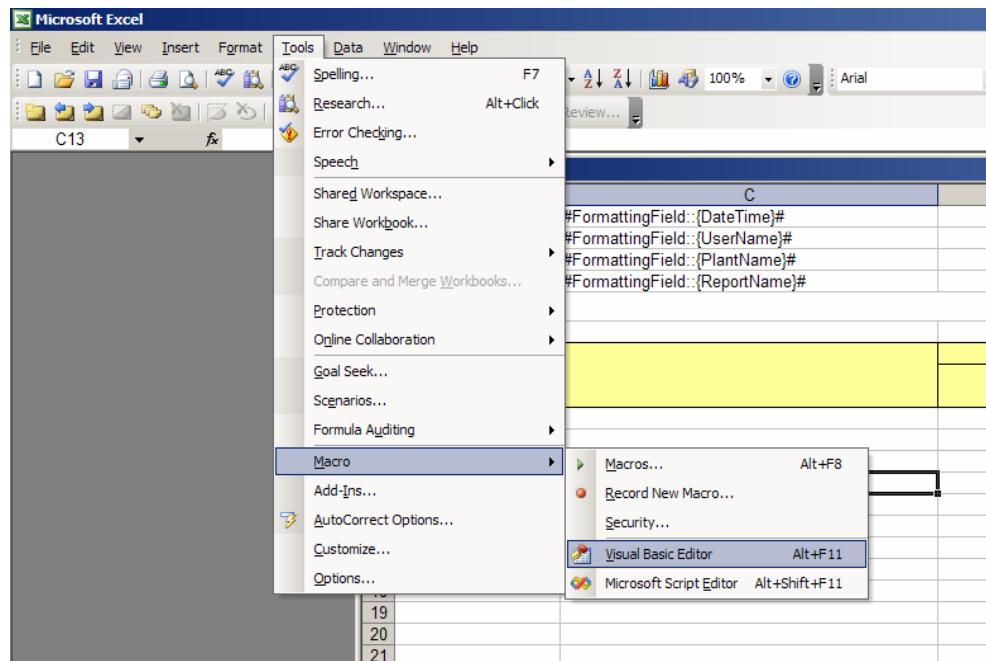
We've already seen how the *.xls can be edited in the GUI through drag-and-drop workflows. The Design Layout Browser is your means to quick editing of the layout.xls.



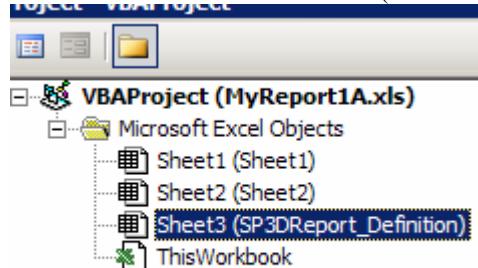
With the Design Layout browser, you can access the queries that you have defined on the template and the predefined System queries. Additionally, you can gain access to the management of labels, grouping, sorting etc.

You can also find the *.xls file in the same symbol share directory as all the XML files pertaining to one report template. Additionally, if you open the XLS (either through the GUI or directly from disk) you can access a hidden sheet:

1. Open the XLS, and from the Tools->Macro menu, select Visual Basic Editor.



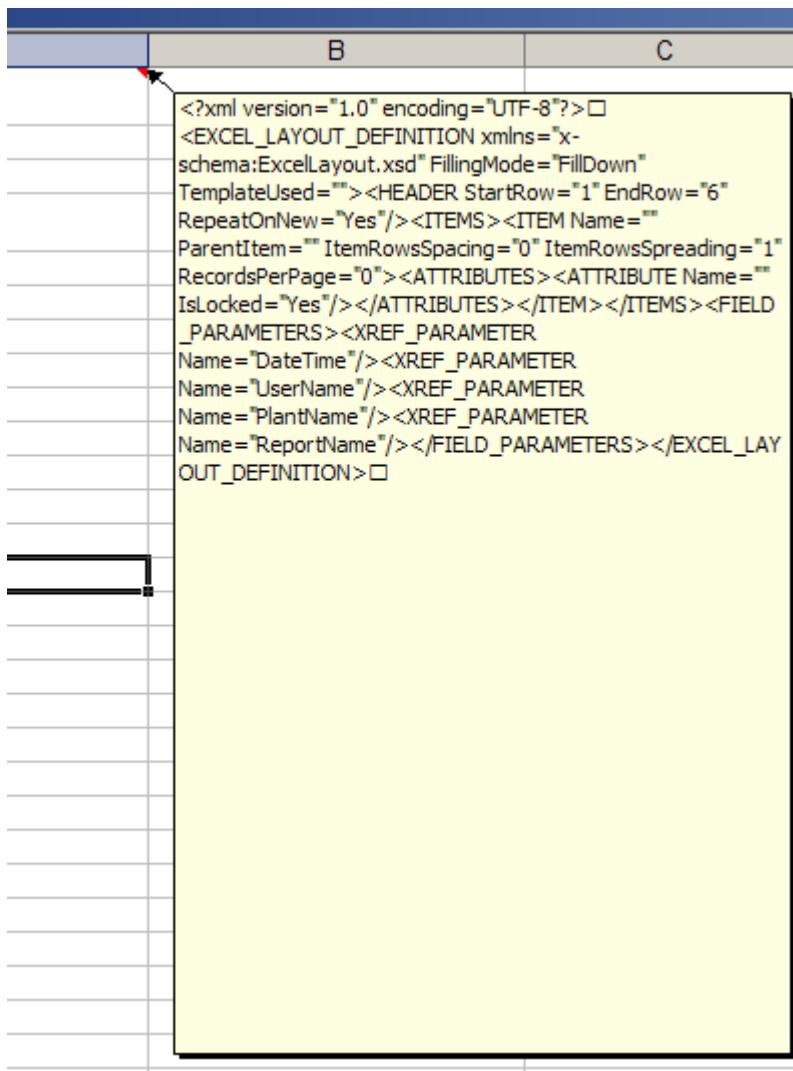
2. When the editor is displayed you will note that there exists a "Sheet3 (SP3DReport_Definition)"



3. Also note that its visibility is set to "2 - xlSheetVeryHidden"

EnableOutlining	True
EnablePivotTable	False
EnableSelection	0 - xlNoRestrictions
Name	SP3DReport_Definition
ScrollArea	
StandardWidth	8.43
Visible	2 - xlSheetVeryHidden

4. If you update this value to visible you will then be able to see what kind of information is stored in the .XLS book specific to formatting.
5. Comments will be displayed that contain raw data about the nature of the resultant data and how it should be handled.



The screenshot shows a Microsoft Excel spreadsheet with three columns labeled A, B, and C. Column B contains a yellow-highlighted cell with the following XML code:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXCEL_LAYOUT_DEFINITION xmlns="x-
schema:ExcelLayout.xsd" FillingMode="FillDown"
TemplateUsed=""><HEADER StartRow="1" EndRow="6"
RepeatOnNew="Yes"/><ITEMS><ITEM Name=""
ParentItem="" ItemRowsSpacing="0" ItemRowsSpreading="1"
RecordsPerPage="0"><ATTRIBUTES><ATTRIBUTE Name=""
IsLocked="Yes"/></ATTRIBUTES></ITEM></ITEMS><FIELD
_PARAMETERS><XREF_PARAMETER
Name="DateTime"/><XREF_PARAMETER
Name="UserName"/><XREF_PARAMETER
Name="PlantName"/><XREF_PARAMETER
Name="ReportName"/></FIELD_PARAMETERS></EXCEL_LAYOUT_DEFINITION>
```

Display Configuration (*.rdy)

The Display Configuration file provides the hook necessary to make use of Excel as the formatting engine. If a different formatting engine was needed to handle it, the Display Configuration file would need to be edited (and additional work beyond the scope of the .rdy file and the report class done).

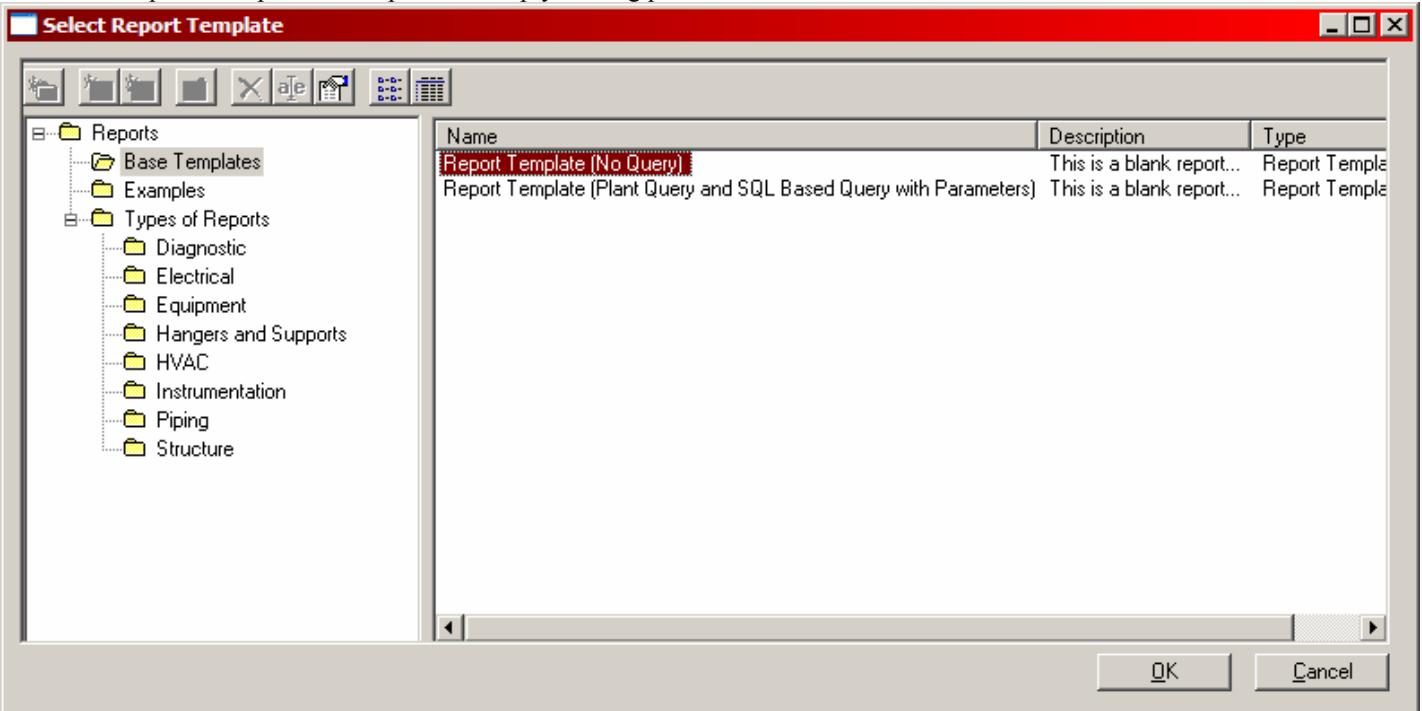
```

<?xml version="1.0" encoding="UTF-8"?>
<REPORT_DISPLAY Name="Report Template (No Query)" Description="Example of viewing with Excel">
  <DESIGN_TIME Progid="" Action="" Arg="" />
  <RUN_TIME Progid="SP3DEXcelFormat.Publisher" Action="DisplayLocal" Arg="True" />
  <PRINT Progid="SP3DEXcelFormat.Publisher" Action="Print" Arg="" />
</REPORT_DISPLAY>

```

Seed Data: Delivered Base Templates and Delivered Reports

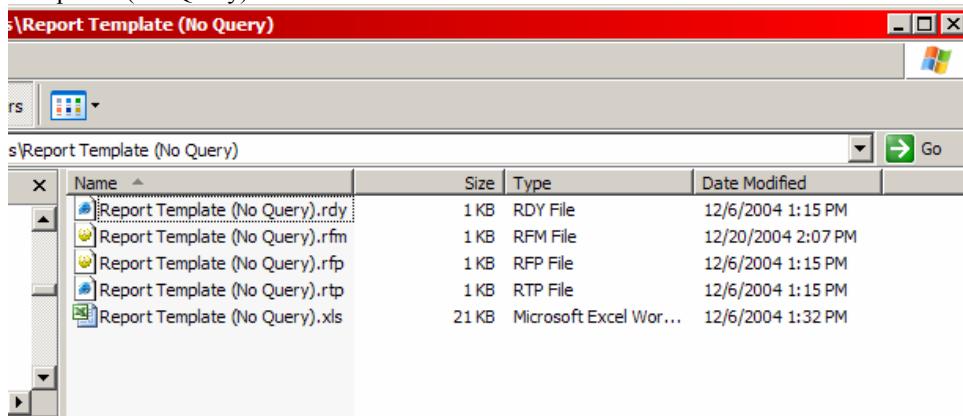
Two base templates are provided as potential empty starting points.



Note: any report in the catalog can represent a starting point for a new reports, the following two are simply empty.

The first base template we have already used, Report Template (No Query). It establishes a report template without an existing query set, without labels, grouping rules, etc.

The XML and XLS files that describe this template are available on the symbol share at <symbols>\Reports\Base Templates\Report Templates (No Query).



The second baseline template is “Report Template (Plant Query and SQL Query with Parameters) It provides you with two Queries, a SQL Query on the Plant, and an example Parameterized Query.

Management Console

Query: Plant

```
SELECT jcpr.oid,
jni.itemName AS Name,
juap.Owner AS Owner,
juap.Site AS Site
FROM JNamedItem jni
JOIN JConfigProjectRoot jcpr ON (jcpr.oid = jni.oid)
JOIN XConfigurableProjectRoot r1 on r1.oidorigin = jcpr.oid
JOIN JDUAProjectRoot juap ON juap.oid= r1.oiddestination
OPTION (MAXDOP 1)
```

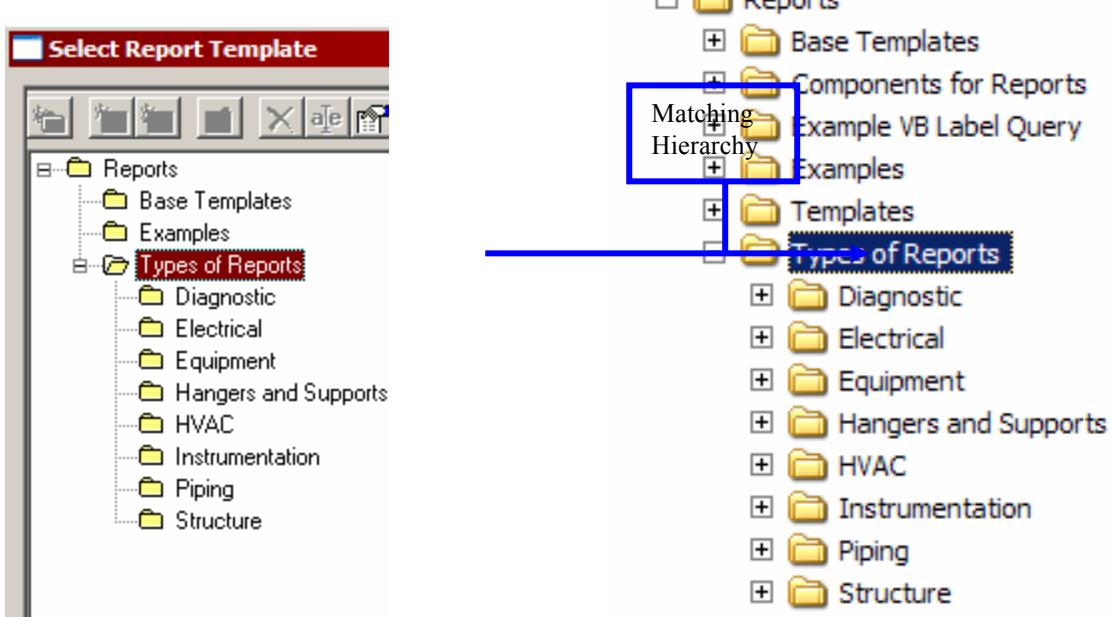
Query: QWP

```
SELECT j.oid, n.itemname as Name
FROM jobject j
join inameditem n on n.oid = j.oid
WHERE n.itemname = ?
```

The Excel layout and other items are still empty in nature. The files that describe this template can also be found on the symbol share. Their location is <symbols>\Reports\Base Templates\Report Template (Plant Query and SQL Based Query with Parameters)

Name	Size	Type	Date Modified
Report Template (Plant Query and SQL Based Query with Parameters).rqe	1 KB	RQE File	2/2/2005 3:43 PM
Report Template (Plant Query and SQL Based Query with Parameters).QWP.rqe	1 KB	RQE File	4/20/2005 3:47 PM
Report Template (Plant Query and SQL Based Query with Parameters).rdy	1 KB	RDY File	4/20/2005 3:52 PM
Report Template (Plant Query and SQL Based Query with Parameters).rfm	1 KB	RFM File	4/20/2005 3:52 PM
Report Template (Plant Query and SQL Based Query with Parameters).rfp	1 KB	RFP File	4/20/2005 3:52 PM
Report Template (Plant Query and SQL Based Query with Parameters).rqp	1 KB	RQP File	4/20/2005 3:52 PM
Report Template (Plant Query and SQL Based Query with Parameters).rtp	1 KB	RTP File	4/20/2005 3:46 PM
Report Template (Plant Query and SQL Based Query with Parameters).xls	21 KB	Microsoft Excel Wor...	12/6/2004 1:34 PM

But, any delivered sample report (or any report that you create) could serve as a starting point for your final report:



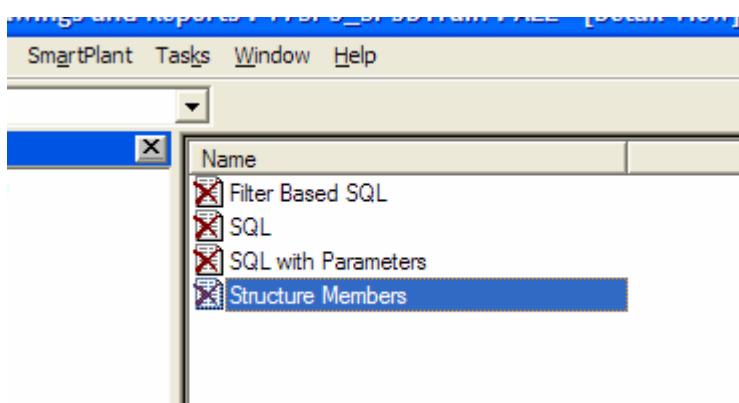
Query Specific Settings

Grouping

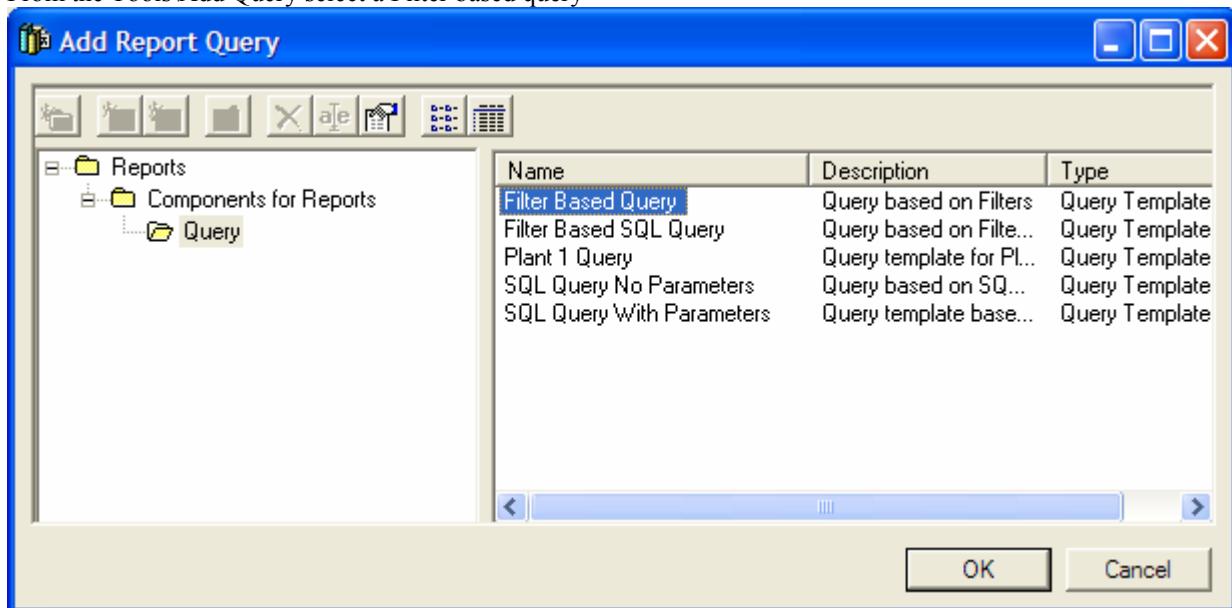
GUI controls for grouping are accessible from within the Design Layout tool.

Let's create a report that will return Length, Weight, and Cross Section name for all structure members.

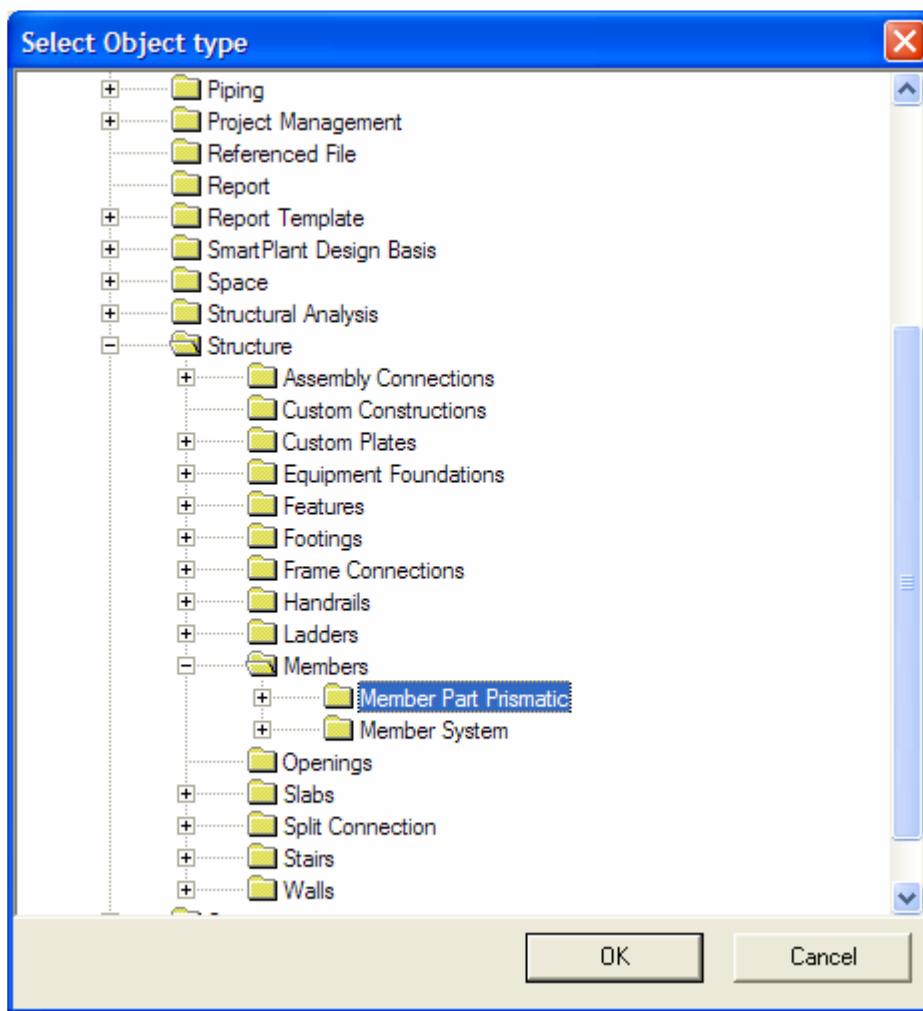
1. Create a new Report Component "Structure Members" starting from a Report Template(No Query)

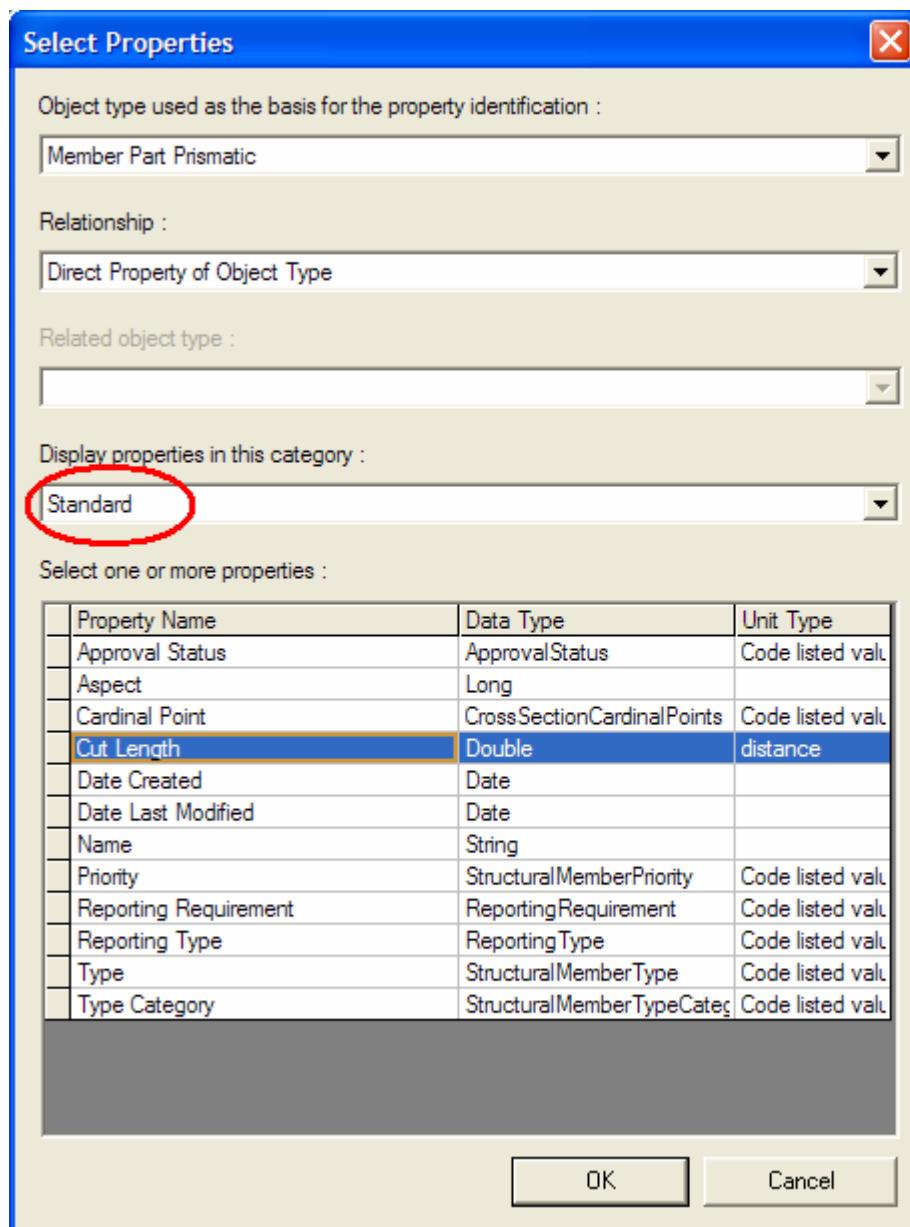


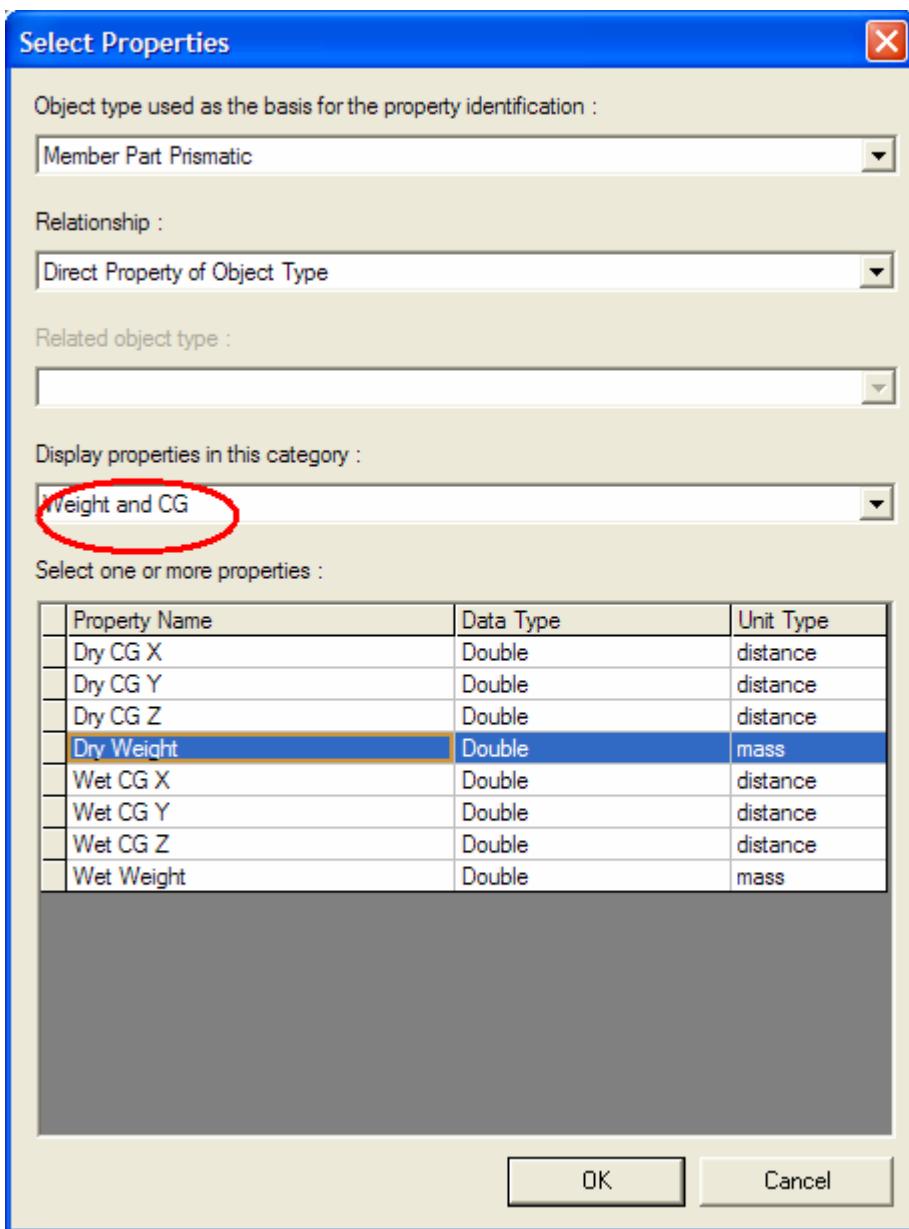
2. Right-click\Edit Template
3. From the Tools\Add Query select a Filter based query



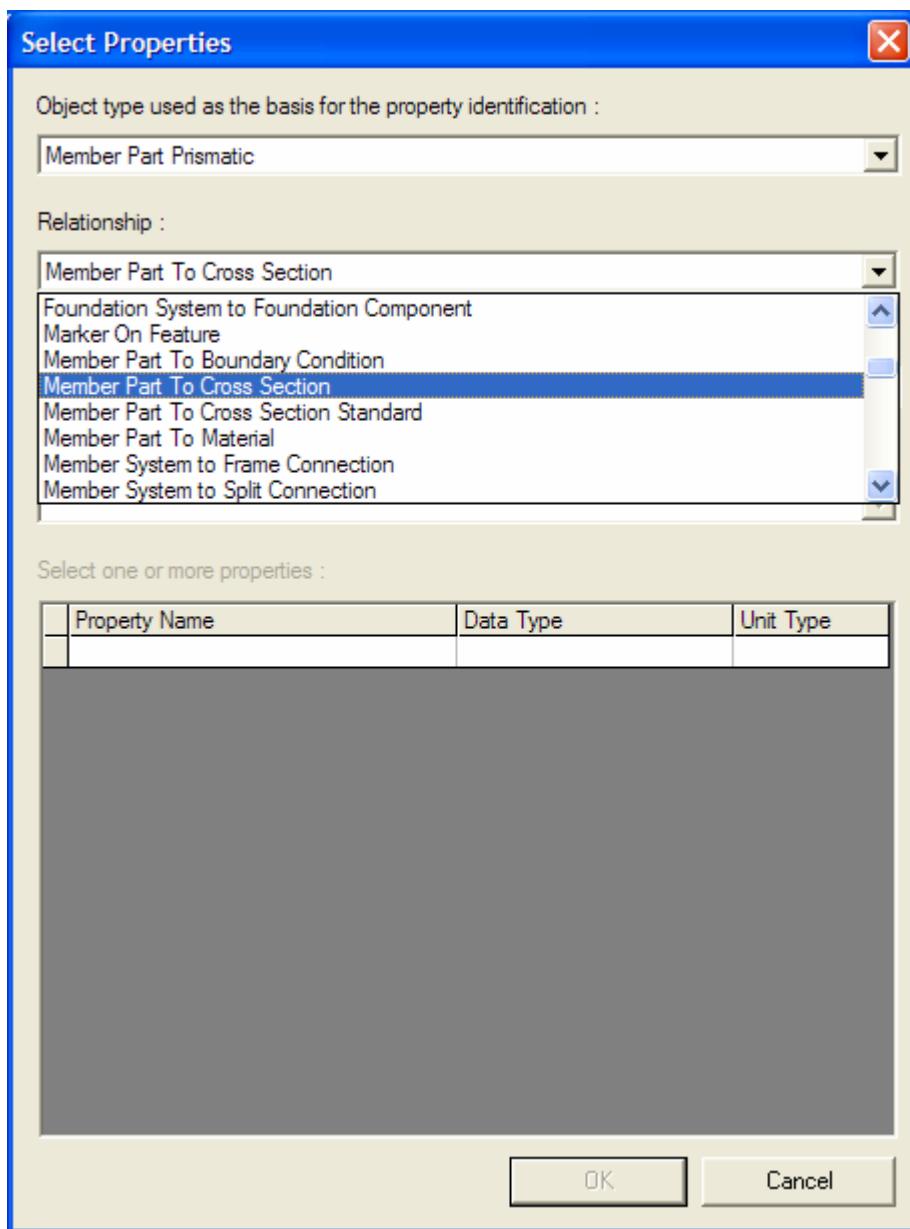
4. Name the query "Members", and select the delivered filter Catalog Filters\Default Filters\SP3D Object Filters\Object Types\Structure\Members\Member Part Prismatic
5. Add the Properties Cut Length and Dry Weight as direct properties of object type Member Part Prismatic as shown:

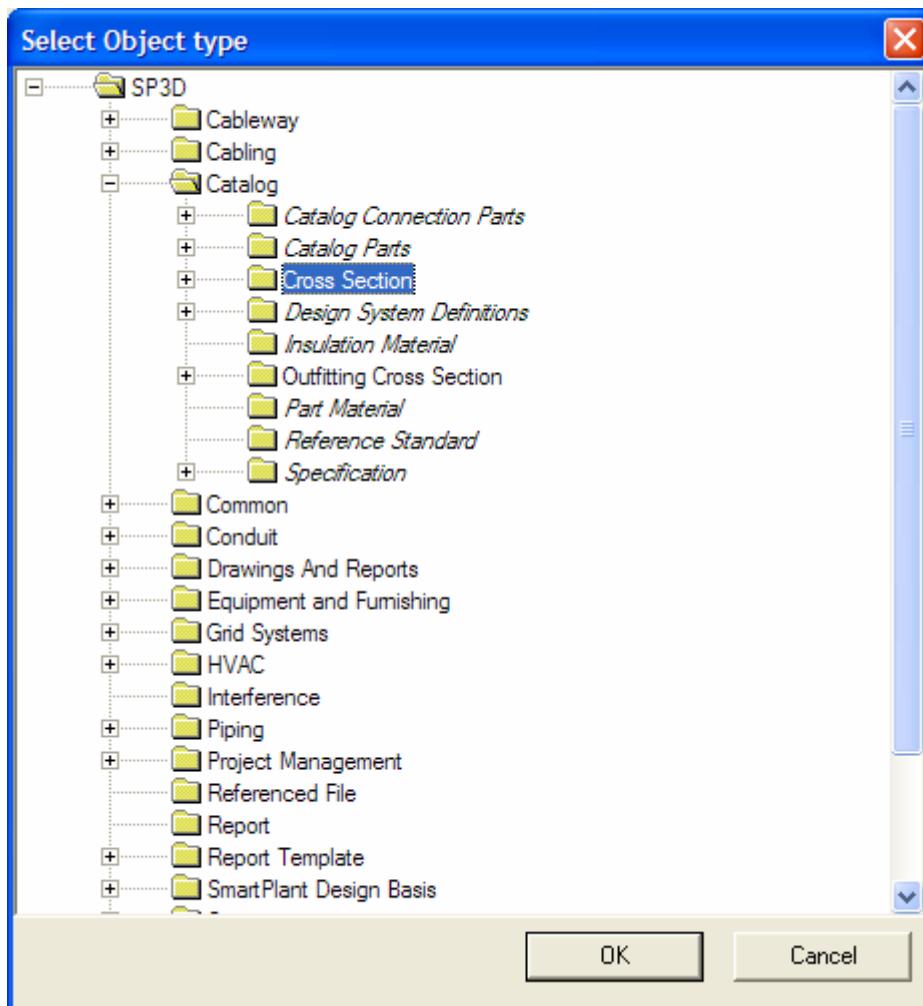


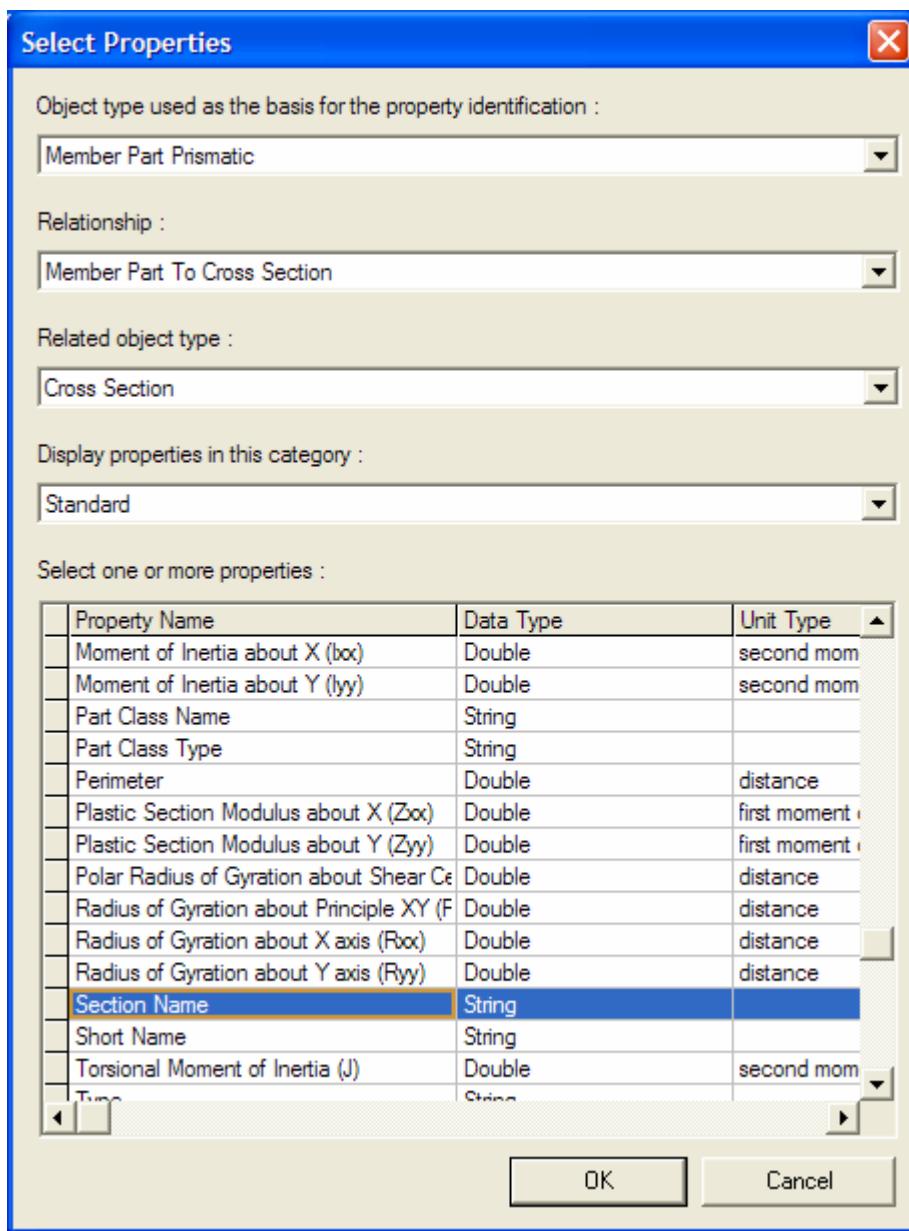




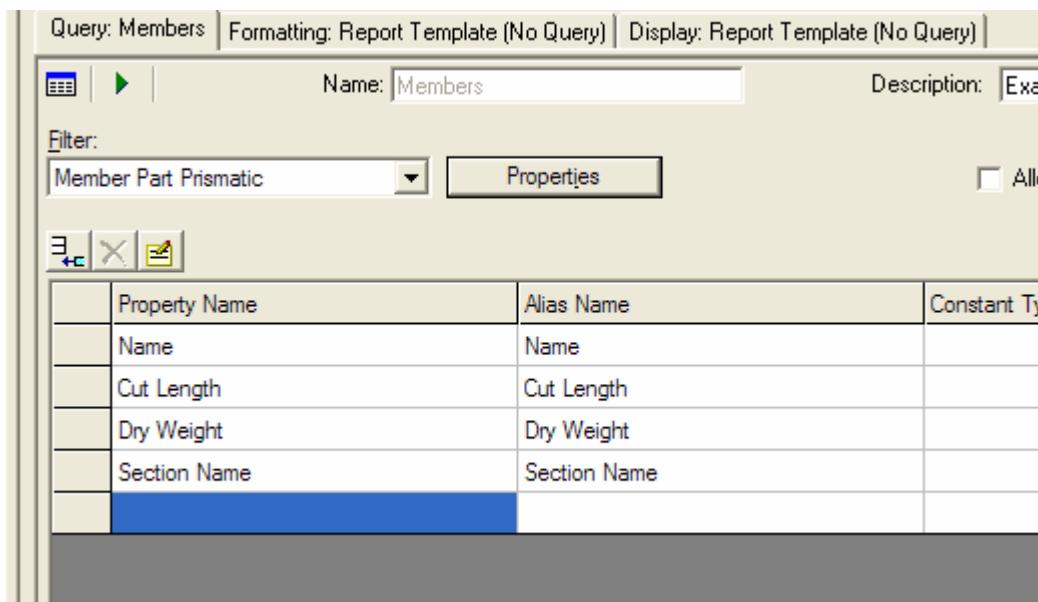
6. Add the property CrossSection Name by starting from a Member Part Prismatic object and traversing the relationship "Member Part To Cross Section" to a Cross Section object in the catalog:







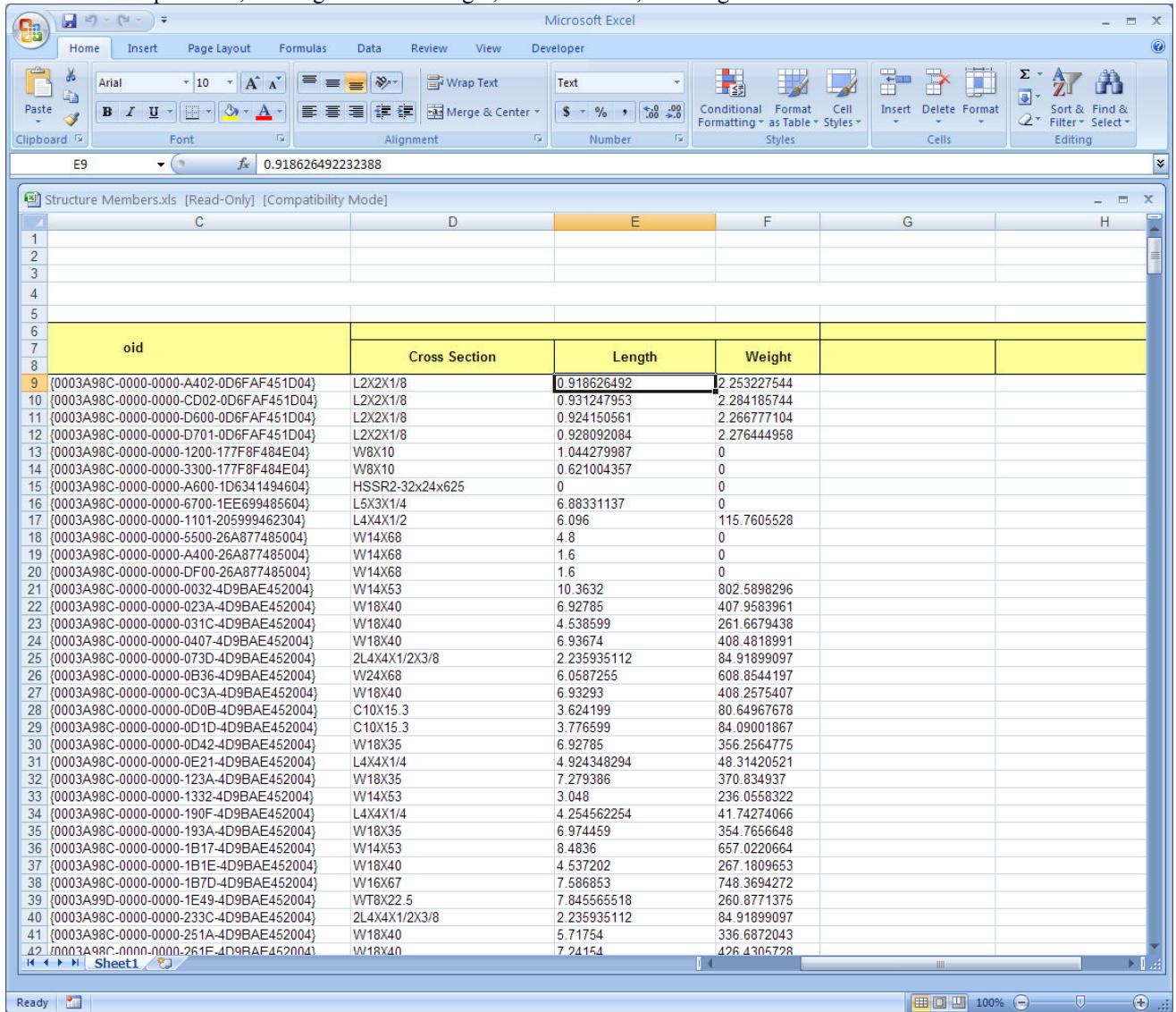
7. Your Query should look like this:



8. Go to the Design Layout and drag the properties on the sheet

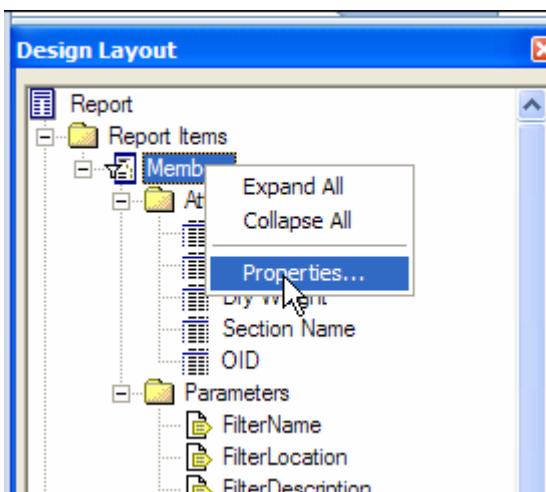
oid	Cross Section	Length	Weight
#Members::OID#	#Members::Section Name#	#Members::Cut Length#	#Members::Dry Weight#

9. If we test the report now, it will give us the Weight, Cross Section, and length of each individual member:

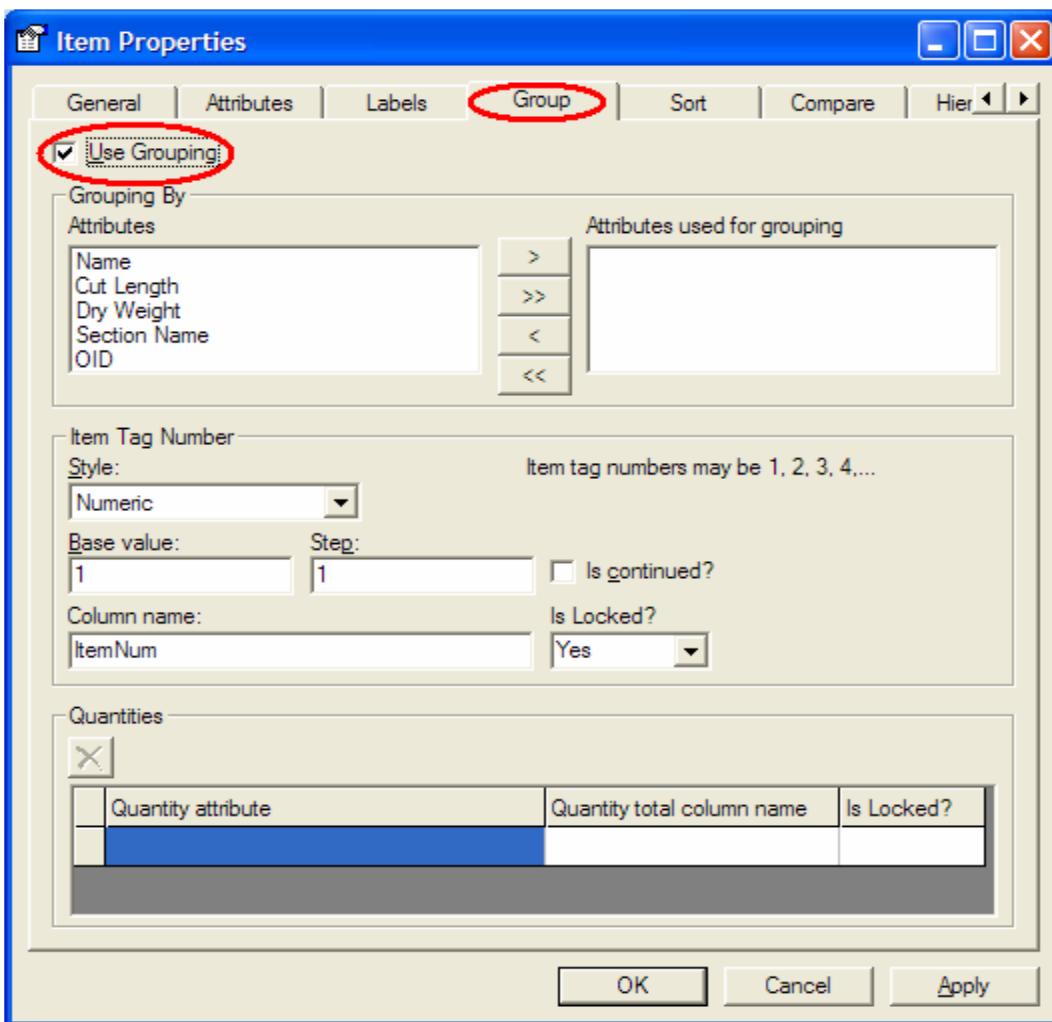


C	D	E	F	G	H
oid	Cross Section	Length	Weight		
9 {0003A98C-0000-0000-A402-0D6FAF451D04}	L2X2X1/8	0.918626492	2.253227644		
10 {0003A98C-0000-0000-CD02-0D6FAF451D04}	L2X2X1/8	0.931247953	2.284185744		
11 {0003A98C-0000-0000-D600-0D6FAF451D04}	L2X2X1/8	0.924150561	2.266777104		
12 {0003A98C-0000-0000-D701-0D6FAF451D04}	L2X2X1/8	0.928092084	2.276444958		
13 {0003A98C-0000-0000-1200-177F8F484E04}	W8X10	1.044279987	0		
14 {0003A98C-0000-0000-3300-177F8F484E04}	W8X10	0.621004357	0		
15 {0003A98C-0000-0000-A600-1D6341494604}	HSSR2-32x24x625	0	0		
16 {0003A98C-0000-0000-6700-1EE699485604}	L5X3X1/4	6.88331137	0		
17 {0003A98C-0000-0000-1101-20599462304}	L4X4X1/2	6.096	115.7605528		
18 {0003A98C-0000-0000-5500-26A877485004}	W14X68	4.8	0		
19 {0003A98C-0000-0000-A400-26A877485004}	W14X68	1.6	0		
20 {0003A98C-0000-0000-DF00-26A877485004}	W14X68	1.6	0		
21 {0003A98C-0000-0000-0032-4D9BAE452004}	W14X53	10.3632	802.5898296		
22 {0003A98C-0000-0000-023A-4D9BAE452004}	W18X40	6.92785	407.9583961		
23 {0003A98C-0000-0000-031C-4D9BAE452004}	W18X40	4.538599	261.6679438		
24 {0003A98C-0000-0000-0407-4D9BAE452004}	W18X40	6.93674	408.4818991		
25 {0003A98C-0000-0000-073D-4D9BAE452004}	2L4X4X1/2X3/8	2.235935112	84.91899097		
26 {0003A98C-0000-0000-0B36-4D9BAE452004}	W24X68	6.0587255	608.8544197		
27 {0003A98C-0000-0000-0C3A-4D9BAE452004}	W18X40	6.93293	408.2575407		
28 {0003A98C-0000-0000-0D0B-4D9BAE452004}	C10X15..3	3.624199	80.64967678		
29 {0003A98C-0000-0000-001D-4D9BAE452004}	C10X15..3	3.776599	84.09001867		
30 {0003A98C-0000-0000-0042-4D9BAE452004}	W18X35	6.92785	356.2564775		
31 {0003A98C-0000-0000-E21-4D9BAE452004}	L4X4X1/4	4.924348294	48.31420521		
32 {0003A98C-0000-0000-123A-4D9BAE452004}	W18X35	7.279386	370.834937		
33 {0003A98C-0000-0000-1332-4D9BAE452004}	W14X53	3.048	236.0558322		
34 {0003A98C-0000-0000-190F-4D9BAE452004}	L4X4X1/4	4.254562254	41.74274066		
35 {0003A98C-0000-0000-193A-4D9BAE452004}	W18X35	6.974459	354.7656648		
36 {0003A98C-0000-0000-1B17-4D9BAE452004}	W14X53	8.4836	657.0220664		
37 {0003A98C-0000-0000-1B1E-4D9BAE452004}	W18X40	4.537202	267.1809653		
38 {0003A98C-0000-0000-1B7D-4D9BAE452004}	W16X67	7.586853	748.3694272		
39 {0003A99D-0000-0000-1E49-4D9BAE452004}	WT8X22..5	7.84565518	260.8771375		
40 {0003A98C-0000-0000-233C-4D9BAE452004}	2L4X4X1/2X3/8	2.235935112	84.91899097		
41 {0003A98C-0000-0000-251A-4D9BAE452004}	W18X40	5.71754	336.6872043		
42 {0003A98C-0000-0000-261F-4D9BAE452004}	W18X40	7.24154	426.4305728		

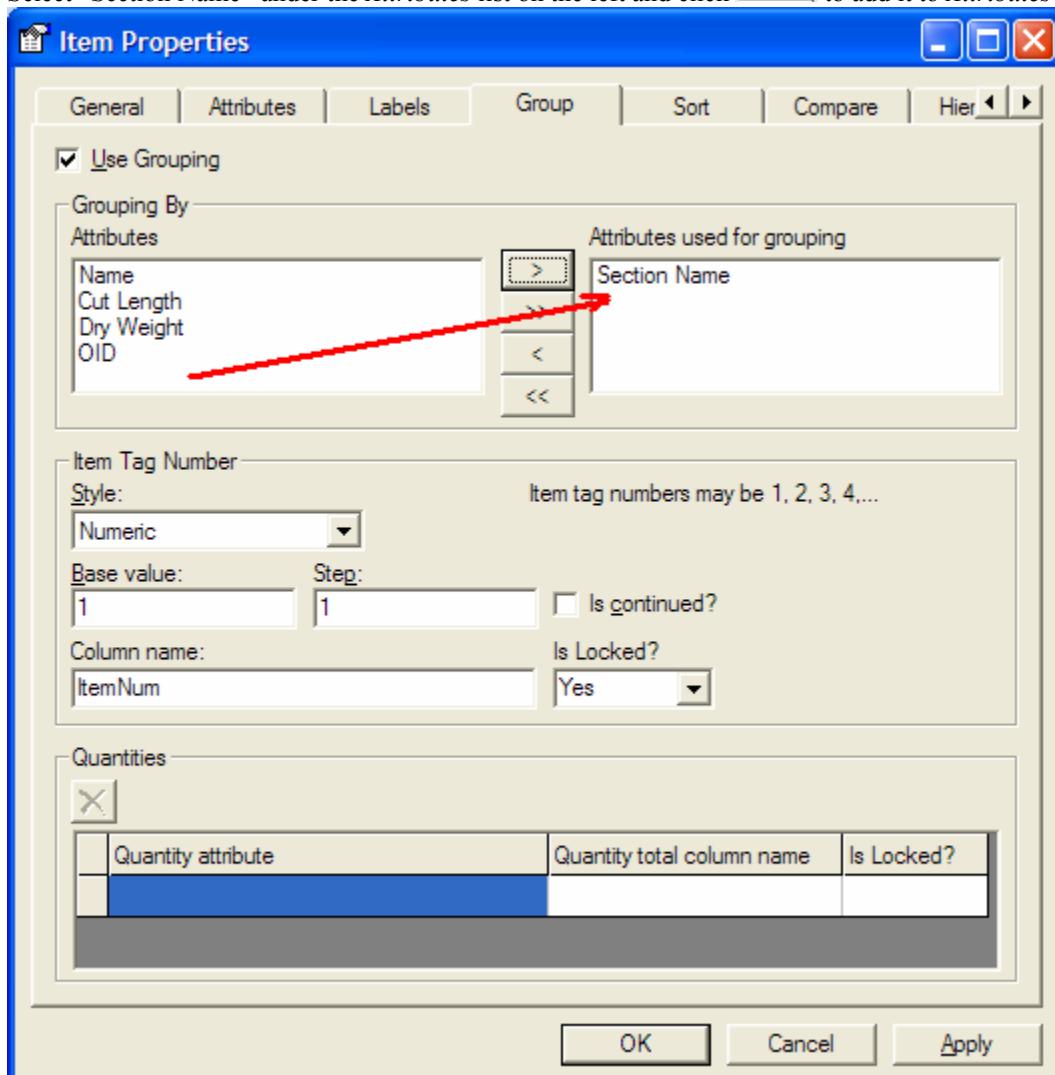
10. To get the **combined** weight of all members with a given Cross Section, we need to create groups based on Cross Sections. Edit the template and go in the Design Layout. Right-click on the query "Members" and select "Properties..."



11. Select the tab “Grouping” and check the “Use Grouping” check-box.

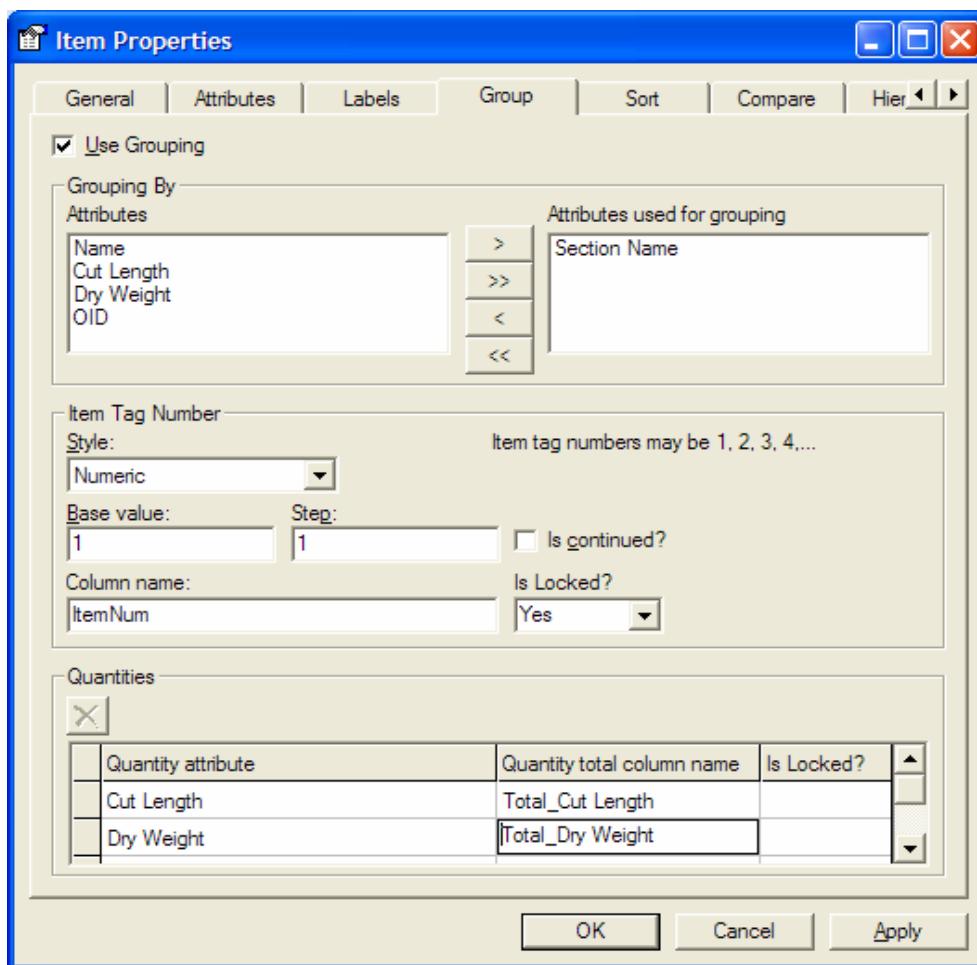


12. Select “Section Name” under the *Attributes* list on the left and click to add it to *Attributes used for grouping*.

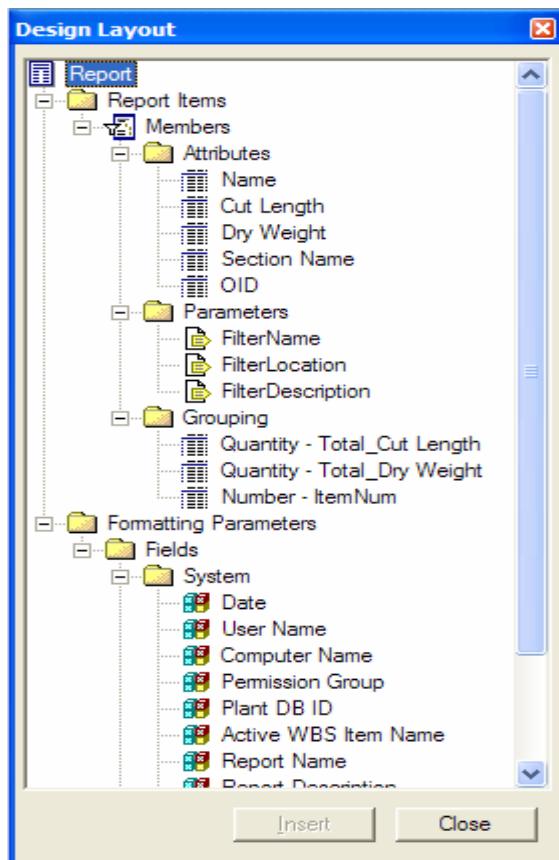


Note: Although here we are grouping based on one attribute, we could also group on multiple attributes – every unique multiple of the selected attributes will form one group.

13. Under *Quantity attribute* at the bottom of the form select “Cut Length” and “Dry Weight”



14. Click OK
15. Observe that a new branch, "Grouping," is displayed in the Design Layout hierarchy with "Quantity – Total_Cut Length", "Quantity – Total_Dry Weight", and "Number – ItemNum" as children.



16. Use Drag-and-Drop and add the 2 new Grouping quantitative attributes - "Quantity – Total_Cut Length", "Quantity – Total_Dry Weight", and "Number – ItemNum".

oid						
	Cross Section	Length	Weight	Total Length	Total Weight	ItemNum
#Members::OID#Members::Section#Members::Cut#Members::#Members::Total_#Members::Total#Members::ItemN						

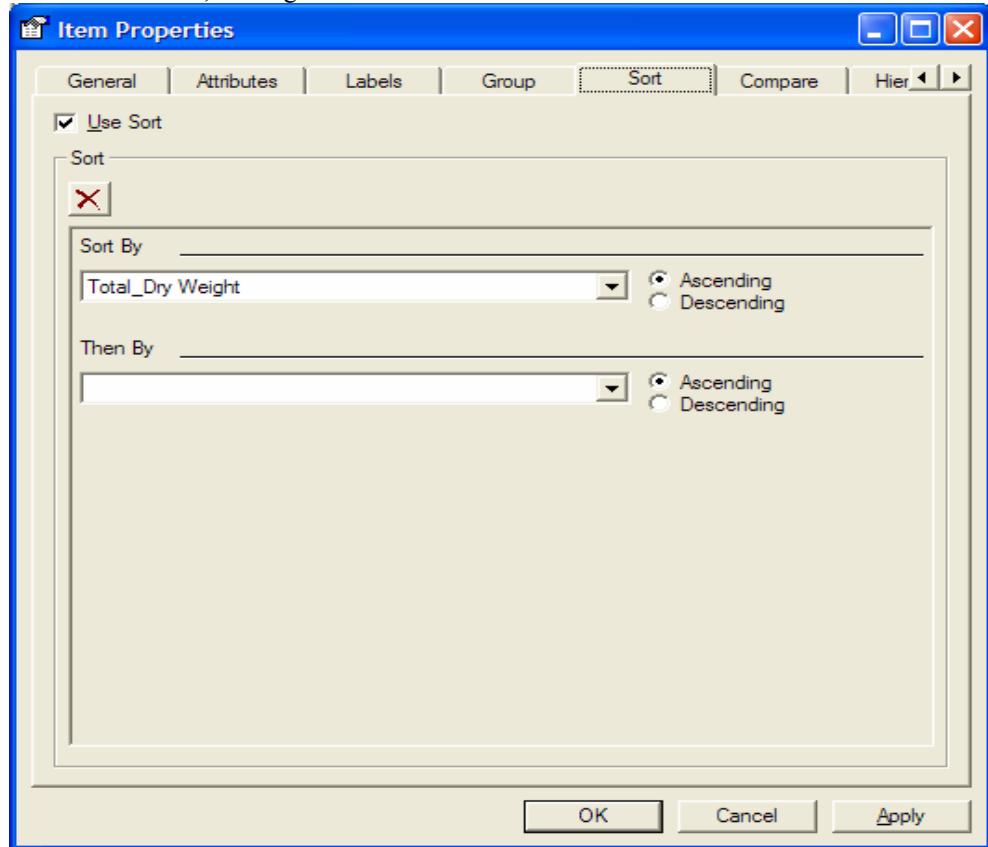
17. Save Changes to the XLS.
 18. Save Changes to the Report Template.
 19. Generate the Report – note the item number assignments, and the Length and Weight Totals.

oid						
	Cross Section	Length	Weight	Total Length	Total Weight	ItemNum
{0003A98C-0000-0000-073D 2L4X4X1/2X3/8	2.235935112	84.918991	38.76441589	1472.240883	1	
{0003A98C-0000-0000-233C 2L4X4X1/2X3/8	2.235935112	84.918991	38.76441589	1472.240883	1	
{0003A98C-0000-0000-473E 2L4X4X1/2X3/8	8.0137	304.35379	38.76441589	1472.240883	1	
{0003A98C-0000-0000-603E 2L4X4X1/2X3/8	8.0184625	304.53466	38.76441589	1472.240883	1	
{0003A98C-0000-0000-AD3C 2L4X4X1/2X3/8	2.228220663	84.626003	38.76441589	1472.240883	1	
{0003A98C-0000-0000-E33E 2L4X4X1/2X3/8	8.0137	304.35379	38.76441589	1472.240883	1	
{0003A98C-0000-0000-FD3E 2L4X4X1/2X3/8	8.0184625	304.53466	38.76441589	1472.240883	1	
{0003A98C-0000-0000-0D0B C10X15.3	3.624199	80.649677	270.4735933	2295.637995	2	
{0003A98C-0000-0000-0D1D C10X15.3	3.776599	84.090019	270.4735933	2295.637995	2	

The ItemNumber creates a unique name for each group. The “Total Weight” and “Total Length” sums the Weight and Length within each group and outputs them as attributes for each member in that group. We’ll see later on how to show only one row for each group.

Sorting

1. Continue using the “Structure Members” Report from the previous section.
2. Edit the template once again and return to the Design Layout tool.
3. Right mouse on the tree (recall how you previously accessed the Group tab) and go to the Sort tab.
4. Complete the form as follows, making sure to check the “Use Sort” checkbox first:



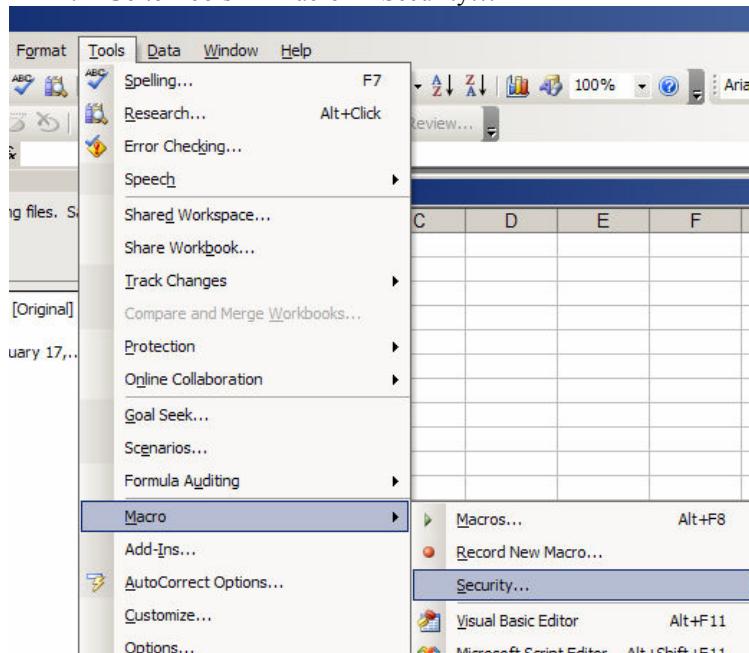
5. Save the layout xls.
6. Save the Report Template.
7. Regenerate the Report and notice how the items are sorted based on Total Weight.

Note: If you have selected to both Group and Sort, the columns used in the “Sort” tab must be identified on the “Group” tab as either an aggregate function (group quantity) or selected as columns used for grouping.

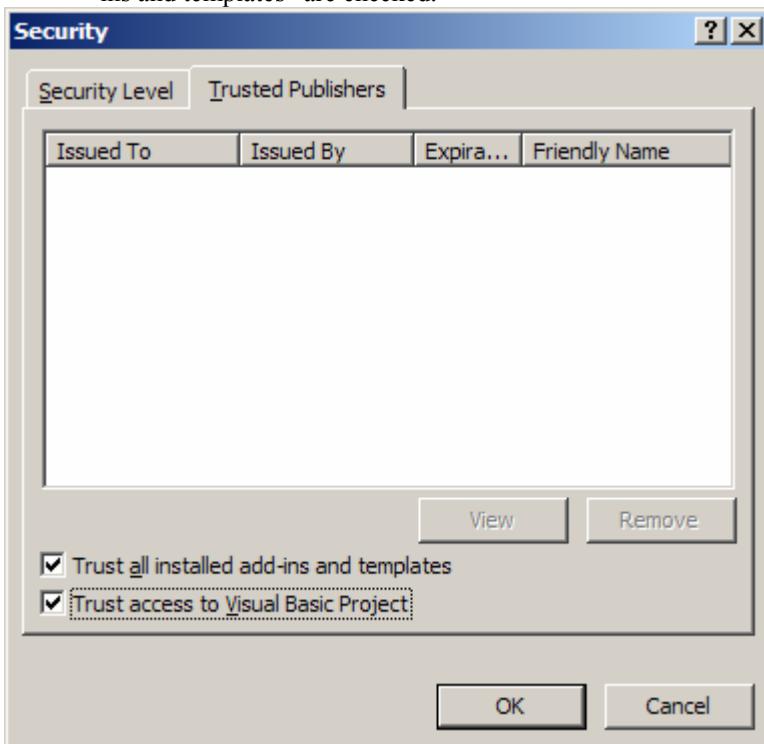
Macros

When working with Macros, it is important that you verify your security settings in Excel. The following screenshots are from Microsoft Office 2003 but similar forms appear on previous versions of Microsoft Office.

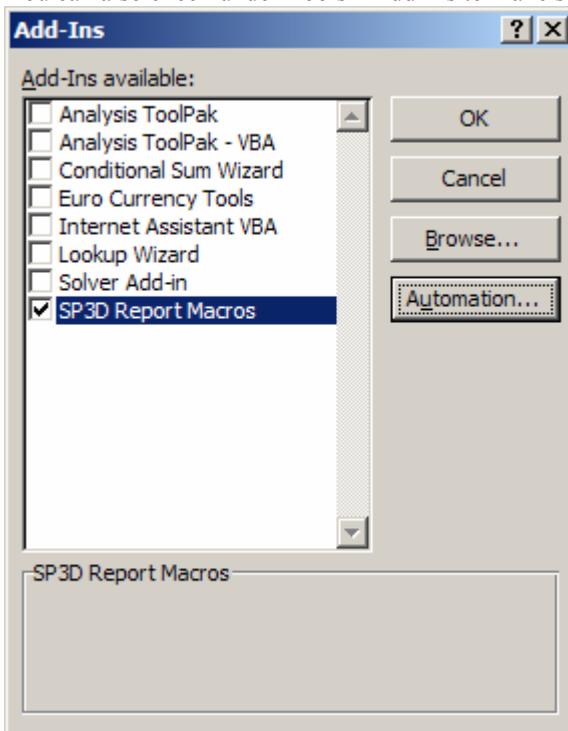
1. Start Excel.
2. Go to Tools → Macro → Security...



3. Set value of Security Level to Medium
4. On Trusted Publishers make sure that “Trust access to Visual Basic Project” and “Trust all installed add-ins and templates” are checked.

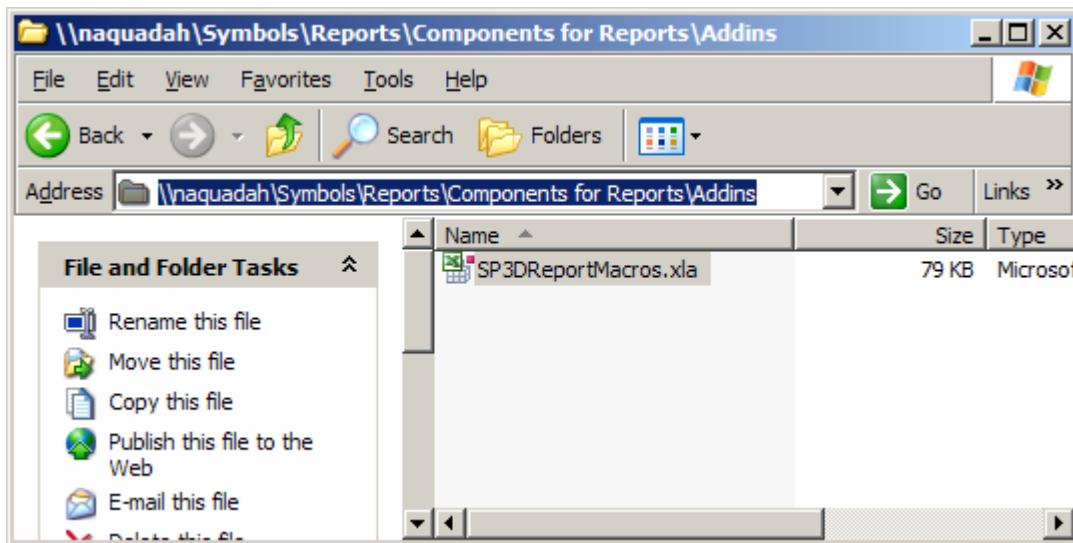


You can also check under Tools->Add-ins to make sure that the SP3D Report Macros is selected.



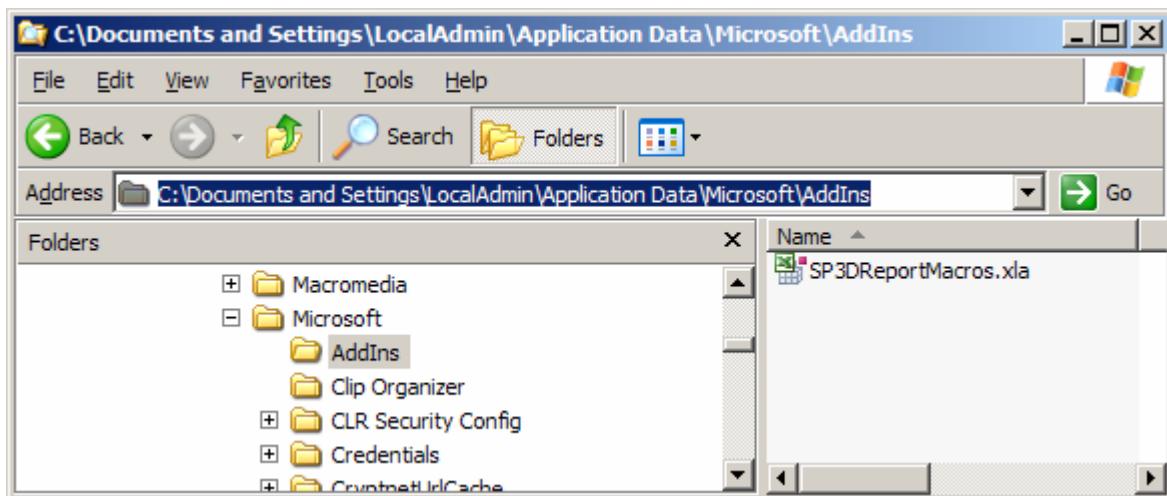
Note: For Excel 2007 those settings are under *Excel Options/Trust Center/Trust Center Settings ...*

The Macros for SmartPlant 3D are initially stored on the symbols directory at “<Symbols>\Reports\Components for Reports\Addins”



Except for that delivered SP3DReportMacros.xla file, the users can place their own macro AddIn (.xla) files in the same location, and they will be automatically recognized and treated as the delivered one.

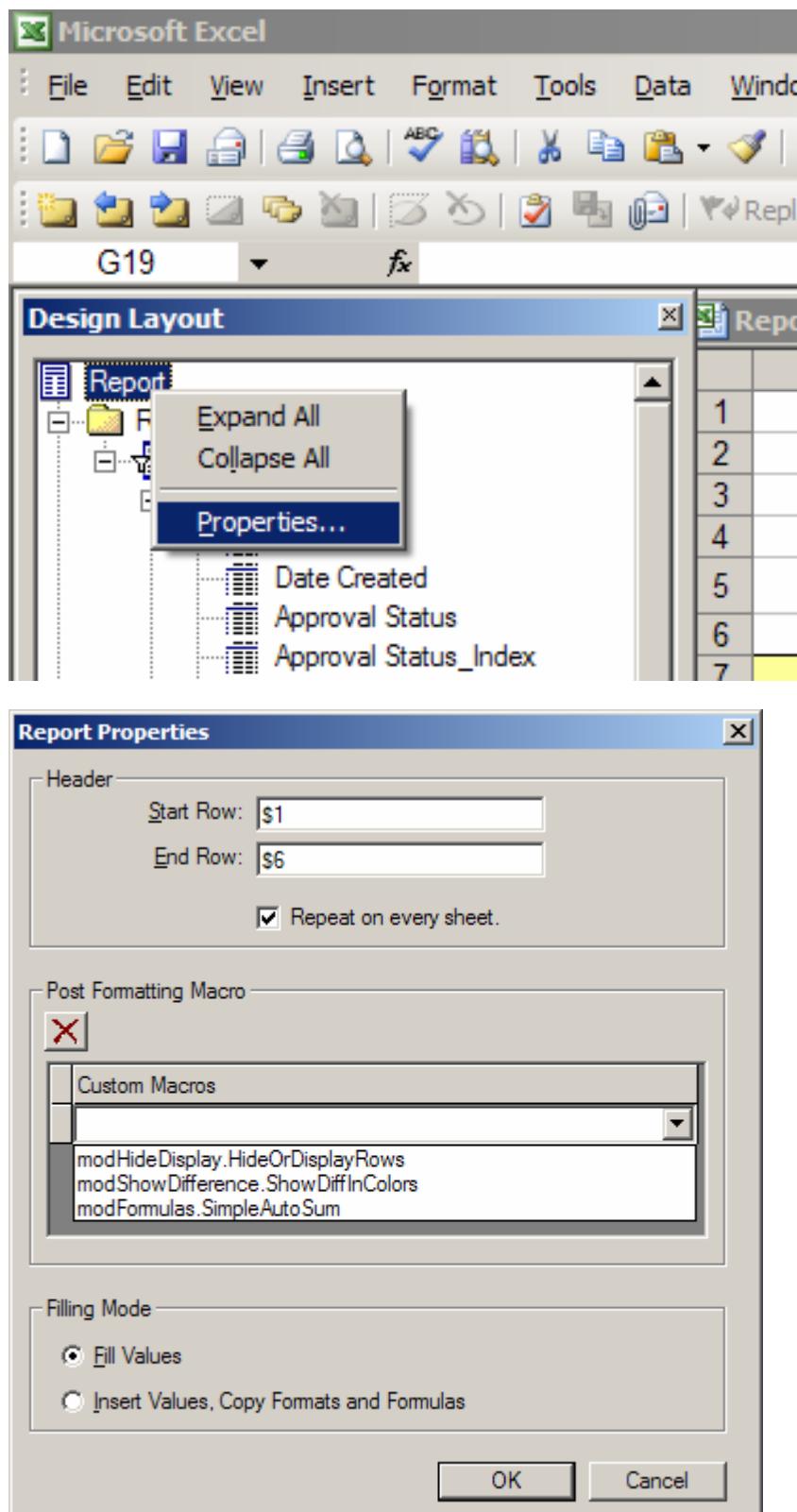
When SP3D Reports functionality is used on an SP3D Workstation for a particular user, the .xla file(s) is copied to <Documents and Settings of the currently logged in user>\Application Data\Microsoft\AddIns.



If you are experiencing problems with SP3D Report Macros you should check for these files and verify that the file on the local machine is the same as the file on the Symbol directory. They should always be in-sync with the latest file posted to the Symbol Share, and then automatically copied down whenever a never file that the one stored locally exists.

You should also verify the Security Settings whenever SP3D Report Macros are failing. Often times these settings may be reset by Office Service Packs or Microsoft Critical Updates.

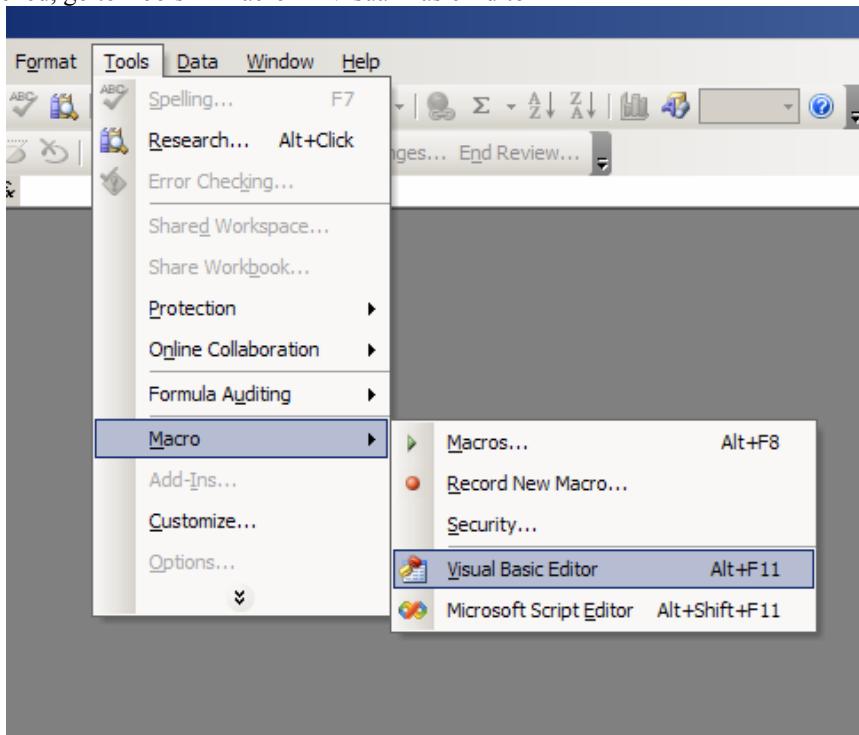
Applying Macros to your SP3D Reports is performed through the Design Layout tool by Right Mouse Clicking on the Report root node in the Design Layout browser.



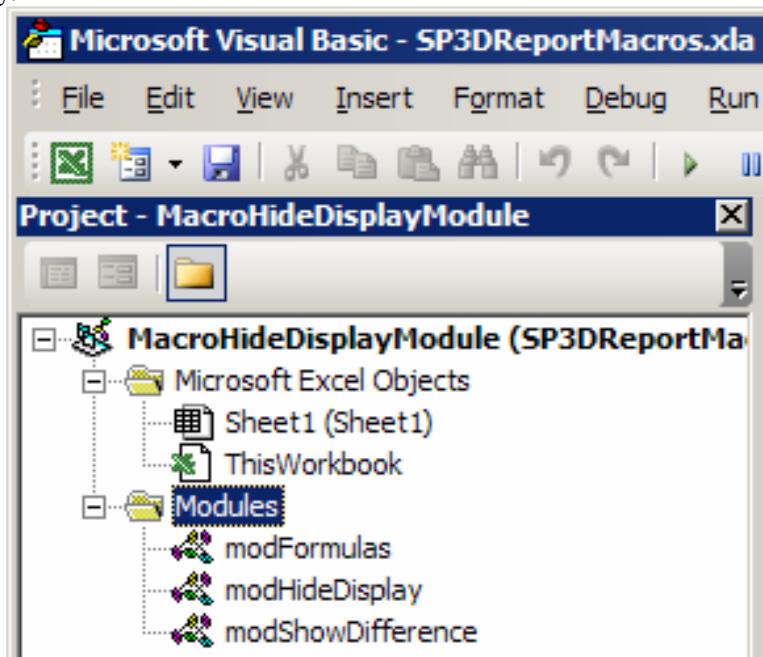
From the drop down list you can select each macro that you wish to apply, the appropriate Start and End Rows that it applies against, and the appropriate Fill Mode.

Open the XLA from within Excel.

Once opened, go to Tools → Macro → Visual Basic Editor

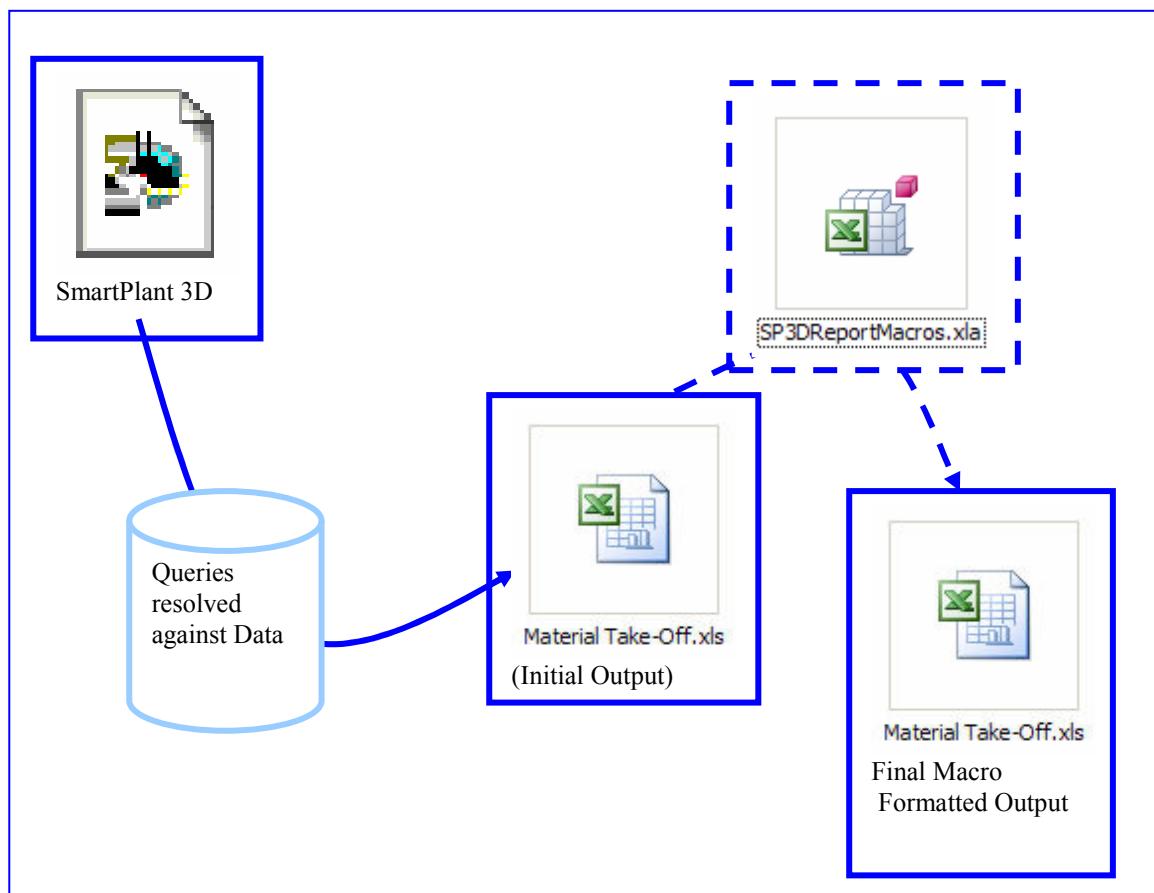


Once the Visual Basic Editor is open you can find the delivered SP3D Macros under the Modules\ branch of the Project hierarchy.



This macro set can be customized to do such things as separate Report information onto multiple sheets within the same workbook, change color, text formatting, etc of cells based on information in the cell(s), etc

These Macros are fully defined and governed within the Macro .xla file, stored within the layout.xls and readily applied to the layout.xls through the Design Layout Tool.



You can also add your own macro to the .xls file of your report template. To enable such custom macros to run, you need to edit the XML contained in the SP3DReport_Definition sheet of the report (see “Layout” for instructions on how to do this). Inside the <MACROS> tag you need to specify the *Module* and *Method* of your macro. *ExcelOrAddInFile* should be empty, since in this case the macro resides in the same xls file as your report:

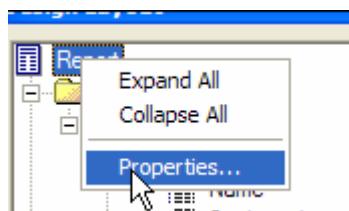
```

LocalFormatParamPath="C8"><INPUT_ATTRIBUTES><XREF_ATTRIBUTE
Name="TCG"/></INPUT_ATTRIBUTES><OUTPUT_ATTRIBUTE Name="outTCG"
IsLocked="No"/></LABEL></LABELS></ITEM></ITEMS><UOM_PARAMETERS><XREF_PARAMETER
Name="Mass"/><XREF_PARAMETER Name="Weight Force"/><XREF_PARAMETER
Name="Distance"/></UOM_PARAMETERS><MATRIX_PARAMETERS><MACROS><MACRO
ExcelOrAddInFile="" Module="Macro" Method="FinalFormat" Type=""
Arg=""></MACROS></REPORT_SHEET_LAYOUT></EXCEL_LAYOUT_DEFINITION>

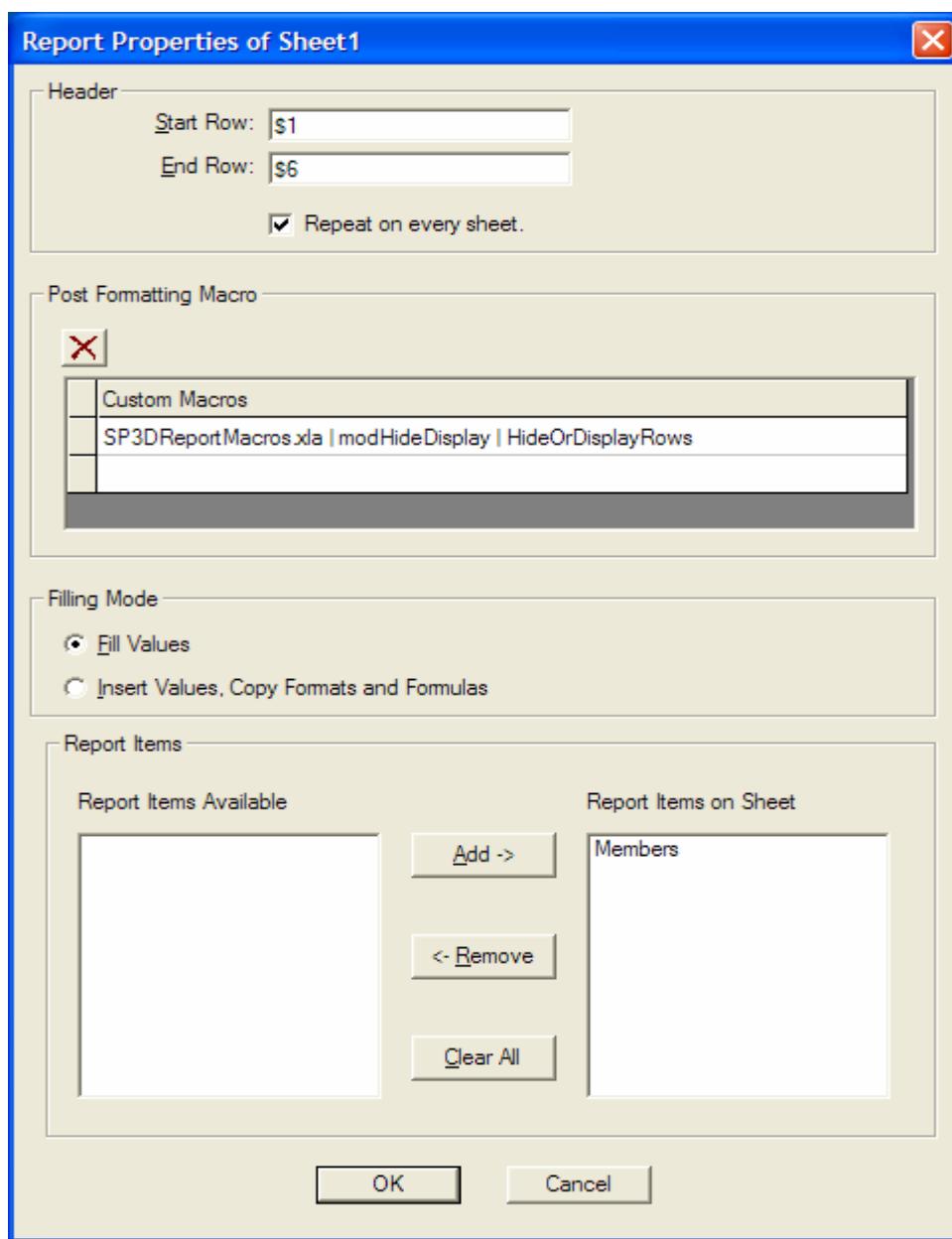
```

Let's apply a macro to the "Structure Members" example to leave only one row per group (Cross Section in our case)

1. Open the report for editing and go in the Design Layout
2. Right-click on the *Report* node and select Properties



3. In the Custom Macros section select the delivered Macro "Hide Or Display Rows":



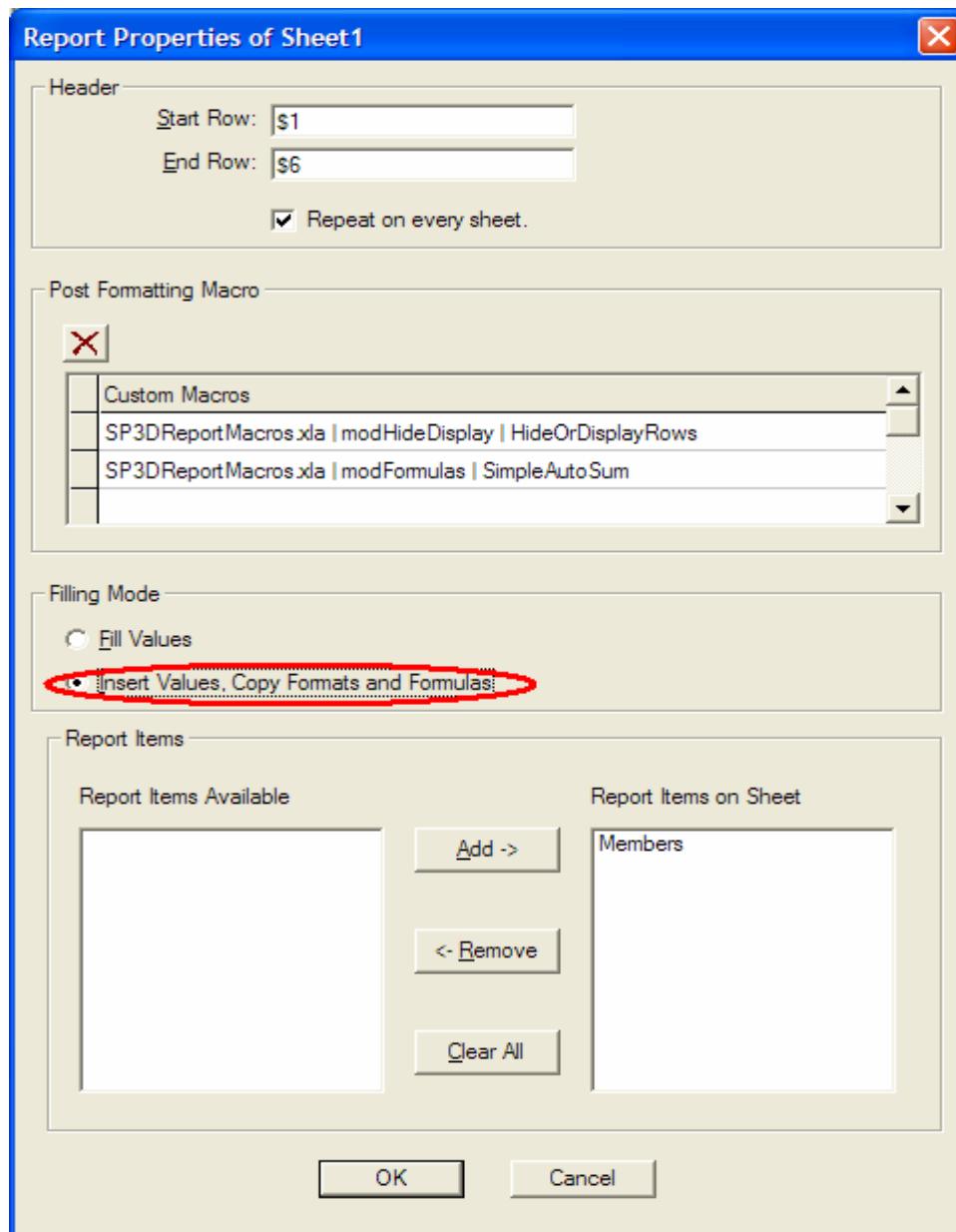
This macro will hide all but the first row for each group. It is a good idea to hide instead of delete the extra rows, so that we can still search on individual items when we need to.

- Save the changes and run the report. You'll notice that we are getting only one row per cross section, but the row numbers "jump" since we have hidden rows.

6	oid	Cross Section	Length	Weight	Total Length	Total Weight	ItemNum
7							
8							
9	{0003W18X40}	6.92785	407.9584	253.1618	14863.10694	29	
51	{0003W14X53}	10.3632	802.58983	165.2829237	12038.84744	22	
75	{0003W18X35}	6.92785	356.25648	318.6799308	5261.94877	28	
154	{0003W24X68}	6.0587255	608.85442	48.472598	4871.116133	34	
162	{0003W14X82}	10.3632	1243.3063	35.3314	3217.969263	25	
167	{0003W18X50}	6.62305	487.49077	39.75862	2926.440291	30	
173	{0003W16X67}	7.586853	748.36943	37.286946	2926.356341	27	

If we want to sum the total weight for all members of all cross sections, we can just use an Excel Sum formula to sum the "Length" column which holds the individual members weights (it will not ignore hidden rows). However, we probably would not even want to see this column, not to mention that it would be much nicer if that Sum is done for us automatically and appears at the bottom of the *Total Weight* column. Since the Total Weight column holds the total weight per group repeated for each member in this group, using the simple Excel formula will not give correct results. We can, however, use another delivered macro.

- Open the template for editing and go in the Design Layout.
- Go to the properties of the *Report* node as before and add the **SimpleAutoSum** macro. Also make sure that you select this time **Insert Values, Copy Formats and Formulas** for **Filling Mode**:



7. Click "OK".
8. In the excel sheet type "=SUM()" and give a range as shown:

B	C	D	E	F	G	H	I
1	#FormattingInternal::<ItemIDsColumn>#						
2							
3							
4							
5							
6							
7		oid					
8			Cross Section	Length	Weight	Total Length	Total Weight
9			#Mer #Members::Section #Members::Cut #Members::Total #Members::Total #Members::Total #Men				
10							
11						Total:	=SUM(I9:I10)
12							

9. Save and test the report. You will see the correct total at the bottom:

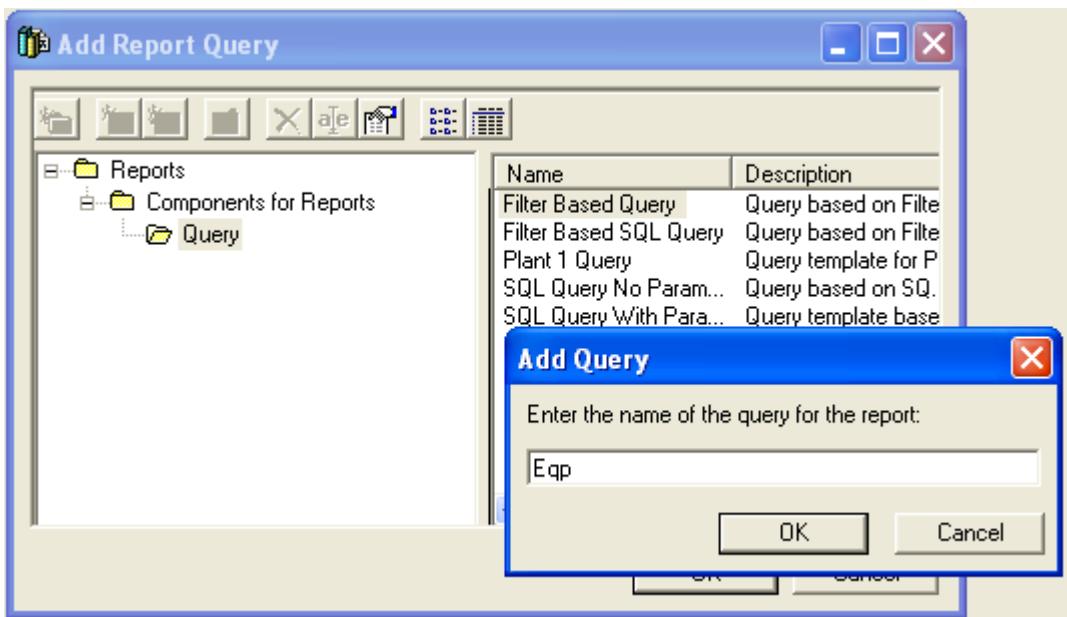
Cross Section	Length	Weight	Total Length	Total Weight	ItemNun
W18X40	6.92785	407.9584	253.1618	14863.10694	29
W14X53	10.3632	802.58983	165.2829237	12038.84744	22
W18X35	6.92785	356.25648	318.6799308	5261.94877	28
W24X68	6.0587255	608.85442	48.472598	4871.116133	34
W14X82	10.3632	1243.3063	35.3314	3217.969263	25
W18X50	6.62305	487.49077	39.75862	2926.440291	30
W16X67	7.586853	748.36943	37.286946	2926.356341	27
W14X74	10.3632	1130.4049	36.79207158	2925.753819	24
C10X15.3	3.624199	80.649677	270.4735933	2295.637995	2
W21X57	6.059424	507.73251	24.237696	2030.930056	32
W14X43	7.8994	493.429	31.5976	1973.715984	20
W14X99	10.3632	1516.9534	13.4112	1963.116227	26
2L4X4X1/2X3/8	2.235935112	84.918991	38.76441589	1472.240883	1
WT8X22.5	7.845565518	260.87714	36.83780825	1224.942422	44
L4X4X1/4	4.924348294	48.314205	71.5795679	702.287841	11
L4X4X1/2	6.096	115.76055	36.576	694.5633166	10
L8X4X1/2	7.957235574	231.69381	15.91447115	463.3876145	14
L2X2X1/8	0.918626492	2.2532275	3.70211709	9.080635351	9
			Total:	61861.44198	

Differential Reporting in SmartPlant 3D

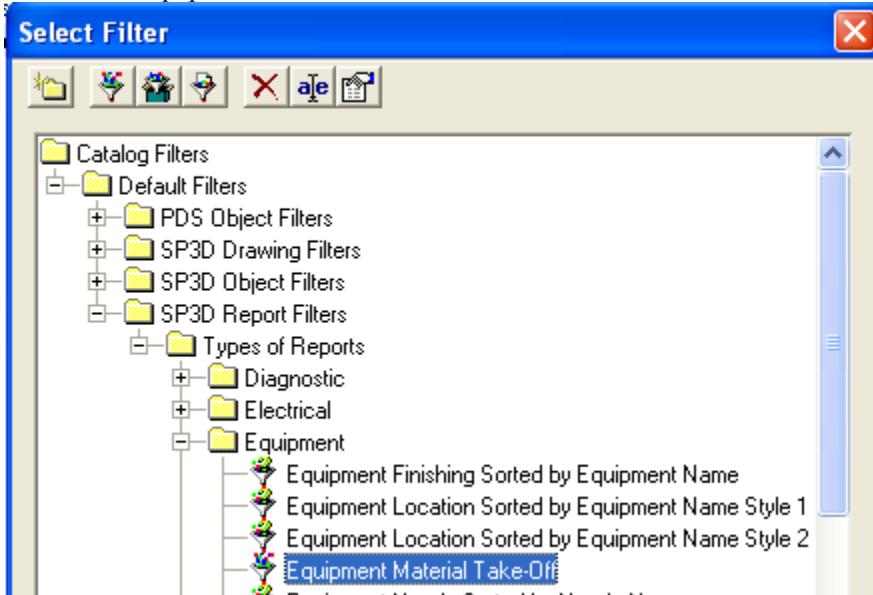
SmartPlant 3D allows users to create a report that will compare against a previously run baseline. Support is available in the user-interface to set up the criteria for comparison.

Creating the report

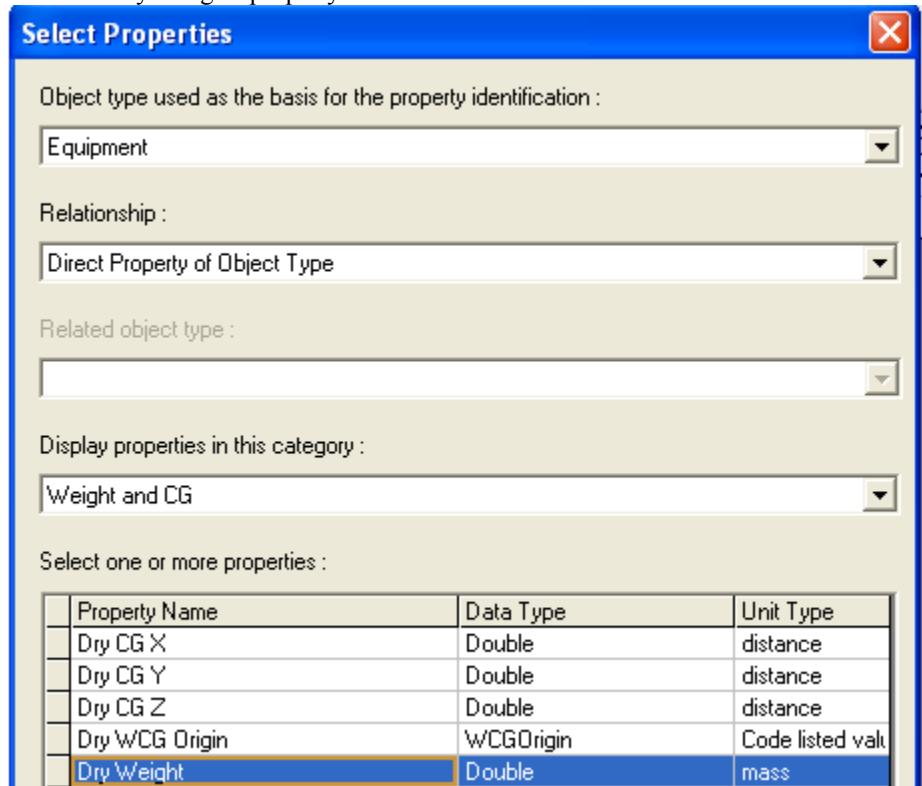
1. Create a new report that uses an empty template.
2. Rename the report ‘Equipment Compare’
3. Edit Template and Tools - Add Query
4. Add a Filter Based Query
5. Name the query ‘Eqp’



6. Choose the 'Equipment Material Take Off' filter as shown



7. Add the 'Dry Weight' property



8. Enter Design Layout

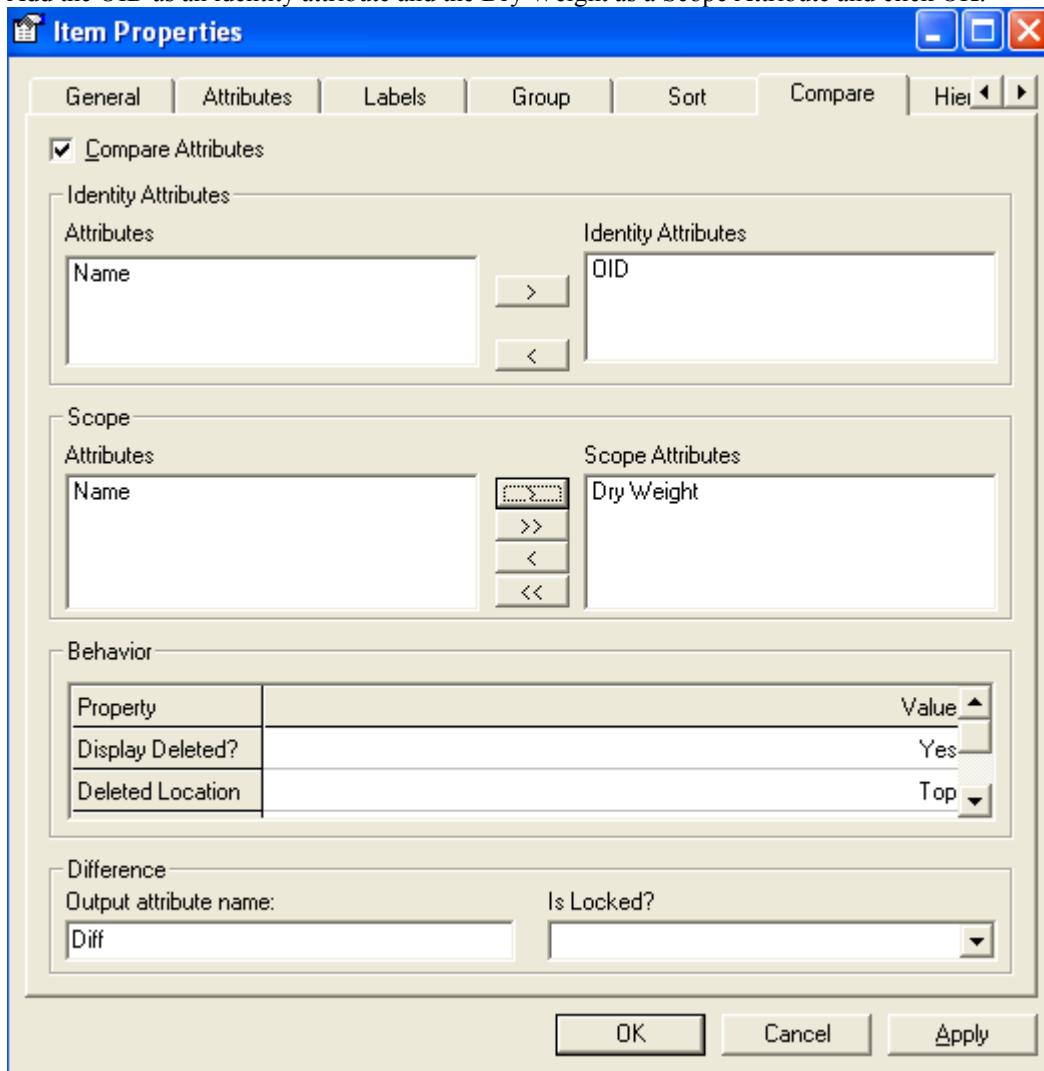
9. Add the Eqp query to Sheet1

10. Drag and drop the properties OID, Name and Dry Weight to columns A, B and C

11. Edit properties of the query 'Eqp' and select the Compare tab

12. Check the box 'Compare Attributes'

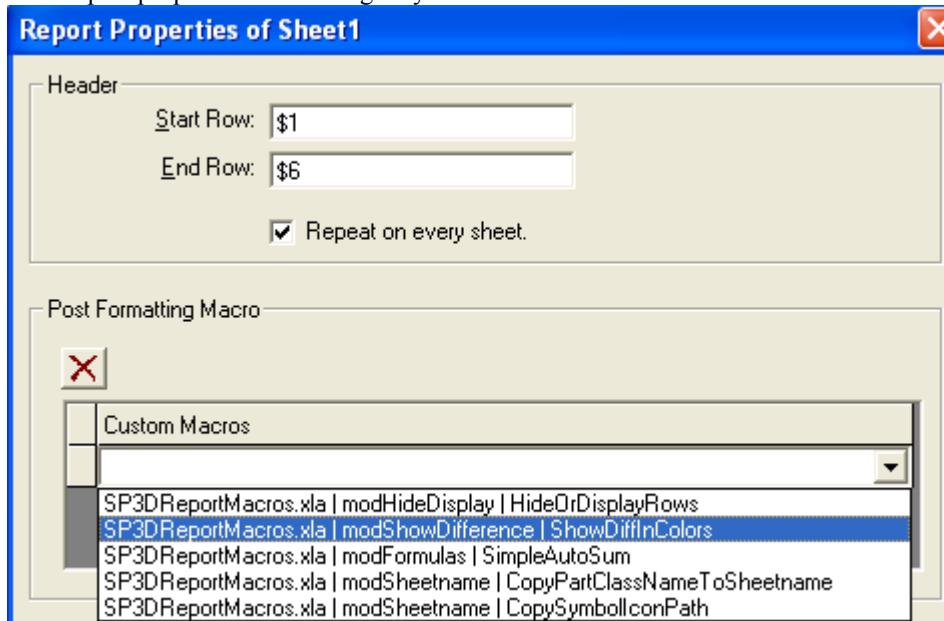
13. Add the OID as an identity attribute and the Dry Weight as a Scope Attribute and click OK.



The identity attribute is the basis for comparison of rows in the old and new reports. The scope attribute is what decides whether a row appears as modified in the report. Since this example has no grouping, it is best to choose the OID as the identity attribute rather than the name since the name could be the same for two equipment, but the OID is guaranteed to be unique. If the dry weight of an equipment changes, it will be highlighted in the report output. If any other property changes, this report will not consider the row as being modified.

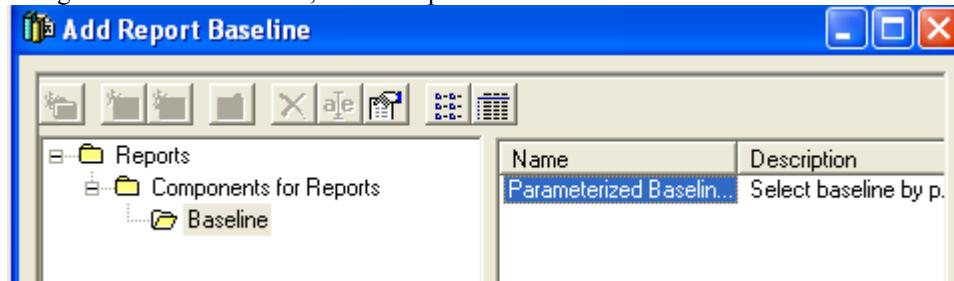
14. A new attribute named 'Diff' has appeared. Drag and drop this onto column D

15. Edit Report properties in the design layout tree and add the macro 'ShowDiffInColors'



16. Close Excel and click 'Yes' when prompted to save.

17. Using Tools – Add Baseline, add the report baseline



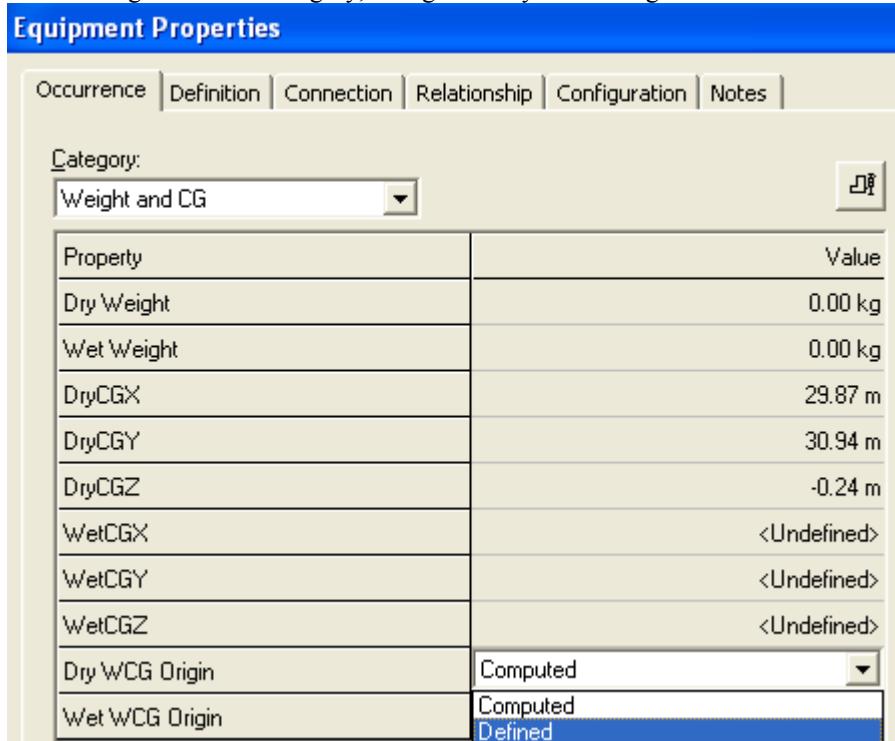
18. Save Report Template and click Finish on the resulting dialog leaving the baseline prompt empty.

19. Copy Report to Catalog under the 'Examples' folder

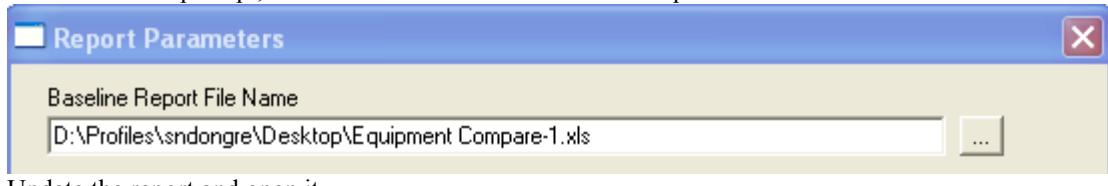
Testing the report

1. Update the Report and save resulting output to Desktop.
2. Rename the file on desktop to Equipment Compare – 1.xls
3. Switch to 'Equipment' task and edit properties on 40V-101

4. In the ‘Weight and CG’ category, change the Dry WCG Origin to ‘Defined’ and click Apply.



5. The Dry Weight field becomes available, enter a value of 100 kg and click OK.
 6. Switch to the ‘Drawings and Reports’ task.
 7. Right-click the ‘Equipment Compare’ and select ‘Parameters’
 8. At the baseline prompt, browse to the file saved on the desktop and click Finish.



9. Update the report and open it.
 10. Notice that column E shows a ‘U’ for every row that is unchanged, except the row for 40V-101
 11. There are two rows for 40V-101, a shaded row with an ‘O’ and a new row with an ‘N’

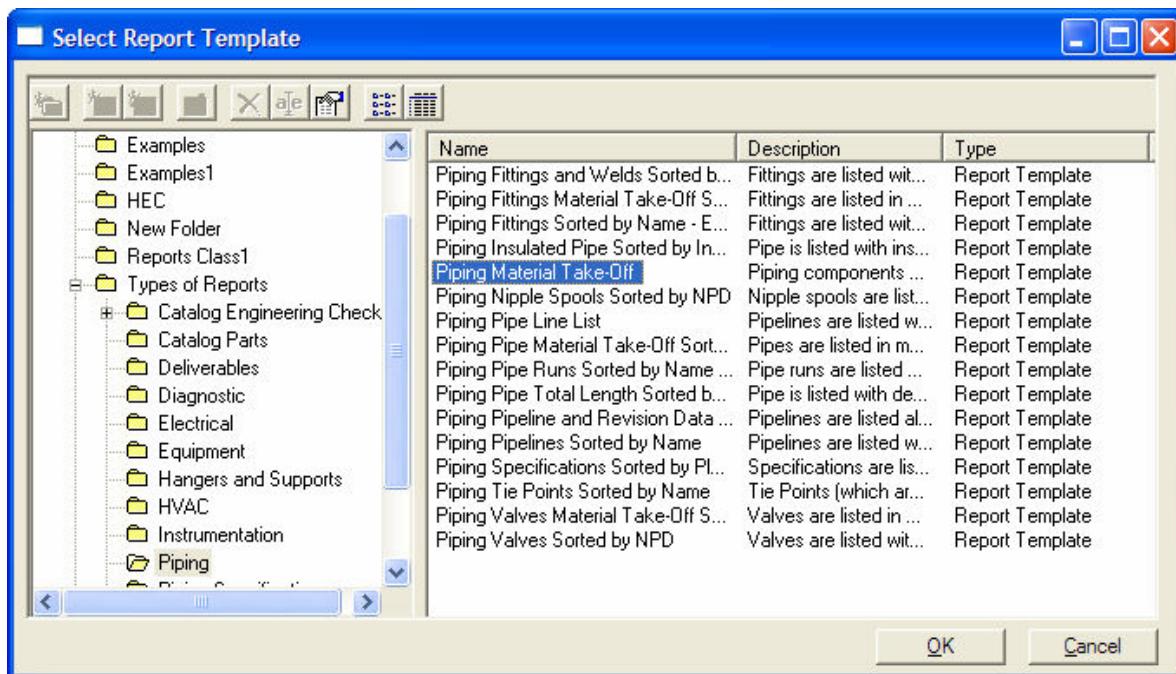
24	(00004E2E-0000-0000 41P-101A	740	U
25	(00004E2E-0000-0000 41P-101B	740	U
26	(00004E2E-0000-0000 40E-101B	0	U
27	(00004E2E-0000-0000 40V-101	0	O
28	(00004E2E-0000-0000 40V-101	100	N
29	(00004E2E-0000-0000 T-101	0	U
30	(00004E2E-0000-0000 E-102		U

12. Similarly you may expect deleted rows shown with a ‘D’ and new rows with an ‘A’

Hierarchical Reporting

We'll modify the delivered Piping Material Take-Off report to make it a report that reports by pipeline.

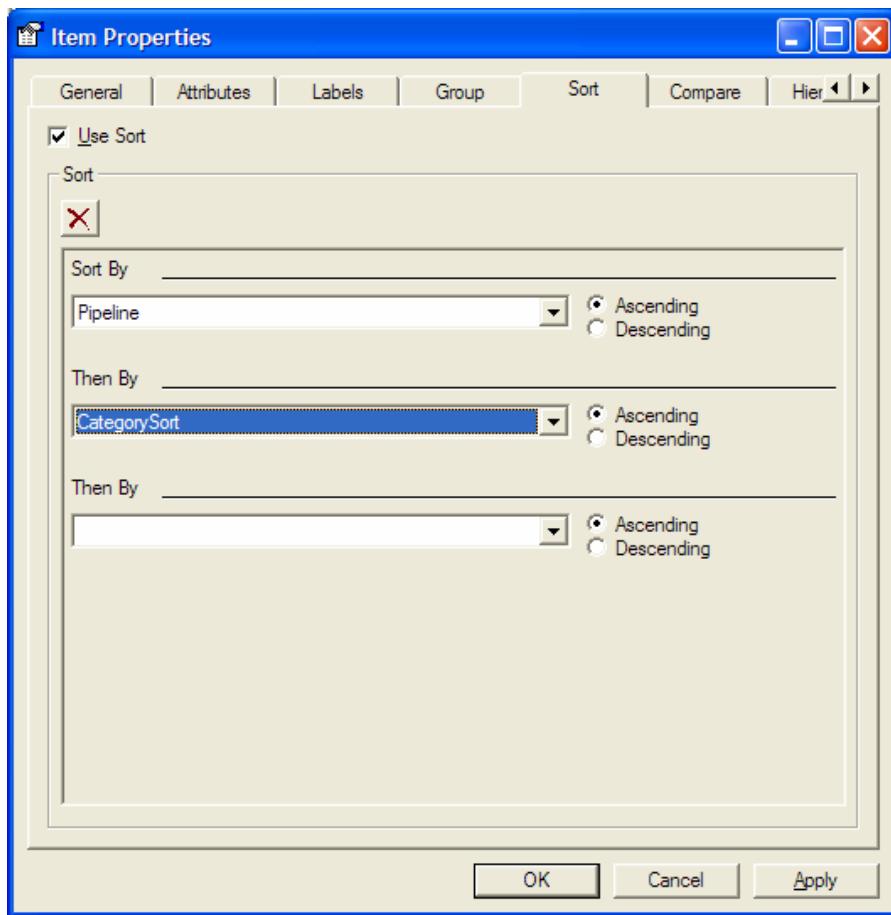
1. Create a new report. Instead of starting from an empty template, select Types of Reports\Piping\Piping Material Take-Off.



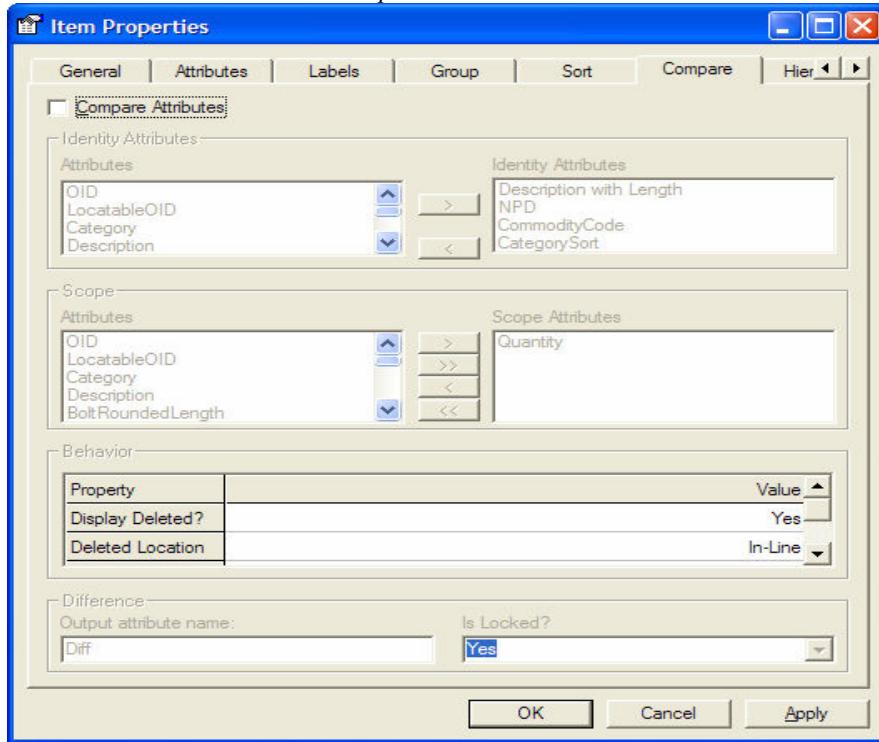
2. Select the System Root node when prompted by the filter dialog. Click “Next” at the baseline prompt, and “Finish” at the UOM dialog.
3. Edit Template and go into the Design Layout. Insert a column before the *Item Number* Column (E).
4. Drag Pipeline attribute to cell E3

Material Take-Off.xls [Compatibility Mode]			
	E	F	G
1	Pipeline	Item Number	Category
2			
3	#Component::Pipeline#	#Component::ItemNum#	#Component::Category#
4			
5			

5. Right-click *Component/Properties*.
6. Under the *Group* tab add *Pipeline* to the attributes to Group By
7. Under the *Sort* tab add “*Pipeline*” as the first attribute to sort by and “*Category Sort*” as the second attribute to sort by.

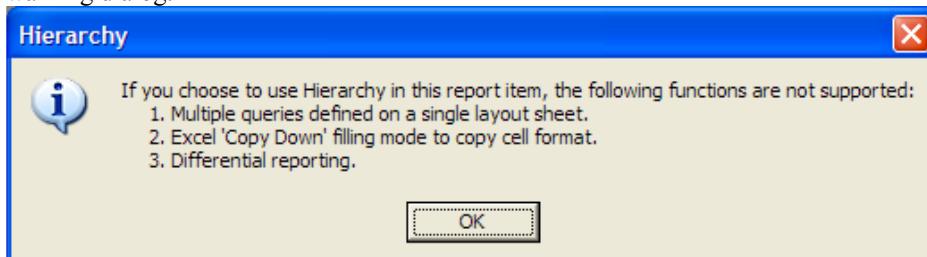


8. Remove the check box on the *Compare* tab

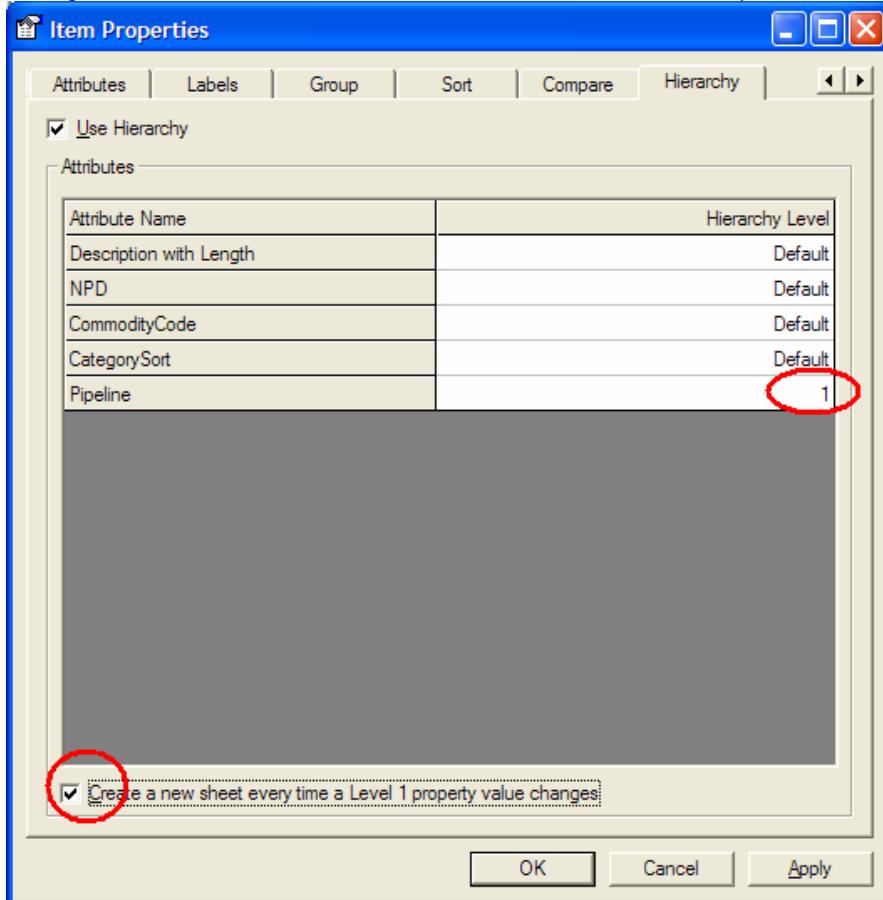


9. Click OK to save settings

10. Edit query properties again (Step 5) and check the check box “*Use Hierarchy*” on the *Hierarchy* tab. Click OK on the warning dialog:



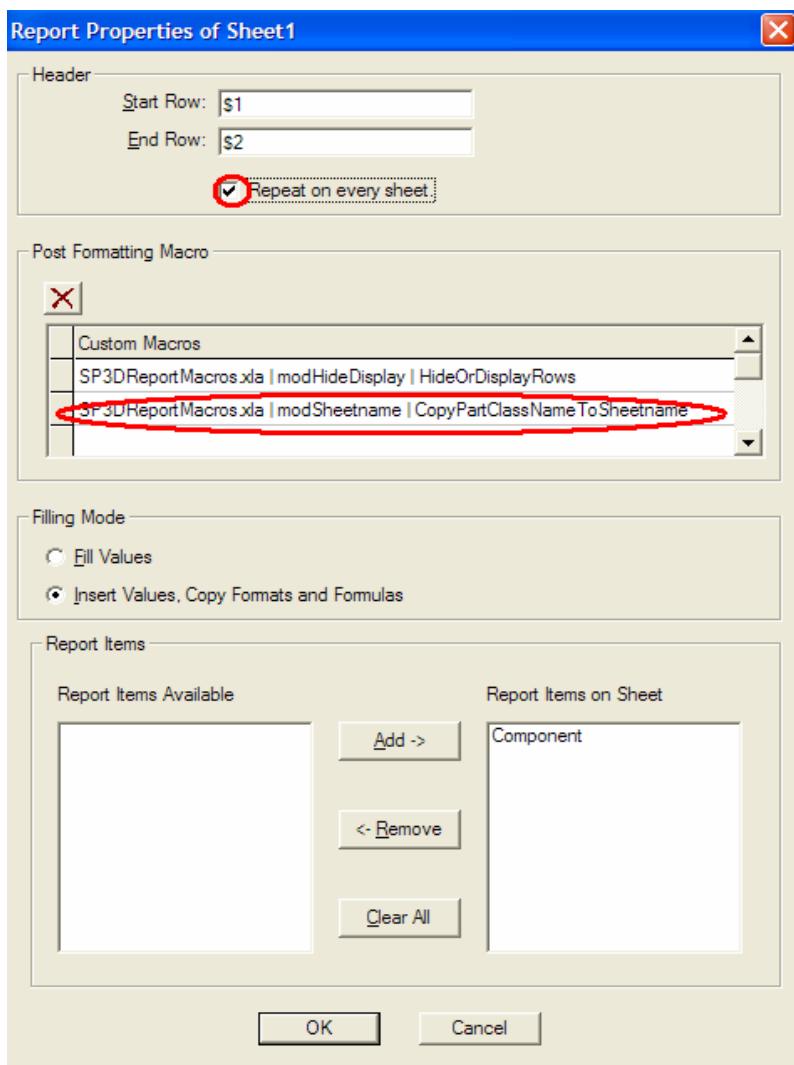
11. Set Pipeline as a Level 1 attribute, and check “*Create new sheet every time a Level 1 property value changes*”



When using Hierarchy, the value for a cell in a given row and a given column will be skipped (not output) according to the rules:

- If that column attribute is Hierarchy level *Default* – never.
- If that column attribute is Hierarchy level k – when all the values in the present row for columns of hierarchy levels between 1 and k including are the same as the values for those same columns in the preceding row.

12. Click OK to save settings. This will create and output the information for each pipeline on a separate sheet, and that pipeline name will be output only once.
 13. On Report Properties remove the Macro ShowDiffInColors . Add the macro CopyPartClassNameToSheetname. Also check the box to repeat header on every sheet.



14. Close excel, save when prompted.
15. Save report template and copy report to catalog
16. Open SP3DReportMacros.xla in Symbols\Reports\Components for Reports>Addins and edit the Line 18 on modSheetName to be cell 3, E (to match cell in step 2 above) also, comment out line that sets cell 6A to blank and save the xla file. Your code should look like

```

Public Sub CopyPartClassNameToSheetname()
    Dim oDataSheet As Worksheet
    Dim sText As String
    On Error GoTo ErrorHandler

    For Each oDataSheet In Application.Worksheets
        If oDataSheet.Visible = xlSheetVisible Then
            If oDataSheet.Name <> "SP3DReport_Layout" And _
                oDataSheet.Name <> "SP3DReport_Definition" And _
                oDataSheet.Name <> "Index" And _
                oDataSheet.Name <> "CustomInterfaces" And _

```

```

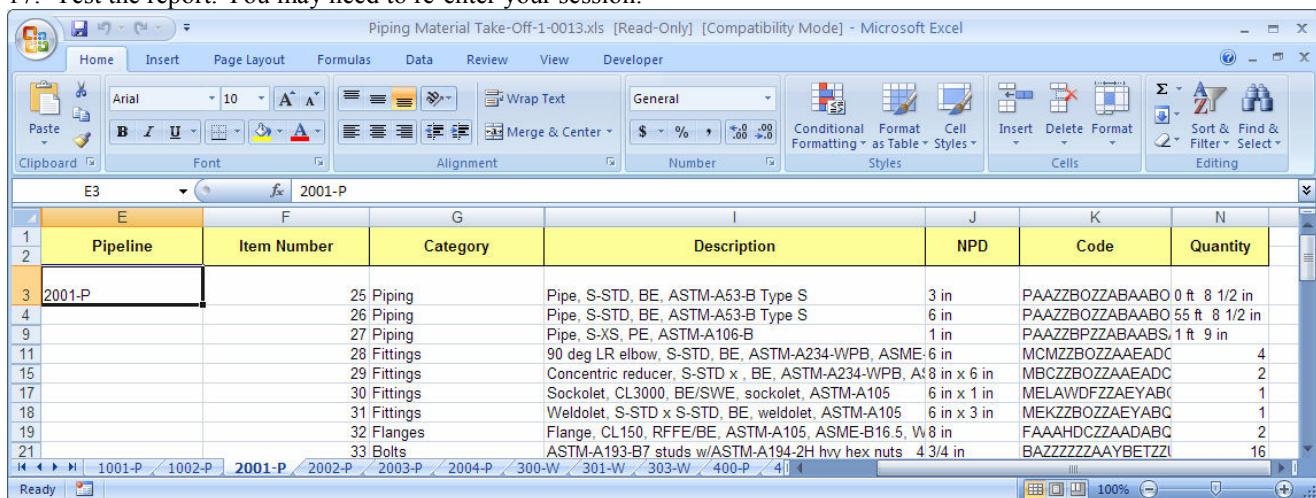
oDataSheet.Name <> "GUIDs" Then
With oDataSheet
    sText = .Cells(3, "E") 'where the parameter was placed in step 1
    m_PartClassName = sText
    .Name = sText
    '.Cells(6, "A") = ""' if you do not like the partclassname in somewhere except on
sheet name
End With
End If
End If
Next
GoTo ShutDown
ErrorHandler:

ShutDown:
End Sub

```

This way the macro will name each sheet with the name of the corresponding pipeline.

17. Test the report. You may need to re-enter your session.

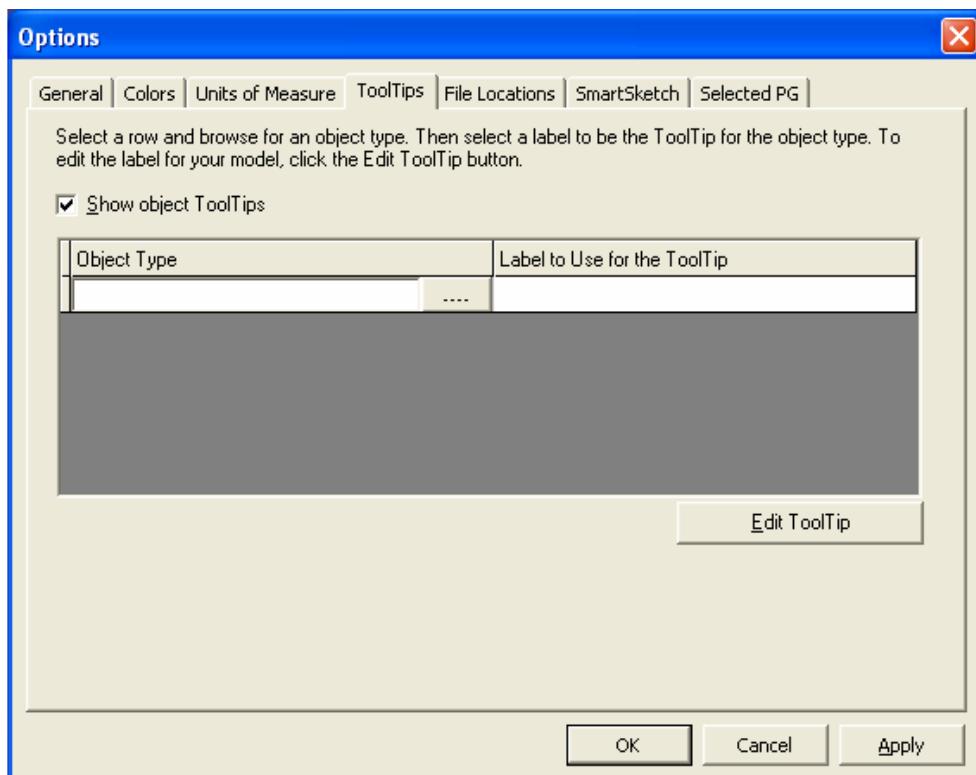


	E	F	G	I	J	K	N
1	Pipeline	Item Number	Category	Description	NPD	Code	Quantity
3	2001-P	25	Piping	Pipe, S-STD BE, ASTM-A53-B Type S	3 in	PAAZBBOZZABAABO	0 ft. 8 1/2 in
4		26	Piping	Pipe, S-STD BE, ASTM-A53-B Type S	6 in	PAAZBBOZZABAABO	55 ft. 8 1/2 in
9		27	Piping	Pipe, S-XS, PE, ASTM-A106-B	1 in	PAAZBPZZABAABS	1 ft. 9 in
11		28	Fittings	90 deg LR elbow, S-STD, BE, ASTM-A234-WPB, ASME	6 in	MCMZZBBOZZAAEADC	4
15		29	Fittings	Concentric reducer, S-STD x , BE, ASTM-A234-WPB, A	8 in x 6 in	MBCZZBBOZZAAEADC	2
17		30	Fittings	Sockolet, CL3000, BE/SWE, sockolet, ASTM-A105	6 in x 1 in	MELAWDFZZAEYABC	1
18		31	Fittings	Weldolet, S-STD x S-STD, BE, weldolet, ASTM-A105	6 in x 3 in	MEKZZBBOZZAEYABC	1
19		32	Flanges	Flange, CL150, RFFE/BE, ASTM-A105, ASME-B16.5, W	8 in	FAAAHDCZZAADABC	2
21		33	Bolts	ASTM-A193-B7 studs w/ASTM-A194-2H hvy hex nuts	4 3/4 in	BAZZZZZZAAYBETZZI	16

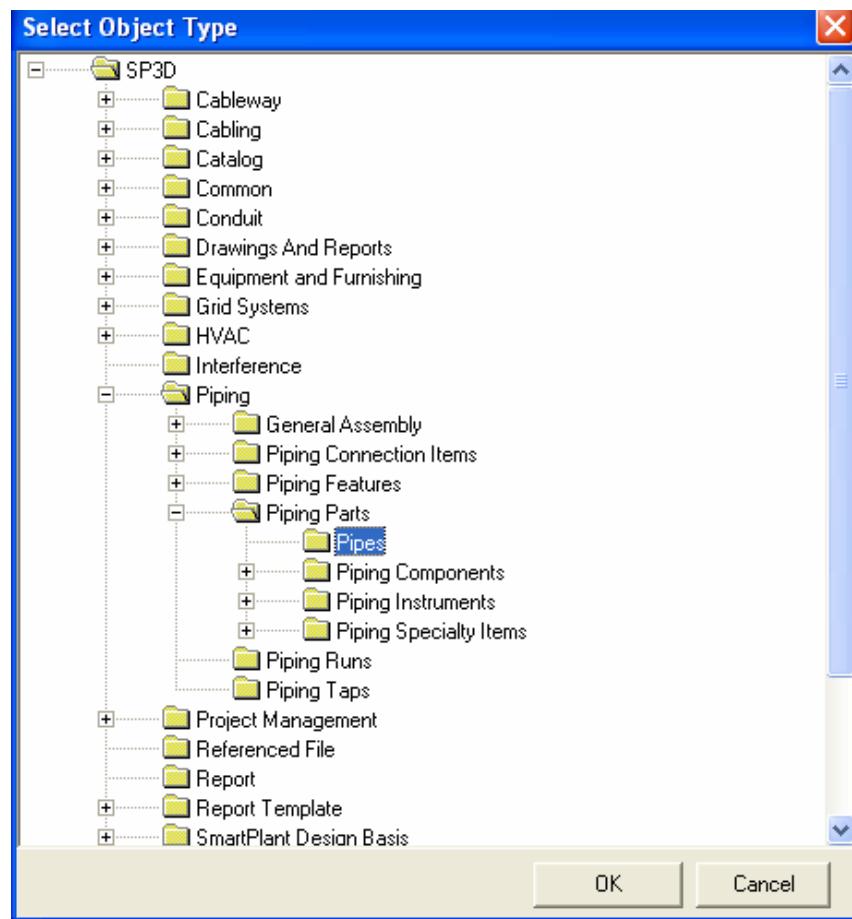
Labels

Tool Tip Labels

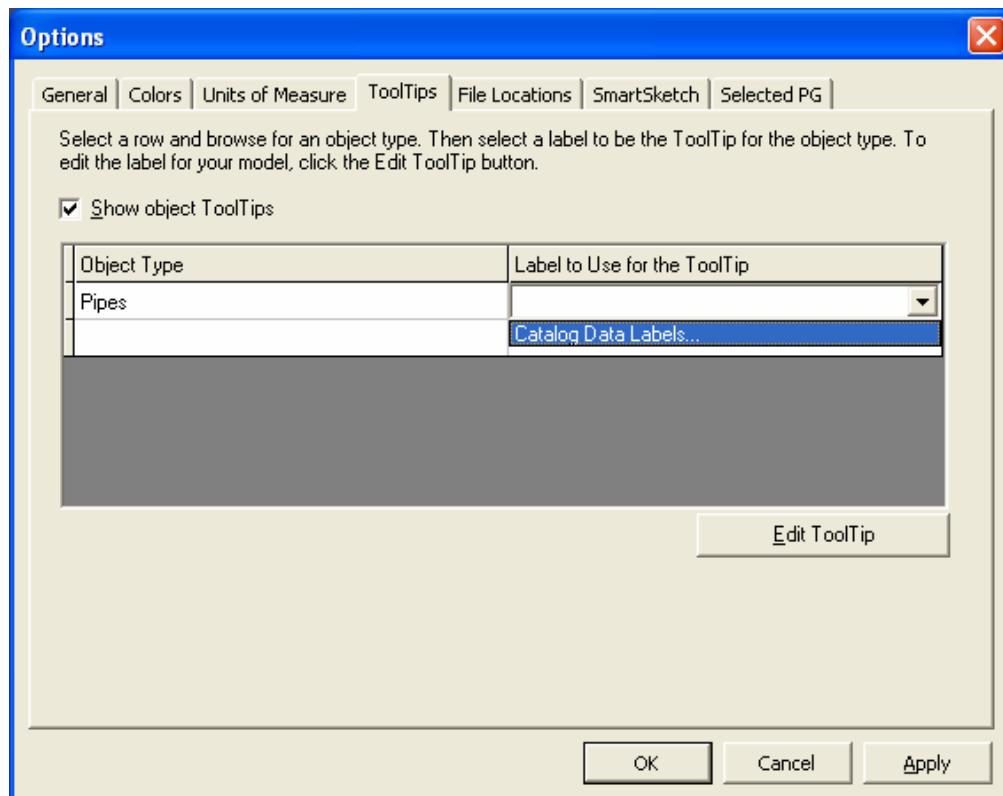
1. From within any 3D environment (Common for example) proceed to Tools->Options, and then on the Tooltips tab you will find the following:



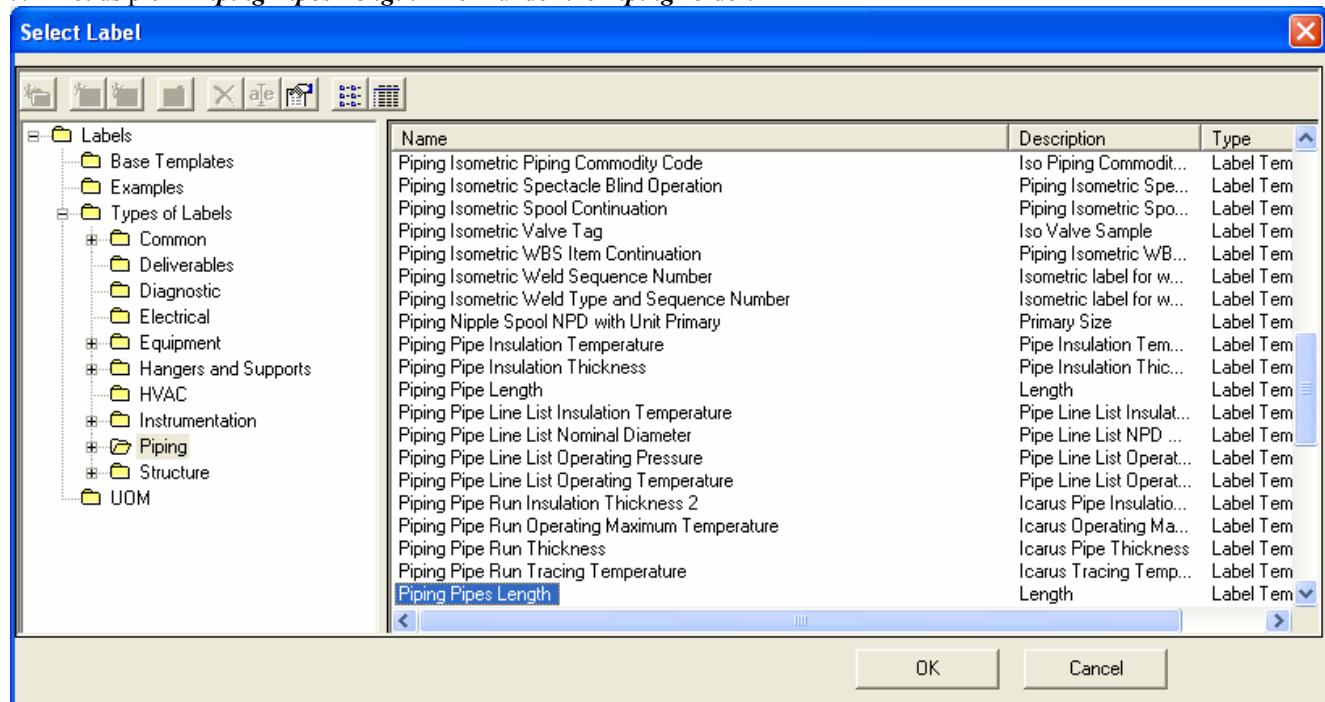
2. Select the ellipse button
3. Parse through the Object tree and pick the particular object type that you wish the Tool Tip to be active against.
4. In this example, let us pick Pipes.



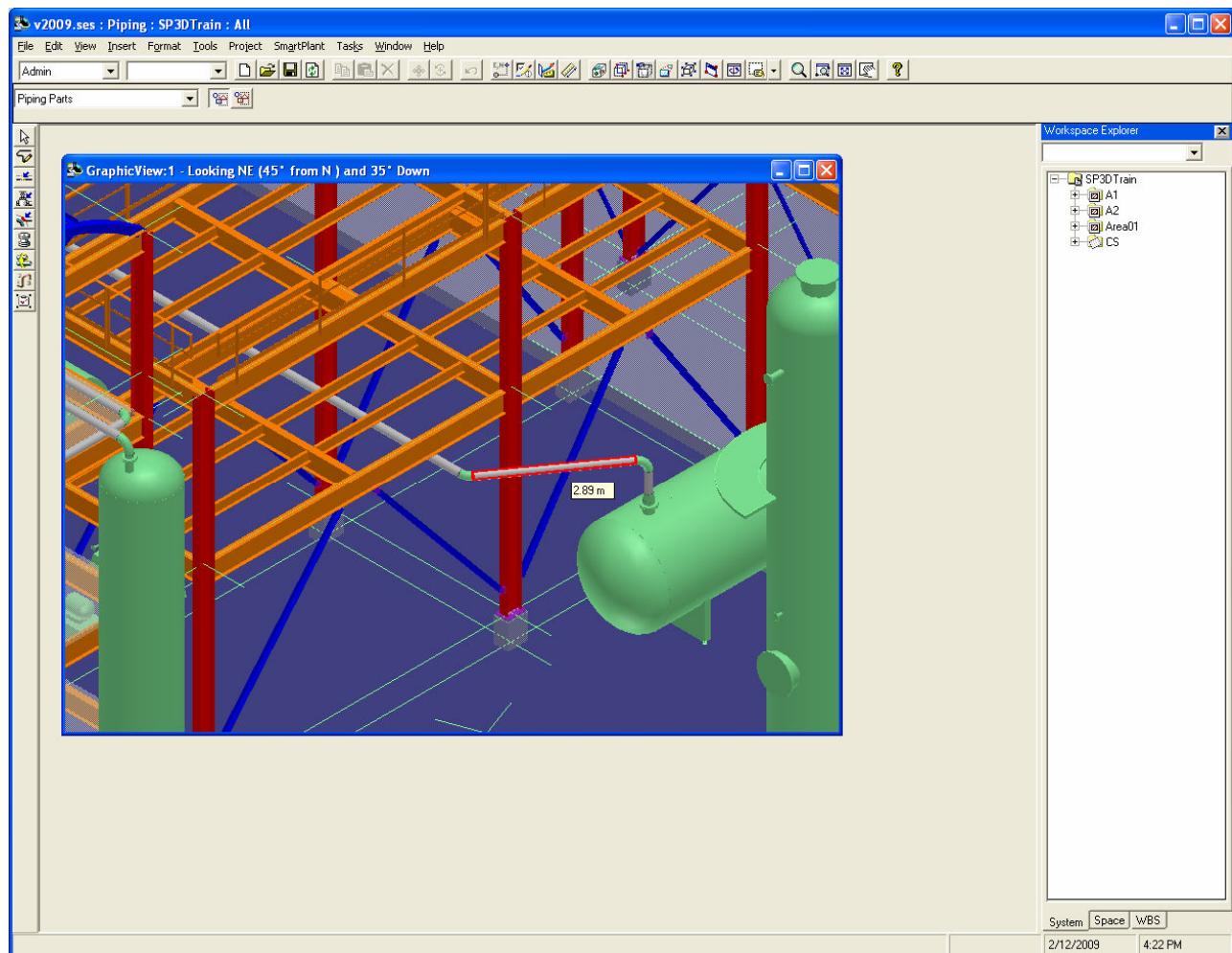
5. From the drop down list under “Label to use...” pick “***Catalog Data Labels ...***”.



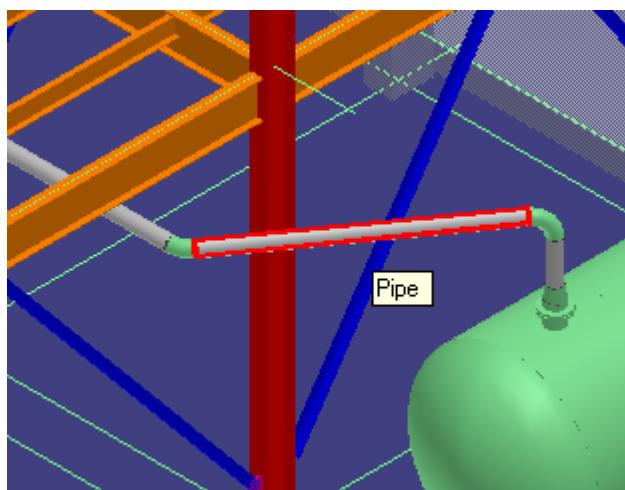
6. Please note, that this listing below could contain any labels you have in the Catalog.
7. Let us pick "**Piping Pipes Length**" from under the **Piping** folder:



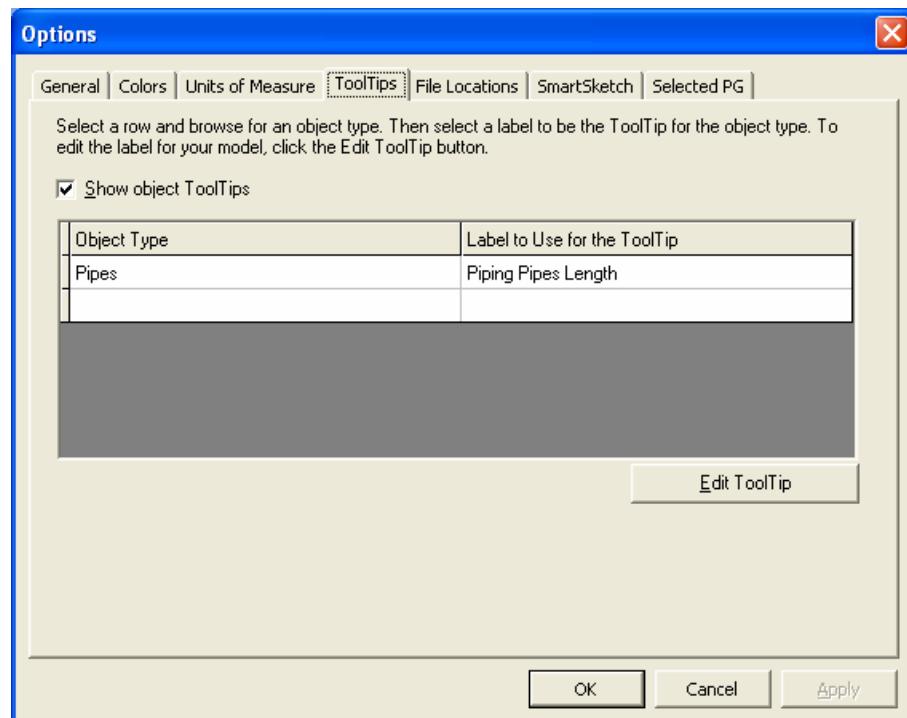
8. Select the OK button.
9. Hit apply on the Options form
10. Now switch to the Piping Task, Set you selection filter to "Piping Parts" and observe that the tooltip is applying the label when you hover over a pipe:



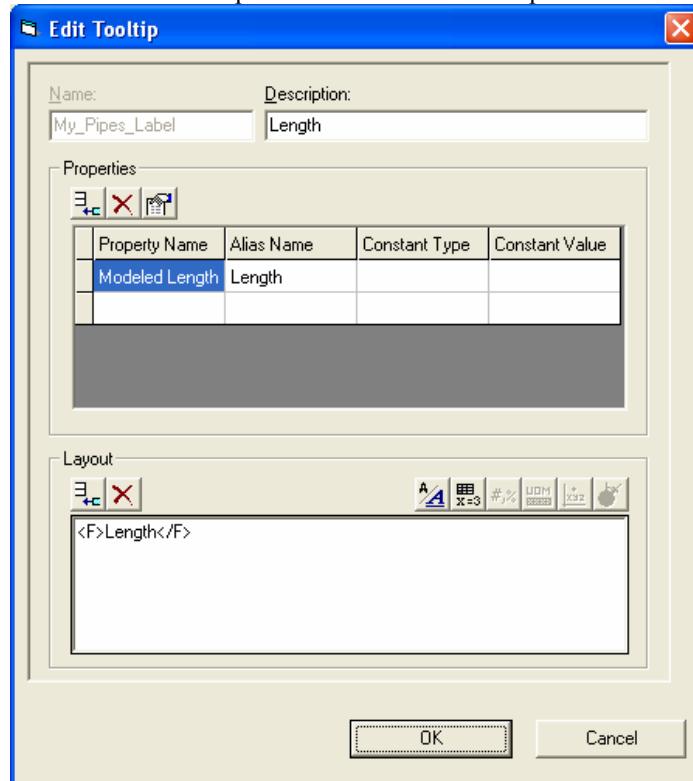
11. Go back to Tools → options, on the Tooltips tab, select the row that we created and then hit the Delete key to delete the tooltip.
12. Observe that when you apply the hover workflow again that the result is the original default Tooltip:



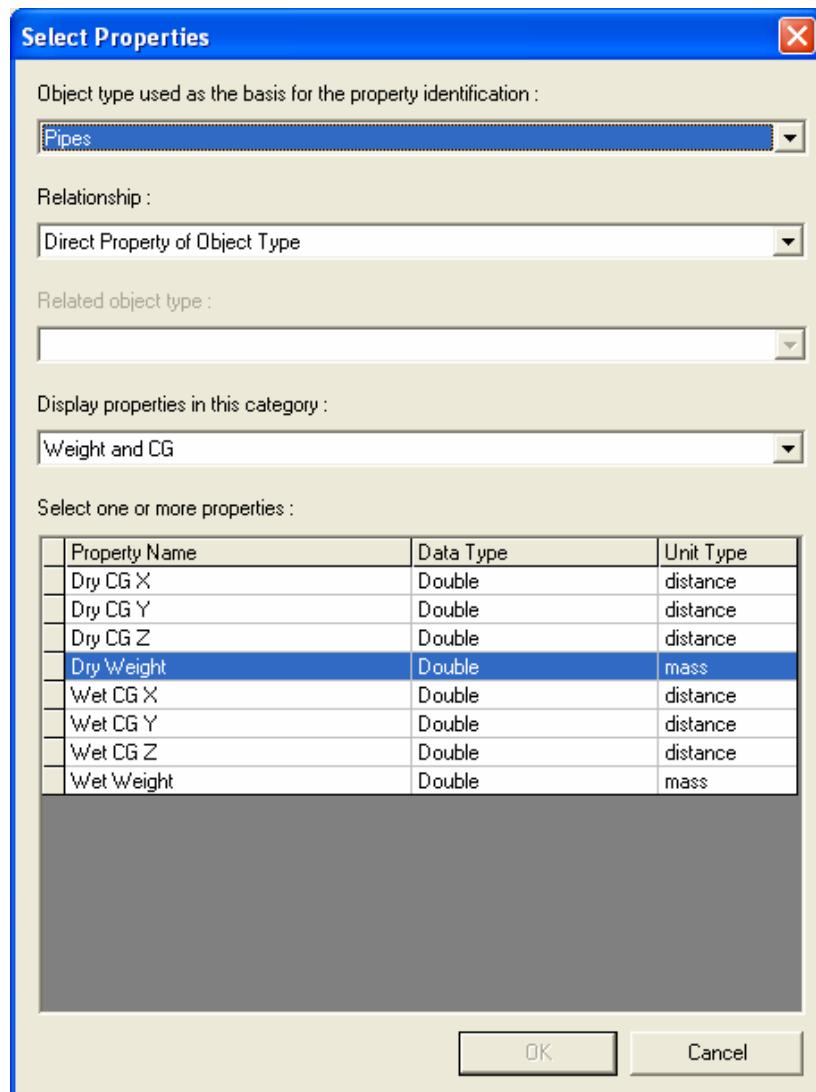
13. Repeat the previous steps to make the Tooltip again, once you have gotten to point screenshotted below, click the Edit ToolTip button.



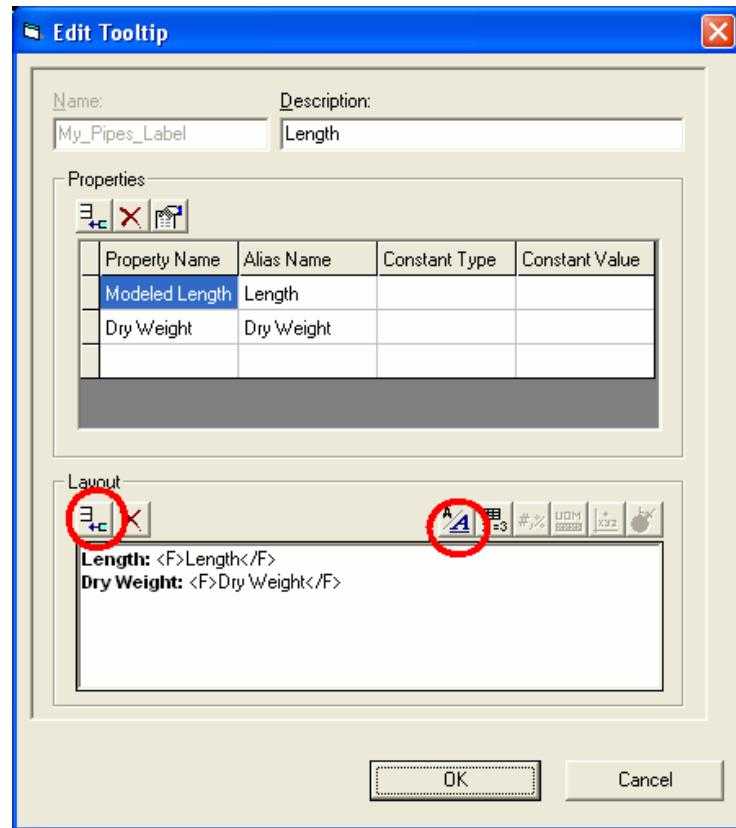
14. The following form will appear to assist you in editing the label. Notice that this form functions very much like the Filter Data Query that we have seen in the Edit Template... environment for Reports.



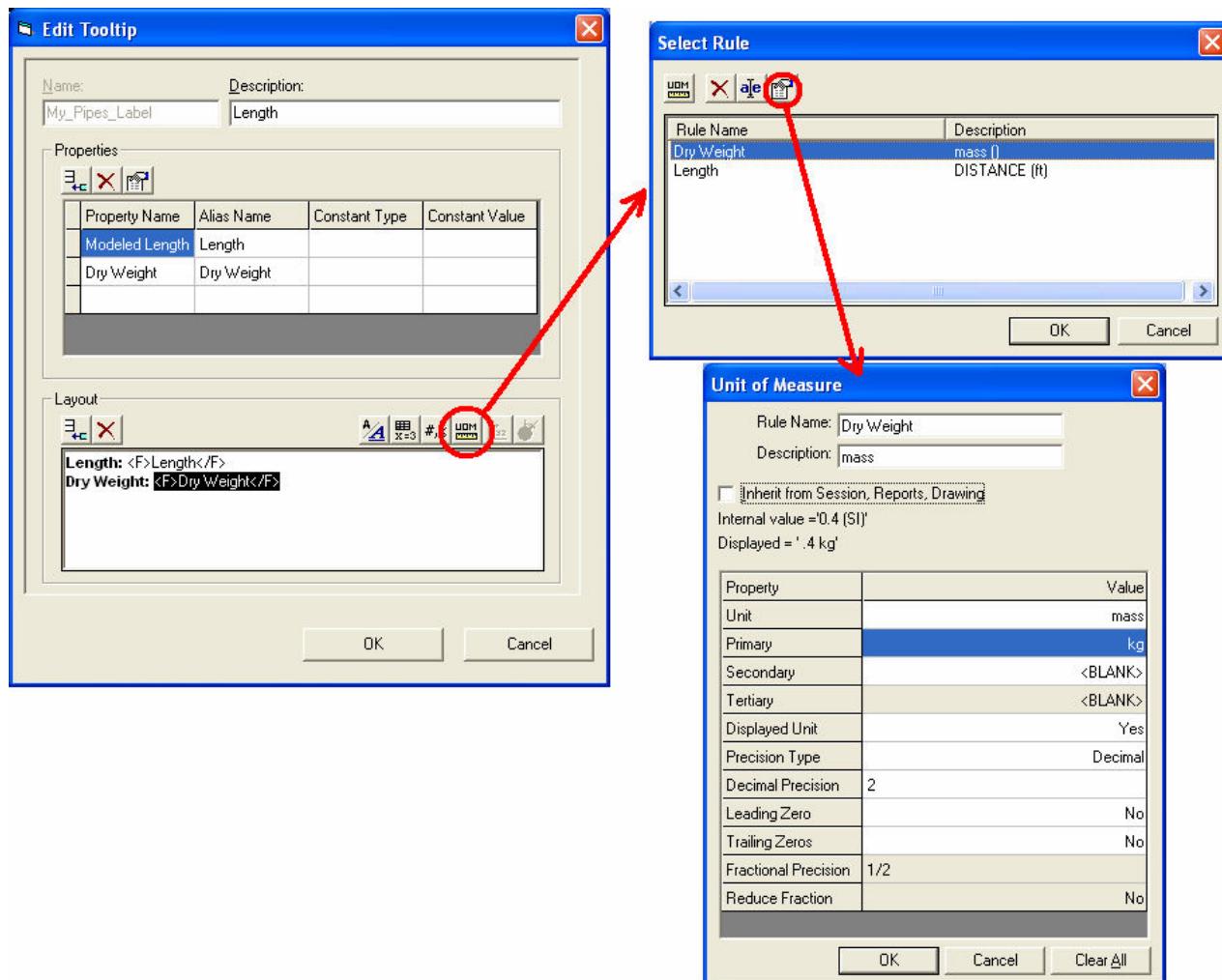
15. Use the add functionality to add Dry Weight information:



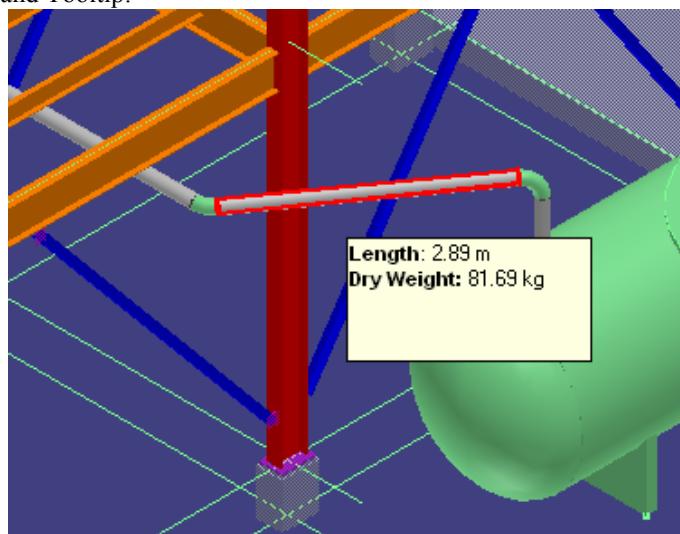
16. You can then use the “Insert a field to layout” button as well as the Rich Text Button and produce the following:



17. Highlight the <F>Dry Weight</F> and change the UOM for the Dry Weight value to use Primary Units = kg and Precision of 2.



18. Hit “OK” on the “**Unit of Measure**”, “OK” on the “**Select Rule**”, and “OK” on the “**Edit Tooltip**” forms.
19. Test out your new Label and Tooltip.



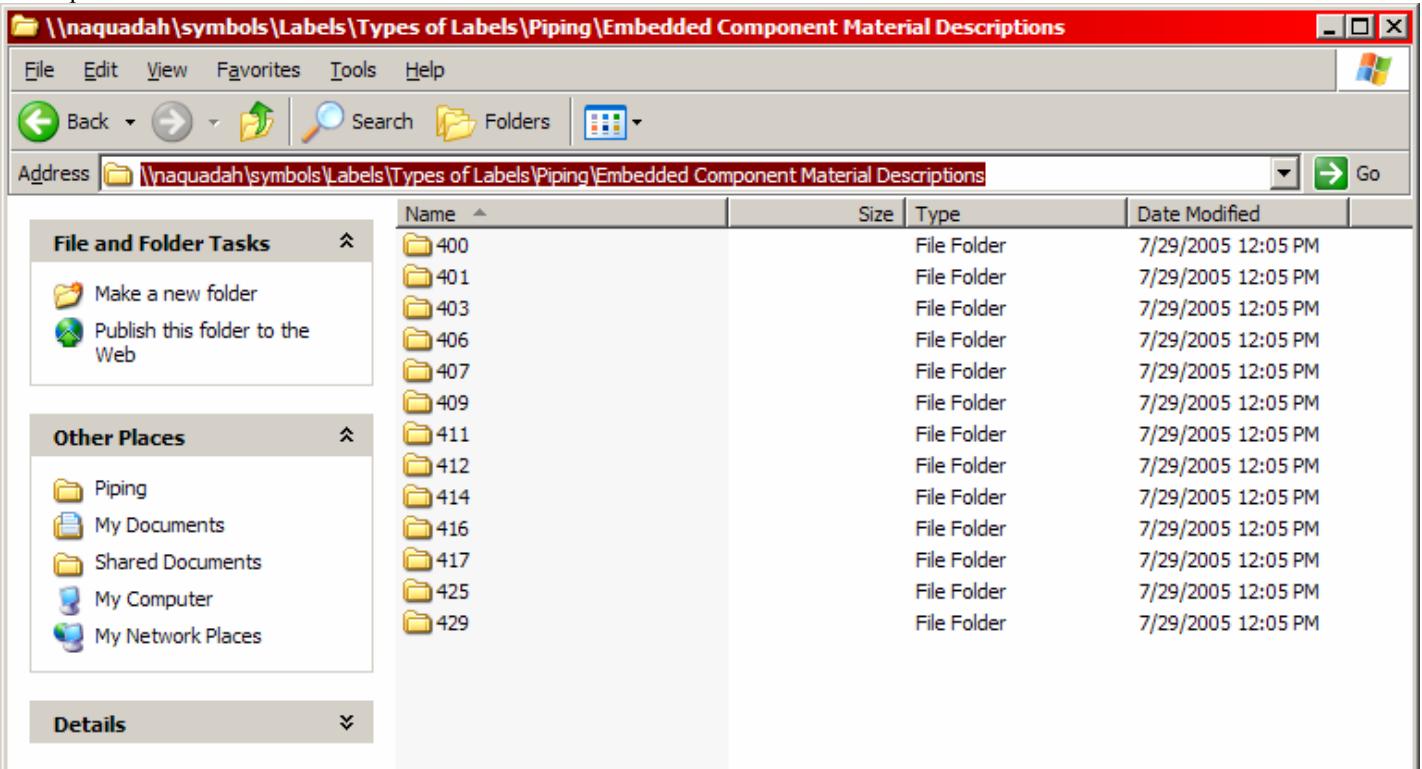
Embedded Labels

Open the Piping Specification.xls Workbook (located at “<installation folder>\CatalogData\Bulkload\DataFiles”) and switch to the “PipingCommodityMatlControl” sheet. Observe in the Material Description field data the existence of [???] such as “[409]”

	D	E	F	G	H	I	J	K	L	M	N
1	FirstSizeTo	FirstSizeUnits	SecondSizeFrom	SecondSizeTo	SecondSizeUnits	MultisizeOption	IndustryCommodityCode	ClientCommodityCode	CIMISCommodityCode	ShortMaterialDescription	
2											
4										Monitor, CL150 FFFE, station type, 4" CL150 in-let by 2.5" NHT stainless steel outlet w/stai	
5										Monitor, CL150 FFFE, station type 4" CL150 in-let by 2.5" NHT stainless steel outlet w/stai	
6										Monitor, CL150 FFFE, elevated type, free standing, 4" CL150 in-let by 2.5" NHT stainless s	
7										Monitor, CL300 FFFE, elevated type, supported, 6" CL300 in-let by 2.5" NHT stainless stee	
8										Fire hydrant, CL125 FFFE, 5" size, counterclockwise open, 4.5" steamer nozzle, two 2.5"	
9										Hose rack, 300#, FTE, w/valve, wall mount, rt hand w/100 ft hose & fog nozzle, Powhatan	
10										Spray sprinkler, MTE, filled cone w/rupture disc, 304, Grinnell, Mulsifyre Projector S-1	
11										Flange, CL150, FFFE/BE, ASTM-A105, ANSI-B16.5, WN, [409]bore to match	
12										Flange, CL150, FFFE/BE, ASTM-A105, ANSI-B16.5, WN, cement lined, [409]bore to match	
13										Flange, CL300, FFFE/BE, ASTM-A105, ANSI-B16.5, WN, [409]bore to match	
14										Flange, CL150, RFFE/BE, ASTM-A182-F304, ANSI-B16.5, WN, S-80S bore	
15										Flange, CL150, RFFE/BE, ASTM-A182-F316, ANSI-B16.5, WN, S-80S bore	
16										Flange, CL150, RFFE/BE, ASTM-B166-600, ANSI-B16.5, WN, S-80S bore	
17										Flange, CL150, RFFE/BE, ASTM-A105, ANSI-B16.5, WN, S-160 bore	

This 409 corresponds to a label of the same name:

Open a windows explorer window and go to “<symbols>\Labels\Types of Labels\Piping\Embedded Component Material Descriptions”



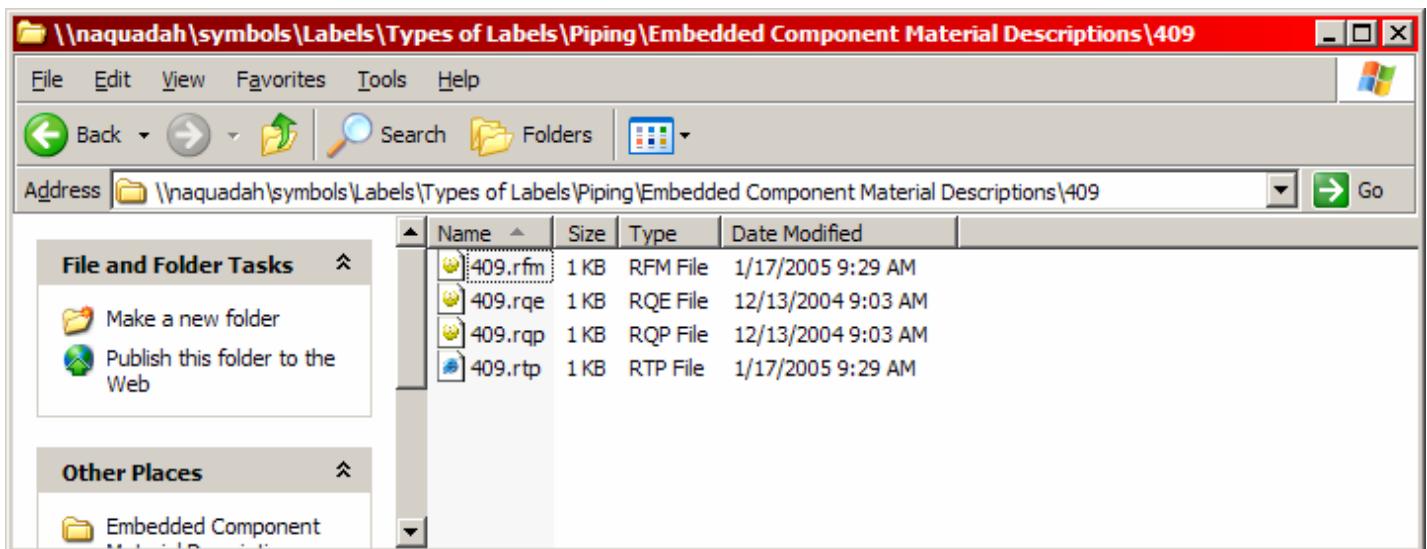
Here you will find a folder corresponding to the field value “409” This label will be evaluated and will produce information in the design location where the [409] string exists.

It will only be evaluated when appropriate: that is whenever a material description is processed for a label (tool-tip, drawing label, etc).

Note: that any string can be defined to construct an embedded label with the format [labelname] being used in the material description.

Important: Currently SP3D does not locate the label according to hierarchy, but instead locates it in the Catalog by name. For this reason, it is important at this time to keep all Label names unique in the Catalog even if they exist in different locations in the Catalog hierarchy.

Looking at the 409 Label folder you will find 1 rfm,, 1 rqe, 1 rqp, and 1 rpt.



Looking at the XML for the Label:

Open the RTP:

```

<?xml version="1.0" encoding="windows-1252" ?>
<REPORT_TEMPLATE
  Name="409"
  Description="Comp sch/thk 2 b">

  <QUERIES>
    <QUERY  Name="409"
      Site="User"
      Description="Query to get the schedule thickness from straight pipe port 2"
      Path="409.rqe"/>
  </QUERIES>

  <FORMATTING
    Name="409"
    Site="User"
    Description="Comp sch/thk 2 b format"
    Path="409.rfm"/>

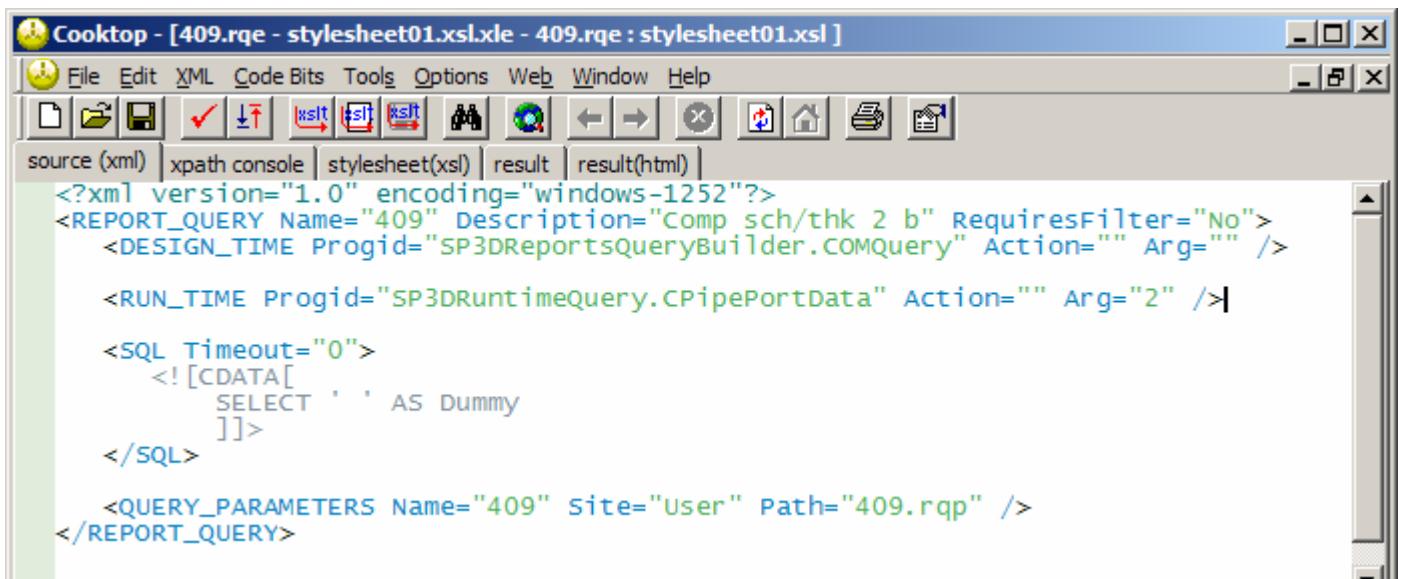
</REPORT_TEMPLATE>

```

The screenshot shows the Cooktop application interface with the following details:

- Title Bar:** Cooktop - [409.rtp - stylesheet01.xsl.xle - 409.rtp : stylesheet01.xsl]
- Menu Bar:** File, Edit, XML, Code Bits, Tools, Options, Web, Window, Help
- Toolbar:** Standard toolbar icons.
- Tool Buttons:** source (xml), xpath console, stylesheet(xsl), result, result(html).
- Code Editor:** Displays the XML code for the report template.
- Status Bar:** Done, PC, XSLT: MSXML, Ln 1, Col 0, READ.

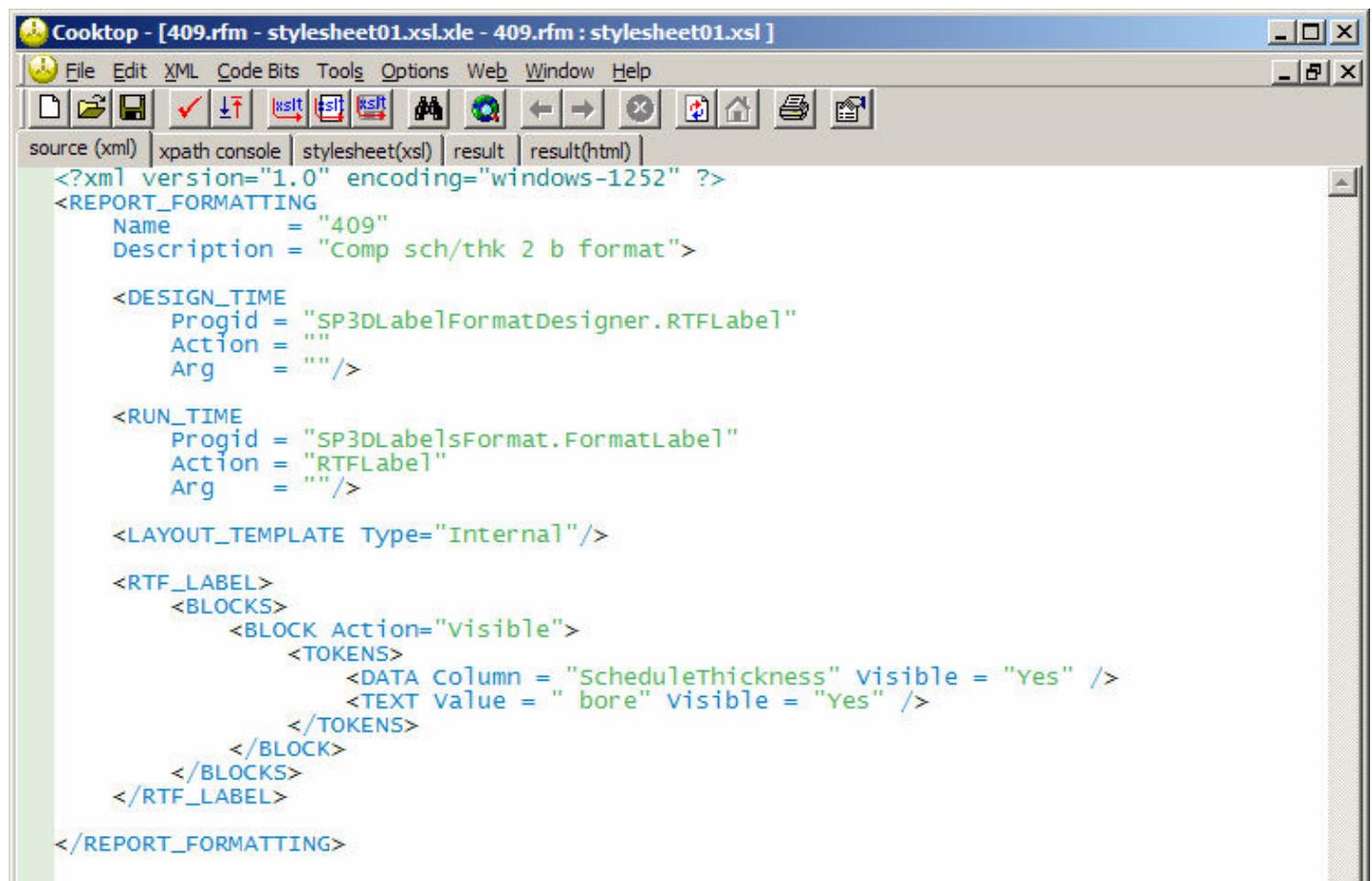
Open the RQE:



The screenshot shows the Cooktop application window. The title bar reads "Cooktop - [409.rqe - stylesheet01.xsl.xle - 409.rqe : stylesheet01.xsl]". The menu bar includes File, Edit, XML, Code Bits, Tools, Options, Web, Window, and Help. Below the menu is a toolbar with various icons. The main area displays an XML document structure. The XML code is as follows:

```
<?xml version="1.0" encoding="windows-1252"?>
<REPORT_QUERY Name="409" Description="Comp_sch/thk 2 b" RequiresFilter="No">
  <DESIGN_TIME Progid="SP3DReportsQueryBuilder.COMQuery" Action="" Arg="" />
  <RUN_TIME Progid="SP3DRuntimeQuery.CPipePortData" Action="" Arg="2" />
  <SQL Timeout="0">
    <! [CDATA[
      SELECT '' AS Dummy
    ]]>
  </SQL>
  <QUERY_PARAMETERS Name="409" Site="User" Path="409.rqp" />
</REPORT_QUERY>
```

Open the RFM:



The screenshot shows the Cooktop application interface with the title bar "Cooktop - [409.rfm - stylesheet01.xsl.xls - 409.rfm : stylesheet01.xls]". The menu bar includes File, Edit, XML, Code Bits, Tools, Options, Web, Window, and Help. Below the menu is a toolbar with various icons. The main window displays the XML code for a report format:

```

<?xml version="1.0" encoding="windows-1252" ?>
<REPORT_FORMATTING
    Name      = "409"
    Description = "Comp sch/thk 2 b format">

    <DESIGN_TIME
        Progid = "SP3DLabelFormatDesigner.RTFLabel"
        Action = ""
        Arg    = ""/>

    <RUN_TIME
        Progid = "SP3DLabelsFormat.FormatLabel"
        Action = "RTFLabel"
        Arg    = ""/>

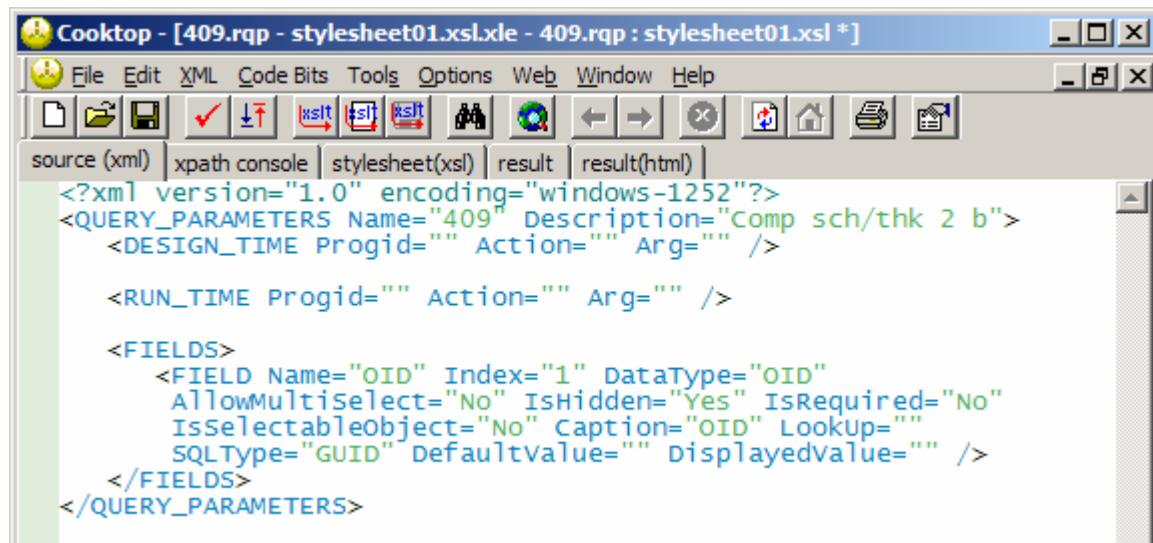
    <LAYOUT_TEMPLATE Type="Internal"/>

    <RTF_LABEL>
        <BLOCKS>
            <BLOCK Action="visible">
                <TOKENS>
                    <DATA Column = "ScheduleThickness" Visible = "Yes" />
                    <TEXT value = " bore" Visible = "Yes" />
                </TOKENS>
            </BLOCK>
        </BLOCKS>
    </RTF_LABEL>

</REPORT_FORMATTING>

```

Open the RQP:



The screenshot shows the Cooktop application interface with the title bar "Cooktop - [409.rqp - stylesheet01.xsl.xls - 409.rqp : stylesheet01.xls *]". The menu bar includes File, Edit, XML, Code Bits, Tools, Options, Web, Window, and Help. Below the menu is a toolbar with various icons. The main window displays the XML code for a query parameters file:

```

<?xml version="1.0" encoding="windows-1252"?>
<QUERY_PARAMETERS Name="409" Description="Comp sch/thk 2 b">
    <DESIGN_TIME Progid="" Action="" Arg="" />

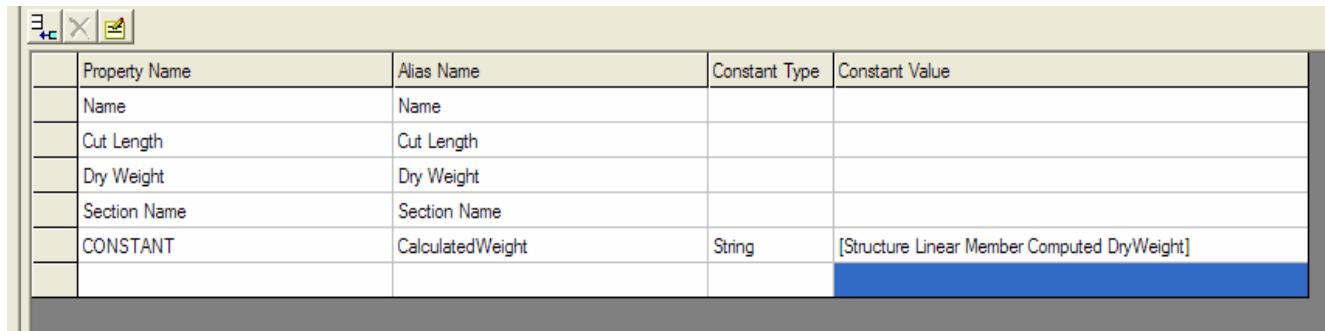
    <RUN_TIME Progid="" Action="" Arg="" />

    <FIELDS>
        <FIELD Name="OID" Index="1" DataType="OID"
            AllowMultiSelect="No" IsHidden="Yes" IsRequired="No"
            IsSelectableObject="No" Caption="OID" Lookup=""
            SQLType="GUID" DefaultValue="" DisplayedValue="" />
    </FIELDS>
</QUERY_PARAMETERS>

```

Now let's add a calculated weight = **Unit Weight x Length** for each member in our "Structure Members" report (the *Dry Weight* is based on this calculated weight, but it excludes the weight of trims and holes). There is a delivered label that does this for us: <Root> Labels\Types of Labels\Structure\Structure Linear Member Computed DryWeight. We don't care how this label works, we just know that it gives us what we want, and we want to incorporate this information in our report with minimum effort. This is a good situation for using embedded labels.

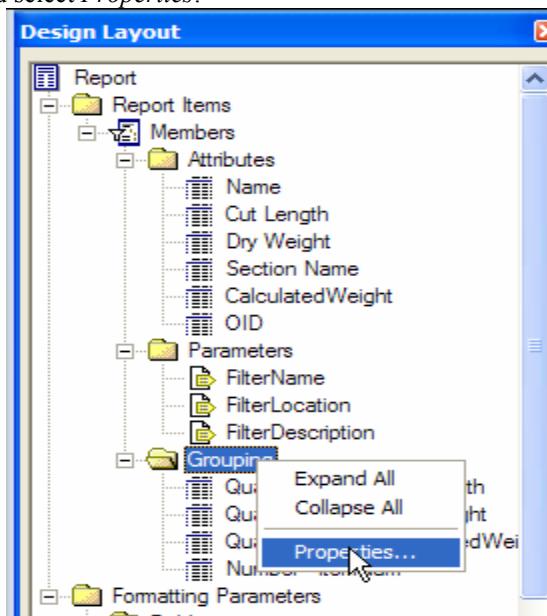
1. Open the "Structure Members" report for editing.
2. Click on the next available row and select "CONSTANT" in the drop-down combo for *Property Name*. Change the *Alias Name*, *Constant Type*, and *Constant Value* as shown.



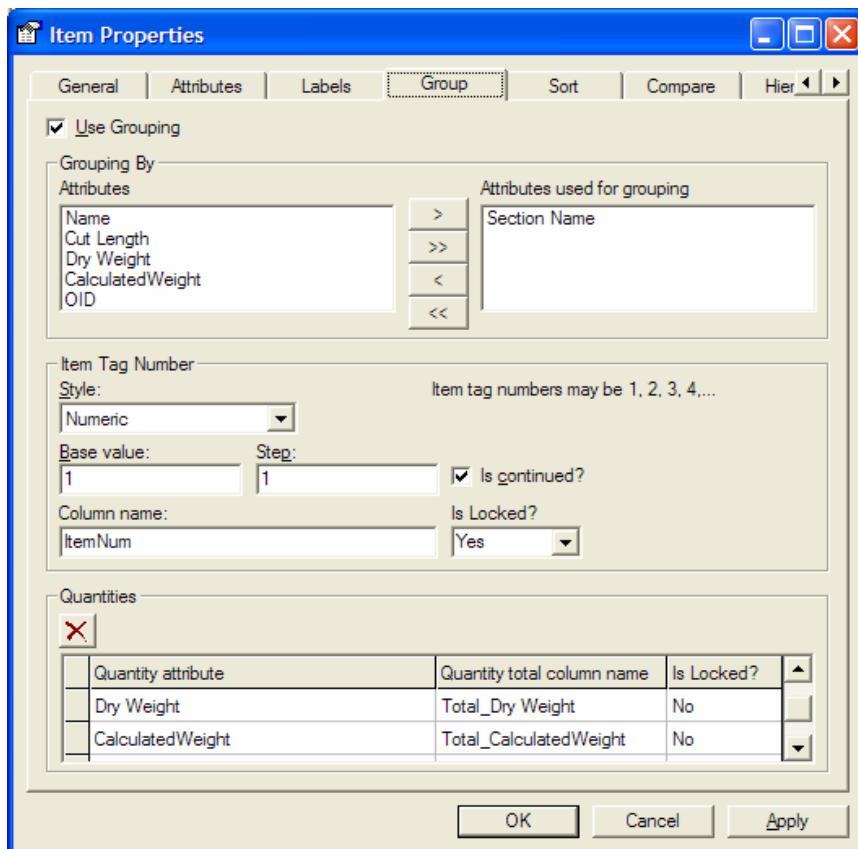
	Property Name	Alias Name	Constant Type	Constant Value
	Name	Name		
	Cut Length	Cut Length		
	Dry Weight	Dry Weight		
	Section Name	Section Name		
	CONSTANT	CalculatedWeight	String	[Structure Linear Member Computed DryWeight]

Important: *Constant Value* contains within [] the exact name of an existing Catalog label. If the name is misspelled you will not get the expected result. Another important point to keep in mind is that the embedded label has to expect the exact same object type as the report is reporting on (Structure Member Parts in our case). This is because unlike labels that do only formatting, embedded labels also have to extract the properties.

3. Enter the Design Layout. Since we have made our "Structure Members" to report on groups of members (grouped by *Cross Section* name), we'll probably want our *Calculated Weight* summed for each group also. Right-click on the "Grouping" node and select *Properties*:



4. Add *Calculated Weight* in the "Quantities":



5. Drag and drop "Quantity – Total_CalculatedWeight", save and test.

Note: If you open the "Structure Linear member Computed DryWeight.rfp file, you'll see that the used weight UOM is lbm – change this if you need different units.

Structure Linear Member Computed DryWeight.rfp - stylesheet01.xsl.xle - Struct

source (xml) xpath console stylesheet(xsl) result result(html)

```
<?xml version="1.0" encoding="UTF-8"?>
<FORMATTING_PARAMETERS
  Name="Structure Linear Member Computed Dryweight"
  Description="">
  <DESIGN_TIME
    Progid=""
    Action=""
    Arg="" />

  <RUN_TIME
    Progid="SP3DPrompts.ct1Tab"
    Action=""
    Arg="" />

  <UOMS>
    <UOM
      Name="weight"
      CanInherit="Yes"
      Type="Mass"
      Primary="1bm" 1bm
      Secondary=""
      Tertiary=""
      UnitsDisplayed="No"
      PrecisionType="Decimal"
      DecimalPrecision="2"
      FractionalPrecision=""
      LeadingZero="yes"
      TrailingZeros="yes"
      ReduceFraction="" />
  </UOMS>
</FORMATTING_PARAMETERS>
```

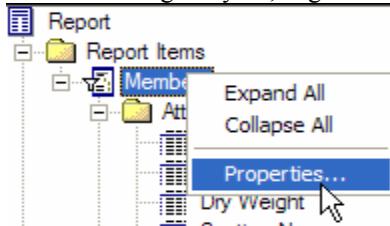
Special Formatting in Reports

Using New Labels

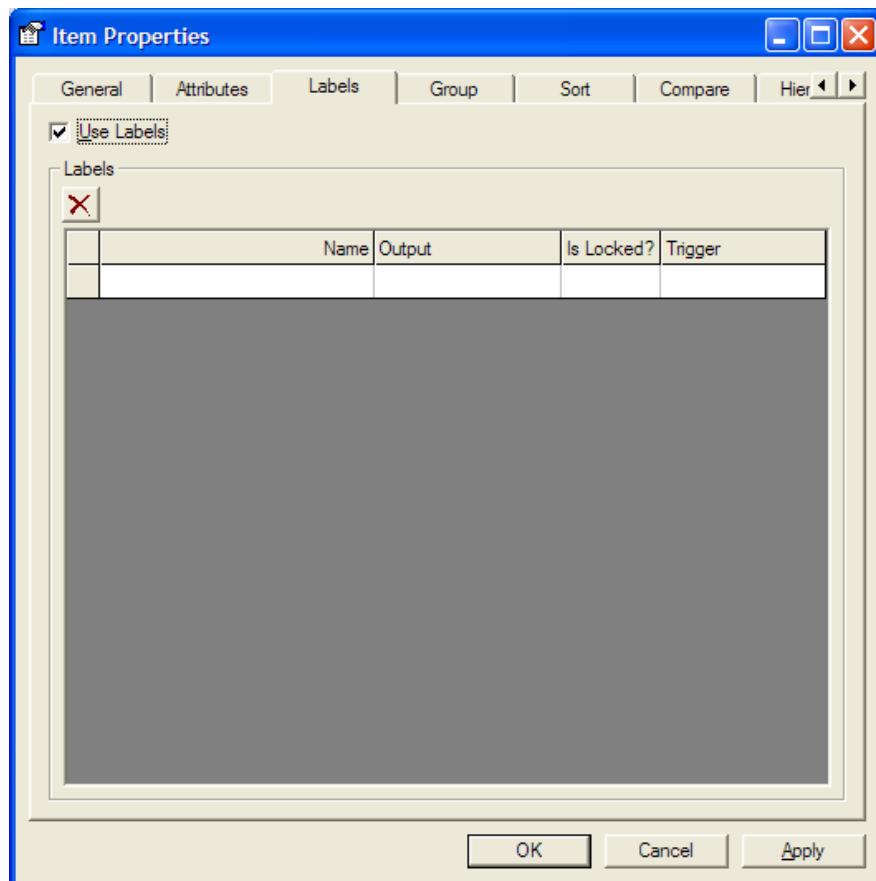
Formatting labels are those labels created for use on the design layout (.xls). They can be used to present the data and change the formatting to suit the appropriate need. Unlike Embedded labels, they do not extract any properties, they are just applied to the properties available in the report.

Let's create a label that will format the weight of the individual members in our "Structure Members" report:

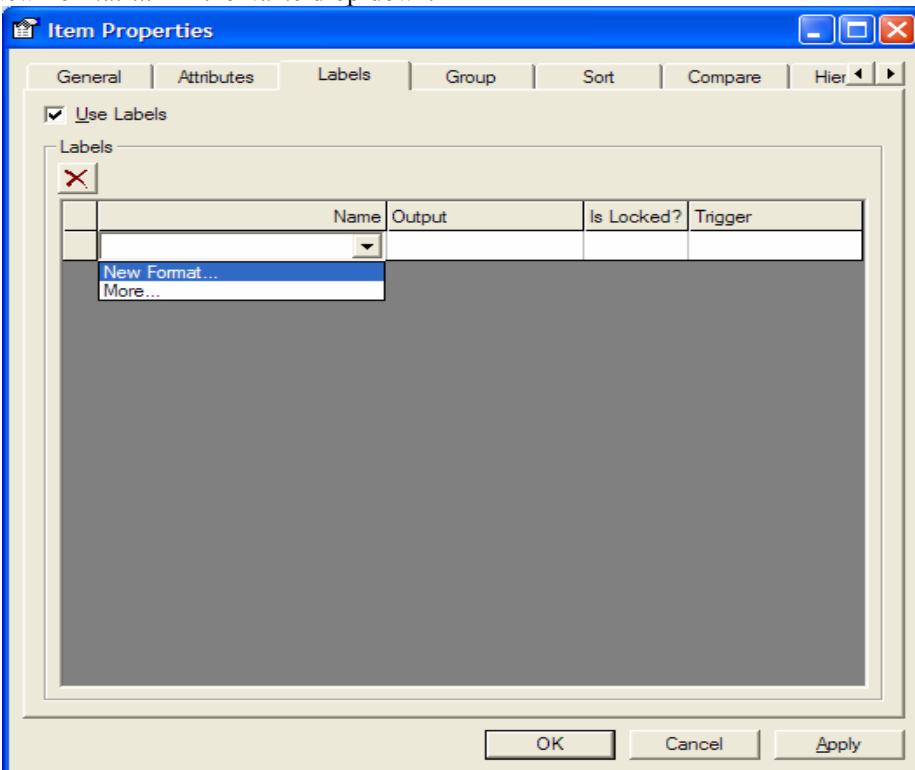
1. Open the "Structure Members" report template for editing.
2. Go in the Design Layout, Right-Click on the "Members" node and select *Properties*



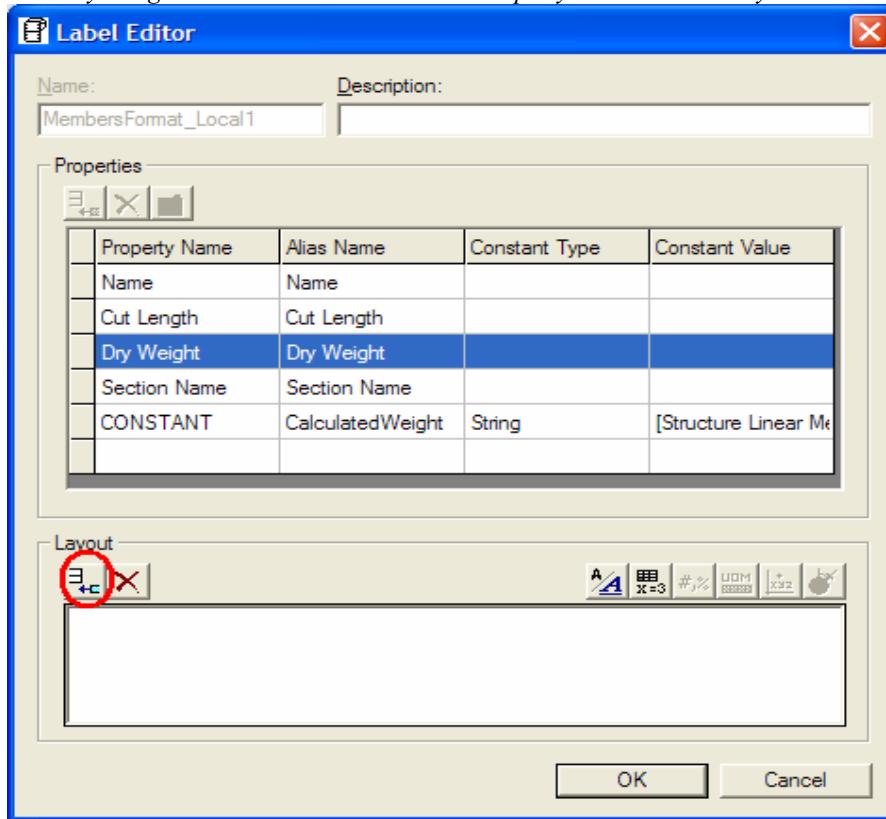
3. Select the *Labels* tab and check the "Use labels" check-box.



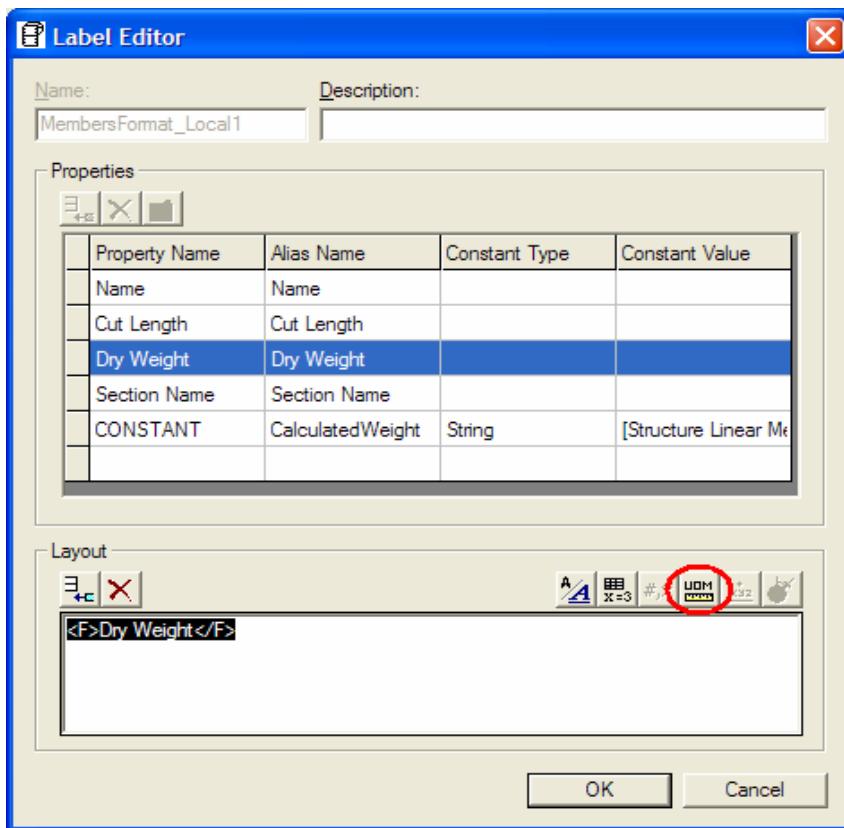
4. Select “New Format ...” in the *Name* drop-down.



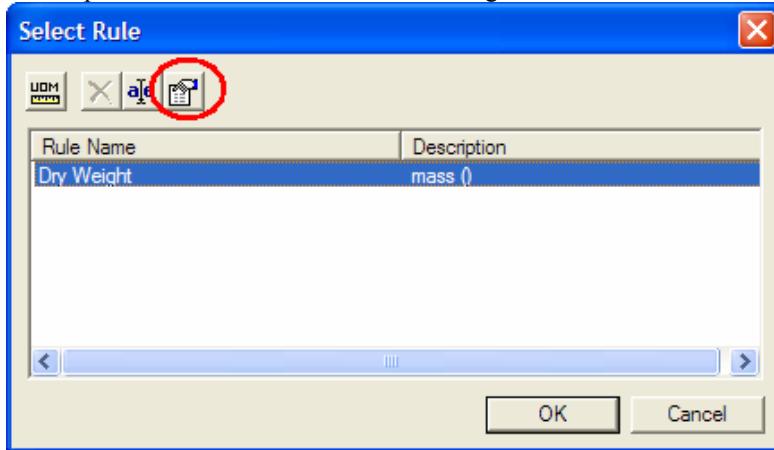
5. Highlight the *Dry Weight* row and click on the “Add Property” button in the *Layout* section:



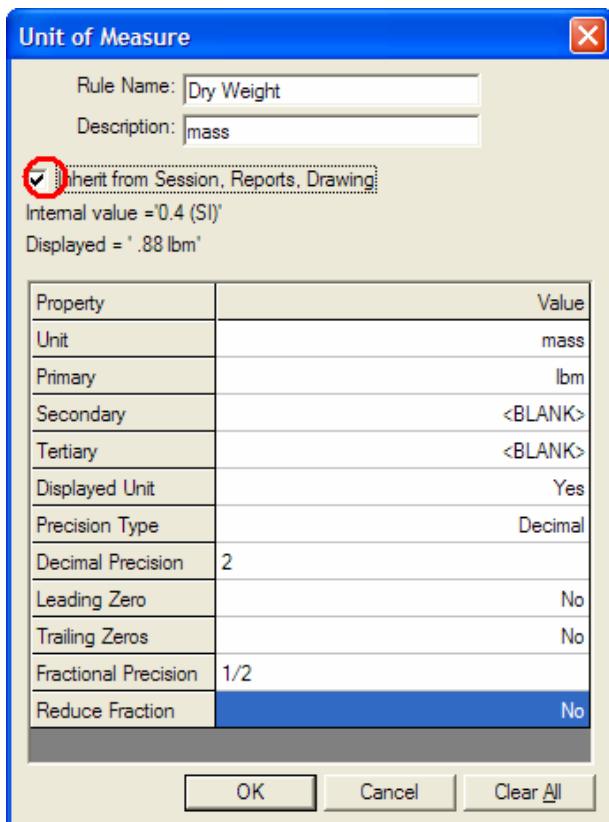
6. Highlight the text <F>Dry Weight</F> in the Layout pane and click the “UOM” button :



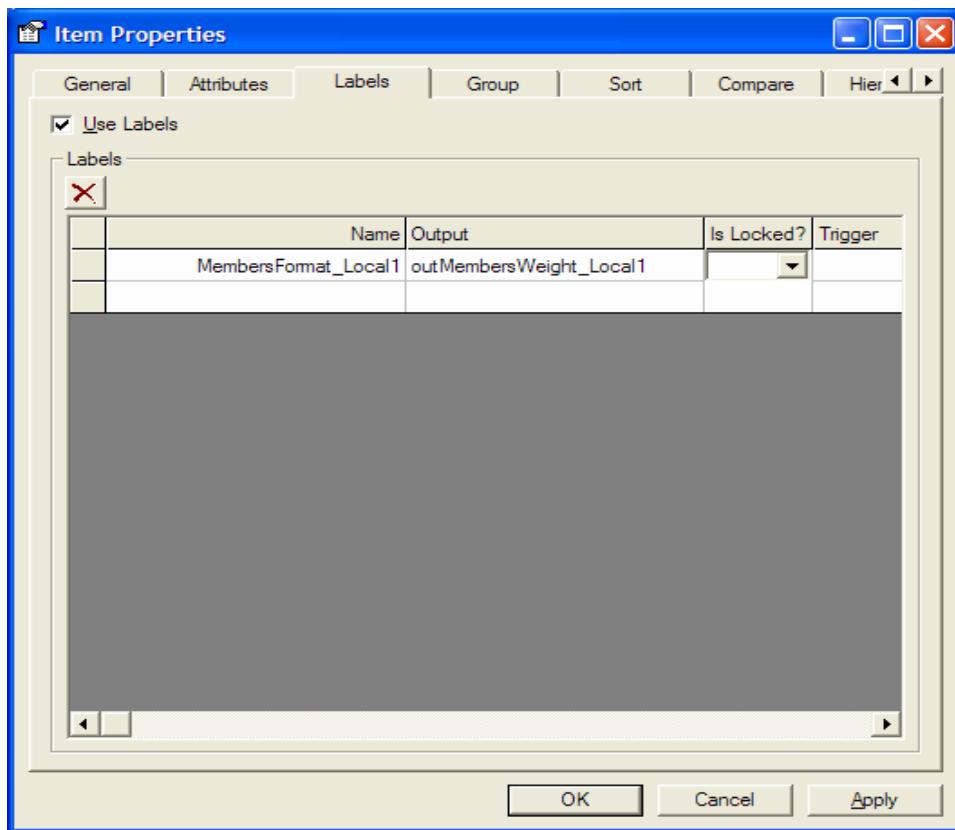
7. Click on the Properties button in the Select Rule dialog:



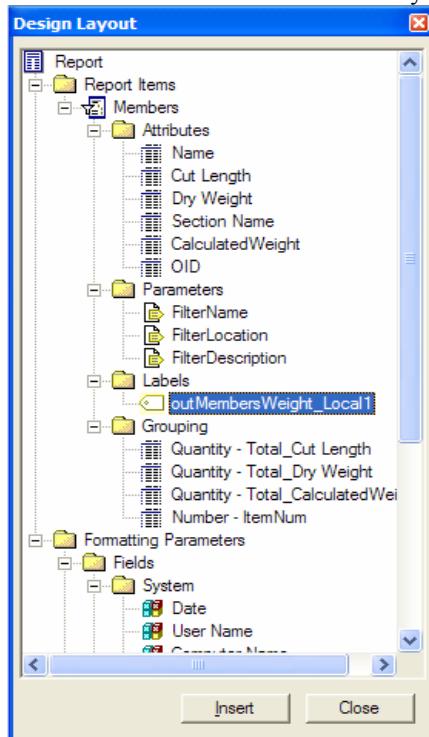
8. Select your units, precision, etc.. After you are done, check the “Inherit from Session, Reports, Drawings” check-box, to allow the user to interactively change those setting at run-time if needed.



9. Click “OK” in the *Unit of Measure* dialog, click “OK” in the *Select Rule* dialog, and click “OK” in the *Label Editor* dialog.
10. Rename the “Output” name to something more meaningful, like *outMembersWeight_Locall*, and click “OK”



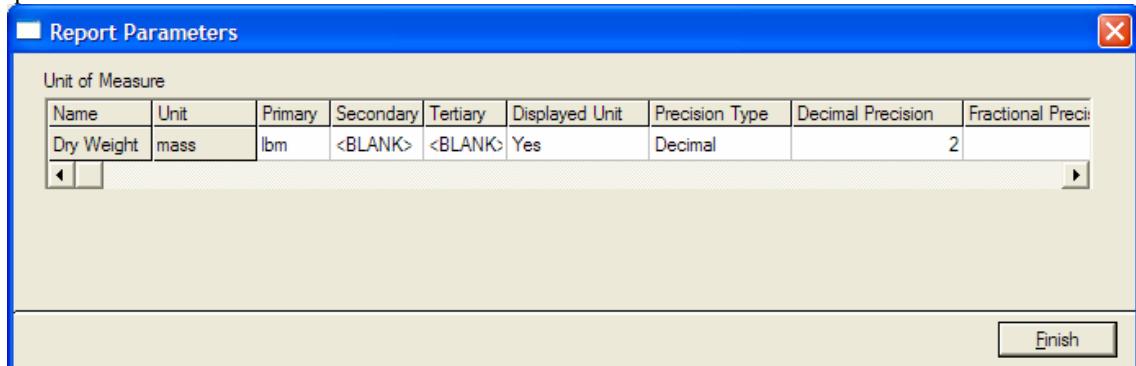
11. You will see a new Labels node with the label you just created.



12. Drag and drop the label in the *Weight* column replacing the unformatted *Dry Weight* property in it.

Weight	Weight	Total
rs::Cur#Members::outMembersWeight_Locat#Men		
		Total

13. Save the xls, and Save report Template. You will see the following dialog allowing you to change the formatting of the weight property at any time, since we checked the “Inherit from Session, Reports, Drawings” in Step 8. Click “Finish” and test the report.



14. You will see the following result.

Cross Section	Length	Weight	Total Length	Total Weight	ItemNum	Calculated Weight
W18X40	6.92785	899.39 lbm	253.1618	14863.10694	29	33223.26
W14X53	10.3632	1769.41 lbm	165.2829237	12038.84744	22	28740.18
W18X35	6.92785	785.41 lbm	318.6799308	5261.94877	28	36593.88
W24X68	6.0587255	1342.29 lbm	48.472598	4871.116133	34	10814.08
W14X82	10.3632	2741.02 lbm	35.3314	3217.969263	25	9505.17

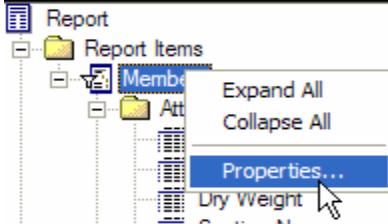
If you want to change your units, precision, etc., you can do this at any time by Right-Clicking on the report and selecting “Parameters...” and then Updating the report.

Using the “New Format...” option is the most convenient and straightforward way to format untyped values. It creates a label local to the Report, and you do not need to worry to supply it separately. Unfortunately, this approach has its limitations – it will not work for Aggregate Quantitative Properties (from Grouping), and it will not work if the values are coming from an SQL or VB report template. For those, as well as for all other cases, however, we can use an Existing Catalog Labels.

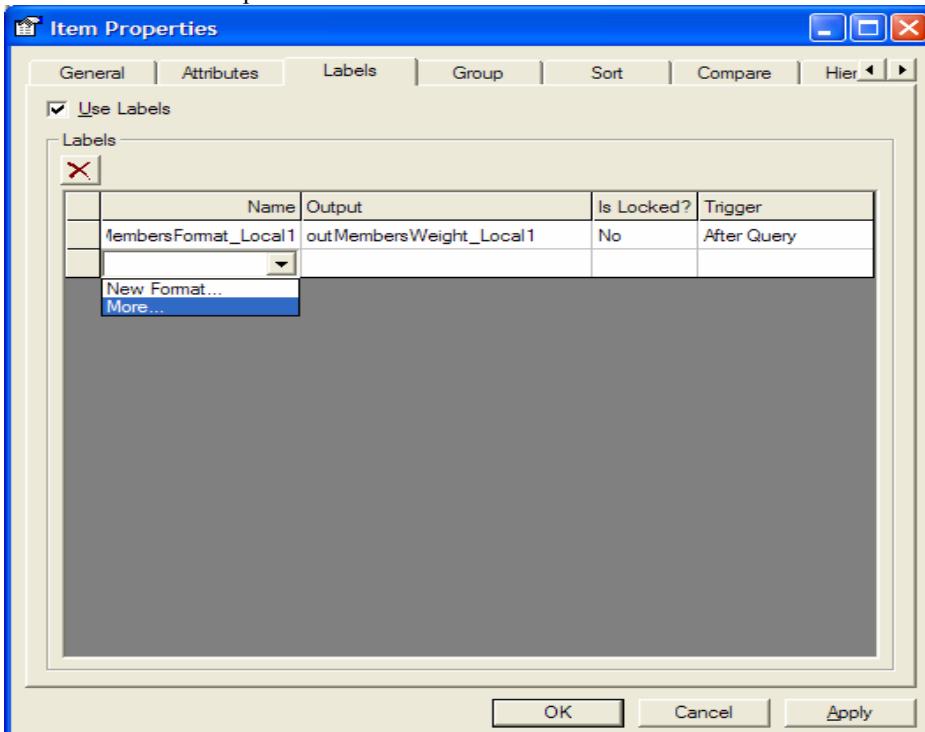
Using Existing Labels

Another way to achieve UOM formatting is to use an already existing Catalog label. If you remember from our previous lab, we could only select the original properties for formatting. If we want to format the Total Weight per Cross Section, we have to use an existing catalog label.

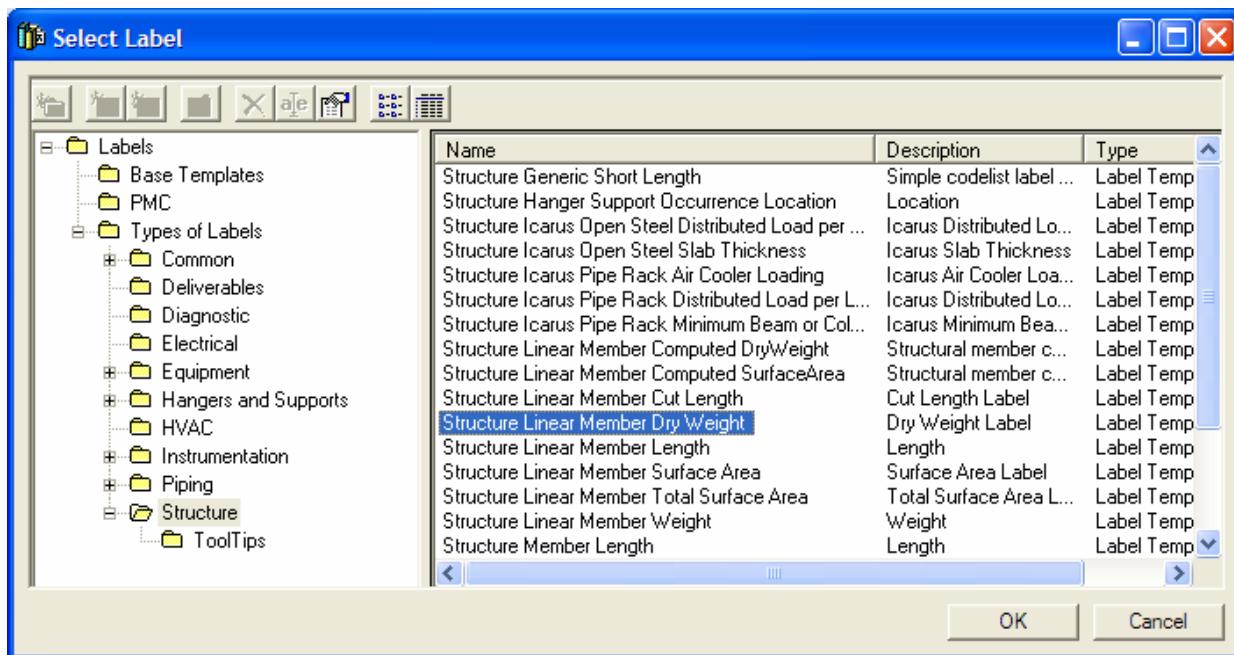
1. Open the “Structure Members” report template for editing.
- 2.
3. Go in the Design Layout, Right-Click on the “Members” node and select *Properties*



4. Select the “More...” option in the next available row:



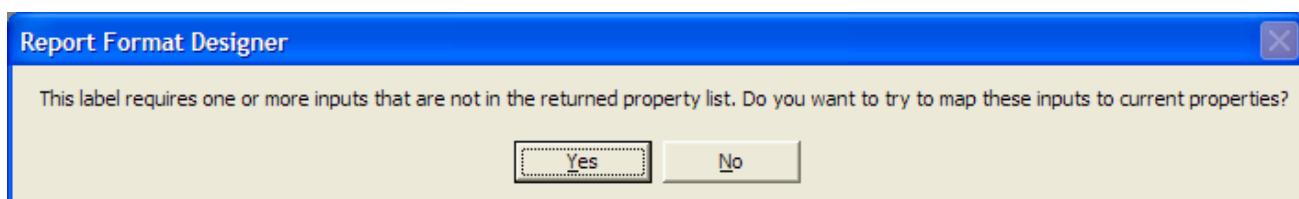
5. In the “Select Label” dialog navigate to *Labels\Types of Labels\Structure\Structure Linear Member Dry Weight* as shown:



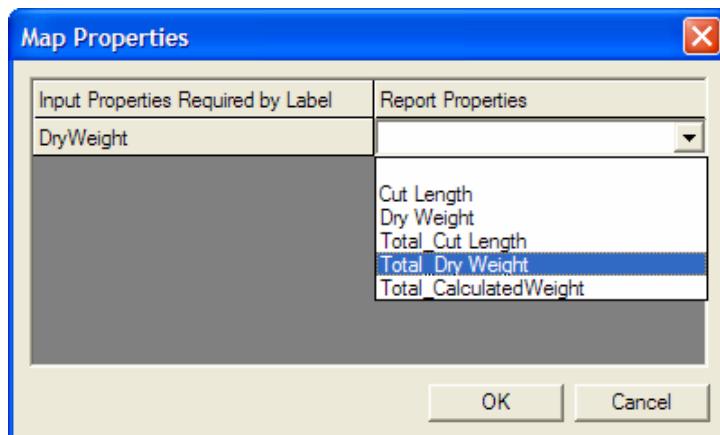
Important: Although we are using a delivered label here, we could have just as well used one that we created. Unlike Embedded Labels, when we use an existing label in this fashion, the software takes ONLY the formatting part of the label and disregards the property extraction part.

- Since the software will take only the formatting part of the existing label, it will not know what property it applies to. It will try to figure this out by comparing the Aliases used for the properties in both the existing catalog label and the report we are editing. If it finds a matching property alias, it will automatically apply the formatting to that property. In this case you will not see the next 2 prompts, and the label will not become a part of the report, i.e. the Report will be dependent on this label existing in the Catalog at all times.

In our case, though, the Alias used by the label is “DryWeight” (<RETURNED_PROPERTY Name="DryWeight" SQLType="Double"> from the Structure Linear Member Dry Weight.rqe file), which is different than any aliases in our report (our “Dry Weight” had a space). The software will ask us to manually do the mapping. Click “Yes”.

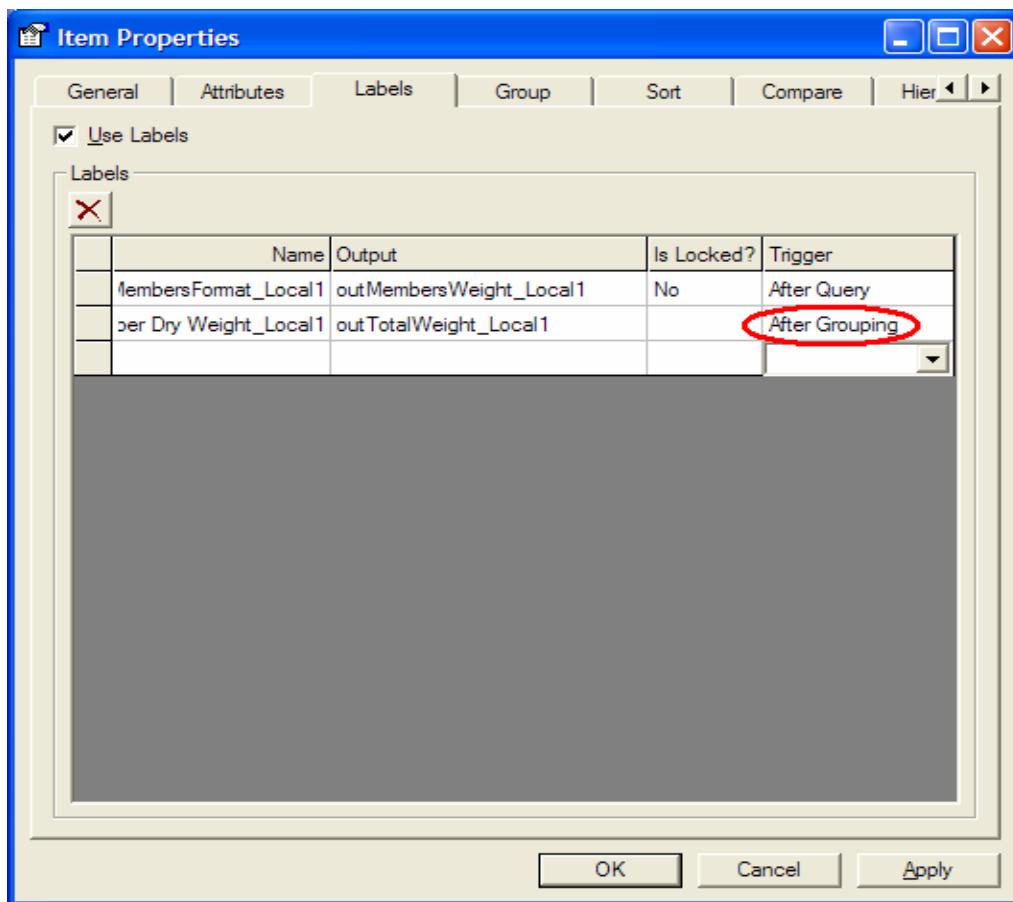


- In the next dialog select “Total_Dry Weight” as the property to map to

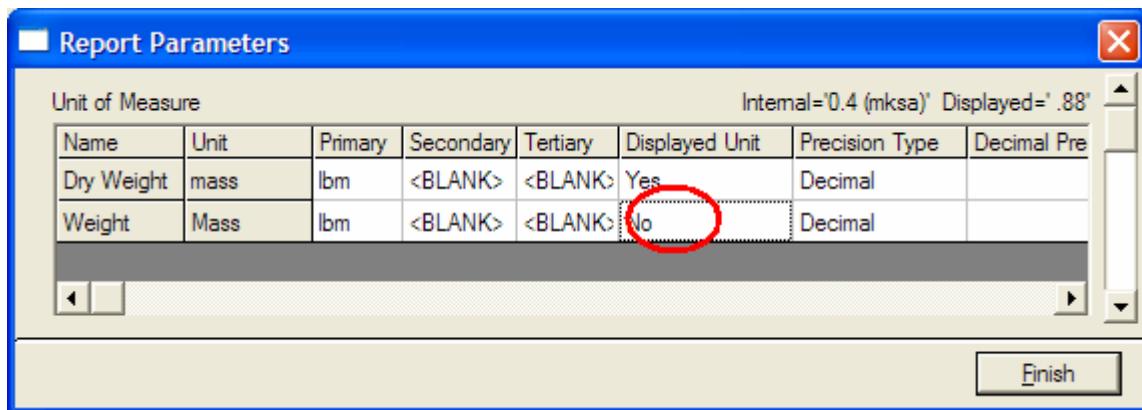


Important: When you manually map a property, the software creates a local copy in your report of the existing Catalog label. This means that your report becomes completely independent of that existing Catalog label - even if you delete your Catalog label, your report will still work.

8. Rename the Output name to something meaningful (*outTotalWeight_Local1*), and change the *Trigger* option to After Grouping because out Total Weight property will not exist before then.



9. Drag-and-drop the label in the Total Weight column replacing the previous property, save and test. You may want to set *Display Unit* to "No" for the Total Weight, since the SimpleAutoSum macro that gives the total weight at the very bottom will not work with anything other than numeric data.



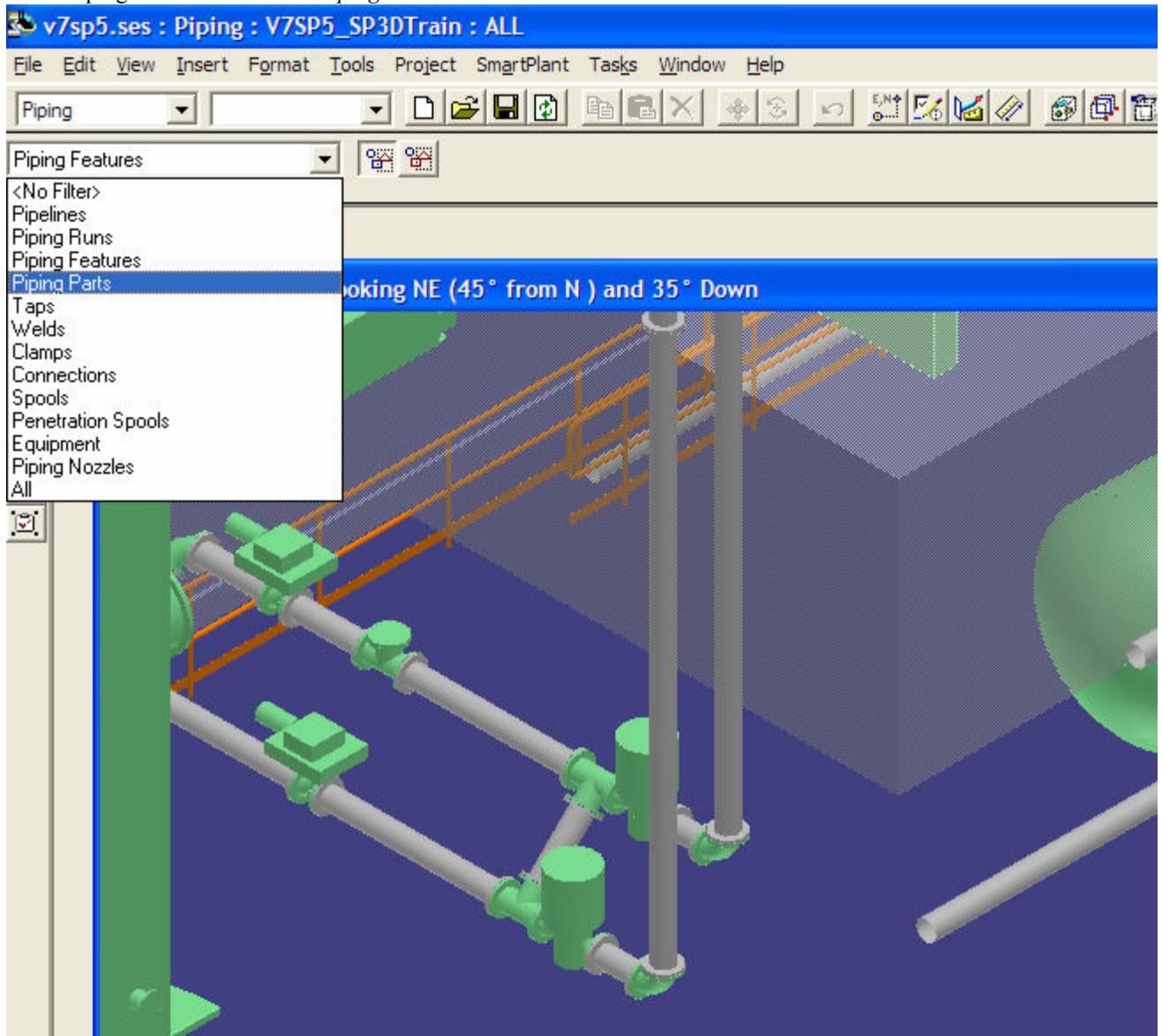
Understanding the data model

Remarks or notes from objects

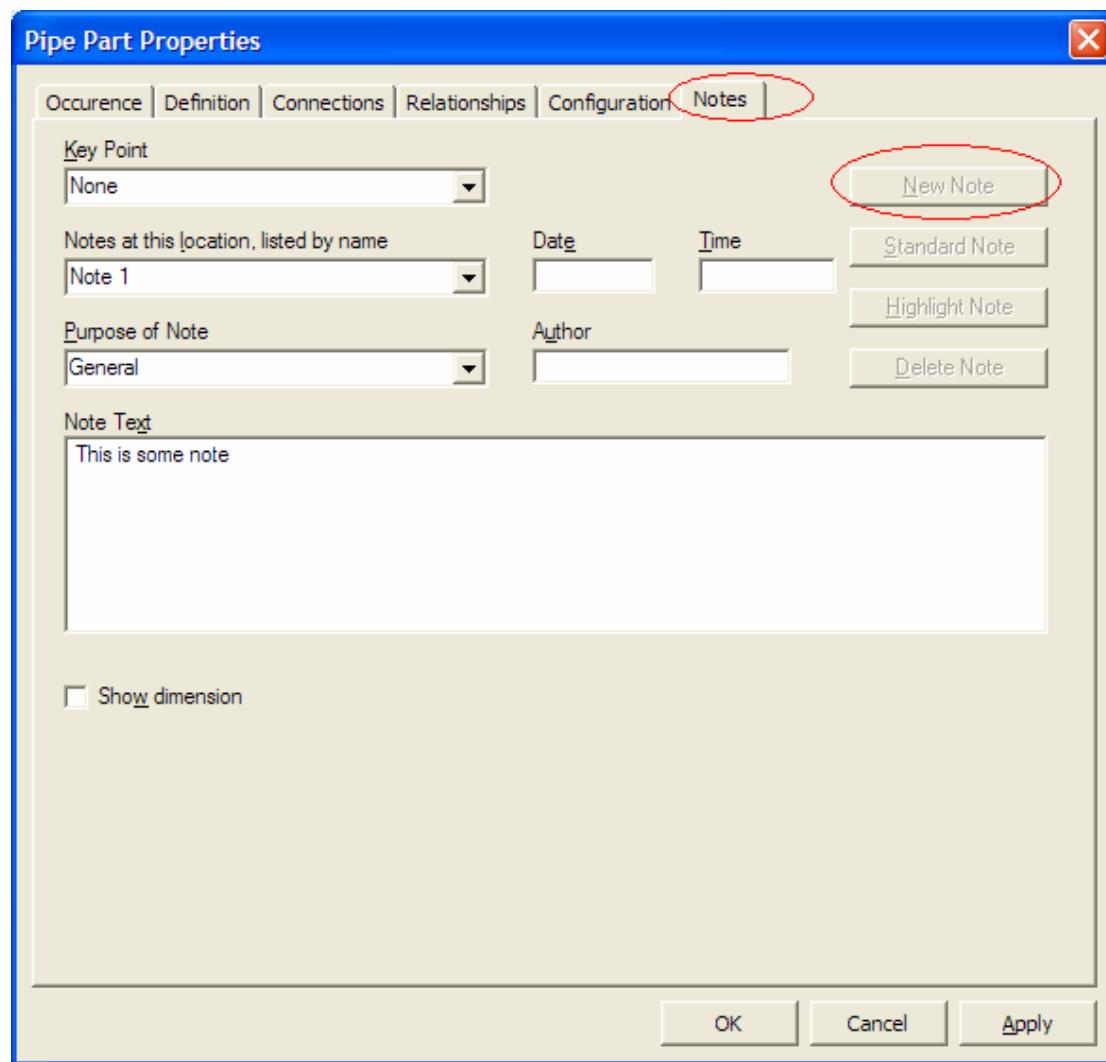
COM Query

We are interested in reporting on notes on piping parts. First let's add notes to a few piping parts:

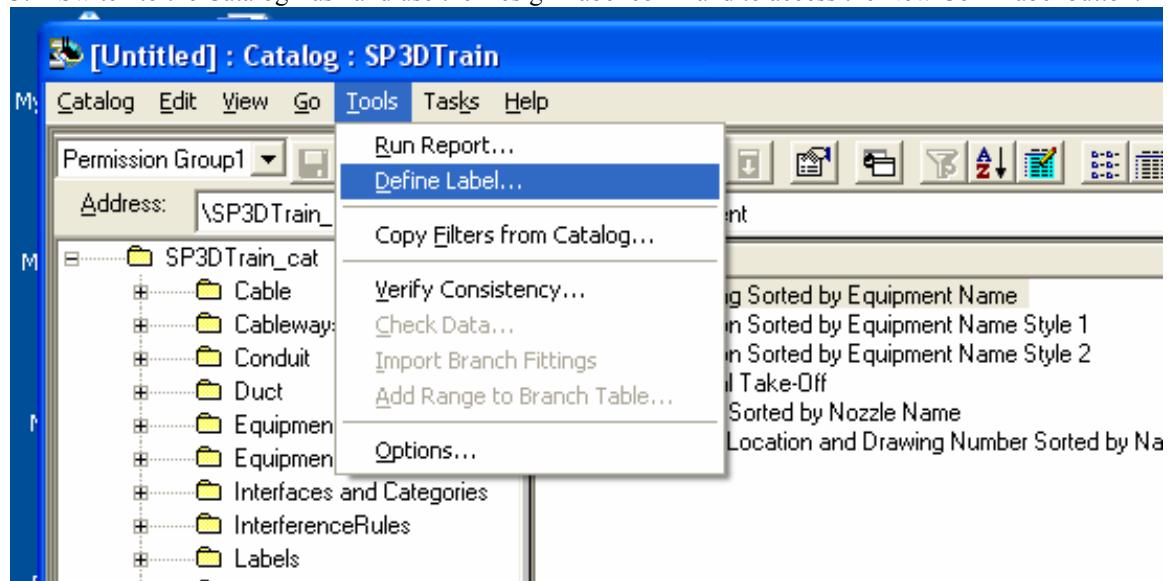
1. In the Piping environment select *Piping Parts* in the selection filter:

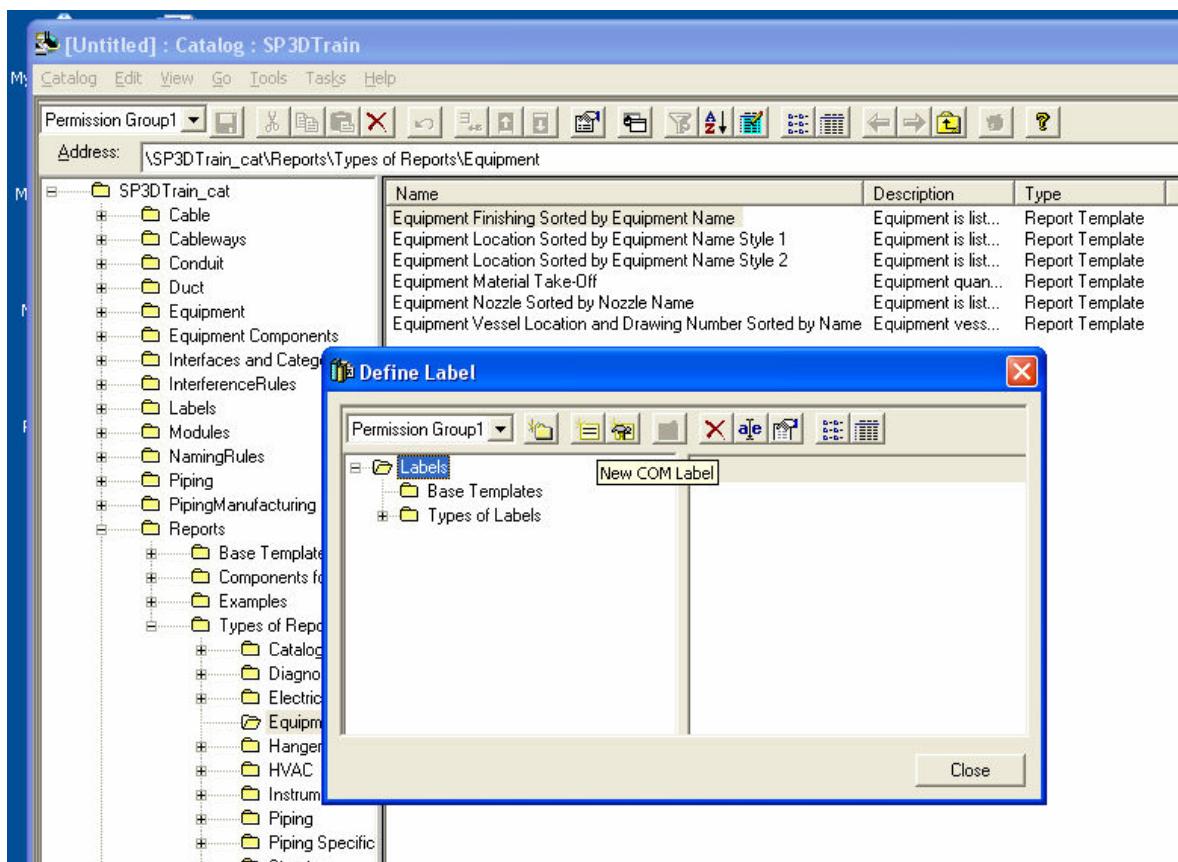


2. Right-click on any Piping Part and select *Properties*. Go to the *Notes* tab and click *New Note*. Select *Purpose* and type in something for *Note Text*: Click "OK" when done.

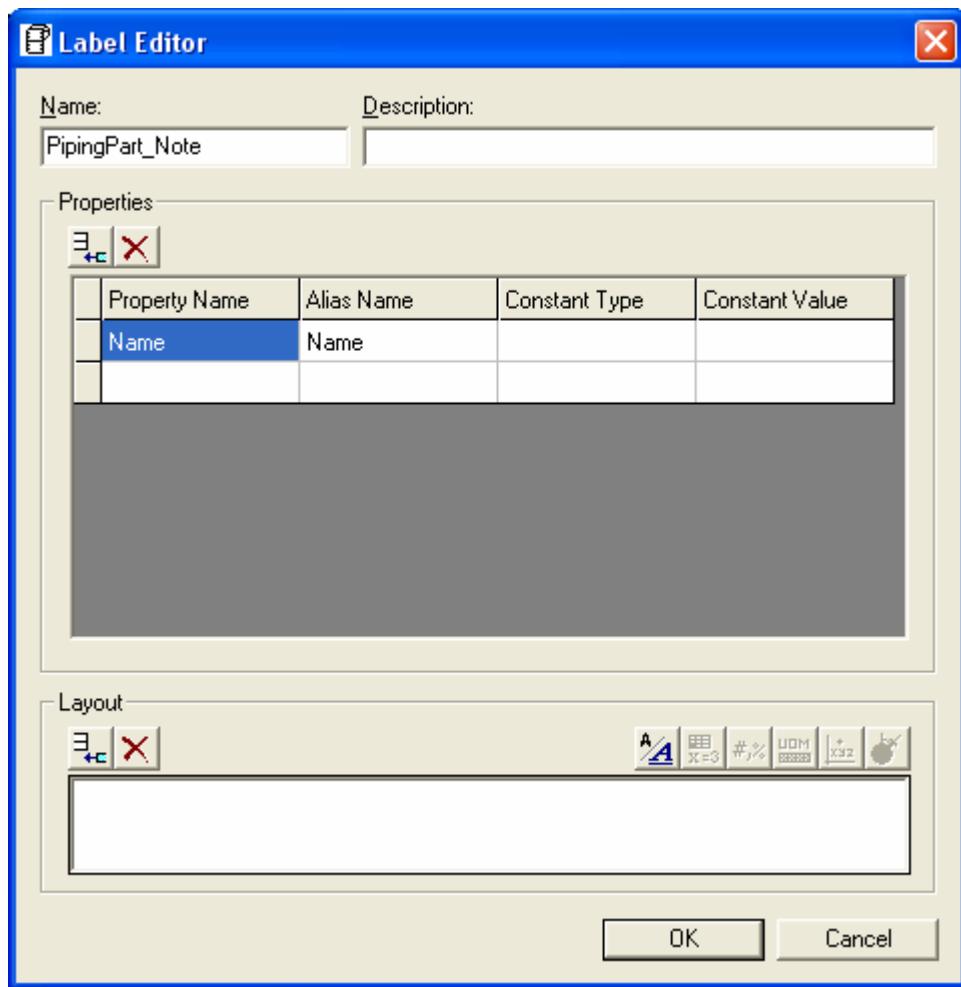


3. Switch to the Catalog Task and use the Design Label command to access the New Com Label button:

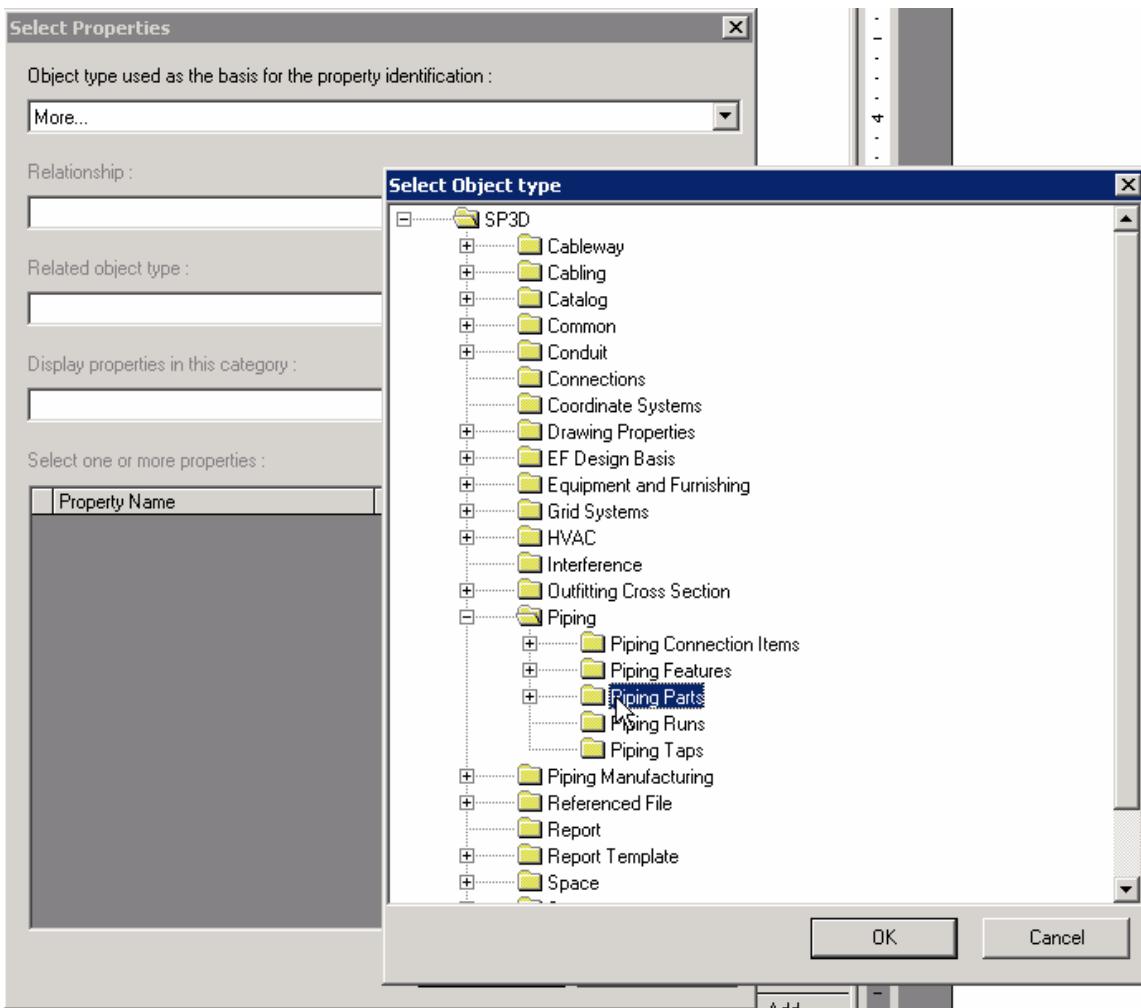




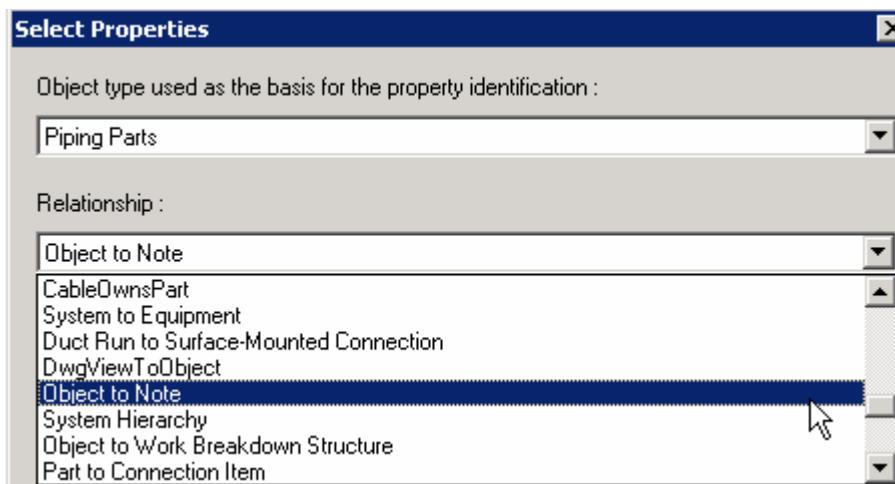
- Using the label editor, start editing a new blank label. Change the name of the label to PipingPart_Note



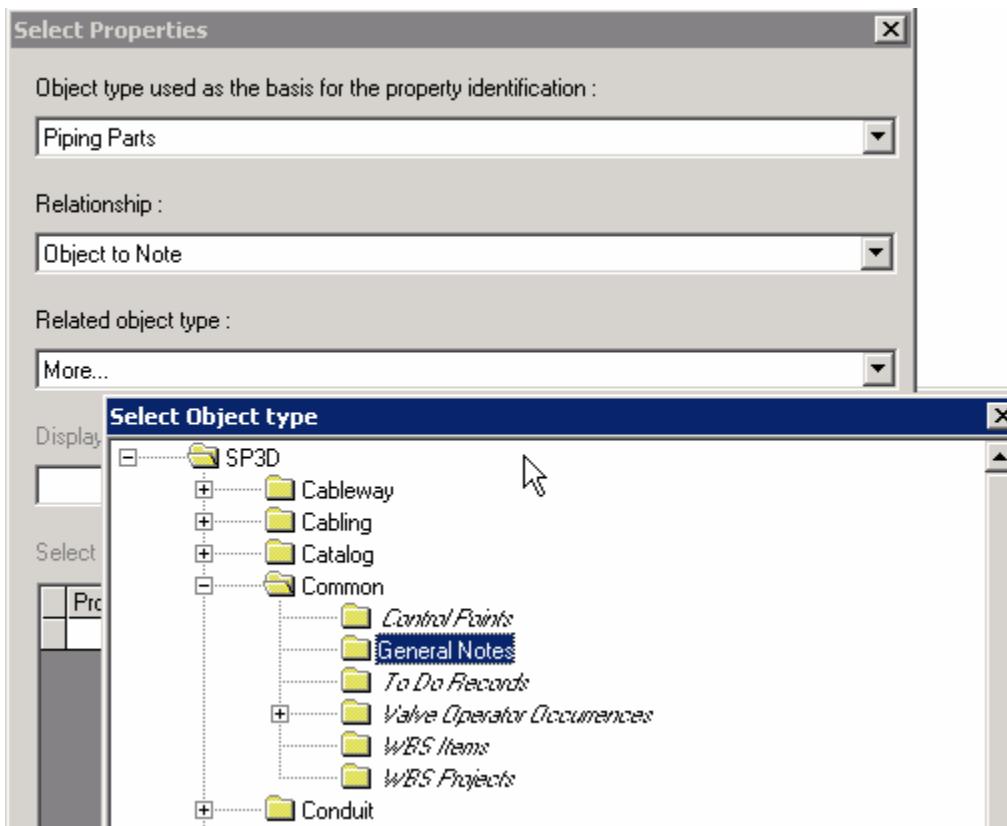
5. Start adding a property using the Add button. Now we must start our navigation from Piping Parts.



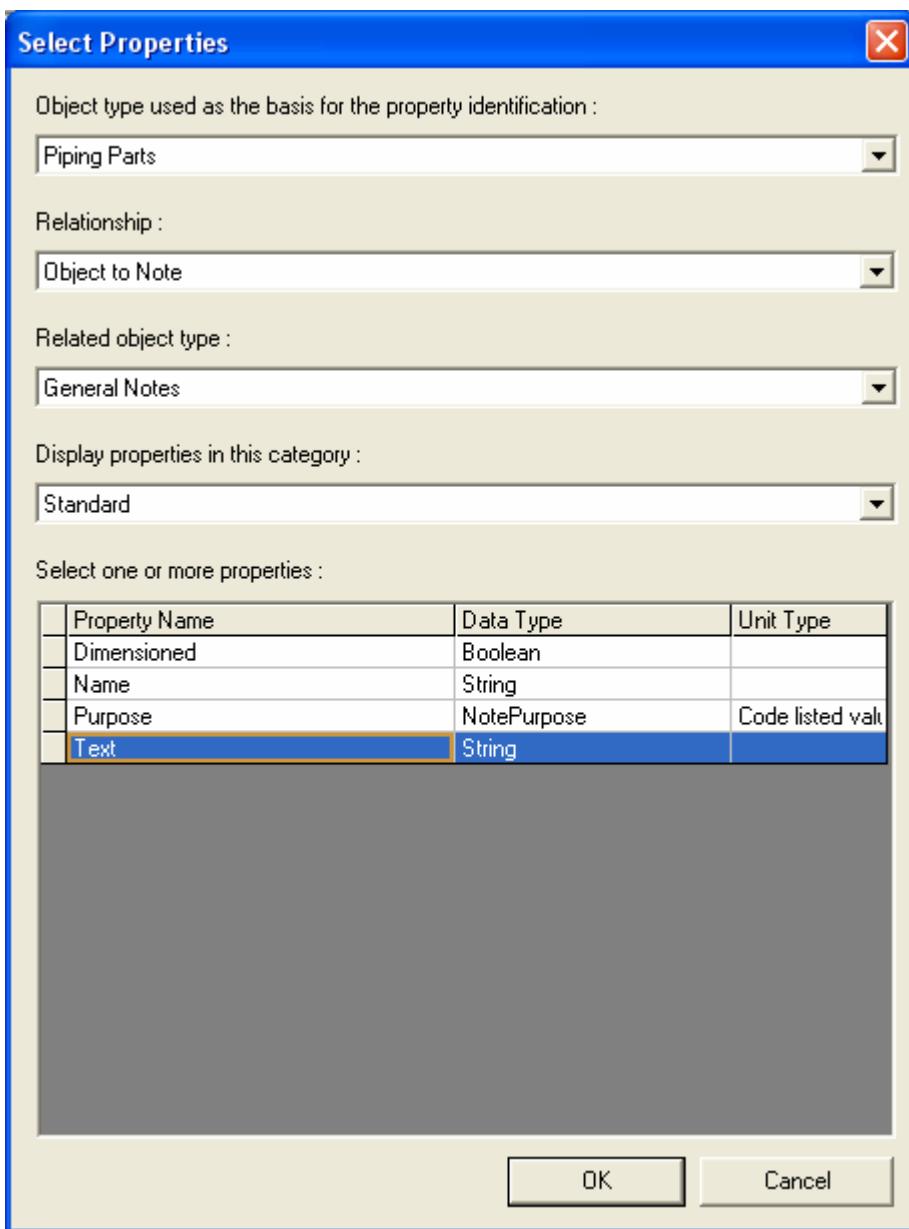
6. We are interested in notes, which are accessed via the relationship, Object to Note



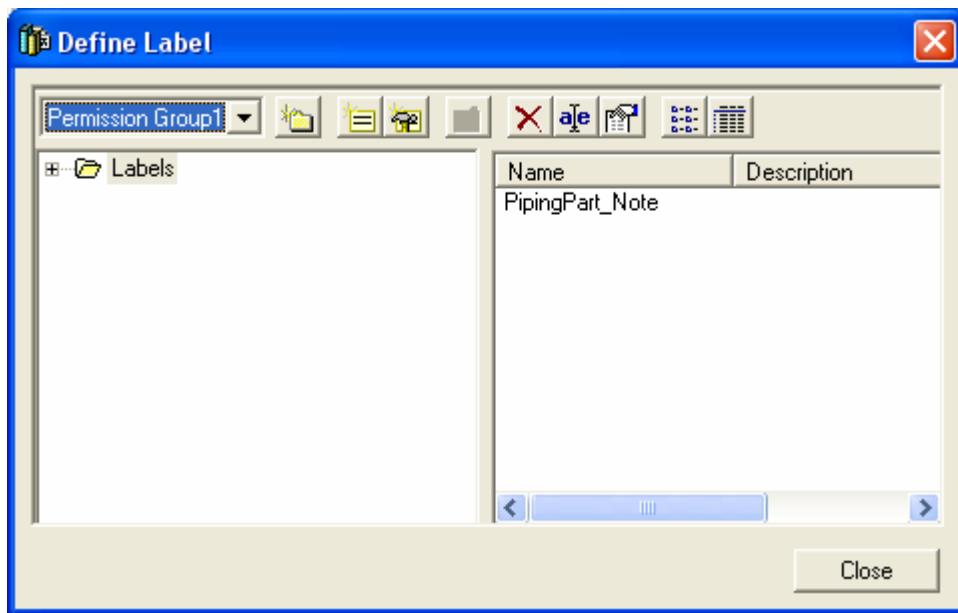
7. The property we are interested in, the note text lies on the General Notes



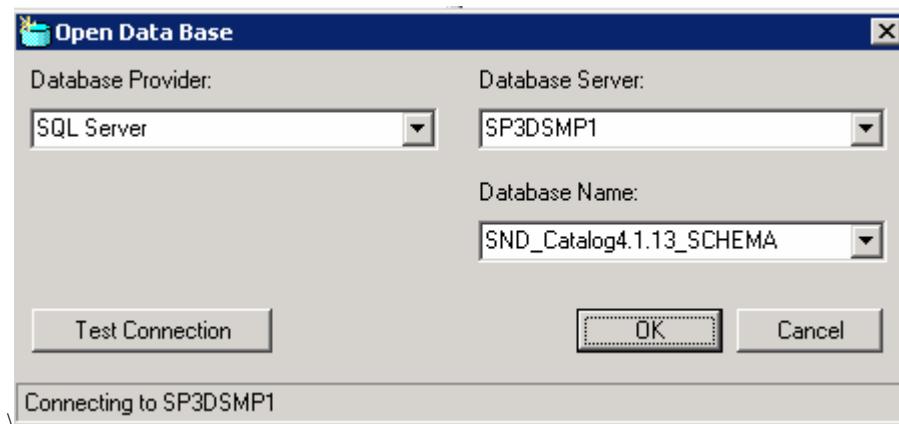
8. The property we are interested in is the Note Text which is on the standard category



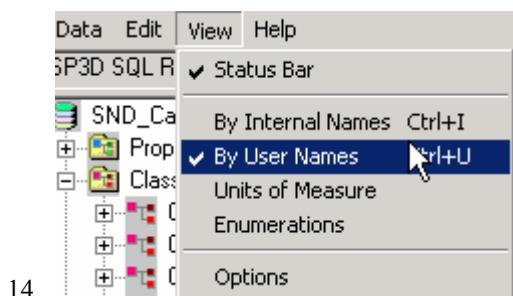
9. This label, when applied to a piping part will return the note associated with it.
10. Click OK, OK to save the label:



11. Let us now trace this through the metadata browser.
12. Open the metadata browser and point to a catalog schema (your instructor will help you identify what goes in this form).



13. Set the view menu to view by user names.



14. The BOC hierarchy that we see in the Object Type tab can be navigated via Classification TopNodes.
15. Expand Piping to find Piping Parts and click on Piping Parts.

The screenshot shows the SmartPlant 3D Schema Browser interface. The left pane displays a tree view of the schema structure under 'SP3D SQL Repository'. The right pane shows the properties of a classification node named 'PipingParts'.

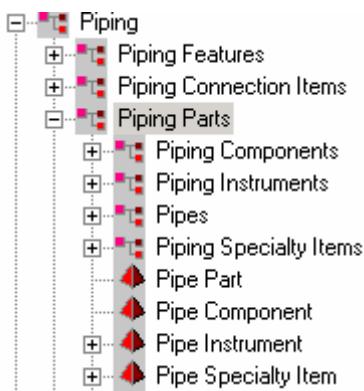
Name	Data
Name	PipingParts
UserName	Piping Parts
OID	{8CDF283C-28DD-493F-858C-18D013024C}
GUID	{8CDF283C-28DD-493F-858C-18D013024C}
UserFlags	[1] - Defined in Rose
Has Children Classification Node	PipeComponents
Has Children Classification Node	PipeInstruments
Has Children Classification Node	Pipes
Has Children Classification Node	PipeSpecialtyItems
Represents Class	CPMPipeOccur
Represents Class	CPRtePipeComponentOcc
Represents Class	CPRtePipeInstrumentOcc
Represents Class	CPRtePipeSpecialtyOcc
Represents Class	CURTE2PRVA10
Represents Class	CURTE3DP10
Represents Class	CURTE3WayBall0
Represents Class	CURTE45DegElbow0
Represents Class	CURTE45DegLRElbow0
Represents Class	CURTE4BOX30
Represents Class	CURTE4BOX40
Represents Class	CURTE4BOX0
Represents Class	CURTE4CYL30
Represents Class	CURTE4CYL40
Represents Class	CURTE4CYL0
Represents Class	CURTE90DegElbow0
Belongs to Package	CommonRoute Business Services

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

The screenshot shows the 'Properties of ClassificationNode: Piping Parts' table. The last row, 'Belongs to Package', is highlighted.

Name	Data
Has Common Interface	IJRtePiping
Has Common Interface	IJWBSItemChild
Has Common Interface	IJWBSProjectChild
Has Common Interface	IJWeightCG
Belongs to Package	CommonRoute Business Services

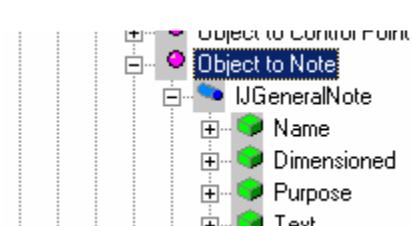
18. Also note in the tree view that specific kinds of piping parts are Pipe Component, Pipe Part, Pipe Instrument etc.



19. Thus we must begin our hunt under the CommonRoute folder.
20. Expand Pipe Component and it shows a list of interfaces that are implemented by Pipe Components. 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’.)
21. You will see a pink bubble that shows the Object to Note relationship.



- 22.
23. Expand the bubble further and you will find the property you are looking for on an interface at the other end of the relationship.

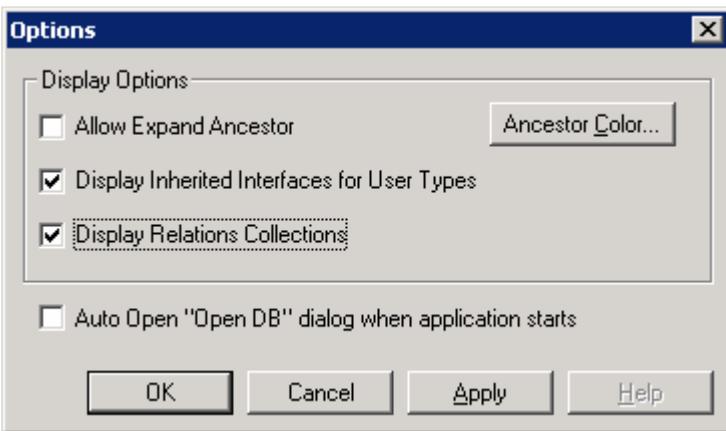


- 24.

25. Now return to SmartPlant 3D and Save As the label, then open the rqe file, then focus on the RETURNED_PROPERTY section which looks like below

```
26.      <RETURNED_PROPERTY  
27.          Name="Text"  
28.          SQLType="BStr">  
29.          <PATHS>  
30.          <PATH  
31.              SourceType="IJDOBJECT"  
32.              DestinationInterface="IJGeneralNote"  
33.              DestinationProperty="Text">  
34.              <STROKES>  
35.              <STROKE  
36.                  Filter="First"  
37.                  Interface="IJDOBJECT"  
38.                  ExitCondition=""  
39.                  ExitValue=""  
40.                  RelationCollection="GeneralNote"  
41.                  Recursive="No"  
42.                  IsVirtualRelationship="No" />  
43.              </STROKES>  
44.              </PATH>  
45.          </PATHS>  
46.      </RETURNED_PROPERTY>
```

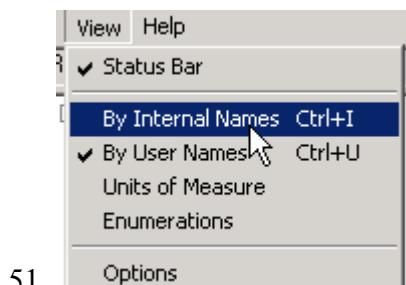
47. To follow this using the MetaData Browser, set the option to display Relations Collections and then close and reopen the MetaData Browser.



48.

49. Expand thru Pipe Component and IJDOBJECT.

50. At this time, also set the option to view by Internal Names.



51.

52. You will see new pink double bubbles in the tree view. These are the relation collections.

The screenshot shows the SmartPlant 3D Schema Browser interface. The left pane displays a tree view of schema objects under 'SP3D SQL Repository'. The right pane shows the 'Properties of Attribute: Text' for a selected object. The properties listed are:

Name	Data
Name	Text
UserName	Text
DBColumnName	Text
OID	{24ED3A83-B64A-4635-AB6B-EDD97BBBB...}
Type	[1] - Char
CodeListTable	
ReadOnly	False
UnitsType	[0] - Undefined
IsValueRequired	False
ComplexTypeID	
UserFlags	[1] - Defined in Rose
Is Supported by Interface	IJGeneralNote

The status bar at the bottom indicates the full path: SP3DTrain_cat_SCHEMA\CommonRoute Business Services\CPRePipeComponentOcc\IJObject\GeneralNote\ContainsNote\Object\IJGeneralNote\Text

53.

54. The returned property is named ‘Text’ and it can be obtained by starting from a source ‘IJDOObject’, traveling ‘through’ GeneralNote relationcollection to find the Text property on the interface at the other end of the relationship.

SQL Query

Let us now map the traversal to a SQL query which achieves the same result. Clicking on any of the entries in the tree view will show the DBViewName corresponding to that entry in the detail view. Click on IJDOObject to see that the DBViewName corresponding to it is JDOObject.

	Name	Data
	Name	IJDObject
	UserName	IJDObject
	DBViewName	JDOObject

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

Select * from JDOObject

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

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

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

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

```
Select * from JDOObject
Join XcontainsNote on JDOObject.oid = XcontainsNote.Oidorigin
Join JgeneralNote on JgeneralNote.oid = XcontainsNote.OidDestination
```

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

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

```
Select * from JDObject jo
Join XcontainsNote rcn on jo.oid = rcn.Oidorigin
Join JgeneralNote jgn on jgn.oid = rcn.OidDestination
```

However this query gives us too much information, what we are interested in is the Note text,

So,

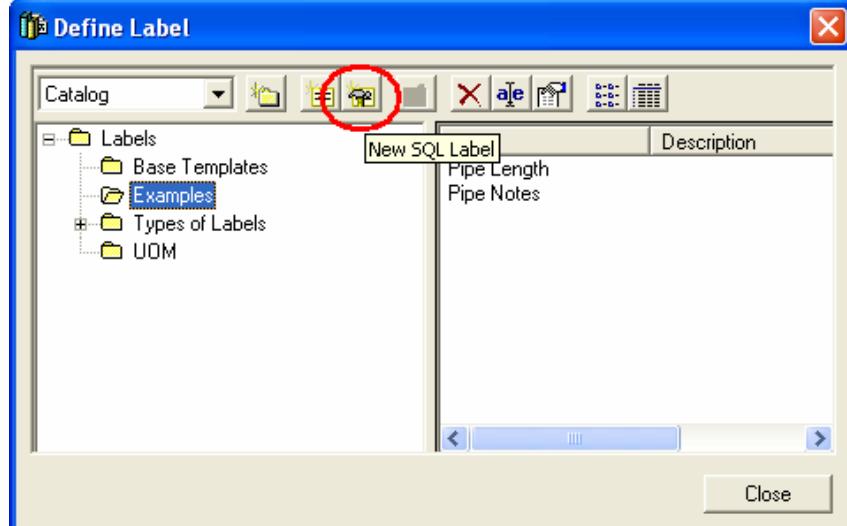
```
Select jgn.Text from JDObject jo
Join XcontainsNote rcn on jo.oid = rcn.Oidorigin
Join JgeneralNote jgn on jgn.oid = rcn.OidDestination
```

(If you are using the View Editor in the SQL2005 Editor draw joins between each of the tables and then return Text.)

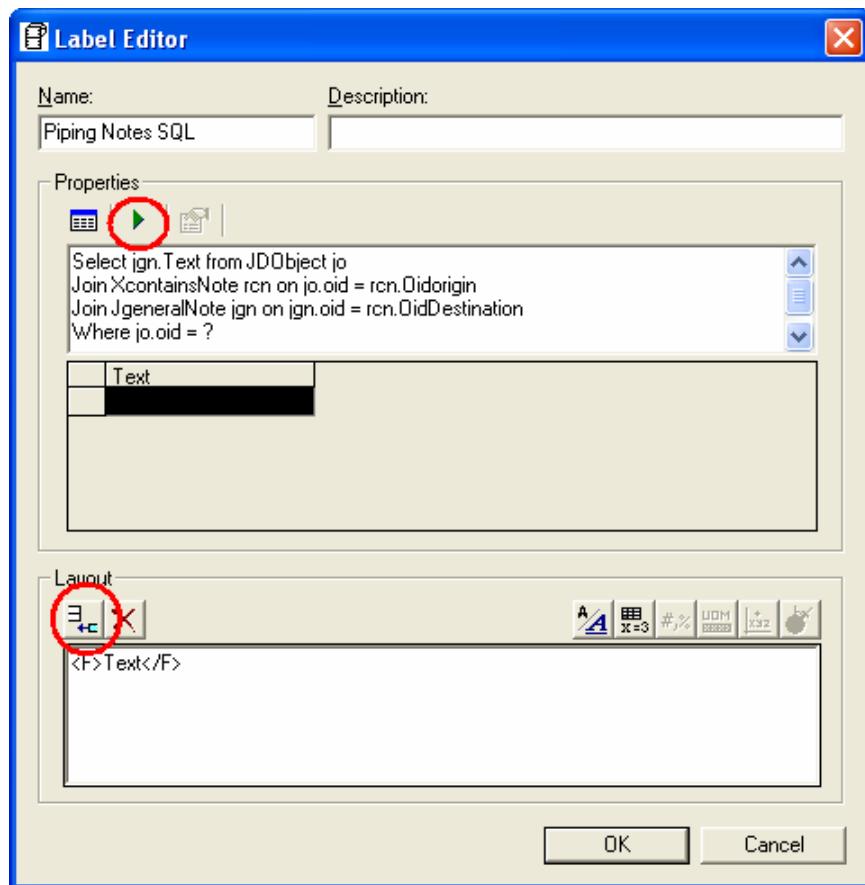
Finally, this query will give us results for all objects, for a label, we are interested in the note on the particular object being labeled, which we do by adding a where clause to the query

```
Select jgn.Text from JDObject jo
Join XcontainsNote rcn on jo.oid = rcn.Oidorigin
Join JgeneralNote jgn on jgn.oid = rcn.OidDestination
Where jo.oid = ?
```

In the Catalog environment, create a new label, this time choosing SQL label:



Paste the above text into the pane and hit the “Run” button . Highlight the “text” column and add it to the layout with the button:



The software passes the oid of the object (piping part) to this query at run time in place of the ? and returns the note for the particular object (piping part).

Alternative method for those who cannot write SQL but who use MSSQL 2005:

Select * from JDObject

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

The screenshot shows the Microsoft SQL Server Management Studio interface. In the top left, the title bar reads "Microsoft SQL Server Management Studio". Below it is a toolbar with various icons. The left pane contains the "Object Explorer" and "Registered Servers" panes. The "Object Explorer" pane shows a tree view of databases under "NAQUADAH\SQL2005 (SQL Server 9.0.1399 - NAQUADAH\Ferguson)". One of the databases, "SP3DTrain_RDB", is selected. The right pane displays the results of a query named "NAQUADAH\SQL...QLQuery1.sql". The query is "select * from JDOBJECT". The results grid has columns: Did, ClassId, UIDCreator, and UIDL. There are 22 rows of data, each containing a unique identifier and timestamp.

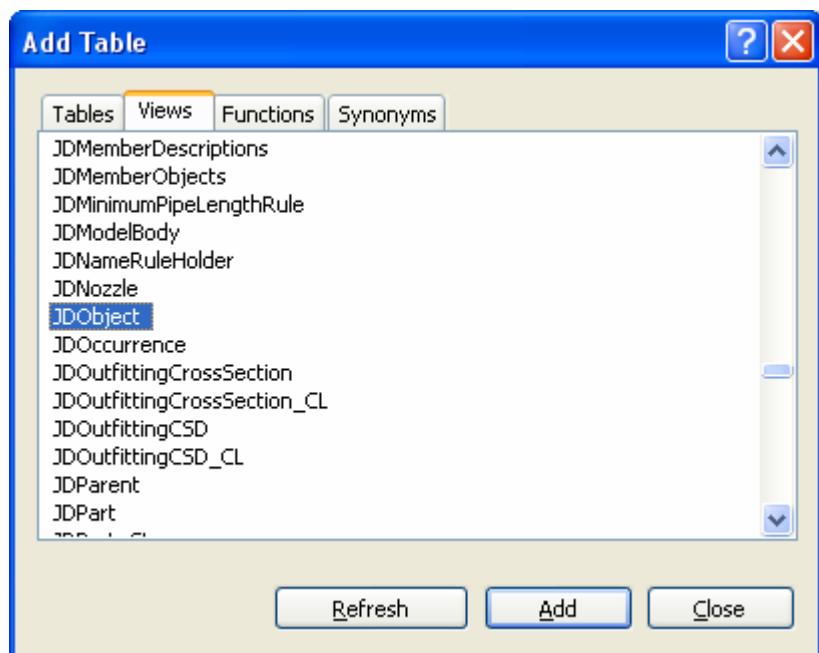
Did	ClassId	UIDCreator	UIDL
1	0000272E-0000-0000-0000-059EAF451D04	10030	E1DAC812-8E75-424A-89A3-EBA166CA694A
2	0000272B-0000-0000-0100-059EAF451D04	10024	E1DAC812-8E75-424A-89A3-EBA166CA694A
3	00002727-0000-0000-0200-059EAF451D04	10023	E1DAC812-8E75-424A-89A3-EBA166CA694A
4	00004E2E-0000-0000-0000-0D6FAF451D04	20014	E1DAC812-8E75-424A-89A3-EBA166CA694A
5	00013883-0000-0000-0004-0D6FAF451D04	80003	E1DAC812-8E75-424A-89A3-EBA166CA694A
6	00013885-0000-0000-0019-0D6FAF451D04	80005	E1DAC812-8E75-424A-89A3-EBA166CA694A
7	00013887-0000-0000-001A-0D6FAF451D04	80007	E1DAC812-8E75-424A-89A3-EBA166CA694A
8	000138A9-0000-0000-001D-0D6FAF451D04	80041	E1DAC812-8E75-424A-89A3-EBA166CA694A
9	000138A5-0000-0000-001E-0D6FAF451D04	80037	E1DAC812-8E75-424A-89A3-EBA166CA694A
10	00013885-0000-0000-011A-0D6FAF451D04	80005	E1DAC812-8E75-424A-89A3-EBA166CA694A
11	00013887-0000-0000-011C-0D6FAF451D04	80007	E1DAC812-8E75-424A-89A3-EBA166CA694A
12	000138AA-0000-0000-011D-0D6FAF451D04	80042	E1DAC812-8E75-424A-89A3-EBA166CA694A
13	00046CD1-0000-0000-0200-0D6FAF451D04	290001	E1DAC812-8E75-424A-89A3-EBA166CA694A
14	00002711-0000-0000-0204-0D6FAF451D04	10001	E1DAC812-8E75-424A-89A3-EBA166CA694A
15	000138A8-0000-0000-021C-0D6FAF451D04	80040	E1DAC812-8E75-424A-89A3-EBA166CA694A
16	00002711-0000-0000-021D-0D6FAF451D04	10001	E1DAC812-8E75-424A-89A3-EBA166CA694A
17	00004E27-0000-0000-0300-0D6FAF451D04	20007	E1DAC812-8E75-424A-89A3-EBA166CA694A
18	000138B5-0000-0000-0305-0D6FAF451D04	80053	E1DAC812-8E75-424A-89A3-EBA166CA694A
19	000138A8-0000-0000-030C-0D6FAF451D04	80040	E1DAC812-8E75-424A-89A3-EBA166CA694A
20	000138BD-0000-0000-0318-0D6FAF451D04	80013	E1DAC812-8E75-424A-89A3-EBA166CA694A
21	00002711-0000-0000-0319-0D6FAF451D04	10001	E1DAC812-8E75-424A-89A3-EBA166CA694A
22	00002711-0000-0000-031A-0D6FAF451D04	10001	E1DAC812-8E75-424A-89A3-EBA166CA694A

We know that we are going to be performing several relationship steps to correlate information in different tables.

For that reason, let us use the “New View...” creator in MS SQL2005 to build the SQL statement for those of us who are not SQL savvy. Oracle users will need to find another solution to help them write the final SQL.

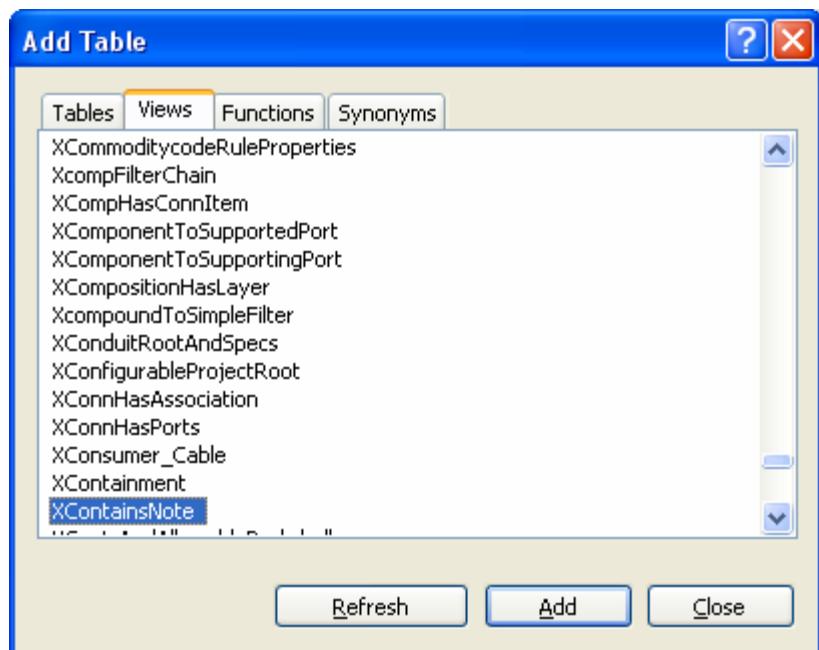
The screenshot shows the "Object Explorer" pane of the SQL Server Management Studio. It lists several databases under "NAQUADAH\SQL2005 (SQL Server 9.0.1399 - NAQ)". One database, "SP3DTrain_RDB", is expanded to show its contents. Under "Tables", there is a "Views" folder. A context menu is open over this "Views" folder, with the option "New View..." highlighted.

Select JDOBJECT from the List of Views:

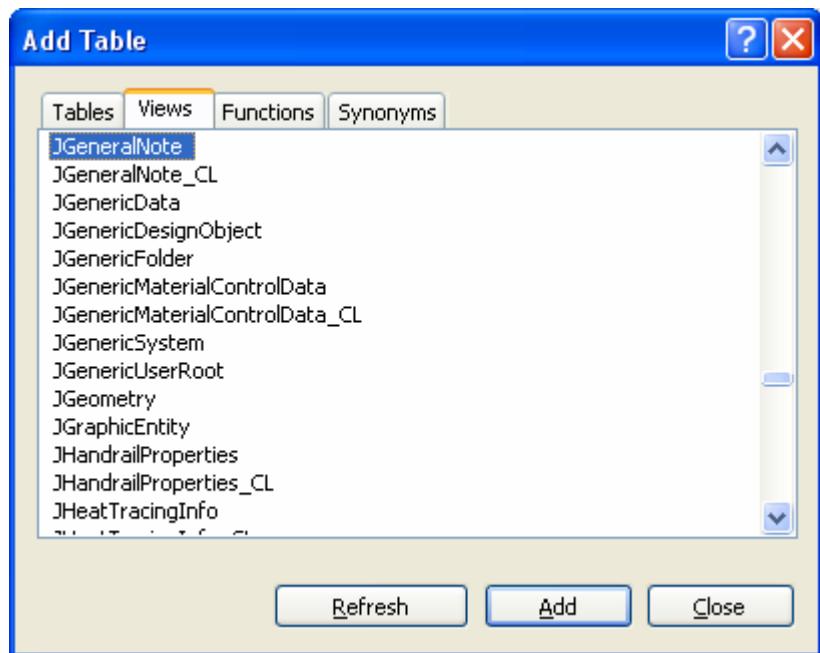


Click Add

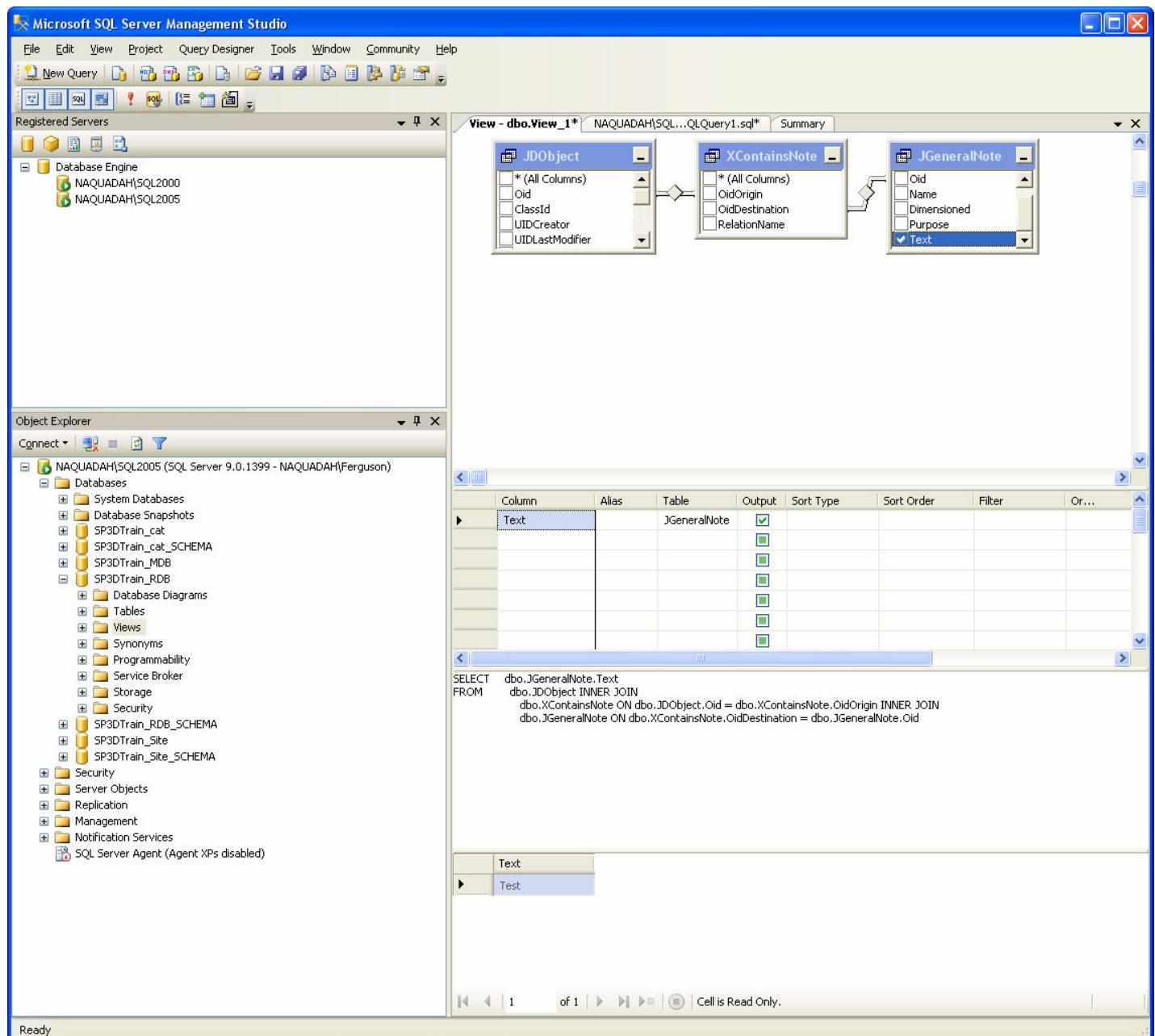
From the View List select XContainsNote, and then Click Add...



From the View List select JGeneralNote, and click Add...

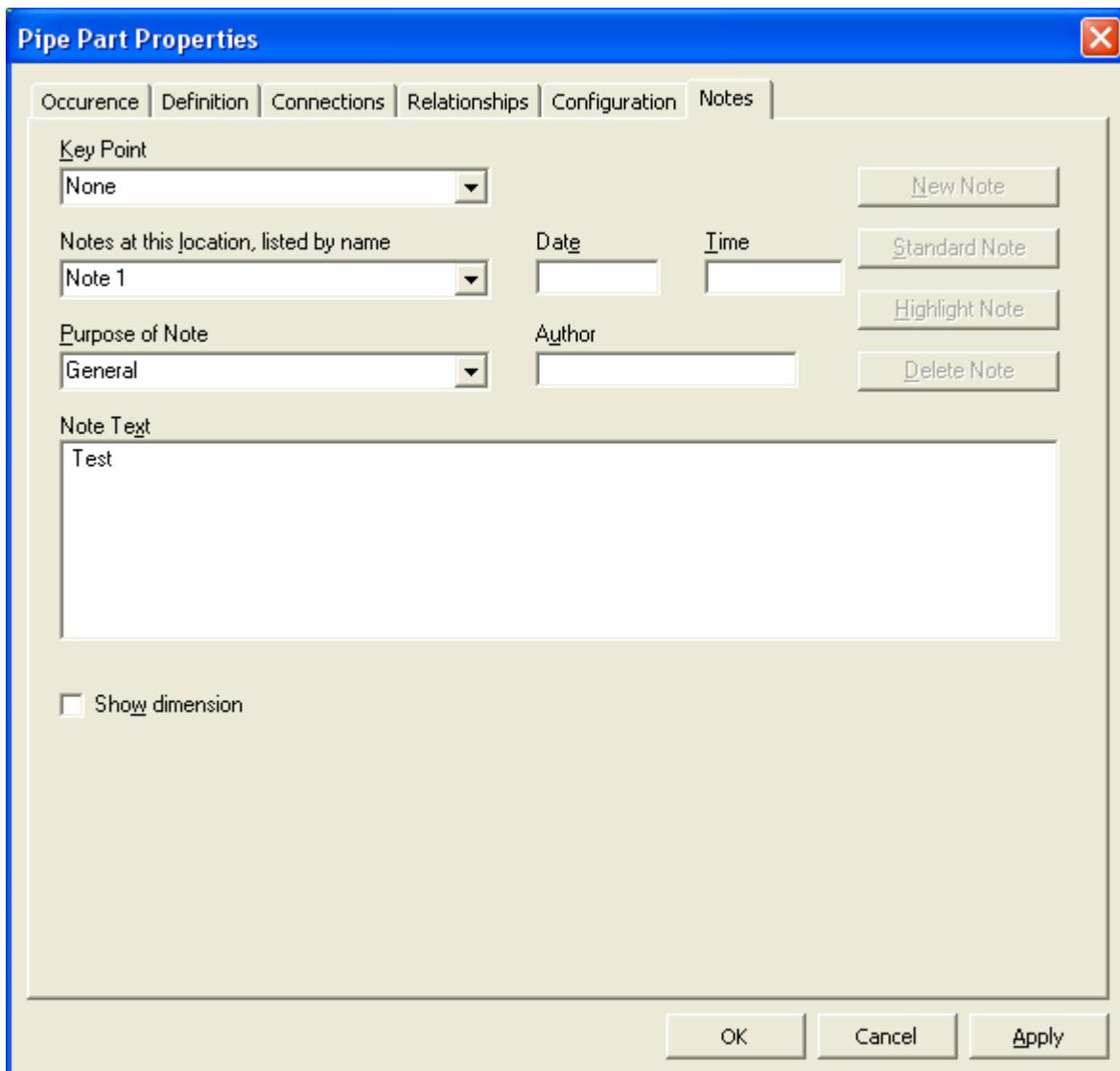


Drag and drop the relationship OID→OIDOrigin, OIDDestination → OID as shown below and click on Text



Then copy out SQL from the 3rd Frame and use that in your report. Remember, we are not making a view here, we are just using the functionality of the view creator to write our SQL for us.

If you do not have any results, pick a Pipe Part and add a note as follows:



Advanced Queries

1. DestinationInterface = “CONSTANT”

Objective: Create a label that will return a short description based on the type of input object as follows:

“COMP” for Piping Components
 “PIPE” for Piping Parts
 “INST” for Piping Instruments
 “SPCL” for Piping Specialty Items
 “SUPP” for Hanger Supports

Note: Those short description do not come from any stored properties and can be changed to whatever suits the task at hand best.

Solution:

What is unusual about this requirement is that we are not trying to extract a stored property. We still have to determine the object’s type in order to label it with its corresponding short description. Also, we have to consider that we will be labeling different object types.

1. Let’s start with an empty label template for a property. We have chosen the alias “LBL”:

```
<?xml version="1.0" encoding="UTF-8"?>
<REPORT_QUERY
Name="Piping IsoMTO RecordType"
Description=""
RequiresFilter="No">
<DESIGN_TIME
Progid="SP3DReportsQueryBuilder.COMQuery"
Action=""
Arg="" />

<RUN_TIME
Progid="SP3DRuntimeQuery.CQueryInterpreter"
Action=""
Arg="" />

<RETURNED_PROPERTIES>
<RETURNED_PROPERTY
Name="LBL"
SQLType="Bstr">
<PATHS>

<PATH/>

</PATHS>
</RETURNED_PROPERTY>
</RETURNED_PROPERTIES>
</REPORT_QUERY>
```

2. Next, let's try to write the PATH statement that will return "COMP" for Piping Components. The typical PATH statement looks like:

<PATH

```
SourceType=""  
DestinationInterface=" "  
DestinationProperty=" " />
```

Since the objects that we'll be labeling with this label can be potentially of different kinds, we have to choose only the ones of Piping Component type. We do this by finding an Interface that only Piping Components implement and then plugging its name in the [SourceType](#) attribute. To do that, we use the *Classification Top Nodes* part of the *Schema Browser* tool.

Let's find in the *Schema Browser Classification Top Nodes* the node corresponding to the *Piping Components* node in the SP3D Object Type Hierarchy. Since those are the same hierarchies, this should be straightforward (fig. 1 and fig. 2):

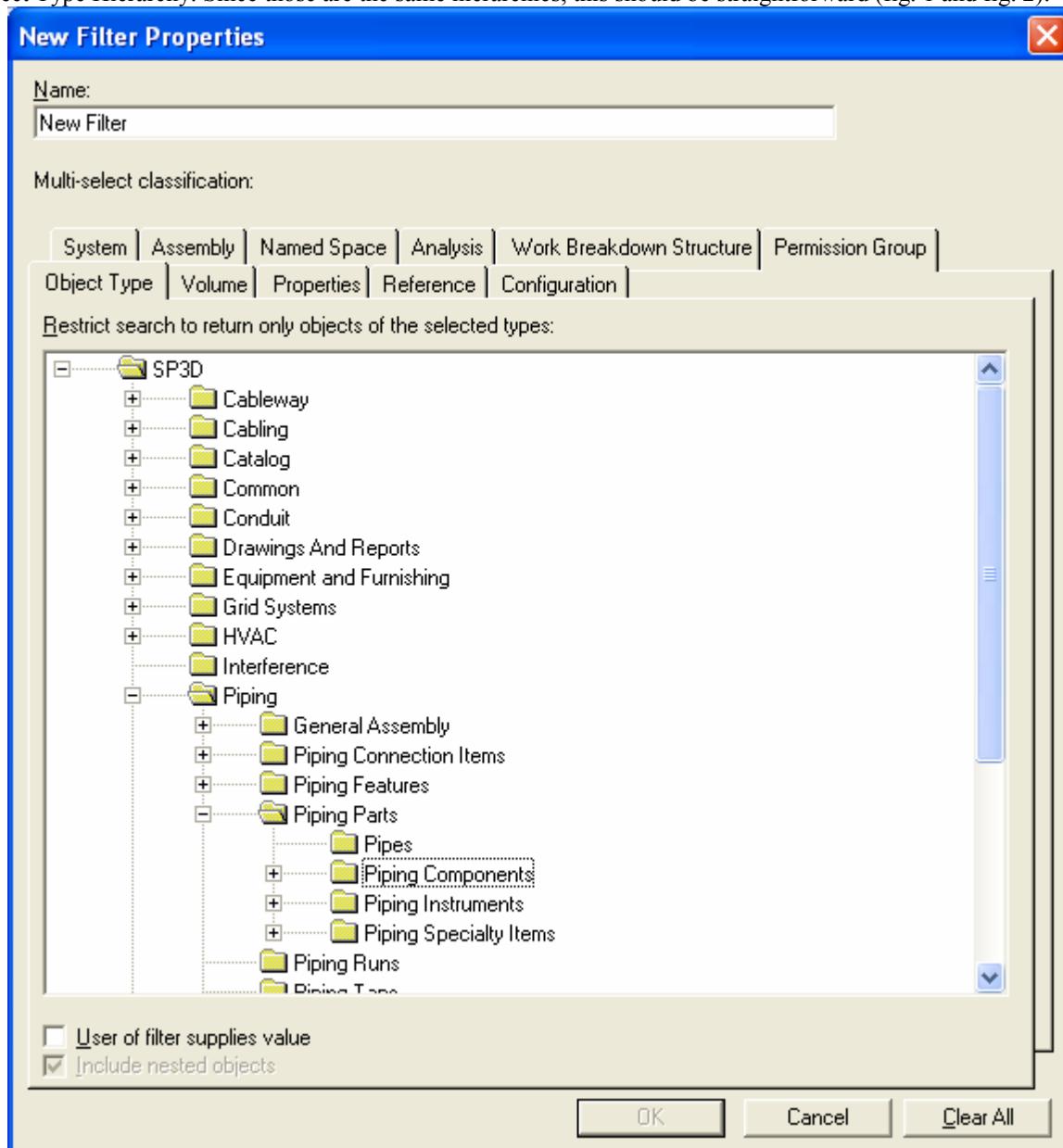


fig 1. Piping Components in the SP3D Object Type Hierarchy

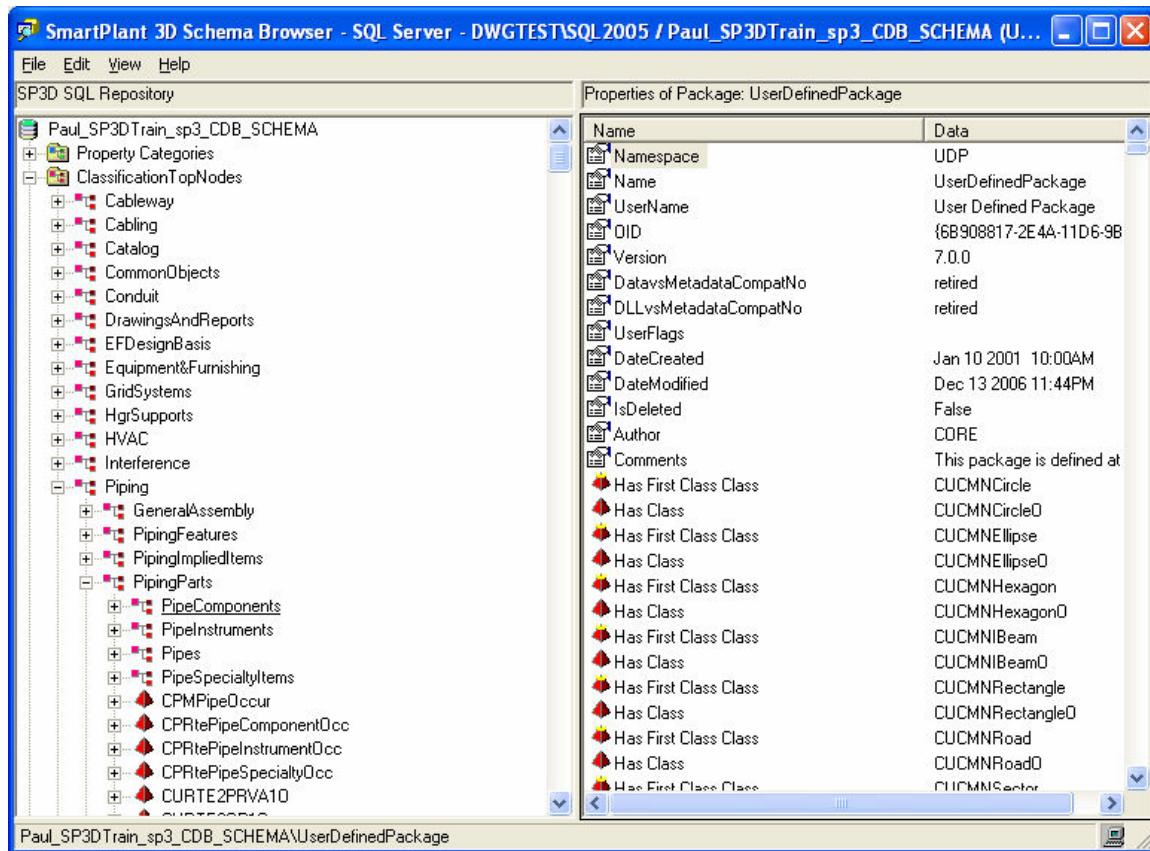


fig 2. Piping Components in the Classification TopNodes Hierarchy

Now let's expand the *PipeComponents* node in the *Schema Browser*. If we scroll down to the interfaces that are implemented by that node we'll see 2 of them: *IJRteCompOccur* and *IJRtePiping* (fig. 3):

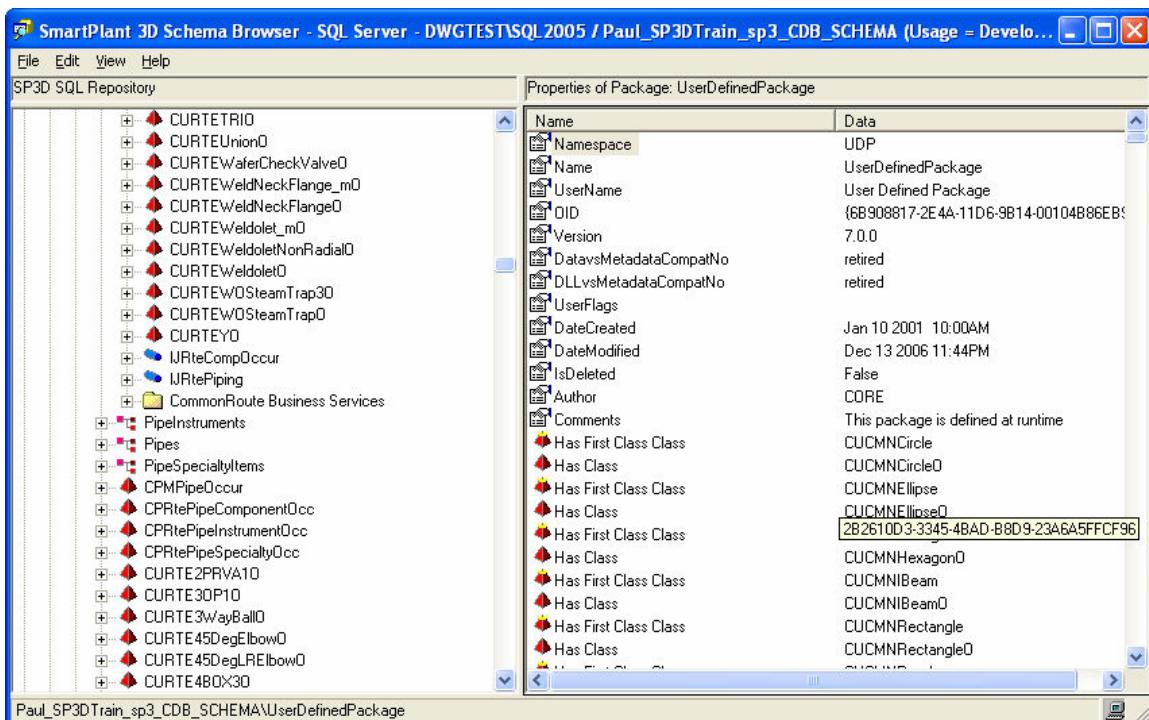


fig. 3 Interfaces defining Piping Components.

The interfaces listed under a given node in the *Classification TopNodes Hierarchy* are not all the interfaces implemented by that object type but rather the “type defining” interfaces – i.e. for an object to be classified as a Piping Component it HAS to implement those interfaces. We need only one interface to uniquely identify an object’s type, but since the interfaces listed under a given node are also used to determine the object type’s place in the overall *Business Object Classification* (BOC) hierarchy, we could have 2 or more as is our case. It us usually very easy and intuitive to pick the defining one, though: in our case *IJRtePiping* seems too general, and indeed, a quick check in few of the other nodes shows that all piping objects are required to implement it. So, this leaves us with *IJRteCompOccur* as the interface that uniquely identifies objects of type Piping Component. Our PATH part becomes

```
<PATH
  SourceType="IJRteCompOccur"
  DestinationInterface=""
  DestinationProperty=" " />
```

Now for the part where we need to apply the predefined text description “COMP”. We achieve this by writing “CONSTANT” for *DestinationInterface*, and then the constant value/description “COMP” in *DestinationProperty*. Thus our label becomes:

```
<?xml version="1.0" encoding="UTF-8"?>
<REPORT_QUERY
  Name="Piping IsoMTO RecordType"
  Description=""
  RequiresFilter="No">
  <DESIGN_TIME
    Progid="SP3DReportsQueryBuilder.COMQuery"
    Action=""
    Arg="" />

  <RUN_TIME
    Progid="SP3DRuntimeQuery.CQueryInterpreter"
    Action=""
    Arg="" />
```

```

<RETURNED_PROPERTIES>
  <RETURNED_PROPERTY
    Name="LBL"
    SQLType="Bstr">
    <PATHS>

      <PATH
        SourceType="IJRteCompOccur"
        DestinationInterface="CONSTANT"
        DestinationProperty="COMP" />

    </PATHS>
  </RETURNED_PROPERTY>
</RETURNED_PROPERTIES>
</REPORT_QUERY>

```

Similarly, we find the following PATH's for the other object types:

Pipe Parts:

```

<PATH
  SourceType="IJRtePipeOccur"
  DestinationInterface="CONSTANT"
  DestinationProperty="PIPE" />

```

Piping Instruments:

```

<PATH
  SourceType="IJRteInstrumentOccur"
  DestinationInterface="CONSTANT"
  DestinationProperty="INST" />

```

Piping Specialty Items:

```

<PATH
  SourceType="IJRteSpecialtyOccur"
  DestinationInterface="CONSTANT"
  DestinationProperty="SPCL" />

```

Support Assemblies:

```

<PATH
  SourceType="IJHgrSupportAssemblyItem"
  DestinationInterface="CONSTANT"
  DestinationProperty="SUPP" />

```

To make the label work for any one of those, we just put their respective paths between the <PATHS> </PATHS> nodes. Also, note that those different paths are all under a single <RETURNED_PROPERTY> node. Given an object, the software will try to execute the instructions in the first <PATH> node under each property (<RETURNED_PROPERTY> node) when trying to extract the value of that property. If the object is not of the SourceType of the first <PATH> it will try the second. If the second one fails too, it will then try the 3rd and so on. (This is also true if the execution of the PATH statement fails for any other reason.) So, the complete label will look like this:

```

<?xml version="1.0" encoding="UTF-8"?>
<REPORT_QUERY
  Name="Piping IsoMTO RecordType"

```

```
Description=""  
RequiresFilter="No">>  
<DESIGN_TIME  
ProgId="SP3DReportsQueryBuilder.COMQuery"  
Action=""  
Arg="" />  
  
<RUN_TIME  
ProgId="SP3DRuntimeQuery.CQueryInterpreter"  
Action=""  
Arg="" />  
  
<RETURNED_PROPERTIES>  
<RETURNED_PROPERTY  
Name="LBL"  
SQLType="Bstr">  
<PATHS>  
  
  <PATH  
  SourceType="IJRteCompOccur"  
  DestinationInterface="CONSTANT"  
  DestinationProperty="COMP" />  
  
  <PATH  
  SourceType="IJRtePipeOccur"  
  DestinationInterface="CONSTANT"  
  DestinationProperty="PIPE" />  
  
  <PATH  
  SourceType="IJRteInstrumentOccur"  
  DestinationInterface="CONSTANT"  
  DestinationProperty="INST" />  
  
  <PATH  
  SourceType="IJRteSpecialtyOccur"  
  DestinationInterface="CONSTANT"  
  DestinationProperty="SPCL" />  
  
  <PATH  
  SourceType="IJHgrSupportAssemblyItem"  
  DestinationInterface="CONSTANT"  
  DestinationProperty="SUPP" />  
  
  </PATHS>  
</RETURNED_PROPERTY>  
</RETURNED_PROPERTIES>  
</REPORT_QUERY>
```

2. Virtual relationships

Objective: Create a label for a Pipe Part or a Hanger Support Assembly (Hanger Support) that returns the name of their parent Pipeline system.

Solution:

It should be pretty obvious that the Pipeline system name is neither a direct property of nor one relationship away from Pipe Parts and Hanger Supports (you can verify this by exploring the “Select Properties” dialog box in the label editor). This means that we’ll have to do some work “by hand”. Since we can start from 2 different types of objects (Pipe Parts and Hanger Supports), our label will have 2 different paths.

Let’s first do the path from Hanger Support Assembly to the Pipeline System.

If we look at the graph of basic relationships, we see that the path from the Hanger Support Assembly to the Pipeline System is as follows:

Hanger Support – Piping Feature – Pipe Run – Pipeline System.

We can start with an empty COM label template as in the previous example, or get a head start with the GUI Label Editor by traversing the first part of our relationship map from *Support Assembly* to *Piping Feature* (fig. 4). For now, pick any *Piping Feature* property – we’ll modify it later so it doesn’t matter (we have picked *Approval Status* here)

Let’s open the created label and examine it.

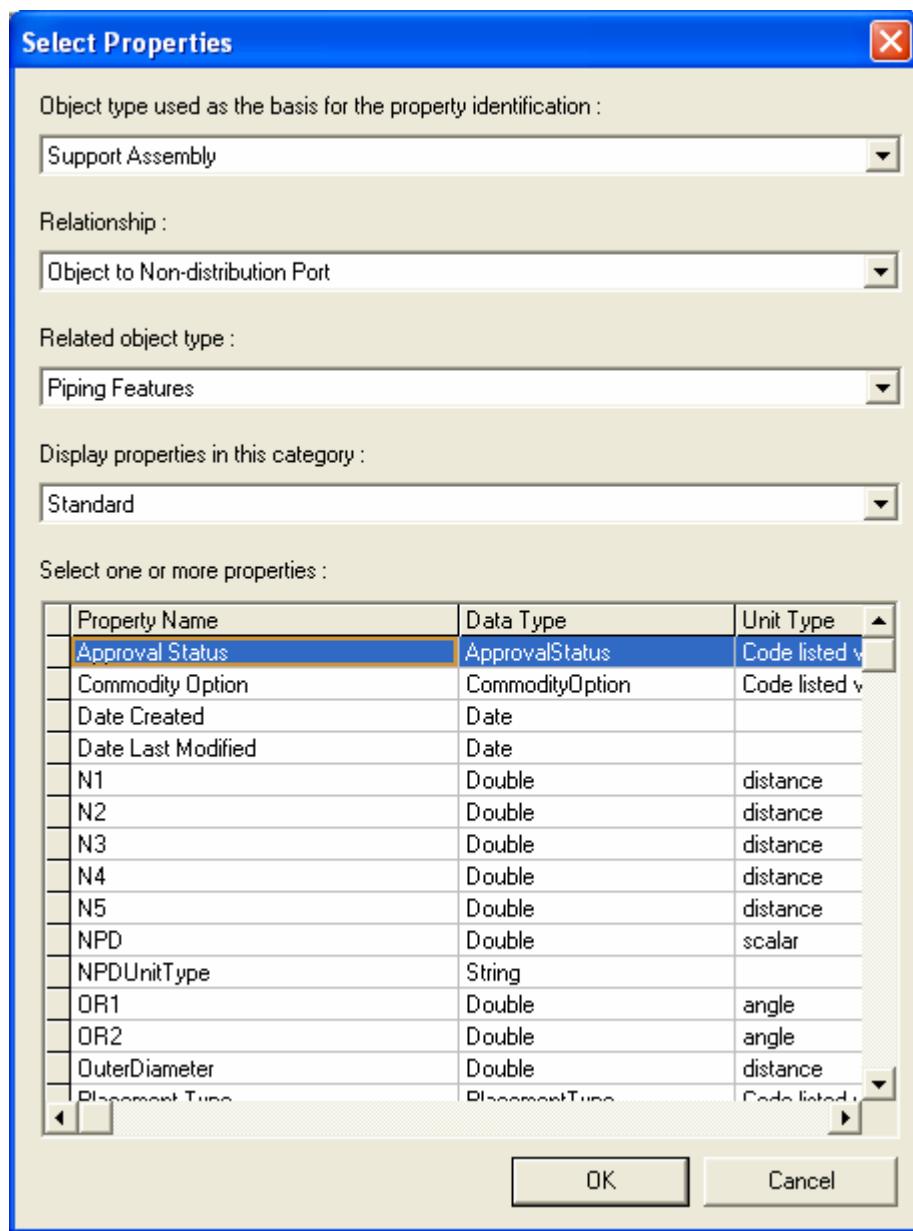


fig. 4

```

<?xml version="1.0" encoding="UTF-8"?>
<REPORT_QUERY
Name=" Piping IsoMTO LineId "
Description="Common Record Data: Line ID - Pipeline Name"
RequiresFilter="No">
<DESIGN_TIME
ProgId="SP3DReportsQueryBuilder.COMQuery"
Action=""
Arg="" />

<RUN_TIME
ProgId="SP3DRuntimeQuery.CQueryInterpreter"
Action="">

```

```

Arg="" />

<RETURNED_PROPERTIES>
  <RETURNED_PROPERTY
    Name="Approval Status"
    SQLType="BStr">
    <PATHS>
      <PATH
        SourceType="IJAppConnection"
        DestinationInterface="IJDBObject"
        DestinationProperty="ApprovalStatus"
        Concatenate="No"
        PathSeparator="">
        <STROKES>
          <STROKE
            Interface="IJAppConnection"
            RelationCollection="ConnHasPorts_ORIG"
            Recursive="No"
            Filter="First"
            IsVirtualRelationship="No" />
        </STROKES>
      </PATH>
    </PATHS>
  </RETURNED_PROPERTY>
</RETURNED_PROPERTIES>
</REPORT_QUERY>

```

The text highlighted in **magenta** will have to be changed:

1. Let's change the **RETURNED_PROPERTY Name** to "Pipeline Name", since that's what we are trying to extract.
2. The source type *IJAppConnection* is too broad. If you see what objects implement it in the *Schema Browser* tool, you'll see that it's not only *Hanger Supports*. (fig. 5)

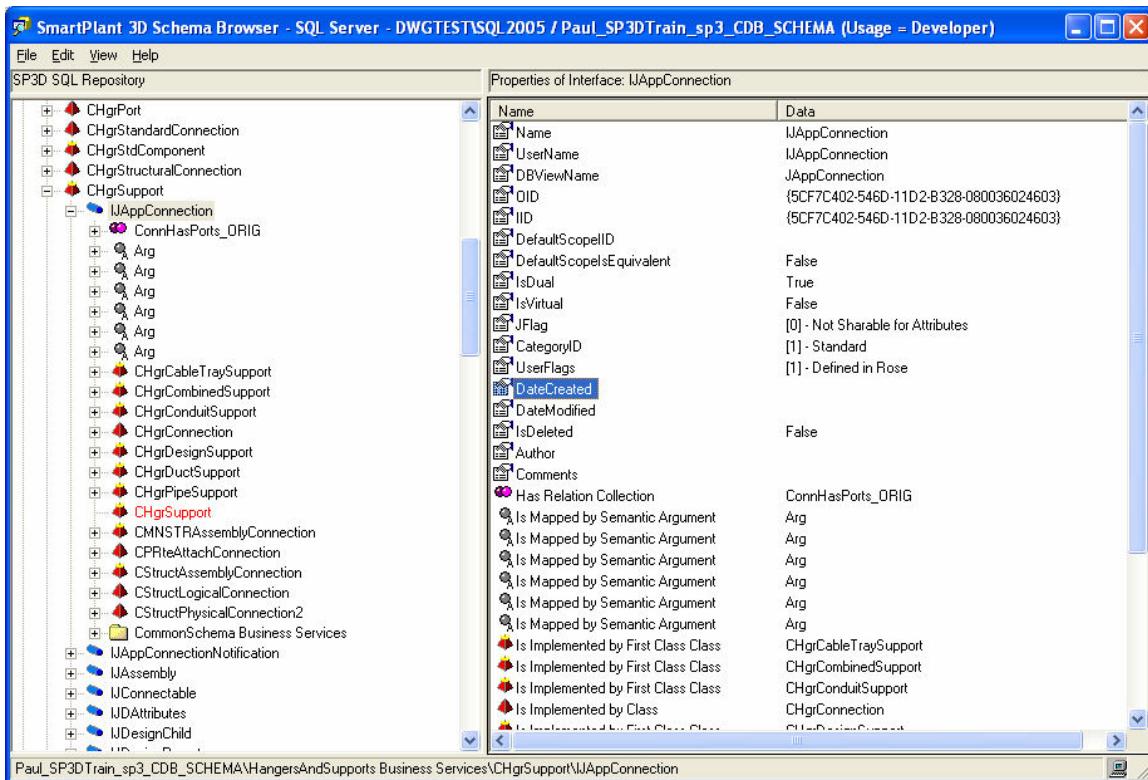


fig. 5

As we saw in example 1, a better qualifying interface is *IJHgrSupportAssemblyItem*.

3. We already know that names of objects are stored in the *Name* property of *IJNamedItem* interface. Thus the *DestinationInterface* should be “*IJNamedItem*” and *DestinationProperty* – “*Name*”

Applying those changes we get:

```
<?xml version="1.0" encoding="UTF-8"?>
<REPORT_QUERY
Name=" Piping IsoMTO LineId "
Description="Common Record Data: Line ID - Pipeline Name"
RequiresFilter="No">
<DESIGN_TIME
ProgId="SP3DReportsQueryBuilder.COMQuery"
Action=""
Arg="" />

<RUN_TIME
ProgId="SP3DRuntimeQuery.CQueryInterpreter"
Action=""
Arg="" />

<RETURNED_PROPERTIES>
<RETURNED_PROPERTY
Name="Pipeline Name"
```

```

SQLType="BStr">
<PATHS>
  <PATH
    SourceType="IJHgrSupportAssemblyItem"
    DestinationInterface=" IJNamedItem"
    DestinationProperty="Name"
    Concatenate="No"
    PathSeparator="">
    <STROKES>
      <STROKE
        Interface="IJAppConnection"
        RelationCollection="ConnHasPorts_ORIG"
        Recursive="No"
        Filter="First"
        IsVirtualRelationship="No" />
    </STROKES>
  </PATH>
</PATHS>
</RETURNED_PROPERTY>
</RETURNED_PROPERTIES>
</REPORT_QUERY>

```

Notice the <STROKE> tag – it traverses the first relationship in our “route”. The most important attributes are *Interface*, which designates the starting interface (the FROM part), and *RelationCollection*, which gives us the destination (TO part). Let’s look at the graph of basic relationships that depicts that relationship (fig. 6)

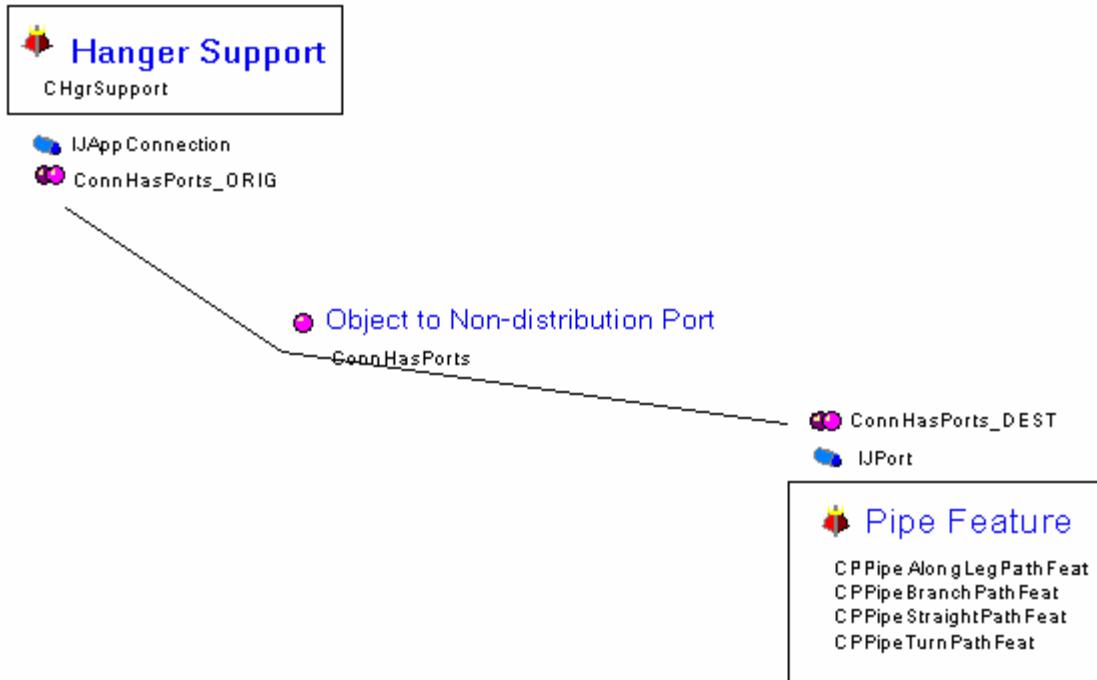


fig. 6

`ConnHasPorts_ORIG` and `ConnHasPorts_DEST` are called `RelationCollection`. They represent the 2 ends of a relationship. Although `ConnHasPorts_ORIG` is written next to the `Hanger Support` class, it actually gives us information about what kind of objects we will get at the OTHER end of the relationship if we start from a `Hanger Support`. Thus the name “collection” – the

RelationCollection at the end A of a particular relationship represent the collection of objects related through that particular relationship to the object at end A, or, in other words, the collection of objects at the other end (end "B") of the relationship. Although we say collection, often that collection can contain 1 object or even 0 related objects.

So, to make a COM label traverse a relationship from "A" to "B", we write a <STROKE> statement where we plug in the Interface at end "A" for the Interface attribute, and the RelationCollection at end "A" for the RelationCollection attribute. In our case this will give us a collection (most likely of count = 1) of Piping Features. The other attributes in the statement tell us that we do not want to traverse this same relationship recursively, that we just want to get the first object in that collection, and that the relationship we are traversing is marked in the Metadata as a real relationship, not virtual (more on that later).

So far we have started from a Hanger Support and found its related Piping Feature. Now let's write the STROKE tag for the next relationship traversal – from Piping Feature to Pipe Run. Those 2 are connected by the relationship Run to Feature (user name). The Interface at the Piping Feature end is IJRtePathFeat, and the relation collection at the Piping Feature is 'thePathSystemInfo'. Also, there's no need for recursion, and the relationship is not marked as virtual, so we'll have

```
<STROKE
  Interface="IJRtePathFeat"
  RelationCollection="thePathSystemInfo"
  Recursive="No"
  Filter="First"
  IsVirtualRelationship="No" />
```

Similarly, we'll get the following <STROKE> tag for the last relationship traversal from Pipe Run to Pipeline System:

```
<STROKE
  Interface="IJSystemChild"
  RelationCollection="SystemParent"
  Recursive="No"
  Filter="First"
  IsVirtualRelationship="No" />
```

This will make our label so far look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<REPORT_QUERY
  Name=" Piping IsoMTO LineId "
  Description="Common Record Data: Line ID - Pipeline Name"
  RequiresFilter="No">
<DESIGN_TIME
  Progid="SP3DReportsQueryBuilder.COMQuery"
  Action=""
  Arg="" />

<RUN_TIME
  Progid="SP3DRuntimeQuery.CQueryInterpreter"
  Action=""
  Arg="" />

<RETURNED_PROPERTIES>
<RETURNED_PROPERTY
  Name="Pipeline Name"
  SQLType="BStr">
<PATHS>
  <PATH
    SourceType="IJHgrSupportAssemblyItem"
    DestinationInterface=" IJNamedItem"
```

```

DestinationProperty="Name"
Concatenate="No"
PathSeparator="">
<STROKES>
<STROKE
Interface="IJAppConnection"
RelationCollection="ConnHasPorts_ORIG"
Recursive="No"
Filter="First"
IsVirtualRelationship="No" />
<STROKE
Interface="IJRtePathFeat"
RelationCollection="thePathSystemInfo"
Recursive="No"
Filter="First"
IsVirtualRelationship="No" />
<STROKE
Interface="IJSystemChild"
RelationCollection="SystemParent"
Recursive="No"
Filter="First"
IsVirtualRelationship="No" />
</STROKES>
</PATH>
</PATHS>
</RETURNED_PROPERTY>
</RETURNED_PROPERTIES>
</REPORT_QUERY>

```

This will take care of finding the name of the parent Pipeline for a given Hanger Support. Now if we want to make this work for Piping Parts too, we need to add another <PATH>. The DestinationInterface and DestinationProperty attributes should be the same since we are still trying to get the name of the parent Pipeline, but the SourceType should be an interface implemented only by Piping Parts. From example 1 we already know that one such interface is IJRtePipeOccur. Note that there could be other such uniquely identifying interfaces, and it is okay to use them instead of IJRtePipeOccur. IJRtePipeOccur is used here because of the ease with which it can be inferred from the Schema Browser tool (see example 1).

This will make just the <PATH> part from Piping Part to Pipeline appear like this:

```

<PATH
SourceType=" IJRtePipeOccur "
DestinationInterface="IJNamedItem"
DestinationProperty="Name"
Concatenate="No"
PathSeparator="">
<STROKES>
<STROKE>
</STROKE>
</STROKES>
</PATH>

```

We still need to fill in the STROKE portion. If we look at the basic relationship map, we see that we can go from Piping Part to Pipe Run to Pipeline System. However, there is a shortcut – and ‘edge’ from Piping Part to Pipeline System (red line). An edge, also called virtual relationship, is a kind of bridge that spans 2 or more relationships mainly for the purpose of making navigating relationships easier. Although there are differences between a virtual and non-virtual relationship, for the purposes of label and

reports queries, they can be considered equal. The only thing we have to make differently is set the IsVirtualRelationship attribute to "Yes". The STROKE tag thus becomes:

```
<STROKE
  Interface="IJRtePathGenPart"
  RelationCollection="Pipeline"
  Recursive="No"
  Filter="First"
  IsVirtualRelationship="Yes" />
```

And the whole label is:

```
<?xml version="1.0" encoding="UTF-8"?>
<REPORT_QUERY
  Name=" Piping IsoMTO LineId "
  Description="Common Record Data: Line ID - Pipeline Name"
  RequiresFilter="No">
  <DESIGN_TIME
    Progid="SP3DReportsQueryBuilder.COMQuery"
    Action=""
    Arg="" />

  <RUN_TIME
    Progid="SP3DRuntimeQuery.CQueryInterpreter"
    Action=""
    Arg="" />

  <RETURNED_PROPERTIES>
    <RETURNED_PROPERTY
      Name="Pipeline Name"
      SQLType="BStr">
      <PATHS>
        <PATH
          SourceType="IJHgrSupportAssemblyItem"
          DestinationInterface=" IJNamedItem"
          DestinationProperty="Name"
          Concatenate="No"
          PathSeparator="">
          <STROKES>
            <STROKE
              Interface="IJAppConnection"
              RelationCollection="ConnHasPorts_ORIG"
              Recursive="No"
              Filter="First"
              IsVirtualRelationship="No" />
            <STROKE
              Interface="IJRtePathFeat"
              RelationCollection="thePathSystemInfo"
              Recursive="No"
              Filter="First"
              IsVirtualRelationship="No" />
            <STROKE
              Interface="IJSystemChild"
              RelationCollection="SystemParent"
```

```

Recursive="No"
Filter="First"
IsVirtualRelationship="No" />
</STROKES>
</PATH>
<PATH
  SourceType=" IJRtePipeOccur "
  DestinationInterface="IJNamedItem"
  DestinationProperty="Name"
  Concatenate="No"
  PathSeparator="">
  <STROKES>
<STROKE
  Interface="IJRtePathGenPart"
  RelationCollection="Pipeline"
  Recursive="No"
  Filter="First"
  IsVirtualRelationship="Yes" />
  </STROKES>
</PATH>
</PATHS>
</RETURNED_PROPERTY>
</RETURNED_PROPERTIES>
</REPORT_QUERY>

```

3. Recursion with *Implements* exitcondition

Objective: Create a label for a Piping Part or a Hanger Support Assembly (Hanger Support) that returns the name of their parent Unit system.

Solution:

In example 2 we found out how to navigate from a Piping Part or a Hanger Support to the parent Pipeline System, so now we have to extend that path all the way to a Unit System.

If we look at the relationship map for system hierarchies, we see that it is actually very simple. Any 2 systems can be related by the System Hierarchy relationship (fig 7)

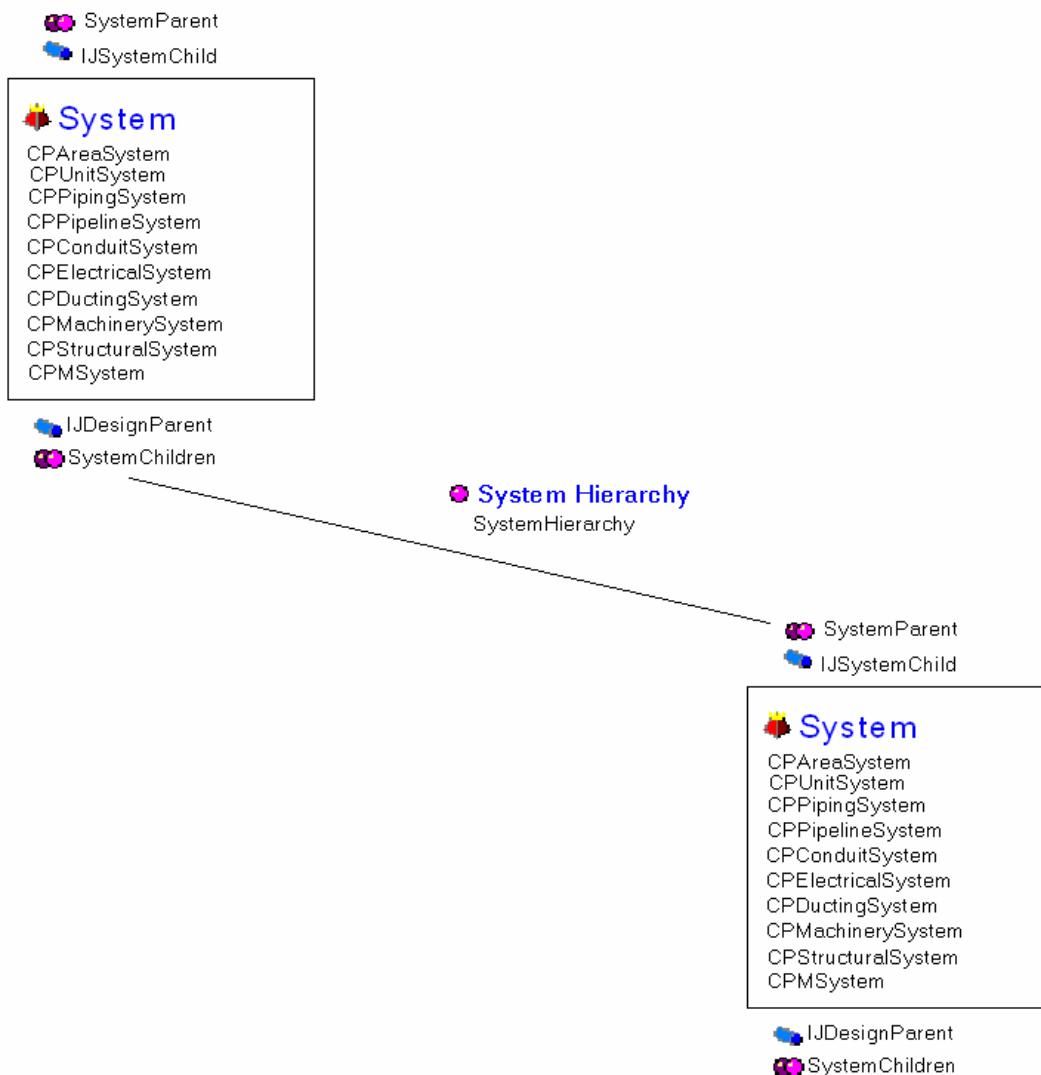


fig. 7 System hierarchy

While this simplifies our task, the fact that all those systems can be arranged and nested in all possible ways (we could actually not have a Unit System at all), can seem unnerving. Fortunately, we have just the right tool to deal with this situation – the recursive traversal of a relationship.

Let's start with the label from example 2:

```
<?xml version="1.0" encoding="UTF-8"?>
<REPORT_QUERY
Name=" Piping IsoMTO LineId "
Description="Common Record Data: Line ID - Pipeline Name"
RequiresFilter="No">
<DESIGN_TIME
ProgId="SP3DReportsQueryBuilder.COMQuery"
Action=""
Arg="" />

<RUN_TIME
```

```

ProgId="SP3DRuntimeQuery.CQueryInterpreter"
Action=""
Arg="" />

<RETURNED_PROPERTIES>
<RETURNED_PROPERTY
  Name="Pipeline Name"
  SQLType="BStr">
  <PATHS>
    <PATH
      SourceType="IJHgrSupportAssemblyItem"
      DestinationInterface=" IJNamedItem"
      DestinationProperty="Name"
      Concatenate="No"
      PathSeparator="">
      <STROKES>
        <STROKE
          Interface="IJAppConnection"
          RelationCollection="ConnHasPorts_ORIG"
          Recursive="No"
          Filter="First"
          IsVirtualRelationship="No" />
      <STROKE
        Interface="IJRtePathFeat"
        RelationCollection="thePathSystemInfo"
        Recursive="No"
        Filter="First"
        IsVirtualRelationship="No" />
    <STROKE
      Interface="IJSystemChild"
      RelationCollection="SystemParent"
      Recursive="No"
      Filter="First"
      IsVirtualRelationship="No" />
    </STROKES>
  </PATH>
<PATH
  SourceType=" IJRtePipeOccur "
  DestinationInterface="IJNamedItem"
  DestinationProperty="Name"
  Concatenate="No"
  PathSeparator="">
  <STROKES>
    <STROKE
      Interface="IJRtePathGenPart"
      RelationCollection="Pipeline"
      Recursive="No"
      Filter="First"
      IsVirtualRelationship="Yes" />
    </STROKES>
  </PATH>
</PATHS>
</RETURNED_PROPERTY>
</RETURNED_PROPERTIES>
</REPORT_QUERY>

```

We see that the last stroke in the traversal from Hanger Support to Pipeline system is already a traversal of the System Hierarchy relationship. From Pipe Run up in the system hierarchy, all objects will be related by the System Hierarchy relationship. Instead of writing another <STROKE> which will look exactly the same as the one in question (since we are traversing the same relationship in the same direction), we can set the Recursive attribute to “Yes”. What happens in this case is the software traverses the System Hierarchy relationship from Pipe Run to Pipeline System, then tries to traverse the System Hierarchy relationship from the Pipeline System to its parent, and so on. We also need to indicate that we want to stop this recursion when we reach a Unit System object. We accomplish this by using the attributes ExitCondition = “Implements” in addition to ExitValue = “IJUnitSystem”. IJUnitSystem is a uniquely identifying interface for Unit System, which we get from Classification TopNodes (see example 1). The new STROKE portion will look like:

```
<STROKE
  Interface="IJSystemChild"
  RelationCollection="SystemParent"
  Recursive="Yes"
  ExitCondition="Implements"
  ExitValue="IJUnitSystem"
  Filter="First"
  IsVirtualRelationship="No" />
```

Following the same logic for the second path from Piping Part to Pipeline system, we need to add the same <STROKE> tag to cover the navigation from Pipeline system to Unit System. The whole label will look like:

```
<?xml version="1.0" encoding="UTF-8"?>
<REPORT_QUERY
  Name=" UnitCode "
  Description=" PipeComponent Unitcode to systemhierarchy relation to unitsystem "
  RequiresFilter="No">
  <DESIGN_TIME
    Progid="SP3DReportsQueryBuilder.COMQuery"
    Action=""
    Arg="" />

  <RUN_TIME
    Progid="SP3DRuntimeQuery.CQueryInterpreter"
    Action=""
    Arg="" />

  <RETURNED_PROPERTIES>
    <RETURNED_PROPERTY
      Name="Pipeline Name"
      SQLType="BStr">
      <PATHS>
        <PATH
          SourceType="IJHgrSupportAssemblyItem"
          DestinationInterface=" IJNamedItem"
          DestinationProperty="Name"
          Concatenate="No"
          PathSeparator="">
          <STROKES>
            <STROKE
              Interface="IJAppConnection"
              RelationCollection="ConnHasPorts_ORIG"
              Recursive="No"
```

```
    Filter="First"
    IsVirtualRelationship="No" />
<STROKE
    Interface="IJRtePathFeat"
    RelationCollection="thePathSystemInfo"
    Recursive="No"
    Filter="First"
    IsVirtualRelationship="No" />
<STROKE
    Interface="IJSystemChild"
    RelationCollection="SystemParent"
    Recursive="Yes"
    ExitCondition="Implements"
    ExitValue="IJUnitSystem"
    Filter="First"
    IsVirtualRelationship="No" />

    </STROKES>
</PATH>
<PATH
    SourceType=" IJRtePipeOccur "
    DestinationInterface="IJNamedItem"
    DestinationProperty="Name"
    Concatenate="No"
    PathSeparator="">
    <STROKES>
<STROKE
    Interface="IJRtePathGenPart"
    RelationCollection="Pipeline"
    Recursive="No"
    Filter="First"
    IsVirtualRelationship="Yes" />
<STROKE
    Interface="IJSystemChild"
    RelationCollection="SystemParent"
    Recursive="Yes"
    ExitCondition="Implements"
    ExitValue="IJUnitSystem"
    Filter="First"
    IsVirtualRelationship="No" />

    </STROKES>
</PATH>
</PATHS>
</RETURNED_PROPERTY>
</RETURNED_PROPERTIES>
</REPORT_QUERY>
```

4. Using user interface/attributes

Objective: Create a label for Spectacle Blind Piping Component that returns the Used Defined attribute Spectacle Position.

Solution:

We can create this label from the UI. In the Label Editor/Add Property dialog select under Piping/Piping Parts/Piping Components Spectacle Blind. The property Spectacle Position, which is on a User Defined interface, will be shown in the property list (fig 8.)

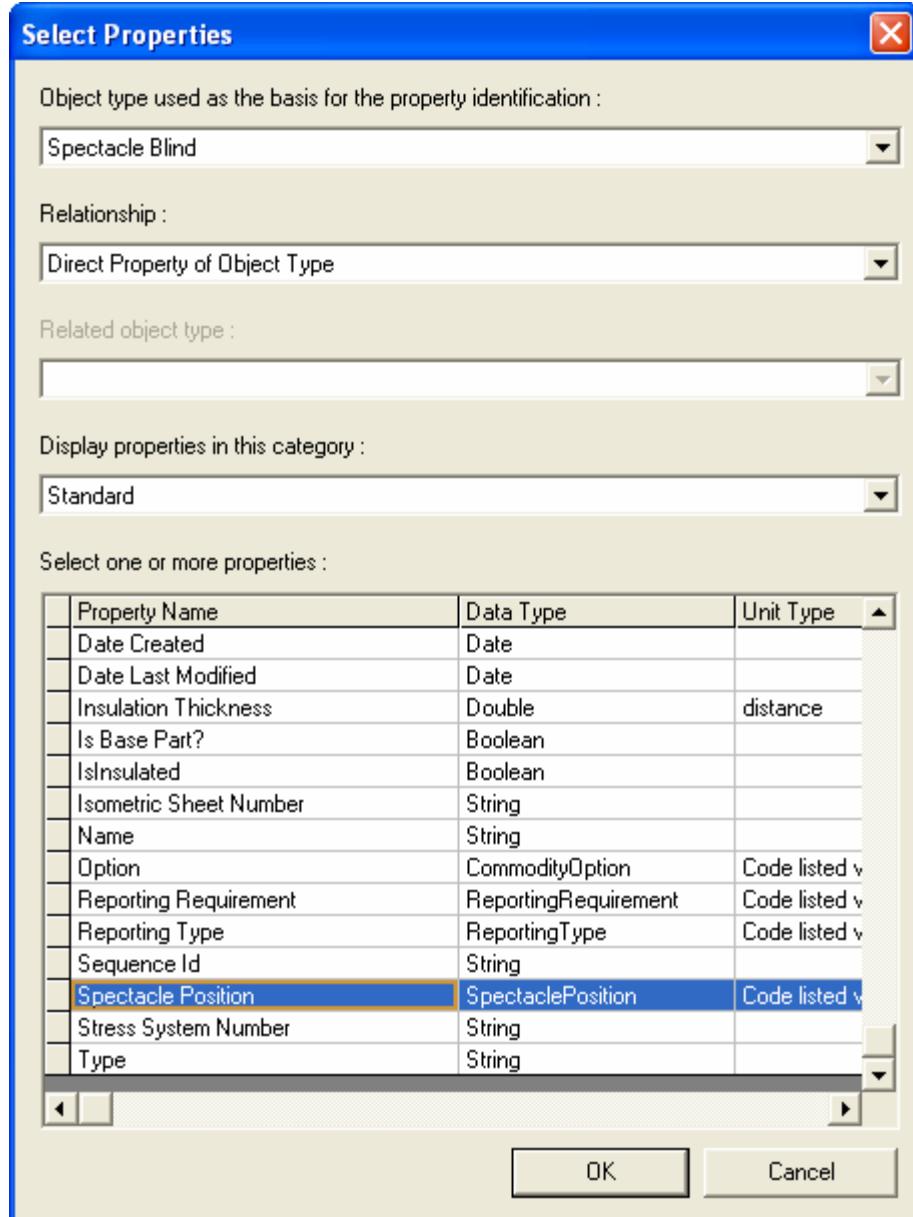


Fig 8. Property page for Spectacle Blind

SP3D will show used defined attributes just the same as system defined one, but we do need to select in the Object Type hierarchy the particular occurrence class (Spectacle blind), as opposed to stopping at the system class Piping Component.

5. Using embedded labels

Objective: Create a label that returns the name of a Piping Part. Create a second label that returns the Cut Length of a Piping Part. Now create a 3rd label that returns both the Name and Cut Length for a Piping Part by directly using (embedding) the output of the first 2 labels.

Solution:

SP3D provides a mechanism for directly embedding the output of other labels. Our example is very simple and, of course, can be achieved through just picking the 2 properties instead of embedding 2 labels, but this mechanism can be very useful in more complicated situations.

Let's first create a label returning the Piping Part Name. Name the label "NameLbl" and the Alias Name - :Pipe_Name". We can do this using the UI. This should be pretty straightforward by now:

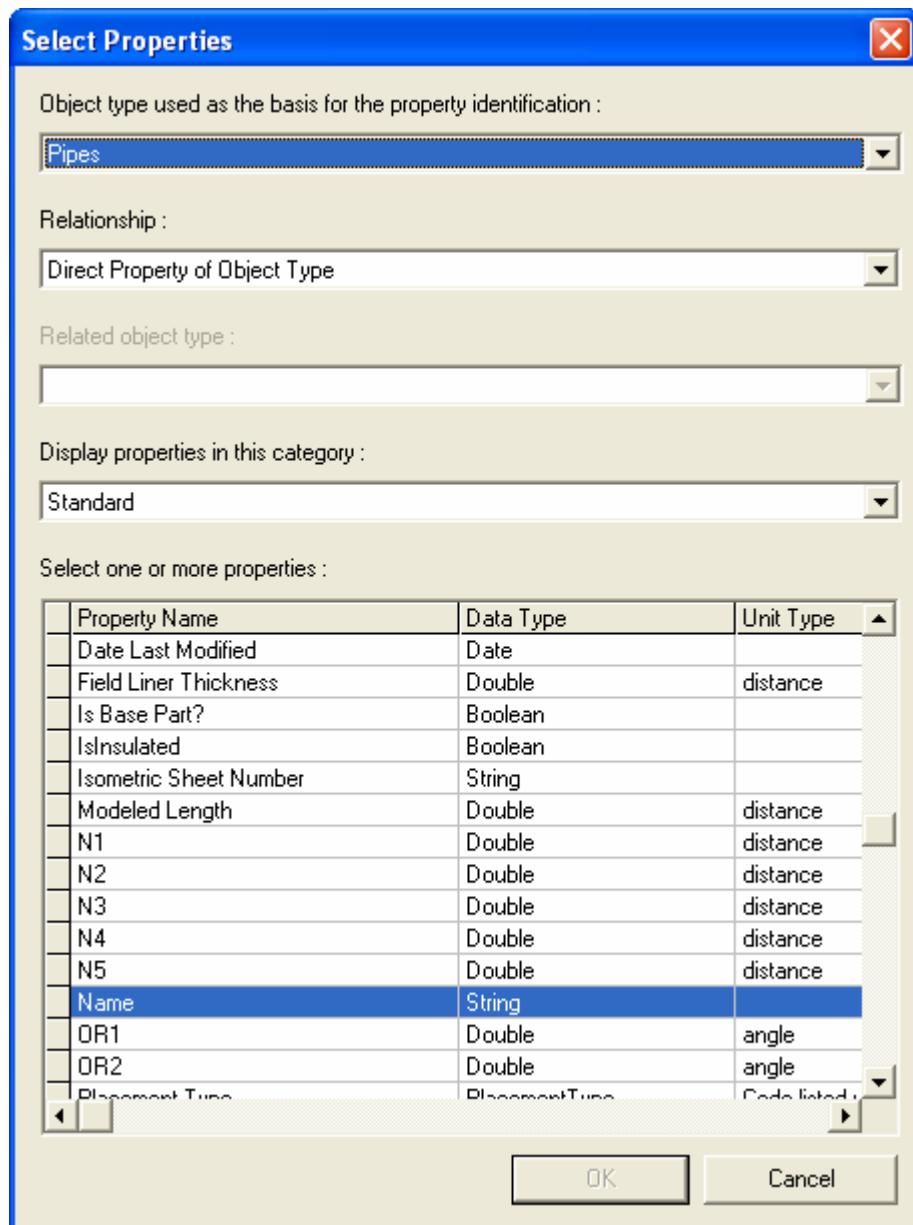


Fig.9

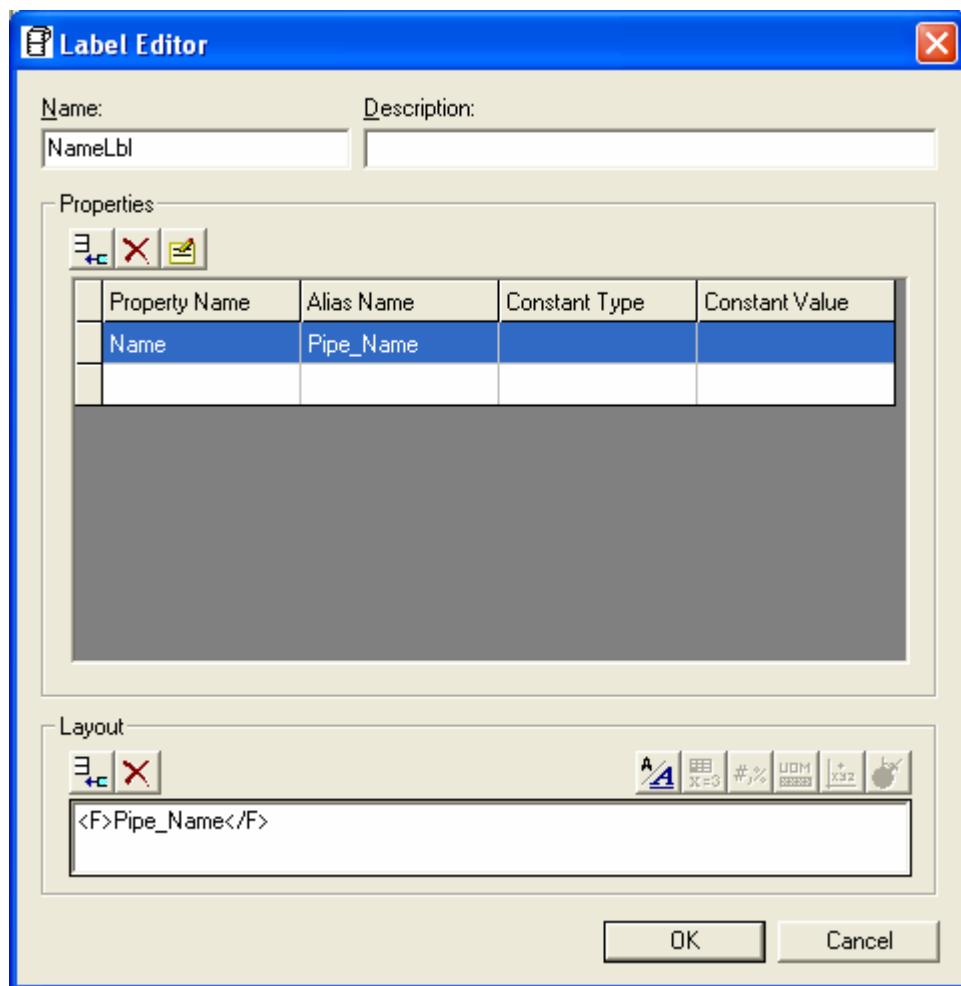


Fig. 10

Now let's create the second label that returns the Cut Length. We'll Name it "CutLengthLbl":

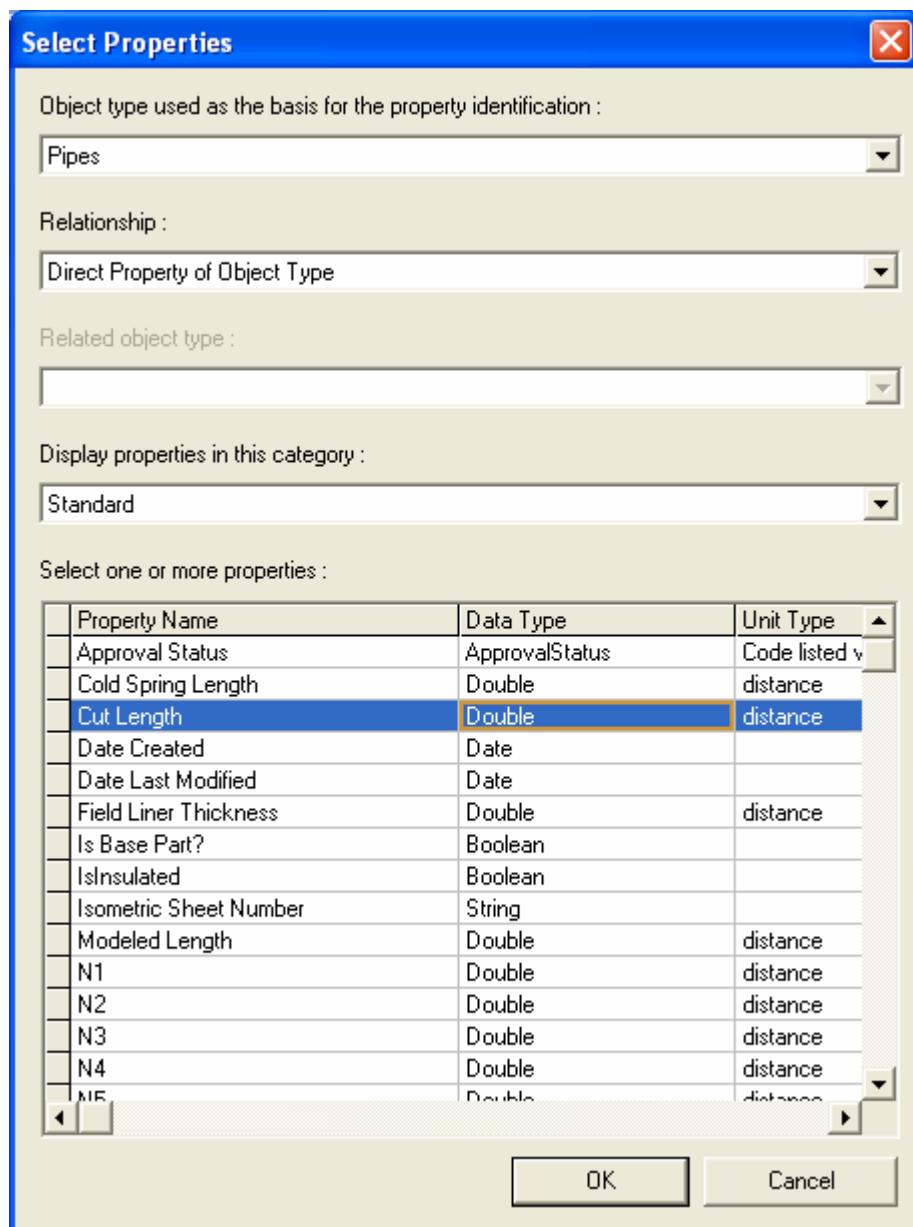


Fig. 11

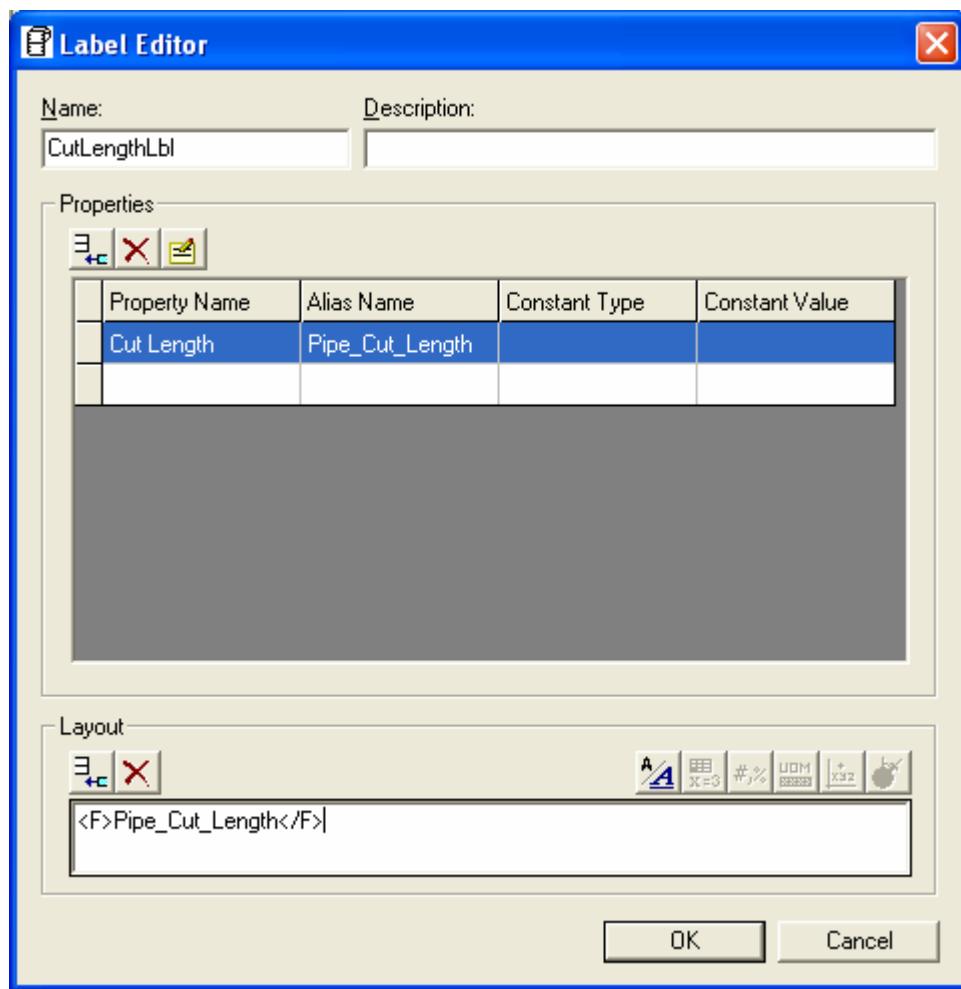


Fig. 12

Now let's create the 3rd label as shown. Note that you need to select "CONSTANT" for Property Name from the drop down combo. (Fig. 13)

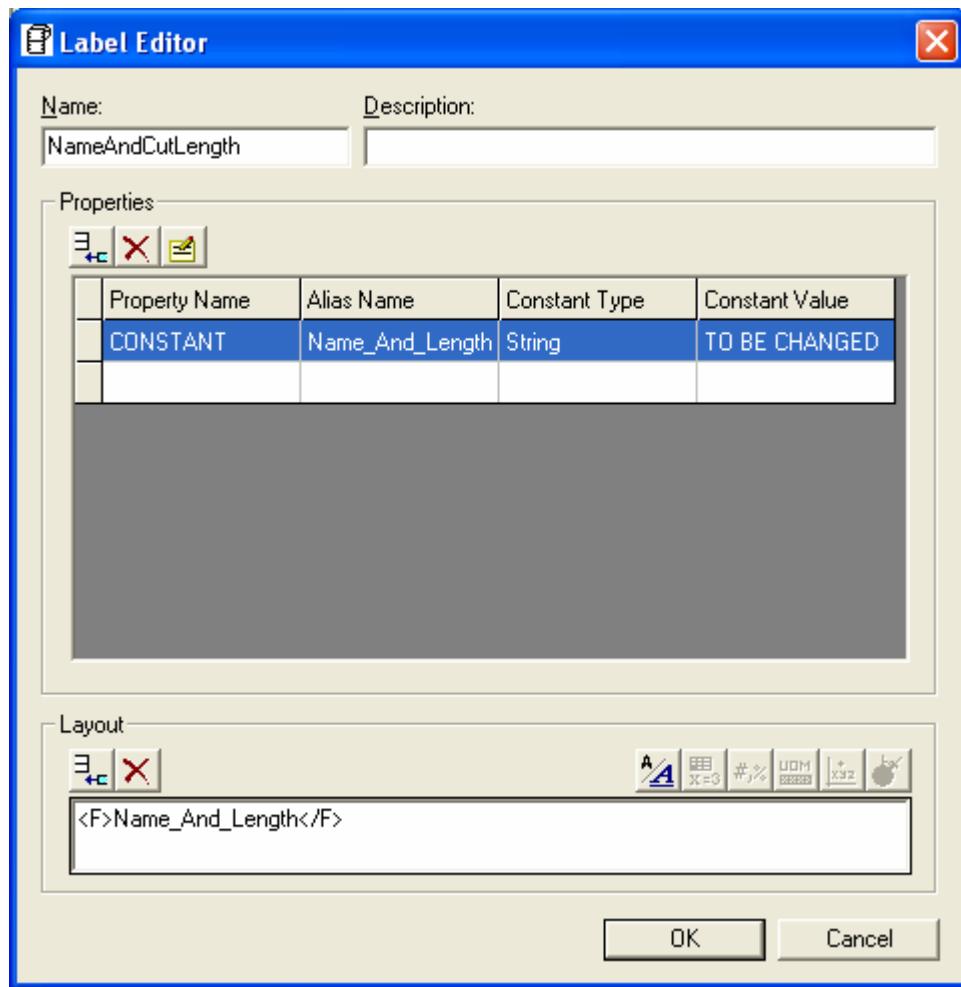


Fig. 13

Note that "Constant Type" has to be String!

Now go to the <SymbolsShare>\Labels\Paul\NameAndCutLength and open the .rqe file. It should look like:

```
<?xml version="1.0" encoding="UTF-8"?>
<REPORT_QUERY
Name="NameAndCutLength"
Description=""
RequiresFilter="No">
<DESIGN_TIME
ProgId="SP3DReportsQueryBuilder.COMQuery"
Action=""
Arg="" />

<RUN_TIME
ProgId="SP3DRuntimeQuery.CQueryInterpreter"
Action=""
Arg="" />

<RETURNED_PROPERTIES>
<RETURNED_PROPERTY
Name="Name_And_Length"
```

```
SQLType="BStr">
<PATHS>
<PATH
SourceType="*"
DestinationInterface="CONSTANT"
DestinationProperty="TO BE CHANGED"
Concatenate="No"
PathSeparator="" />
</PATHS>
</RETURNED_PROPERTY>
</RETURNED_PROPERTIES>
</REPORT_QUERY>
```

Make the following change:

```
DestinationProperty="Name: [NameLbl] Cut Length: [CutLengthLbl]"
```

This tells the software to insert in the string the output of the label(s) enclosed in [].

The NameAndCutLength.rqe will now look like:

```
<?xml version="1.0" encoding="UTF-8"?>
<REPORT_QUERY
Name="NameAndCutLength"
Description=""
RequiresFilter="No">
<DESIGN_TIME
ProgId="SP3DReportsQueryBuilder.COMQuery"
Action=""
Arg="" />

<RUN_TIME
ProgId="SP3DRuntimeQuery.CQueryInterpreter"
Action=""
Arg="" />

<RETURNED_PROPERTIES>
<RETURNED_PROPERTY
Name="Name_And_Length"
SQLType="BStr">
<PATHS>
<PATH
SourceType="*"
DestinationInterface="CONSTANT"
DestinationProperty="Name: [NameLbl] Cut Length: [CutLengthLbl]"
Concatenate="No"
PathSeparator="" />
</PATHS>
</RETURNED_PROPERTY>
</RETURNED_PROPERTIES>
</REPORT_QUERY>
```

Save it and test it.

Alternatively, you could have just typed :

Name: [NameLbl] Cut Length: [CutLengthLbl]

for the Constant Value in Fig. 13, an not have to open the .rqe file, but knowing what happens “under the hood” can be helpful for more complicated labels.

6. Filter First and Last (Optional)

Objective: Given a connection, get the bolt Short Material Description, bolt Calculated Length and gasket Short Material Description if Flanged connection, or weld type if welded connection.

Solution:

If we look at the Piping Data Model graph, we should by now be able to relatively easily see the relationships we need to navigate. We have to start from the IJDistribConnection interface of the Distributed Connection object and navigate through its ConnectionItems relation collection. Based on the type of the connection, this relation collection can contain a single Weld object, or a Gasket object and a Bolt Set object.

When we deal with relation collections that return us multiple objects, we can use the attribute Filter in the <STROKE> tag to help us differentiate between the objects in the collection. The values are as follows:

Filter = “First”
Filter = “Last”

This will return us respectively the first and last object in the collection. However, it will automatically take into consideration the “Target” interface we are after. The “Target” interface for a given STROKE is either the Interface attribute of the STROKE immediately following it, or the DestinationInterface of the parent PATH if this is the last stroke.

Instead of having the software automatically use the implied target interface, we can explicitly set it by enclosing it in [] immediately following “First” or “Last”. In fact, this is the preferred method. In our case, to get the Bolt description from the Distributed Connection, we would normally write the following PATH:

```
<RETURNED_PROPERTY
  Name="BoltDescription"
  SQLType="BStr">
<PATHS>
  <PATH
    SourceType="IJDistribConnection"
    DestinationInterface="IJBolt"
    DestinationProperty="ShortMaterialDescription"
    Concatenate="No"
    PathSeparator="\\">
    <STROKES>
      <STROKE
        Interface="IJDistribConnection"
        RelationCollection="ConnectionItems"
        Recursive="No"
        IsVirtualRelationship="No"
      />
    <STROKE
      Interface="IJRteConnectionPart"
```

```

    RelationCollection="UsedImpliedPart"
    Recursive="No"
    IsVirtualRelationship="No"
  />
</STROKES>
</PATH>
</PATHS>
</RETURNED_PROPERTY>

```

However, after we have traversed the first relationship, we will have 2 objects in the relation collection – one Bolt Set, and one Gasket. We could use Filter = “First” or Filter = “Last” but we don’t know which is which, and the implied target interface IJRteConnectionPart (from the next STROKE) is supported by both. So we’ll have to explicitly tell the software to give us the First object that supports IJRteBoltSet. We have picked IJRteBoltSet, but it could be any interface supported by the Bolt Set and not supported by the Gasket. The BoltDescription property will then look like:

```

<RETURNED_PROPERTY
Name="BoltDescription"
SQLType="BStr">
<PATHS>
<PATH
SourceType="IJDistribConnection"
DestinationInterface="IJBolt"
DestinationProperty="ShortMaterialDescription"
Concatenate="No"
PathSeparator="\\">
<STROKES>
<STROKE
Interface="IJDistribConnection"
RelationCollection="ConnectionItems"
Recursive="No"
Filter="First|IJRteBoltSet"
IsVirtualRelationship="No"
/>

<STROKE
Interface="IJRteConnectionPart"
RelationCollection="UsedImpliedPart"
Recursive="No"
IsVirtualRelationship="No"
/>
</STROKES>
</PATH>
</PATHS>
</RETURNED_PROPERTY>

```

Note that to get the Bolt Calculated length we need to traverse only one relationship (one STROKE). We can write it as:

```

<RETURNED_PROPERTY
Name="Calculated Length"
SQLType="Double">
<PATHS>
<PATH
SourceType="IJDistribConnection"
DestinationInterface="IJRteBolt"

```

```
DestinationProperty="CalculatedLength"
Concatenate="No"
PathSeparator="\">
<STROKES>
  <STROKE
    Interface="IJDistribConnection"
    RelationCollection="ConnectionItems"
    Recursive="No"
    Filter="First"
    IsVirtualRelationship="No"
  />
</STROKES>
</PATH>
</PATHS>
</RETURNED_PROPERTY>
```

In this case we do not need to qualify the Filter with an interface, because the implied one (DestinationInterface="IJRteBolt") is enough.

After we do similar Filter for the Gasket, the complete label becomes:

```
<?xml version="1.0" encoding="UTF-8"?>
<REPORT_QUERY
  Name="Connection Tooltip"
  Description=""
  RequiresFilter="No">
  <DESIGN_TIME
    Progid="SP3DReportsQueryBuilder.COMQuery"
    Action=""
    Arg="" />

  <RUN_TIME
    Progid="SP3DRuntimeQuery.CQueryInterpreter"
    Action=""
    Arg="" />

  <RETURNED_PROPERTIES>
    <RETURNED_PROPERTY
      Name="BoltDescription"
      SQLType="BStr">
      <PATHS>
        <PATH
          SourceType="IJDistribConnection"
          DestinationInterface="IJBolt"
          DestinationProperty="ShortMaterialDescription"
          Concatenate="No"
          PathSeparator="\">
          <STROKES>
            <STROKE
              Interface="IJDistribConnection"
              RelationCollection="ConnectionItems"
              Recursive="No"
              Filter="First[IJRteBoltSet ]"
```

```
IsVirtualRelationship="No"
/>
<STROKE
Interface="IJRteConnectionPart"
RelationCollection="UsedImpliedPart"
Recursive="No"
Filter="First"
IsVirtualRelationship="No"
/>
</STROKES>
</PATH>
</PATHS>
</RETURNED_PROPERTY>

<RETURNED_PROPERTY
Name="Calculated Length"
SQLType="Double">
<PATHS>
<PATH
SourceType="IJDistribConnection"
DestinationInterface="IJRteBolt"
DestinationProperty="CalculatedLength"
Concatenate="No"
PathSeparator="\\">
<STROKES>
<STROKE
Interface="IJDistribConnection"
RelationCollection="ConnectionItems"
Recursive="No"
Filter="First"
IsVirtualRelationship="No"
/>
</STROKES>
</PATH>
</PATHS>
</RETURNED_PROPERTY>

<RETURNED_PROPERTY
Name="GasketDescription"
SQLType="BStr">
<PATHS>
<PATH
SourceType="IJDistribConnection"
DestinationInterface="IJGasket"
DestinationProperty="ShortMaterialDescription"
Concatenate="No"
PathSeparator="\\">
<STROKES>
<STROKE
Interface="IJDistribConnection"
RelationCollection="ConnectionItems"
Recursive="No"
Filter="First[IJRteGasket]"
IsVirtualRelationship="No"
```

```

    />
<STROKE
  Interface="IJRteConnectionPart"
  RelationCollection="UsedImpliedPart"
  Recursive="No"
  Filter="First"
  IsVirtualRelationship="No"
 />
</STROKES>
</PATH>
</PATHS>
</RETURNED_PROPERTY>

<RETURNED_PROPERTY
  Name="WeldType"
  SQLType="BStr">
<PATHS>
  <PATH
    SourceType="IJDistribConnection"
    DestinationInterface="IJRteWeld"
    DestinationProperty="Type"
    Concatenate="No"
    PathSeparator="\\">
    <STROKES>
      <STROKE
        Interface="IJDistribConnection"
        RelationCollection="ConnectionItems"
        Recursive="No"
        Filter="First"
        IsVirtualRelationship="No"
      />
    </STROKES>
  </PATH>
</PATHS>
</RETURNED_PROPERTY>

</RETURNED_PROPERTIES>
</REPORT_QUERY>

```

Filter can take one more value – “All”. This can be used in cases of 1 to many relationships, for example to display all notes associated to a piping component. This will often be used together with repeatable blocks (more on that in special labels formatting)

7. Recursion with Exit condition Depth (Optional)

Objective: Create a label for Equipment that returns the name of the Equipment, the name of its System Parent, the name of its parent Unit System, and the name of its Area System. Assume that those are nested in the enumerated order.

Solution:

Fig. 14 shows the hierarchy we are asked to assume.

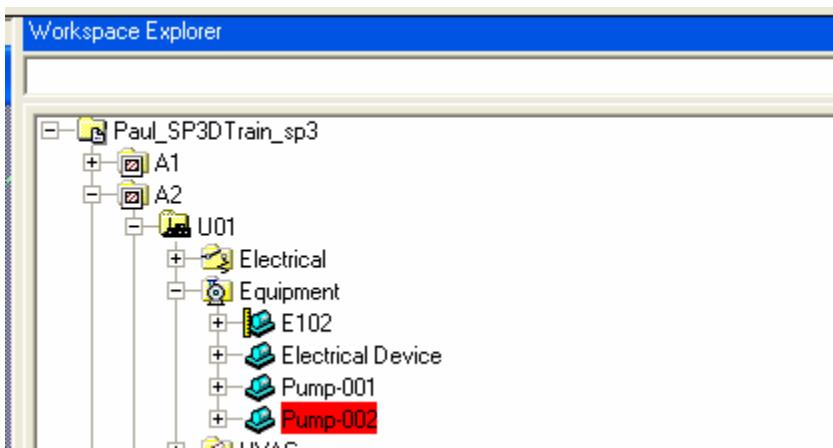


Fig. 14

We should be able to create a label returning at least the first 2 names using the UI, since we have direct property and a single relation traversal. We'll have to manually change the .rqe file to get the last 2 systems in the hierarchy, since that means traversal of more than one relationship.

Let's create the label. You can leave the default Name property as we'll need it anyway.

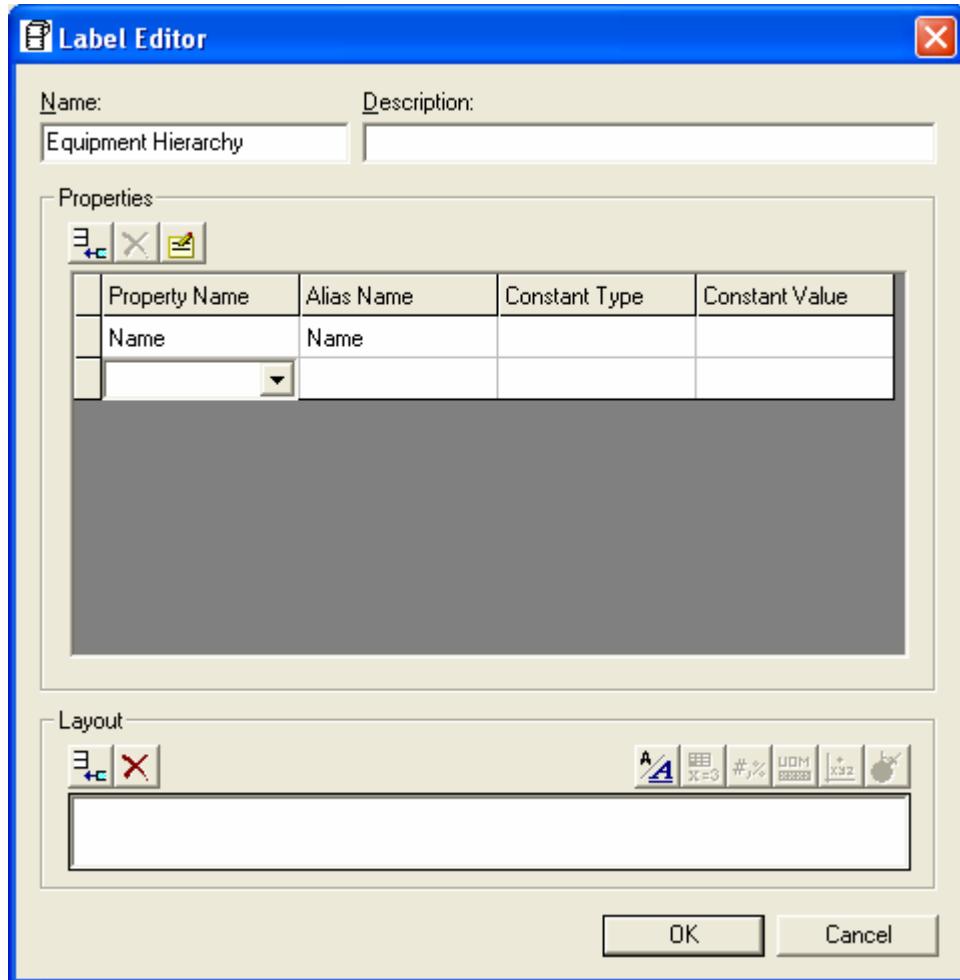


Fig. 15

Add another property. Select for object type Equipment:

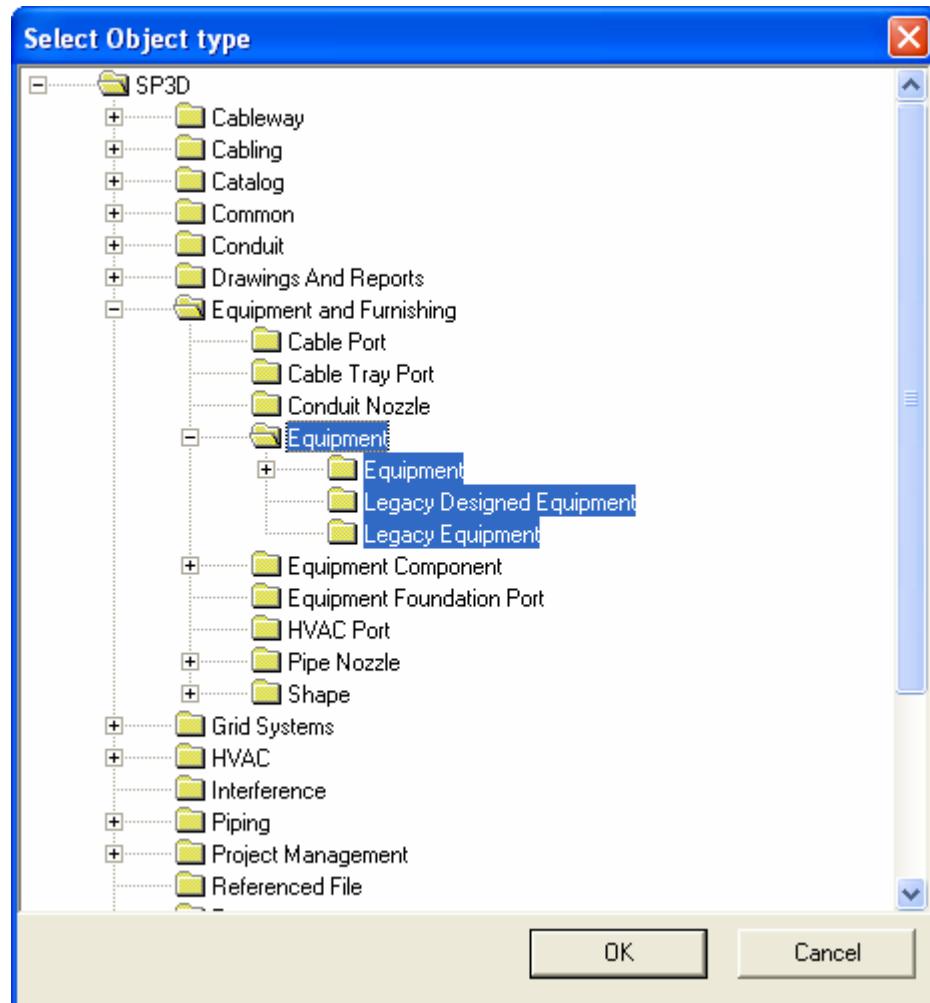


Fig. 16

Select "System to Equipment" in the Relationship drop-down

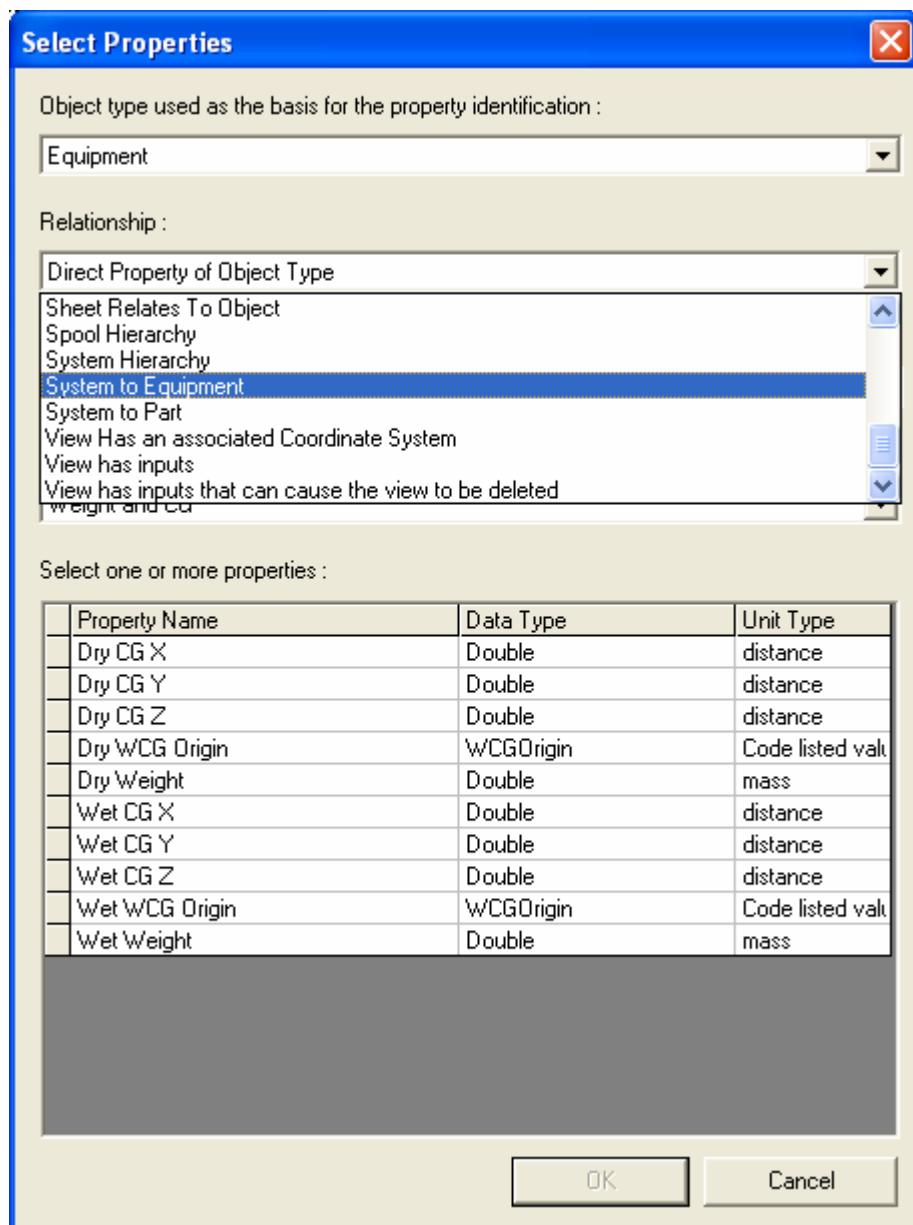


Fig. 17

Select “Equipment System” for related object:

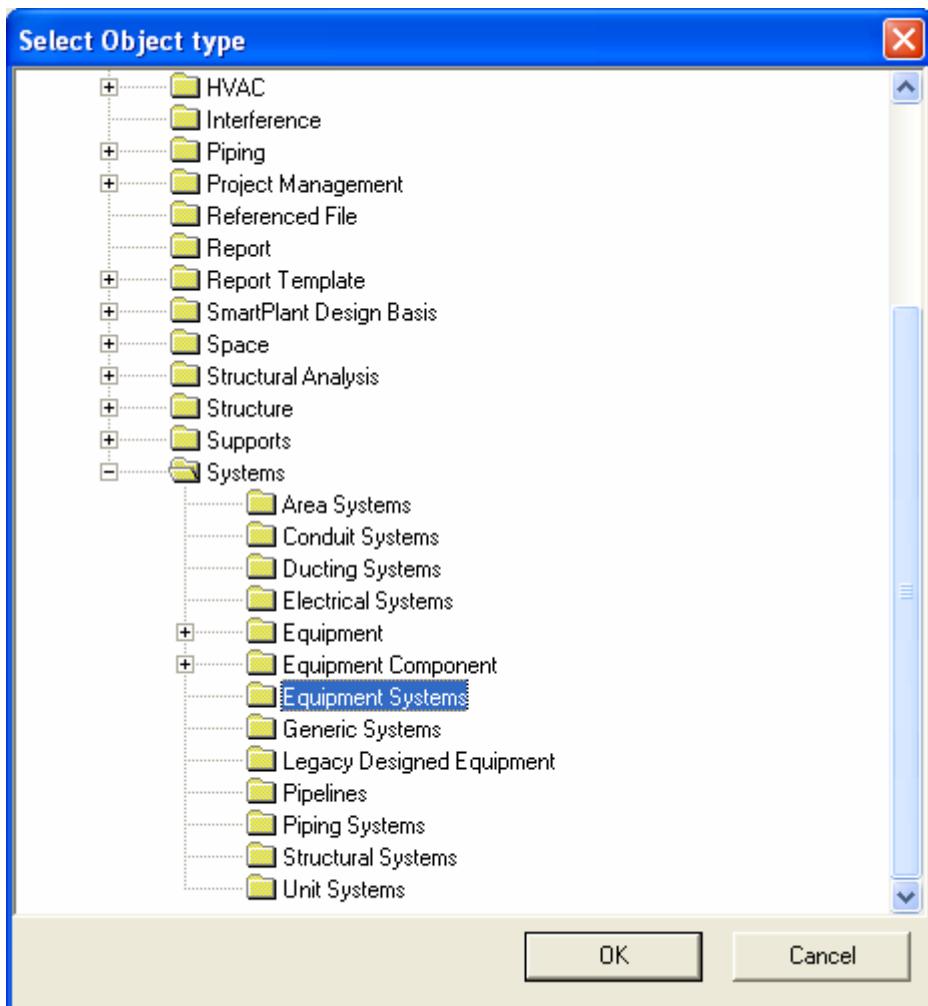


Fig. 18

Select "Name" from the standard properties:

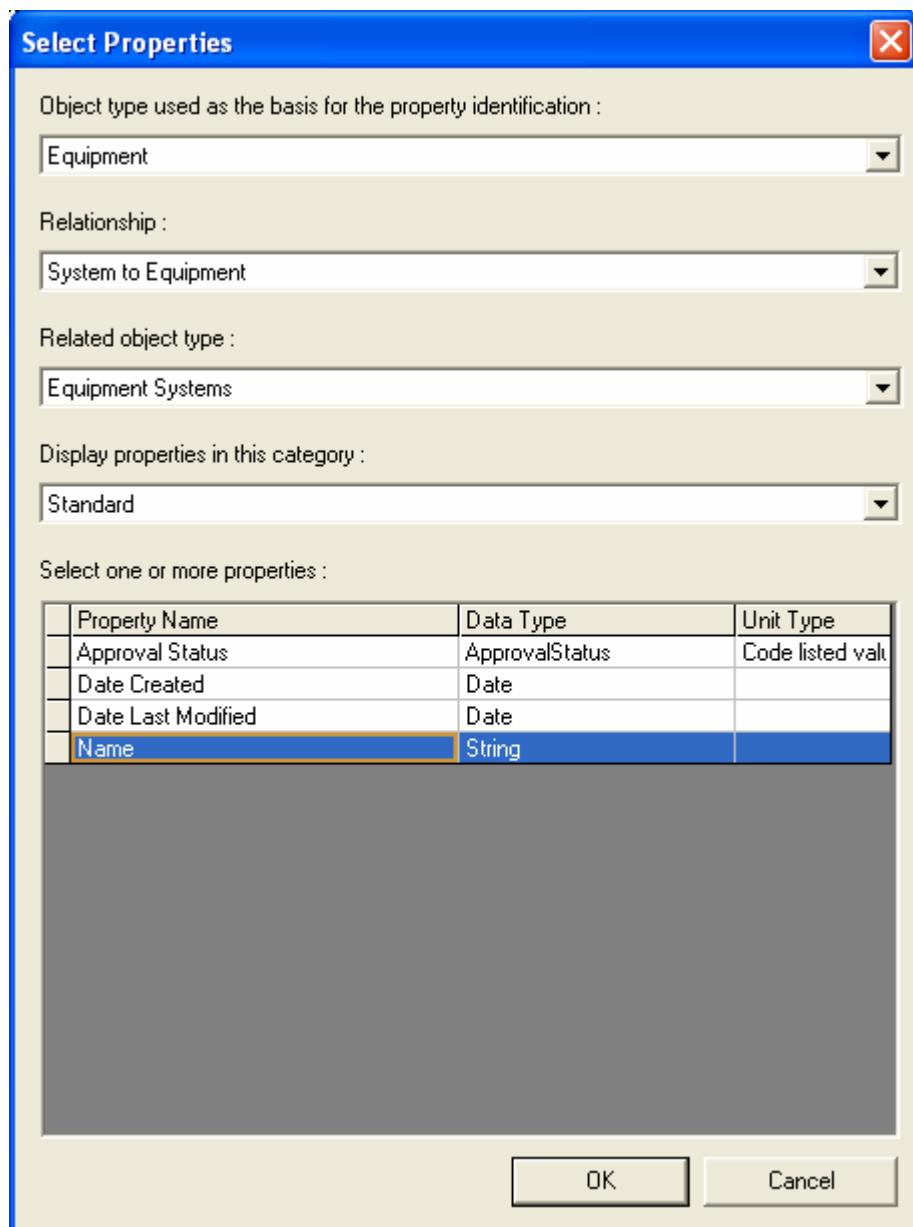


Fig. 19

Click “OK” and give it the name “Equp_Sys_Name”. Now add 2 CONSTANT properties as shown in Fig 20. We’ll use those just as “space holders” for right now, so we can still do the label formatting.

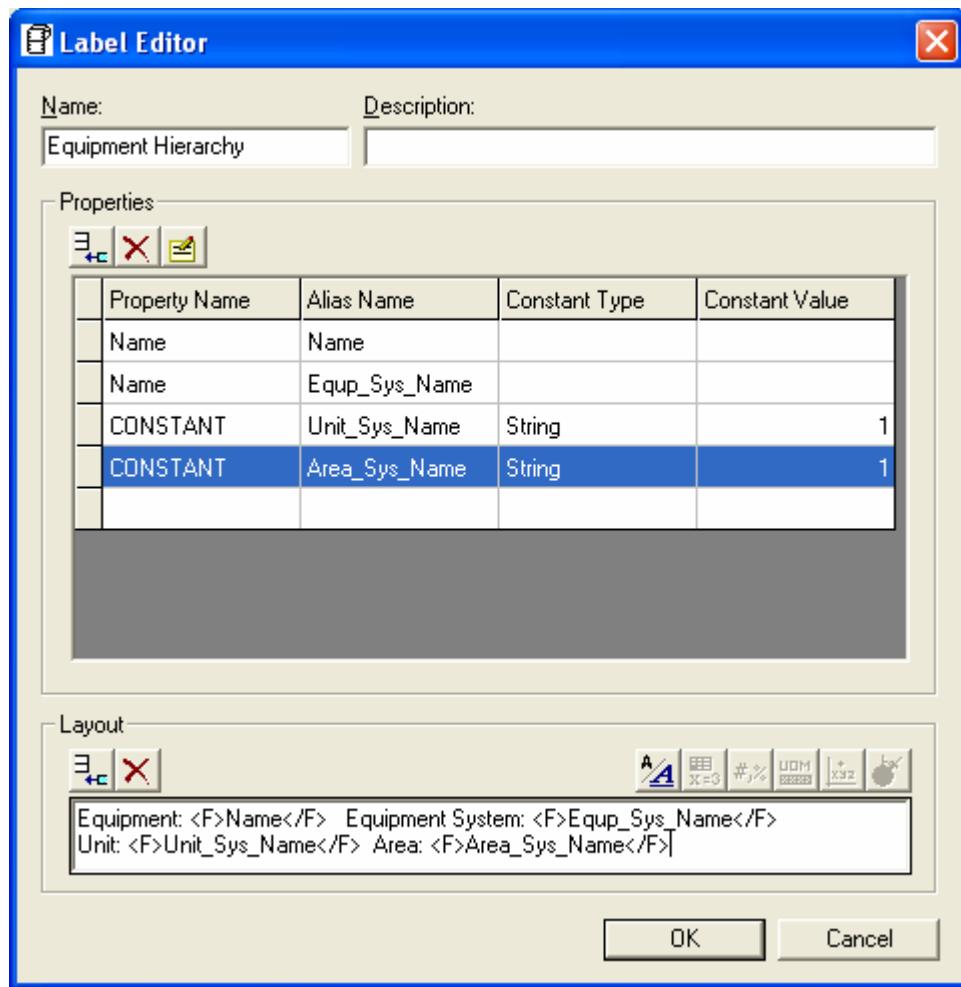


Fig. 20

Now let's open the Equipment Hierarchy.rqe and modify it so it will work for the Unit and Area systems too. The file should look like this:

```

<?xml version="1.0" encoding="UTF-8"?>
<REPORT_QUERY
Name="Equipment Hierarchy"
Description=""
RequiresFilter="No">
<DESIGN_TIME
ProgId="SP3DReportsQueryBuilder.COMQuery"
Action=""
Arg="" />

<RUN_TIME
ProgId="SP3DRuntimeQuery.CQueryInterpreter"
Action=""
Arg="" />

<RETURNED_PROPERTIES>
<RETURNED_PROPERTY
Name="Name"
SQLType="BStr">

```

```
<PATHS>
<PATH
SourceType="*"
DestinationInterface="IJNamedItem"
DestinationProperty="Name"
Concatenate="No"
PathSeparator="\\" />
</PATHS>
</RETURNED_PROPERTY>

<RETURNED_PROPERTY
Name="Equp_Sys_Name"
SQLType="BStr">
<PATHS>
<PATH
SourceType="IJEquipment"
DestinationInterface="IJNamedItem"
DestinationProperty="Name"
Concatenate="No"
PathSeparator="">
<STROKES>
<STROKE
Interface="IJEquipment"
RelationCollection="EqpParent"
Recursive="No"
Filter="First"
IsVirtualRelationship="No" />
</STROKES>
</PATH>
</PATHS>
</RETURNED_PROPERTY>

<RETURNED_PROPERTY
Name="Unit_Sys_Name"
SQLType="BStr">
<PATHS>
<PATH
SourceType="*"
DestinationInterface="CONSTANT"
DestinationProperty="1"
Concatenate="No"
PathSeparator="" />
</PATHS>
</RETURNED_PROPERTY>

<RETURNED_PROPERTY
Name="Area_Sys_Name"
SQLType="BStr">
<PATHS>
<PATH
SourceType="*"
DestinationInterface="CONSTANT"
DestinationProperty="1"
Concatenate="No"
PathSeparator="" />
</PATHS>
```

```
</RETURNED_PROPERTY>
</RETURNED_PROPERTIES>
</REPORT_QUERY>
```

Let's look at the Unit_Sys_Name property. We already know that all systems are related through the System Hierarchy relationship. This means that to get the Unit System, from the Equipment System we need to navigate the System Hierarchy relationship once from IJSystemChild interface through the SystemParent relation collection. Property Equp_Sys_Name already has the navigation from Equipment to its Equipment System, so let's copy that STROKE (in *italic*) and then just add another STROKE (**bold**) for the *System Hierarchy* traversal (see example 3 if you need a refresher on this particular relationship):

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

That should take care of the Unit System. Now, to get the Area System, we essentially need to copy the code for the Unit System and add yet another STROKE to traverse the System Hierarchy relationship one more time, since the Area is one more level above the Unit System in the hierarchy given in our example. Instead of just writing the last STROKE twice, we can again use recursion. This time we can explicitly tell the software how many times to execute this statement instead of checking for an implemented interface as we did in example 3. This is controlled by setting the following attributes as shown:

```
Recursive = "Yes"
ExitCondition="Depth"
ExitValue="2"
```

So the Area_Sys_Name becomes:

```
<RETURNED_PROPERTY
Name="Area_Sys_Name "
```

```

SQLType="BStr">
<PATHS>
<PATH
  SourceType="IJEquipment"
  DestinationInterface="IJNamedItem"
  DestinationProperty="Name"
  Concatenate="No"
  PathSeparator="">
  <STROKES>
    <STROKE
      Interface="IJEquipment"
      RelationCollection="EqpParent"
      Recursive="No"
      Filter="First"
      IsVirtualRelationship="No" />
    <STROKE
      Filter="First"
      Interface="IJSystemChild"
      ExitCondition="Depth"
      ExitValue="2"
      RelationCollection="SystemParent"
      Recursive="Yes"
      IsVirtualRelationship="No" />
  </STROKES>
</PATH>

</PATHS>
</RETURNED_PROPERTY>

```

And the whole label is then:

```

<?xml version="1.0" encoding="UTF-8"?>
<REPORT_QUERY
  Name="Equipment Hierarchy"
  Description=""
  RequiresFilter="No">
  <DESIGN_TIME
    Progid="SP3DReportsQueryBuilder.COMQuery"
    Action=""
    Arg="" />
  <RUN_TIME
    Progid="SP3DRuntimeQuery.CQueryInterpreter"
    Action=""
    Arg="" />
<RETURNED_PROPERTIES>
  <RETURNED_PROPERTY
    Name="Name"
    SQLType="BStr">
    <PATHS>
      <PATH
        SourceType="*"
        DestinationInterface="IJNamedItem"
        DestinationProperty="Name"
        Concatenate="No"

```

```
>
</PATHS>
</RETURNED_PROPERTY>
<RETURNED_PROPERTY
Name="Equp_Sys_Name"
SQLType="BStr">
<PATHS>
<PATH
SourceType="IJEquipment"
DestinationInterface="IJNamedItem"
DestinationProperty="Name"
Concatenate="No"
PathSeparator="">
<STROKES>
<STROKE
Interface="IJEquipment"
RelationCollection="EqpParent"
Recursive="No"
Filter="First"
IsVirtualRelationship="No" />
</STROKES>
</PATH>
</PATHS>
</RETURNED_PROPERTY>
<RETURNED_PROPERTY
Name="Unit_Sys_Name"
SQLType="BStr">
<PATHS>
<PATH
SourceType="IJEquipment"
DestinationInterface="IJNamedItem"
DestinationProperty="Name"
Concatenate="No"
PathSeparator="">
<STROKES>
<STROKE
Interface="IJEquipment"
RelationCollection="EqpParent"
Recursive="No"
Filter="First"
IsVirtualRelationship="No" />
<STROKE
Filter="First"
Interface="IJSystemChild"
ExitCondition=""
ExitValue=""
RelationCollection="SystemParent"
Recursive="No"
IsVirtualRelationship="No" />
</STROKES>
</PATH>
</PATHS>
</RETURNED_PROPERTY>
<RETURNED_PROPERTY
Name="Area_Sys_Name"
SQLType="BStr">
```

```
<PATHS>
<PATH
  SourceType="IJEquipment"
  DestinationInterface="IJNamedItem"
  DestinationProperty="Name"
  Concatenate="No"
  PathSeparator="">
  <STROKES>
    <STROKE
      Interface="IJEquipment"
      RelationCollection="EqpParent"
      Recursive="No"
      Filter="First"
      IsVirtualRelationship="No" />
      <STROKE
        Filter="First"
        Interface="IJSystemChild"
        ExitCondition="Depth"
        ExitValue="2"
        RelationCollection="SystemParent"
        Recursive="Yes"
        IsVirtualRelationship="No" />
    </STROKES>
  </PATH>
</PATHS>
</RETURNED_PROPERTY>
</RETURNED_PROPERTIES>
</REPORT_QUERY>
```

8. Example of using recursion with Concatenate (Optional)

Objective: Create a label for a System that returns the names of all of its System Parents (regardless of their number) concatenated with "/" (Fig. 21)

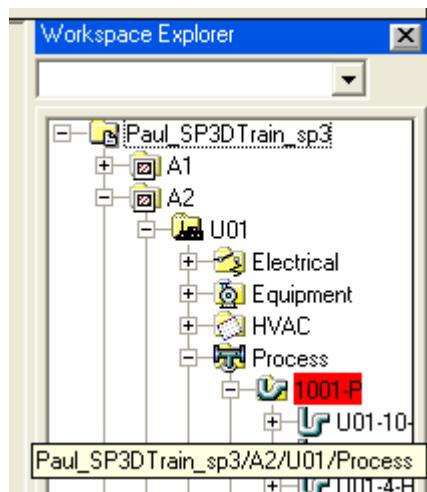


Fig. 21

Solution:

The problem with System Hierarchies is that we don't know how many nested systems we have. We can still use the recursive pattern shown in the previous examples, though, since all Systems will be nested using the same *System Hierarchy* relationship. If we do not specify an *ExitCondition*, the recursion will keep going till we reach the top parent, which is exactly what we want. We will also use the attributes *Concatenate* = "Left" and *PathSeparator* = "/" to "remember" the System Names at each recursive traversal. The *Concatenate* attribute tells the software to add the string extracted at the current step of recursion to the string already extracted from the previous recursive steps. The *PathSeparator* is simply an additional string that will be added between the 2 strings to concatenate. *Concatenate* can take the values "Left" and "Right".

So let's create as much of the label as possible using the UI. We'll try to create a label that given a System returns its parent System name. Open the Label Editor and create a new label called "Hierarchy" (Fig 22)

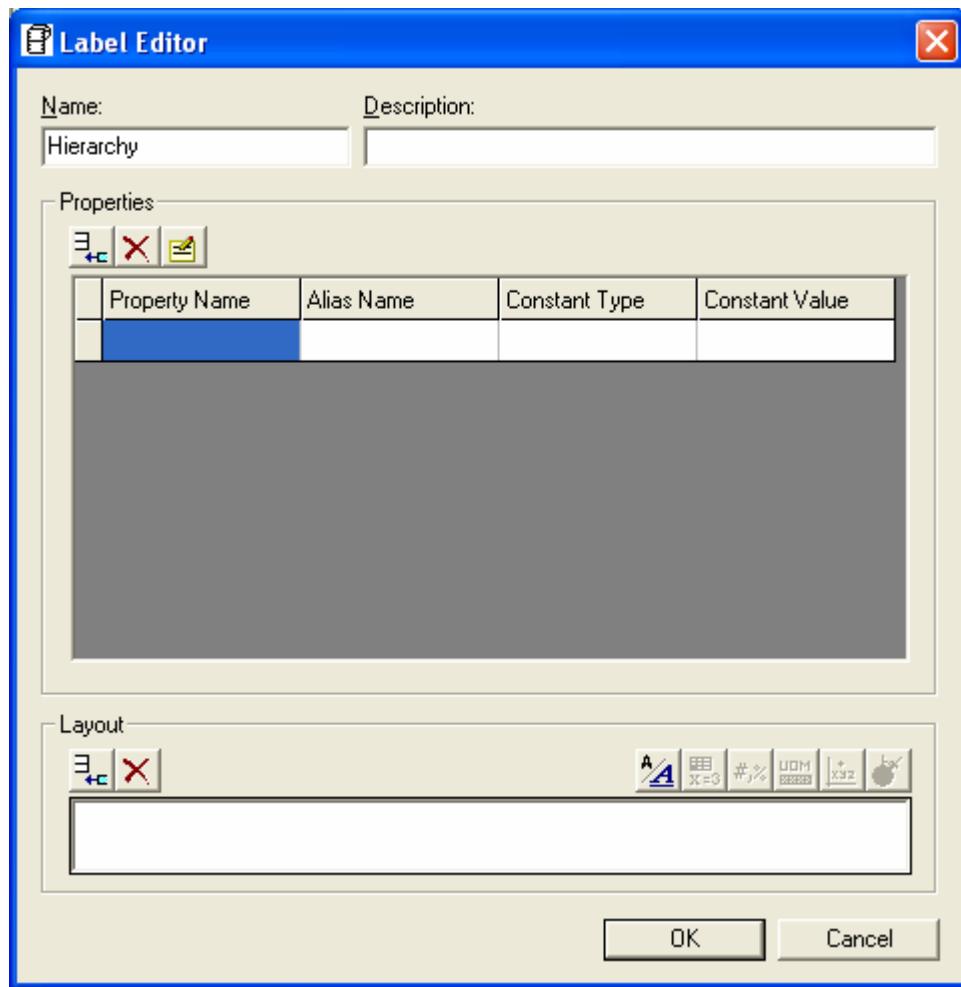


Fig 22

Select for object type “Equipment System” (or any other system for that matter).

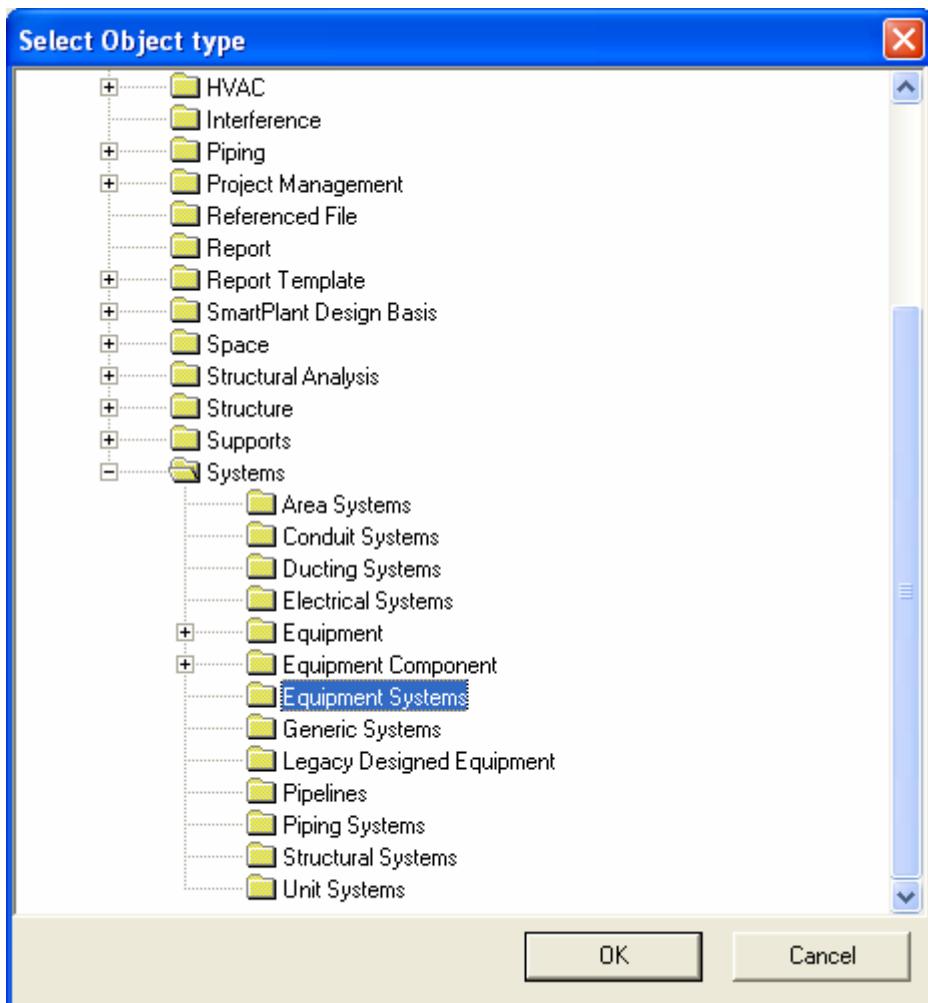


Fig .23

Select the *System Hierarchy-SystemParent* relationship:

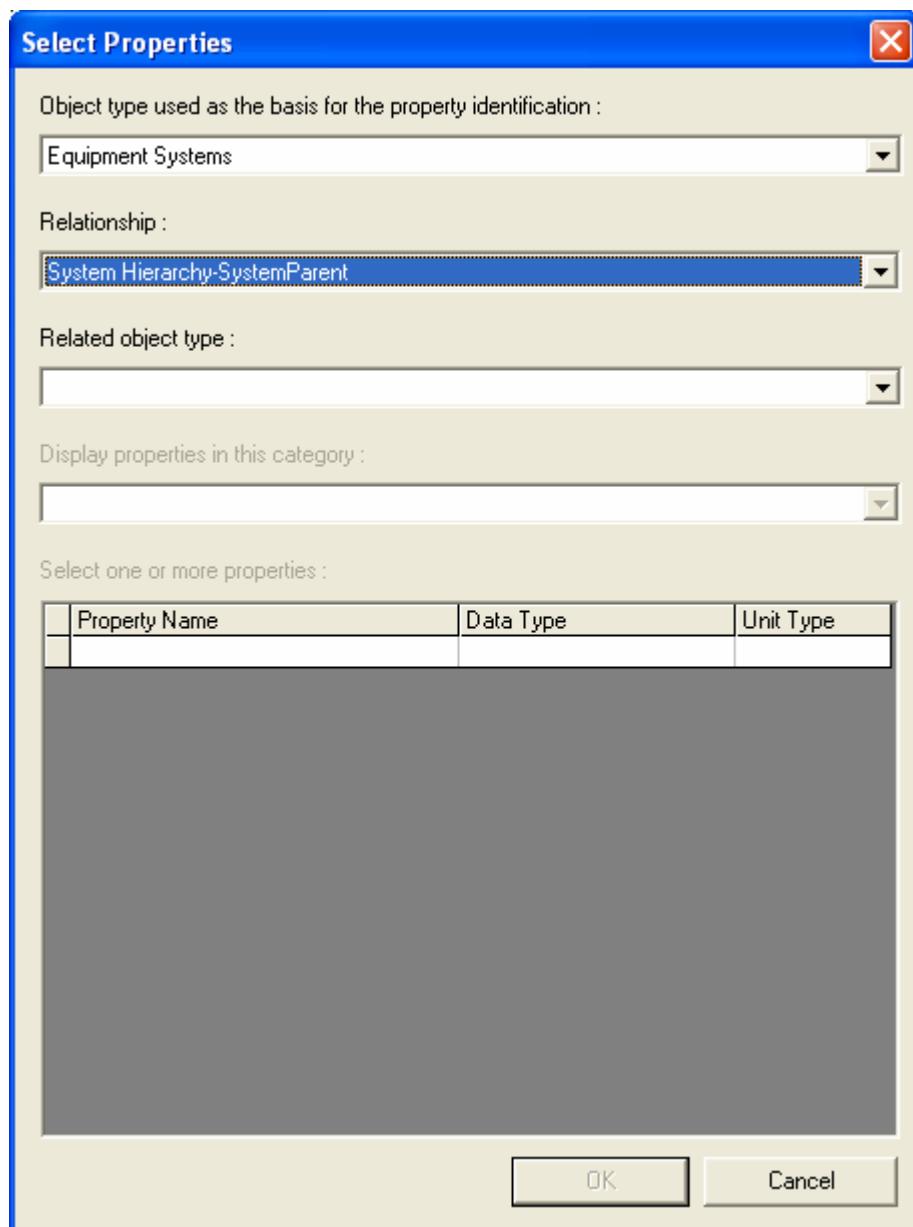


Fig 24

Select *System* for related object

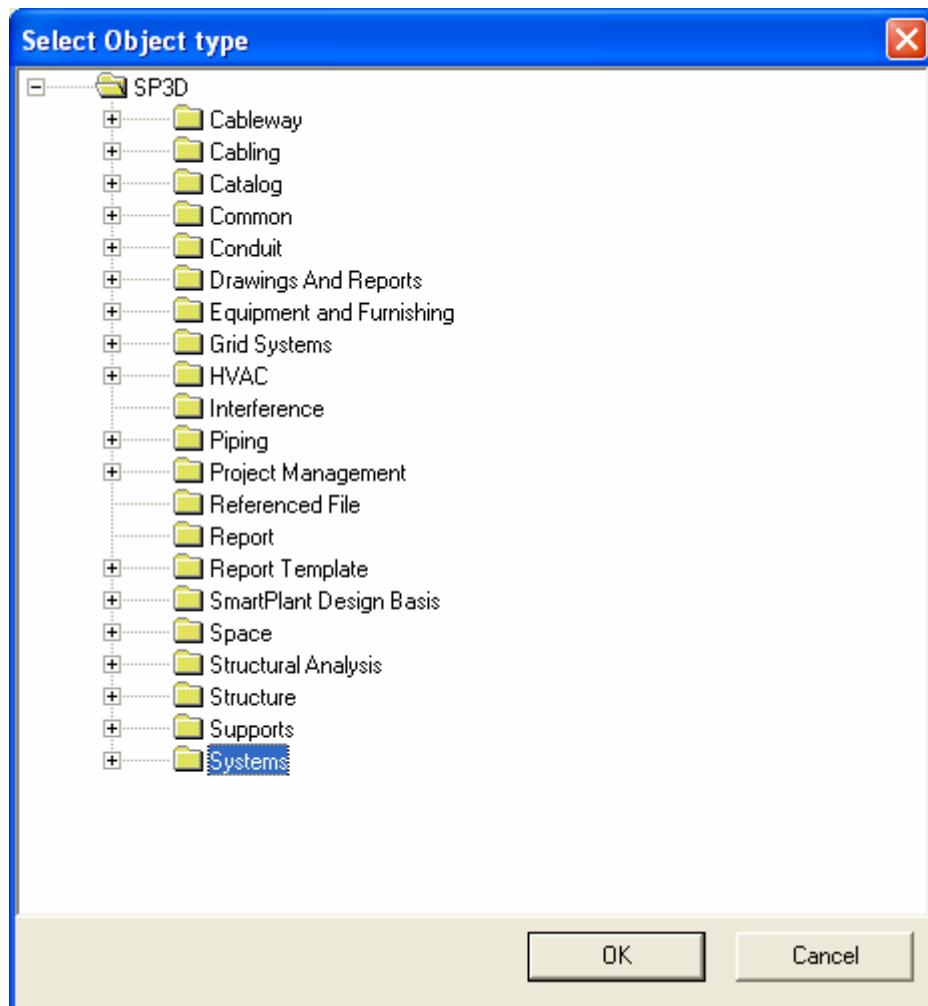


Fig 25

Select the *Name* property

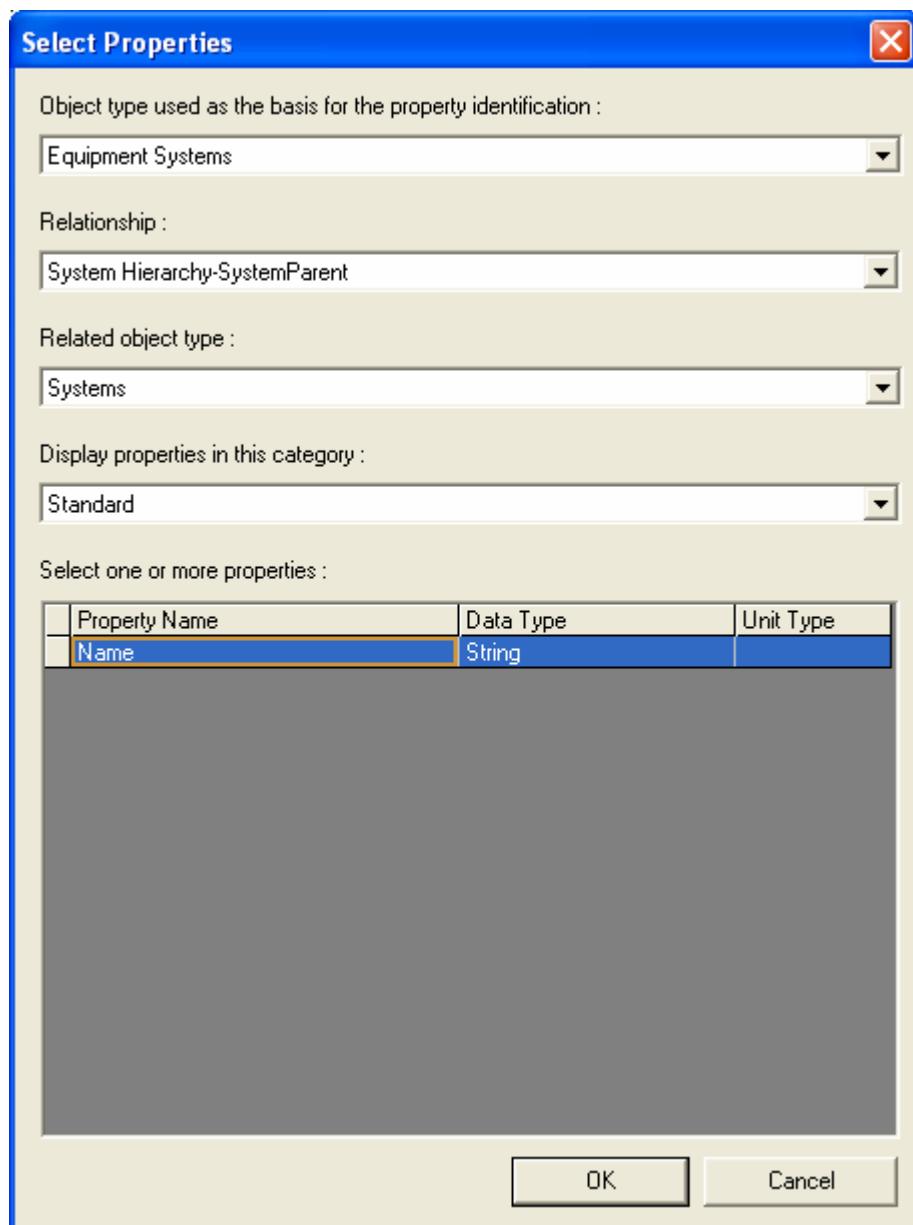


Fig 26

Finish the label:

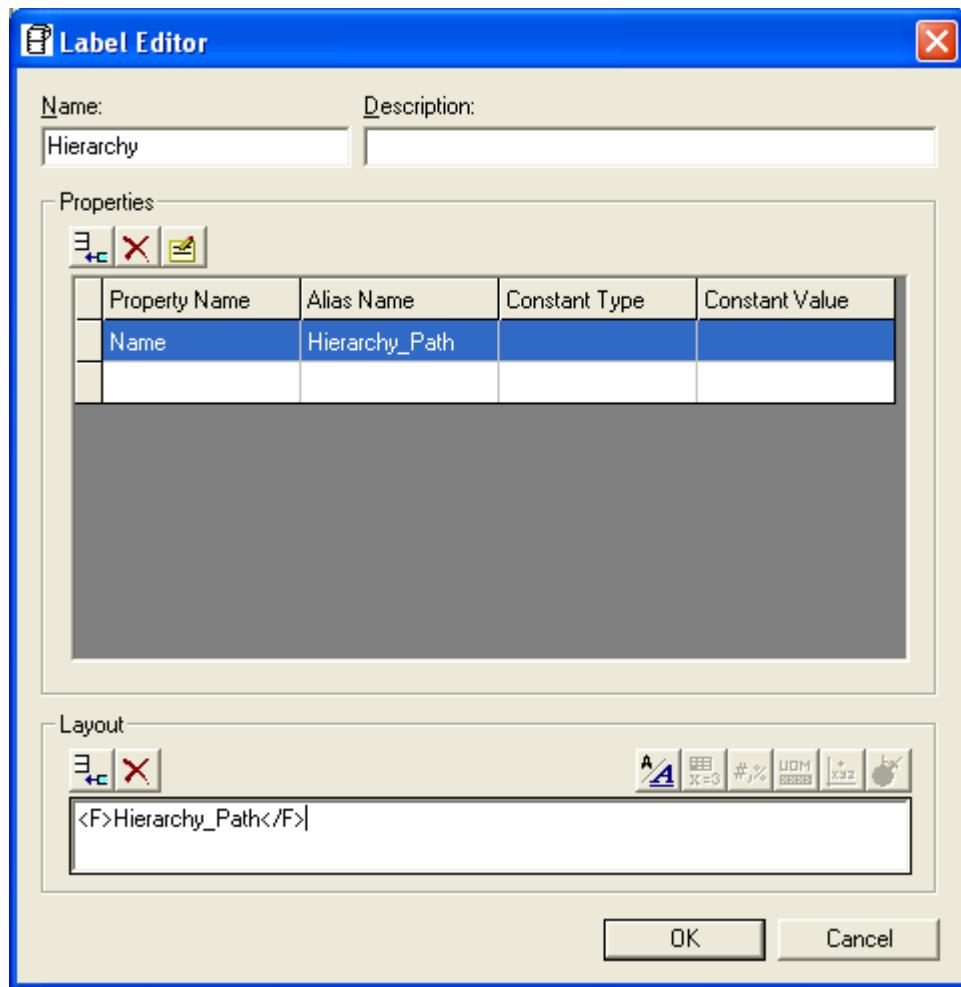


Fig 27

Now open the .Hierarchy.rqe file:

```

<?xml version="1.0" encoding="UTF-8"?>
<REPORT_QUERY
Name="Hierarchy"
Description=""
RequiresFilter="No">
<DESIGN_TIME
ProgId="SP3DReportsQueryBuilder.COMQuery"
Action=""
Arg="" />

<RUN_TIME
ProgId="SP3DRuntimeQuery.CQueryInterpreter"
Action=""
Arg="" />

<RETURNED_PROPERTIES>
<RETURNED_PROPERTY
Name="Hierarchy_Path"
SQLType="BStr">

```

```

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

```

As you can see, we have a single traversal of the System Hierarchy relationship from child to parent. Since all the upper hierarchy system parents will be connected using the same relationship, we can just make it recursive. Also, we don't need an ExitCondition, since we want the recursion to continue till there are no more parents. Lastly, use the Concatenate = "Left" and PathSeparator = "/" to indicate that the parent found at each step is to be added to the left of the hierarchy string extracted so far. The label will look like:

```

<?xml version="1.0" encoding="UTF-8"?>
<REPORT_QUERY
  Name="Hierarchy"
  Description=""
  RequiresFilter="No">
  <DESIGN_TIME
    Progid="SP3DReportsQueryBuilder.COMQuery"
    Action=""
    Arg="" />

  <RUN_TIME
    Progid="SP3DRuntimeQuery.CQueryInterpreter"
    Action=""
    Arg="" />

<RETURNED_PROPERTIES>
  <RETURNED_PROPERTY
    Name="Hierarchy_Path"
    SQLType="BStr">
    <PATHS>
      <PATH
        SourceType="IJSystemChild"
        DestinationInterface="IJNamedItem"
        DestinationProperty="Name"
        Concatenate="Left"
        PathSeparator="/">
        <STROKES>
          <STROKE

```

```

Interface="IJSystemChild"
RelationCollection="SystemParent"
Recursive="Yes"
Filter="First"
IsVirtualRelationship="No" />
</STROKES>
</PATH>
</PATHS>
</RETURNED_PROPERTY>
</RETURNED_PROPERTIES>
</REPORT_QUERY>

```

Save it and test it.

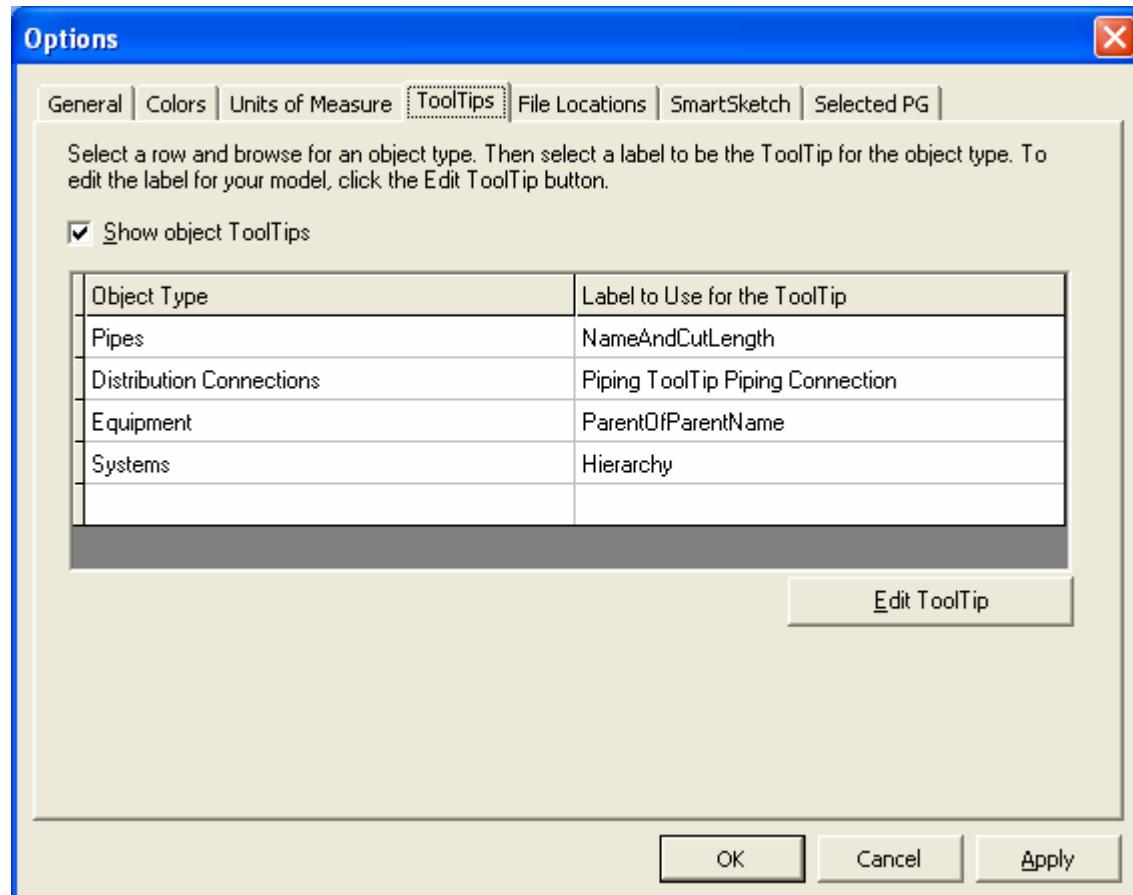


Fig 28

Advanced Labels Formatting

1. Using a PHYSICAL token with UOM to format output in proper units

Objective: Create a label that will return the Dry Weight for Equipment. Format the Dry Weight as follows:

Primary UOM - kg

Secondary UOM - g

PrecisionType - Decimal

DecimalPrecision – 2 places after the decimal point

LeadingZero - None

TrailingZeros - None

Solution:

Open the Label Editor and create a new COM label *Equipment Dry Weight* as shown:
Select Equipment for Object Type (Fig 1)

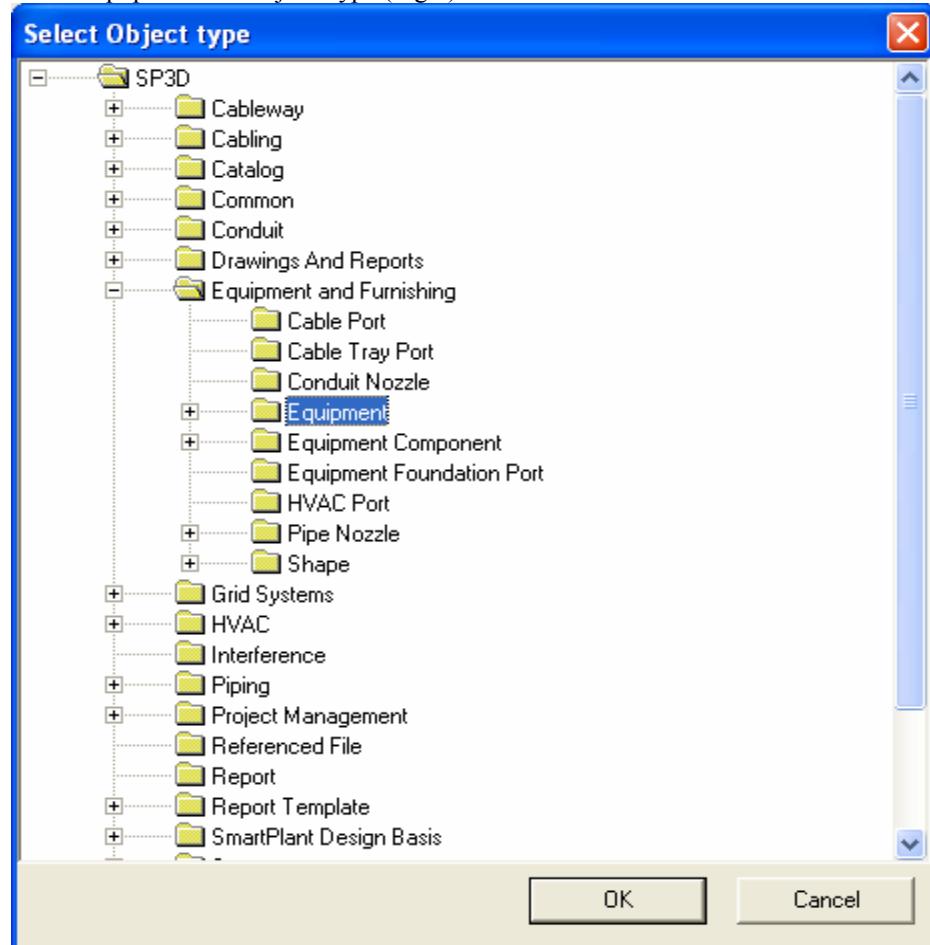


Fig. 1

Select the *Dry Weight* property from the *Weight and CG* category of the Equipment direct properties as shown on Fig 2.

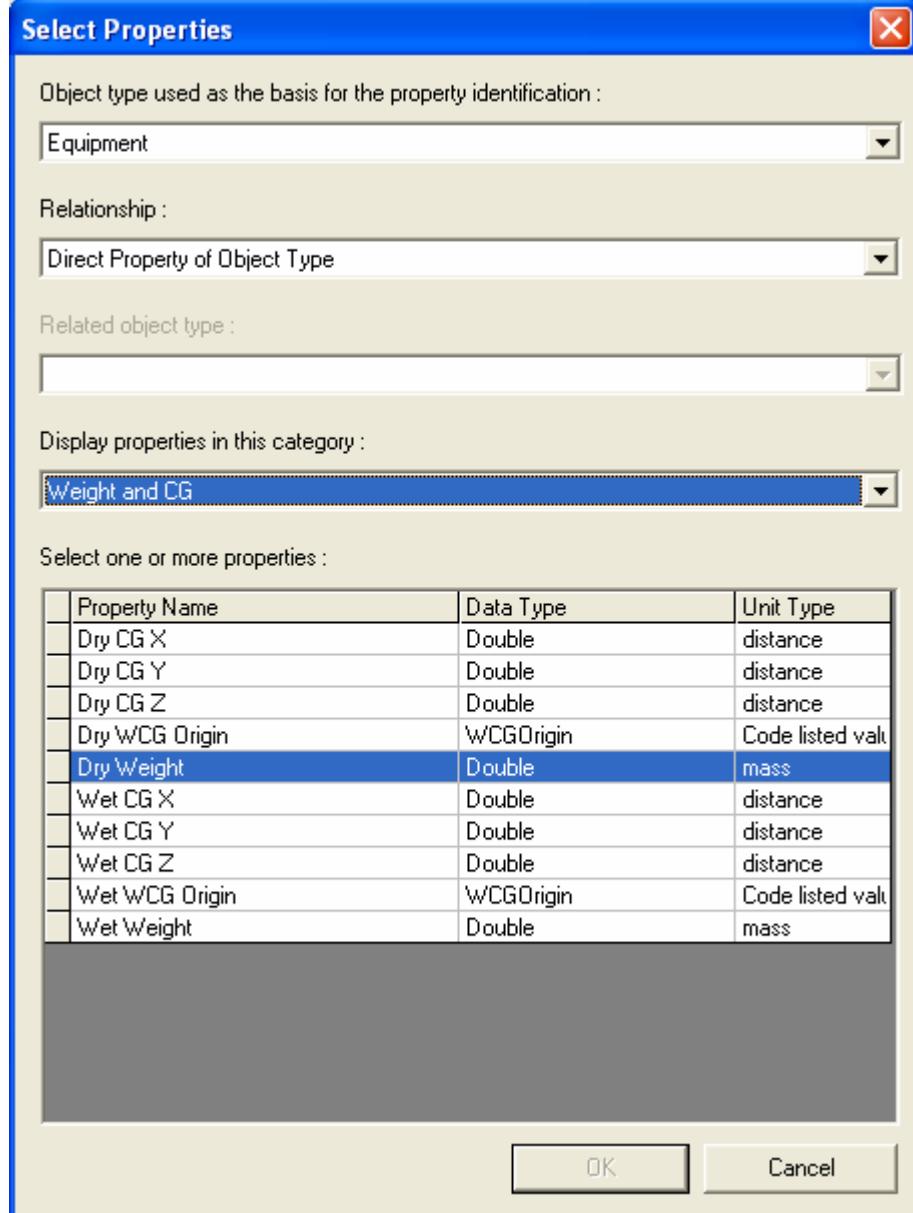


Fig. 2

Click “OK”. This creates a query that will extract this property. Now we need to add it to the label layout so that it can be shown, and we also need to format it.

Click in the Layout pane and type “Dry Weight:” so that we know what the label is returning.

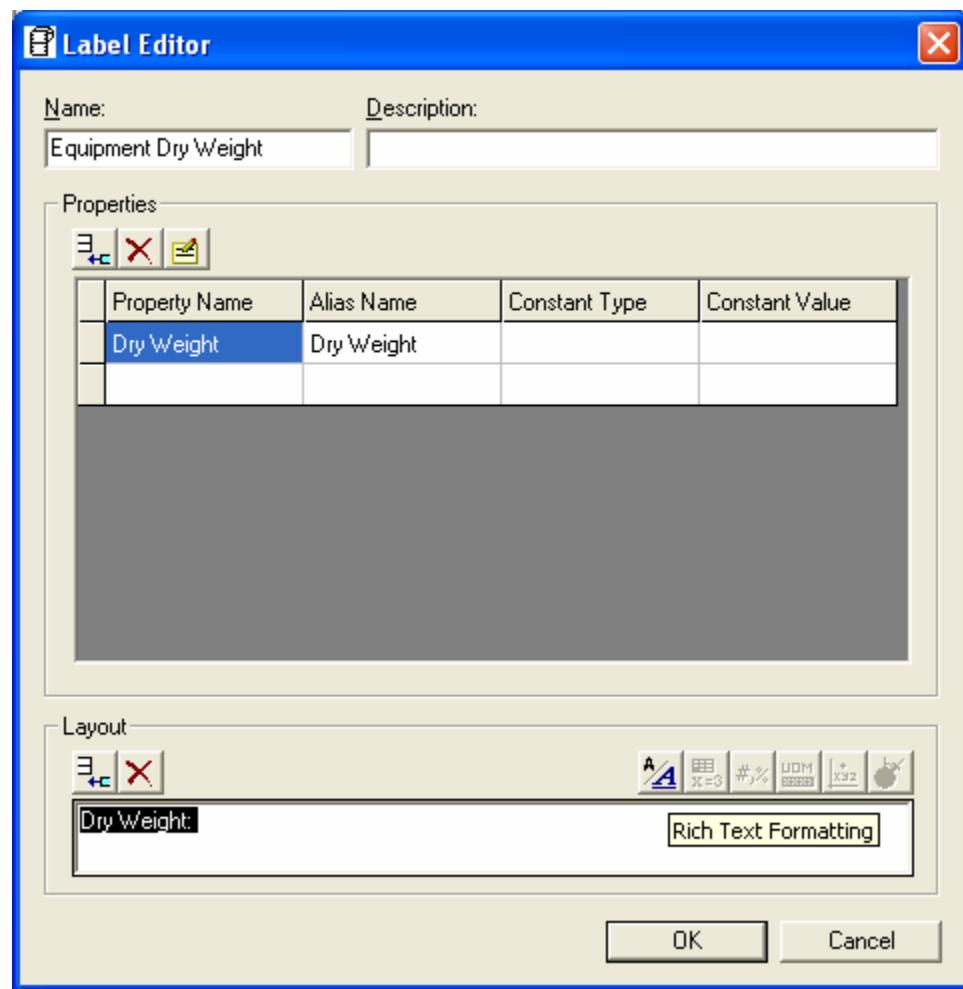


Fig 3.

Highlight the text and click on the “Rich Text Formatting” button.

In the “Font” dialog select “Bold”

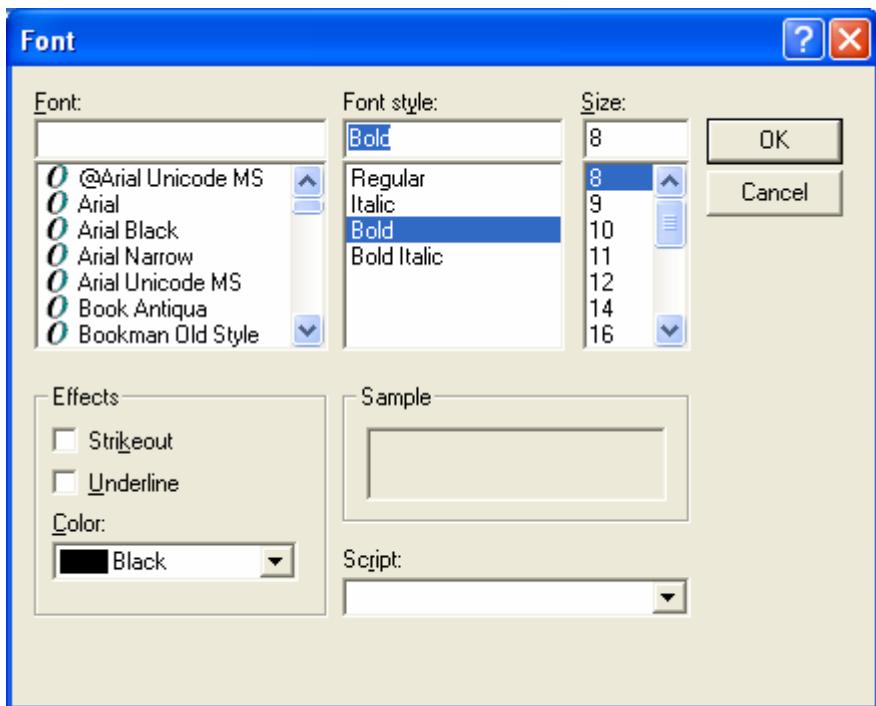


Fig. 4

Click “OK”

Select the Dry Weight property row and click “Insert a field to layout” button in the “Layout” section at the bottom of the dialog box. (Fig. 5)

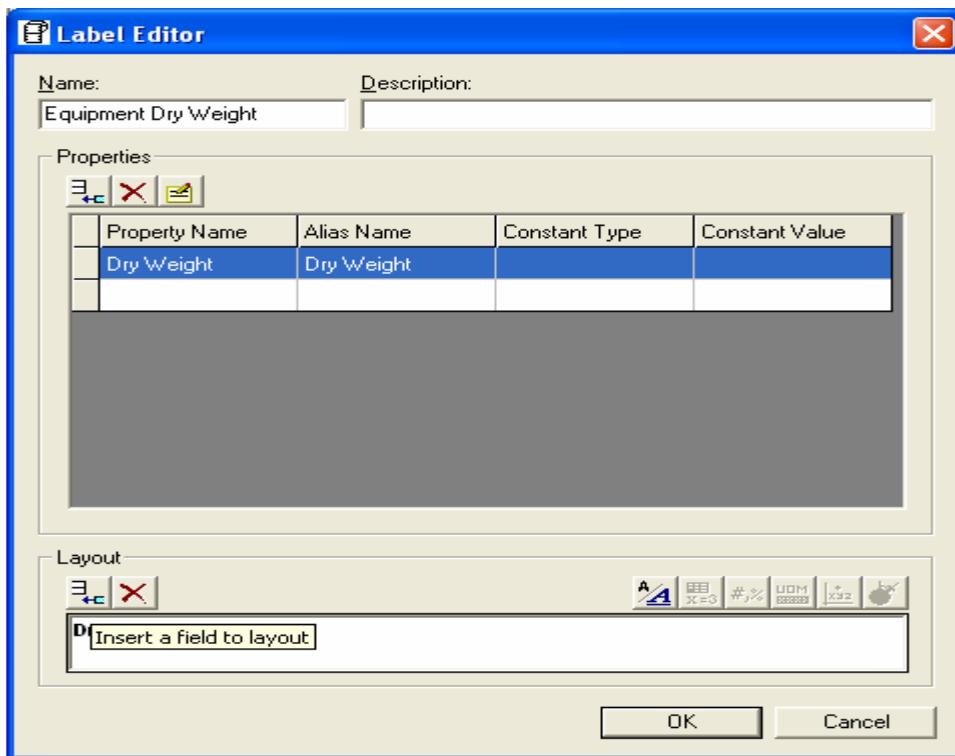


Fig. 5

The text <F>Dry Weight</F> will appear in the Layout pane next to **Dry Weight:**. Select it, and press the “Unit of Measure” button as shown in Fig. 6

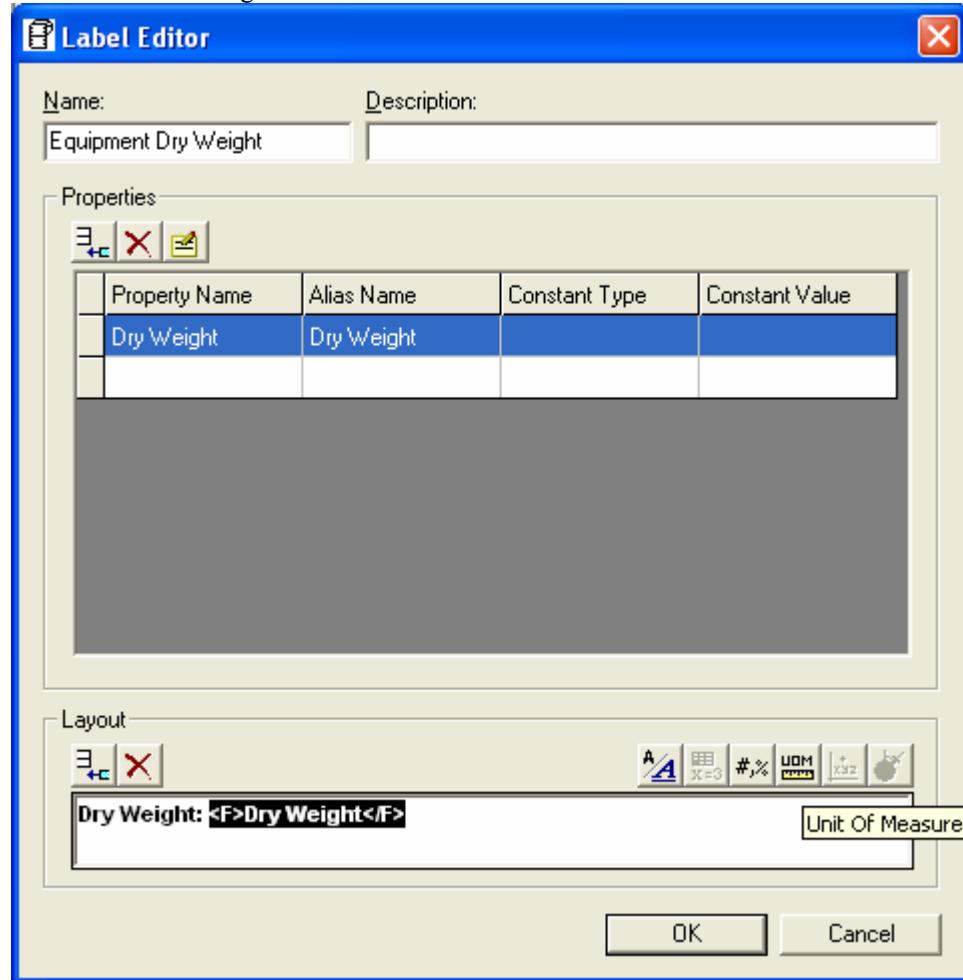


Fig. 6

The “Select Rule” dialog appears.

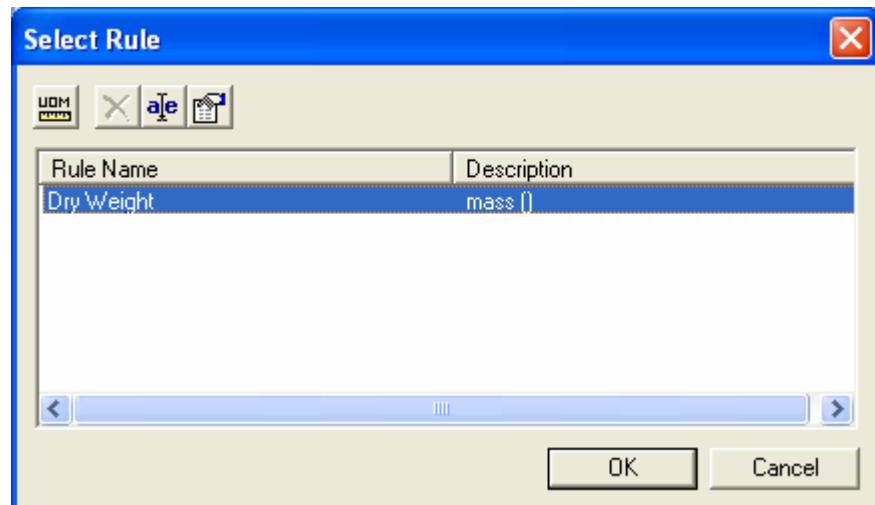


Fig. 7

Click the “Properties” button to open the selected rule (Fig. 8):

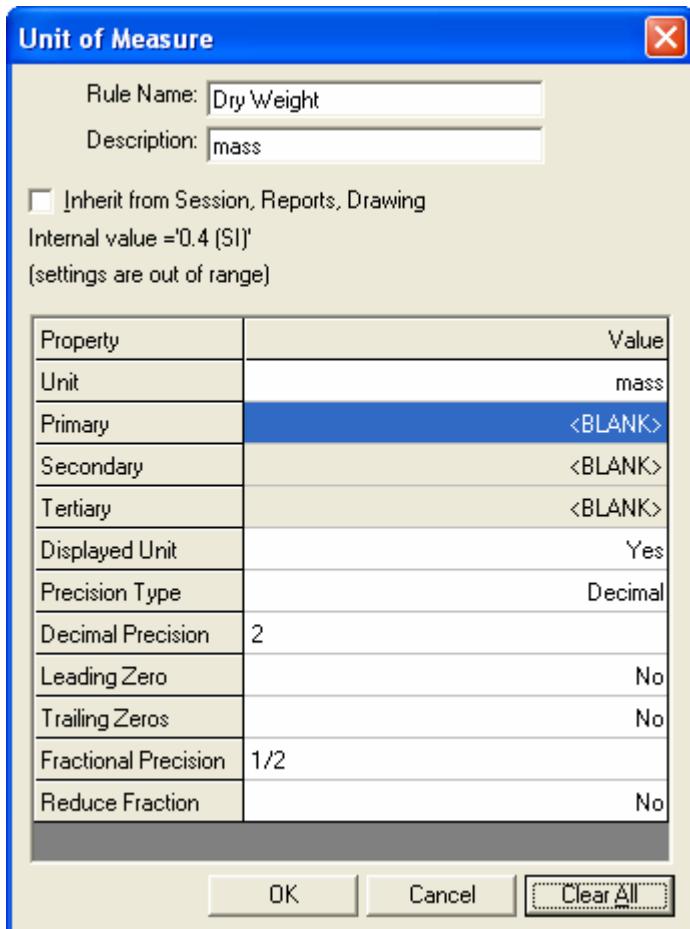


Fig. 8

The “Unit of Measure” dialog controls the units used and their formatting. To get the formatting we want for this example, change the values as follows:

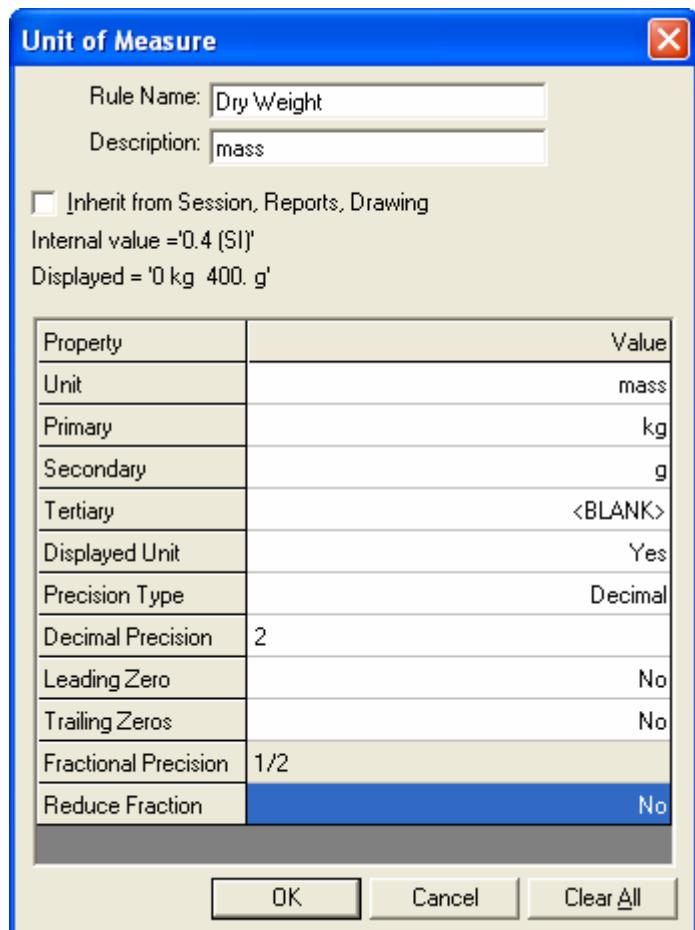


Fig. 9

Click “OK” on the “Unit of Measure” dialog. Click “OK” on the “Select Rule” dialog. Click “OK” on the “Label Editor” dialog.

You can now test the label as a tool tip:

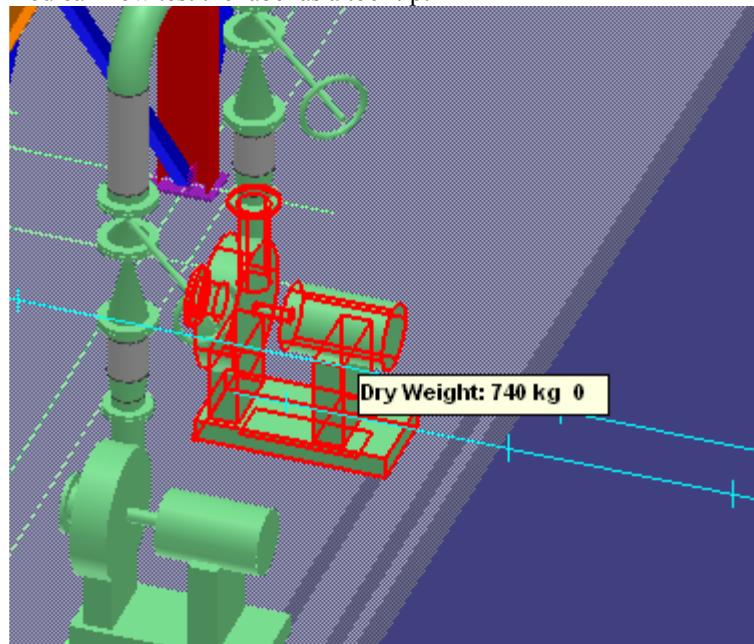


Fig. 10

Now let's open the Equipment Dry Weight.frm file and examine it.

```

<?xml version="1.0" encoding="UTF-8"?>
<REPORT_FORMATTING
Name="Equipment Dry Weight"
Description="">
<DESIGN_TIME
ProgId="SP3DLabelFormatDesigner.RTFLabel"
Action=""
Arg="" />

<RUN_TIME
ProgId="SP3DLabelsFormat.FormatLabel"
Action="RTFLabel"
Arg="" />

<FORMATTING_PARAMETERS
Name="Equipment Dry Weight"
Site="User"
Path="Equipment Dry Weight.rfp" />

<LAYOUT_TEMPLATE
Type="Internal" />

<RTF_LABEL>
<POINTS />

<VECTORS />

<BLOCKS>
<BLOCK
Action="Visible">
<TOKENS>
<TEXT
Value="\rtf1\ansi\ansicpg1252\deff0{\fonttbl{\f0\fswiss\fcharset0 Arial;}\{ \f1\fnil\fcharset0 MS Shell Dlg;}} {\colortbl
\red0\green0\blue0; } \viewkind4\uc1\pard\cfl\lang1033\b\f0\fs16 Dry Weight: "
ToParse="no"
Visible="yes" />

<PHYSICAL
Column="Dry Weight"
UOM="Dry Weight"
Visible="yes" />

<TEXT
Value="\cf0\b0\f1\fs17 \par } "
ToParse="no"
Visible="yes" />
</TOKENS>
</BLOCK>
</BLOCKS>
</RTF_LABEL>
</REPORT_FORMATTING>
```

The formatting here happens in the <BLOCKS> section. This section is made of one BLOCK (more on that later), and each BLOCK consists of TOKENS. **Appendix A** gives a detailed enumeration of all possible tokens we can have. Here we have just 2 types:

TEXT – user defined (constant) text formatted as Rich Text. The *Value* attribute shows the actual text string together with the Rich Text formatting. In our case, the user text is “Dry Weight” (shown in **bold** here to separate it visually from the Rich Text). The Rich Text formatting itself is rather cryptic, but all you need to know is that it applies to all tokens up until the last closing “}” (in our case: `Value="\cf0\b0\f1\fs17 \par } "`). That means that it will be applied to the data returned from the PHYSICAL token too:

```
<TOKENS>
  <TEXT
    Value=" {\rtf1\ansi\ansicpg1252\deff0{\fonttbl{\f0\fswiss\fcharset0 Arial;}{\f1\fnil\fcharset0 MS Shell Dlg;}} {\colortbl
    \red0\green0\blue0;} \viewkind4\uc1\pard\cf1\lang1033\b\f0\fs16 Dry Weight:"
      ToParse="no"
      Visible="yes" />

  <PHYSICAL
    Column="Dry Weight"
    UOM="Dry Weight"
    Visible="yes" />

  <TEXT
    Value="\cf0\b0\f1\fs17 \par } "
    ToParse="no"
    Visible="yes" />
</TOKENS>
```

PHYSICAL – describes dimensioning data that will come from the database and will require transformation to some units of measure. In our case this is the Dry Weight. The *Column* attribute describes the name of the property (<RETURNED_PROPERTY Name="Dry Weight"> from the .rqe file, or the alias from the SQL statement if we have an SQL label). The attribute *UOM*=“Dry Weight” tells the software how to format the Dry Weight value. The specifics of the UOM formatting are found in the Equipment Dry Weight.rfp file:

```
<?xml version="1.0" encoding="UTF-8"?>
<FORMATTING_PARAMETERS
Name="Equipment Dry Weight"
Description="">
  <DESIGN_TIME
    Progid=""
    Action=""
    Arg="" />

  <RUN_TIME
    Progid="SP3DPrompts.ctlTab"
    Action=""
    Arg="" />

  <FIELDS />

  <UOMS>
    <UOM
      Name="Dry Weight"
      CanInherit="No"
      Type="mass"
      Primary="kg"
      Secondary="g"
```

```
Tertiary=""  
UnitsDisplayed="Yes"  
PrecisionType="Decimal"  
DecimalPrecision="2"  
FractionalPrecision="2"  
LeadingZero="No"  
TrailingZeros="No"  
ReduceFraction="No" />  
</UOMS>  
</FORMATTING_PARAMETERS>
```

As you can see, the <UOMS> just repeat the values we set in the Unit of Measure dialog and are pretty straightforward. The one important attribute to remember here is *CanInherit* – it determines if the UOM's can be overwritten by the UOM of the calling application. This means that if you set this attribute to “Yes”, even if you select Metric UOM, you’ll still see your values expressed in the English system if that’s what you have selected in your session file.

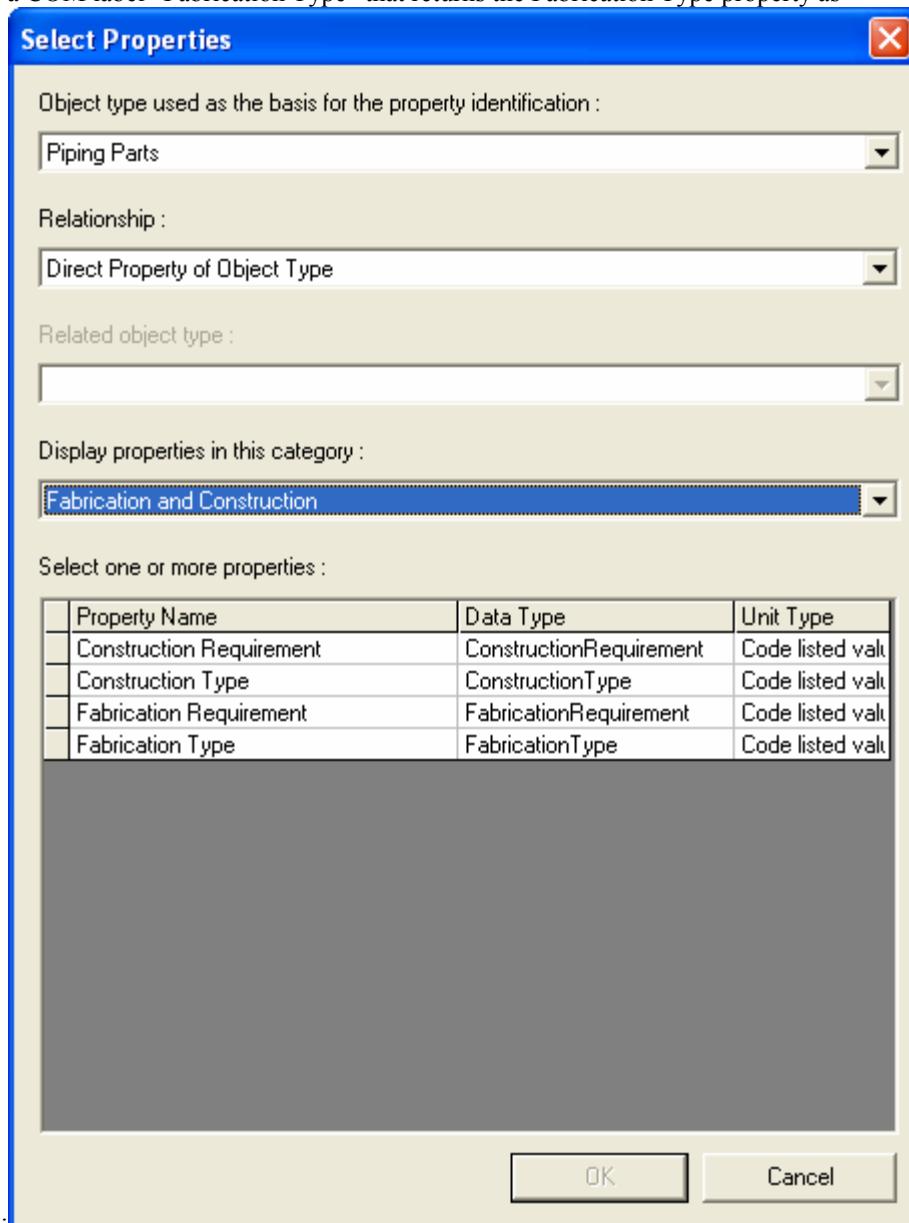
It is not necessary to remember all the attributes of the <UOM> tag, but it’s good to have a general idea of what they are and where they located. Probably the only times you’ll ever need to create one of those “by hand” will be if you are creating an SQL label, but even then you can just copy them from another label.

2. Outputting codelisted values

Objective: Create a label for Piping Parts that returns the Code List value for Fabrication Type

Solution:

Create a COM label “Fabrication Type” that returns the Fabrication Type property as



shown:

Fig. 16

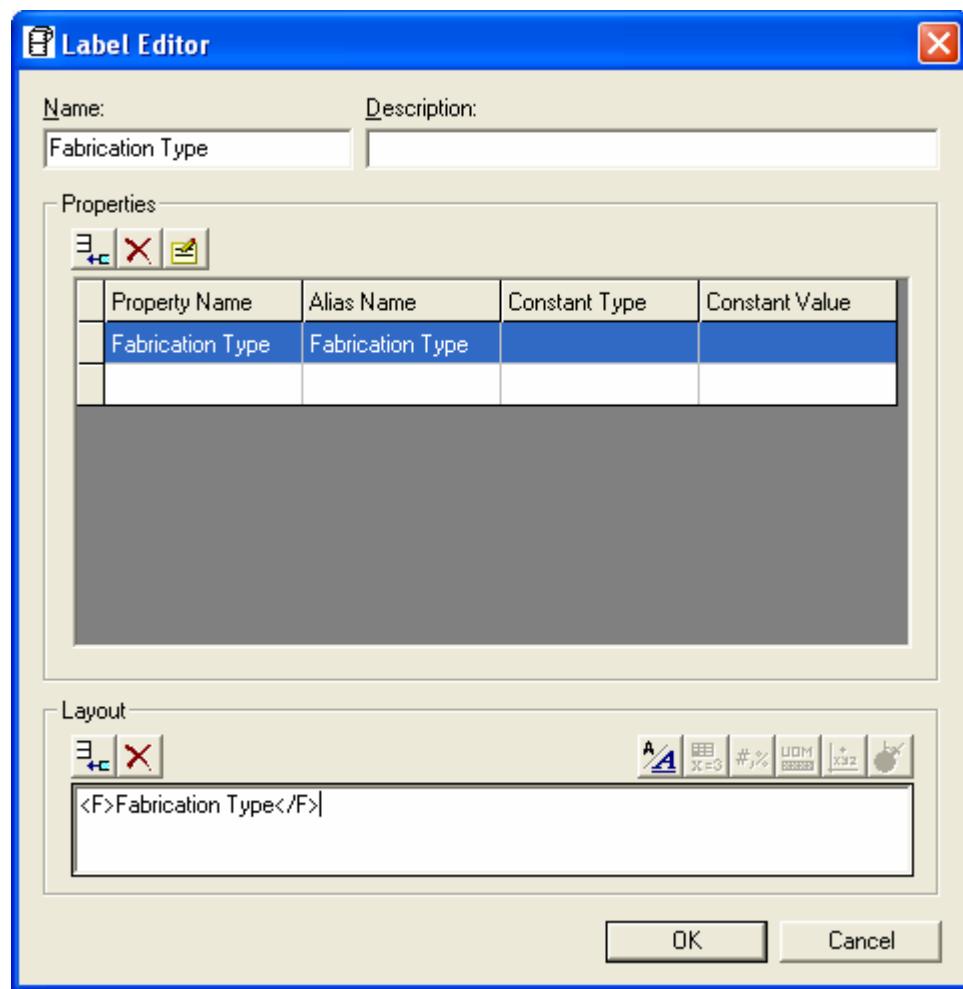


Fig. 17

The so created label will return the Short Description, by default:

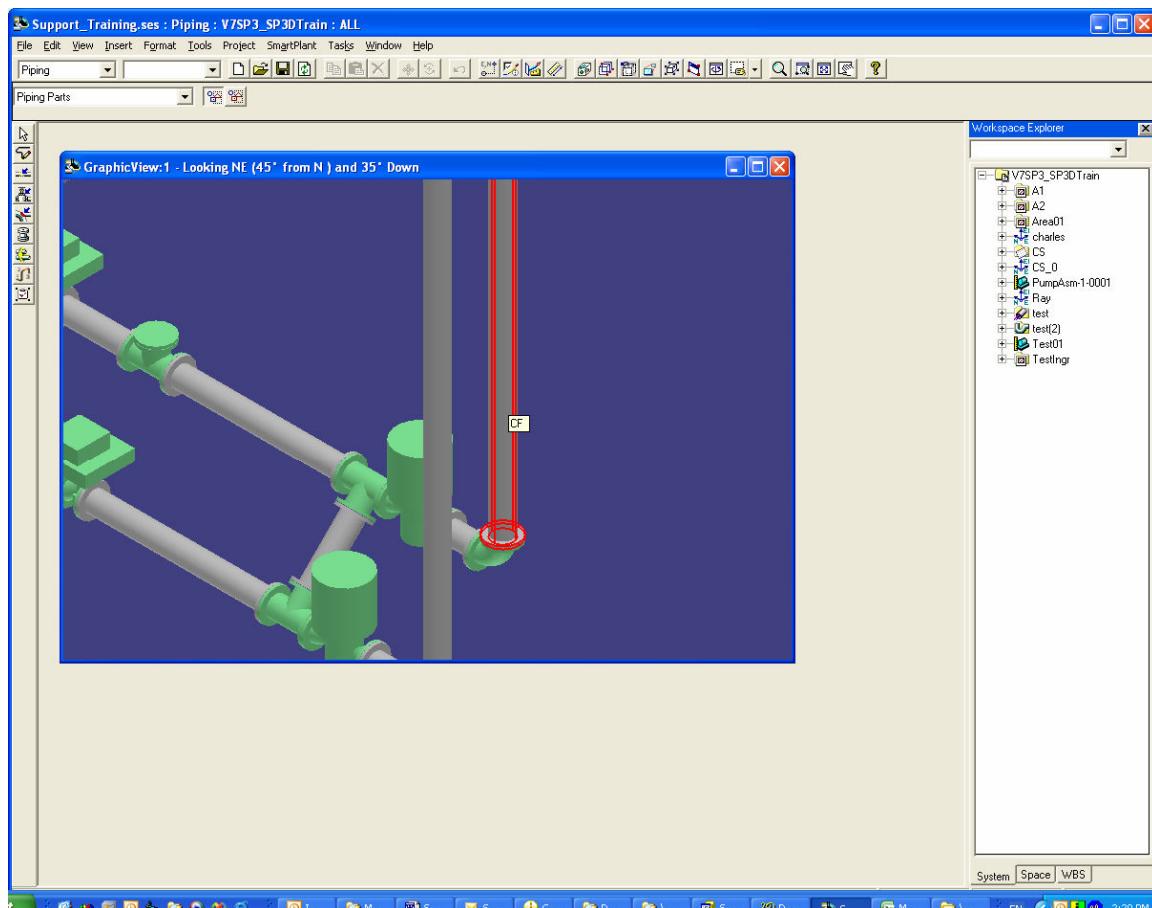


Fig. 18

Open the .rfm file.

```

<?xml version="1.0" encoding="UTF-8"?>
<REPORT_FORMATTING
Name="Fabrication Type"
Description="">
<DESIGN_TIME
ProgId="SP3DLabelFormatDesigner.RTFLLabel"
Action=""
Arg="" />

<RUN_TIME
ProgId="SP3DLabelsFormat.FormatLabel"
Action="RTFLLabel"
Arg="" />

<FORMATTING_PARAMETERS
Name="Fabrication Type"
Site="User"
Path="Fabrication Type.rfp" />

<LAYOUT_TEMPLATE
Type="Internal" />
```

```

<RTF_LABEL>
  <POINTS />

  <VECTORS />

  <BLOCKS>
    <BLOCK
      Action="Visible">
      <TOKENS>
        <TEXT
          Value=" {\rtf1\ansi\deff0 {\fonttbl{\f0\fni\fcharset0 MS Shell Dlg;}} \viewkind4\uc1\pard\lang1033\f0\fs17 "
          ToParse="no"
          Visible="yes" />

        <DATA
          Column="Fabrication Type"
          ToParse="yes"
          Visible="yes" />

        <TEXT
          Value=" \par } "
          ToParse="no"
          Visible="yes" />
      </TOKENS>
    </BLOCK>
  </BLOCKS>
</RTF_LABEL>
</REPORT_FORMATTING>

```

To make the label return the numeric Code List value, change the highlighted text to:

Column="Fabrication Type_index

If we want the label to return the long string value instead, change the line to

Column="Fabrication Type_LongString

Now suppose that none of those work for our case. Let's assume that we want to output in the case of value = 7 (CF, "Contractor Field Fabricated") simply the word "CONTRACTOR", and "NON-CONTRACTOR" in all other cases (by the same logic we could output a different string for every possible value, but we'll keep this example simple). This is, in fact, like creating our own long string values for a given code list. This can be achieved through blocks and conditions that control those blocks visibility (see 5. Conditional Formatting). The .rfm of the label is given below:

```

<?xml version="1.0" encoding="UTF-8"?>
<REPORT_FORMATTING Name="Fabrication Type" Description="">
  <DESIGN_TIME Progid="SP3DLabelFormatDesigner.RTFLLabel" Action="" Arg="" />

  <RUN_TIME Progid="SP3DLLabelsFormat.FormatLabel" Action="RTFLLabel" Arg="" />

  <FORMATTING_PARAMETERS Name="Fabrication Type" Site="User" Path="Fabrication Type.rfp" />

  <LAYOUT_TEMPLATE Type="Internal" />

<RTF_LABEL>
  <POINTS />

```

```

<VECTORS />

<BLOCKS>

    <BLOCK Action="Visible">
        <TOKENS>
            <TEXT Value=" {\rtf1\ansi\deff0 {\fonttbl{\f0\fnil\fcharset0 MS Shell Dlg;}} \viewkind4\uc1\pard\lang1033\f0\fs17 "
ToParse="no" Visible="yes" />
        </TOKENS>
    </BLOCK>

    <BLOCK Action="Visible">
        <RULE RuleOperator="All" RuleOperand="True" />
        <TOKENS>

            <DATA Column="Fabrication Type" ToParse="yes" Visible="no" >
                <CONDITION ConditionOperator="Equal" ConditionOperand1="CF" StateOfValue="Raw" />
            </DATA>

            <TEXT Value="CONTRACTOR" ToParse="no" Visible="yes" />

        </TOKENS>
    </BLOCK>

    <BLOCK Action="Visible">
        <RULE RuleOperator="All" RuleOperand="FALSE" />
        <TOKENS>

            <DATA Column="Fabrication Type" ToParse="yes" Visible="no" >
                <CONDITION ConditionOperator="Equal" ConditionOperand1="CF" StateOfValue="Raw" />
            </DATA>

            <TEXT Value="NON-CONTRACTOR" ToParse="no" Visible="yes" />

        </TOKENS>
    </BLOCK>

<BLOCK Action="Visible">
    <TOKENS>
        <TEXT Value=" \par } " ToParse="no" Visible="yes" />
    </TOKENS>
</BLOCK>

</BLOCKS>
</RTF_LABEL>
</REPORT_FORMATTING>

```

Note also how we are using the 1st and 4th block just to encapsulate the whole output with the given Rich Text Formatting.

3. Using *ToParse* = yes for embedded labels

Objective: Create a label that embeds another label

Solution:

We've already created the label NameAndCutLength that embeds the output of 2 other labels. Now we'll examine the generated .frm file. The one thing to notice here is that the attribute ToParse of the token that returns the string with the embedded labels (`<DATA Column="Name_And_Length" >`) is set to "yes". This is how we tell the software to look for and resolve embedded labels.

```

<?xml version="1.0" encoding="UTF-8"?>
<REPORT_FORMATTING
Name="NameAndCutLength"
Description="">
<DESIGN_TIME
ProgId="SP3DLabelFormatDesigner.RTFLabel"
Action=""
Arg="" />

<RUN_TIME
ProgId="SP3DLabelsFormat.FormatLabel"
Action="RTFLabel"
Arg="" />

<FORMATTING_PARAMETERS
Name="NameAndCutLength"
Site="User"
Path="NameAndCutLength.rfp" />

<LAYOUT_TEMPLATE
Type="Internal" />

<RTF_LABEL>
<POINTS />

<VECTORS />

<BLOCKS>
<BLOCK
Action="Visible">
<TOKENS>
<TEXT
Value="{\rtf1\ansi\deff0{\fonttbl{\f0\fnil\fcharset0 MS Shell Dlg;}} \viewkind4\uc1\pard\lang1033\f0\fs17 "
ToParse="no"
Visible="yes" />

<DATA
Column="Name_And_Length"
ToParse="yes"
Visible="yes" />

<TEXT

```

```

    Value=" \par } "
    ToParse="no"
    Visible="yes" />
  </TOKENS>
</BLOCK>
</BLOCKS>
</RTF_LABEL>
</REPORT_FORMATTING>

```

4. Using a POINT object

Objective: Create a label that will return the position of Pipe Components, Specialties, and Instruments

Solution:

The report label for positioning consists of 4 separate files – a .rtp file that has references to the other files, a query that returns raw data, a formatting file that formats the data appropriately and a formatting parameters file that helps in formatting units of measure as well as coordinate transformations between different coordinate systems.

To write a label that returns positions, we have to write all 4 files by hand, since there isn't UI support for it.

Definition (Common Occurrence Location.rtp):

Let's start with the .rtp file. We'll call the label "Common Occurrence Location" :

```

<?xml version="1.0" encoding="UTF-8"?>
<REPORT_TEMPLATE
Name="Common Occurrence Location"
Description=" "
<QUERIES>
  <QUERY
    Name="Common Occurrence Location"
    Site="User"
    Path="Common Occurrence Location.rqe" />
</QUERIES>

<FORMATTING
Name="Common Occurrence Location"
Site="User"
Path="Common Occurrence Location.rfm" />
</REPORT_TEMPLATE>

```

We can use this file as a template – all it does is say that we'll get the raw data/query from a "Common Occurrence Location.rqe" file, and the formatting from "Common Occurrence Location.rfm". If you change the parts in bold, you can use it for any other label.

Query (Common Occurrence Location.rqe):

Coordinates in SP3D are stored on different interfaces depending on the type of object. To define a coordinate label, the query must get the desired values from the appropriate interfaces. This can be done using a COM query or an SQL query. It is a must to always return all three coordinates in the query even if one is interested in displaying only one or two.

For Piping Components, Instruments and Specialties, the position comes from IJOccurrence interface as follows:

IJOccurrence::ServerToClient12 – X
 IJOccurrence::ServerToClient13 – Y
 IJOccurrence::ServerToClient14 – Z

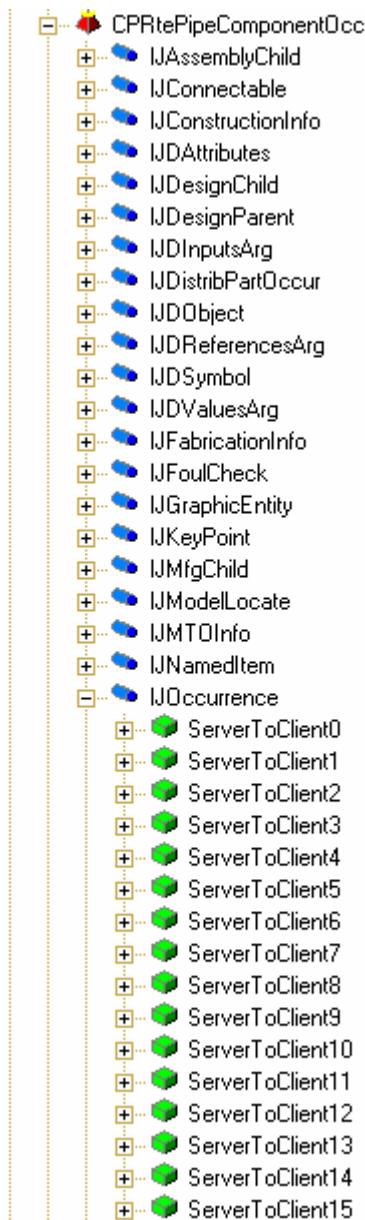


Fig. 11

Combining this with the knowledge we already have about how to write queries, we get the following for the .rqe file:

```
<?xml version="1.0" encoding="UTF-8"?>
<REPORT_QUERY Name="Common Occurrence Location" Description=" " RequiresFilter="No">
  <DESIGN_TIME Progid="SP3DReportsQueryBuilder.COMQuery" Action="" Arg="" />

  <RUN_TIME Progid="SP3DRuntimeQuery.CQueryInterpreter" Action="" Arg="" />

  <RETURNED_PROPERTIES>
```

```

<RETURNED_PROPERTY Name="X" SQLType="Double">
<PATHS>
  <PATH SourceType="*" DestinationInterface="IJOccurrence" DestinationProperty="ServerToClient12">
    <STROKES />
  </PATH>
</PATHS>
</RETURNED_PROPERTY>

<RETURNED_PROPERTY Name="Y" SQLType="Double">
<PATHS>
  <PATH SourceType="*" DestinationInterface="IJOccurrence" DestinationProperty="ServerToClient13">
    <STROKES />
  </PATH>
</PATHS>
</RETURNED_PROPERTY>

<RETURNED_PROPERTY Name="Z" SQLType="Double">
<PATHS>
  <PATH SourceType="*" DestinationInterface="IJOccurrence" DestinationProperty="ServerToClient14">
    <STROKES />
  </PATH>
</PATHS>
</RETURNED_PROPERTY>
</RETURNED_PROPERTIES>
</REPORT_QUERY>

```

Formatting (Common Occurrence Location.rfm):

This file will look like:

```

<?xml version="1.0" encoding="UTF-8"?>
<REPORT_FORMATTING
Name="Common Occurrence Location"
Description="">
<DESIGN_TIME
ProgId="SP3DLabelFormatDesigner.RTFLabel"
Action=""
Arg="" />

<RUN_TIME
ProgId="SP3DLabelsFormat.FormatLabel"
Action="RTFLabel"
Arg="" />

<LAYOUT_TEMPLATE
Type="Internal" />

<FORMATTING_PARAMETERS
Name="Common Occurrence Location"
Site="User"
Path="Common Occurrence Location.rfp" />

<RTF_LABEL>
<POINTS>
  <POINT
    Name="Occurrence Location"

```

```
X="X"  
Y="Y"  
Z="Z"  
Matrix="MyTransformA"  
UOM="Location">>  
  <READOUT>  
    <DIRECTION />  
  
    <VALUE />  
  </READOUT>  
  </POINT>  
</POINTS>  
  
<BLOCKS>  
  <BLOCK  
    Action="Visible">  
      <TOKENS>  
        <POSITION  
          Visible="Yes"  
          Axis="X"  
          Point="Occurrence Location"/>  
  
        <TEXT  
          Visible="Yes"  
          Value=" " />  
  
        <POSITION  
          Visible="Yes"  
          Axis="Y"  
          Point="Occurrence Location"/>  
  
        <TEXT  
          Visible="Yes"  
          Value=" " />  
  
        <POSITION  
          Visible="Yes"  
          Axis="Z"  
          Point="Occurrence Location"/>  
      </TOKENS>  
    </BLOCK>  
  </BLOCKS>  
</RTF_LABEL>  
</REPORT_FORMATTING>
```

This time the formatting file has <POINTS>:

```
<POINTS>  
  <POINT  
    Name="Occurrence Location"  
    X="X"  
    Y="Y"  
    Z="Z"
```

```

Matrix="MyTransformA"
UOM="Location">
<READOUT>
<DIRECTION />

<VALUE />
</READOUT>
</POINT>
</POINTS>

```

This defines points we'll be using. In our case, we have only one. The <POINT> statement defines a point. It is given the name "Occurrence Location" (user defined), and it has as x, y, and z coordinates the "X", "Y", and "Z" properties returned from the .rqe file. Also, the <POINT> specifies a Matrix and UOM which will play a role in the formatting of the position values, and we know where they are defined by the <FORMATTING_PARAMETERS> tag.

Now that we have declared a point, we can reference it in the <TOKENS> section. The <POSITION> tag returns one of the coordinate values (specified by the Axis) of a point (specified by the Point attribute) expressed in the Matrix and UOM of the point.

Formatting Parameters (Common Occurrence Location.rfp):

We now need to create the Formatting parameters file Common Occurrence Location.rfp:

```

<?xml version="1.0" encoding="UTF-8"?>
<FORMATTING_PARAMETERS
Name="Common Occurrence Location"
Description="" >
<DESIGN_TIME
ProgId=""
Action=""
Arg="" />

<RUN_TIME
ProgId=""
Action=""
Arg="" />

<UOMS>
<UOM
Name="Location"
CanInherit="Yes"
Type="DISTANCE"
Primary="ft"
Secondary="in"
Tertiary=""
UnitsDisplayed="yes"
PrecisionType="Decimal"
DecimalPrecision="0"
FractionalPrecision=""
LeadingZero=""
TrailingZeros=""
ReduceFraction="Yes" />
</UOMS>

<MATRIXES>
<MATRIX

```

```
LinkName="MyTransformA"
DisplayName="Equipment Location"
ModelName=""
PosX="E "
PosY="N "
PosZ="U "
NegX="W "
NegY="S "
NegZ="D "
XYReferenceAxis="X"
CanInherit="Yes" />
</MATRIXES>
</FORMATTING_PARAMETERS>
```

We are already familiar with the <UOMS> section – it defines distance UOM in ft and in for the coordinate values. The <MATRIXES> section defines a <MATRIX>. The matrix has the following attributes:

LinkName – the internal name by which the matrix will be referred from within the label files

DisplayName – the name which will appear in the UI (if label is used in a report)

ModelName - can specify a coordinate system in the model in whose coordinates to express (transform) the values. If there's nothing specified for this attribute (“”), the system will default to Global.

PosX, NegXP PosY , etc.. - determine how the different axis values will be labeled.

XYReferenceAxis – from which axis to measure angles (used for orientations, not much use for it here)

CanInherit – if the matrix can use a coordinate system passed down to it by the calling application. So even if you have specified a ModelName, if you have set CanInherit to Yes your coordinates will be shown expressed in the coordinate system picked in PinPoint. The same will be true in Drawings if you have selected a Coordinate System in the properties.

5. Conditional Formatting - I

Part 1: Raw Values

We will create a label that returns a value of High, Medium or Low based on the unit weight (weight per unit length) of a steel member.

The criteria are:

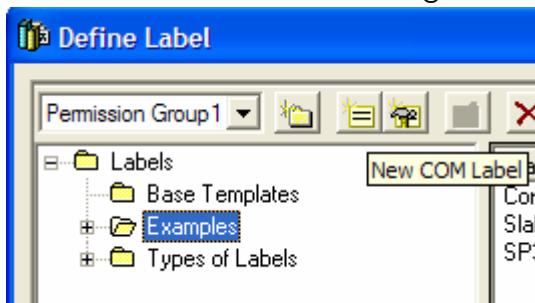
Unit Weight > 60 High

Between 20 and 60 Medium

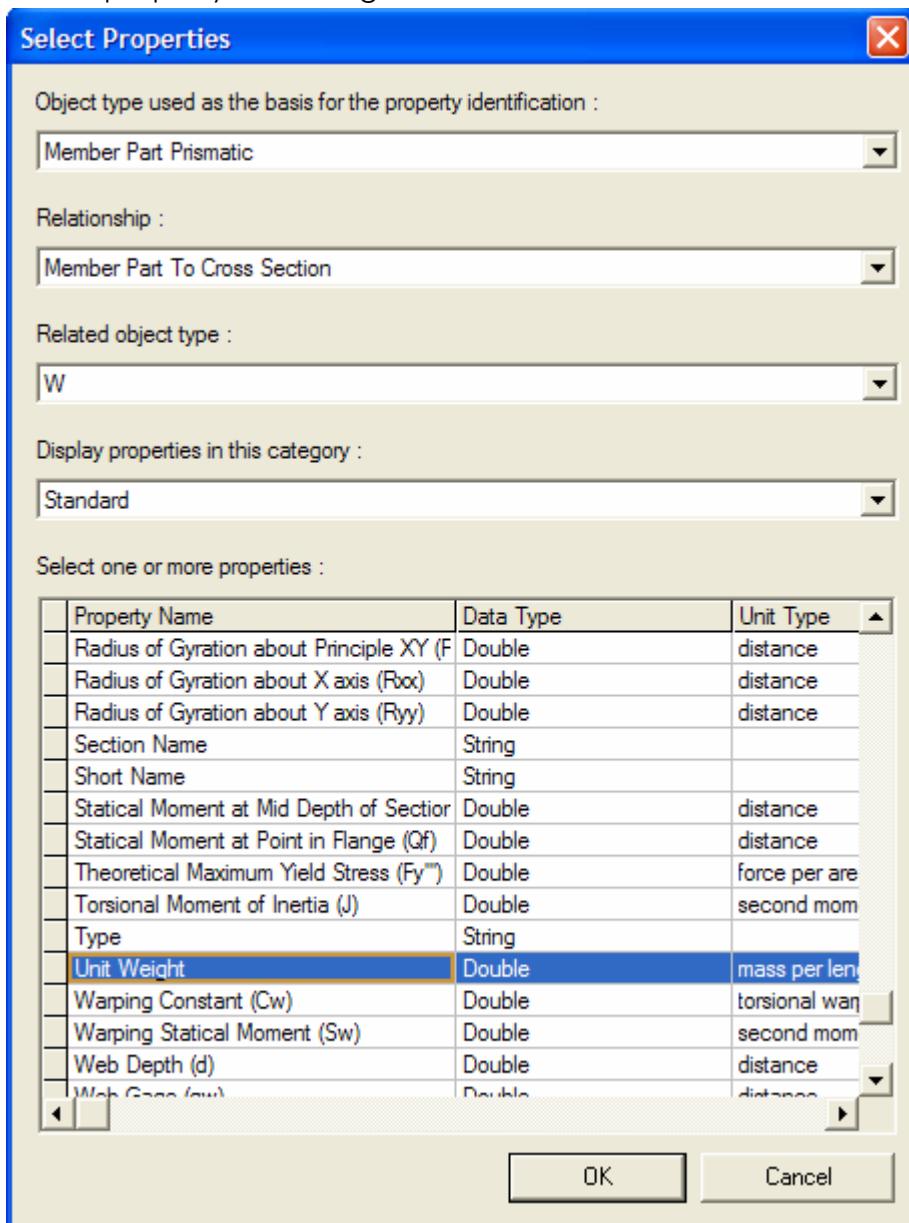
Less than 20 Low

Based on the raw values from the database.

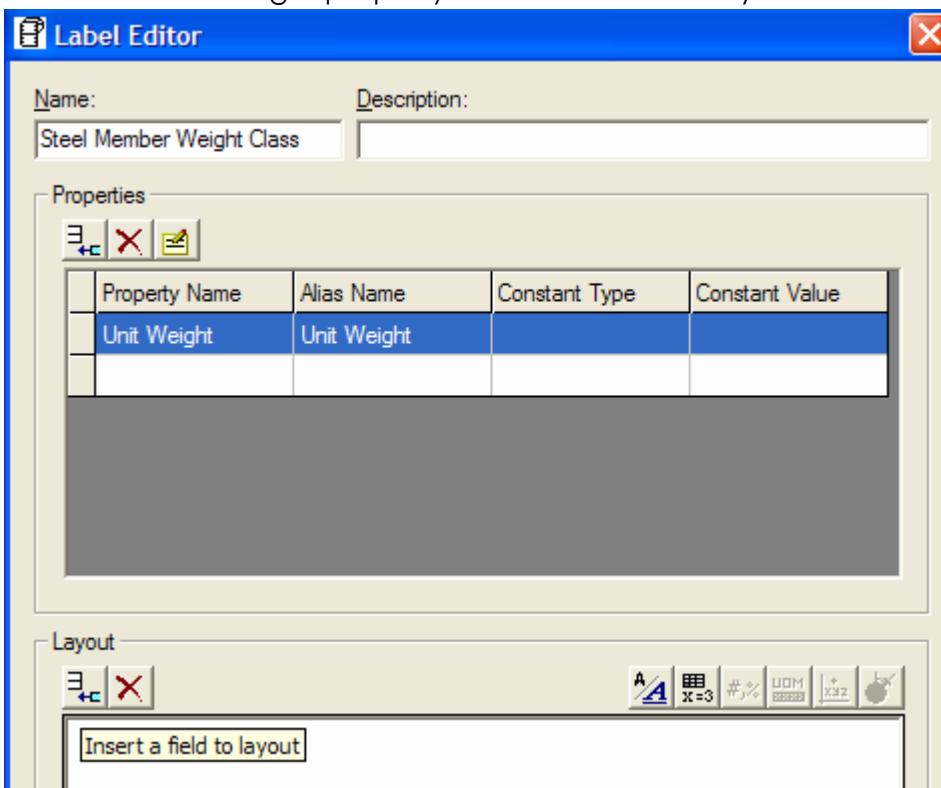
- Define a new COM label using the New COM Label command in the catalog task.



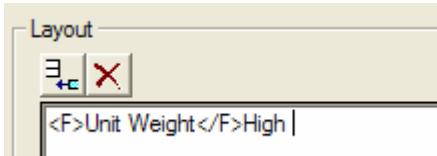
- Name the label Steel Member Weight Class
- Remove the existing property 'Name'
- Add a property 'Unit Weight' as below



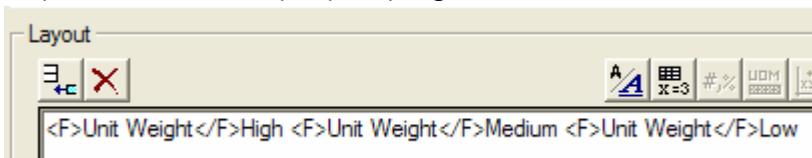
5. Select the Unit Weight property and add it to the Layout as below



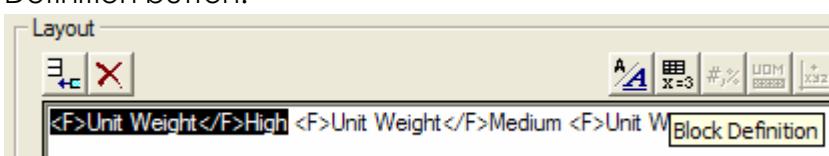
6. Type in the word 'High' and a space after it.



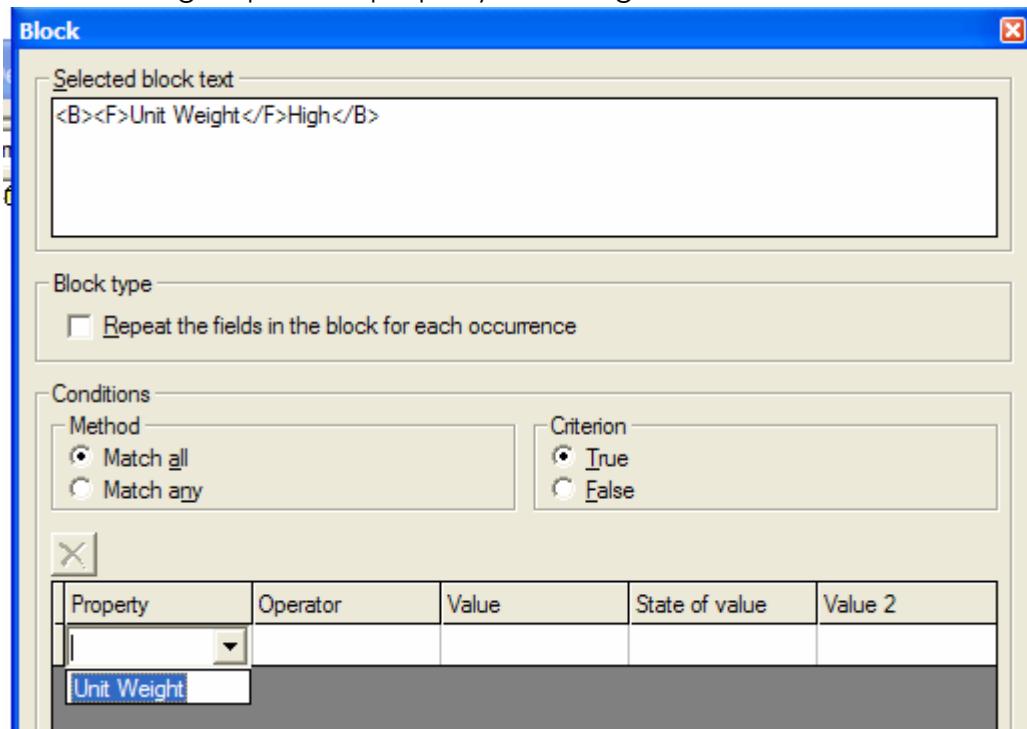
7. Add the property to layout again and then enter the word 'Medium' and a space after it.
 8. Repeat to add the property again and the word 'Low'



9. Select the first instance of the property in the layout and the word High and click the Block Definition button.



10. In the lower grid, pick the property Unit Weight



11. Select the operator '>' and the value 60

Property	Operator	Value	State of value	Value 2
Unit Weight	>	60	Raw	

12. Repeat for the second instance of the property as below.

Block

Selected block text
<F>Unit Weight</F>Medium

Block type
 Repeat the fields in the block for each occurrence

Conditions

Method	Criterion
<input checked="" type="radio"/> Match all	<input checked="" type="radio"/> True
<input type="radio"/> Match any	<input type="radio"/> False

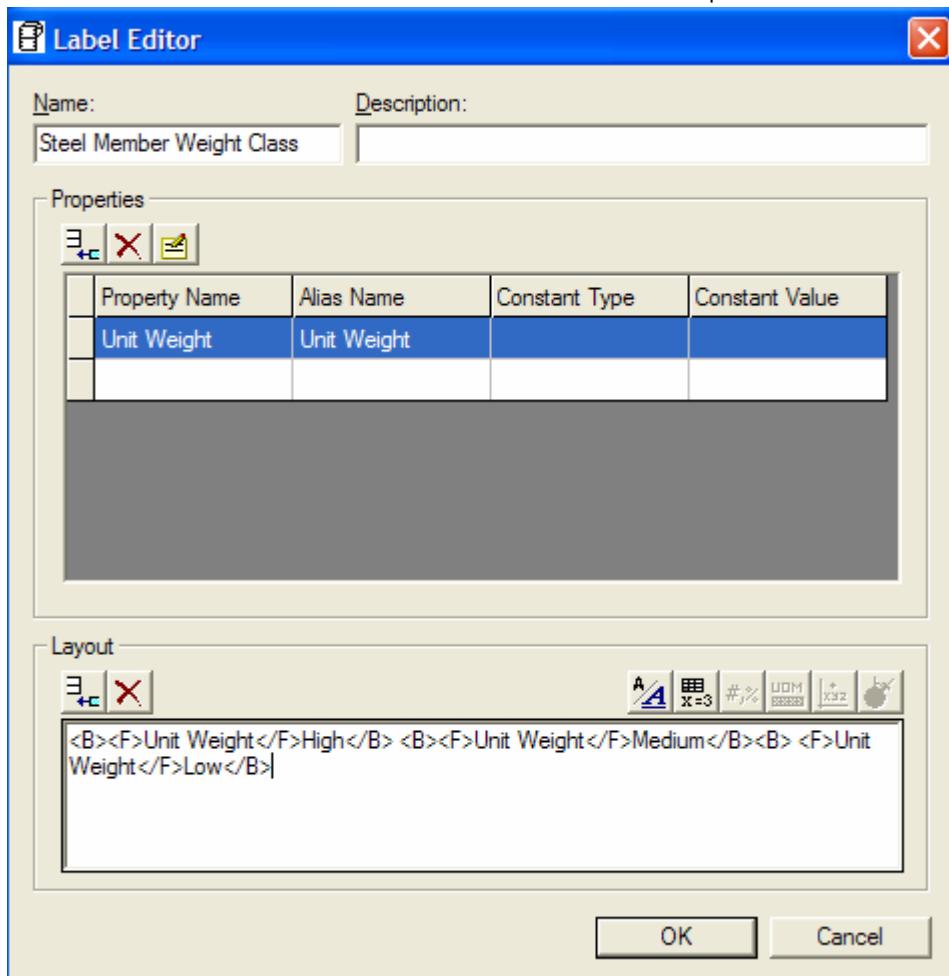
X

Property	Operator	Value	State of value	Value 2
Unit Weight	Between	20	Raw	60

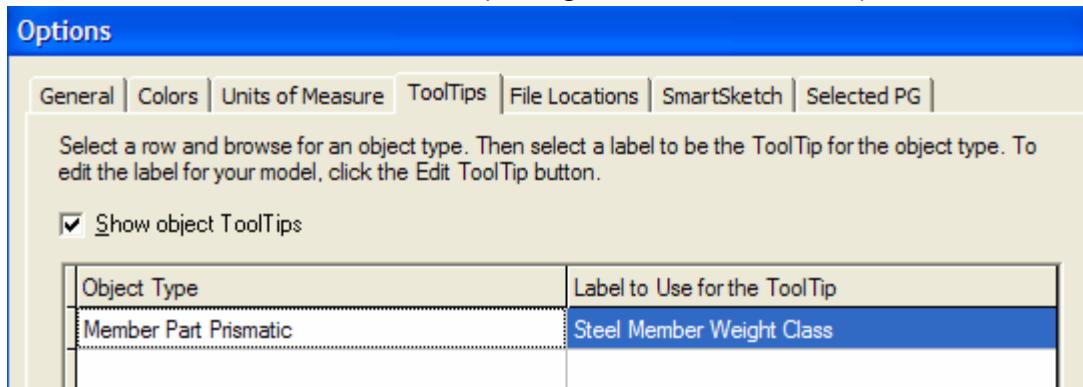
13. Finally, repeat for the third instance to add the third block.

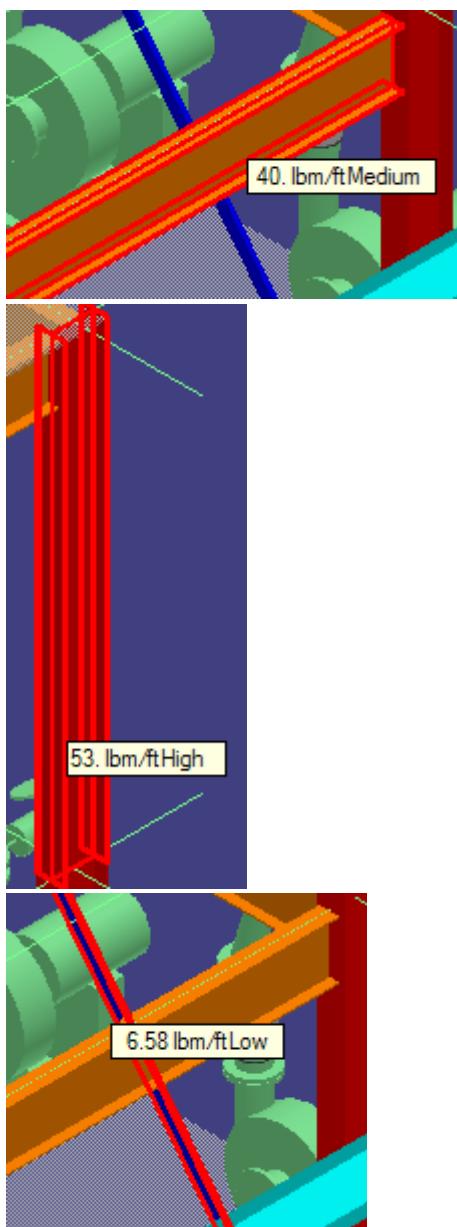
Property	Operator	Value	State of value	Value 2
Unit Weight	<	20	Raw	

14. Once the three blocks are done, click OK to complete the label.



15. Test the label on steel members by using the label as a tooltip.

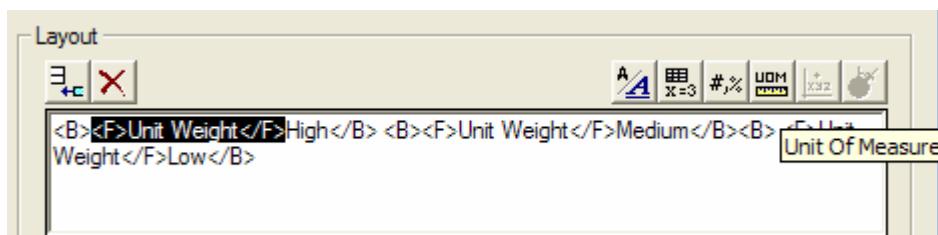




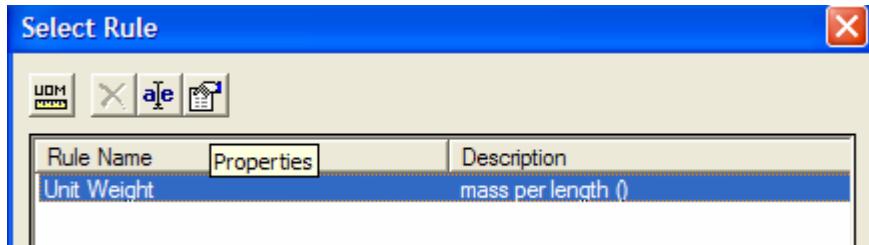
Part 2: Formatted Values

We will now modify above label to operate based on formatted values of the unit weight.

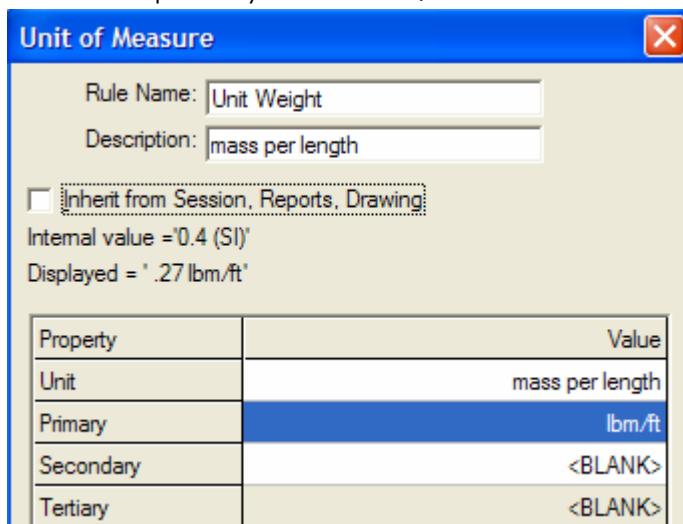
1. Edit the label we just created and select the first instance of the Unit Weight property and pick the UOM button



2. Select the existing rule and change its properties



3. Choose a primary unit of lbm/ft and click OK



4. Click OK again to accept the unit.

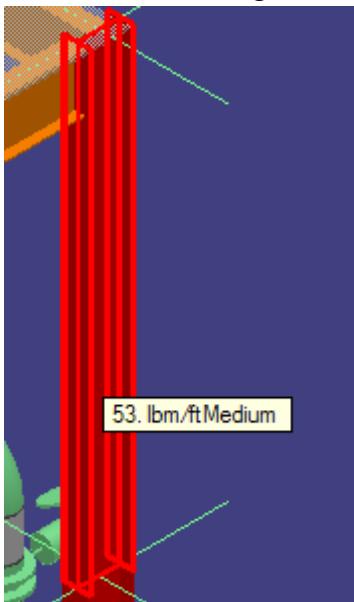
5. Select the 'High' block and change the state of value to post-formatted.

Property	Operator	Value	State of value	Value 2
Unit Weight	>	60	Post-formatted	

6. Similarly change the state of value for the other two blocks.

7. Click OK to save the label.

8. Switch to the Structure task and test the label again as a tooltip. Notice that the output for the column changes to Medium since we are now testing for formatted values



6. Conditional formatting - II

Objective: Create a label for Piping Valve that will return the Tag of its defining feature if that Tag is set (not “”), and the name of the Piping Valve otherwise. This example examines the XML used to create conditional labels – normally we would use the UI shown in the previous example.

Solution:

Although we are asked to write a label for Piping Valves, those properties are common for all Piping Parts, so we can make this label apply to all Piping Parts. Getting the Name of the Piping Part and the Tag of its defining feature should be pretty straightforward. The problem is the formatting part. Essentially, we'll have to implement the following logic.

```
If Tag = “” Then
    LabelOutput = Name
Else
    LabelOutput = Tag
```

Let's first create a label that return the 2 properties from its query. First get the name of Piping Parts

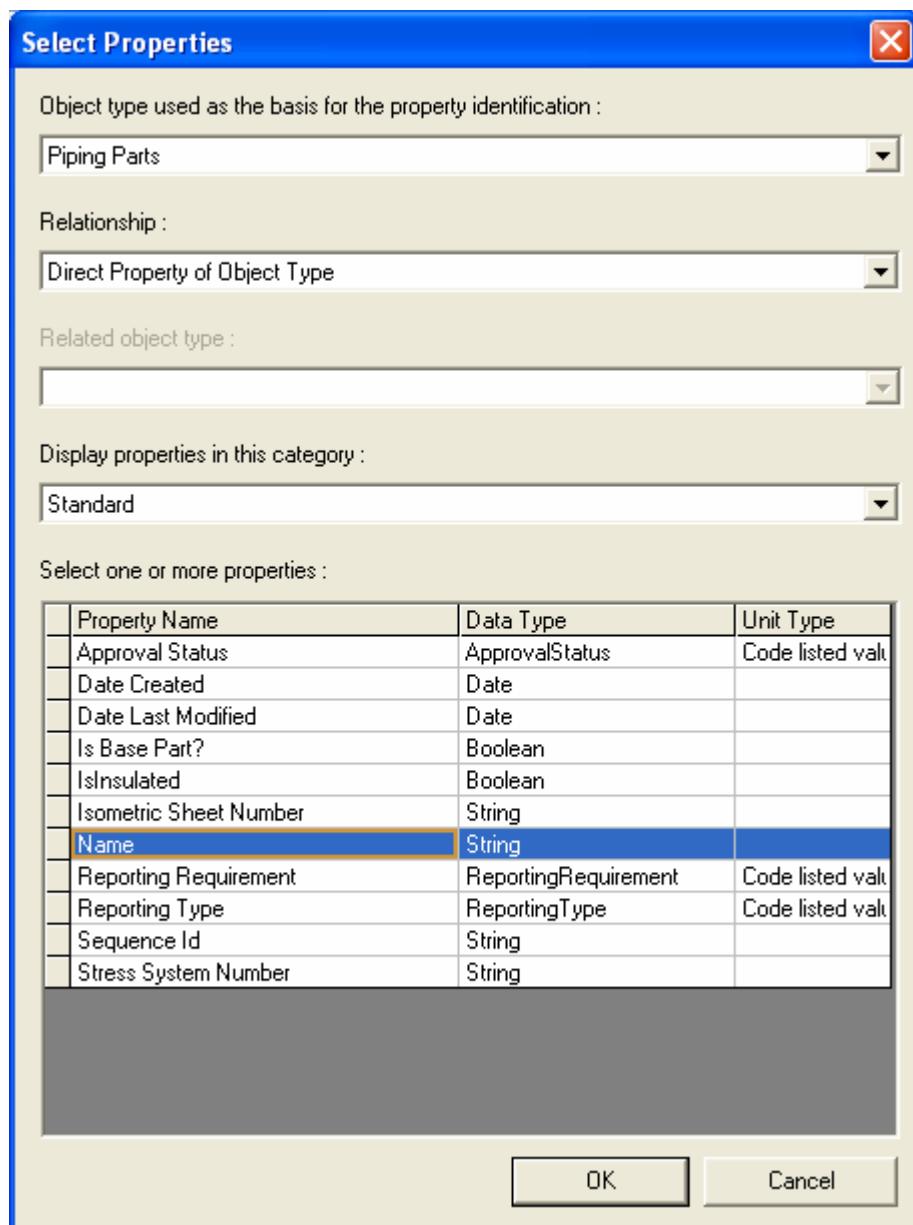


Fig. 12

To add the Tag property, from *Piping Parts* traverse the *Feature To Part* relationship to *Piping Features* as shown:

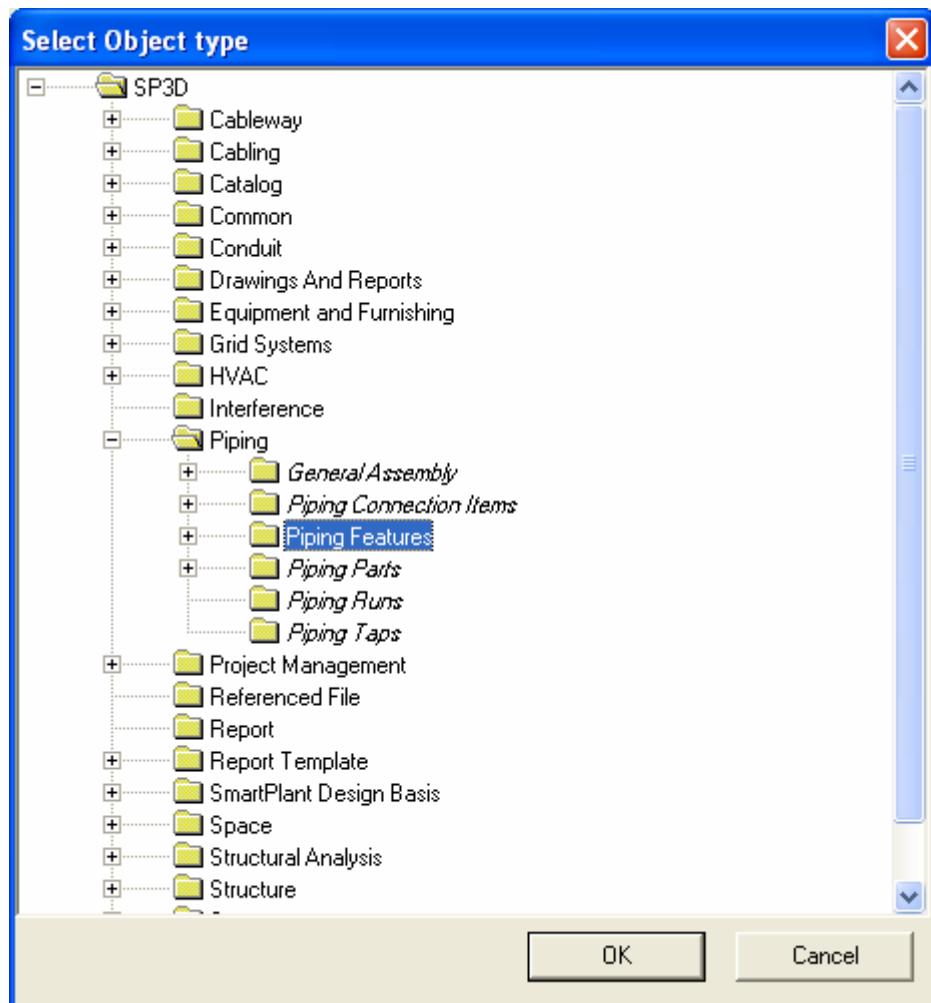


Fig. 13

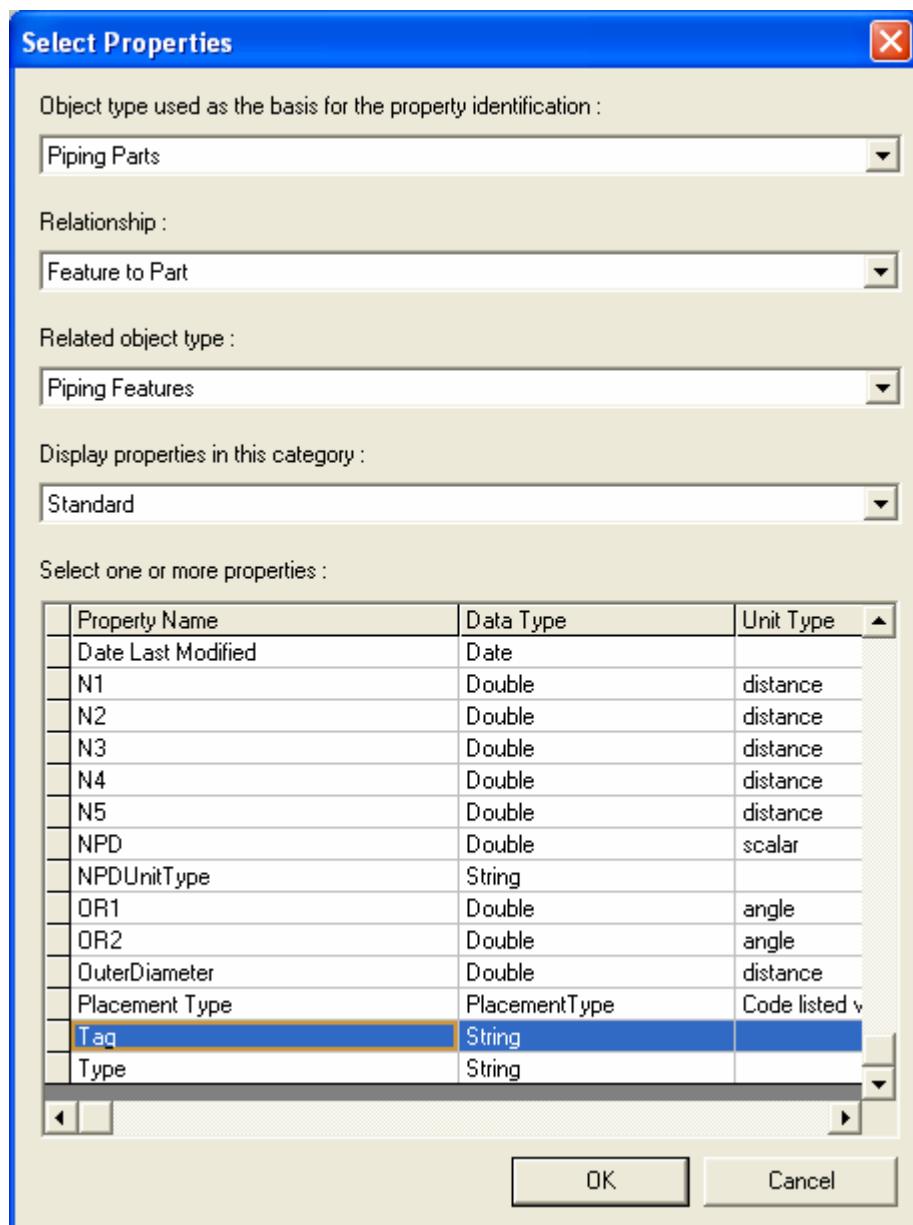


Fig. 14

You don't have to add the 2 properties to the layout, since we cannot do our desired formatting through the UI and will have to edit the .rfm file anyway.

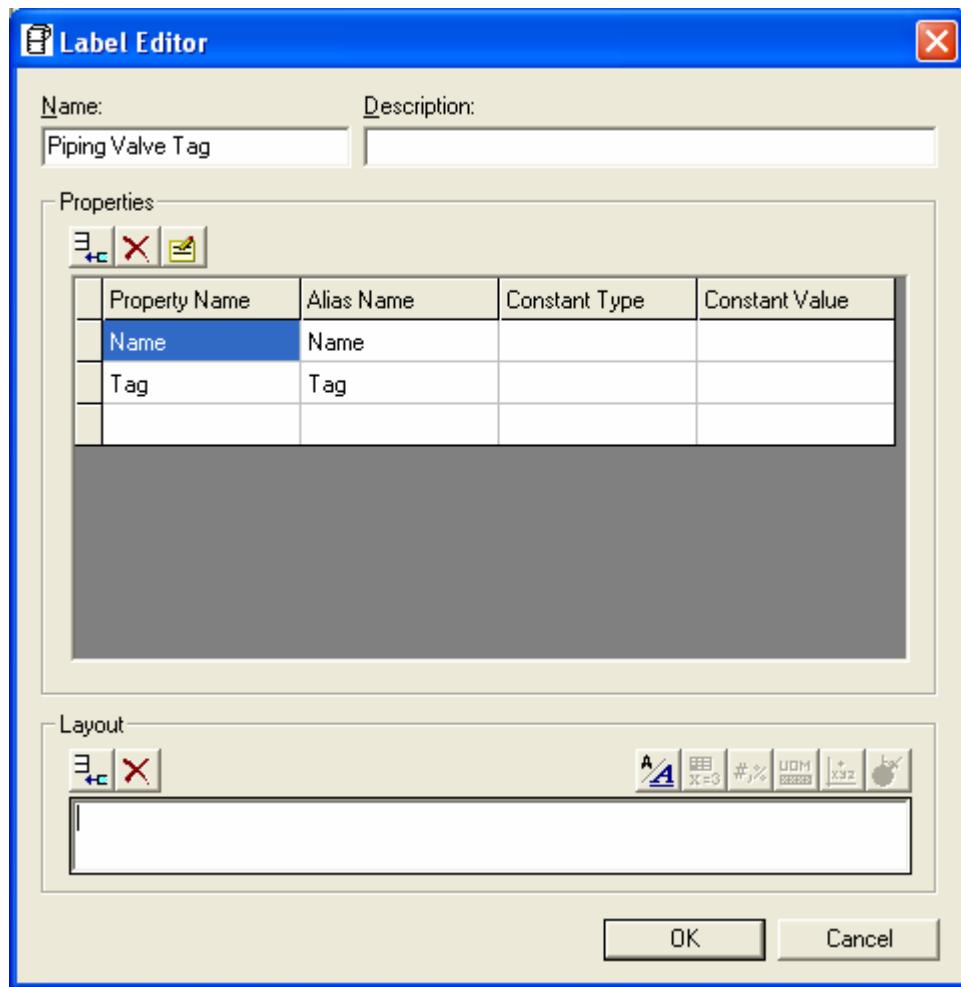


Fig. 15

To accomplish the formatting logic, we'll use the <BLOCK> statement. Every <BLOCK> delimits a block of output (tokens). The advantage of grouping the tokens into blocks is that we can apply an action for the whole block. In our case, the action will be "Visible":

<BLOCK Action="Visible">

Furthermore, we can specify a rule under which this action is to be applied with the <RULE> tag directly nested under <BLOCK>. The <RULE> has the following syntax:

<RULE RuleOperator="Any" RuleOperand="True" />

RuleOperator can take the values "Any" and "All"

RuleOperand can take the values "True" and "False"

What the <RULE> means is "apply the action for the block if Any/All (comes from *RuleOperator*) of the CONDITIONS in the block evaluate to True/False (comes from *RuleOperand*)".

Conditions are defined by the <CONDITION> tag directly nested under the token they refer to.

<CONDITION ConditionOperator="Equal" ConditionOperand1="" StateOfValue="Raw" />

It basically constructs and evaluates a logic expression comparing the Token (in its “Raw”, “PreFormatted”, or “PostFormatted” state – see Appendix), and the *ConditionOperand1* based on the *ConditionOperator* (Equal, Not Equal, GreaterThan, etc). If the *ConditionOperator* is *Between* or *NotBetween*, we can use a second *ConditionOperand2*. The expression returns True or False.

Let's apply all this in our case. Open the Piping Valve Tag.rfm:

```
<?xml version="1.0" encoding="UTF-8"?>
<REPORT_FORMATTING
Name="Piping Valve Tag"
Description="">
<DESIGN_TIME
ProgId="SP3DLabelFormatDesigner.RTFLabel"
Action=""
Arg="" />

<RUN_TIME
ProgId="SP3DLabelsFormat.FormatLabel"
Action="RTFLabel"
Arg="" />

<FORMATTING_PARAMETERS
Name="Piping Valve Tag"
Site="User"
Path="Piping Valve Tag.rfp" />

<LAYOUT_TEMPLATE
Type="Internal" />

<RTF_LABEL>
<POINTS />

<VECTORS />

<BLOCKS>
<BLOCK
Action="Visible">
<TOKENS>
<TEXT
Value="{\rtf1\ansi\ansicpg1252\deff0{\fonttbl{\f0\fnil\fcharset0 MS Shell Dlg;}} \viewkind4\uc1\pard\lang1033\f0\fs17
\par } "
ToParse="no"
Visible="yes" />
</TOKENS>
</BLOCK>
</BLOCKS>
</RTF_LABEL>
</REPORT_FORMATTING>
```

We see one block containing a single TEXT token. We don't need this TEXT token, so we can delete it. We also need 2 <BLOCK>'s – one will be made visible only when the *Tag* property is “” and it will output the *Name* property, and the second one will be visible only when the *Tag* is not “” and it will output the *Tag* itself. Let's add the 2 blocks with their respective outputs, and not worry about the conditions at this point:

```
<?xml version="1.0" encoding="UTF-8"?>
<REPORT_FORMATTING
Name="Piping Valve Tag"
```

```

Description=""
<DESIGN_TIME
ProgId="SP3DLabelFormatDesigner.RTFLabel"
Action=""
Arg="" />

<RUN_TIME
ProgId="SP3DLabelsFormat.FormatLabel"
Action="RTFLabel"
Arg="" />

<FORMATTING_PARAMETERS
Name="Piping Valve Tag"
Site="User"
Path="Piping Valve Tag.rfp" />

<LAYOUT_TEMPLATE
Type="Internal" />

<RTF_LABEL>
<POINTS />

<VECTORS />

<BLOCKS>
<BLOCK Action="Visible">
<TOKENS>
<DATA Column="Name" ToParse="NO" Visible="yes" />
</TOKENS>
</BLOCK>

<BLOCK Action="Visible">
<TOKENS>
<DATA Column="Tag" ToParse="NO" Visible="yes" />
</TOKENS>
</BLOCK>

</BLOCKS>
</RTF_LABEL>
</REPORT_FORMATTING>

```

Now let's add a condition for the second block – we want it to be visible only when the Tag is not “”. This means we'll have to add the following condition to the <DATA>:

```
<CONDITION ConditionOperator="Equal" ConditionOperand1="" StateOfValue="Raw" />
```

this translates to:

```
If Tag=""
    condition = True
Else
    Condition=False
```

To make the whole block visible ONLY when Tag is not “”, we need to put a RULE under the <BLOCK>

```
<RULE RuleOperator="Any" RuleOperand="False" />
```

We have only one CONDITION in this block, so we could have put RuleOperator="All" just as well. This line says "make this whole block visible only if ANY of the conditions contained in it (a single one here) returns False". The whole block becomes:

```
<BLOCK Action="Visible">
  <RULE RuleOperator="Any" RuleOperand="False" />

  <TOKENS>
    <DATA Column="Tag" ToParse="NO" Visible="yes">
      <CONDITION ConditionOperator="Equal" ConditionOperand1="" StateOfValue="Raw" />
    </DATA>

  </TOKENS>
</BLOCK>
```

Now let's do the first block. The problem we see right away is that we need a condition on *Tag*, but we only have *Name*. To get around this, we add the *Tag* DATA again, but since we need it to just evaluate a condition and we are not interested in outputting it from that block, we set its *Visible* attribute to false. Following the pattern above we get for the 1st block:

```
<BLOCK Action="Visible">
  <RULE RuleOperator="Any" RuleOperand="True" />

  <TOKENS>
    <DATA Column="Tag" ToParse="NO" Visible="no">
      <CONDITION ConditionOperator="Equal" ConditionOperand1="" StateOfValue="Raw" />
    </DATA>

    <DATA Column="Name" ToParse="NO" Visible="yes" />

  </TOKENS>
</BLOCK>
```

Thus, our complete .rfm file becomes

```
Name="Piping Valve Tag"
Description=""
<DESIGN_TIME
  Progid="SP3DLabelFormatDesigner.RTFLabel"
  Action=""
  Arg="" />

<RUN_TIME
  Progid="SP3DLabelsFormat.FormatLabel"
  Action="RTFLabel"
  Arg="" />

<FORMATTING_PARAMETERS
  Name="Piping Valve Tag"
  Site="User"
  Path="Piping Valve Tag.rfp" />

<LAYOUT_TEMPLATE
```

```
Type="Internal" />

<RTF_LABEL>
  <POINTS />
  <VECTORS />

  <BLOCKS>

    <BLOCK Action="Visible">
      <RULE RuleOperator="Any" RuleOperand="True" />
      <TOKENS>

        <DATA Column="Tag" ToParse="NO" Visible="no">
          <CONDITION ConditionOperator="Equal" ConditionOperand1="" StateOfValue="Raw" />
        </DATA>

        <DATA Column="Name" ToParse="NO" Visible="yes" />

      </TOKENS>
    </BLOCK>

    <BLOCK Action="Visible">
      <RULE RuleOperator="Any" RuleOperand="False" />
      <TOKENS>

        <DATA Column="Tag" ToParse="NO" Visible="yes">
          <CONDITION ConditionOperator="Equal" ConditionOperand1="" StateOfValue="Raw" />
        </DATA>

      </TOKENS>
    </BLOCK>

  </BLOCKS>
</RTF_LABEL>
</REPORT_FORMATTING>
```

7. Using a VECTOR object for orientations (Optional)

Objective: Create a label that will return the orientation for Equipment Nozzles.

Solution:

Similarly to the POINT example, to get the orientation we need to construct a geometric object VECTOR, and then we can extract the angles of that vector from a horizontal and a vertical plane. Again, we'll have to write all the files by hand.

Definition (Equipment Nozzle Orientation.rtp):

Let's start with the .rtp file. We'll call the label "Equipment Nozzle Orientation":

```
<?xml version="1.0" encoding="UTF-8"?>
<REPORT_TEMPLATE
Name="Equipment Nozzle Orientation"
Description="" >
<QUERIES>
<QUERY
Name="Equipment Nozzle Orientation"
Site="User"
Path="Equipment Nozzle Orientation.rqe" />
</QUERIES>

<FORMATTING
Name="Equipment Nozzle Orientation"
Site="User"
Path="Equipment Nozzle Orientation.rfm" />
</REPORT_TEMPLATE>
```

We can use this file as a template – all it does is say that we'll get the raw data/query from a "Equipment Nozzle Orientation.rqe" file, and the formatting from " Equipment Nozzle Orientation.rfm".

Query (Equipment Nozzle Orientation.rqe)

Nozzle direction is 3 properties on IJDistribPort (OrientationX, OrientationY, OrientationZ). The query is then:

```
<?xml version="1.0" encoding="UTF-8"?>
<REPORT_QUERY Name="Equipment Nozzle Orientation" Description="" RequiresFilter="No">
<DESIGN_TIME Progid="SP3DReportsQueryBuilder.COMQuery" Action="" Arg="" />

<RUN_TIME Progid="SP3DRuntimeQuery.CQueryInterpreter" Action="" Arg="" />

<RETURNED_PROPERTIES>
<RETURNED_PROPERTY Name="NozDirX" SQLType="Double">
<PATHS>
<PATH SourceType="*" DestinationInterface="IJDistribPort" DestinationProperty="OrientationX">
<STROKES />
</PATH>
</PATHS>
</RETURNED_PROPERTY>

<RETURNED_PROPERTY Name="NozDirY" SQLType="Double">
<PATHS>
<PATH SourceType="*" DestinationInterface="IJDistribPort" DestinationProperty="OrientationY">
<STROKES />
</PATH>
</PATHS>
</RETURNED_PROPERTY>

<RETURNED_PROPERTY Name="NozDirZ" SQLType="Double">
<PATHS>
<PATH SourceType="*" DestinationInterface="IJDistribPort" DestinationProperty="OrientationZ">
<STROKES />
</PATH>
</PATHS>
</RETURNED_PROPERTY>
```

```
</RETURNED_PROPERTIES>
</REPORT_QUERY>
```

Formatting (Equipment Nozzle Orientation.rfm):

Similar to the point example, we'll need to construct a <VECTOR> in the <VECTORS> section:

```
<VECTORS>
  <VECTOR
    Name="NozzleOrientation"
    X="NozDirX"
    Y="NozDirY"
    Z="NozDirZ"
    Matrix="MyTransformA"
    UOM="HVOrientationDegMinSec"
    HorizontalAngleStyle="QuadrantCW"
    VerticalAngleStyle="ElevationMatrixZ" />
</VECTORS>
```

The vector is made out of the properties we get from the rqe file (attributes X,Y,Z). We specify a *Matrix* (of a Coordinate System) and *UOM* in which to express the angles (the *Matrix* and the *UOM* will again come from the .rqp file). We also specify the style in which both the Horizontal and the Vertical angle will be expressed (the description and possible values of those are given in the Appendix).

To find the angle of this VECTOR, we need to specify an ORIENTATION token.

The <ORIENTATION> tag has the following attributes:

Plane: determines if we want the angle in the "Horizontal" or "Vertical" plane

Vector: The name of the vector to use

Visible: Yes/No

So if we want to express the orientation as
horizontal angle, vertical angle

We need to add the following tokens to the <TOKENS> section:

```
<ORIENTATION
  Plane="Horizontal"
  Vector="NozzleOrientation"
  Visible="Yes" />

<TEXT
  Visible="Yes"
  Value="," />
```

```
<ORIENTATION
  Plane="Vertical"
  Vector="NozzleOrientation"
  Visible="Yes" />
```

This will be enough to write the label. However, we can further enhance it: if the vector is aligned with the vertical axis, there's no need to output horizontal angle. Similarly, if the vector is perpendicular to the vertical axis, there's no need to output a vertical angle. Using <CONDITION>'s and <BLOCK>'s we can do that as shown:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```

<REPORT_FORMATTING Name="Equipment Nozzle Orientation" Description="">
<DESIGN_TIME Progid="SP3DLabelFormatDesigner.RTFLabel" Action="" Arg="" />

<RUN_TIME Progid="SP3DLLabelsFormat.FormatLabel" Action="RTFLabel" Arg="" />

<LAYOUT_TEMPLATE Type="Internal" />

<FORMATTING_PARAMETERS Name="Equipment Nozzle Orientation" Site="User" Path="Equipment Nozzle
Orientation.rfp" />

<RTF_LABEL>
  <VECTORS>
    <VECTOR Name="NozzleOrientation" X="NozDirX" Y="NozDirY" Z="NozDirZ" Matrix="MyTransformA"
UOM="HVOrientationDegMinSec" HorizontalAngleStyle="QuadrantCW" VerticalAngleStyle="ElevationMatrixZ" />
  </VECTORS>

  <BLOCKS>
    <BLOCK Action="Visible">
      <RULE RuleOperator="Any" RuleOperand="True" />

      <TOKENS>
        <DATA Column="NozDirZ" Visible="No">
          <CONDITION ConditionOperator="Between" ConditionOperand1="-0.001" ConditionOperand2="0.001"
StateOfValue="Raw" />
        </DATA>

        <ORIENTATION Plane="Horizontal" Vector="NozzleOrientation" Visible="Yes" />
      </TOKENS>
    </BLOCK>

    <BLOCK Action="Visible">
      <RULE RuleOperator="Any" RuleOperand="True" />

      <TOKENS>
        <DATA Column="NozDirZ" Visible="No">
          <CONDITION ConditionOperator="Between" ConditionOperand1="0.999" ConditionOperand2="1.001"
StateOfValue="Raw" />
        </DATA>

        <ORIENTATION Plane="Vertical" Vector="NozzleOrientation" Visible="Yes" />
      </TOKENS>
    </BLOCK>

    <BLOCK Action="Visible">
      <RULE RuleOperator="Any" RuleOperand="True" />

      <TOKENS>
        <DATA Column="NozDirZ" Visible="No">
          <CONDITION ConditionOperator="Between" ConditionOperand1="-1.001" ConditionOperand2="-0.999"
StateOfValue="Raw" />
        </DATA>

        <ORIENTATION Plane="Vertical" Vector="NozzleOrientation" Visible="Yes" />
      </TOKENS>
    </BLOCK>
  </BLOCKS>
</RTF_LABEL>

```

```

<BLOCK Action="Visible">
  <RULE RuleOperator="Any" RuleOperand="True" />

  <TOKENS>
    <DATA Column="NozDirZ" Visible="No">
      <CONDITION ConditionOperator="Between" ConditionOperand1="0.001" ConditionOperand2="0.999"
StateOfValue="Raw" />
    </DATA>

    <ORIENTATION Plane="Horizontal" Vector="NozzleOrientation" Visible="Yes" />

    <TEXT Visible="Yes" Value="," />

    <ORIENTATION Plane="Vertical" Vector="NozzleOrientation" Visible="Yes" />
  </TOKENS>
</BLOCK>

<BLOCK Action="Visible">
  <RULE RuleOperator="Any" RuleOperand="True" />

  <TOKENS>
    <DATA Column="NozDirZ" Visible="No">
      <CONDITION ConditionOperator="Between" ConditionOperand1="-0.999" ConditionOperand2="-0.001"
StateOfValue="Raw" />
    </DATA>

    <ORIENTATION Plane="Horizontal" Vector="NozzleOrientation" Visible="Yes" />

    <TEXT Visible="Yes" Value="," />

    <ORIENTATION Plane="Vertical" Vector="NozzleOrientation" Visible="Yes" />
  </TOKENS>
</BLOCK>
</BLOCKS>
</RTF_LABEL>
</REPORT_FORMATTING>

```

Formatting Parameters (Equipment Nozzle Orientation.rfp):

The formatting file is very similar to the one given in the POINT example, so we won't go into details here.

Pay attention to the XYReferenceAxis attribute – we didn't really use it for the POINT example, but depending on the value of the <VECTOR HorizontalAngleStyle= >, it may be needed here (see more in Appendix A)

```

<?xml version="1.0" encoding="UTF-8"?>
<FORMATTING_PARAMETERS
Name="Equipment Nozzle Orientation"
Description="">
<DESIGN_TIME
ProgId=""
Action=""
Arg="" />

<RUN_TIME
ProgId=""
Action=""
```

```
Arg="" />

<UOMS>

<UOM
Name="HVOrientationDegMinSec"
CanInherit="No"
Type="ANGLE"
Primary="deg"
Secondary="min"
Tertiary="sec"
UnitsDisplayed="yes"
PrecisionType="Decimal"
DecimalPrecision="0"
FractionalPrecision="2"
LeadingZero="No"
TrailingZeros="No"
ReduceFraction="" />
</UOMS>

<MATRIXES>
<MATRIX
LinkName="MyTransformA"
DisplayName="Equipment Location"
ModelName=""
CSType="0"
PosX="E "
PosY="N "
PosZ="U "
NegX="W "
NegY="S "
NegZ="D "
XYReferenceAxis="1"
CanInherit="Yes" />
</MATRIXES>
</FORMATTING_PARAMETERS>
```

8. Using a FORMAT_EXPRESSION

Objective: Create a label that will return a system's name all in UPPER CASE

Solution:

We can apply VB style formatting to the label. Let's see how this is done.

Create a label called System Name. You could just use the default label with which you are presented when you create a new label:

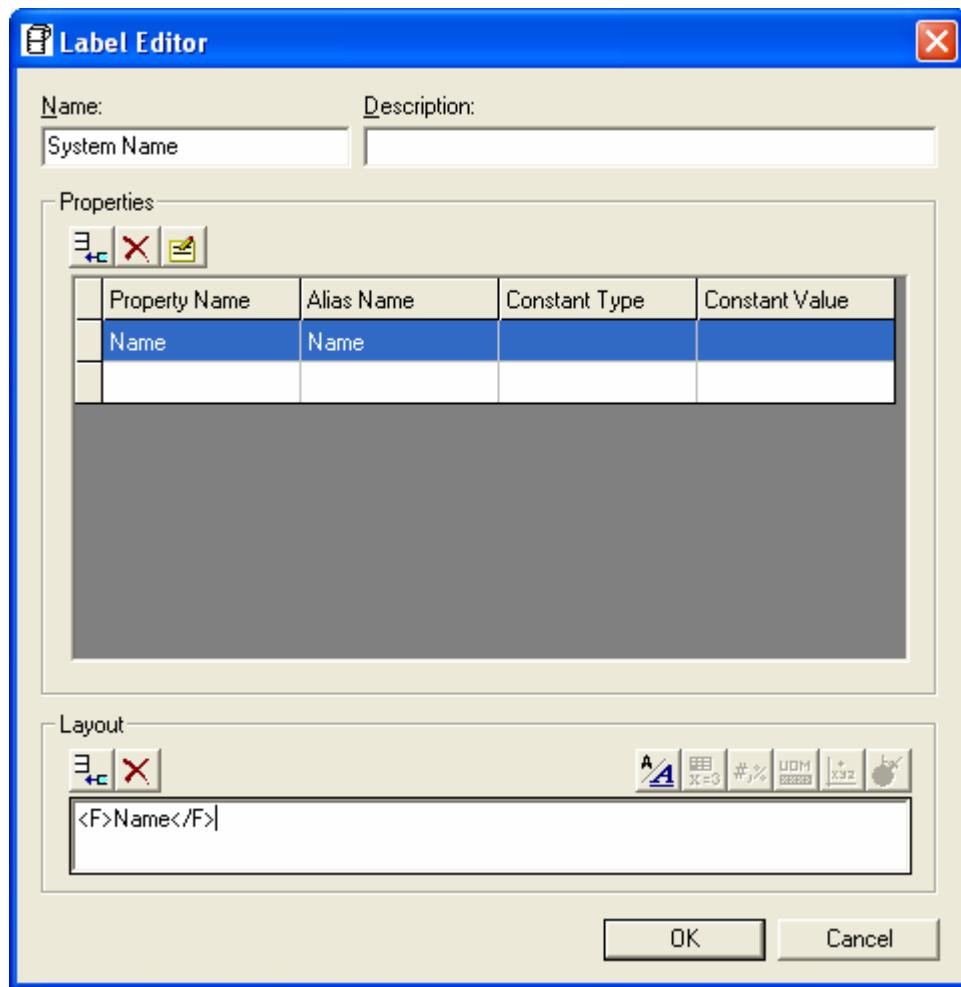


Fig. 19

Open the .rfm file:

```

<?xml version="1.0" encoding="UTF-8"?>
<REPORT_FORMATTING Name="System Name" Description="">
  <DESIGN_TIME Progid="SP3DLabelFormatDesigner.RTFLabel" Action="" Arg="" />

  <RUN_TIME Progid="SP3DLabelsFormat.FormatLabel" Action="RTFLabel" Arg="" />

  <FORMATTING_PARAMETERS Name="System Name" Site="User" Path="System Name.rfp" />

  <LAYOUT_TEMPLATE Type="Internal" />

  <RTF_LABEL>
    <POINTS />

    <VECTORS />
  
```

```

<BLOCKS>
  <BLOCK Action="Visible">
    <TOKENS>
      <TEXT Value=" {\rtf1\ansi\deff0{\fonttbl{\f0\fnil\fcharset0 MS Shell Dlg;}} \viewkind4\uc1\pard\lang1033\f0\fs17 "
ToParse="no" Visible="yes" />

    <DATA Column="Name" ToParse="yes" Visible="yes" />

    <TEXT Value=" \par " ToParse="no" Visible="yes" />
  </TOKENS>
  </BLOCK>
</BLOCKS>
</RTF_LABEL>
</REPORT_FORMATTING>

```

We have an option to format the output as if we'd call the VB function "Format" (see Appendix B). This is done with the tag

```
<FORMAT_EXPRESSION Expression="">
```

Where *Expression* gives the format we'd pass to the VB function. In our case it will be "&", but because we are using this from within XML, we have to write it as

```
<FORMAT_EXPRESSION Expression=">">/>
```

Thus our label will be:

```

<?xml version="1.0" encoding="UTF-8"?>
<REPORT_FORMATTING Name="System Name" Description="">
  <DESIGN_TIME Progid="SP3DLabelFormatDesigner.RTFLabel" Action="" Arg="" />

  <RUN_TIME Progid="SP3DLabeledFormat.FormatLabel" Action="RTFLabel" Arg="" />

  <FORMATTING_PARAMETERS Name="System Name" Site="User" Path="System Name.rfp" />

  <LAYOUT_TEMPLATE Type="Internal" />

<RTF_LABEL>
  <POINTS />

  <VECTORS />

  <BLOCKS>
    <BLOCK Action="Visible">
      <TOKENS>
        <TEXT Value=" {\rtf1\ansi\deff0{\fonttbl{\f0\fnil\fcharset0 MS Shell Dlg;}} \viewkind4\uc1\pard\lang1033\f0\fs17 "
ToParse="no" Visible="yes" />

        <DATA Column="Name" ToParse="yes" Visible="yes" >
          <FORMAT_EXPRESSION Expression=">">/>
        </DATA>

        <TEXT Value=" \par " ToParse="no" Visible="yes" />
      </TOKENS>
    </BLOCK>
  </BLOCKS>
</RTF_LABEL>

```

</REPORT_FORMATTING>

Note that you can also apply a rich set of formatting options through the UI by simply highlighting the property and clicking on the Field Formatting button as shown in the picture. The set of formatting is similar to this offered by Excel.

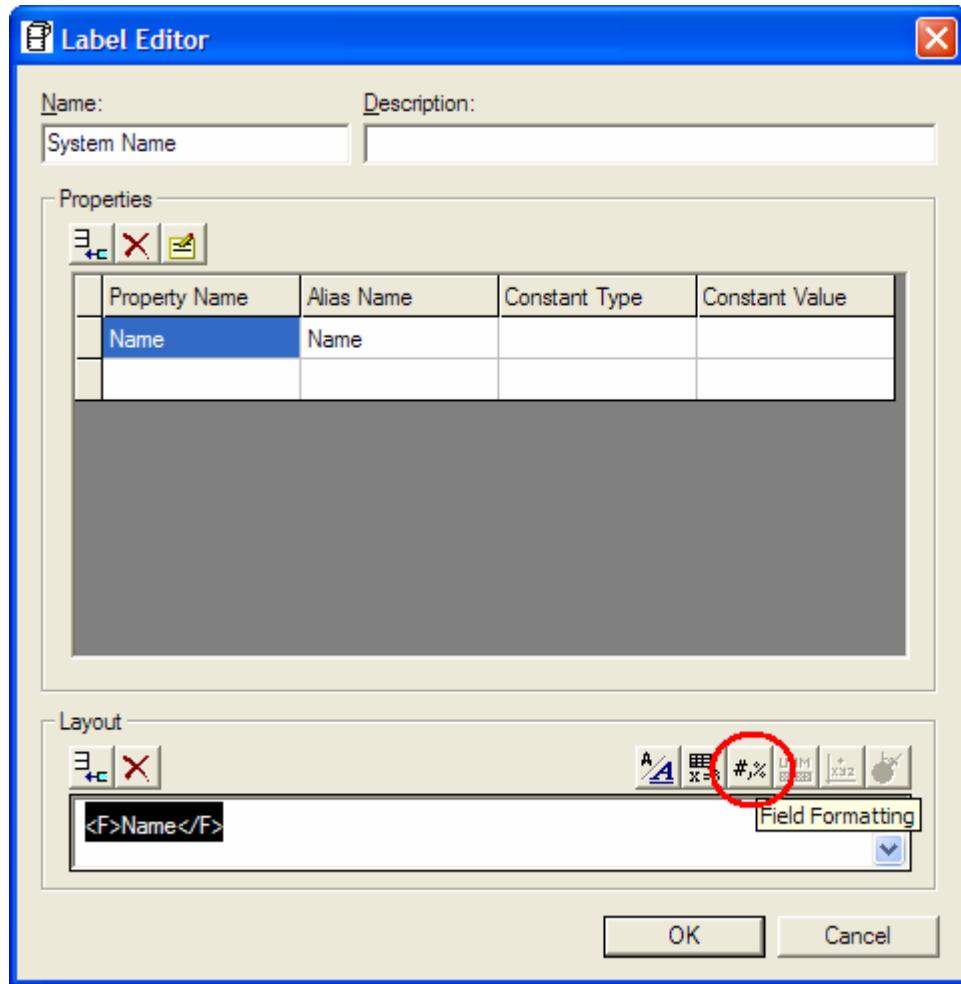


Fig 20

Note: Since some of the VB Format function parameters (as given in the Appendix) are also XML special characters, you will need to take this into consideration. For example, to convert the output to all lower case you need to write:

<**FORMAT_EXPRESSION** Expression="<"

instead of:

<**FORMAT_EXPRESSION** Expression="<"

Appendix A

RTF_Label Schema Definition

The RTFLabel.xsd is the schema to be used to validate the RTF_LABEL element of a REPORT_FORMATTING element.

RTF_LABEL - consists of zero or more **POINTS**, zero or more **BLOCKS**, and zero or more **VECTORS** with Rich Text formatting.

VECTORS - consists of zero or more **VECTOR** elements.

<p>VECTOR - describes a 3D point and contains attributes and READOUT element; provides information about representation for display.</p>	<p>READOUT - contains zero or more elements that define the order of the components in the text output of a point.</p>	<p>VALUE - portion of a point in distance units; used to provide a place holder in the readout for the value component.</p> <p>DIRECTION - refers to the axis (X/East-West, Y/North-South, Z/Elevation-up-down); used to provide a place holder in the readout for the axis component.</p> <p>COORDINATESYSTEM - display name of the coordinate system containing the point value; used to provide a place holder in the readout for the name of the coordinate system.</p>
---	---	--

POINTS - consists of zero or more **POINT** elements.

<p>POINT - describes a 3D point and contains attributes and a READOUT element; provides information about how a point is organized and transformed.</p>	<p>READOUT - contains zero or more elements that define the order of the components in the text output of a point.</p>	<p>VALUE - portion of a point in distance units; used to provide a place holder in the readout for the value component.</p> <p>DIRECTION - refers to the axis (X/East-West, Y/North-South, Z/Elevation-up-down); used to provide a place holder in the readout for the axis component.</p> <p>COORDINATESYSTEM - display name of the coordinate system containing the point value; used to provide a place holder in the readout for the name of the coordinate system.</p>
--	--	--

BLOCKS - consists of zero or more **BLOCK** elements.

<p>BLOCK - consists of zero or more TOKENS elements.</p>	<p>RULE - allows the BLOCK to be displayed or hidden based on a condition (VISIBLE attribute) returned from the TOKENS.</p>	<p>TOKEN - smallest unit in a BLOCK. Can be text or data extracted from a database using SQL. Some data must be transformed (Unit of measure must be extracted from its storage unit - MKSA - into a user preference, possibly imperial).</p>	<p>TEXT - formatted data as Rich Text Format.</p> <p>DATA - provides for simple data to be extracted from the database with no translation.</p> <p>PHYSICAL - displays data that requires a Unit of Measure description.</p> <p>POSITION - displays one axis of a point.</p> <p>ORIENTATION - displays horizontal and vertical angle of a vector.</p> <p>USER - custom data that is not contained in the model database. For example, the name of the user generating the report.</p>
--	--	--	---

			CONDITION - mechanism used to determine whether the TOKEN's value compares to a pre-defined value. Operators are Equal, NotEqual, GreaterThan, LessThan, GreaterThanOrEqual, LessThanOrEqual, Between, and NotBetween; Raw value for PHYSICAL = MKSA and for TEXT = text value. Preformatted value = that just prior to applying FORMAT_EXPRESSION. PostFormatted value = final value.
--	--	--	--

FORMAT_EXPRESSION - a Visual Basic format expression used to format a value. For more information on using the Format Function, see the following references in the Microsoft Development Network (MSDN) documentation:

User-defined numeric formats

Named numeric formats

User-defined date and time formats

Named date and time formats

Attributes found in label <TOKEN>'s :

<AttributeType>	<description>
name=ToParse dt:type=enumeration dt:values=Yes, No	Requires the RTF_LABEL processor to parse the data searching for an embedded label which it must process.
name=Value dt:type=string	The text to be displayed in an RTF_LABEL.
name=Point dt:type=string	The Name of a POINT element in the RTF_LABEL.
name=Column dt:type=string	The Name of a column that is returned by an SQL statement.
name=Field dt:type=string	The Name of a FIELD in the FIELDS collection of Formatting Parameters.
name=Axis dt:type=enumeration dt:values=X, Y, Z	A reference to one of the valid names of an axis.
name=Vector dt:type=string dt:values=X, Y, Z	The Name of a VECTOR element in the RTF_LABEL.
name=Plane dt:type=enumeration dt:values=Vertical, Horizontal	When displaying the angles of a vector, this identifies which plane of a vector is to be displayed - the vertical angle or the horizontal angle.
name=Visible dt:type=enumeration dt:values=Yes, No	Used to evaluate a condition of the value of the token.

<!-- CONDITION -->

<ElementType>	<description>	< <u>attribute</u> >
name=CONDITION content=empty model=closed	Provides a true or false value based on the value of the token.	type=ConditionOperator required=yes type=ConditionOperand1 required=yes type=ConditionOperand2 required=no type=StateOfValue required=yes type=OperandType1 required=no type=OperandType2 required=no
<AttributeType>	<description>	
name=ConditionOperator dt:type=enumeration dt:values=Equal, NotEqual, GreaterThan, LessThan, GreaterThanOrEqual, LessThanOrEqual, Between, NotBetween	Used to evaluate a condition of the value of the token.	
name=ConditionOperand1 dt:type=string	Used to determine the outcome of a condition.	
name=ConditionOperand2 dt:type=string	Used to determine the upper range of a condition.	
name=StateOfValue dt:type=enumeration dt:values=Raw, PreFormatted, PostFormatted	Used to determine the state of the value of the token when processing the token. Raw - Value for Physical is the MKSA value, for Text it is just the Text value. PreFormatted - Value just prior to applying the FORMAT_EXPRESSION. PostFormatted - Final value which may be after applying the FORMAT_EXPRESSION.	
name=OperandType1 dt:type=enumeration dt:values=Constant, Variable	Defines the operand type for purposes of comparison.	
name=OperandType2 dt:type=enumeration dt:values=Constant, Variable	Defines the operand type for purposes of comparison.	

<!-- VECTOR -->

<AttributeType>	<description>
name=Name dt:type=string	Name of the point so it can be referenced by the POSITION element.

name=X dt:type=string	Name of a column for the X-axis returned by an SQL statement.
name=Y dt:type=string	Name of a column for the Y-axis returned by an SQL statement.
name=Z dt:type=string	Name of a column for the Z-axis returned by an SQL statement.
name=Matrix dt:type=string	LinkName of a MATRIX in the MATRIXES collection of Formatting Parameters.
name=UOM dt:type=string	The Name of a UOM in the UOMS collection of Formatting Parameters.
name=HorizontalAngleStyle dt:type=enumeration dt:values=R45, QuadrantNoDir, QuadrantCCW, QuadrantCW, AzimuthCCW, AzimuthCW	<p>Defines the styles allowed to represent an angle in the horizontal plane of a vector.</p> <p>R45 - Range of the angle is 0 to 45. The XYreference of the Matrix is not used. Reference direction (North East South West) :: value :: rotate towards direction (North East South West).</p> <p>Example: W 20 N is equal to 290 degrees.</p> <p>Quadrant - Range of the angle is 0 to 90.</p> <p>QuadrantNoDir - Direction of rotation is from the reference axis. Use the XYreference of the Matrix to determine the reference axis. (North South) :: value :: rotate towards direction (East West).</p> <p>Example: N 70 W is equal to 290 degrees.</p> <p>QuadrantCCW - XYreference of the Matrix is not used. Example: W 20 S is equal to 250 degrees in a compass clockwise direction.</p> <p>QuadrantCW - XYreference of the Matrix is not used. Example: S 70 W is equal to 250 degrees in compass clockwise direction.</p> <p>Azimuth - Range of the angle is 0 to 360. No reference is displayed, because it is assumed to be known. Use the XYreference of the Matrix to determine the reference axis.</p> <p>AzimuthCW - Example: "290" is 290 degrees from the reference axis in the clockwise direction.</p> <p>AzimuthCCW - Example: "290" is 290 degrees from the reference axis in the counter-clockwise direction.</p>
name=VerticalAngleStyle dt:type=enumeration dt:values=ElevationMatrixZ, ElevationPosNeg, Zenith	<p>Defines the styles allowed to represent an angle in the vertical plane of a vector.</p> <p>ElevationMatrixZ - Range of the angle is 0 to 90 and is a reference from the horizontal. When the angle is above the horizon, the value in the PosZ of the Matrix is appended. When the angle is below the horizon, the value in the NegZ of the Matrix is appended.</p> <p>Example: When the NegZ value in the Matrix is "Down", and the angle is 20 degrees below the horizon, the display will be "20 Down".</p> <p>ElevationPosNeg - Range of the angle is -90 to +90 and is a reference from the horizontal. Positive value refers to an angle in the counter-clockwise direction (above the horizon). Negative value refers to an angle in the clockwise direction (below the horizon). Example: "-20" is 20 degrees downward inclination</p>

	<p>from the horizontal.</p> <p>Zenith - Range of the angle is 0 to 180 and is a reference from the zenith or full up. It is a positive angle in the clockwise direction. Example: "110" is 20 degrees downward inclination from the horizontal.</p>
--	---

<!-- ORIENTATION -->

<ElementType>	<description>	<attribute/>	<element/>
name=ORIENTATION content=eltOnly model=closed	A token; provides either a horizontal or a vertical angle and is formatted based on the vector.	type=Plane required=yes type=Vector required=yes type=Visible required=yes	type= FORMAT_EXPRESSION minOccurs=0 maxOccurs=1 type= CONDITION minOccurs=0 maxOccurs=1

XYReferenceAxis – “1” - X axis, “2” – Y axis

Appendix B

Visual Basic Format Functions

`Format(expression[,format]) or Format(expression[, format[, firstdayofweek[, firstweekofyear]]])`

User-Defined String Formats (Format Function)

You can use any of the following characters to create a format expression for strings:

Character	Description
@	Character placeholder
&	Character placeholder. Display a character or nothing. If the string has a character in the position where the ampersand (&) appears, display it; otherwise, display nothing. Placeholders are filled from right to left unless there is an exclamation point character (!) in the format string.
<	Force lowercase. Display all characters in lowercase format.
>	Force uppercase. Display all characters in uppercase format.
!	Force left to right fill of placeholders. The default is to fill placeholders from right to left.

User defined numeric formats

Character	Description
None	Display the number with no formatting.
(0)	Digit placeholder. Display a digit or a zero.
(#)	Digit placeholder. Display a digit or nothing.
(.)	Decimal placeholder.
(%)	Percentage placeholder. The expression is multiplied by 100.
(,)	Thousand separator.
(:)	Time separator.
(/)	Date separator.
(E- E+ e- e+)	Scientific format.
- + \$ ()	Display a literal character. To display a character other than one of those listed, precede it with a backslash (\) or enclose it in double quotation marks ("").

(\)	Display the next character in the format string. To display a character that has special meaning as a literal character, precede it with a backslash (\). To display a backslash, use two backslashes (\\\).
("ABC")	Display the string inside the double quotation marks (" "). To include a string in <i>format</i> from within code, you must use Chr(34) to enclose the text (34 is the character code for a quotation mark ("')).

User-Defined Date/Time Formats (Format Function)

The following table identifies characters you can use to create user-defined date/time formats:

Character	Description
(:)	Time separator.
(/)	Date separator.
c	Display the date as dddd and display the time as tttt, in that order.
d	Display the day as a number without a leading zero (1 – 31).
dd	Display the day as a number with a leading zero (01 – 31).
ddd	Display the day as an abbreviation (Sun – Sat).
dddd	Display the day as a full name (Sunday – Saturday).
ddddd	Display the date as a complete date (including day, month, and year),
dddddd	Display a date serial number as long date.
w	Display the day of the week as a number (1 for Sunday through 7 for Saturday).
ww	Display the week of the year as a number (1 – 54).
m	Display the month as a number without a leading zero (1 – 12). If m immediately follows h or hh, the minute rather than the month is displayed.
mm	Display the month as a number with a leading zero (01 – 12). If m immediately follows h or hh, the minute rather than the month is displayed.
mmm	Display the month as an abbreviation (Jan – Dec).
mmmm	Display the month as a full month name (January – December).
q	Display the quarter of the year as a number (1 – 4).
y	Display the day of the year as a number (1 – 366).
yy	Display the year as a 2-digit number (00 – 99).

yyyy	Display the year as a 4-digit number (100 – 9999).
h	Display the hour as a number without leading zeros (0 – 23).
Hh	Display the hour as a number with leading zeros (00 – 23).
N	Display the minute as a number without leading zeros (0 – 59).
Nn	Display the minute as a number with leading zeros (00 – 59).
S	Display the second as a number without leading zeros (0 – 59).
Ss	Display the second as a number with leading zeros (00 – 59).
tttt	Display a time as a complete time (including hour, minute, and second) The default time format is h:mm:ss.
AM/PM	Use the 12-hour clock and display an uppercase AM with any hour before noon; display an uppercase PM with any hour between noon and 11:59 P.M.
am/pm	Use the 12-hour clock and display a lowercase AM with any hour before noon; display a lowercase PM with any hour between noon and 11:59 P.M.
A/P	Use the 12-hour clock and display an uppercase A with any hour before noon;
a/p	Use the 12-hour clock and display a lowercase
AMPM	Use the 12-hour clock and display the AM string literal

FormatCurrency Function

Returns an expression formatted as a currency value using the currency symbol defined in the system control panel.

FormatCurrency(Expression[,NumDigitsAfterDecimal [,IncludeLeadingDigit [,UseParensForNegativeNumbers [,GroupDigits]]]])

Part	Description
<i>Expression</i>	Required. Expression to be formatted.
<i>NumDigitsAfterDecimal</i>	Optional. Numeric value indicating how many places to the right of the decimal are displayed.
<i>IncludeLeadingDigit</i>	Optional. Tristate constant that indicates whether or not a leading zero is displayed for fractional values.
<i>UseParensForNegativeNumbers</i>	Optional. Tristate constant that indicates whether or not to place negative values within parentheses.
<i>GroupDigits</i>	Optional. Tristate constant that indicates whether or not numbers are grouped using the group delimiter specified in the computer's regional settings. See Settings section for values.

Settings

The *IncludeLeadingDigit*, *UseParensForNegativeNumbers*, and *GroupDigits* arguments have the following settings:

Constant	Value	Description
TristateTrue	-1	True
TristateFalse	0	False
TristateUseDefault	-2	Use the setting from the computer's regional settings.

When one or more optional arguments are omitted, the values for omitted arguments are provided by the computer's regional settings.

The position of the currency symbol relative to the currency value is determined by the system's regional settings.

FormatDateTime Function

Returns an expression formatted as a date or time.

FormatDateTime(*Date*[,*NamedFormat*])

Part	Description
<i>Date</i>	Required. Date expression to be formatted.
<i>NamedFormat</i>	Optional. Numeric value that indicates the date/time format used. If omitted, vbGeneralDate is used.

The *NamedFormat* argument has the following settings:

Constant	Value	Description
vbGeneralDate	0	Display a date and/or time. If there is a date part, display it as a short date. If there is a time part, display it as a long time. If present, both parts are displayed.
vbLongDate	1	Display a date using the long date format specified in regional settings.
vbShortDate	2	Display a date using the short date format specified in regional settings.
vbLongTime	3	Display a time using the time format specified in regional settings.
vbShortTime	4	Display a time using the 24-hour format (hh:mm).

FormatNumber Function

Returns an expression formatted as a number.

FormatNumber(*Expression*[,*NumDigitsAfterDecimal* [,*IncludeLeadingDigit* [,*UseParensForNegativeNumbers* [,*GroupDigits*]]]])

The **FormatNumber** function syntax has these parts:

Part	Description
Expression	Required. Expression to be formatted.

NumDigitsAfterDecimal	Optional. Numeric value indicating how many places to the right of the decimal are displayed.
IncludeLeadingDigit	Optional. Tristate constant that indicates whether or not a leading zero is displayed for fractional values
UseParensForNegativeNumbers	Optional. Tristate constant that indicates whether or not to place negative values within parentheses
GroupDigits	Optional. Tristate constant that indicates whether or not numbers are grouped using the group delimiter specified in the computer's regional settings. See Settings section for values.

Settings:

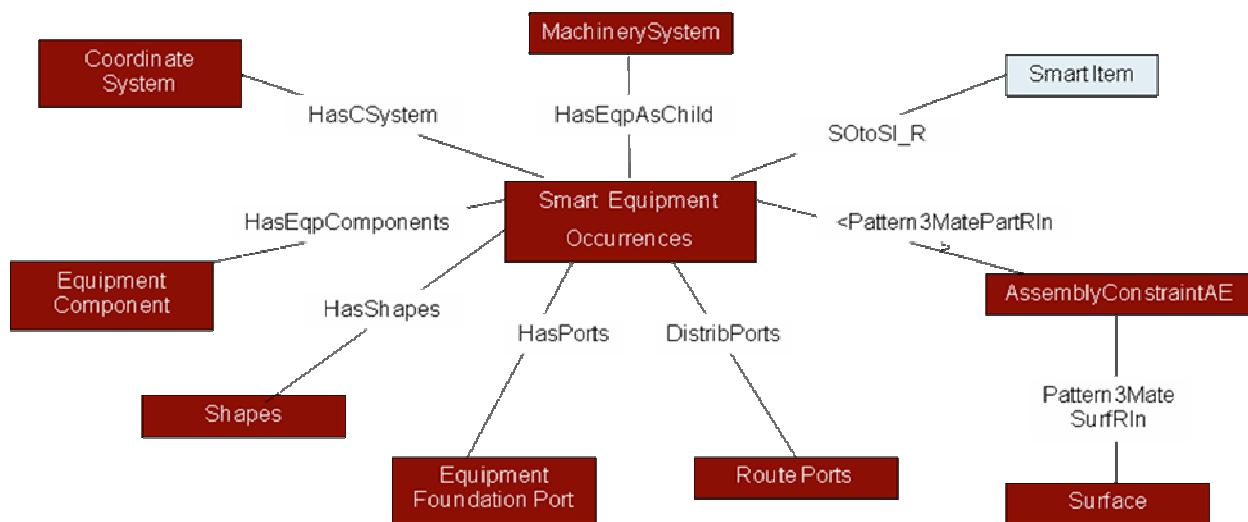
The **IncludeLeadingDigit**, **UseParensForNegativeNumbers**, and **GroupDigits** arguments have the following settings:

Constant	Value	Description
TristateTrue	-1	True
TristateFalse	0	False
TristateUseDefault	-2	Use the setting from the computer's regional settings.

Appendix C

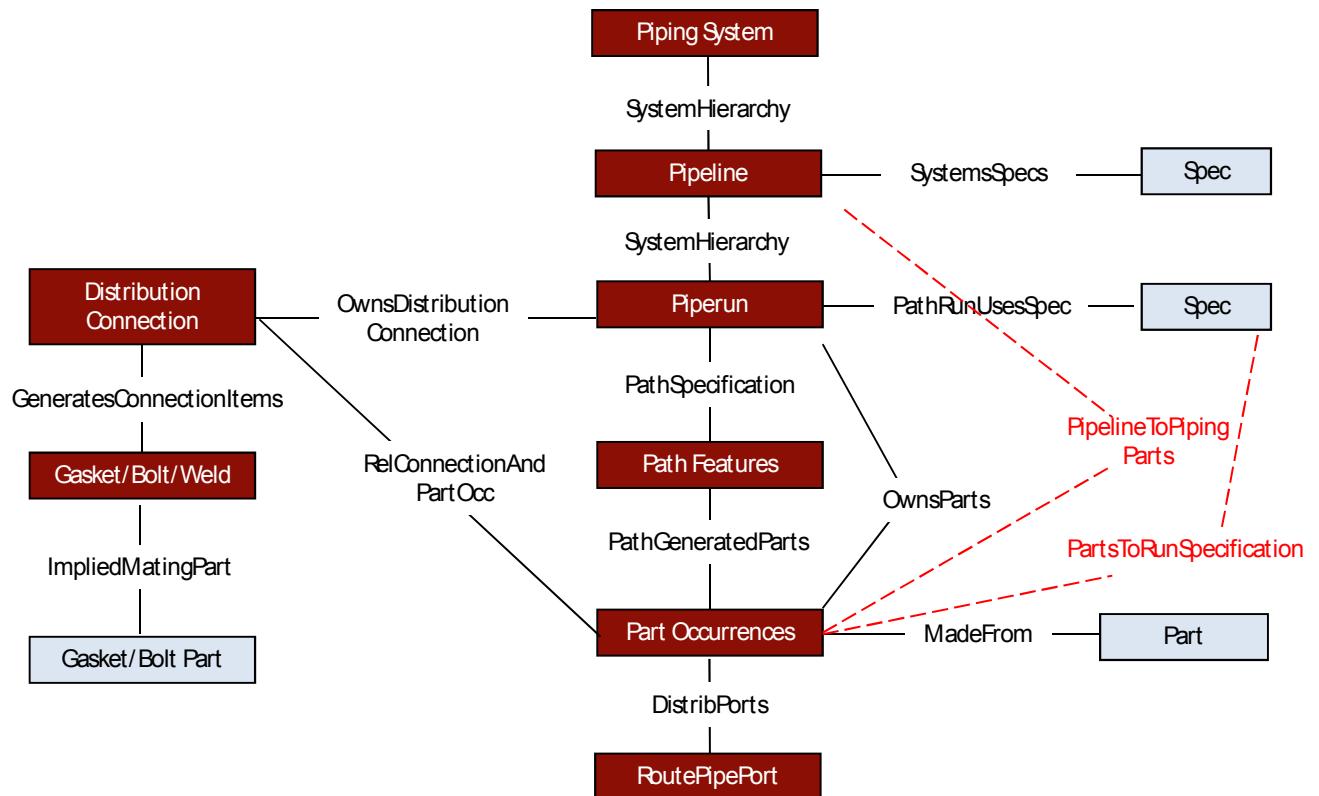
Application specific data models

Equipment



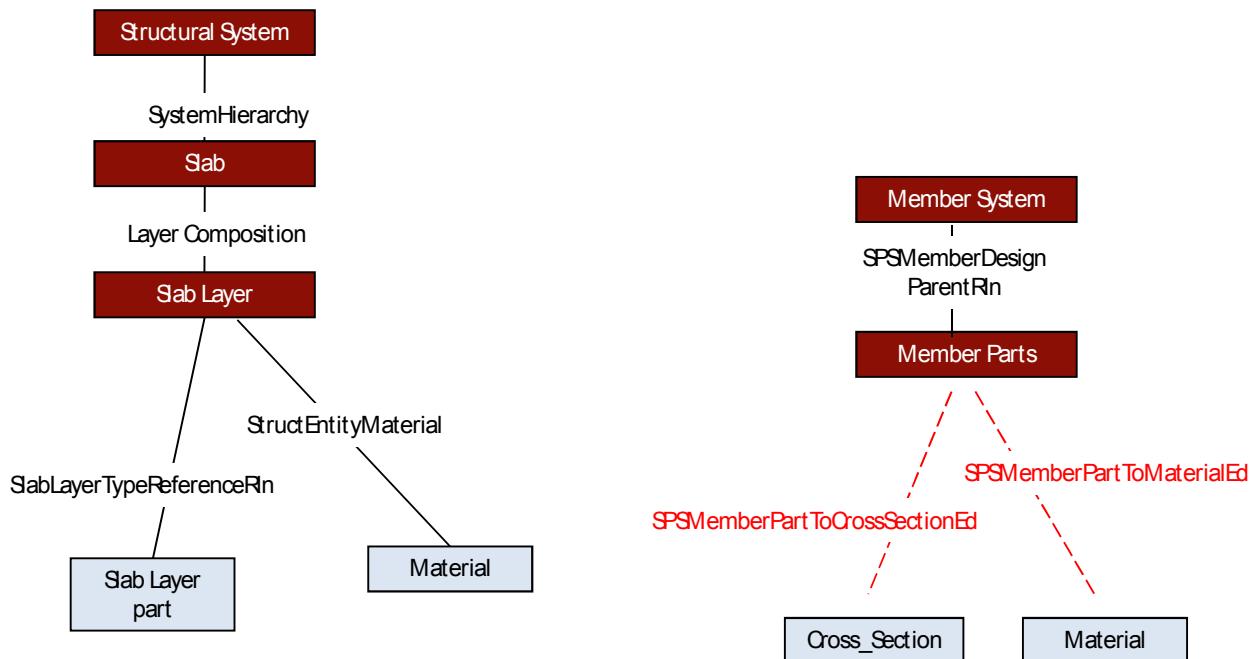
Piping

Piping Data Model



Structure

Structure Data Model



Electrical Cable

Electrical Cable Data Model

