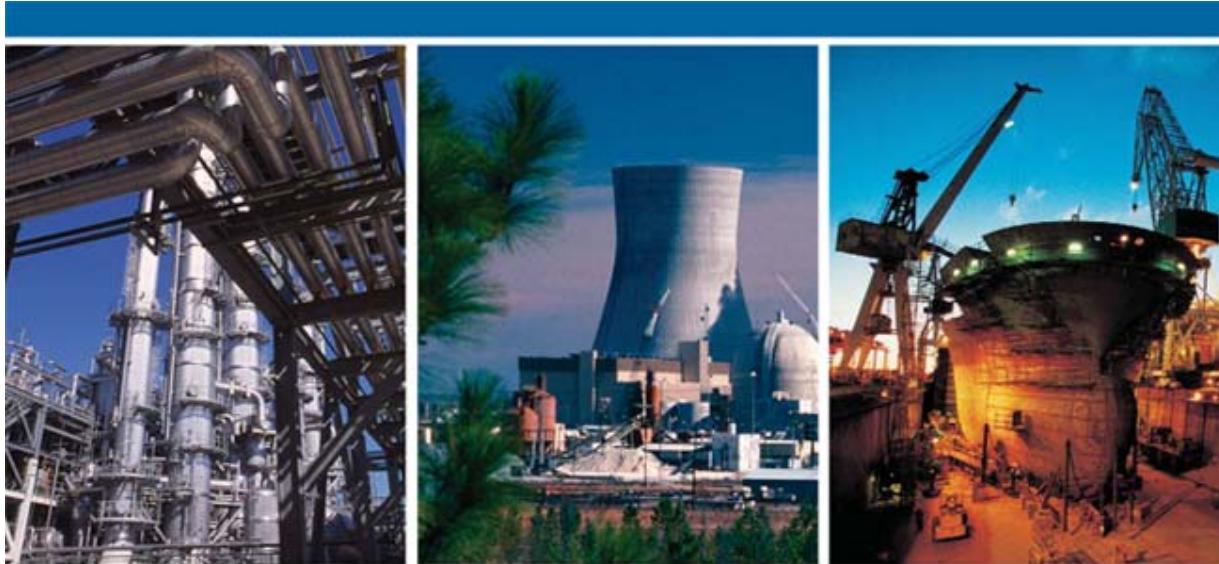


SmartPlant Foundation 2008 (4.2) Modeling and Mapping *Course Guide*



Process, Power & Marine



Version 4.2.1	January 2009	DSPF-TP-100014A
---------------	--------------	-----------------

SmartPlant Foundation 2008 (4.2) Modeling and Mapping *Course Guide*

January 2009

Version 4.2.1

Copyright

Copyright © 2002 - 2009 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Unpublished---rights reserved under the copyright laws of the United States.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, SmartPlant Foundation, SmartPlant P&ID, SmartPlant Instrumentation and INTools are registered trademarks of Intergraph Corporation. Microsoft and Windows are registered trademarks of Microsoft Corporation. Other brands and product names are trademarks of their respective owners.

Table of Contents

1. Overview of SmartPlant	1-3
1.1 SmartPlant Components	1-7
1.1.1 The SmartPlant Client	1-8
1.2 The SmartPlant Schema	1-11
1.2.1 Authoring versus Integrated Schemas	1-16
1.2.2 Components of the Schema	1-20
1.2.3 Introduction to Class Definitions	1-21
1.2.4 Introduction to Property Definitions	1-23
1.2.5 Introduction to Interface Definitions	1-25
1.2.6 Relationships	1-28
1.3 Authoring Tool Schemas	1-29
1.4 Schema Mapping	1-32
1.5 Introduction to the Schema Editor	1-34
1.6 Using the Class VM Session	1-37
2. Using the Schema Editor	2-3
2.1 Starting the Schema Editor	2-4
2.2 Viewing an Open Schema File	2-10
2.2.1 Schema Tree/Properties View	2-16
2.2.2 Schema Tree/Table View	2-21
2.2.3 Schema Tree/Drag-Drop UML View	2-23
2.2.4 Interface Relationships	2-28
2.2.5 Schema Tree/Viewable UML View	2-32
2.2.6 Editor View	2-36
2.3 Finding Schema Objects	2-38
2.3.1 View Schema Find Tab	2-38
2.3.2 Finding Objects from the Workflows Dialog	2-44
2.3.3 Exiting the SmartPlant Schema Editor	2-47
2.4 Custom Editable UML View	2-49
2.5 Change Object Display	2-61
2.6 Change Object Color	2-63
2.7 Placing Text in a View	2-65
2.8 Erasing Displayed Relationships	2-69
2.9 Open a Package	2-71
2.10 Erasing the View	2-76
2.11 Exiting the Schema Editor	2-78
2.12 Activity 1 – Using the Schema Editor	2-81
2.13 Creating a Extension Schema	2-83
2.14 Activity 2 – Creating a Extension Schema	2-95

3. Meta Schema Concepts	3-3
3.1 Model Definitions	3-5
3.2 Component Schemas	3-8
3.2.1 Properties of a Component Schema	3-10
3.3 Interactive Activity – Creating a Component Schema	3-11
3.4 Class Definitions	3-19
3.4.1 Properties of a Class Definition	3-20
3.5 Interactive Activity – Creating a Class Definition	3-22
3.6 Interface Definitions	3-29
3.6.1 Role Example	3-30
3.6.2 Interface Definitions and Relationships	3-32
3.6.3 Properties of an Interface Definition	3-35
3.7 Interactive Activity – Creating Interface Definitions	3-37
3.8 Activity 1 – Reviewing Schema Concepts	3-49
3.9 Activity 2 – Creating Objects and Relationships with Schema Editor	3-49

4. Creating Properties, Enumerated Lists, and Relationships	4-3
4.1 Property Definitions	4-4
4.1.1 Properties of a Property Definition	4-5
4.2 Property Types	4-6
4.3 Interactive Activity – Creating Property Definitions	4-7
4.4 Enumerated List Type	4-36
4.4.1 Properties of an Enumerated List Type	4-41
4.4.2 Unit of Measure List Types	4-43
4.4.3 Properties of a Unit of Measure List Type	4-45
4.5 Interactive Activity – Creating an Enumerated List Property	4-47
4.6 Relationship Definitions	4-71
4.6.1 Properties of Relationship Definitions	4-74
4.6.2 Interactive Activity – Creating Relationship Definitions	4-77
4.7 Shared Object Definitions	4-88
4.7.1 Properties of a Shared Object Definition	4-90
4.8 Activity – Creating Properties, Enumerated Lists and Relationships	4-91

5. Creating EdgeDefs, Graph Defs, View Defs and Class View Maps	5-3
 5.1 Edge Definitions	5-4
5.1.1 Properties of an Edge Definition	5-6
 5.2 Interactive Activity – Creating Edge Definitions	5-17
 5.3 Graph Definitions	5-31
5.3.1 Properties of a Graph Definition	5-33
 5.4 Interactive Activity – Creating Graph Definitions	5-35
 5.5 View Definitions	5-40
5.5.1 Properties of a View Definition	5-42
 5.6 Interactive Activity – Creating View Definitions	5-44
 5.7 Class View Maps	5-55
5.7.1 Properties of a Class View Map	5-56
5.7.2 Save the Changes to the Schema File	5-57
6. Viewing and Finding Data	6-3
 6.1 XML Files Overview	6-6
 6.2 Opening and Viewing a Data File	6-8
6.2.1 Data File Tree/UML View	6-12
6.2.2 Data Tree/Table View	6-18
6.2.3 Data Tree/Properties View	6-21
 6.3 Finding Data Objects	6-23
6.3.1 View Data Find Tab	6-23
 6.4 Activity – Viewing and Finding Data	6-29
7. Introduction to Schema Mapping	7-3
 7.1 Mapping Process Options	7-9
 7.2 Mapping Example – Extending an Enumerated List	7-11
7.2.1 Extending the Authoring Tool Enumerated List	7-12
7.2.2 Launching the CMF File	7-18
7.2.3 Synchronize the Tool Data and Map File	7-24
7.2.4 Extending an Existing Enumerate List	7-29
7.2.5 Mapping the Enumerate List	7-34
7.2.6 Saving the Schema Changes	7-39
 7.3 Planning Changes to the SmartPlant Schema	7-43
 7.4 Creating a Custom Interface for New Properties	7-46

7.5	SmartPlant Foundation Mapping Spreadsheets	7-51
7.6	Activity 1 – Extending and Mapping an Existing Enumerated List	7-65
7.7	Activity 2 – Creating the Custom Interface	7-65
8.	<i>Mapping with SmartPlant P&ID</i>	8-3
8.1	Adding Simple Properties to the Tool Meta Schema	8-6
8.2	Updating the Tool Map Schema	8-10
8.3	Extend the SmartPlant Schema	8-16
8.4	Map the New Property for Publish	8-24
8.5	Save the Schema Changes	8-28
8.6	Activity 1 – Adding and Mapping a Custom Simple Property with SmartPlant P&ID	8-31
8.7	Adding a New Select List/Enum List	8-33
8.8	Extend the Tool Map Schema	8-40
8.9	Extend the SmartPlant Schema	8-44
8.10	Map the New Property	8-48
8.11	Save the Mapping Changes	8-54
8.12	Activity 2 – Adding and Mapping a Complex Property	8-57
9.	<i>Extending the SmartPlant Database with Schema Changes</i>	9-3
9.1	Validate the Schema Changes	9-4
9.2	Extracting Schema Changes from the CMF File	9-9
9.3	Load the Changes into the SPF Database	9-13
9.4	Regenerate the Component Schemas	9-16
9.5	Testing the Schema Changes	9-17
9.5.1	Modify the P&ID Drawing Data	9-19
9.5.2	Publish the P&ID Drawing	9-21
9.5.3	Checking the Published File for New Properties	9-25
9.6	Activity – Loading and Testing the Schema Changes	9-31
10.	<i>Mapping with SmartPlant Instrumentation</i>	10-3
10.1	Modifying the Authoring Tools and Mapping the New List Values	10-4
10.1.1	Extending an Enumerated List in SmartPlant P&ID	10-5
10.1.2	Extending the SPPID Tool Map Schema	10-9
10.1.3	Extending the SmartPlant Schema and Mapping the New Value	10-13
10.1.4	Saving the Map File and Schema File Changes	10-17

10.1.5	Extending an Enumerated List in SmartPlant Instrumentation	10-20
10.1.6	Extending the SPI Tool Map Schema	10-25
10.1.7	Mapping New Enum Entries for SPI	10-30
10.2	Modifying the View Def	10-33
10.3	Save the Schema Changes	10-37
10.4	Loading the Schema Changes	10-40
10.5	Testing the SPI Mapping	10-45
10.5.1	Publishing a Change from SmartPlant P&ID	10-46
10.5.2	Retrieving a Change into SmartPlant Instrumentation	10-51
10.6	Activity 1 – Extending an Existing Enumerated List	10-61
10.7	Creating and Mapping a Custom Property in SmartPlant Instrumentation	10-63
10.7.1	Adding a Custom Property to SPI	10-64
10.7.2	Adding the Custom Field to the Instrument Index Browser	10-69
10.7.3	Mapping a Custom Field Value in SPI	10-75
10.8	Test the Complex Property Mapping for SPI	10-85
10.8.1	Retrieve the P&ID Drawing	10-86
10.8.2	Publish the Instrument Index	10-91
10.8.3	Confirm the Published Data	10-96
10.9	Activity 2 – Adding a Custom Property to SmartPlant Instrumentation	10-101

11.	Mapping with SmartPlant SPEL	11-3
11.1	Adding New Simple Properties	11-5
11.2	Updating the Tool Map Schema	11-12
11.3	Modifying the Custom Interface	11-17
11.4	Mapping the New Property	11-20
11.5	Saving Mapping Changes	11-25
11.6	Activity 1 – Adding and Mapping a Simple Property	11-29
11.7	Adding a New Select List/Enum List	11-31
11.8	Adding Enum Extensions to the Tool Schema	11-38
11.9	Mapping Enumeration List Entries	11-40
11.10	Mapping the Complex Property	11-43
11.11	Save the Mapping Changes	11-46
11.12	Test Mapped Properties	11-48
11.13	Activity 2 – Adding and Mapping a Complex Property	11-59

12. Mapping for SmartPlant 3D Overview	12-3
12.1 Adapters	12-4
12.2 Map Files	12-5
12.2.1 General Mapping Information	12-6
12.2.2 Publish Mapping	12-8
12.2.3 Retrieve Mapping	12-11
12.3 Extending the Application Schema	12-13
12.4 Loading objects into SP3D	12-18
12.5 Mapping Configuration for Publish in SP3D	12-27
12.6 Verify the SP3D Publish Mapping	12-36
12.7 Activity – Mapping SP3D Properties for Publish	12-49

C H A P T E R

1

Overview/Review of SmartPlant Integrated Engineering Architecture

1. Overview of SmartPlant

SmartPlant supports the integration of engineering authoring tools, such as SmartPlant® P&ID, SmartPlant Electrical, SmartPlant 3D, SmartPlant Instrumentation, PDS, and Aspen Zygad. This integration addresses the flow of data as it moves from one engineering application to another through its lifecycle.



Introduction of SmartPlant Modeling

SmartPlant is a software-based platform that allows for integration of a wide-variety of engineering functions that occur during the process plant life cycle, from initial concept to decommissioning.

Tantamount to SPF is the facilitation of data flow as data moves from one engineering software application (such as piping & instrumentation diagrams, electrical schematics, 3-D drawings, etc.), to another.

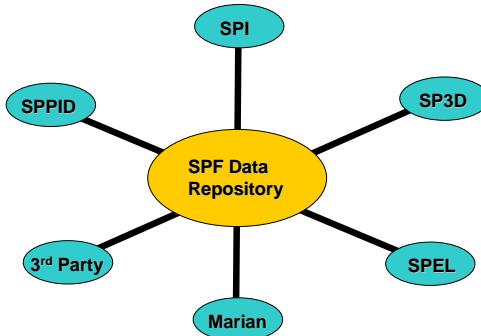
© 2005, Intergraph Corp.
All Rights Reserved.

At the center of SmartPlant is **SmartPlant Foundation**, which provides the repository for data published by the authoring tools. SmartPlant components make the exchange of data from the authoring tools to SmartPlant Foundation and back possible.



Introduction of SmartPlant Modeling

SmartPlant Foundation (SPF) is based on a "hub-and-spoke" model. An SPF data repository is at the hub and each integrated software application is a spoke.



© 2005, Intergraph Corp.
All Rights Reserved.

SPF is a "loose integration" solution that allows each software application to maintain its own data model, independent of the SmartPlant Foundation data model. The benefit of "loose integration" is the flexibility to add or remove applications for any desired configuration of SPF.



Overview of SmartPlant

The design of today's SmartPlant provides the following features:

- Transfer of engineering data from one tool to another, eliminating the manual reentry of data
- Management of change resulting from ongoing engineering in upstream applications
- Accessibility of engineering information to other collaborators without requiring the original engineering tools
- Recording of change in data as it moves through the plant lifecycle
- Correlation of shared objects from multiple authoring tools. For example, the full definition of a pump may come from multiple disciplines (electrical, mechanical, and so on), and the data comes from different authoring tools.
- Support for engineering workflows, especially versioning, approval/release, and configuration control

© 2005, Intergraph Corp.
All Rights Reserved.



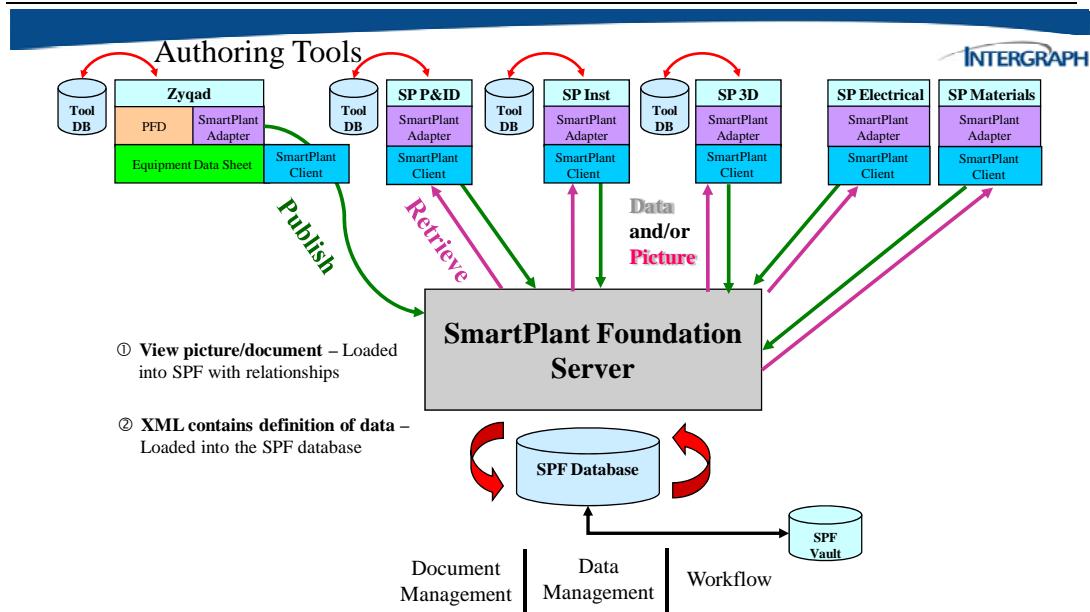
SmartPlant Architecture

SmartPlant Enterprise is functionality built on top of the **SmartPlant Foundation** database. Each tool that integrates with SmartPlant supports:

- The SmartPlant schema:** A standard data definition designed to facilitate the flow of engineering information through its lifecycle. Each tool works with XML files that are representations of this schema.
- Commands to publish, retrieve, subscribe, unsubscribe and compare between the engineering tool and SmartPlant Foundation database.**
- An adapter that allows communication with SmartPlant.** Engineering tools Zyqad, SmartPlant P&ID, SmartPlant Electrical, SmartPlant Instrumentation, and SmartPlant 3D, and PDS integrate with SmartPlant by providing an adapter.

© 2005, Intergraph Corp.
All Rights Reserved.

The following graphic shows the architecture for SmartPlant. At the heart of the system is SmartPlant Foundation (SPF) and the SmartPlant Server. Each tool has an adapter (SmartPlant Adapter) that allows for communication between the tool and the underlying SmartPlant Foundation database and vault.



SmartPlant Architecture

© 2008, Intergraph Corp.
All Rights Reserved.

1.1 SmartPlant Components

SmartPlant is comprised of the following components:

- SmartPlant Client** - An ActiveX .dll that allows the authoring tools to register with SmartPlant, connect to SmartPlant, and publish and retrieve data. After you install the SmartPlant Client on the client computer with an authoring tool and register the authoring tool project with SmartPlant, the SmartPlant Client is transparent to the user.
- Schema Component** – A suite of ActiveX components that provide functionality surrounding the creation, parsing, validation, and comparison of the SmartPlant schema and data.
- PDS Framework** - uses the PDS Framework Manager to load documents exported from PDS into the SmartPlant Foundation database.

1.1.1 The SmartPlant Client

The SmartPlant Client facilitates communication between the tool adapter and the SmartPlant Server. The SmartPlant Client is a set of components that provide the client-side services of SmartPlant. The SmartPlant common user interface is one part of the SmartPlant Client along with the data services that manage the communications between an application and the SmartPlant Server.



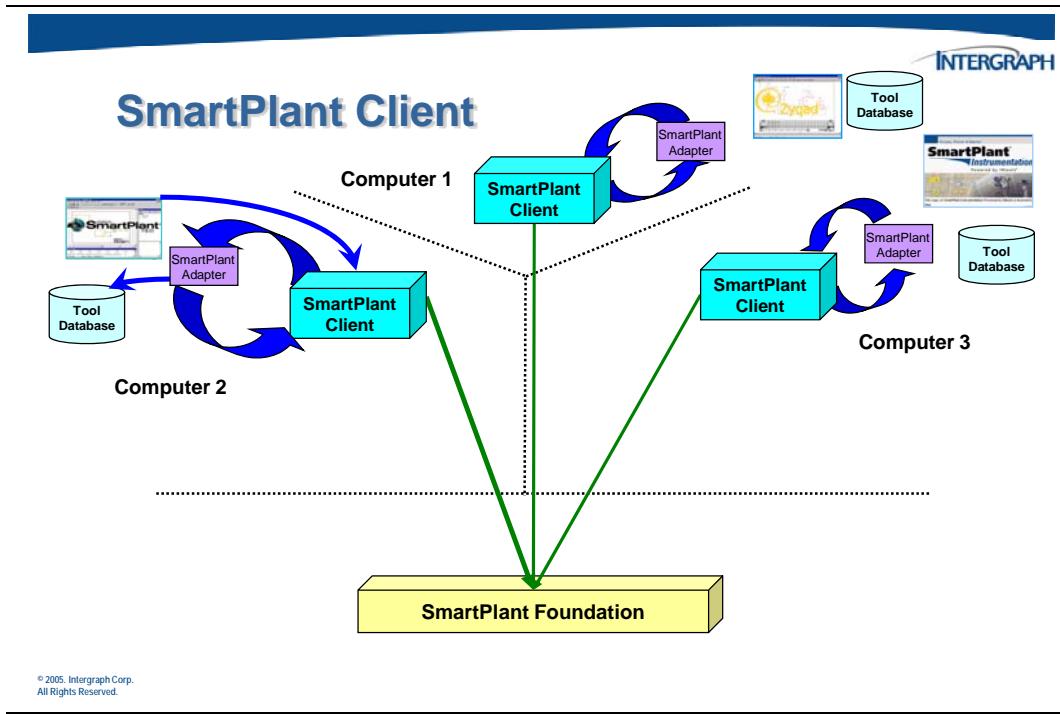
SmartPlant Client

The SmartPlant Client provides the methods for communication between the tool **adapter and the **SmartPlant Server**.**

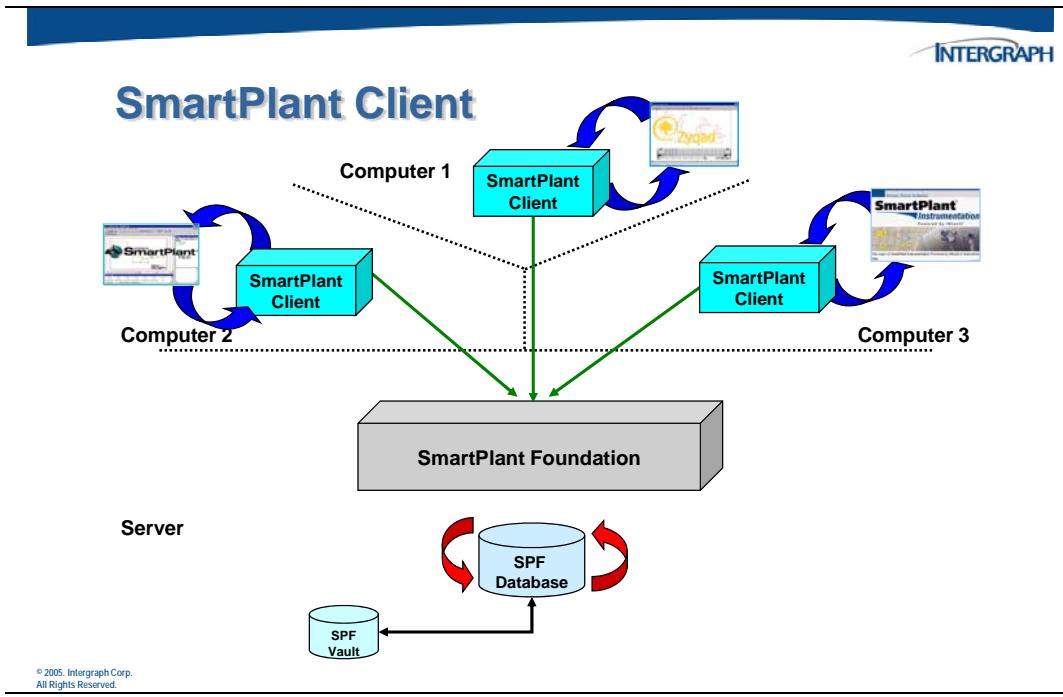
The SmartPlant Client serves three roles:

- Communicates with SPF and returns the results of that communication back to the tool adapter**
- Calls interface methods on the tool adapter to perform functions associated with the integration process**
- Provides common user interface components that tools may use to present a consistent UI across all SmartPlant-enabled applications**

Common tools, such as Zygad, SmartPlant P&ID, SmartPlant Instrumentation, PDS, and SmartPlant 3D, as well as SmartPlant Electrical and SmartPlant Materials, use the **SmartPlant Client**, the **SmartPlant Adapter**, and default configuration included in order to communicate with SPF.



The SmartPlant Client component work directly with a portion of the SmartPlant Foundation software to share information . When a publish operation occurs, the SmartPlant Foundation scheduler is responsible for loading the data from the published XML files into the SPF database. The SmartPlant Foundation software is also responsible for making sure that the files published from the authoring tool are placed in the appropriate SPF vault.



A different tool can then retrieve that same data for the next phase of design in the engineering workflow.

1.2 The SmartPlant Schema

One of the key elements to full integration is to have a "schema", and enforce some kind of rules between a "publisher" and a "retriever" of data. It should be simple for two different systems to talk about something that they have in common, e.g., a motor, a pump, a valve, etc. But it is really not that simple because each system has its own idea of what things are, and how they are used. A "schema" is simply another way of saying "a diagrammatic representation, an outline or a model."

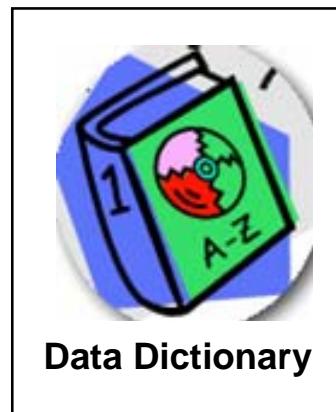
The SmartPlant schema describes the structure of data passed through SmartPlant along with its rules.



SmartPlant Schema

The SmartPlant schema, effectively SPF data model, provides the structure and semantics of the data that can be published to, and retrieved from, the SPF data repository.

**SmartPlant Foundation
Data Model**



Data Dictionary



SmartPlant Schema

Models are used to convey the essence of what you're talking about.



A model of a car isn't something that you really get into and drive to the office, but it is "real-enough" that you make the mental leap that it represents a car strongly enough that you can actually "pretend" to drive the car to work.

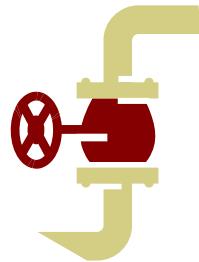
© 2005, Intergraph Corp.
All Rights Reserved.

If you think about the problem of integrating data from a lot of different software applications, you can understand how a "model" of the data that is being passed around might be a good way to simplify the discussion of "how do I get that valve from a P&ID (piping diagram) into my 3-D program?"



SmartPlant Schema

A "model of a valve" is not a valve, but it can expose many aspects of a valve, e.g., its flow-rate, its weight, its inlet size, etc.



Some other program may not care about any of that, and just wants to talk about the height of the valve, or about who carries the valve "in-stock", so that it can be ordered quickly.

© 2005, Intergraph Corp.
All Rights Reserved.



SmartPlant Schema

Each SmartPlant Enterprise software program or "tool" has some conceptual model of a valve. When we say "tool", we're talking about:

- SmartPlant Foundation** for total information management for the life of your plant.
- SmartPlant 3D** our next generation, data-centric, rule-driven engineering design solution.
- SmartPlant Electrical** an application designed by electrical engineers for electrical engineers.
- SmartPlant Instrumentation** the industry-leading instrumentation solution.

© 2005, Intergraph Corp.
All Rights Reserved.



SmartPlant Schema

SmartPlant Enterprise tools (con't)

- SmartPlant P&ID** for generating "intelligent" P&IDs and management of related engineering data for use throughout a plant's life.
- SmartPlant Markup** a precision viewing and markup tool.
- SmartPlant Review** for dynamically viewing 3D computer models for in-depth review and analysis.

No one tool cares about "all" the possible attributes of a valve.

© 2005, Intergraph Corp.
All Rights Reserved.

So how does the data about "this particular valve" get "shared" between tools? Make a "model of a valve", and share the model between the tools!



SmartPlant Schema

A "data model" is an abstract representation of the objects that software tools can share, use, and think about.

It makes it easy for each tool, because they can still think about "a valve" in their own way.

It makes it easy for all tools because they can share the concept of "this particular valve" by agreeing on what the data model for a valve is.

© 2005, Intergraph Corp.
All Rights Reserved.



SmartPlant Schema

The SmartPlant schema defines the structure of the SmartPlant Foundation (SPF) database, and is used as the neutral format through which authoring tools publish and retrieve data, independent of their specific data model.

Other functions of the SmartPlant schema include:

- **Enumerated lists:** engineering is full of lists of valid data that can be applied to data values (and shared between applications), i.e. types of valves, fluid codes, etc.
- **Connectivity -** engineering is as much about the relationships between equipment as the equipment itself, i.e. a vessel has a nozzles that connects to different pipelines.

© 2005, Intergraph Corp.
All Rights Reserved.



SmartPlant Schema

SmartPlant schema functions (con't)

- User-defined properties - Software applications provide the means for users to specify user-defined variables. Procedures must be established to assure that this data is implemented in the SmartPlant schema.**
- Support for Units-of-Measure, to allow clear flow of data regardless of the units specified.**
- Rules for evolving the schema, so changes to the structure and semantics of the data can be made due to new or changing requirements.**
- Specific support for engineering concepts such addressing process data, relationships between equipment and nozzles, and so on.**

1.2.1 Authoring versus Integrated Schemas

Starting with SPF 2008, the SmartPlant schema is actually divided into two schemas, both of which will be discussed in this class.

Both SmartPlant schemas are basically XML files, or a number of files. In an integrated environment, this file describes the structure of the XML files generated by the authoring tools in much the same way as a data dictionary describes the structure of a database. As tools publish documents (data) in XML format, those documents must adhere to the format defined by the schema to ensure that the XML data can be loaded into SmartPlant Foundation and retrieved into the other authoring tools.

The other schema is used for objects that are not part of the integrated environment. While this means objects that are not published and retrieved, that does not mean that this schema is used only in standalone environments. Many objects that are integral parts of SmartPlant Foundation are now modeled in this section of the SmartPlant schema, such as non-published documents, users, functional areas and units, and so forth.

For the sake of simplicity, this text will frequently refer, generically, to the SmartPlant schema, but to clarify, the first half of this course, the modeling chapters, will focus on the basic concepts of modeling, using the non-integrated schema. The second half of the course, the mapping chapters, will extend on the modeling concepts to illustrate, in the integrated schema, how to extend the SmartPlant schema for the purpose of sharing data between authoring tools – which can include SmartPlant Foundation.



SmartPlant Schema

The SmartPlant schema is actually divided into two schemas:

SPF / Authoring Schema

- Consists of many xml files, dividing the schema information according to the applicable layer/container**
- Managed in the Schema Editor tool using a configuration (cfg) file**

Integrated Schema

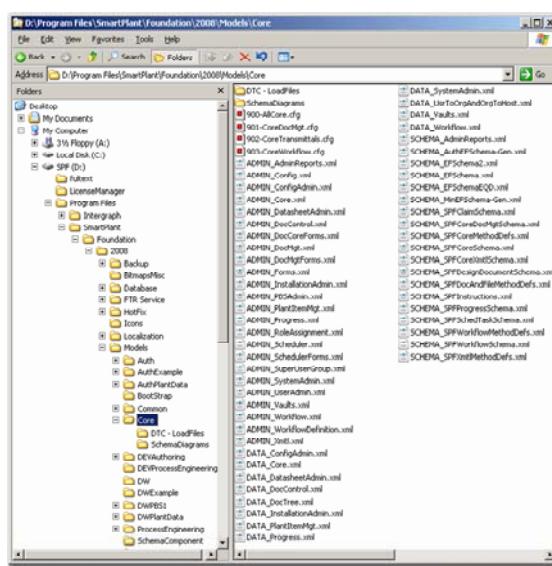
- Managed in the Schema Editor as a CMF file that is stored in, and accessed through, SmartPlant Foundation**
- Allows you to manage multiple versions of the schema to support forward/backward compatibility with authoring tools**

The files used for managing the Authoring schema are stored in the installation folder (..\SmartPlant\Foundation\2008\Models). Inside the Models directory, multiple folders separate xml files according to the layer of the data in that file. These xml files can be managed individually, or using configuration (cfg) files to group multiple xml file together. You may open the cfg files and xml files directly from within the Schema Editor application.

SmartPlant Schema

- The SmartPlant schema for *Authoring* is divided into a number of XML files, for each layer/container containing objects or data.**
- Make changes and extensions to these files for objects that will be part of SPF, but not published in an integrated environment.**

© 2005, Intergraph Corp.
All Rights Reserved.



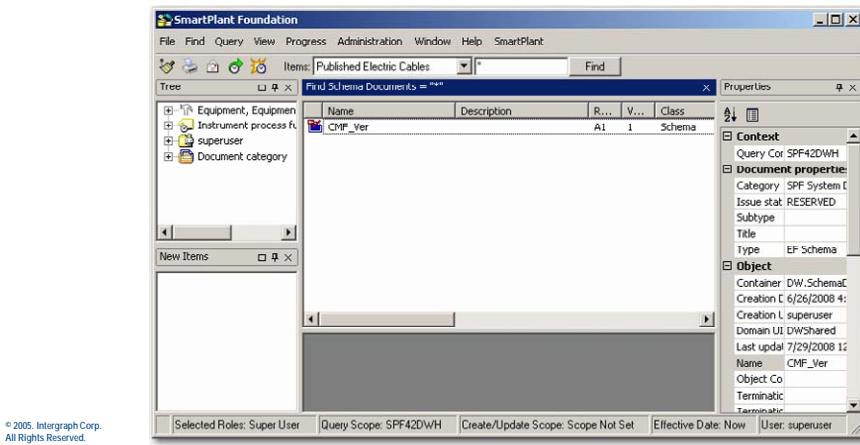
The integration schema file is managed with a CMF file. This CMF allows you to manage changes to multiple versions of the SmartPlant schema.

With SPF version 2008, SPF is compatible with authoring tools versions 2007 SP4 or later. For that reason, you will need to have multiple versions of the schema. For example, if your integrated environment includes one or more version 2007 authoring tools, you will need to have the SPF schema for versoin 2007 for mapping, but SmartPlant Foundation itself uses the 2008 version of the schema. You will need to have both on your SPF server. Using the CMF file you can write changes to multiple versions of the schema at once.

The CMF file is managed by SmartPlant Foundation. To make changes to that file, you will need to check out the file and (initially, at least) launch it from the SmartPlant Foundation application.

SmartPlant Integrated Schema

- The SmartPlant schema for **Integration** is managed using one large file – called the CMF file – that is under document management in SmartPlant Foundation.





SmartPlant Schema

Because SmartPlant is intended to facilitate data exchange from multiple engineering tools, the following rules apply to the Integration schema:

- All schema data (data definitions) is defined as part of the SmartPlant schema (CMF file)**
- The SmartPlant schema describes everything that goes on in SmartPlant**
- A copy of the SmartPlant schema resides on the server(s)**
- Component schemas are selective extracts from the SmartPlant schema, and reside on every client**
- All changes are made to the SmartPlant schema and then propagated to the component schemas**

1.2.2 Components of the Schema

Both SmartPlant schemas contain specific schema objects that will define the data that will be stored in the SmartPlant Foundation schema, regardless of where it was created, and help user view the data once it is in the database.



SmartPlant Schema

Both the Authoring and Integrated SmartPlant schemas store the following types of schema objects to define data.

SmartPlant Schema

Contains definitions of:

- Classes
- Interfaces
- Properties (attributes)
- Relationships
- Views
- Picklists
- Units of Measurement
- Navigation Paths
- Shared Objects (classes)

© 2005, Intergraph Corp.
All Rights Reserved.

The building blocks of the schema are classes, interfaces, and relationships. Classes are typically real world objects like instruments, tanks (vessels) or pipe runs.

1.2.3 Introduction to Class Definitions

Humans seem to be able to naturally classify things. If you put a big handful of coins down in front of most people, their first instinct is to sort them into groups - pennies, nickels, dimes, and so forth, going into separate groups. It is easier to think about things that are somehow grouped properly.

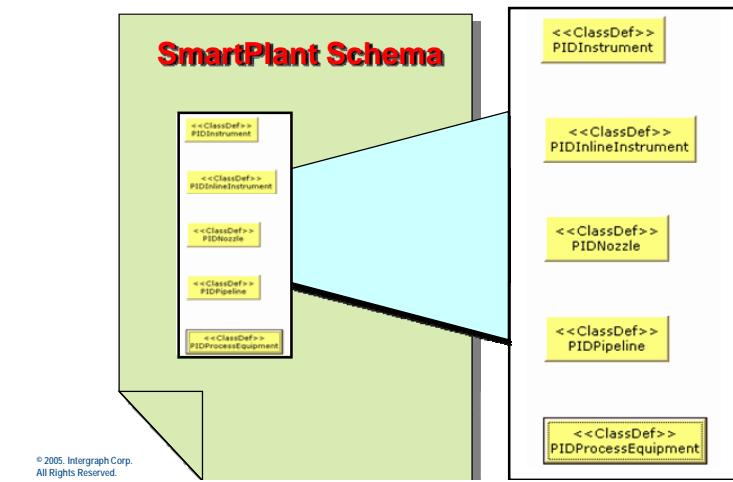
The building blocks of the SmartPlant schema are classes, interfaces, and relationships:

- Classes* typically represent real-world objects like instruments, equipment or pipe runs and are defined by Class Definitions or *ClassDef's*.
- Interfaces* are used to tightly bind “roles” and are defined by Interface Definitions or *InterfaceDef's*.
- Relationships* represent the association between two objects and are defined by Relationship Definitions or *RelDef's*.



Class Definitions

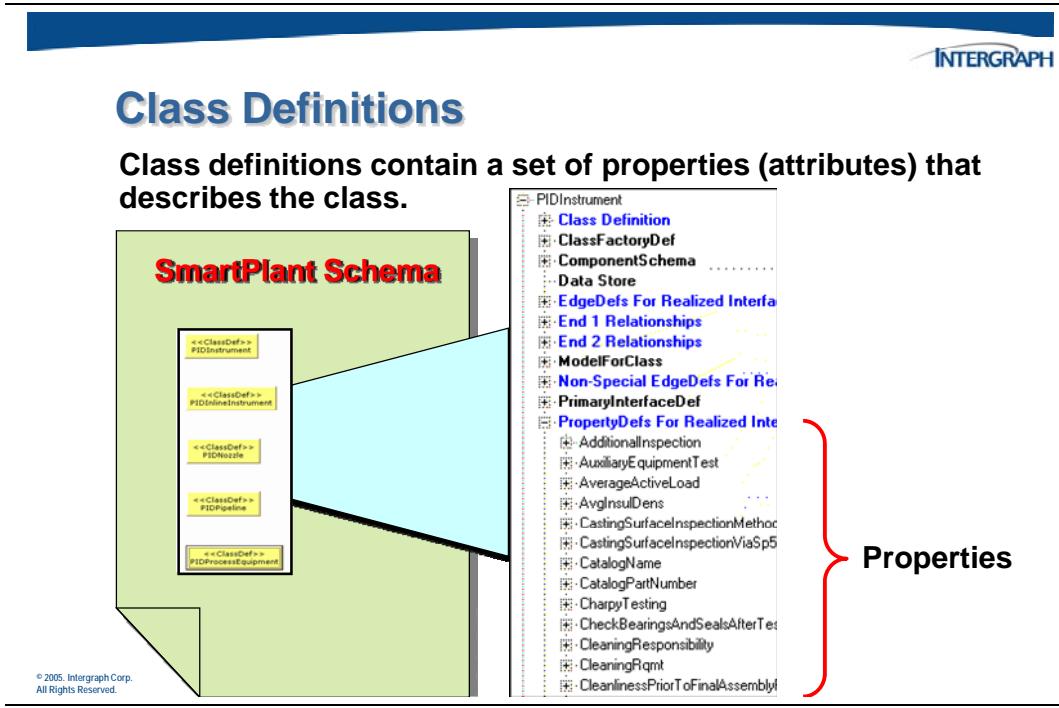
Class definitions define physical or logical types of object data.



A ClassDef is the SmartPlant schema element that will eventually become something important in the data that is “published” and “retrieved” from your software tool.

To be more precise, a ClassDef can represent physical things, such as pumps, or conceptual things, such as projects. Instances of classes become “objects” within a container (XML) that gets published by a software program, during the “publish” process. For example, an instance of a class that was defined as a ClassDef named **ELEWire** can be published, and is expected to contain data that is the electrical system’s idea about a wire.

The ClassDef also acts as a container to group a set of properties or characteristics that is used to describe the object class.



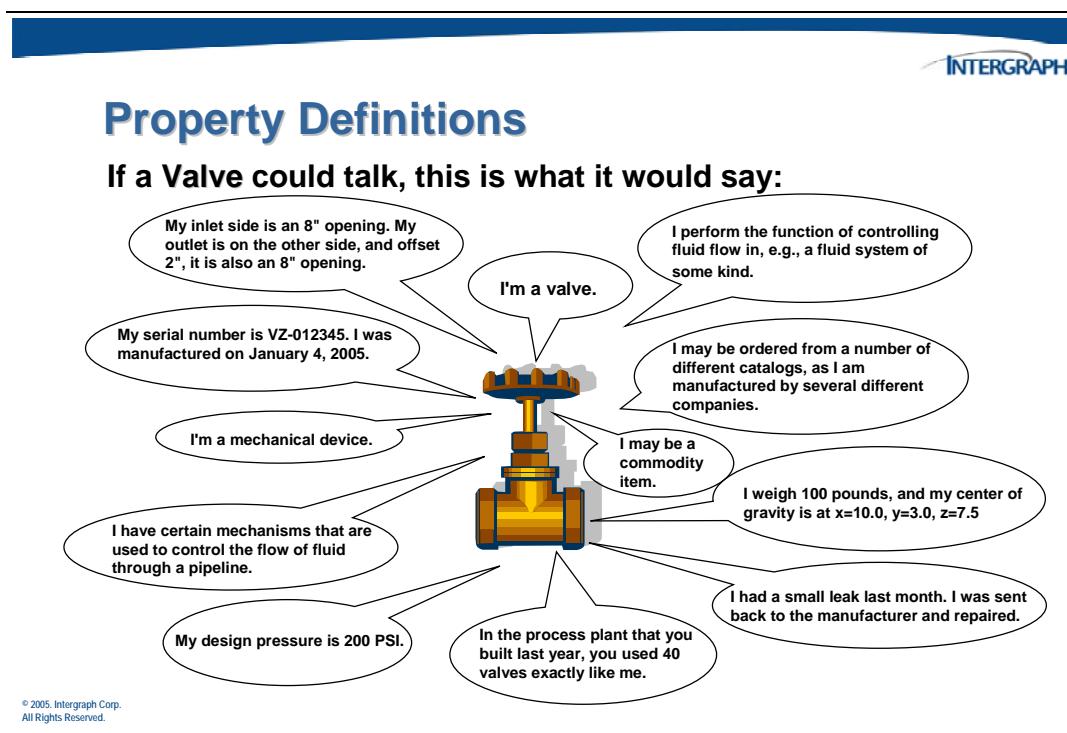
In the schema, class definitions are a named description of a set of objects that support or realize the same interface definitions and share the same property definitions and relationships.

Class definitions realize interface definitions. For example, the ***EQDProcessVessel*** class definition realizes the ***IProcessVessel*** interface definition. The ***IProcessVessel*** interface definition exposes property definitions, such as ***ProcessVesselNormalLiquidVolume***.

1.2.4 Introduction to Property Definitions

When different people talk about a "valve", what should come to mind? The problem is: everyone who thinks about a "valve" thinks about in his/her own terms. In fact, there is no one "correct" way to think about a valve, and that's OK.

How can we all think different things about a "valve", and come to some conclusions and agreements before we have fluid flowing through it? Let's try a different approach: instead of us guessing at the nomenclature of a valve, how about if we let the valve "tell us what it is", and we'll just listen carefully.

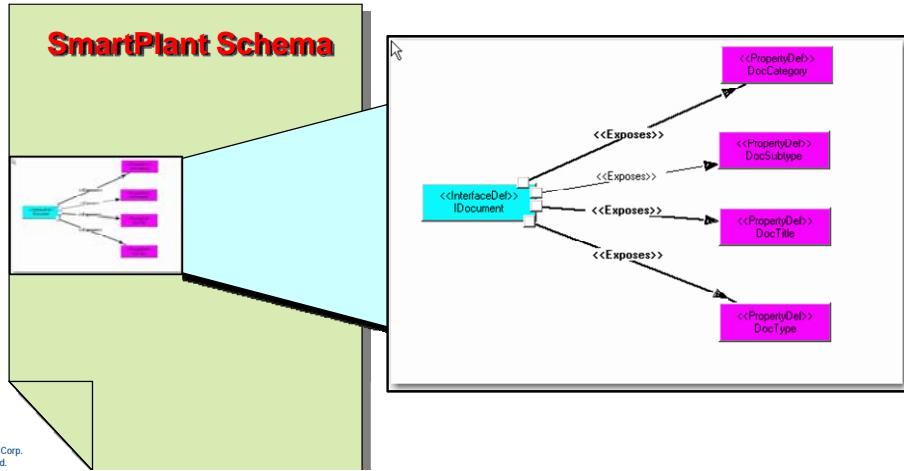


Property Definitions (PropertyDef's) are used to provide identification and naming of all object classes. PropertyDef's allow you to describe the class with respect to its position in the hierarchy and provide inheritance of information to other classes. All property definitions for a class are exposed through interface definitions and never directly by the class.



Property Definitions

A property is a characteristic or attribute used to describe a class.



The property definitions that apply to a particular interface definition are defined by the ***Expose*** relationship between objects of type InterfaceDef and objects of type PropertyDef.

For example, the ***IDocument*** interface definition exposes the following property definitions:

- DocCategory**
- DocSubType**
- DocTitle**
- DocType**

These specific properties are a set of attributes that are always used together on an object to describe the role an object can play – the role of being a document.

Property information can be used to determine the property type or, in other words, the possible values for that property definition. Standard schema property types include *Boolean*, *integer*, *double*, and *string*.

Property definitions of the enumerated property type (picklist) have a list of possible string property values defined for them. A property definition of this type must match an entry in the list of enumerated property values defined for the property type.

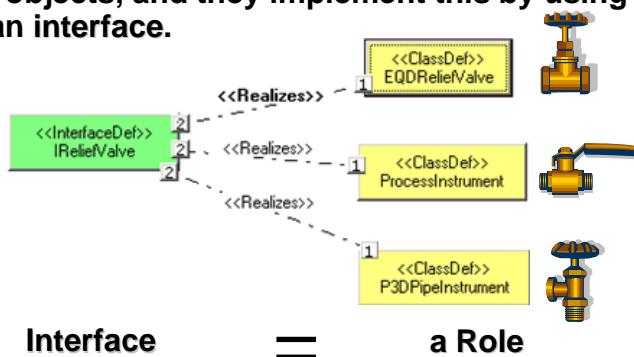
1.2.5 Introduction to Interface Definitions

The "valve" exposes certain "roles" to the universe, and the roles that a valve object "realizes" are there for us to use, if we want to listen and understand. This is called a "role-based" approach to classifying things - let them tell what they are (meaning what roles the object thinks it fulfills).



Interface Definitions

Microsoft has been a leading proponent of the "role-based" view of objects, and they implement this by using an artifact called an interface.



The use of interfaces is common in Microsoft .NET technology.

© 2005, Intergraph Corp.
All Rights Reserved.

Interfaces are just convenient "places" to group the tightly-bound properties of a role:

- ❑ A role is something that an object yields up to the universe, i.e., "I am a driver, here is my 'driver-ness'."
- ❑ A role consists of closely-related properties, such as driver's license number, expiration date, restrictions, picture.
- ❑ Properties that you purposefully keep together, such as on a driver's license, in your wallet, in your purse, in your car.
- ❑ Because a role makes data retrieval easier, and less ambiguous.

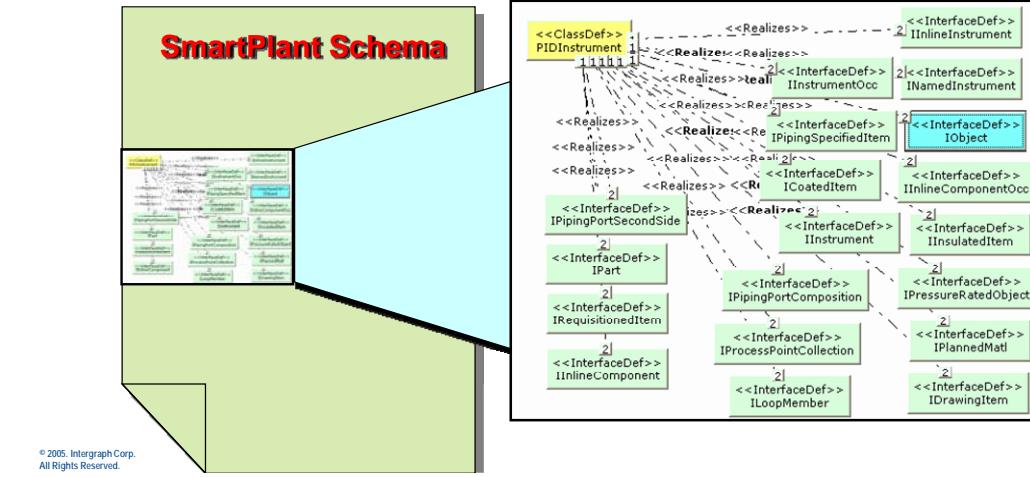
If you want to get more technical - an interface could be thought of as a named table of property definitions that are role-based.

Thus an interface definition is a named collection of property definitions. Interface definitions expose the property definitions for class definitions. Each interface definition represents a "role" for a class definition.



Interface Definitions

Interface definitions define different roles (functions) for a class definition.

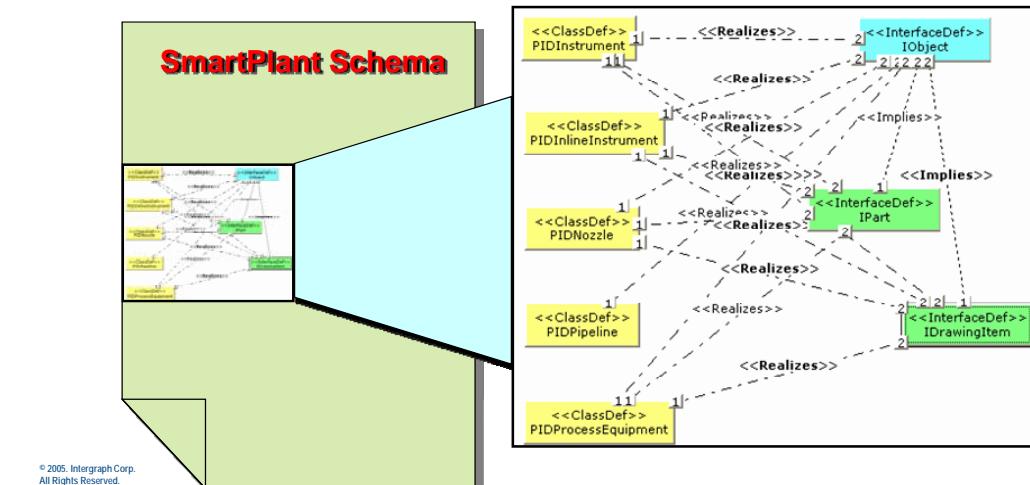


By sharing specific interface definitions, class definitions can also share property definitions, but not the data associated with the properties.



Interface Definitions

Classes can share interfaces/roles in the SmartPlant Schema.

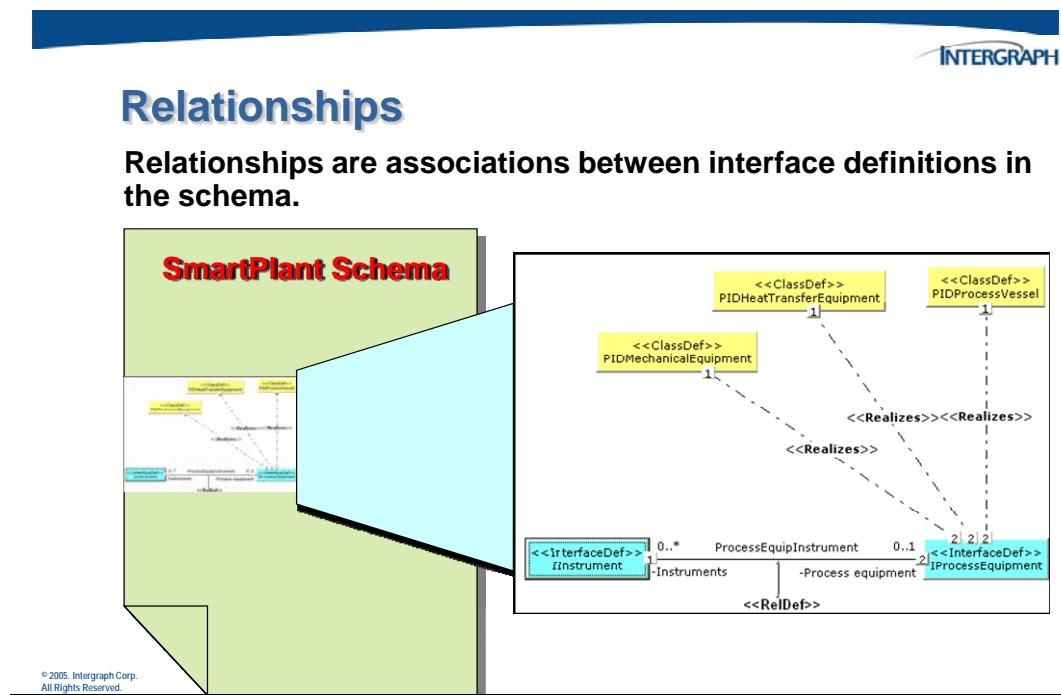


Different class definitions can share the same interface definitions, and therefore, the same role. For example, every class definition in the schema shares the **IObject** interface, which means that every class definition in the schema has the role of an

object. When a class definition has this role, it has an object name, an object description, an object identifier, and any other property definitions exposed by the IObject interface.

1.2.6 Relationships

Relationships are associations between interface definitions in the schema. They identify two specific objects that are related by a specific relationship type. The relationship type identifies the interface definitions on the two objects that fulfill the roles associated with the relationship type.

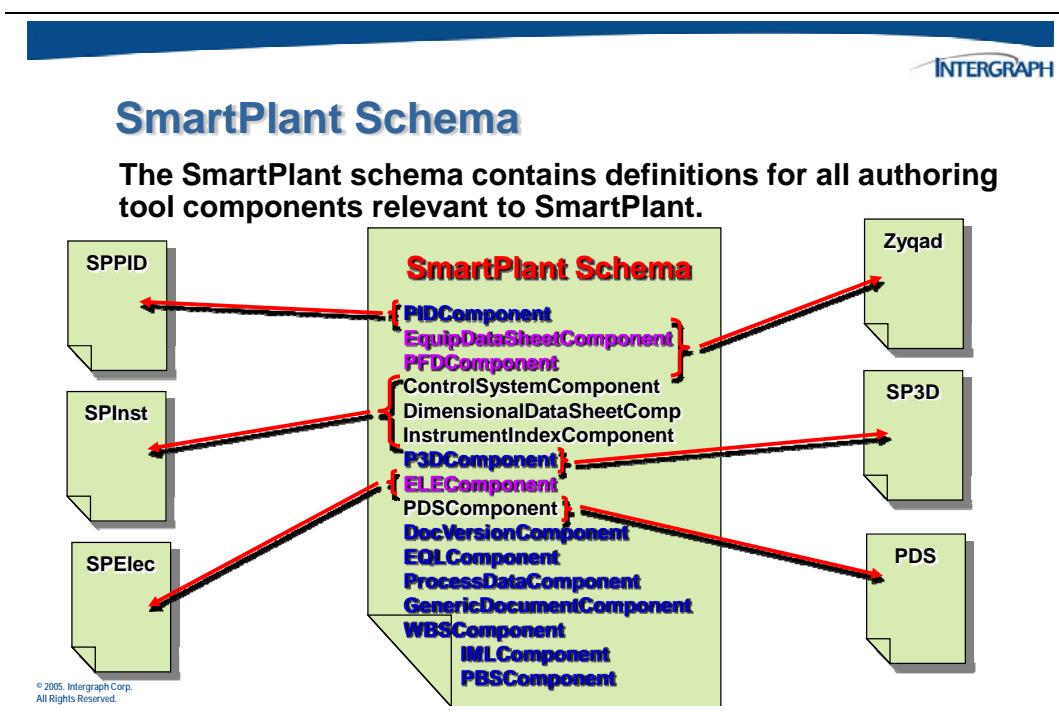


The relationship type also identifies the roles defined at both ends of the relationship. In the schema, relationship definitions are defined between interface definitions, not between class definitions. A relationship can only exist between objects that support (realize) the interface definitions at each end of the relationship definition.

1.3 Authoring Tool Schemas

The engineering application tool user, such as the SmartPlant Instrumentation (SPI) customer, does not need to understand and comprehend the SmartPlant schema to use SPF. In fact, they are able to conduct their workflows without concern for database, data model, and underlying architecture. The SmartPlant schema assures that data will be stored and retrieved in a proper fashion in accordance with the defined structure and rules.

The SmartPlant integrated schema can be hard to understand, and the full schema file will contain a great deal of information specific to only one or two authoring tools. **Schema Component** exist to break the SmartPlant schema into smaller sections that make it easier to manage, and faster for tools adapter to interpret when publishing data. The Schema Component is a set of .DLLs that assists the tools with the generation and subsequent parsing of the XML data. The tool adapter interfaces with the Schema Component (the main interface point) to read the SmartPlant schema.



The SmartPlant schema will then encompass the items and relationships for all the authoring tools in order to facilitate the transfer of information from one tool to another. This helps reduce the need to recreate data from one tool to the next.

The following defines the component schemas that encompass SPF:

- ❑ **PBS** stands for Plant Breakdown Structure. Plant breakdown is created in SPF and retrieved into the tools to create an identical structure.

- ❑ **WBS** stands for Work Breakdown Structure, and like PBS, the objects are created and published in SPF and retrieved by the tools.
- ❑ The **GenericDocumentComponent** contains the Document class and represents a very generic document.
- ❑ The **DocVersionComponent** schema is all that is required to define the data that appears in the metadata container for publish and retrieve.
- ❑ **EQLComponent** is the component schema required to support an Equipment List application.
- ❑ **ProcessDataComponent** is a component schema defining the Instrument Process Datasheet integration between SPF and SPI.
- ❑ **IMLComponent** is the component schema to support the Instrument Master List application which functions as an instrument index.

A tool schema file, also known as a map file or a tool map file, describes the structure of data as it is defined in the authoring tool database and how the authoring tool classes and properties map into and out of the SmartPlant schema.



Authoring Tool Schemas

Tool map schemas or "Map files", describe the mapping between an authoring tool's internal data model and the SmartPlant schema.

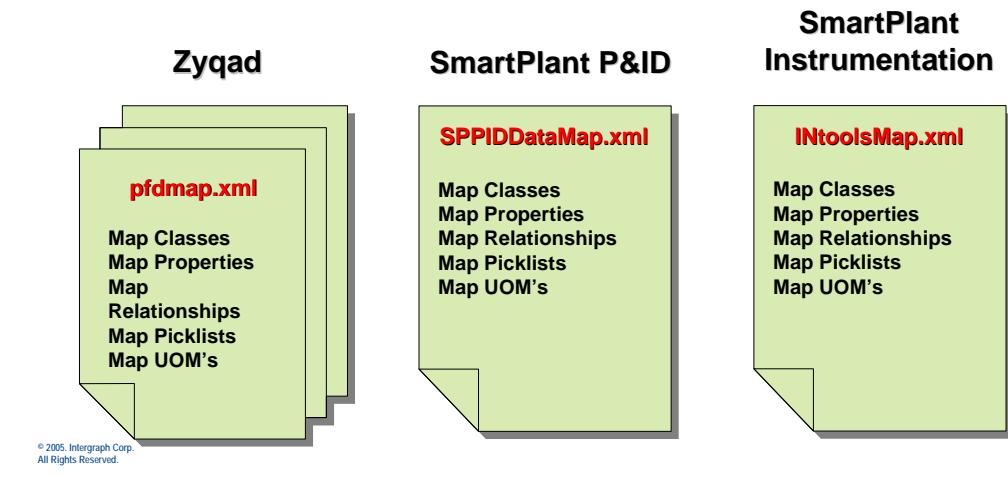
When a document* is published, the authoring tool adapter, with the help of the SmartPlant schema component API uses the tool map schema to convert the data from the tool's internal data model to the SmartPlant schema.

Note: * The word document is "overloaded" with meaning by everyone. For our purposes, we're talking about an XML file that contains data, under control of a schema.



Authoring Tool Schemas

Each authoring tool is responsible for delivering a tool schema which contains the definitions specific to that tool.



Some authoring tools, such as SmartPlant P&ID, will have one tool schema XML file while other applications such as Zyqad has several tool schemas; eqdmap.xml, pbsmap.xml, pfdmap.xml, pidmap.xml, and wbsmap.xml.



Authoring Tool Adapters

Each authoring tool that is part of SmartPlant Enterprise has an authoring tool "Adapter", which facilitates the sharing of data between the authoring tool and SmartPlant Enterprise.

Tool adapters generate XML files for "publish" operations, and consume XML files for "retrieve" operations.

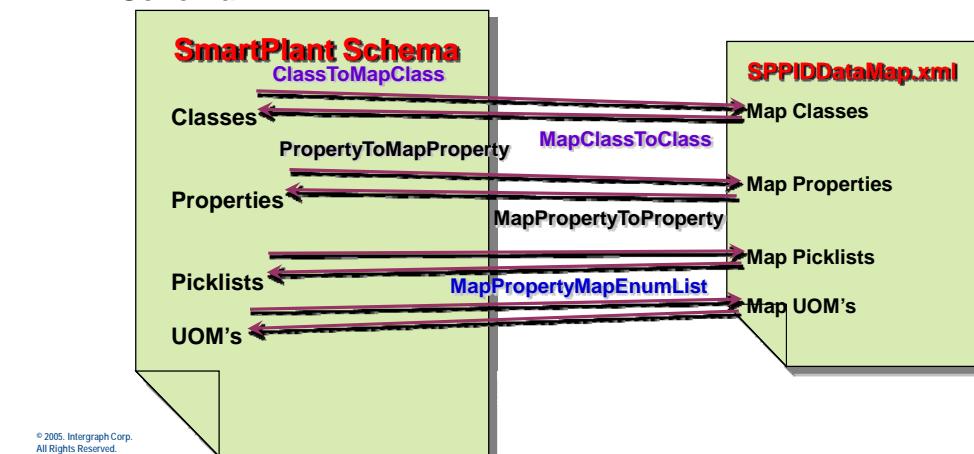
1.4 Schema Mapping

Each authoring tool that uses configurable mapping has its own schema called a tool schema. In order to make data publishing and retrieval easier, mapping is done between the SmartPlant schema and the tool schema.



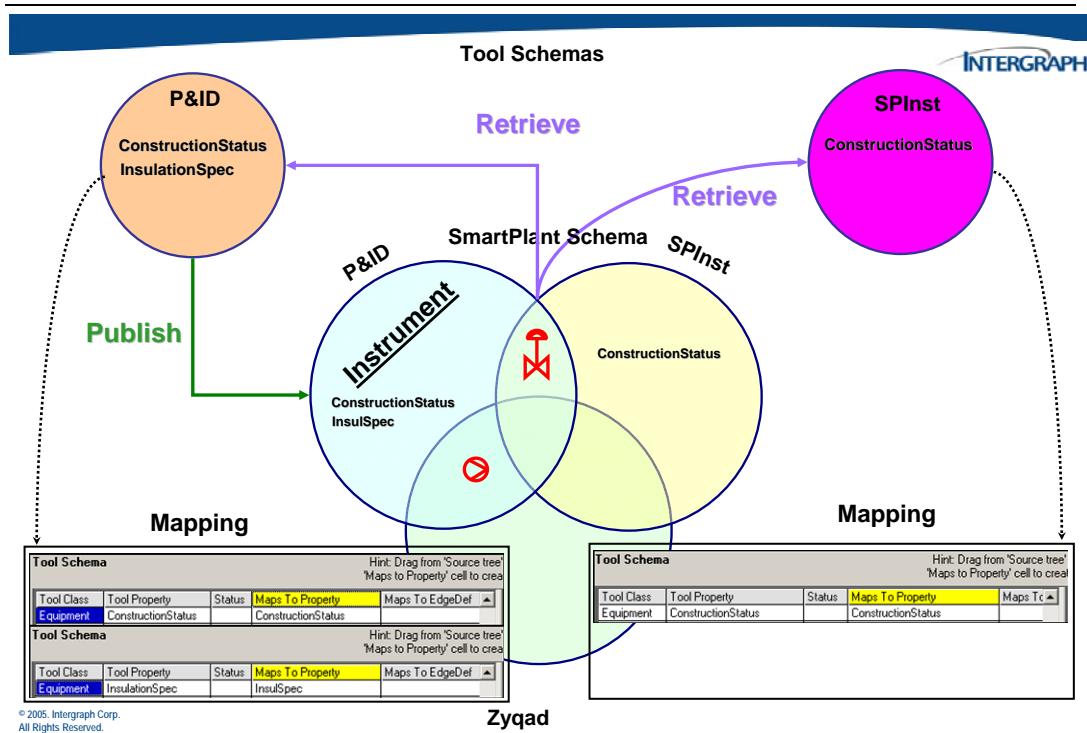
Schema Mapping

Every class in the tool schema that is to be published to SmartPlant maps to a **class**, **property**, or **picklist** in the EF Schema.



Integration with SmartPlant is done in a modular fashion with minimal impact to the application itself. Each authoring tool delivers an adapter that supports the key functionality to publish and retrieve data to SmartPlant. For tools that support configurable mapping, a **map** must be defined between the applications internal data structures and the SmartPlant schema. The Schema Component can be used to help generate the necessary XML files to exchange data and assist with many of the integration operations.

Mapping is not required, because there is some hard coding done in the tool adapter so that the adapter can publish default data. However, without mapping, data retrieval can be a big problem. If mapping is done correctly, a tool will be able to retrieve data from other tools.



If a new property is added, it needs to be mapped. Each document/container that is published has an associated component schema that describes the contents that are being published. Therefore, each document type corresponds to a component schema.

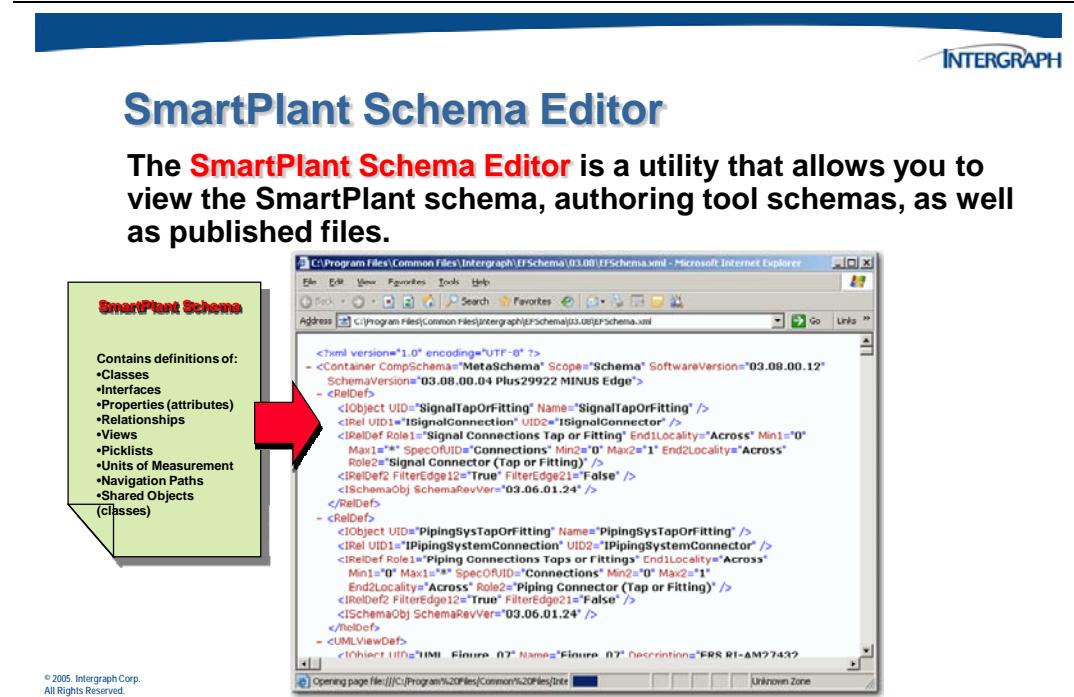
A particular tool may publish documents of one type or of multiple types. Therefore, a particular tool may use a single component or multiple components.

SmartPlant only cares about components and not about tools. No correlation exists as to what tool publishes which document types.

While tools will typically span the different components associated with a particular discipline (for example an instrumentation tool will span the various instrument and wiring document types), no rules exist that require this. As long as a tool can publish documents of the document type for at least one component, it can be a contributor to SmartPlant.

1.5 Introduction to the Schema Editor

The **SmartPlant Schema Editor** is a utility that allows you to view the SmartPlant schema, the meta schema, authoring tool schemas, and data files. This utility is especially useful for familiarizing yourself with the schema and its class definitions, interface definitions, relationship definitions, and properties.



The **SmartPlant Schema Editor** is a utility that allows you to view the SmartPlant schema, authoring tool schemas, as well as published files.

SmartPlant Schema Editor

© 2005, Intergraph Corp.
All Rights Reserved.

The **SmartPlant Schema Editor** is a utility that allows you to view the SmartPlant schema, authoring tool schemas, as well as published files.

SmartPlant Schema Editor

© 2005, Intergraph Corp.
All Rights Reserved.

Software developers can also use the Schema Editor to modify schemas, tool schemas, and data files. However, you cannot edit objects and relationships in the meta schema.

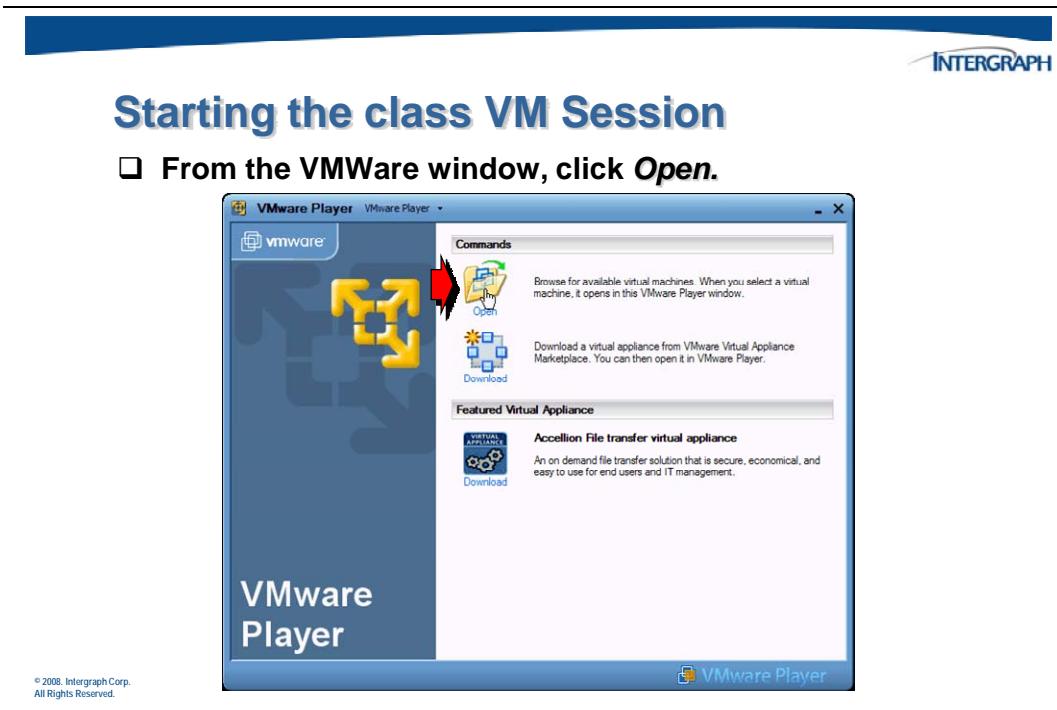
1.6 Using the Class VM Session

Your class will be using an application called VMWare Player to enable you to login and run the SmartPlant Foundation application and the class hands-on activities. This software is a virtual installation of an entire PC machine complete with the Windows 2003 Server operating system and all other necessary applications. You will find an icon on the desktop of your native class machine called *VMWare Player*.



Double click on this icon to start the VMWare application.

The *VMWare Player* window will appear.

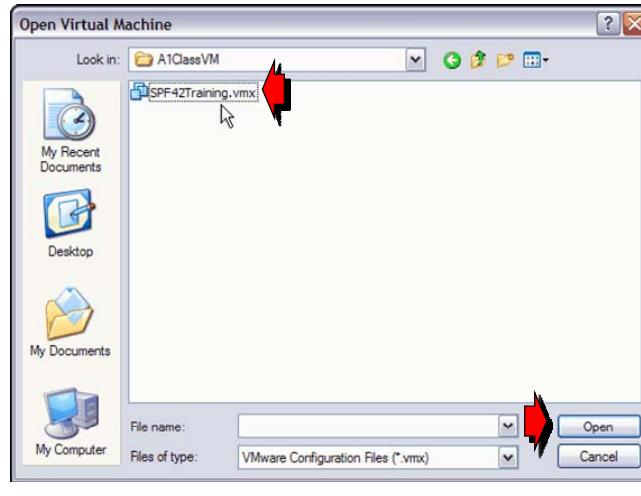


Set your open browser to the folder path specified by your instructor. Write down the path here _____.



Starting the class VM Session

- Choose the class VMWare configuration file as shown.



In the future, this session will be at the top of a list of recent VM Session, for faster access.



Starting the class VM Session

- The virtual machine boots up.

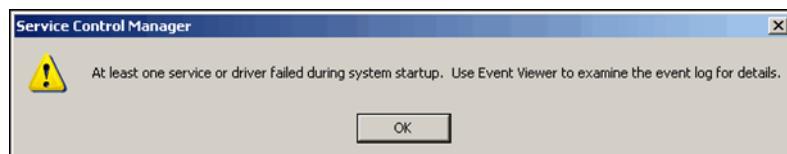


You may or may not get the next dialog displayed.

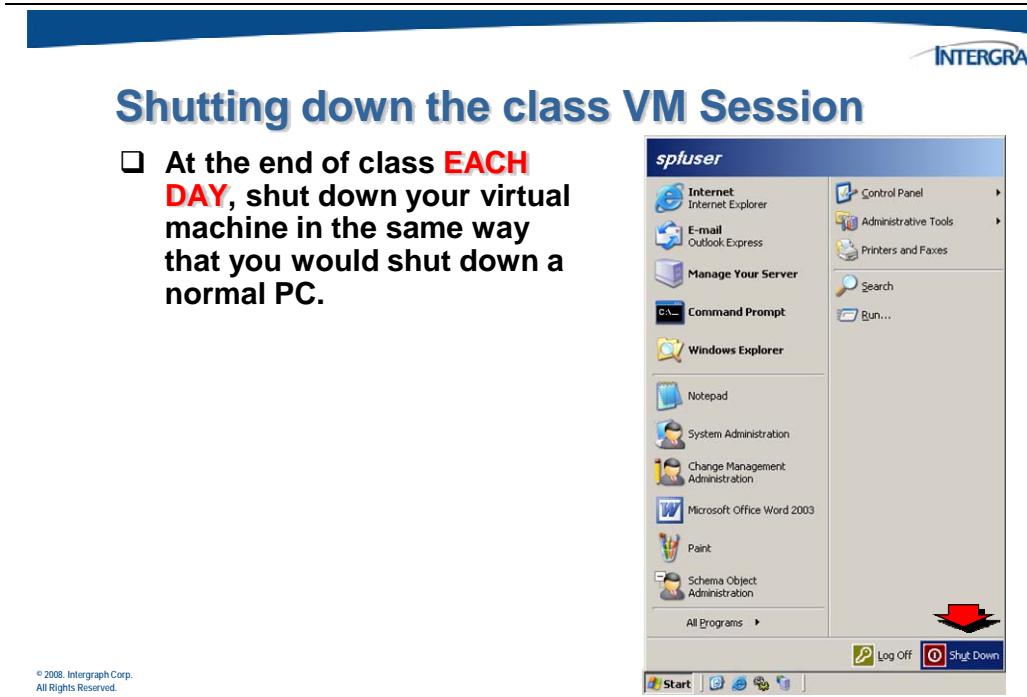


Starting the class VM Session

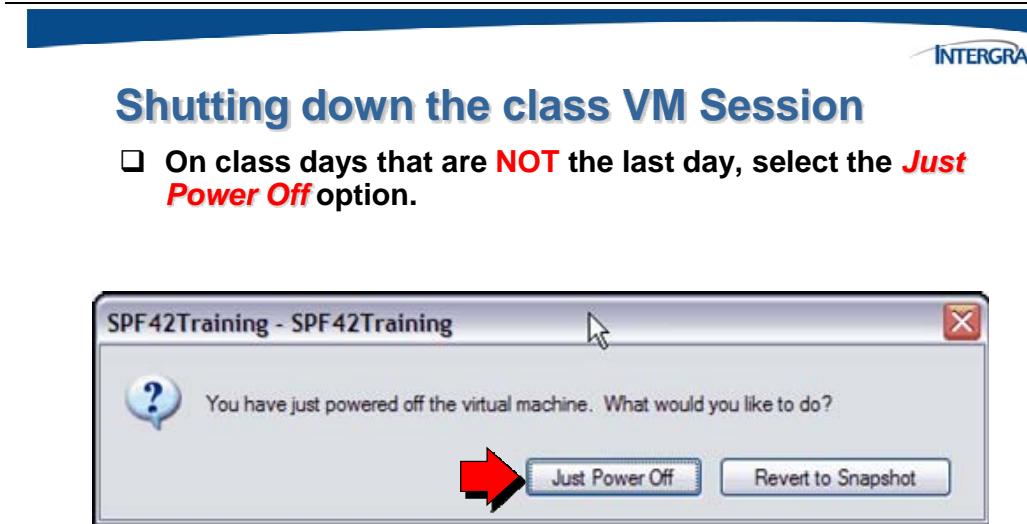
- If you get this informational dialog, click **OK**.



When you have finished with your hands-on exercises at the end of each day, please shut down your VM session to free up memory in your native machine in preparation for the next day.



You will be prompted for an option when powering off the virtual machine.

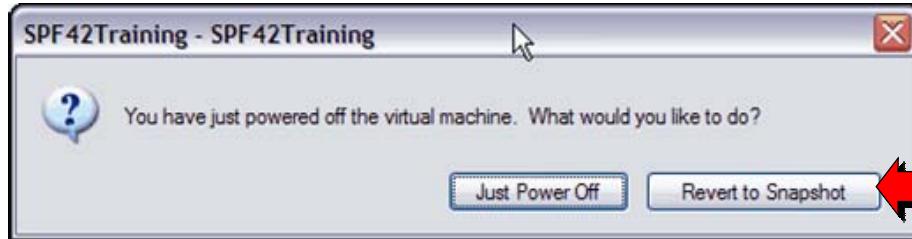


This will keep all of your work just as you left it from that day.

You will only revert to the snapshot on the last day of class when you shut down for the last time. Using the revert option will cause you to lose all of your work from the week.

Shutting down the class VM Session

- On the last class day ONLY, select the *Revert to Snapshot* option.**



C H A P T E R

2

Using the Schema Editor

2. Using the Schema Editor

The SmartPlant Schema Editor allows you to perform various tasks required for configuring the SmartPlant schema and authoring tool map schemas for integration. The Schema Editor is also useful for familiarizing yourself with the SmartPlant schema and its class definitions, interface definitions, relationship definitions, and property definitions.

Software developers can also use the Schema Editor to modify schemas, tool schemas, and data files. SmartPlant Foundation system administrators also use the Schema Editor to define the SmartPlant Foundation data model. Relationships between schema objects and other SmartPlant Foundation objects must be created in the Desktop Client, however.



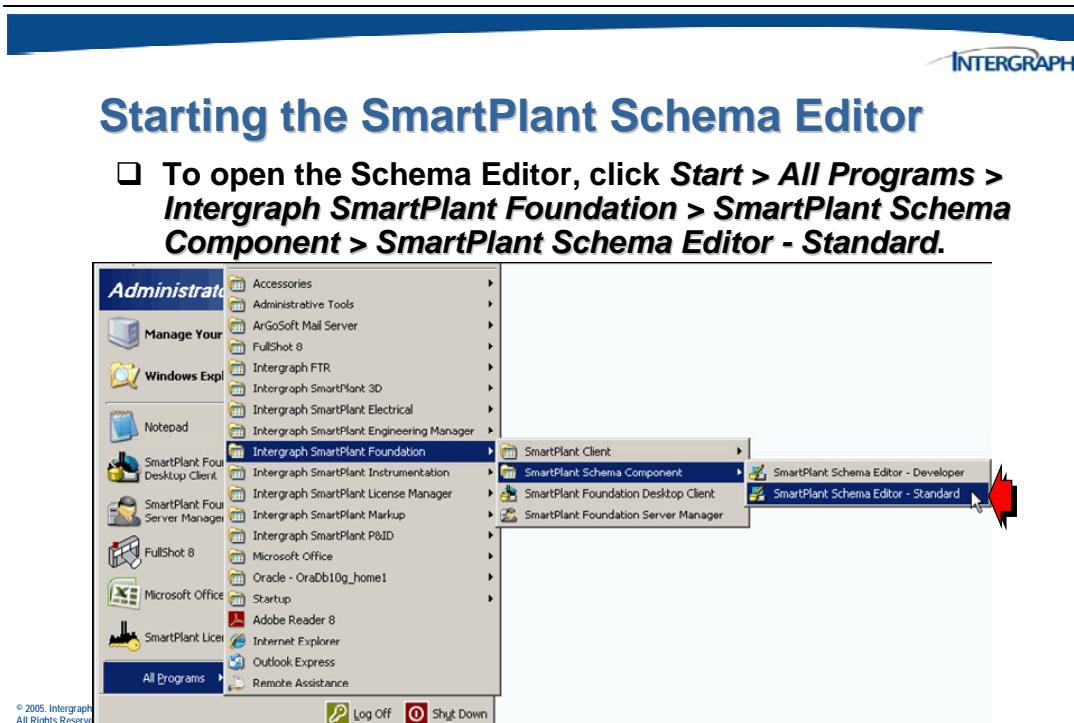
SmartPlant Schema Editor

The SmartPlant Schema Editor provides a mechanism for advanced users, usually software engineers, to perform the following types of tasks, among others:

- View the schema, meta schema, data files, and authoring tool schemas in a variety of ways**
- View mapping between the tool schemas and the SmartPlant schema**
- Compare schemas, data files, and tool schemas and view comparison instructions**
- Add new objects and relationships to the schema, tool schemas, and data files**
- Edit existing objects and relationships in the schema, tool schemas, and data files**
- Create new data files for testing**

2.1 Starting the Schema Editor

The *Schema Editor* is installed as part of the Schema Component installation. Intergraph delivers two basic Schema Editor user interfaces for you to use. The *developer* mode is the user interface that was used in previous versions of the software. The *standard* mode is a new user interface for version 2008. The *standard* mode is designed to be more user friendly and easier to navigate.



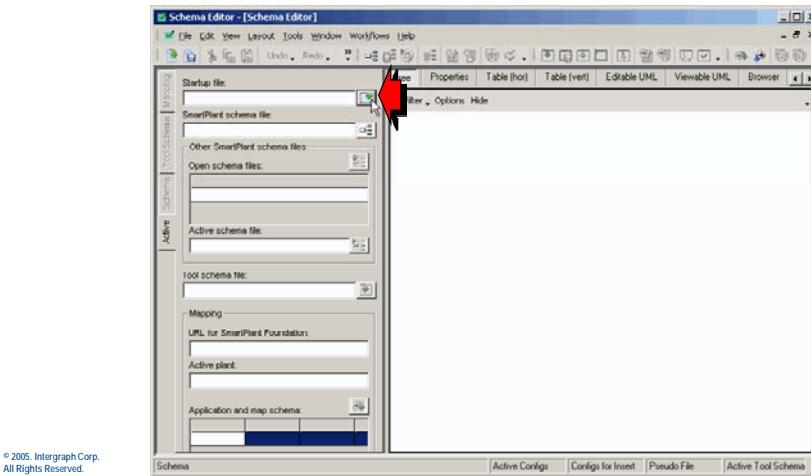
Over the following chapters, we will use both user interfaces, and you may choose which ever you are more comfortable with for regular use. Both user interfaces support all the same functions.

This interface includes such features as viewing and modifying the schema and defining mapping between authoring tools.

To see the Schema Editor in action, we must first open a file to navigate. Click the button beside the *Startup file* field.

Starting the SmartPlant Schema Editor

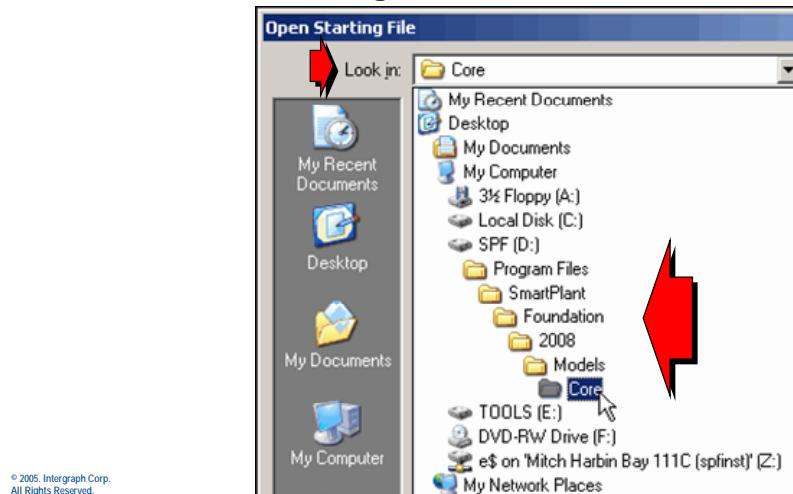
- From the top of the Schema Editor window, click the open **Startup file** button to open a schema configuration file.



Browse to the XML, CFG, CMF, or session file to open. In the illustration below, we are opening the **D:\Program Files\SmartPlant\Foundation\2008\Models\Core** folder.

Starting the SmartPlant Schema Editor

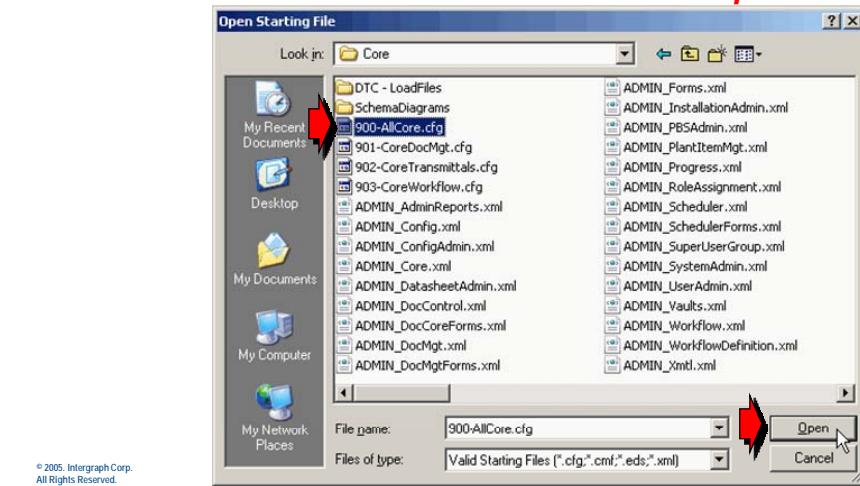
- Set the browser window to the location for a the list of available configuration files.



A configuration file (CFG) can open multiple schemas at once and sets the active file.

Starting the SmartPlant Schema Editor

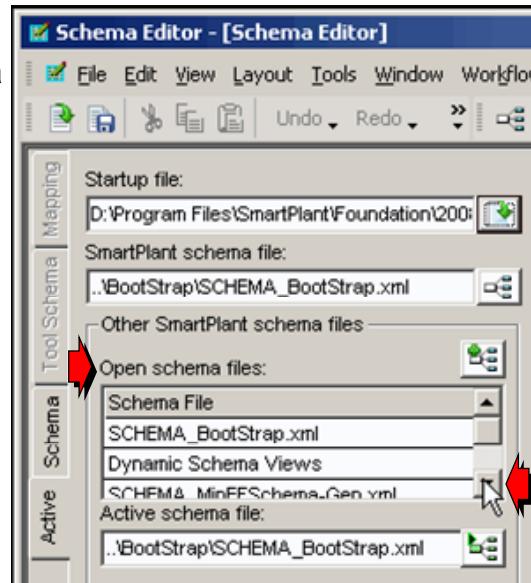
- Click the file (**900-AllCore.cfg**) to use to open with the SmartPlant Schema Editor. Click the **Open** button.



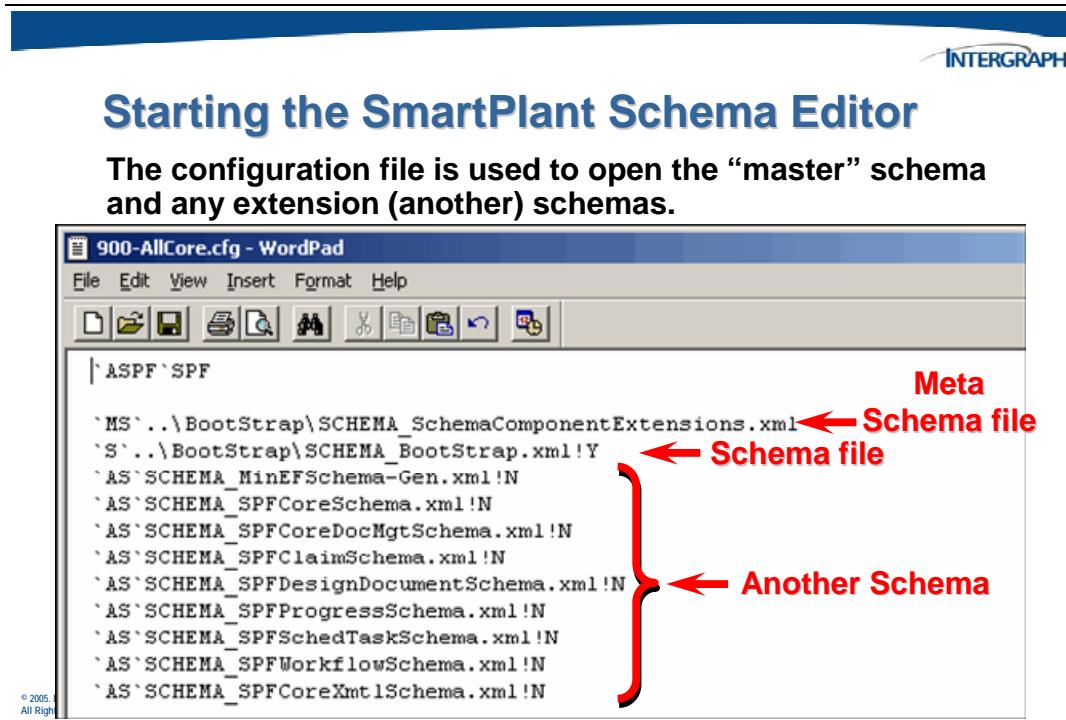
The *Active* tab allows you to view a list of the open files, as well as change the active schema or open additional files.

Starting the SmartPlant Schema Editor

- Use the scroll bar to view the opened schema files.



The **900-AllCore.cfg** file is the delivered schema configuration containing all of the SmartPlant Foundation definitions. Extension schemas are used to add additional definitions to the base SmartPlant Foundation definitions in **SCHEMA_BootStrap.xml**.



The **SCHEMA_BootStrap.xml** file defines the minimum SmartPlant Foundation configuration to enable a user to launch the Desktop Client.

The SmartPlant schema is represented by a Unified Modeling Language (UML) model entitled SmartPlant schema. A solid understanding of UML is needed to completely comprehend and digest the complexities of the SmartPlant schema.



Unified Modeling Language

Data modelers have agreed to use symbols that are part of a set of modeling practices called Unified Modeling Language, or UML .

UML is used as a symbolic framework upon which the ideas that are being modeled are expressed.

Even an experienced data modeler needs to know what symbols represent what ideas.

The precise nature of the symbols (and symbolic logic, in general) virtually guarantees that ambiguity is minimized, and accuracy is maximized.

© 2006, Intergraph Corp.
All Rights Reserved.

There were many attempts during the last 20 years to arrive at a "common" understanding of how to represent ideas on a diagram. UML won the war because it is so easy to use, and is extensible (for new ideas that might come along at any time).

In the *Schema Editor*, you can view any schema in .XML format. The many different schema views in the Schema Editor provide different types of information that help you understand how the schema works. For example, when you view the schema in any of the tree views (except the schema tree/UML view and the schema hierarchy views) you can typically see the schema class definitions.

As you expand nodes in the tree views, you can navigate the relationships in the schema and view **interface definitions**, **property definitions**, and even instances of the class if you also have a data file open in the Schema Editor.

The Schema Editor also provides graphical views of the schema. These views allow you to see both static and dynamic UML diagrams containing the classes, relationships, interfaces, and properties defined in the schema.

There are many other ways to view the schema in the Schema Editor depending on what aspect of the schema that you want to understand.



Viewing a Schema File

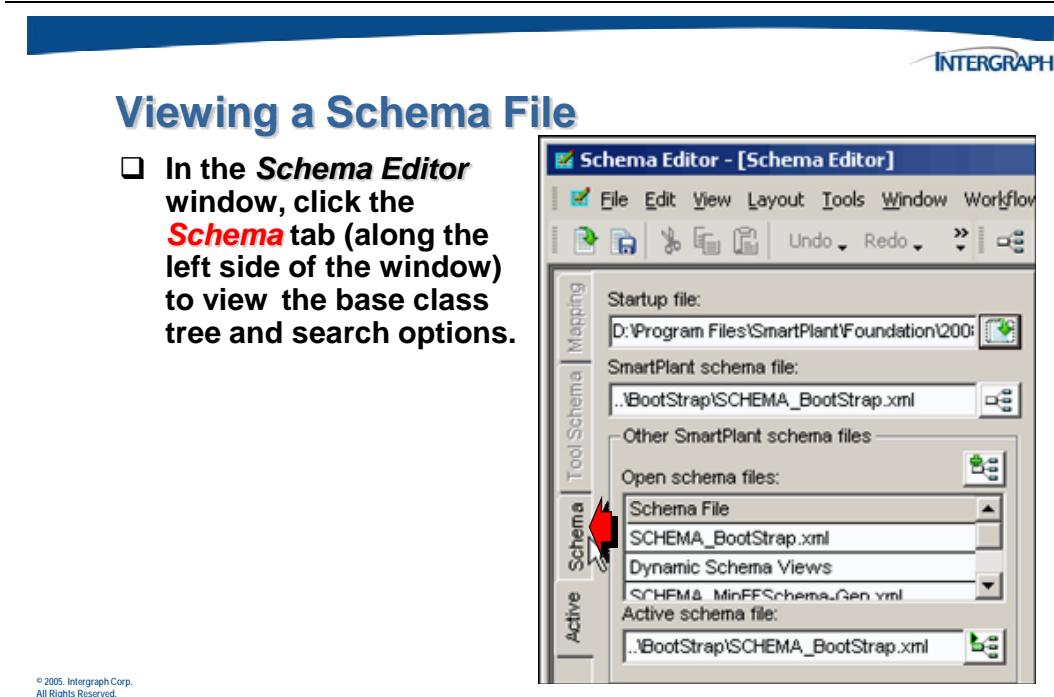
The **Schema Editor** provides graphical views of the schema. These views allow you to see both static and dynamic UML diagrams containing the **classes**, **relationships**, **interfaces**, and **properties** defined in the schema.

Some of the most useful ways to view the schema in the Schema Editor are the following:

- Using the **Tree**, **Tree/Table**, and **Tree/Properties** views
- Using the **Tree/Viewable UML** graphical view
- Using the **Editor** view
- Using the **Editable UML** views
- Using the **Tree/MultiTab** view

2.2 Viewing an Open Schema File

Before you can begin viewing the schema or data files, you must first open the schema in the Schema Editor.



Shown below are some short explanations for the tabs visible in the *Standard Workflows* window:

Active Tab

The **Active** tab allows you to select a startup file and view related files. You can also open tool schema files, connect to SmartPlant Foundation for mapping, and synchronize tool map schemas and tool metadata for tools with metadata adapters. You can also change the active schema and open tool schemas from the Active tab.

Schema Tab

The **Schema** tab allows you to search the active schema for specific objects or browse the schema in the tree view. When you select an object in the tree view, information for that object is displayed on the active tab on the right side of the window.

Tool Schema Tab

The **Tool Schema** tab allows you to search the active tool schema for specific objects or browse the schema in the tree view. When you select an object in the tree view, information for that object is displayed on the active tab on the right side of the window.

Mapping Tab

The **Mapping** tab allows you to view loaded tool map schemas, expand them, and select tool map schema objects for which you want to define mapping.

Schema Editor toolbars provide easy access to the most commonly used commands. You can move the toolbars around in the Schema Editor window and dock them wherever you want. For descriptions of the commands on the Schema Editor toolbars, see the *SmartPlant Schema Editor User's Guide* (DSPF1-PE-200008I-Updated).

Command buttons on the Schema Editor vertical tabs allow you to identify commands easily with icons.



Schema Editor Icons/Toolbar

Click



To

Click this to specify the startup file for the Schema Editor, such as a session file, a CFG file, a CMF file, or an XML file.



Click this to select a schema file to open. If the selected startup file contains a schema file, the name and path for that schema file is displayed in the SmartPlant schema file box.



Click this to select another schema file to open.



Click this to select the active schema file when multiple schema files are open in the Schema Editor. The active schema is the one that you are editing and searching. Although you can see multiple schemas at once, you can only edit one at a time.



Schema Editor Icons/Toolbar

Click **To** (cont.)



Click this to search for schema objects or tool map schema objects that match the criteria you define on the corresponding tab.

Note: For more details on the *Standard Workflow* window, see the SmartPlant Schema Editor User's Guide (DSPF1-PE-200008I-Updated)

© 2005, Intergraph Corp.
All Rights Reserved.

Horizontal tabs across the top of the *Standard Workflows* window provide easy access to different views of schema, tool schema, and mapping. To change the view, click a tab. The context of the tabs change depending on your selections on the vertical tabs on the left of the *Standard Workflows* window, and the selection from the Tree in the *Schema* tab.



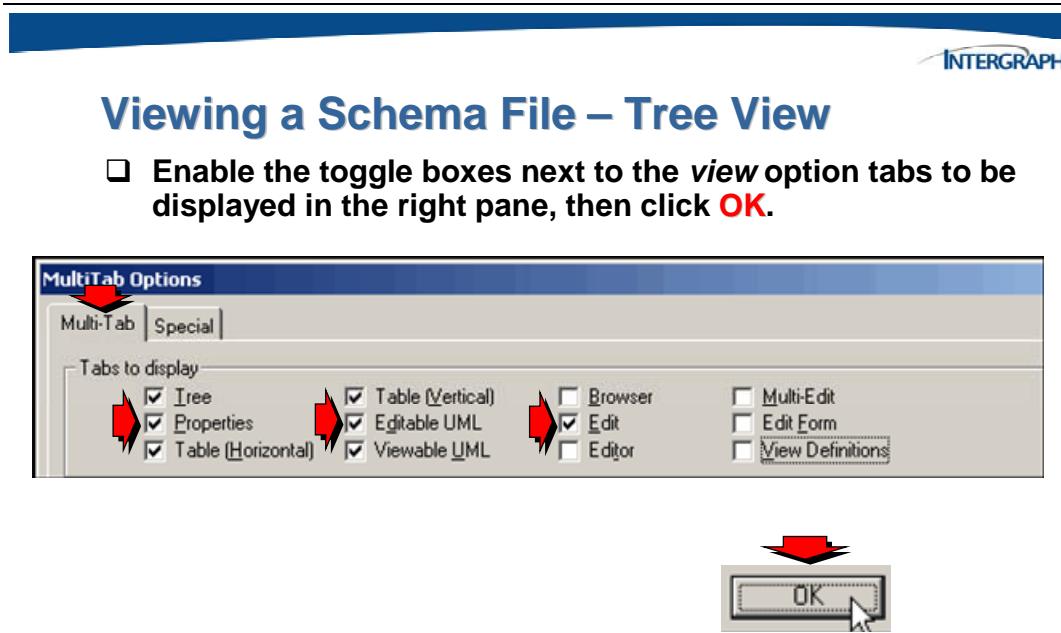
Viewing a Schema File – Tree View

- Right-click on one of the displayed tabs and select ***Update MultiTab Options*** from the dynamic menu that is displayed.



© 2005, Intergraph Corp.
All Rights Reserved.

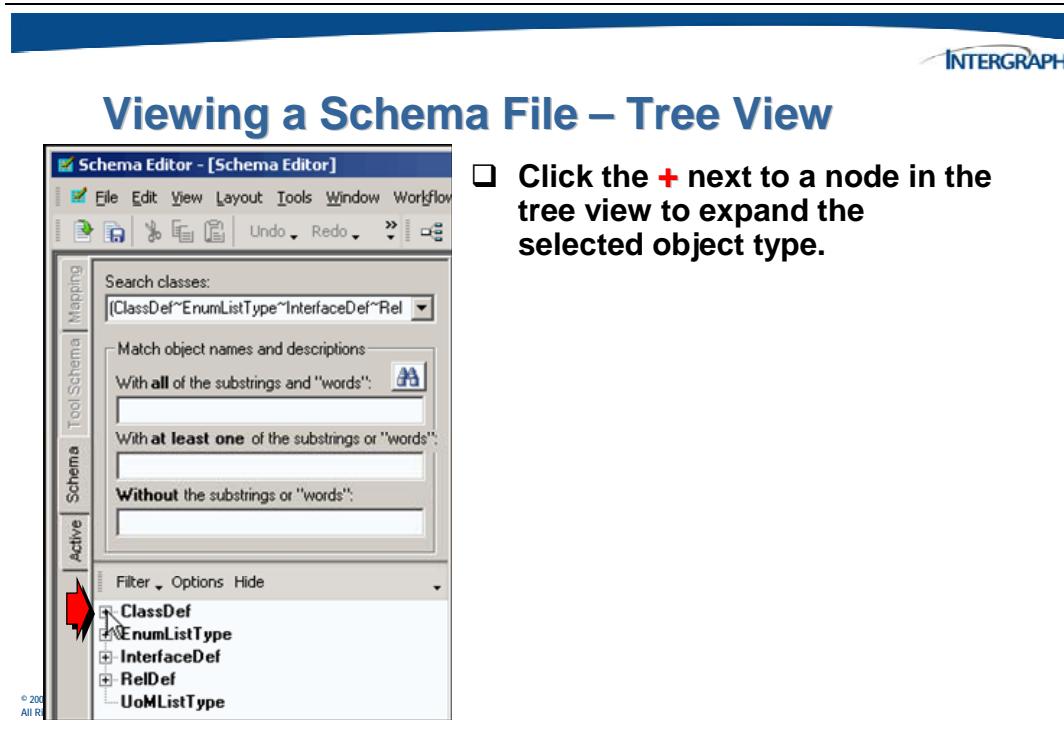
The *MultiTab Options* dialog box will appear. This dialog box will allow you to add/remove more views to/from the list of horizontal tabs.



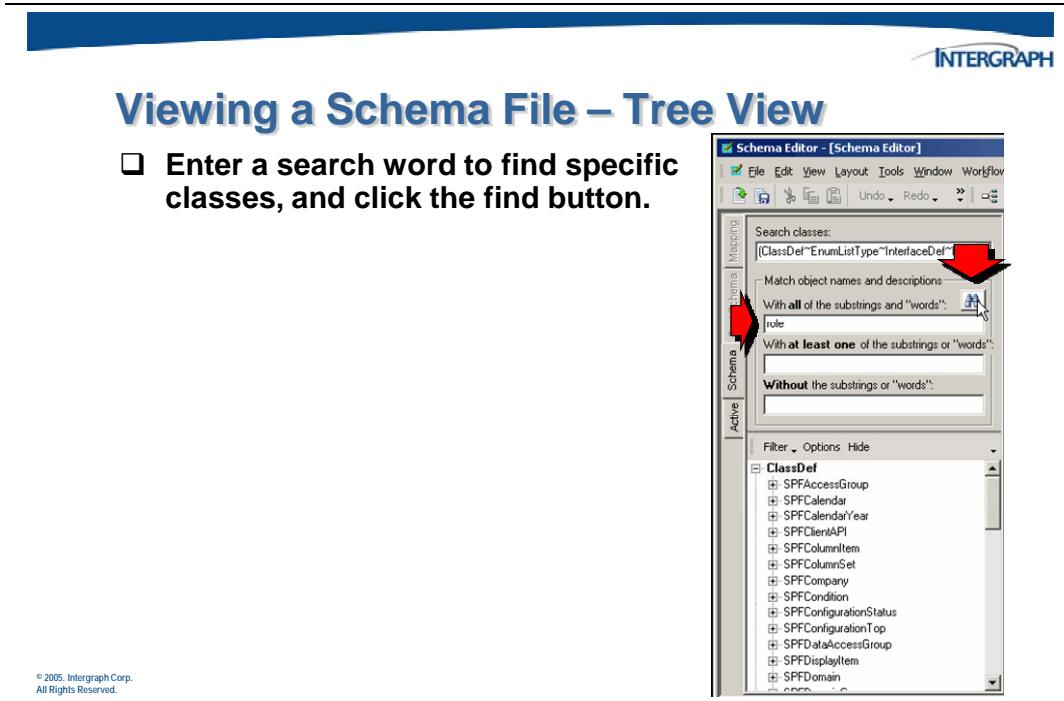
© 2005. Intergraph Corp.
All Rights Reserved.

The first example, the tree view, is useful for general navigation through relationships, but not for looking at property values.

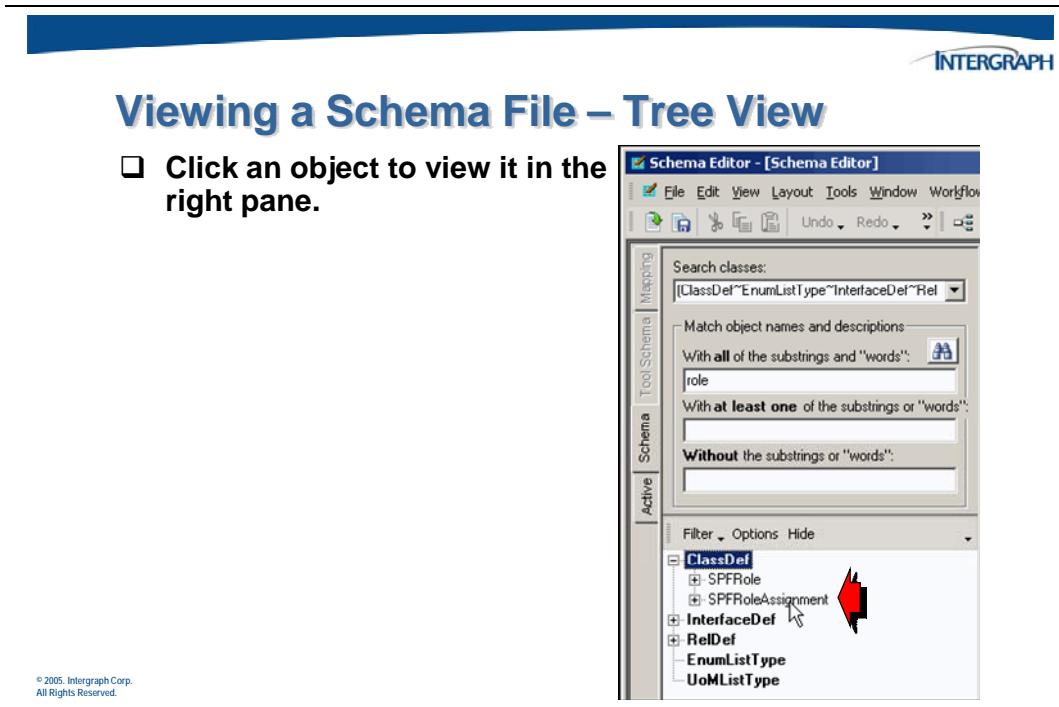
The *Schema Tree* window will be displayed. You can navigate all the relationships defined in the schema using the Schema Tree view.



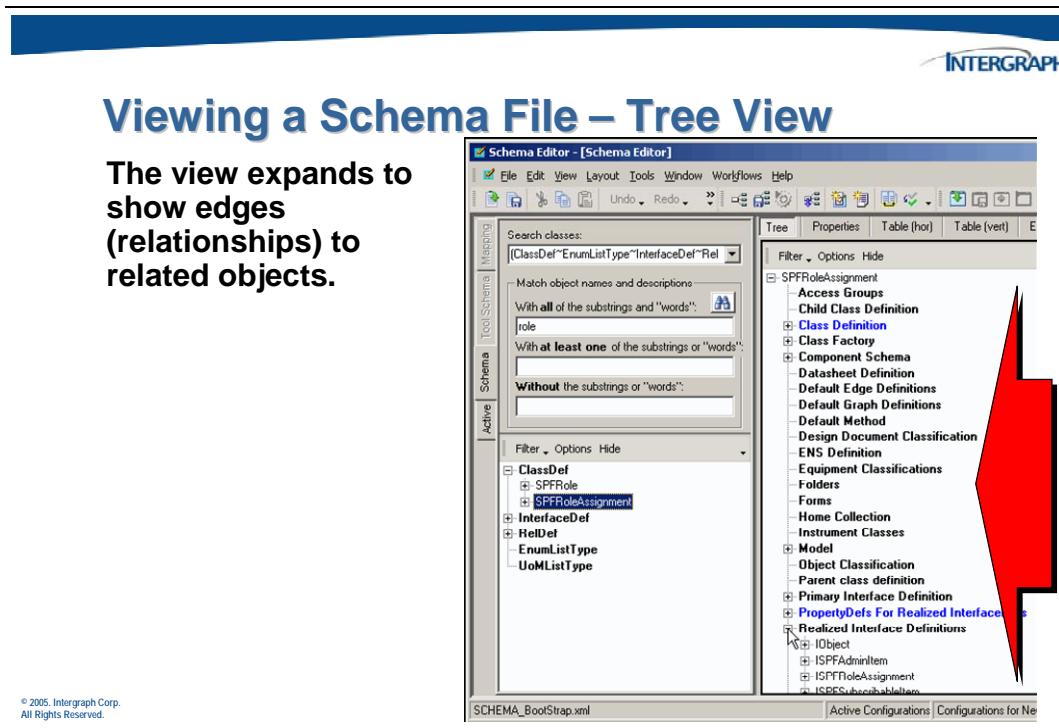
The *ClassDef* object relationships are shown.



Expand the ClassDef node to see a list of the class definition objects that met your search criteria.

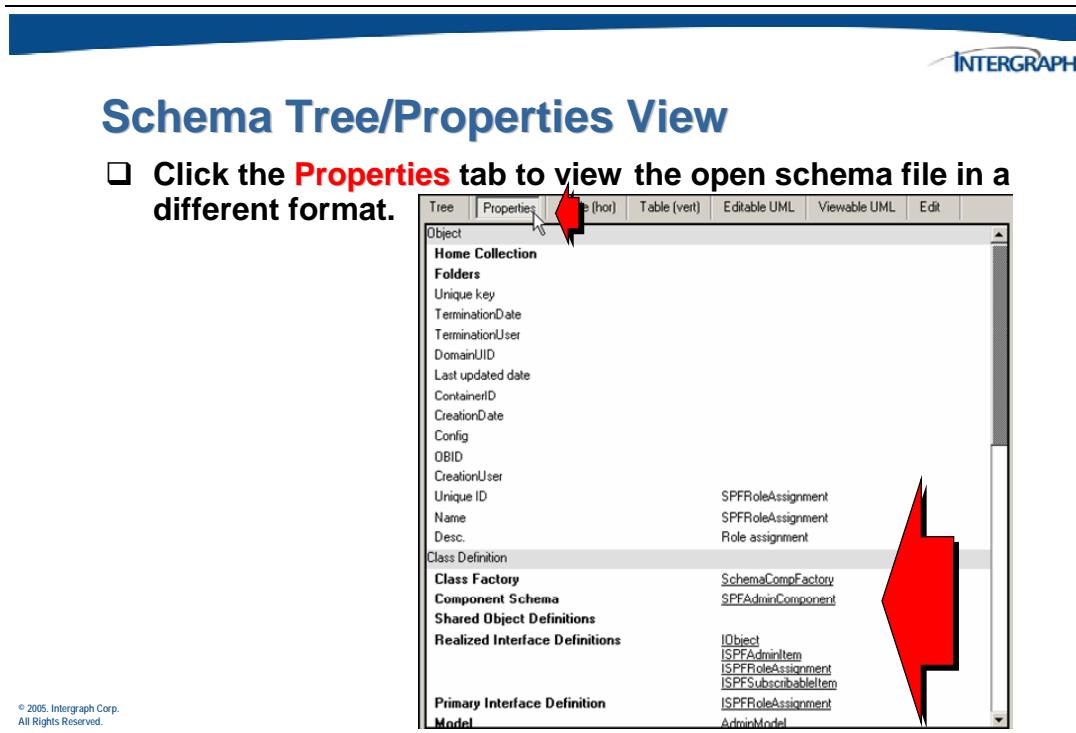


In the tree, objects that are colored **black** are **standard objects** while those in **blue** are **special** (multiple relationships) and are limited.

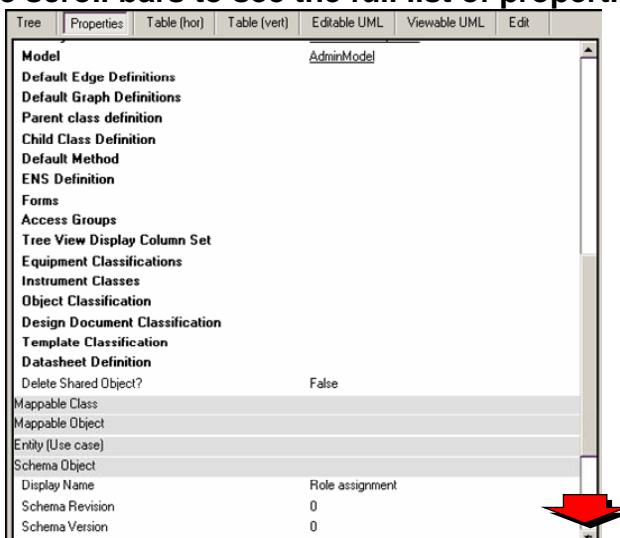


2.2.1 Schema Tree/Properties View

The **Tree/Properties** view combines the same tree view used elsewhere in the Schema Editor with a properties view that emulates display of properties in the right pane in SmartPlant Foundation. This *properties* view identifies each interface for the selected object with text on a shaded background. Following each interface are the properties exposed by that interface and their values for the selected object.

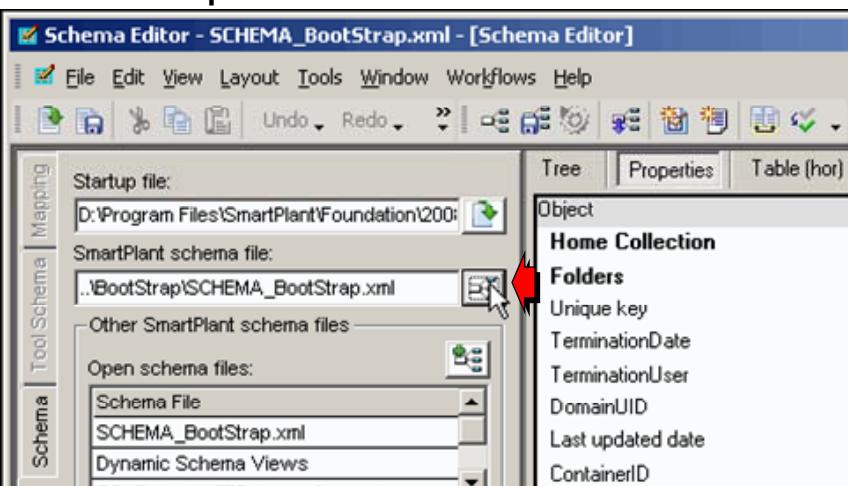


Scroll down to view the rest of the *Tree/Properties* view.



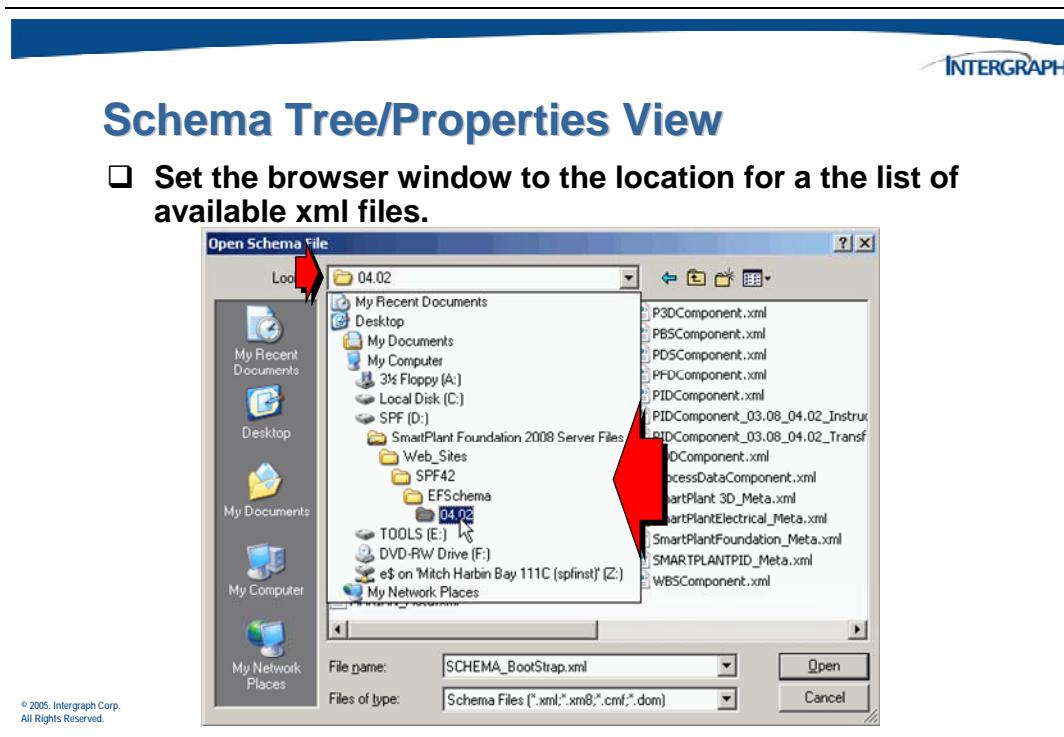
The screenshot shows the 'Schema Tree/Properties View' window. At the top, there is a toolbar with tabs: Tree, Properties, Table (hor), Table (vert), Editable UML, Viewable UML, and Edit. Below the toolbar is a scroll bar. The main area contains a list of properties for an object named 'AdminModel'. The properties listed include Model, Default Edge Definitions, Default Graph Definitions, Parent class definition, Child Class Definition, Default Method, ENS Definition, Forms, Access Groups, Tree View Display Column Set, Equipment Classifications, Instrument Classes, Object Classification, Design Document Classification, Template Classification, Datasheet Definition, Delete Shared Object? (set to False), Mappable Class, Mappable Object, Entity (Use case), Schema Object, Display Name (set to Role assignment), Schema Revision (set to 0), and Schema Version (set to 0). A red arrow points downwards at the bottom of the scroll bar, indicating where to scroll to see more properties.

Use the open button beside the *SmartPlant Schema file* field to open the **PIDComponnt.xml** schema file.

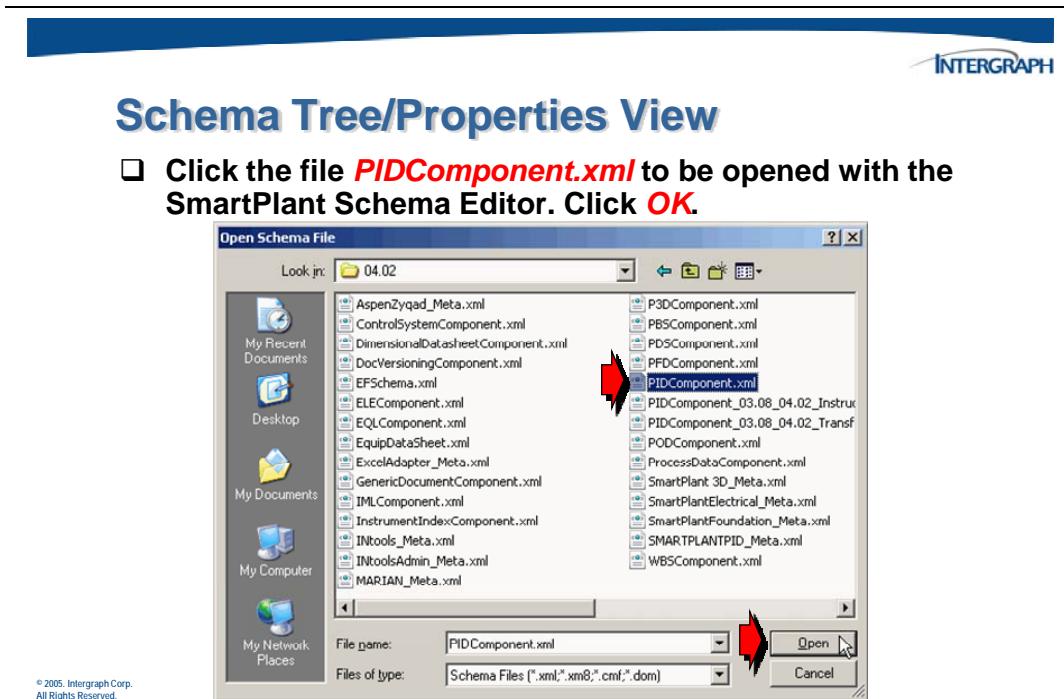


The screenshot shows the 'Schema Editor - SCHEMA_BootStrap.xml - [Schema Editor]' window. The title bar has tabs: File, Edit, View, Layout, Tools, Window, Workflows, Help. Below the title bar is a toolbar with various icons. On the left, there is a vertical pane with tabs: Mapping, Tool Schema, and Schema. The 'Tool Schema' tab is currently selected. In the main area, there are three sections: 'Startup file:' with the path 'D:\Program Files\SmartPlant\Foundation\200', 'SmartPlant schema file:' with the path '..\BootStrap\SCHEMA_BootStrap.xml', and 'Other SmartPlant schema files:' which lists 'Schema File', 'SCHEMA_BootStrap.xml', 'Dynamic Schema Views', and 'SCHEMA_MinFFSchema_Gen.xml'. A red arrow points to the 'SCHEMA_BootStrap.xml' file in the 'SmartPlant schema file:' section. To the right, there is a 'Tree' tab selected in a tab bar, followed by 'Properties' and 'Table (hor)'. The 'Properties' tab displays a list of properties for an object named 'Home Collection': Folders, Unique key, TerminationDate, TerminationUser, DomainUID, Last updated date, and ContainerID. A red arrow also points to the 'Folders' property in the list.

Browse to the **SmartPlant Foundation 2008 Server**
Files\Web_Sites\SPF42\EFSchema\ 04.02 folder to find the XML file to be opened.

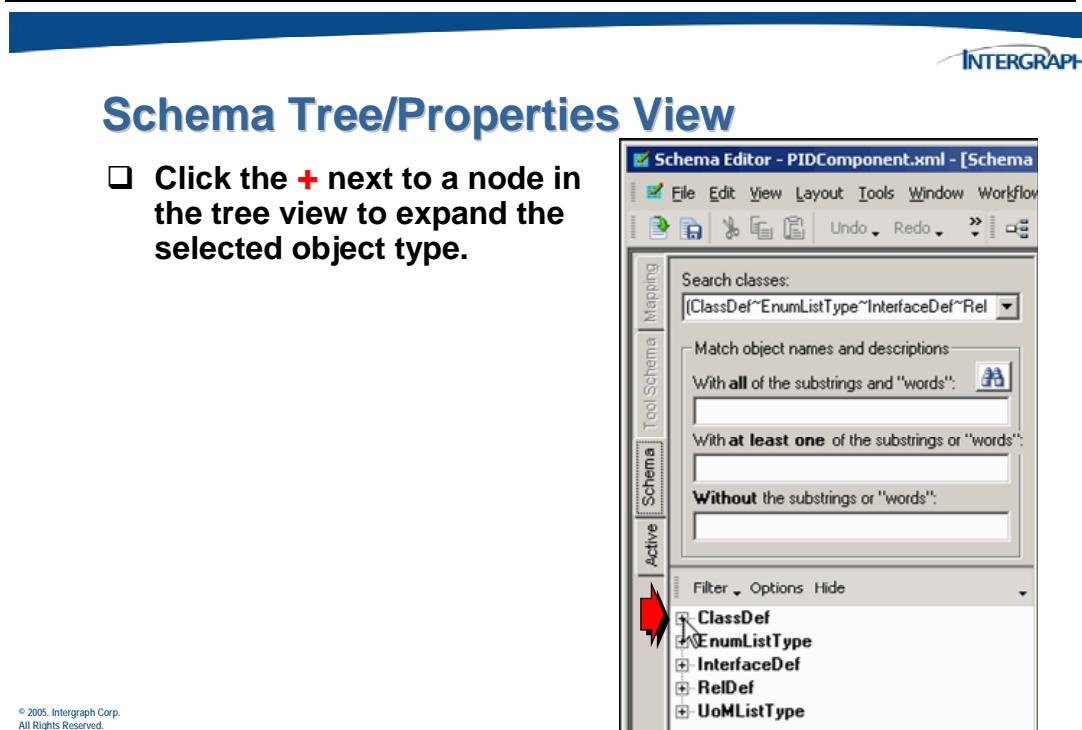


Select the **PIDComponent.xml** file from the list of available files.

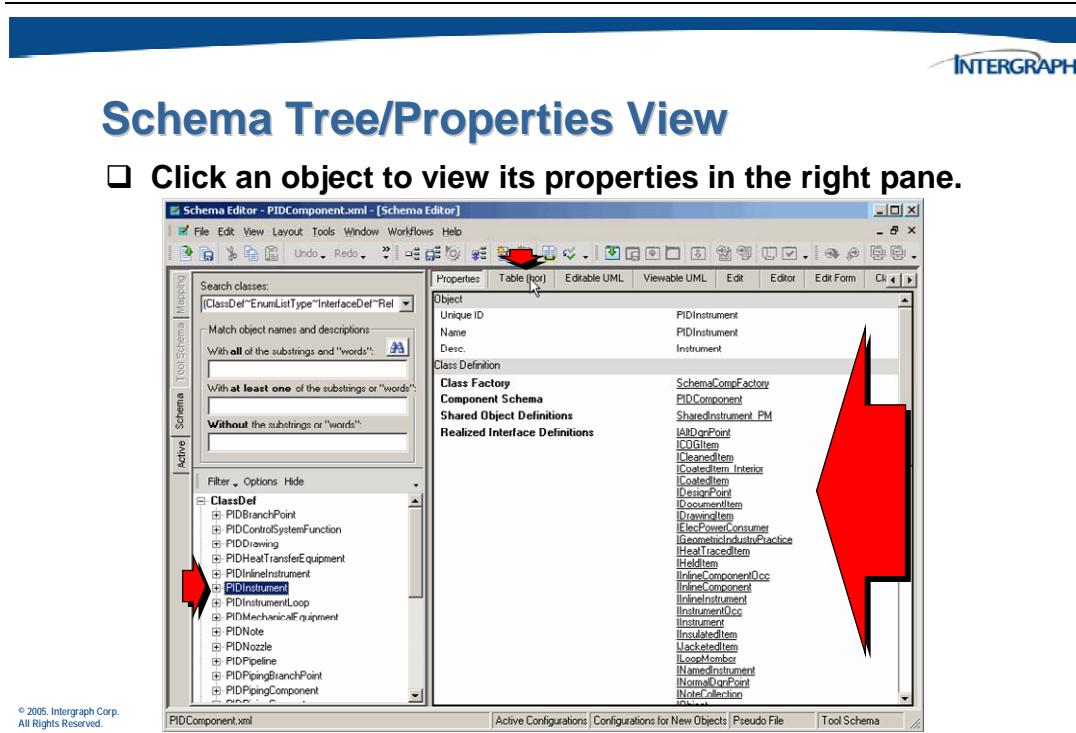


The reason for starting with the PIDComponent.xml file is that it is a smaller version of the “master” SmartPlant schema file, EFSchema.xml. The PIDComponent.xml contains all of the same object types as the master schema, *ClassDefs*, *InterfaceDefs*, *PropertyDefs*, *RelDefs*, and so forth. However, instead of all application definitions, only those specific to SmartPlant P&ID will be shown. Therefore, there will be fewer objects to scroll through in the tree.

Also, if a definition is accidentally deleted, the PIDComponent.xml schema file is easy to re-create.



Click on the **Properties** tab and the *Schema Tree/Properties* view is displayed.

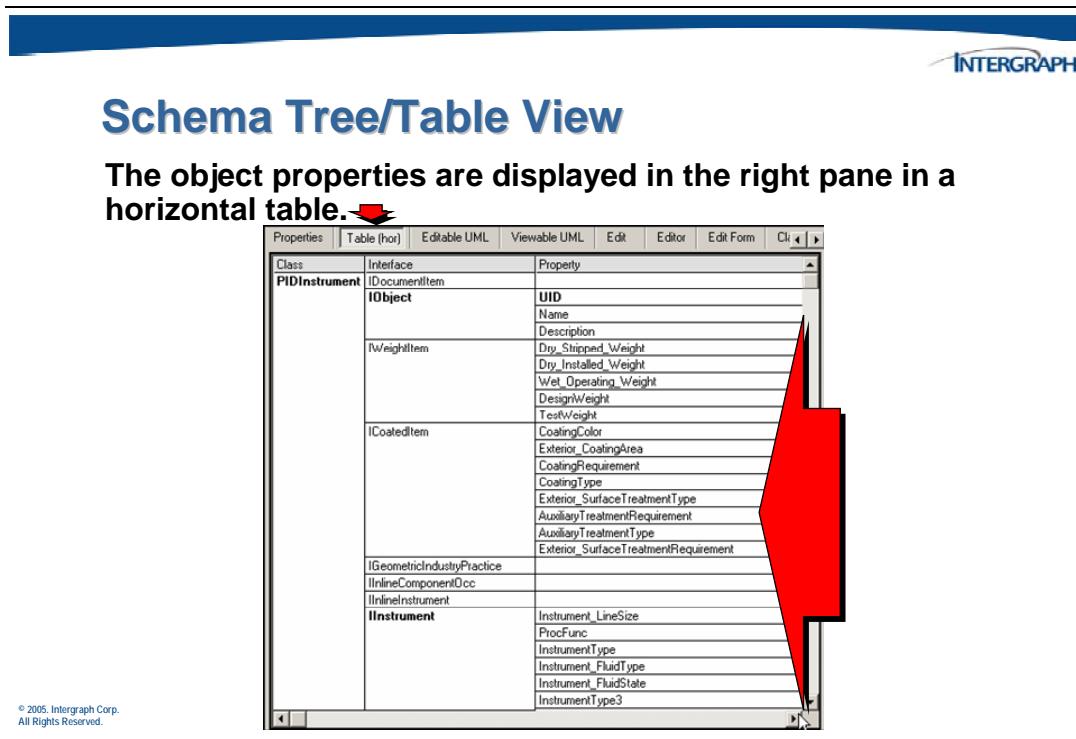


The view expands to show properties related to the class objects.

2.2.2 Schema Tree/Table View

The Schema **Tree/Table** view is a combination of the standard tree view with a table that provides a different view of the information based on what you select in the tree view. The table views are predefined by class to show the most useful information for each type of object.

Click on the **Table (hor)** tab, and the *Schema Tree/Table* view is displayed.



There are two types of tables that you can choose from: vertical and horizontal. The vertical table displays row headers along the left side of the table and data in vertical columns.

The horizontal table displays column headers across the top of the table, with the data displayed in horizontal rows. In the following example, a horizontal table is display is used.

In the table view, bold text is used in class definitions and interface definitions to identify required entries. Bold interfaces in the table view indicate that the interface is required for the associated class.

Properties that appear in bold text in the table view are required for the associated interface.

Bold property types in the table view indicate that the value for the property cannot be null.

The screenshot shows a software interface titled "Schema Tree/Table View". At the top, there is a toolbar with buttons for Properties, Table (hor), Editable UML, Viewable UML, Edit, Editor, Edit Form, and Click. Below the toolbar is a table with two columns: "Property" and "Type". The "Type" column contains several bolded terms: "string128", "MassUoM", "CoatingColors", "AreaUoM", "CoatingReq1", "CoatingReq2", "ExteriorSurfaceTreatment2", "AuxiliaryTreatmentRequirement", "AuxiliaryTreatmentType2", "ExteriorSurfaceTreatmentRequirement", "LengthUoM", "InstrumentProcessFunctions", "InstrumentTypes2", "InstrumentFluidTypes", "FluidPhases", and "InstrumentTypes3". Red arrows point from the text "Use the scroll bars to see the full list of properties." to the vertical scroll bar on the right side of the table and to the bottom scroll bar at the bottom of the table. In the bottom left corner of the table area, there is a small copyright notice: "© 2005. Intergraph Corp. All Rights Reserved."

Property	Type
UID	string128
Name	
Description	string
Dry_Shipped_Weight	MassUoM
Dry_Installed_Weight	
Wet_Operating_Weight	
DesignWeight	
TestWeight	
CoatingColor	CoatingColors
Exterior_CoatingArea	AreaUoM
CoatingRequirement	CoatingReq1
CoatingType	CoatingReq2
Exterior_SurfaceTreatmentType	ExteriorSurfaceTreatment2
AuxiliaryTreatmentRequirement	(A106CA33-CD24-45E6-A96B-36316EA43666)
AuxiliaryTreatmentType	AuxiliaryTreatmentType2
Exterior_SurfaceTreatmentRequirement	(9C560101-4F4B-4B28-9133-325F937637C9)
Instrument_LineSize	LengthUoM
ProcFunc	InstrumentProcessFunctions
InstrumentType	InstrumentTypes2
Instrument_FluidType	InstrumentFluidTypes
Instrument_FluidState	FluidPhases
InstrumentType3	InstrumentTypes3

Bold text in the tree view represents relationship edges. Relationship edges define paths that go from one point to another point in the data without traversing the entire tree.

2.2.3 Schema Tree/Drag-Drop UML View

In the Schema Editor, you can view the schema using a variety of Unified Modeling Language (UML) views. The most customizable of all the UML views are the two UML views. These views allow you to drag objects from a tree view into the UML view. As you drag items to the UML view and drop them, the relationships among those items are automatically displayed in the UML view.



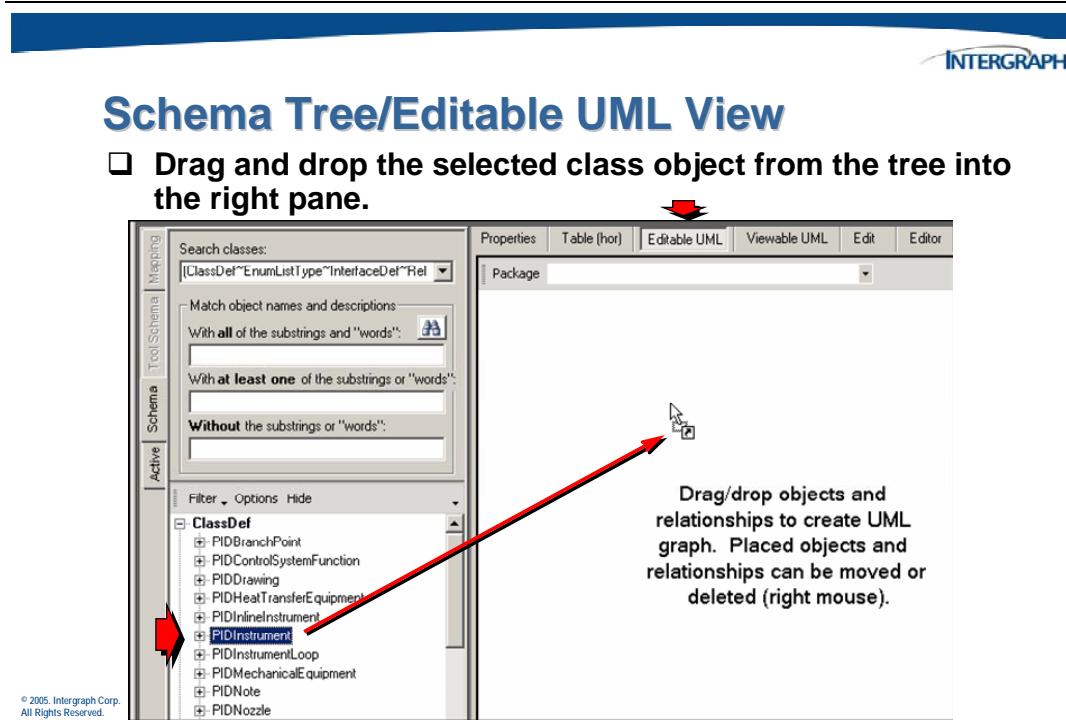
Schema Tree/Editable UML View

You can create your own UML view by selecting the objects that you want to display in the UML view.

The UML view can be customized even further by:

- Moving objects and labels**
- Changing the display color for objects**
- Removing objects, relationships, and labels**
- Changing the display of interfaces**

The **Tree/Editable UML View** displays a tree view on the left of the drag-and-drop UML pane.



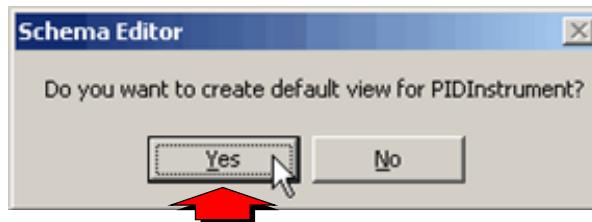
You can select items from the tree view to display in the UML view by dragging the objects from the tree view to the UML view.

When you drag and drop an object to the UML view, you will be prompted to create either a *custom* UML view or a default view. An example of creating a custom UML view will be demonstrated later.



Schema Tree/Editable UML View

Click **Yes** to create a default UML view.



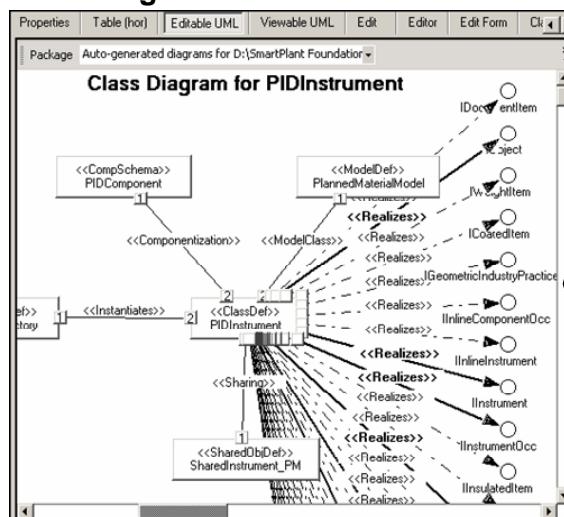
© 2005, Intergraph Corp.
All Rights Reserved.

In the resulting UML view, you can view the realizes relationships between the selected objects and the relationship ends for each one.



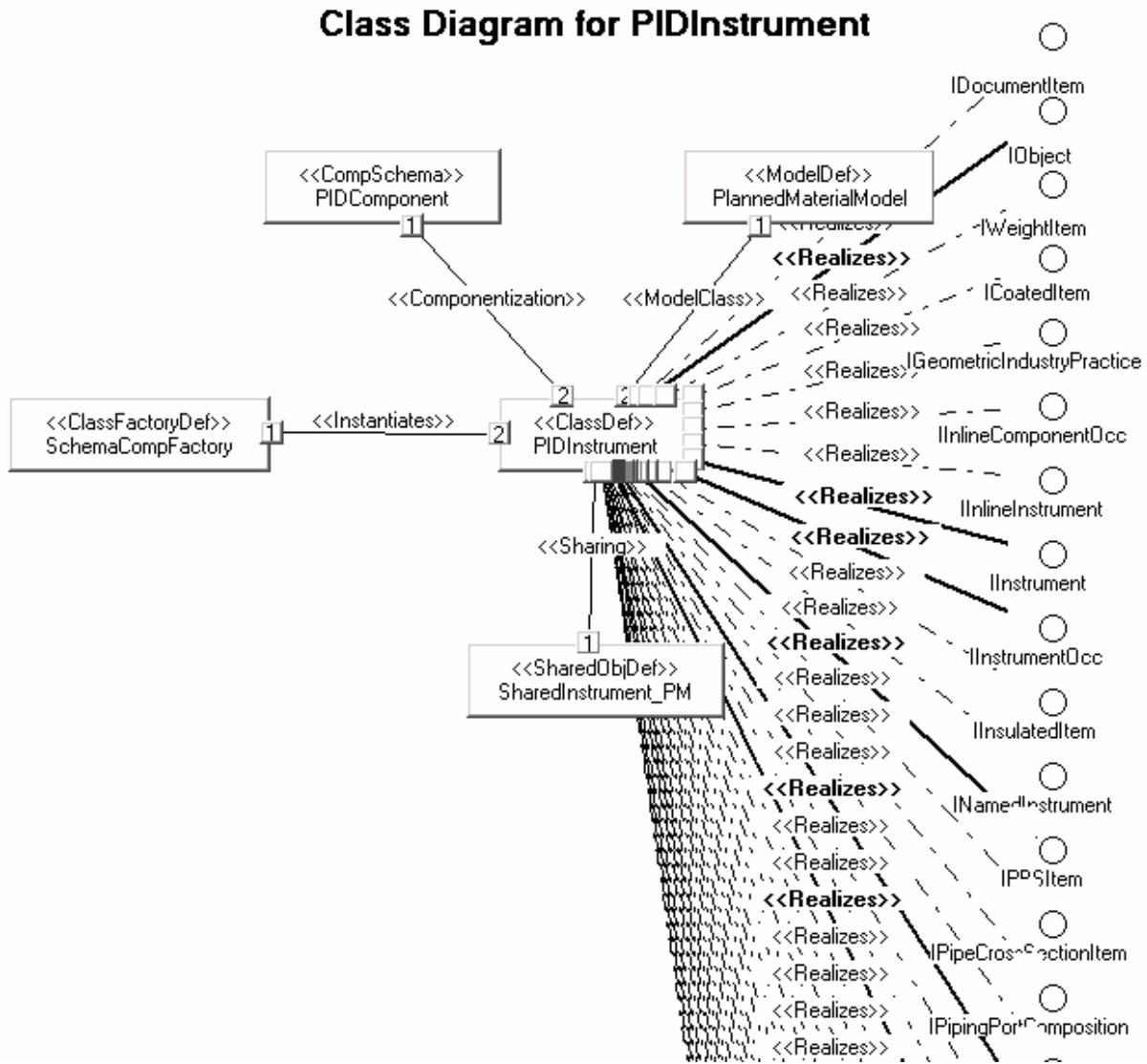
Schema Tree/Editable UML View

A default class diagram for the **PIDInstrument** object is created.

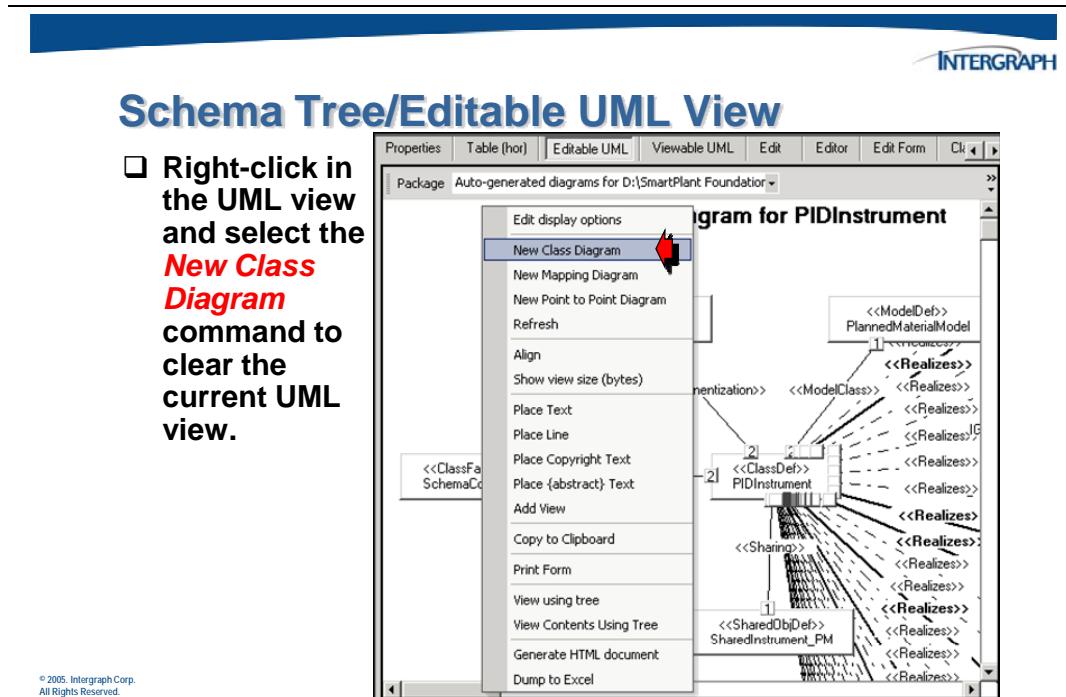


© 2005, Intergraph Corp.
All Rights Reserved.

Below is a magnified example of the Schema Tree/Editable UML view.



Use the right mouse button menu to clear the UML view back to a blank view.

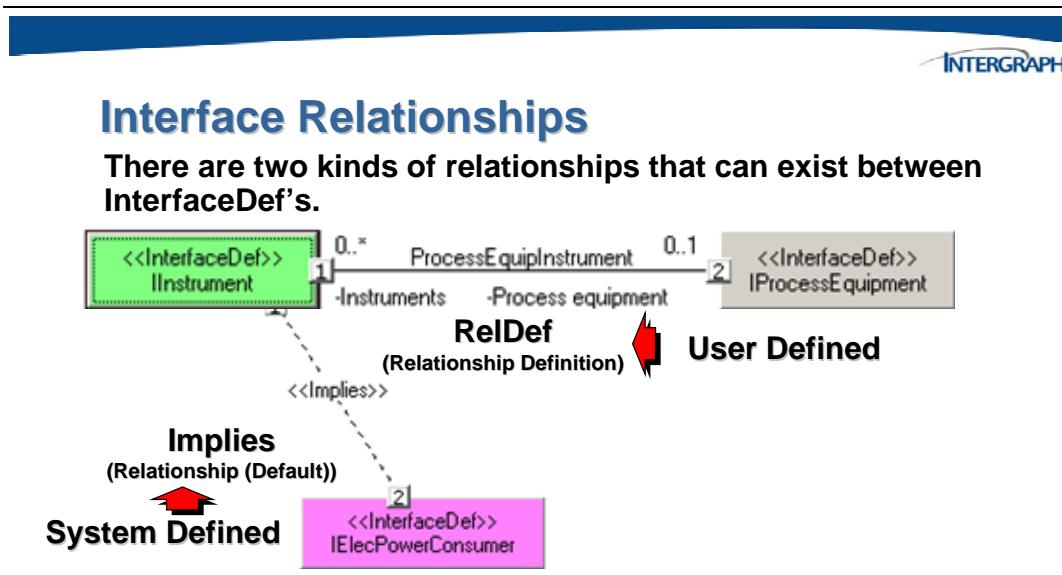


2.2.4 Interface Relationships

In the previous chapter, relationships between interfaces were introduced. In the examples used in this chapter, you will be exposed to different kinds of relationships that are used to associate interfaces.

The two types of relationships used from interface to interface are:

- ❑ *Relationship Definitions* (RelDef) – these are user defined relationships defined by the data modeler and are used to associate interfaces that are realized by different class definitions (ClassDef's).
- ❑ *Implied* (Relationship) – this type of relationship is thought of as a system default relationship. It is used primarily for inheritance from one interface to another interface where both are realized by the **same** ClassDef.

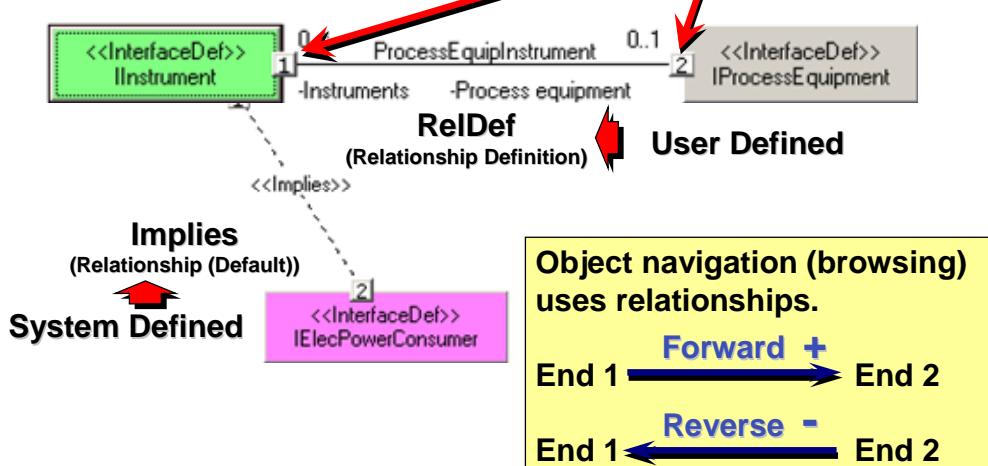


Both *RelDef's* and *Implied* relationships can be used for object navigation or browsing, but RelDef's provide for more browsing control. This will be discussed in more detail in chapter 5.



Interface Relationships

Each relationship has 2 ends, End 1 and End 2.



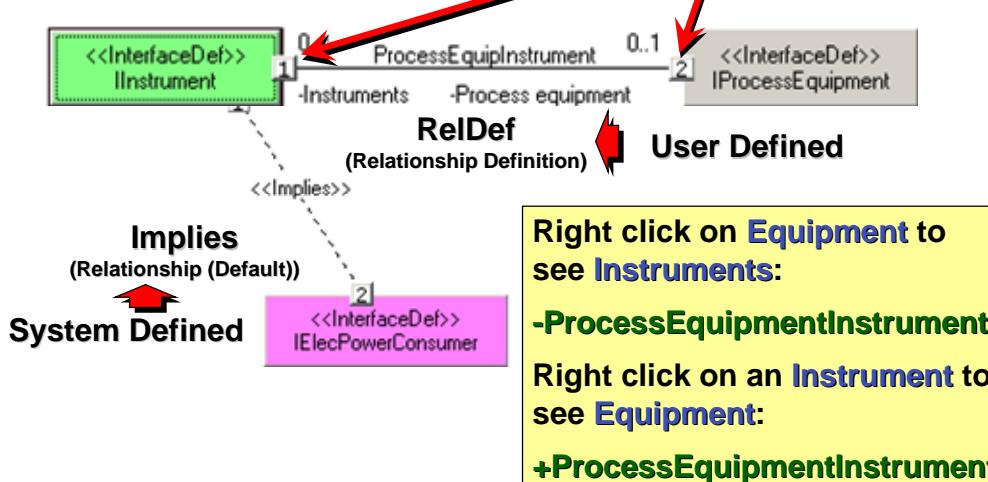
© 2005, Intergraph Corp.
All Rights Reserved.

Either forward or reverse browsing is possible in the Desktop Client.



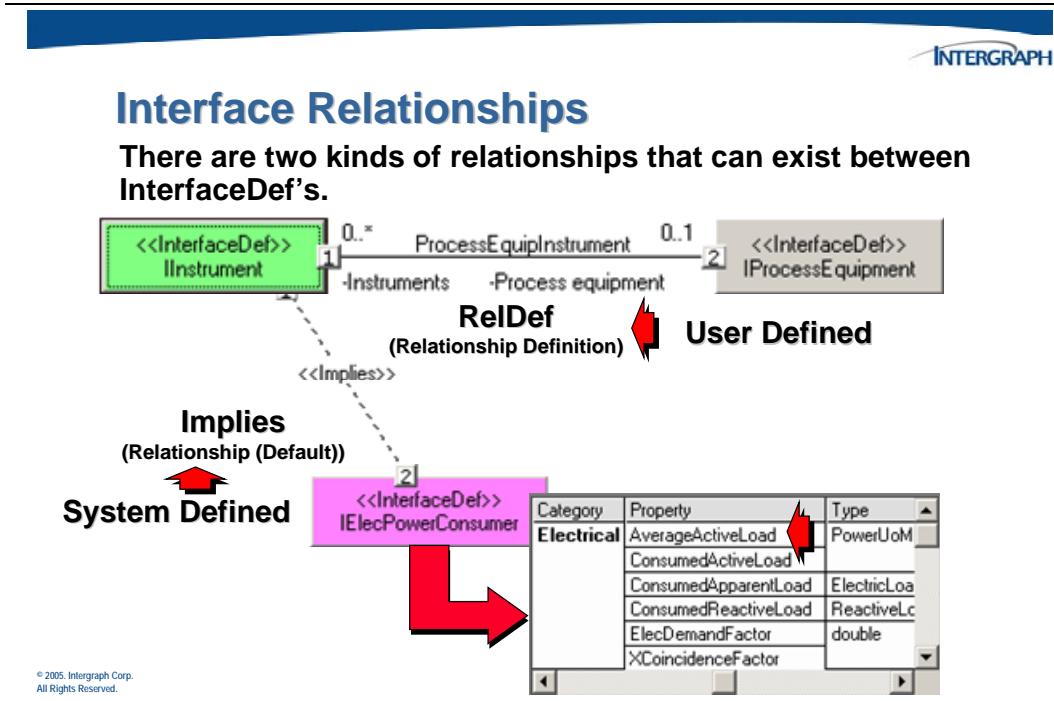
Interface Relationships

Each relationship has 2 ends, End 1 and End 2.

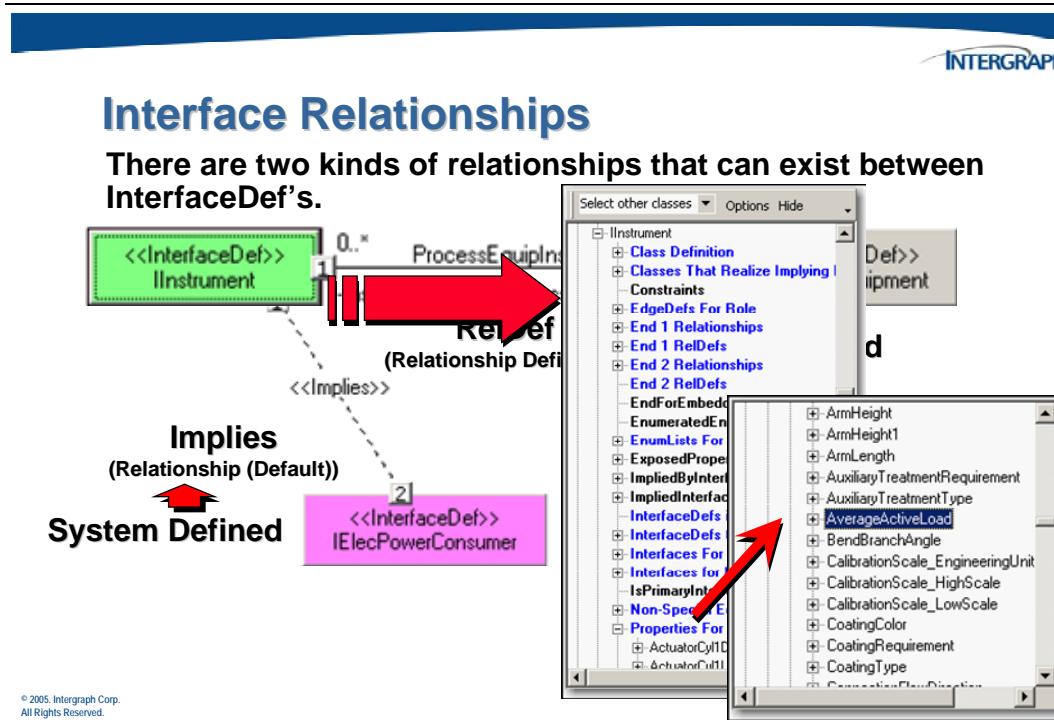


© 2005, Intergraph Corp.
All Rights Reserved.

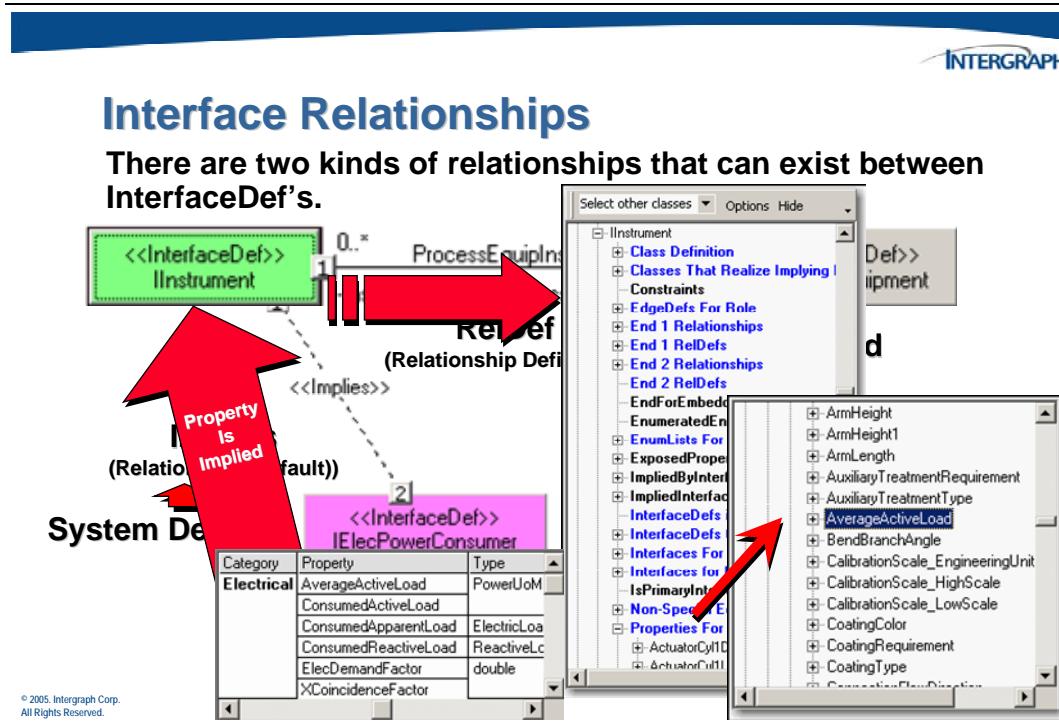
One of the primary functions of the **Implies** relationship is to allow an interface to inherit properties and relationships from an associated interface. For example, the interface *IElecPowerConsumer* exposes a property called **AverageActiveLoad**.



The property *AverageActiveLoad* will display as one of the **Properties For Role** for the interface *IInstrument*.



Because IInstrument **Implies** IElecPowerConsumer, it also inherits AverageActiveLoad as though this property was exposed by IInstrument directly.

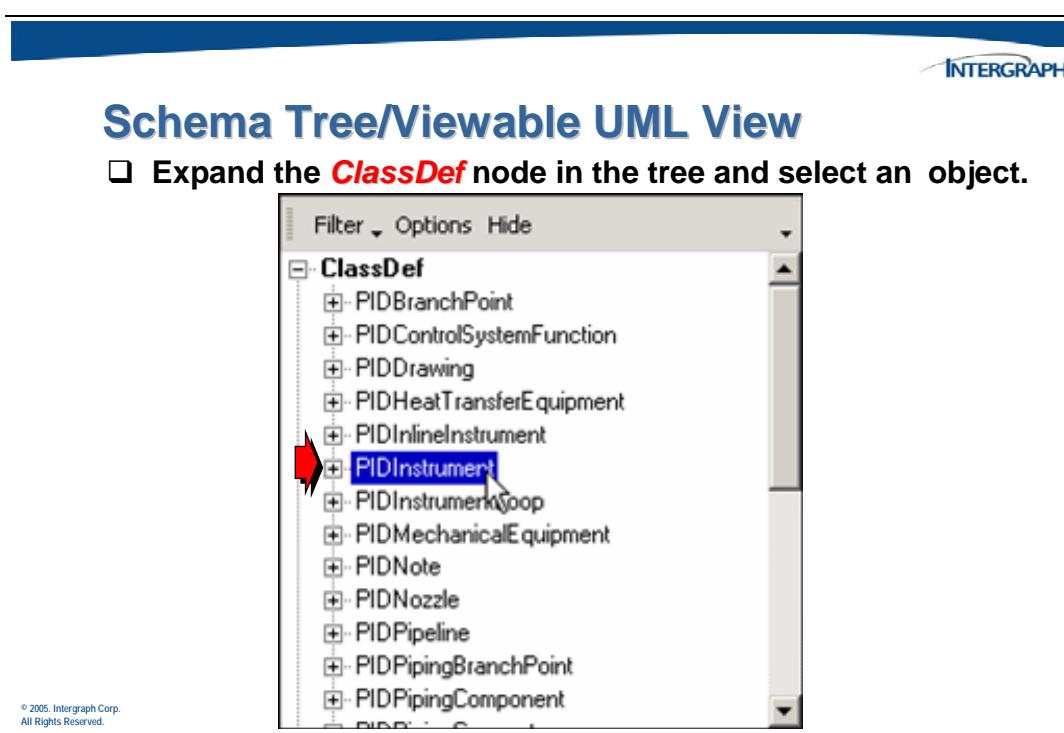


2.2.5 Schema Tree/Viewable UML View

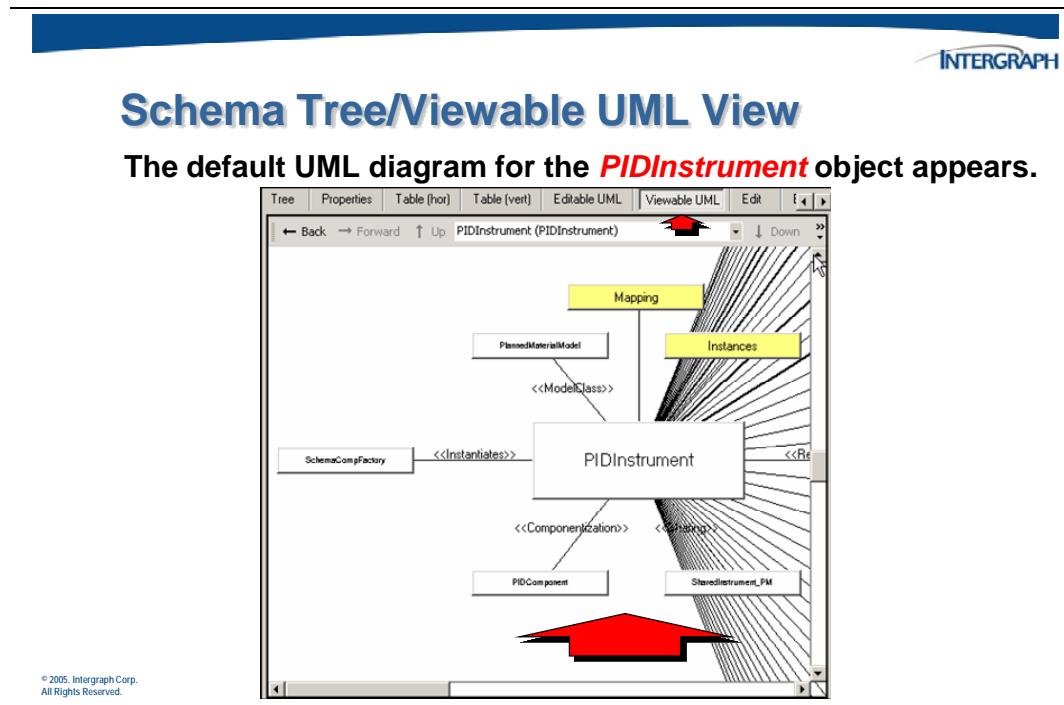
The **Tree/Viewable UML** view is a combination of the standard tree view with a UML view. The Viewable UML view provides a graphical representation of the relationships and relationship definitions involving the object selected in the tree view. The selected object appears in the center of the Viewable UML view. In the Viewable UML view, you can click any object to center the UML around the selected object to view its relationships and relationship definitions.

The UML view is useful for viewing relationships in the schema graphically. You can follow relationships by clicking objects, which is a lot like clicking links in your Web browser to explore Web sites.

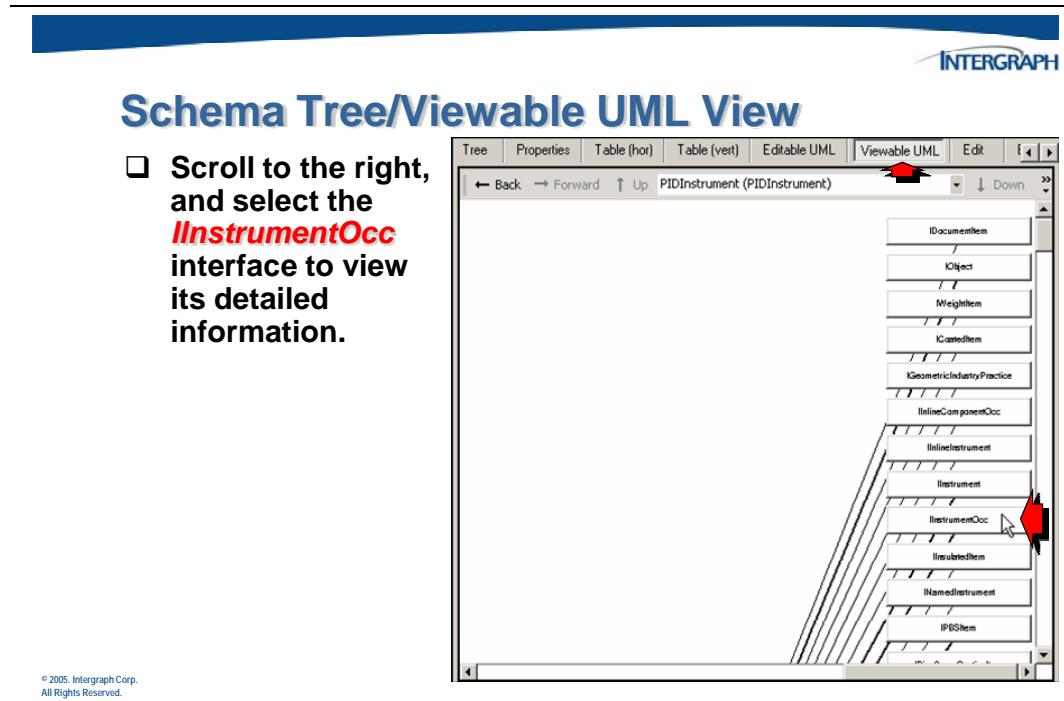
This view is also useful for visualizing relationship definitions that are not displayed elsewhere.



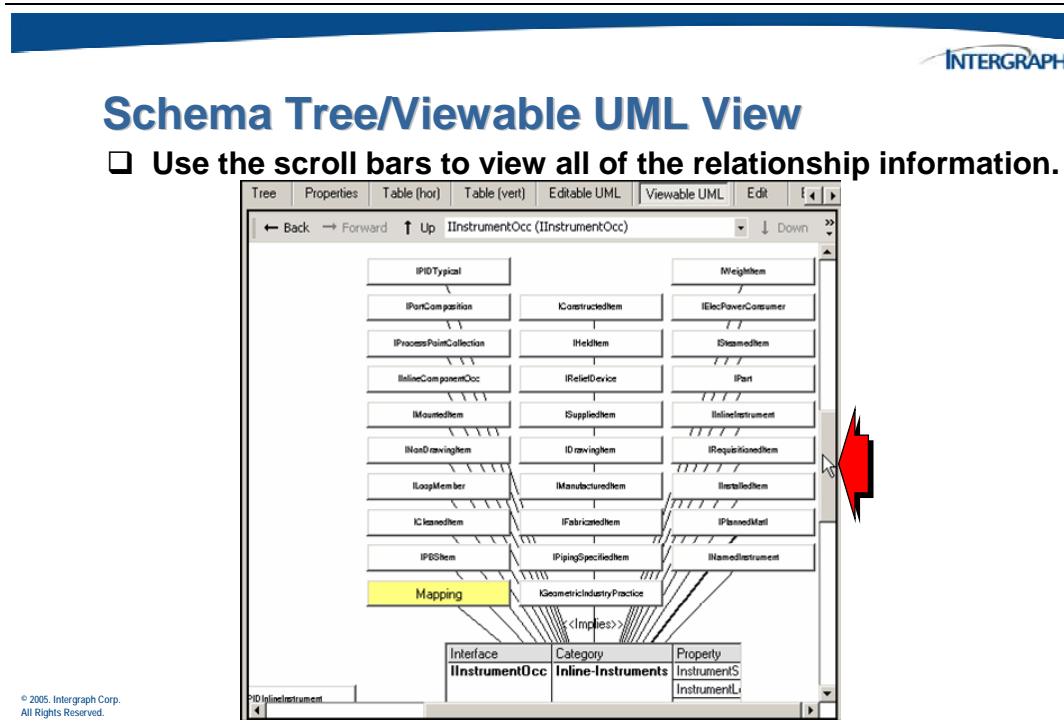
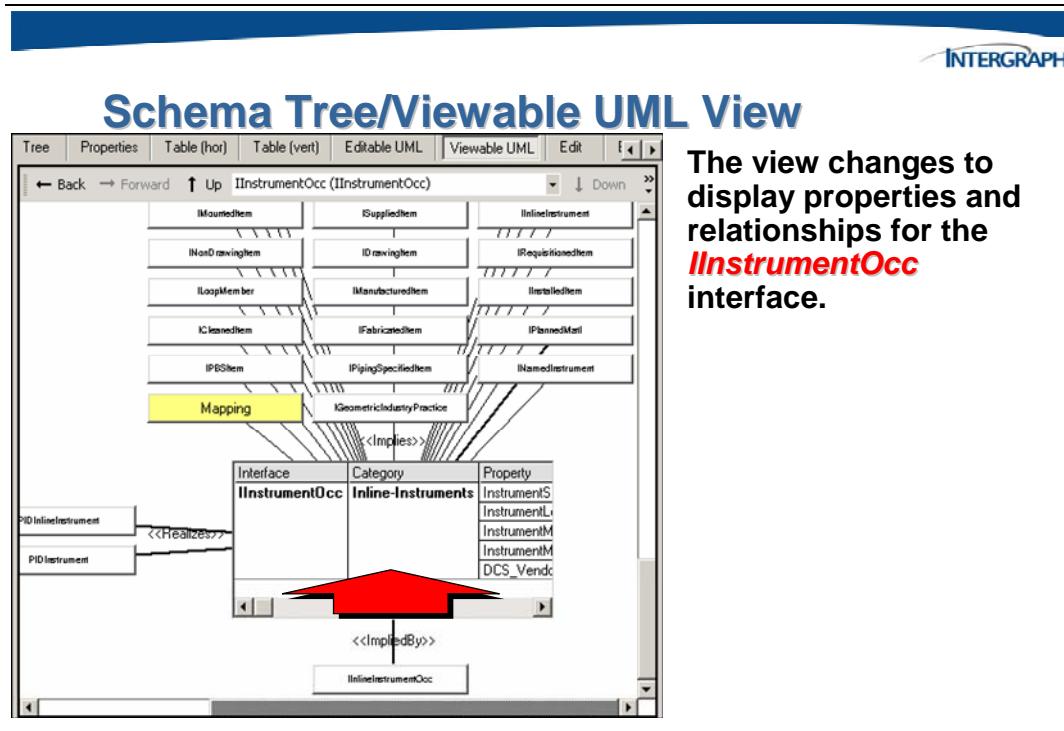
To view the UML for an object, click the object in the tree view.



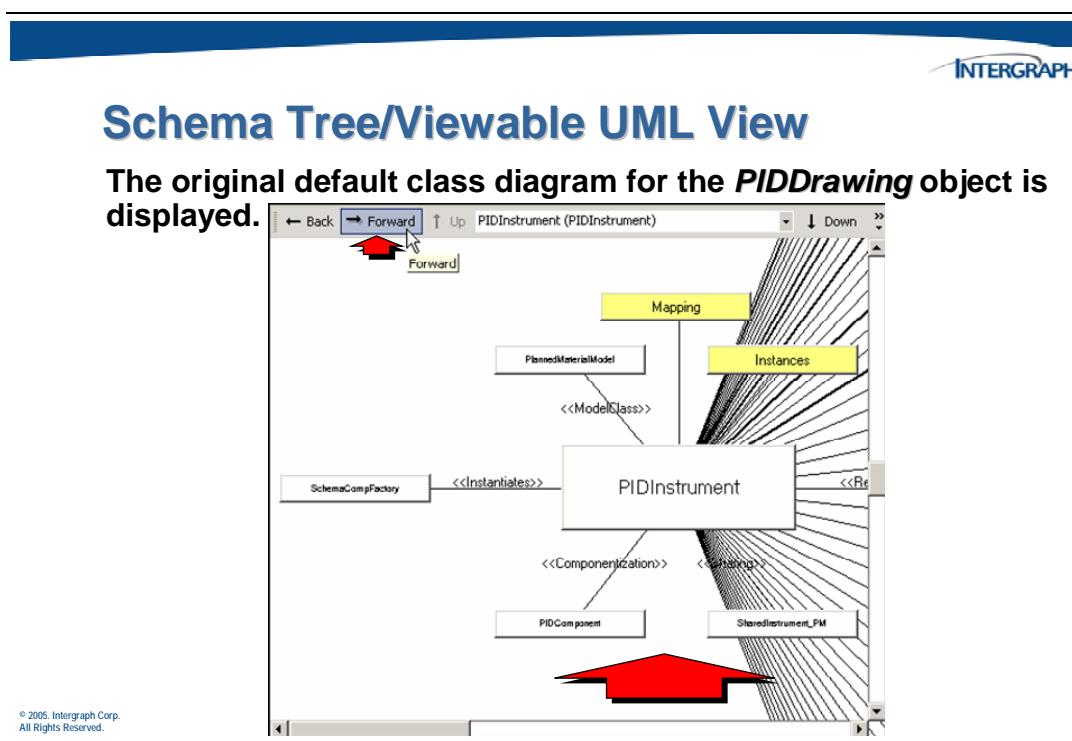
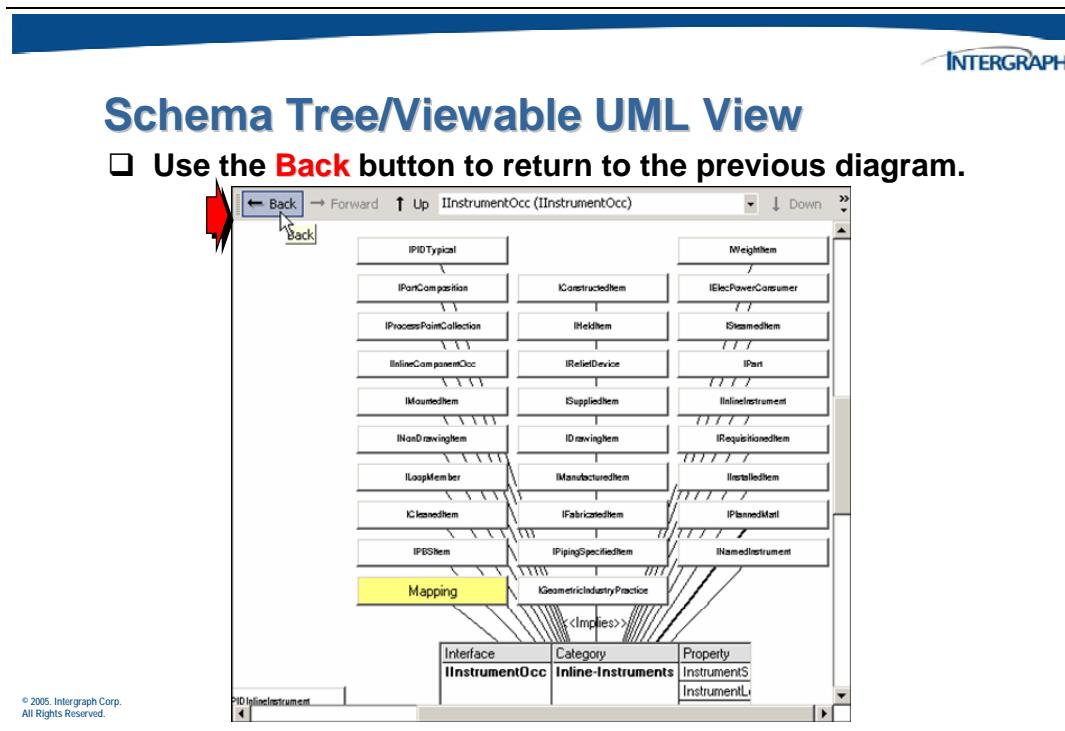
You can use the *Tree/Viewable UML* view to see details of any related edges.



Each type of relationship, such as <<Realizes>> or <<Implies>>, is shown inside angle brackets between the selected object and other classes and interfaces in the UML view.



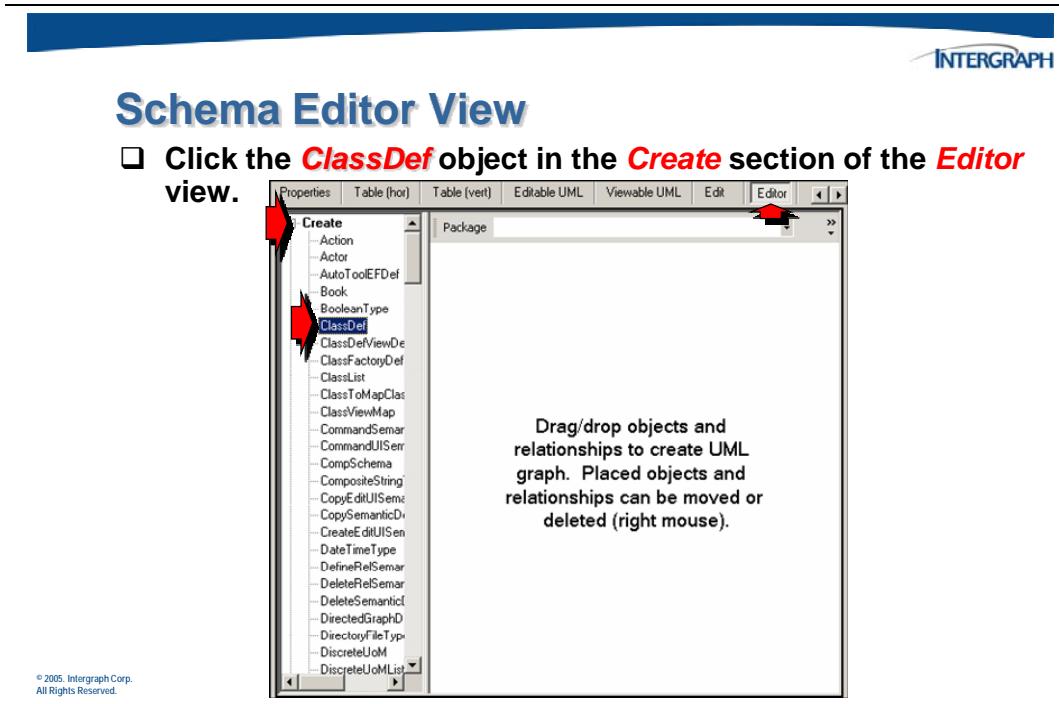
To go back to the last UML view, click **Back** on the toolbar.



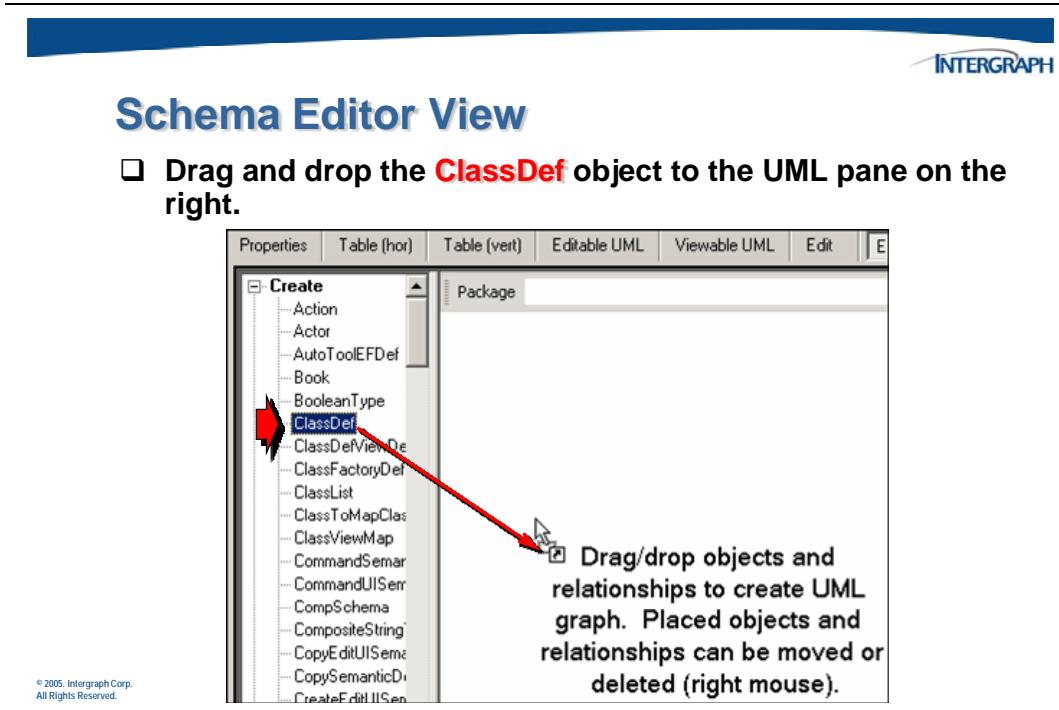
2.2.6 Editor View

In the Schema Editor, you can define new objects, such as classes or properties, in the schema. To add objects to the schema, you can use the **Editor** view.

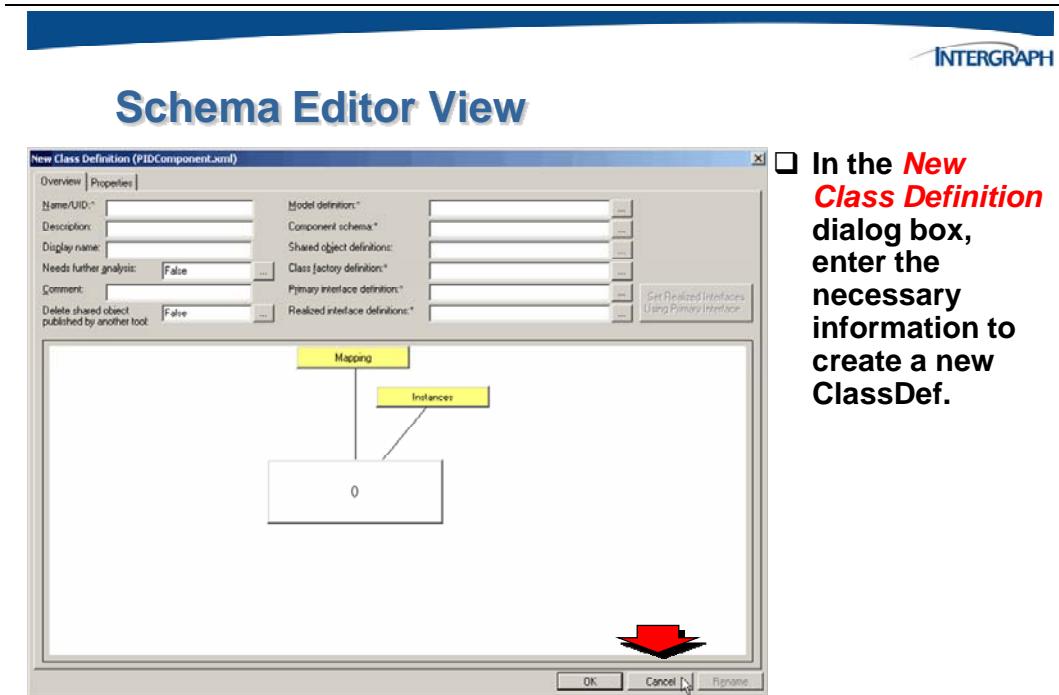
Click the **Editor** tab in the *view* pane.



To create a new object in the **Editor** view, drag the object type from the **Create** tree to the UML view.



When you drag an object type to the UML view, the *New* dialog box for that object type appears. Only those objects that have *New* dialog boxes defined for them in the Schema Editor appear in the **Create** tree.

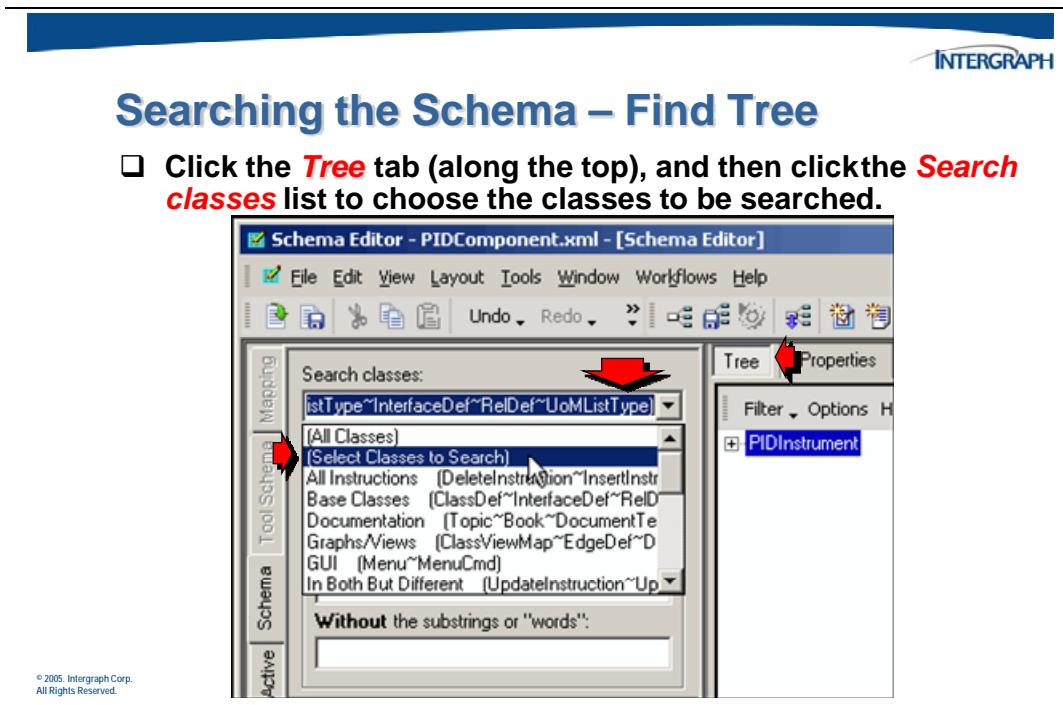


2.3 Finding Schema Objects

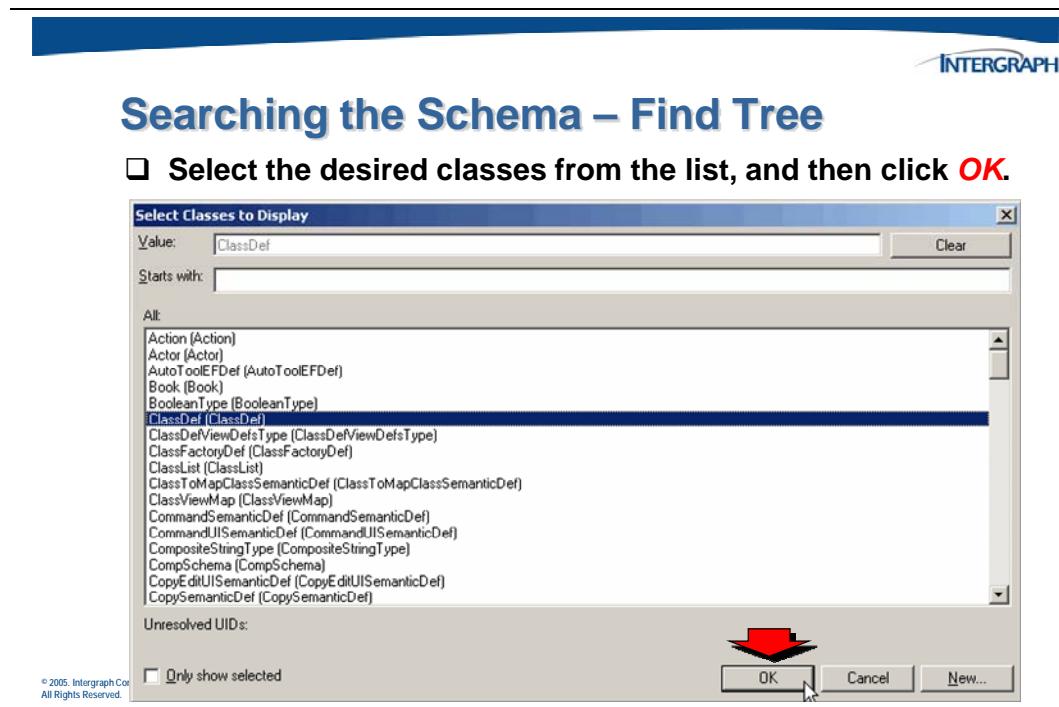
In the Schema Editor, you can locate particular objects in the schema, meta schema, tool schemas, or data files by defining search criteria for the object that you want to find, including the name and description of the object and its UID.

2.3.1 View Schema Find Tab

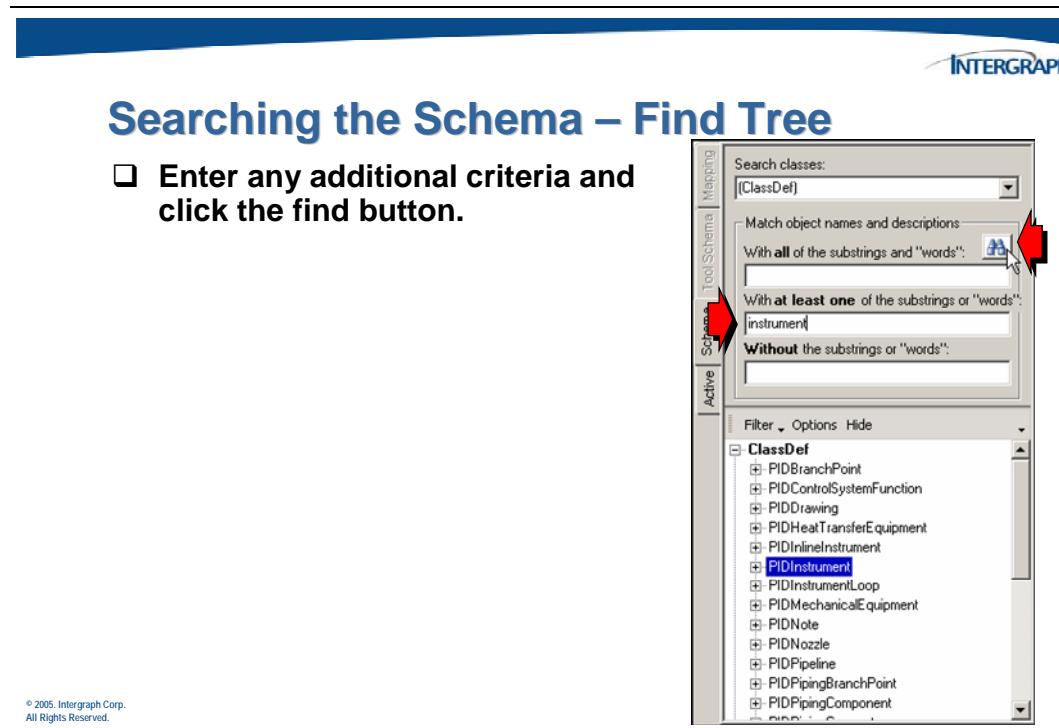
To find an object in the Schema Editor, click the list button to the right of the *Search classes* field. This allows you to define the search classes that you want to include in your search.



The *Search Classes to Display* dialog will appear.



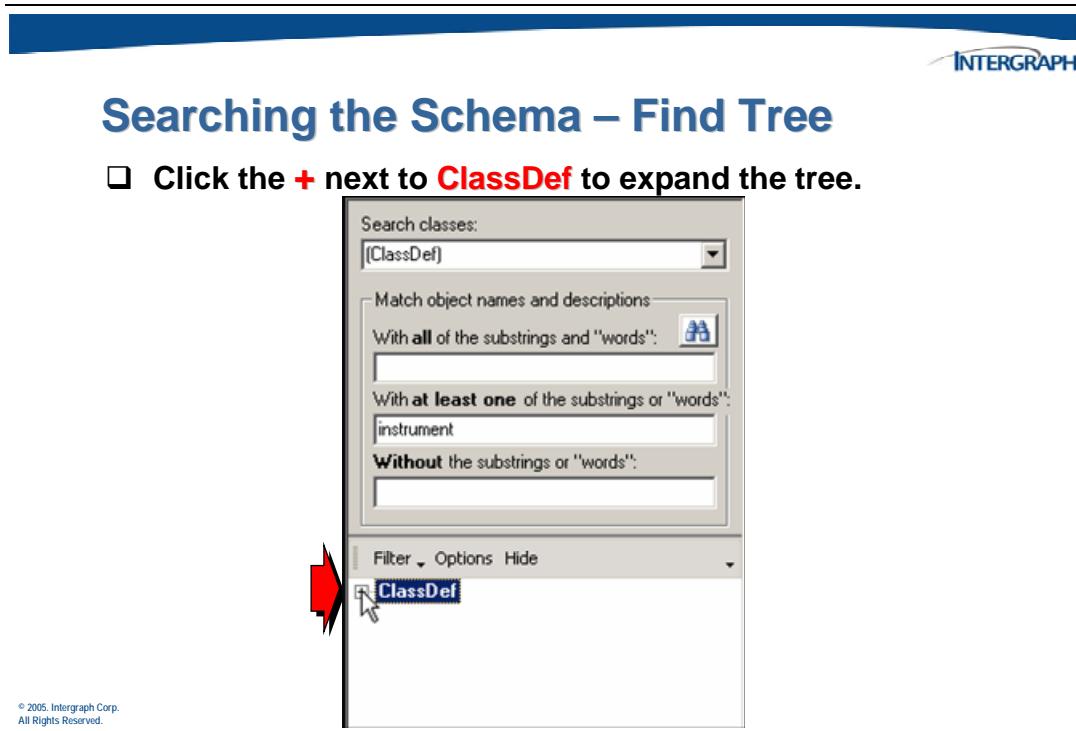
Under **Match object names and descriptions**, type text you want to include and exclude from your search. The “*” character can be used to perform wildcard searches.



The following defines the fields used the *Find* dialog box:

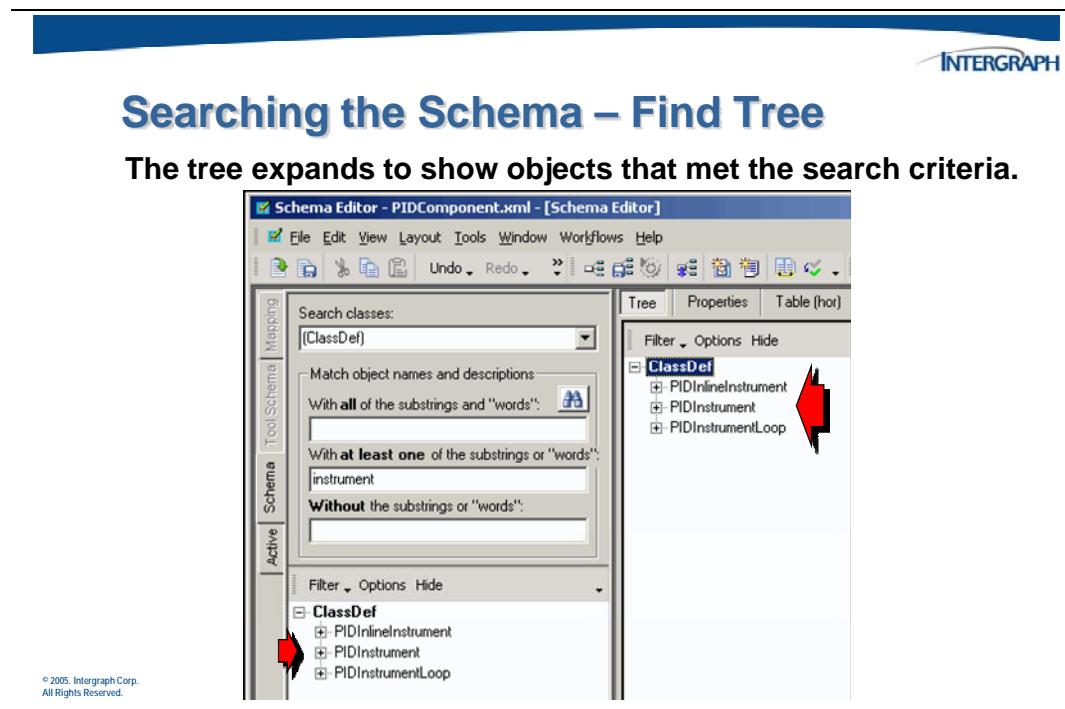
- ❑ Click **Find**  to search for objects matching your criteria in the active file.
- Search results appear in a tree view under the search criteria. Nodes are organized by the type of results, such as *ClassDef* and *InterfaceDef*.
- ❑ Select an object in the tree view to view more information about that object on the tabs in the right pane. To switch between tabs, click the tab name at the top of the **Standard Workflows** window.

The tree displays the results of the search. Expand nodes in the tree to see related edges.

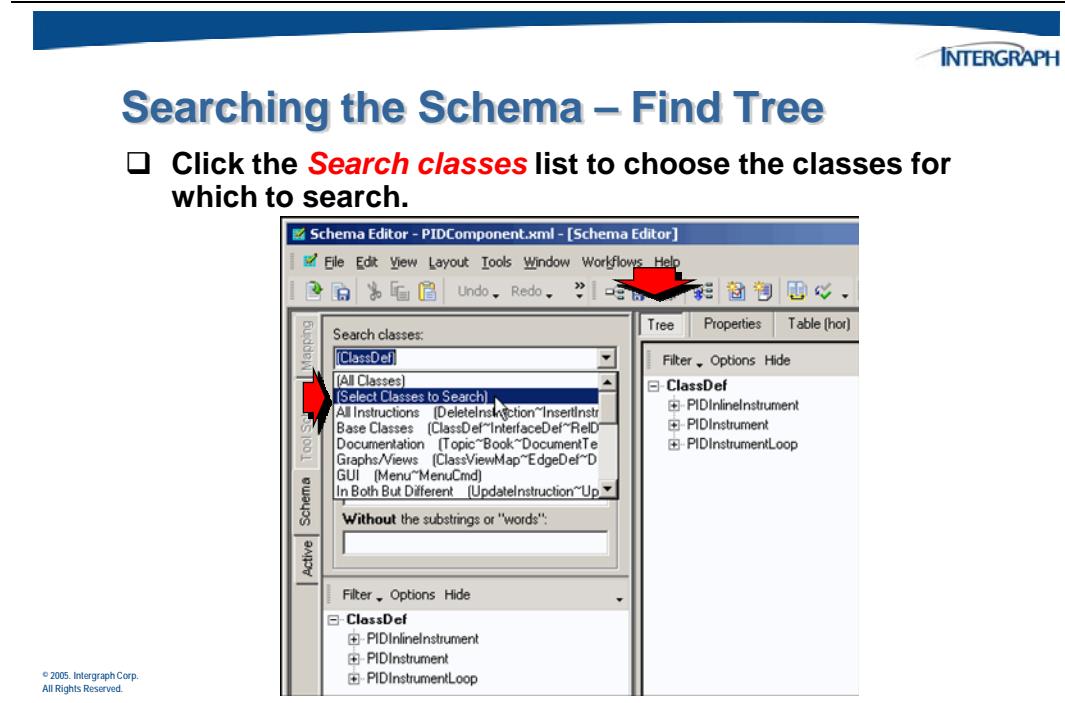


Click on the **ClassDef** heading to expand the tree contents in the right pane.

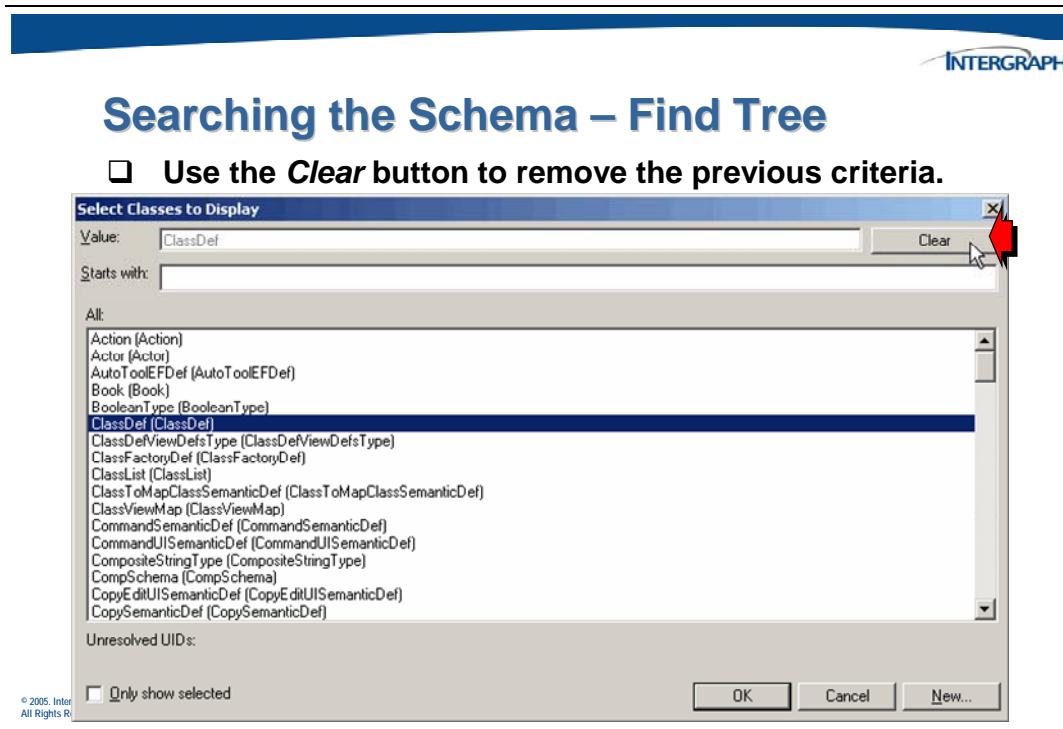
Continue to drill down in the tree view.



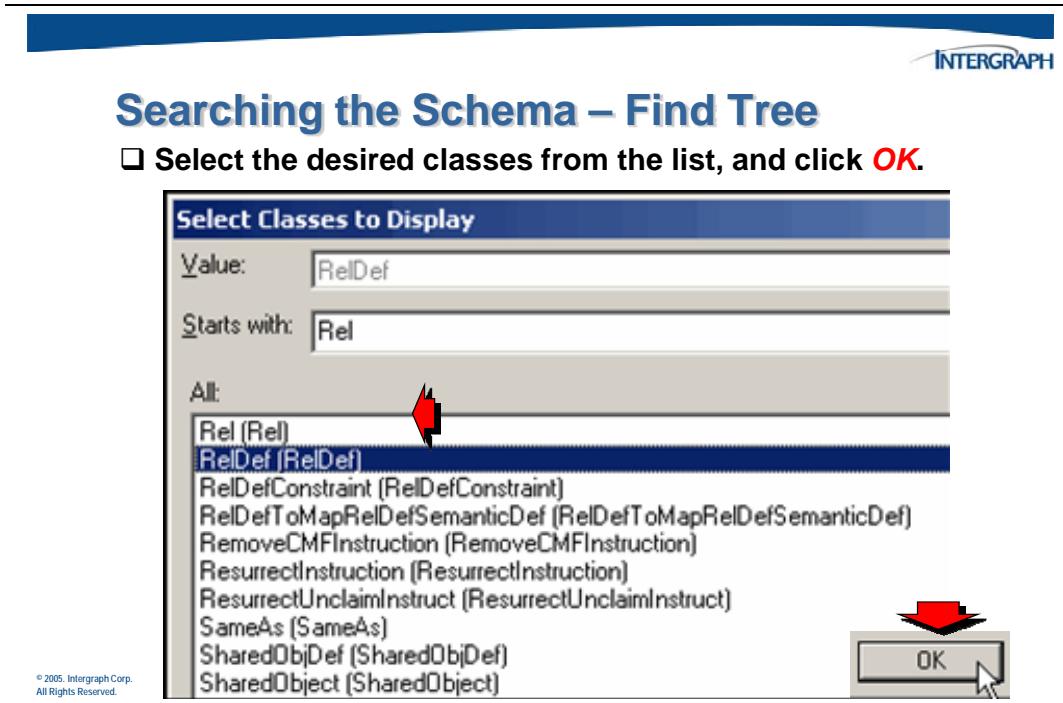
Use the **Search classes** list button to select a new class of object to search.



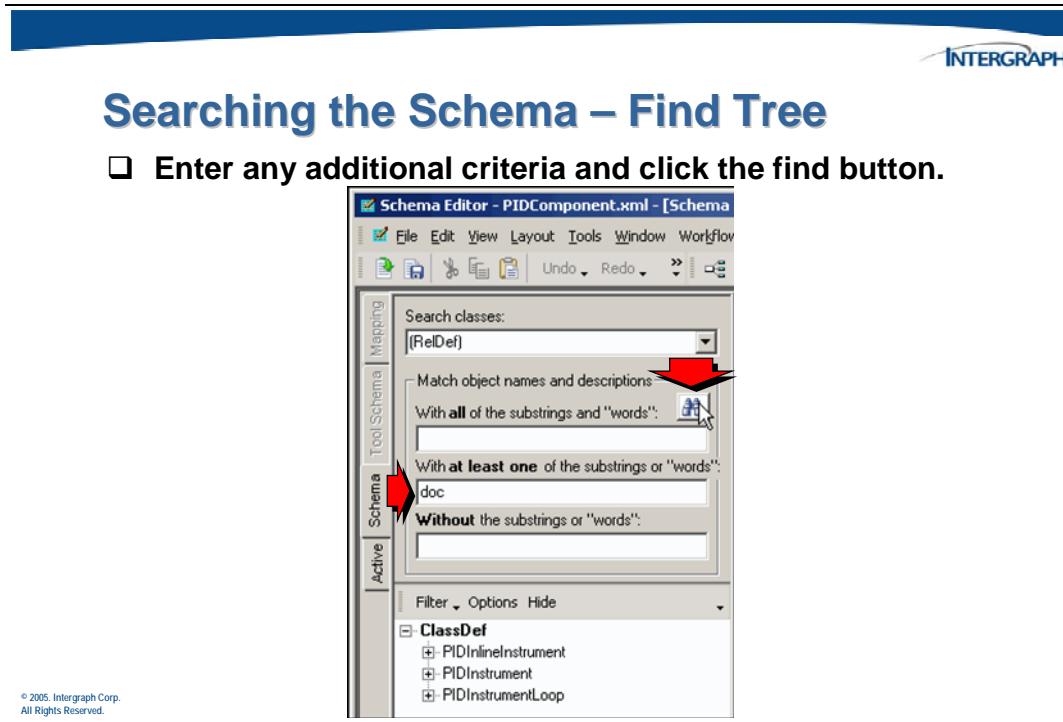
The *Search Classes to Display* dialog will appear.



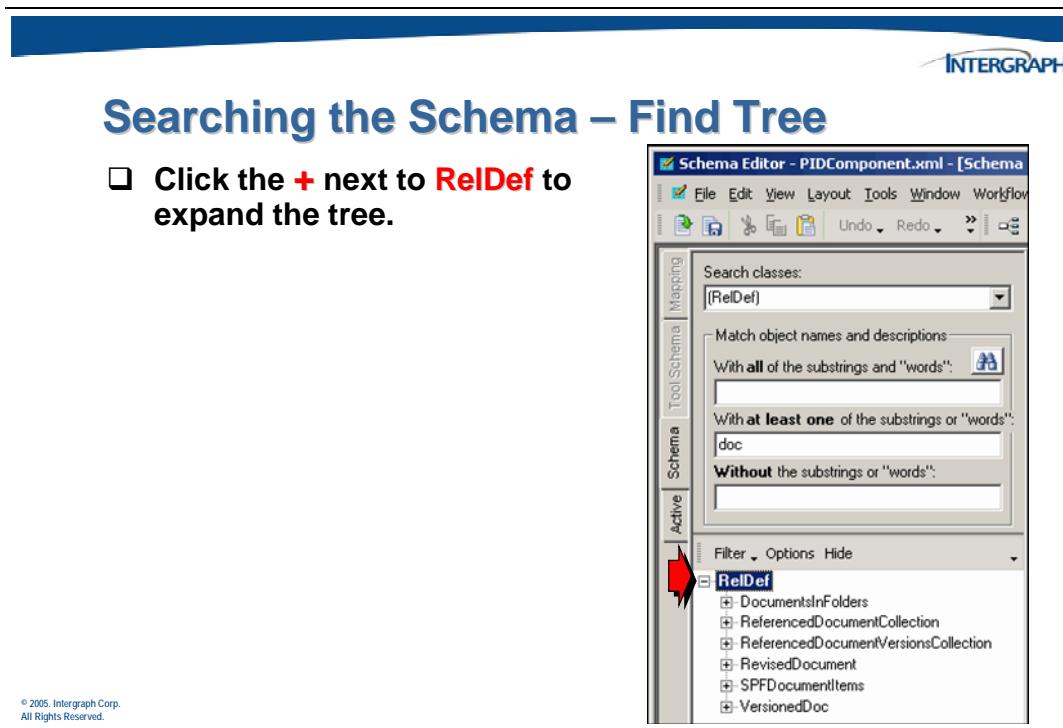
From the displayed list, select the **RelDef** object class.



In this example, the search is limited to the **RelDef** object class and any name that contains the phrase **doc**.

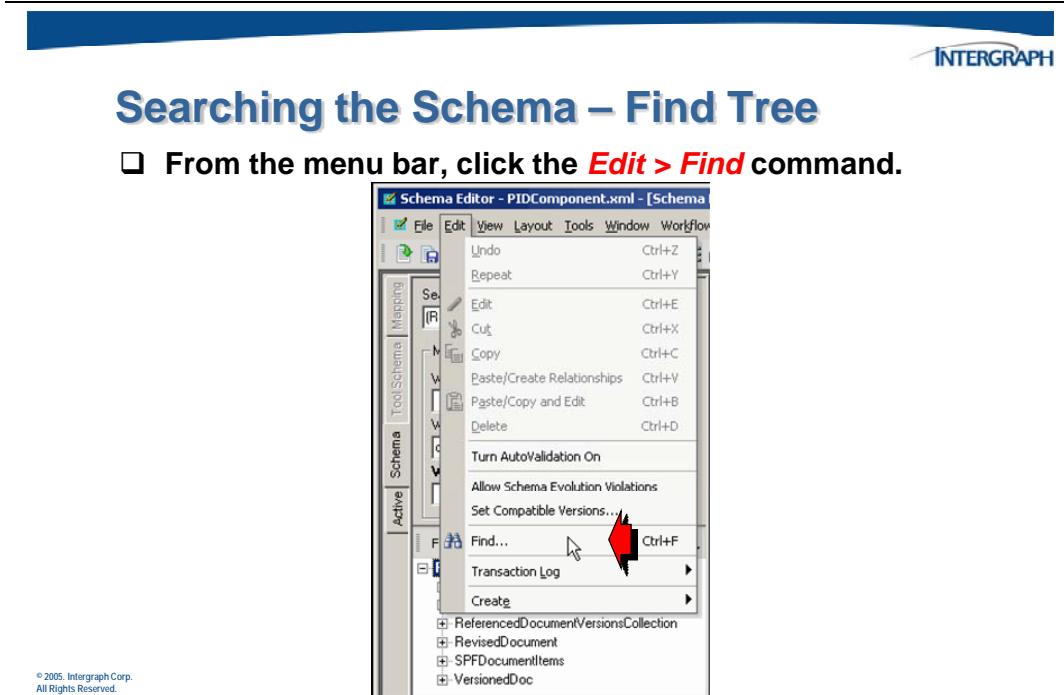


The tree will display the search results. Again, expand nodes in the tree to see related edges.

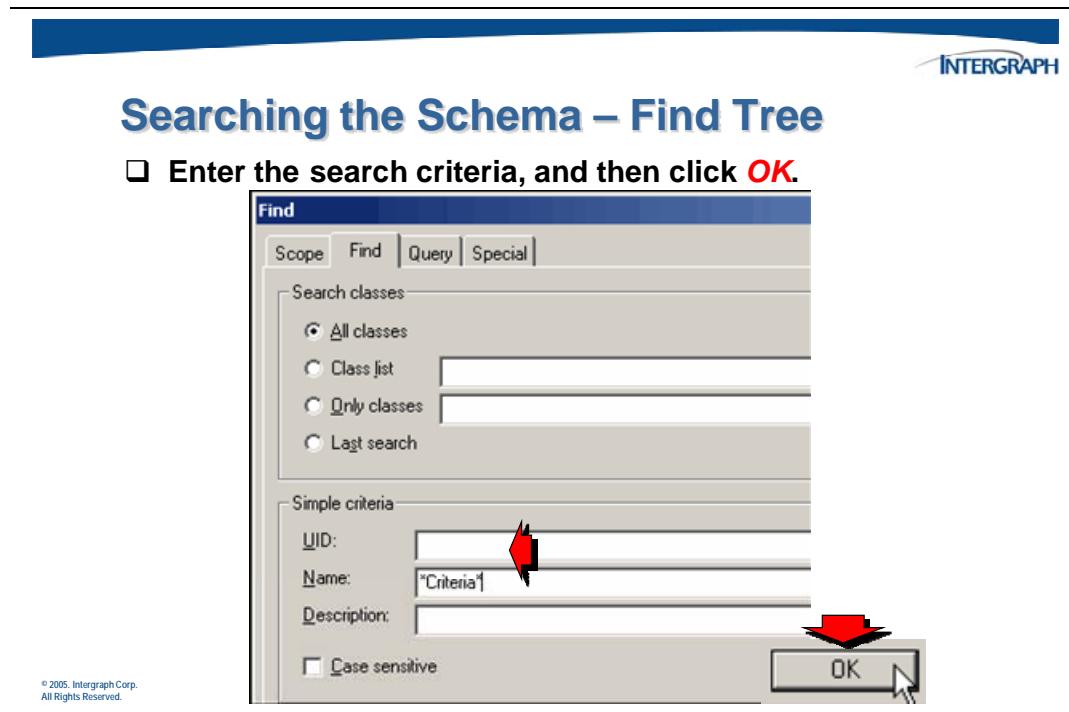


2.3.2 Finding Objects from the Workflows Dialog Box

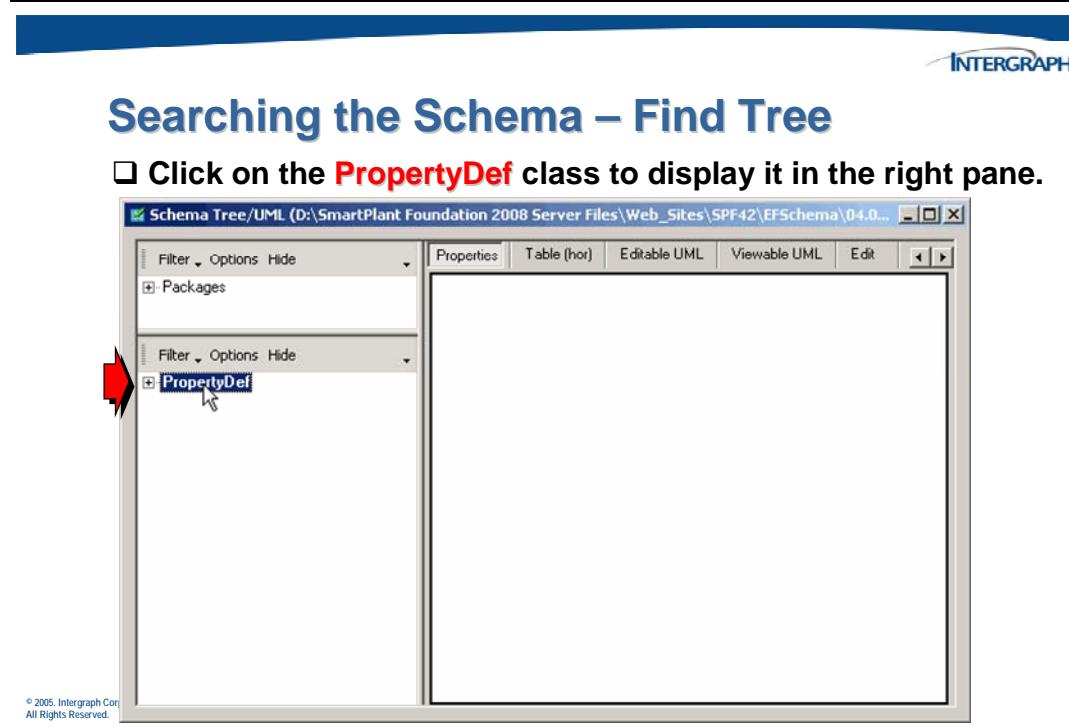
An alternate method for searching for schema objects is to use the **Find** button in the *Workflows* dialog box.



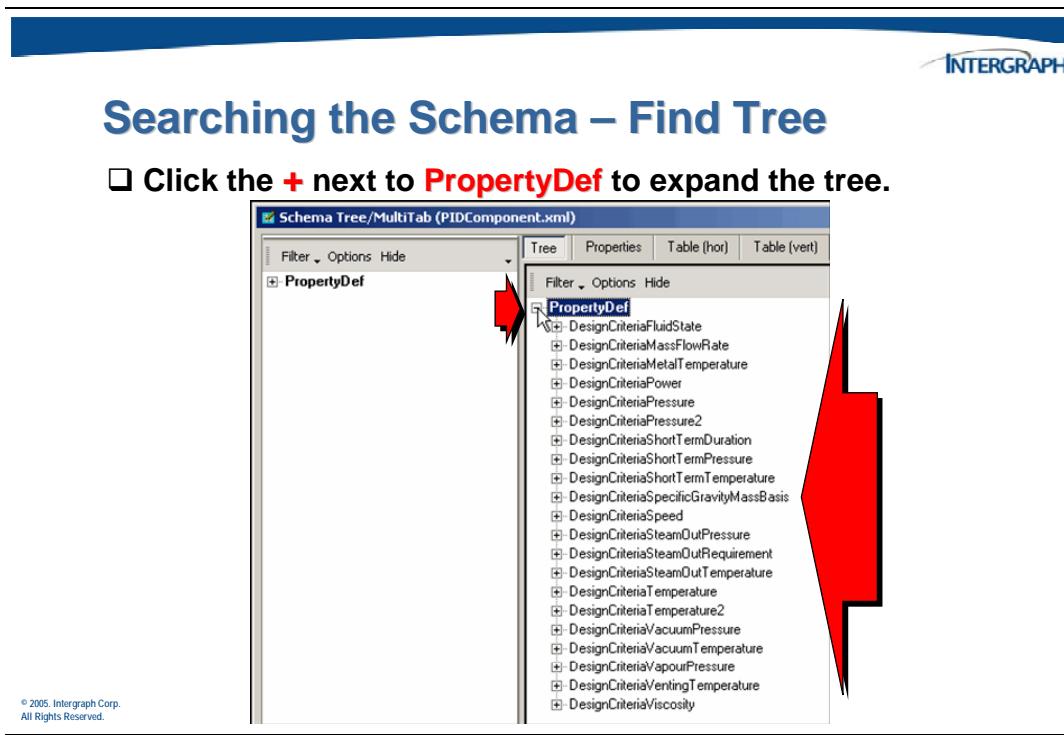
A *Find* dialog box appears.



The tree will initially display the results of the find operation.



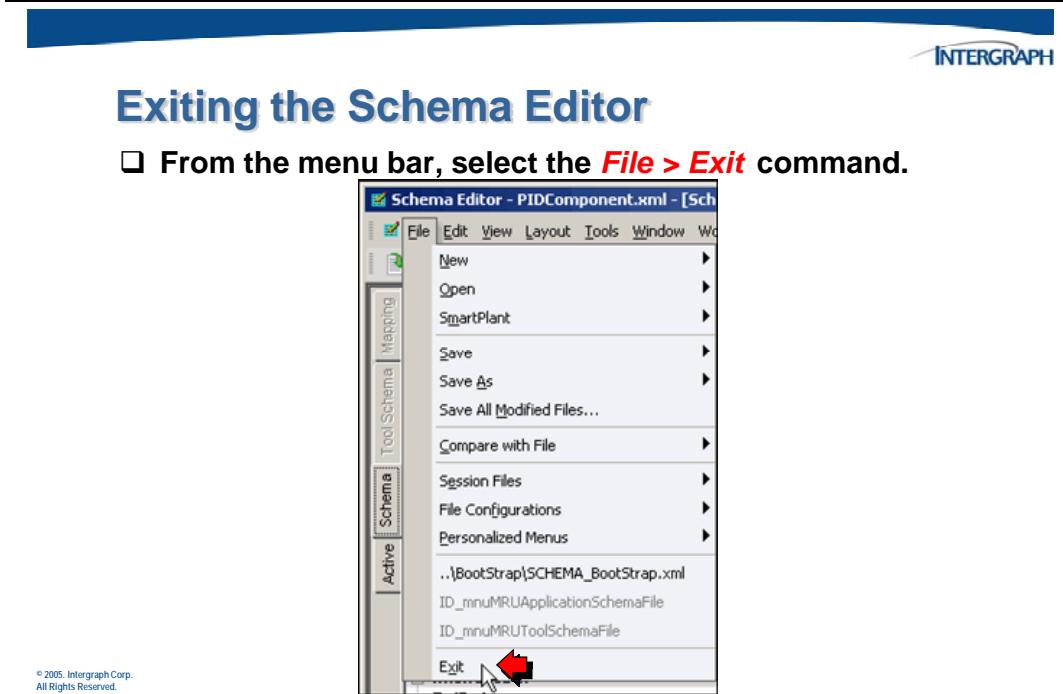
The **Tree** view displays the search results.



2.3.3 Exiting the SmartPlant Schema Editor

Since the *PIDComponent* schema is used as a read-only file, in order to save custom views, exit the Schema Editor and re-start it with a different schema file.

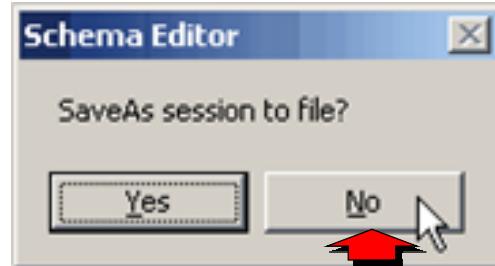
Once the view windows have been closed, use the *File* command on the main menu to *Exit* from the Schema Editor.



A *Schema Editor* dialog window will display to allow you to save a session file. For now, we will not save session files, but they will be covered in depth in later chapters.

Exiting the Schema Editor

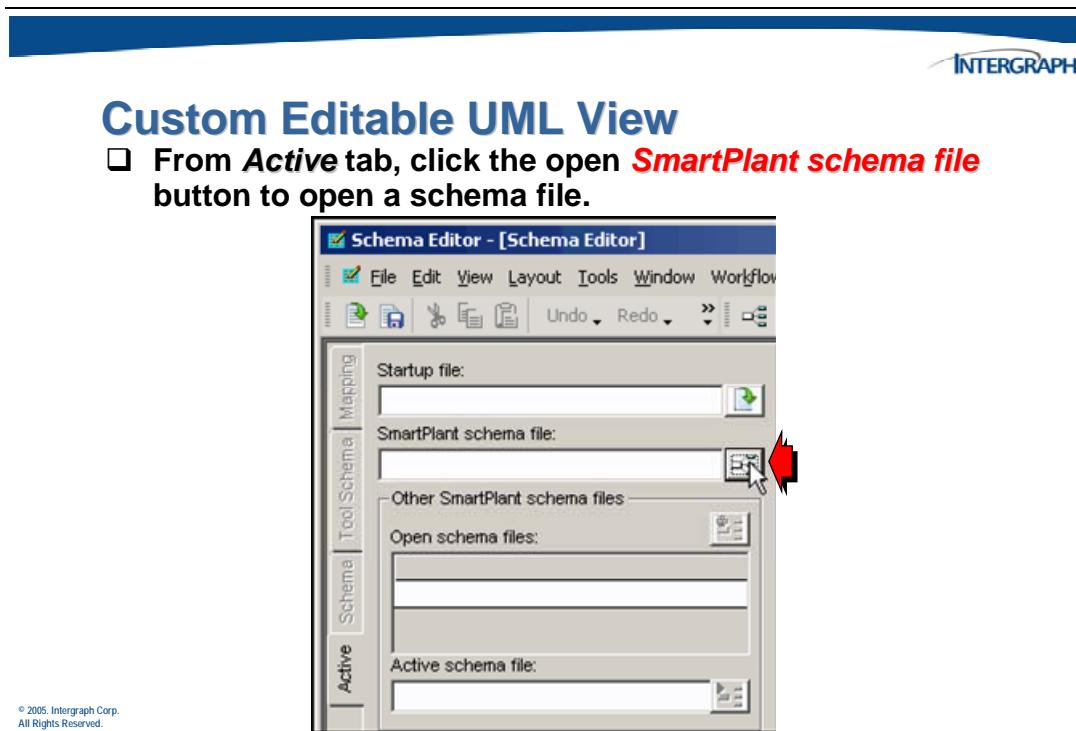
- Click **No** to save the session configuration information to a file.



2.4 Custom Editable UML View

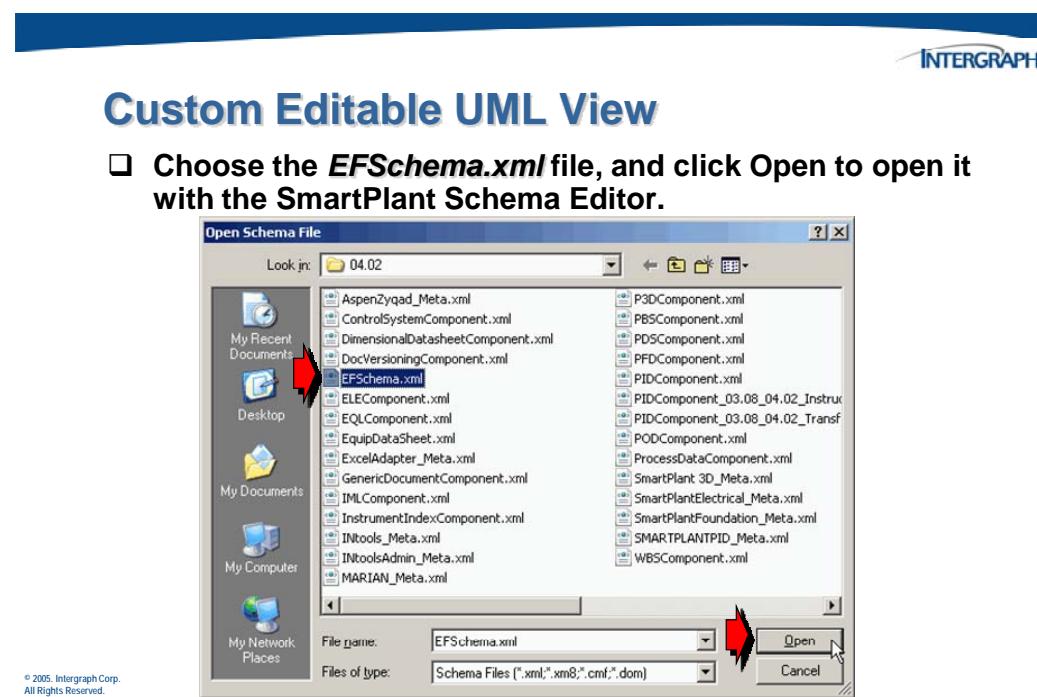
You can search for an object and then view the results in a Tree Drag-Drop UML diagram. You can create your own UML views, name them, and store them as part of a package. In the following example, a custom view will be defined to show the search results.

Since the *PIDComponent* schema is used as a read-only file, to save a custom editable view, start the Schema Editor and open the SmartPlant master schema file, **EFSchema.xml**.

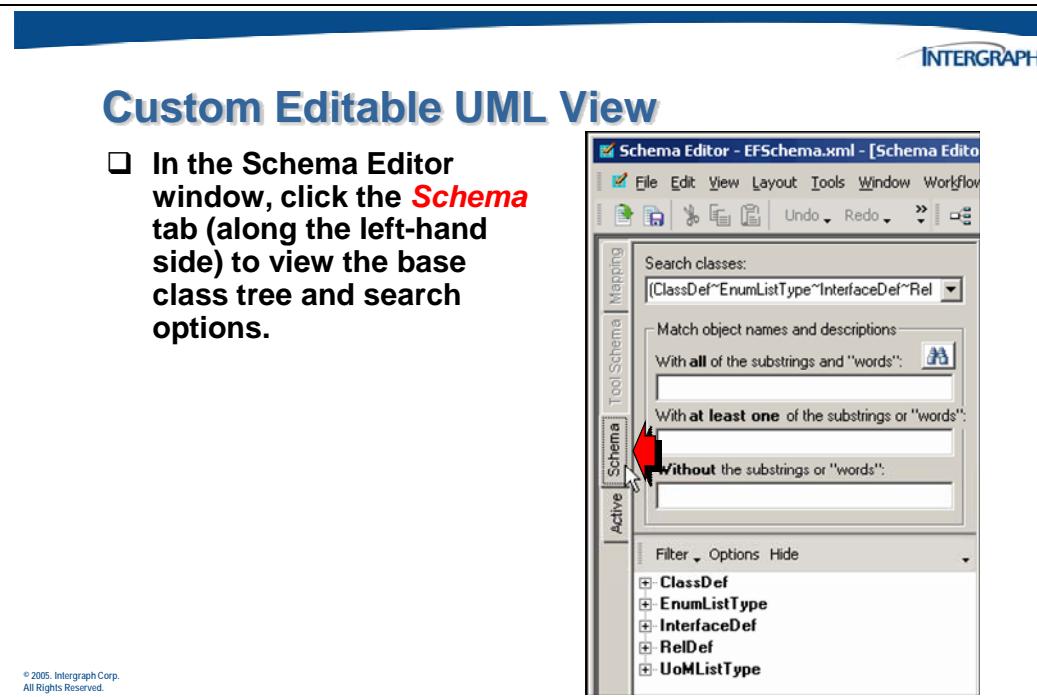


In the following example, the relationship between **Instruments** and **Signal Ports** as defined in the schema, will be displayed graphically using UML.

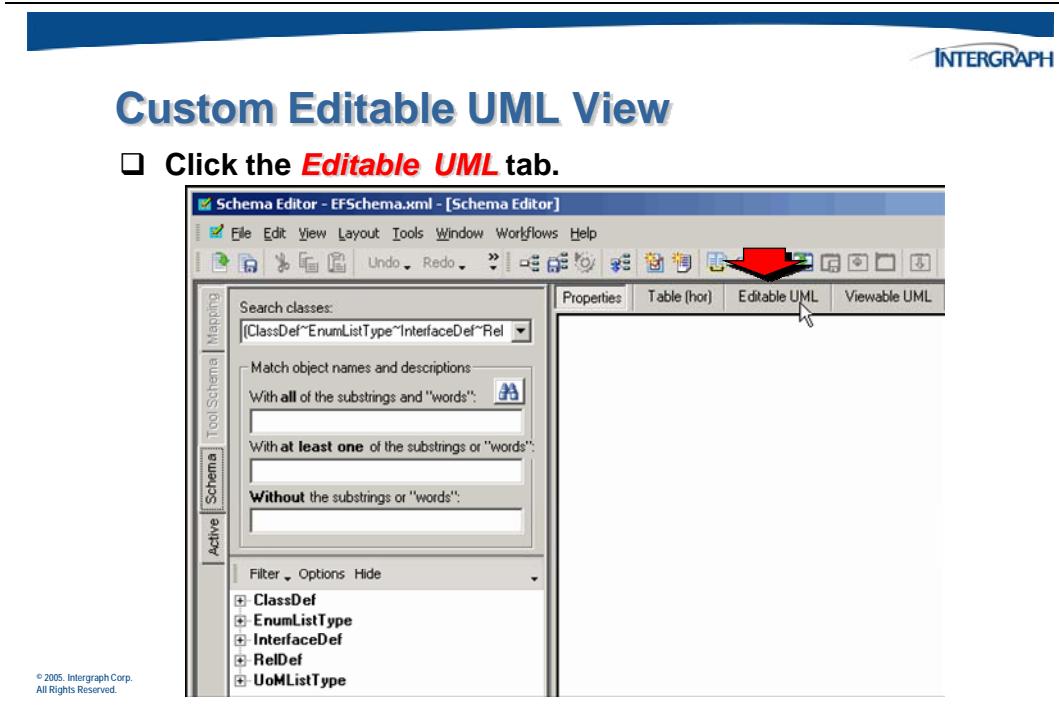
Open the following file: **D:\SmartPlant Foundation 2008 Server Files\Web_sites\SPF42\EFSchema\04.02\EFSchema.xml.**



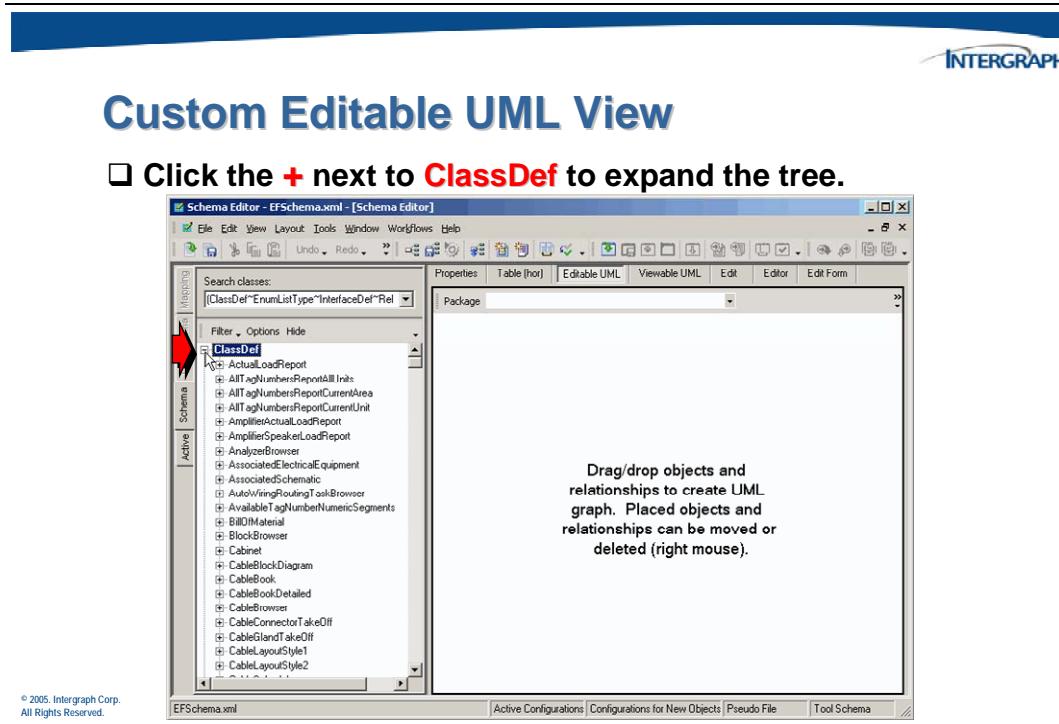
Switch to the **Schema** tab after the *EFSchema.xml* has been opened.



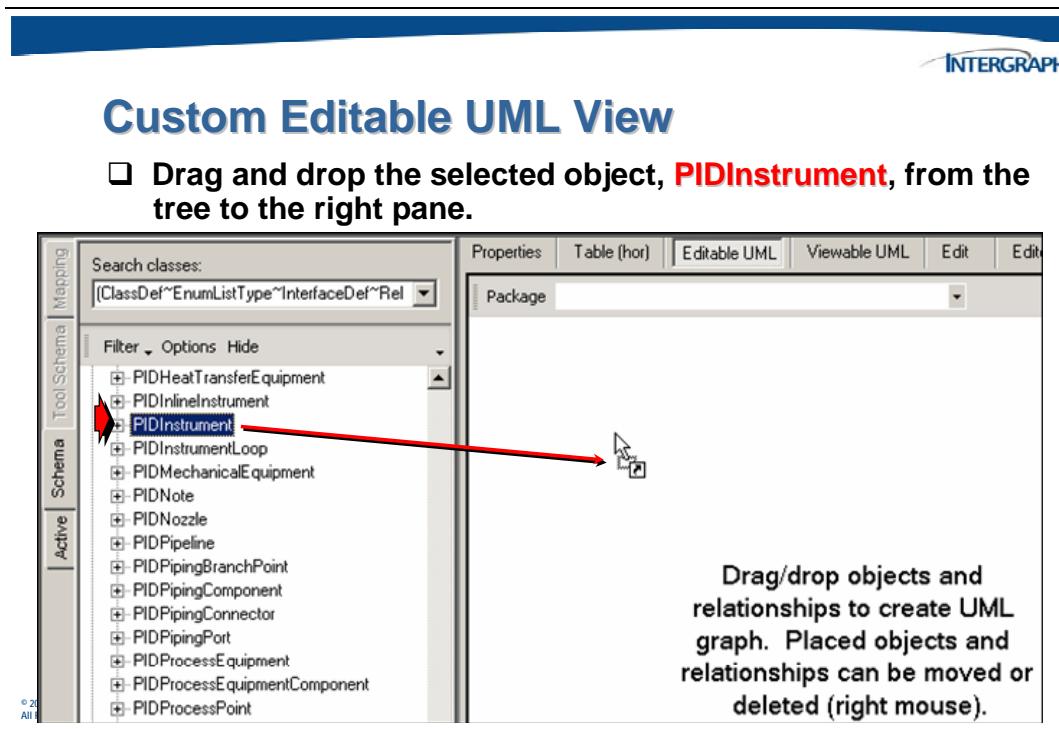
Once the schema has been opened, select the viewing format using the tabs across the top of the window.



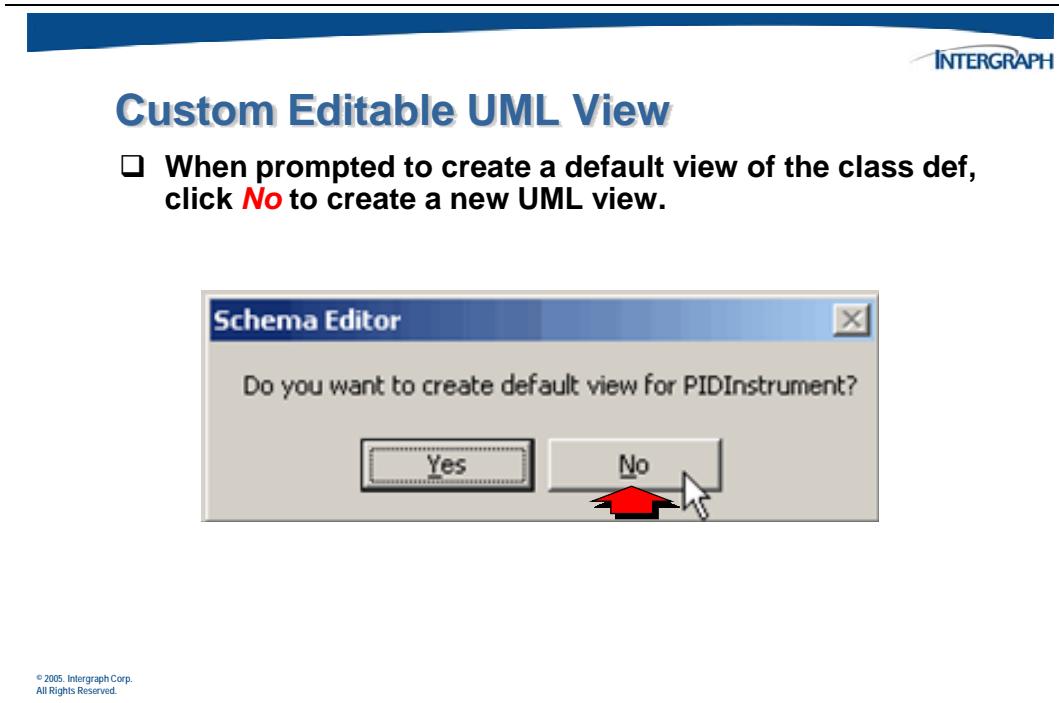
Select the **Editable UML** tab in the right view pane.



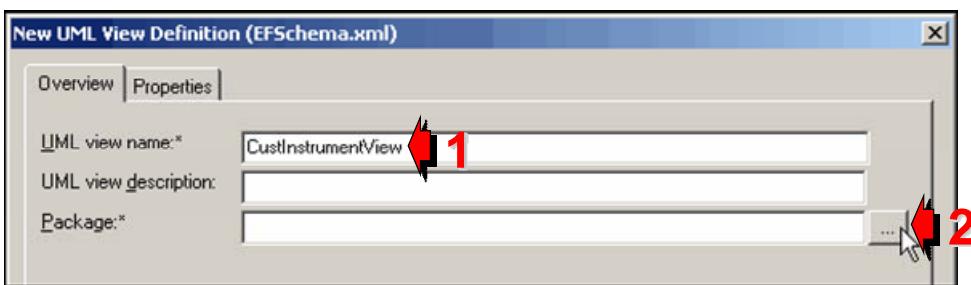
Select the object to be dropped into the UML view.



The **Schema Editor** dialog box appears. Click **No** to create a customized UML view.



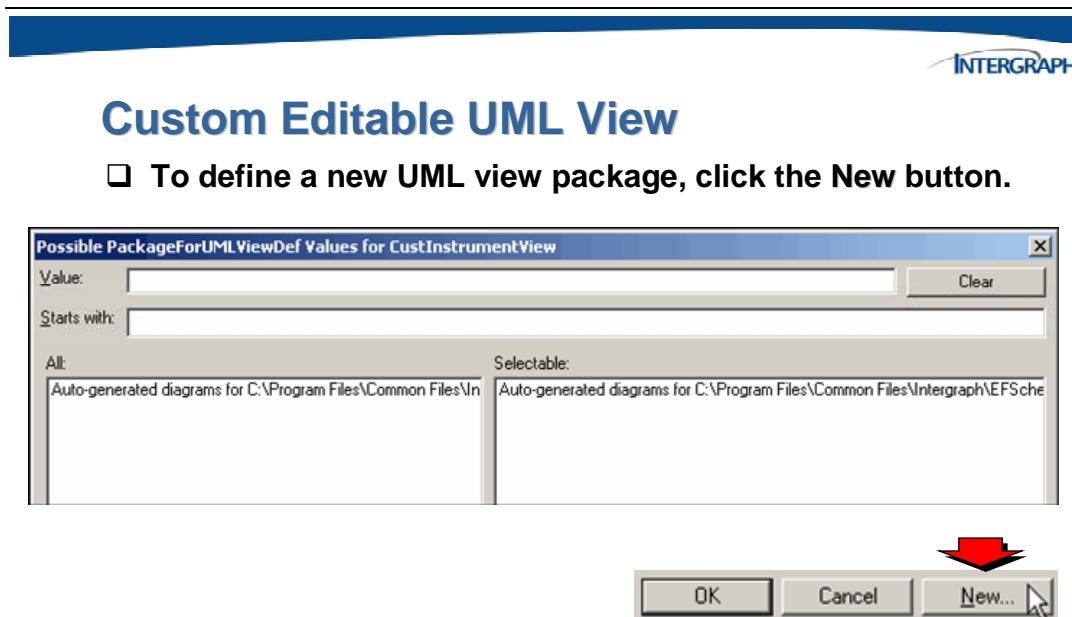
Next, the *New UML View Definition* dialog appears.



The following defines the fields used in the *New UML View Definition* dialog box:

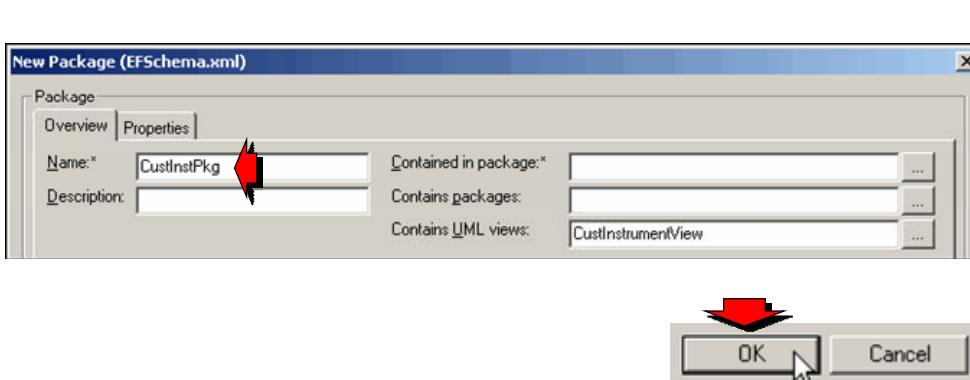
- ❑ **UML view name** – Specifies the name for the new custom UML view. This is a required field.
- ❑ **UML view description** – This is an optional field used to give the new view a description.
- ❑ **Package** – Use the browse button beside this field to select an existing package in which to store the UML view. You can also create a new package to use in storing your custom UML view.

The **Possible PackageForUMLViewDef Values** dialog will appear.



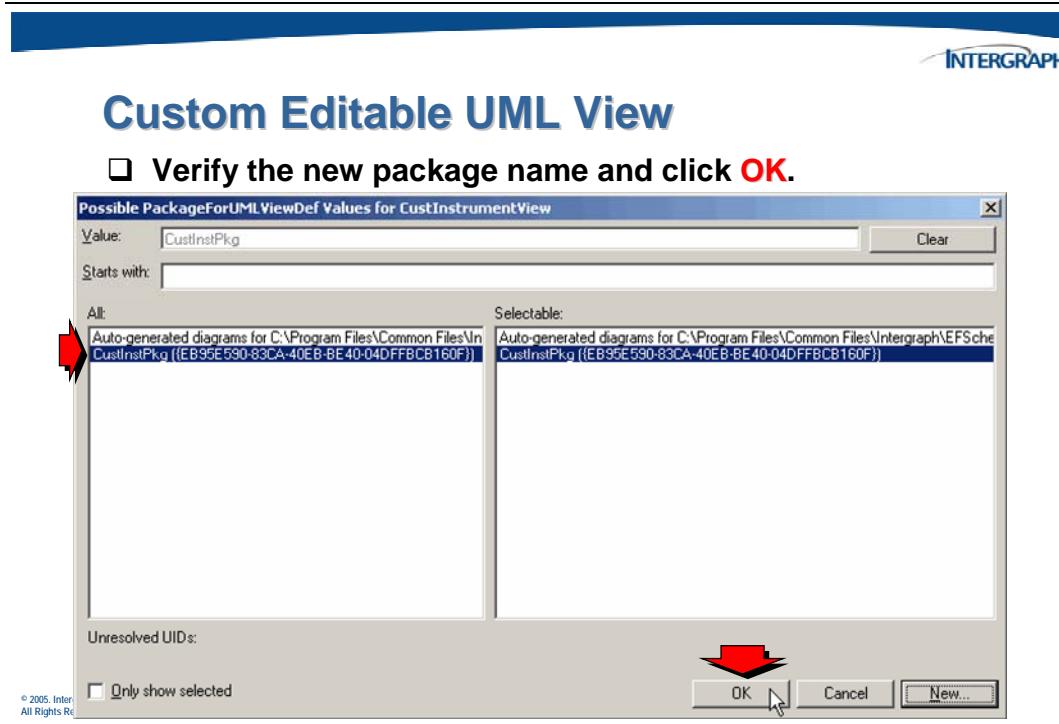
© 2005, Intergraph Corp.
All Rights Reserved.

The **New Package** dialog box appears.

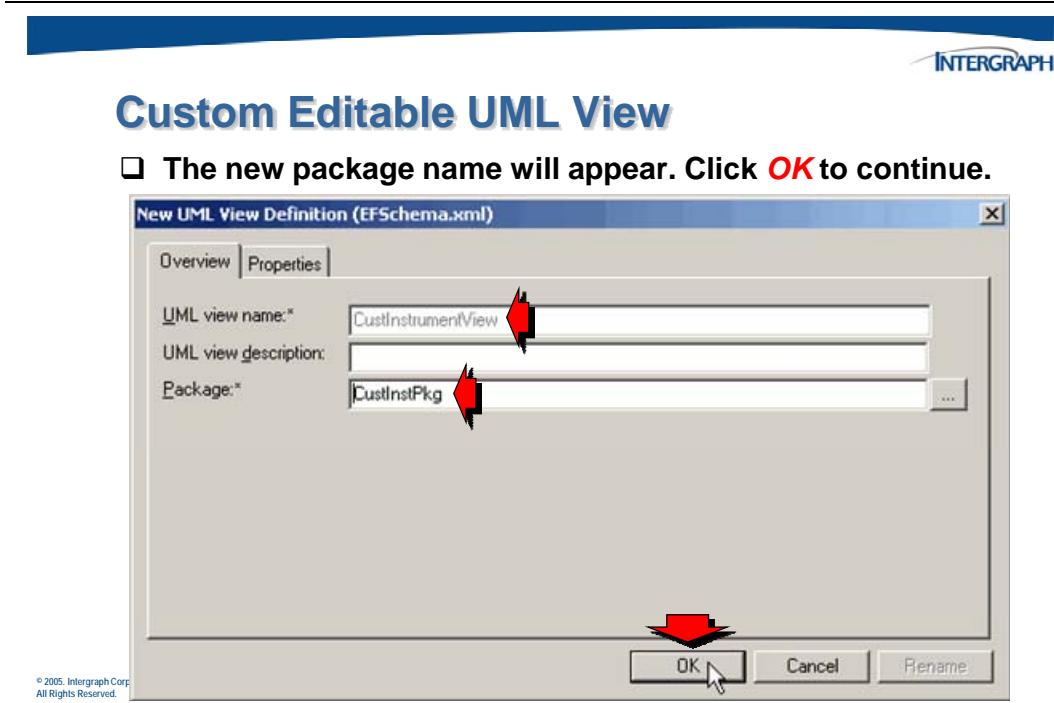


© 2005, Intergraph Corp.
All Rights Reserved.

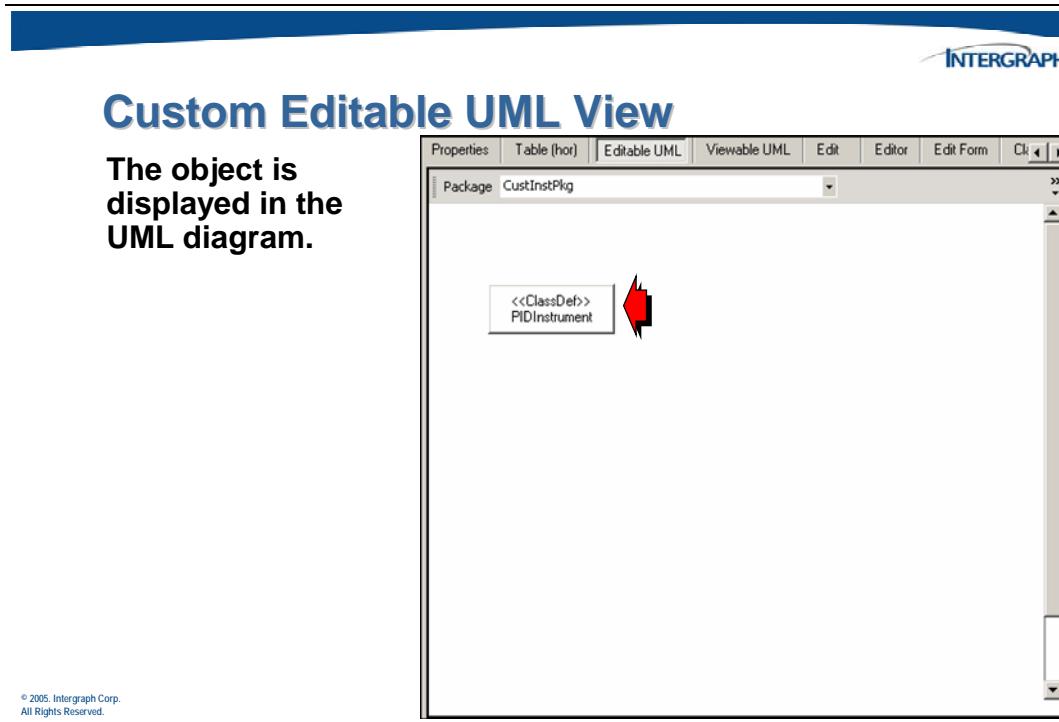
The new package appears in the **Possible PackageForUMLViewDef Values** dialog box and is automatically highlighted.



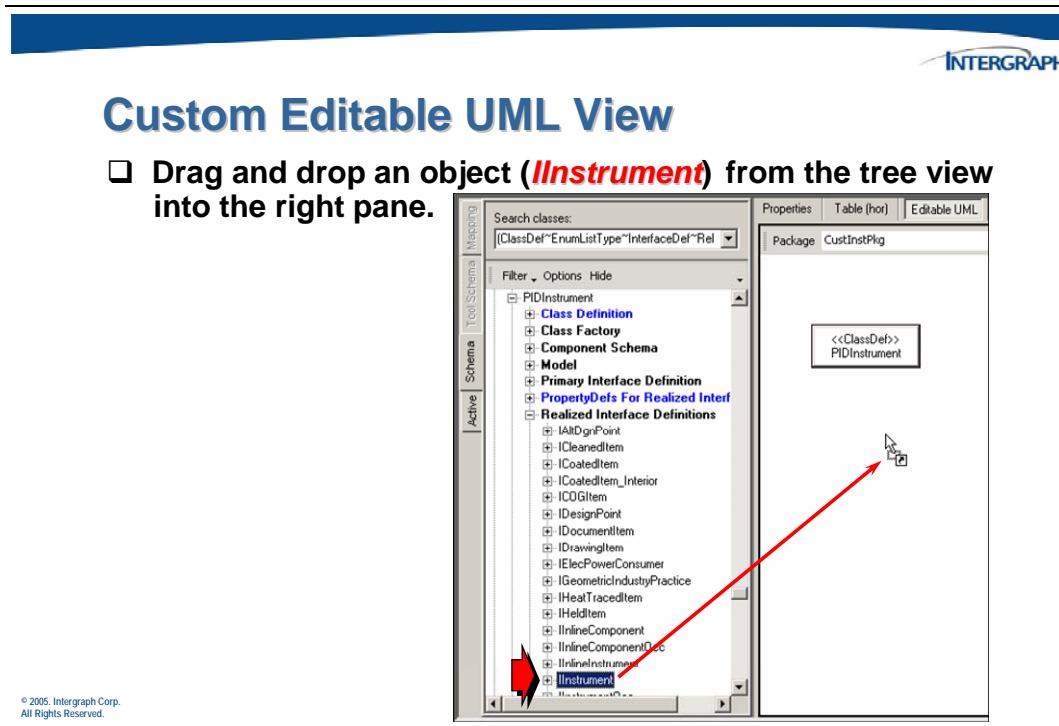
In the **New UML View Definition** dialog box, the new package name appears in the **Package** field.



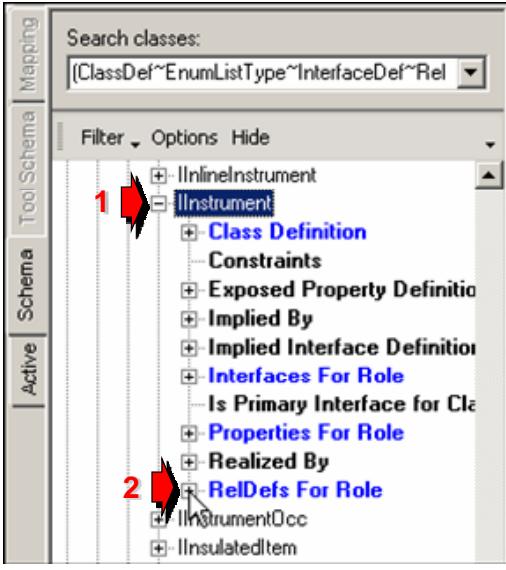
The dropped object, *PIDInstrument*, appears in the custom *CustInstrumentView* UML view.



Expand the **PIDInstrument ClassDef**, and click the + next to **Realized Interface Definitions** to see the related objects.



Expand nodes in the tree to see other objects.



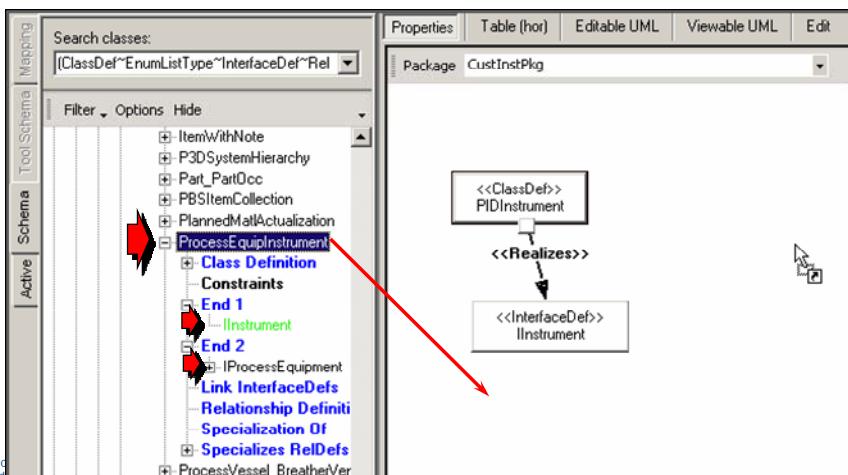
The screenshot shows the 'Custom Editable UML View' interface. On the left is a tree view with the following structure:

- Search classes: (ClassDef~EnumListType~InterfaceDef~Rel)
 - + IInlineInstrument
 - + **Instrument** (selected, highlighted in blue)
 - + Class Definition
 - + Constraints
 - + Exposed Property Definition
 - + Implied By
 - + Implied Interface Definition
 - + Interfaces For Role
 - + Is Primary Interface for Class
 - + Properties For Role
 - + Realized By
 - + RelDefs For Role
 - + InstrumentOcc
 - + InsulatedItem

Two red arrows point to the 'Instrument' node (arrow 1) and the 'RelDefs For Role' node (arrow 2).

© 2005, Intergraph Corp.
All Rights Reserved.

Select the next object to be dropped into the UML view.



The screenshot shows the 'Custom Editable UML View' interface. On the left is a tree view with the following structure:

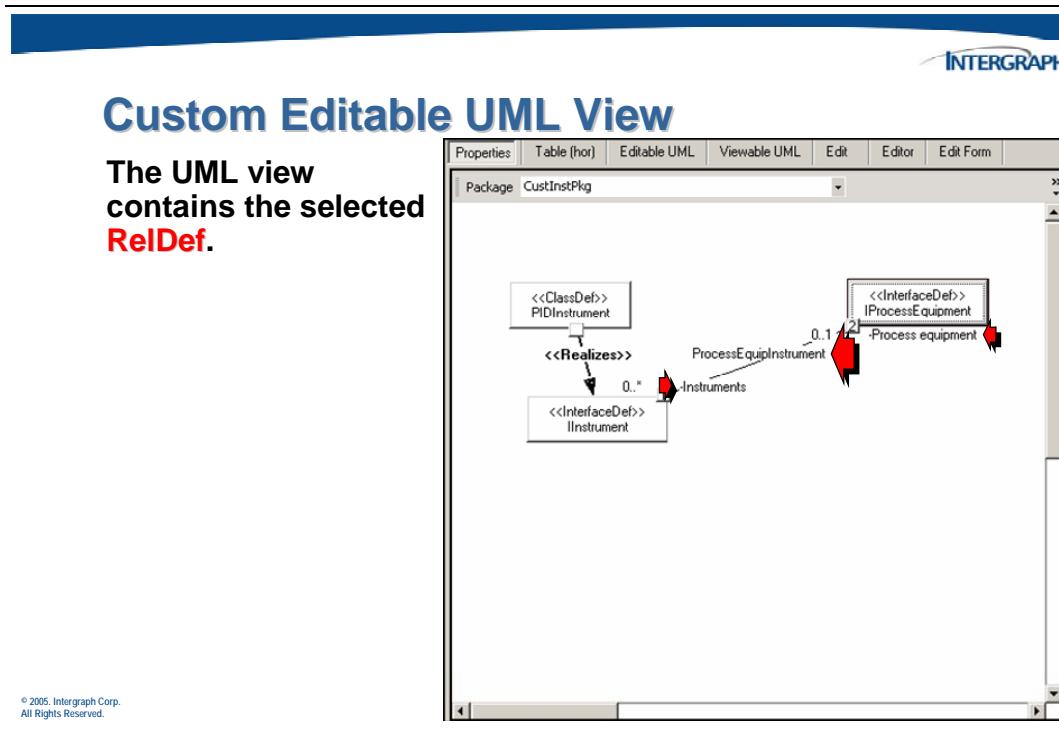
- Search classes: (ClassDef~EnumListType~InterfaceDef~Rel)
 - + ItemWithNote
 - + P3DSystemHierarchy
 - + Part_PartOcc
 - + PBSItemCollection
 - + PlannedMallActualization
 - + **ProcessEquipInstrument** (selected, highlighted in blue)
 - + Class Definition
 - + Constraints
 - + End 1
 - + End 2
 - + IProcessEquipment
 - + Link InterfaceDefs
 - + Relationship Definition
 - + Specialization Of
 - + Specializes RelDefs
 - + ProcessVessel_BreatherVer

A red arrow points from the 'ProcessEquipInstrument' node in the tree view to the 'Realizes' relationship in the UML diagram area.

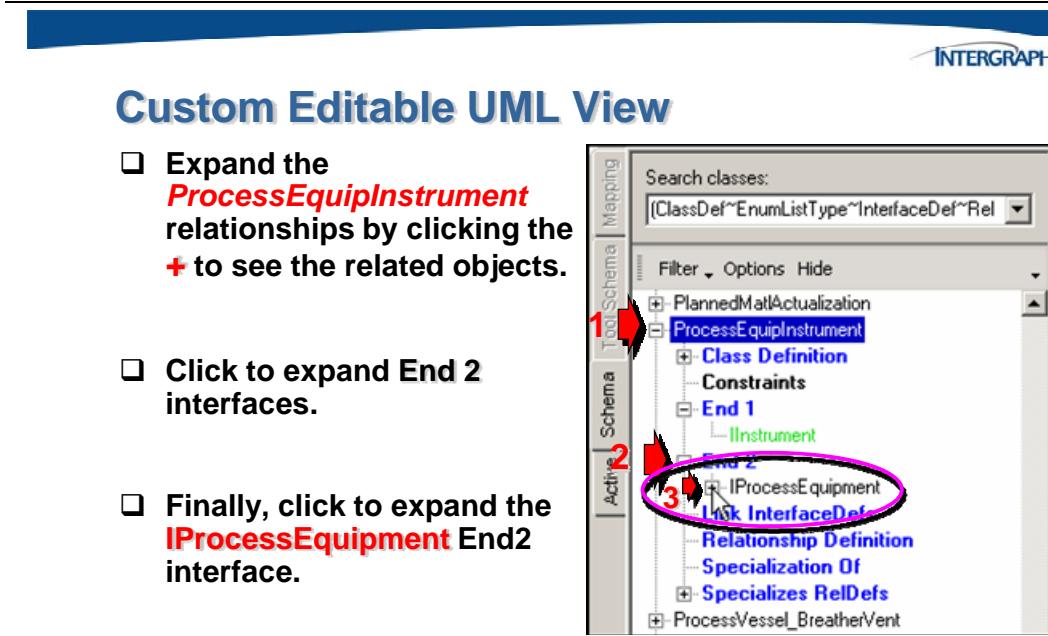
The UML diagram area shows a package named 'CustInstPkg'. It contains two classes: 'PIDInstrument' (represented by a rectangle with 'PIDInstrument' inside) and 'Instrument' (represented by a rectangle with 'Instrument' inside). A directed association labeled '<<Realizes>>' connects 'PIDInstrument' to 'Instrument'.

© 2005, Intergraph Corp.
All Rights Reserved.

Expand nodes in the tree then select the object then drag/drop them into the UML view.



By dragging and dropping objects from the tree, you are able to define your custom UML view objects and their relationships.



Expand nodes in the tree to show the related edges.

Custom Editable UML View

- Expand the **RelDefs For Role** relationships by clicking the + to see the related objects.
- Expand the **SignalPorts** relationship by clicking the + to see the related objects.

The screenshot shows the INTERGRAPH schema editor interface. On the left is a tree view of schema classes under the 'Mapping' tab. A red arrow points to the '+' sign next to the 'RelDefs For Role' node. Another red arrow points to the '+' sign next to the 'SignalPorts' node. The tree includes nodes like 'Class Definition', 'Constraints', 'End 1', 'End 2', 'IProcessEquipment', 'Link InterfaceDefs', 'Relationship Definition', and 'Specialization Of'. The bottom left corner of the tree view has a copyright notice: '© 2005, Intergraph Corp. All Rights Reserved.'

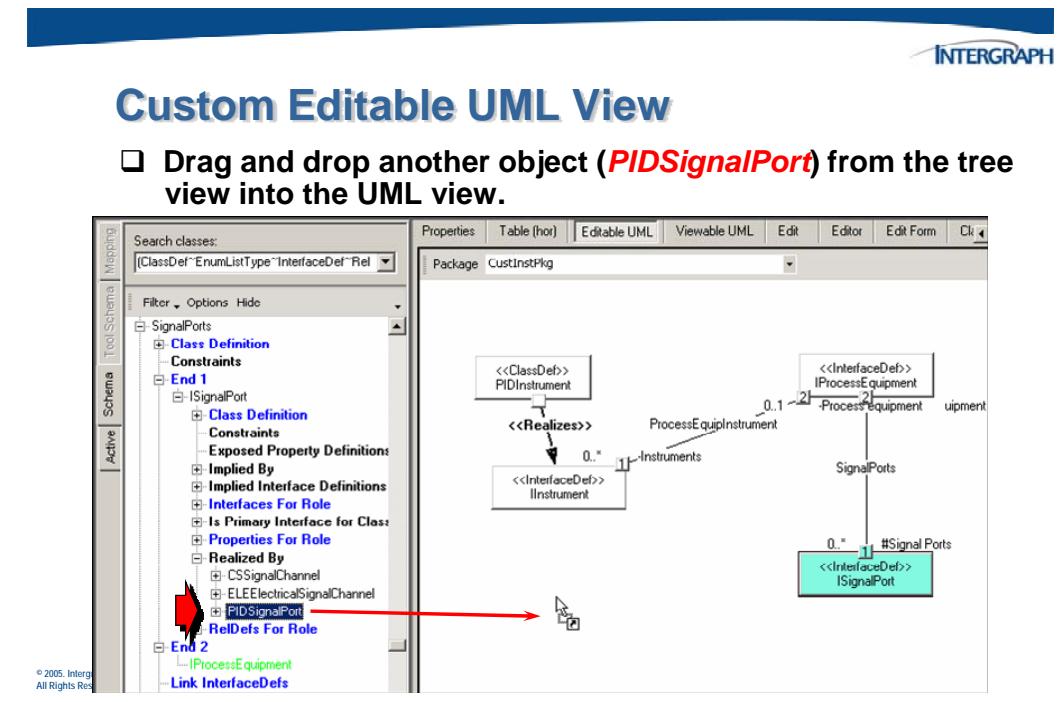
Select the next object to be dropped into the UML view.

Custom Editable UML View

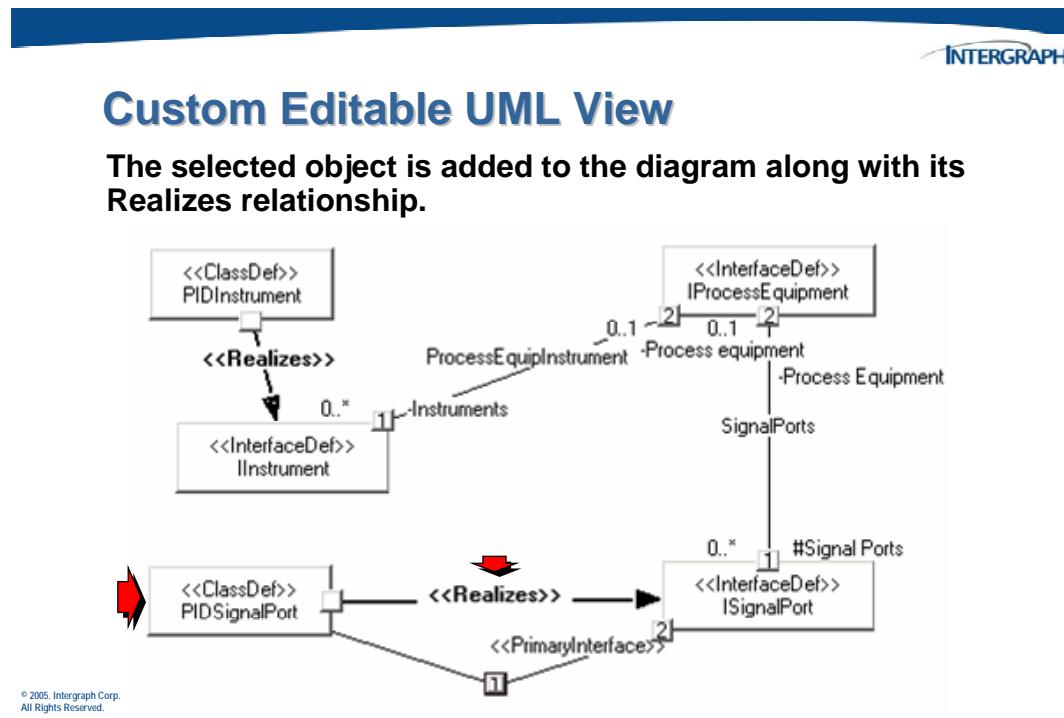
- Drag and drop **ISignalPort** from the tree into the UML view.

The screenshot shows the INTERGRAPH schema editor interface with the UML view open. On the left is the schema browser tree. A red arrow points to the 'ISignalPort' node in the 'SignalPorts' section of the tree. On the right is the UML diagram area. It contains a class 'PIDInstrument' and an interface 'IProcessEquipment'. A relationship 'Realizes' connects them. Another relationship 'Instruments' connects 'PIDInstrument' to 'IInstrument'. A red arrow points from the 'ISignalPort' node in the tree towards the UML diagram area. The bottom left corner of the tree view has a copyright notice: '© 2005, Intergraph Corp. All Rights Reserved.'

The selected object is added to the diagram along with any RelDefs defined between it and any other interfaces on the diagram.



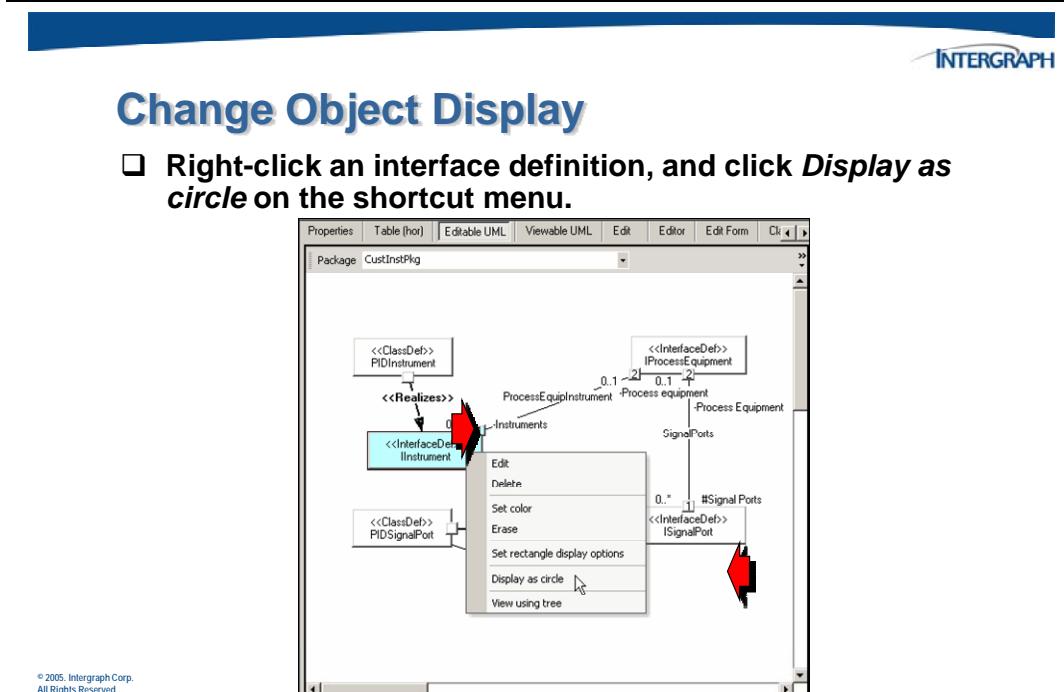
Note that you are building your own custom UML view diagram.



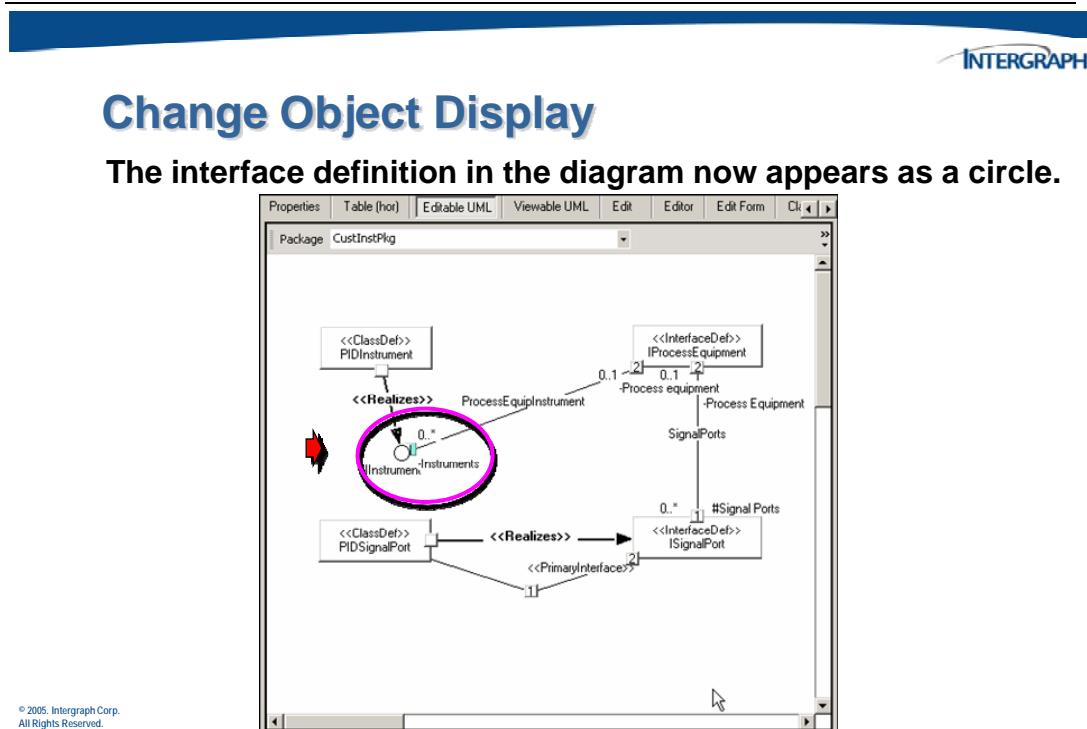
2.5 Change Object Display

By default when you drag and drop an object into a UML view, the dropped object will display as a rectangle. If you have a UML diagram with quite a few objects, the display can become cluttered. An option available to you is to change the display of the object from the default rectangle to a circle.

To do this, right-click an object to display the shortcut menu.



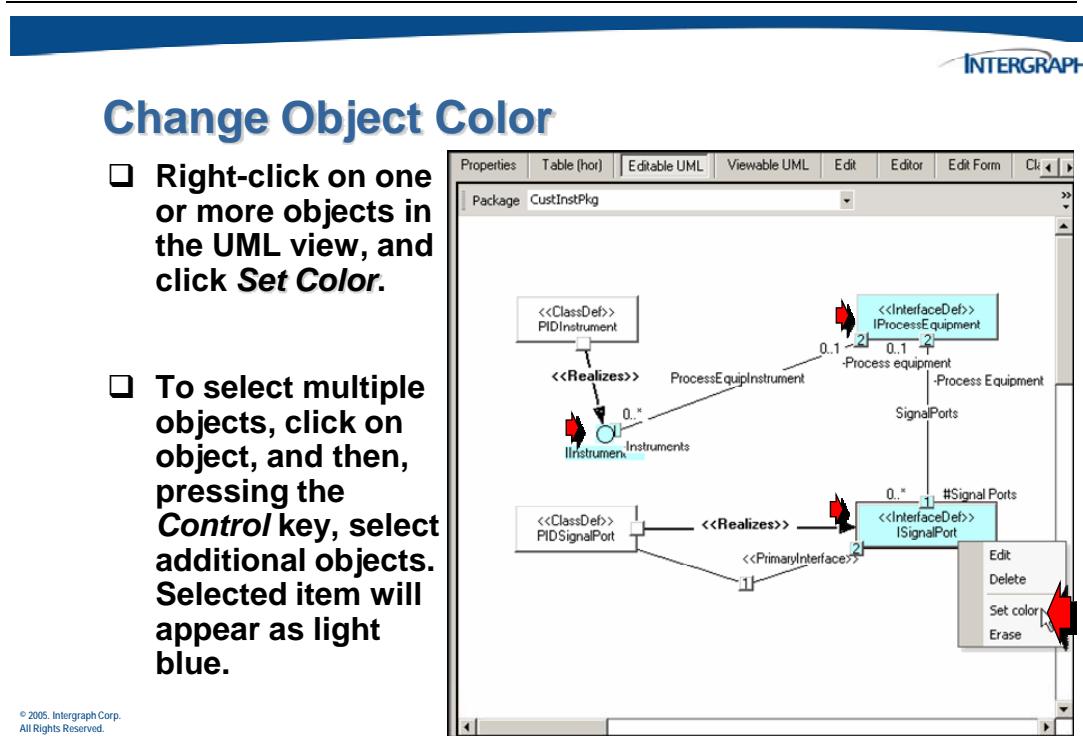
This operation will change the selected object display from a rectangle to a circle.



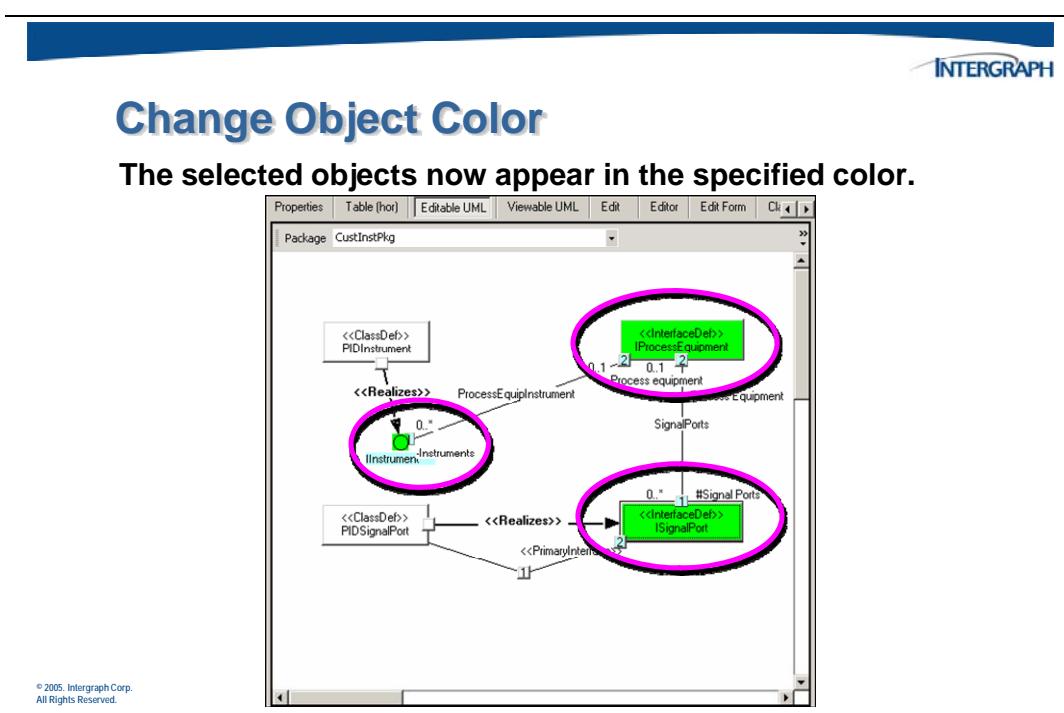
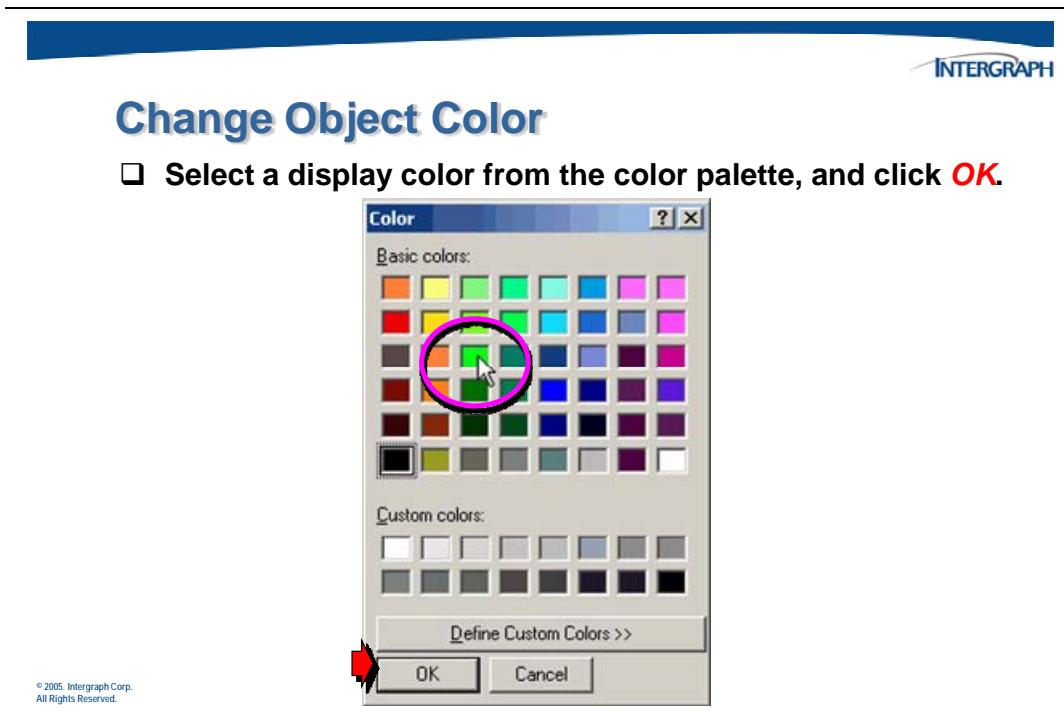
2.6 Change Object Color

To make certain objects easier to see on a UML diagram, you can change the color of the rectangle or circle representing the object.

To do this, right-click an object to display the shortcut menu.



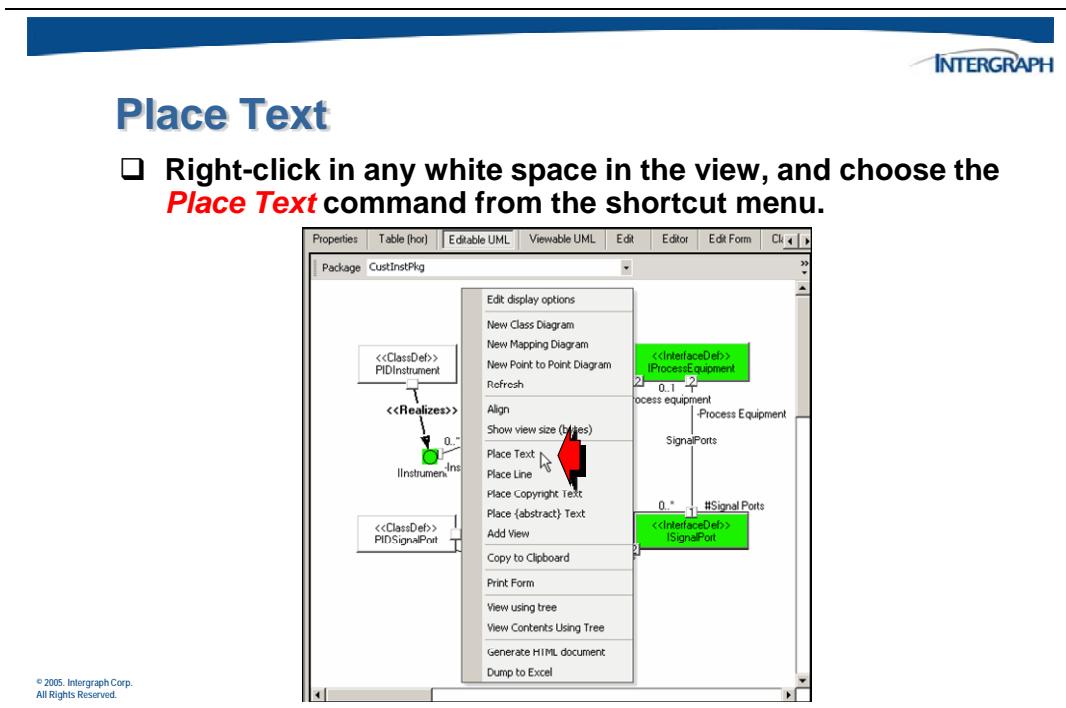
A color palette dialog box appears.



2.7 Placing Text in a View

A UML view diagram can be annotated with text. Use the **Place Text** command from the pop up menu to enter and format any text needed to clarify the diagram.

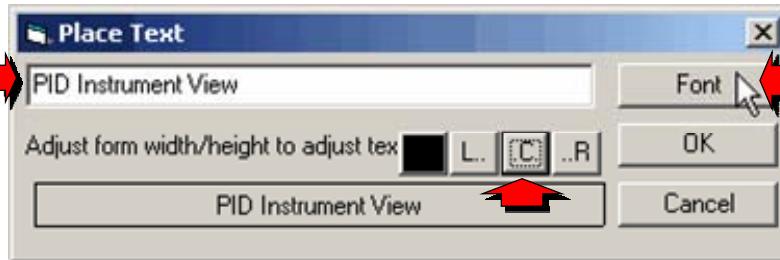
Right-click anywhere in the view background (except on an object) to display the shortcut menu.



A *Place Text* dialog box appears.

Place Text

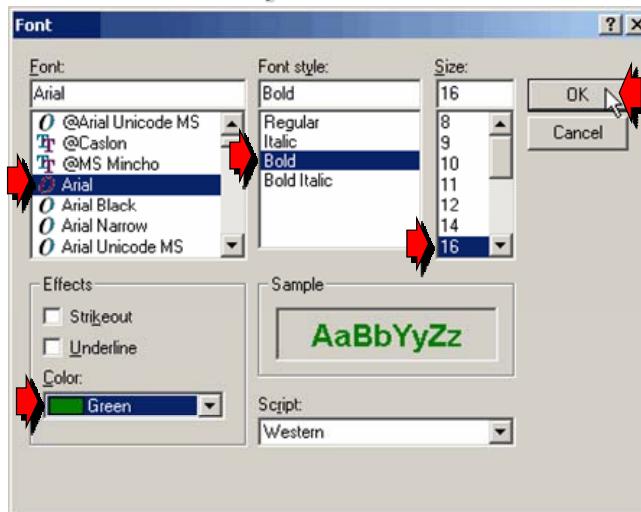
- Enter the desired text, and specify any additional parameters, such as text alignment. Click the **Font** button to access an additional dialog for font settings.


© 2005, Intergraph Corp.
All Rights Reserved.

The **Place Text** dialog box allows you to set the font color and alignment (left, center or right). Click the **Font** button to set the font parameters.

Place Text

- Select a **Font**, **Font Style**, **Size** and **Color**, and then click **OK**.


© 2005, Intergraph Corp.
All Rights Reserved.

Note the approximation of the font display at the bottom of the **Text** dialog box.



Place Text

- Once the parameters are set, click **OK** to return to the UML view.



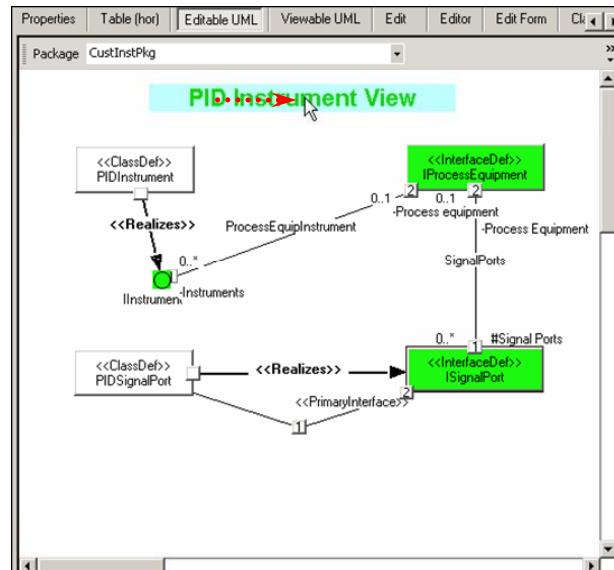
© 2005, Intergraph Corp.
All Rights Reserved.

Use the mouse to position the text to the desired location.



Place Text

- Click and hold the left mouse button to drag the text to the appropriate location.

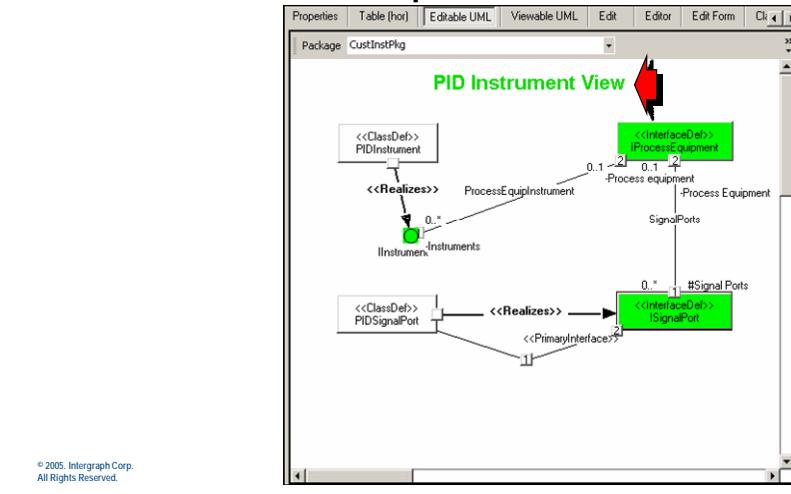


© 2005, Intergraph Corp.
All Rights Reserved.



Place Text

- Position the text in the desired location, and release the mouse button to place it.

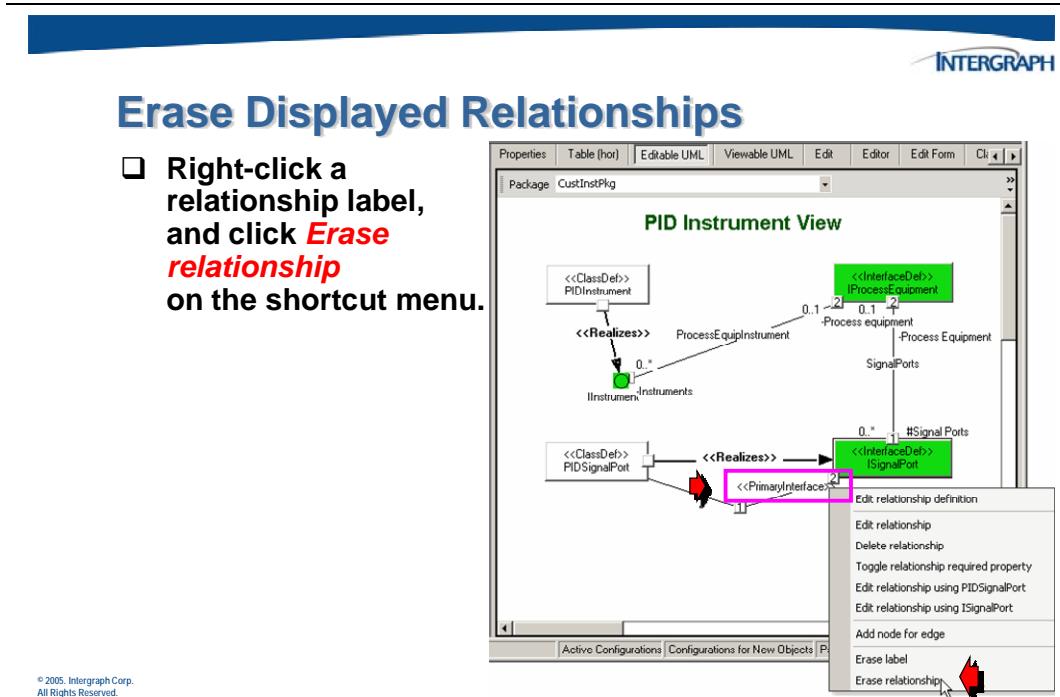


© 2005, Intergraph Corp.
All Rights Reserved.

2.8 Erasing Displayed Relationships

The **Erase Relationship** command allows you to remove a displayed object from a UML view. The erased object remains in the schema, but it is no longer shown on the displayed view.

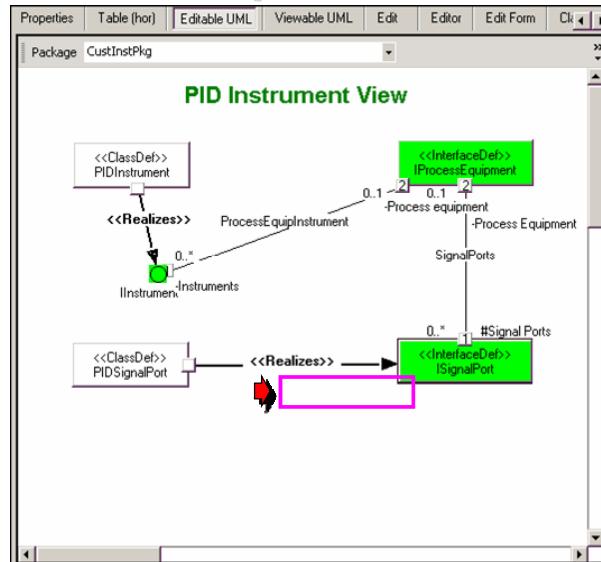
In the **Editable UML** view, right-click the object that you want to erase.





Erase Displayed Relationships

The relationship and label are erased from the UML view.

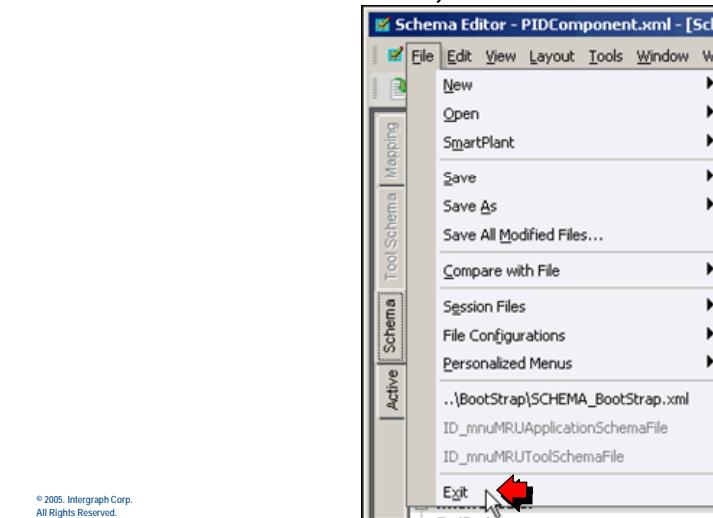


In order to show an example of using the Schema Editor in a different mode, use the exit command.



Exit the Schema Editor

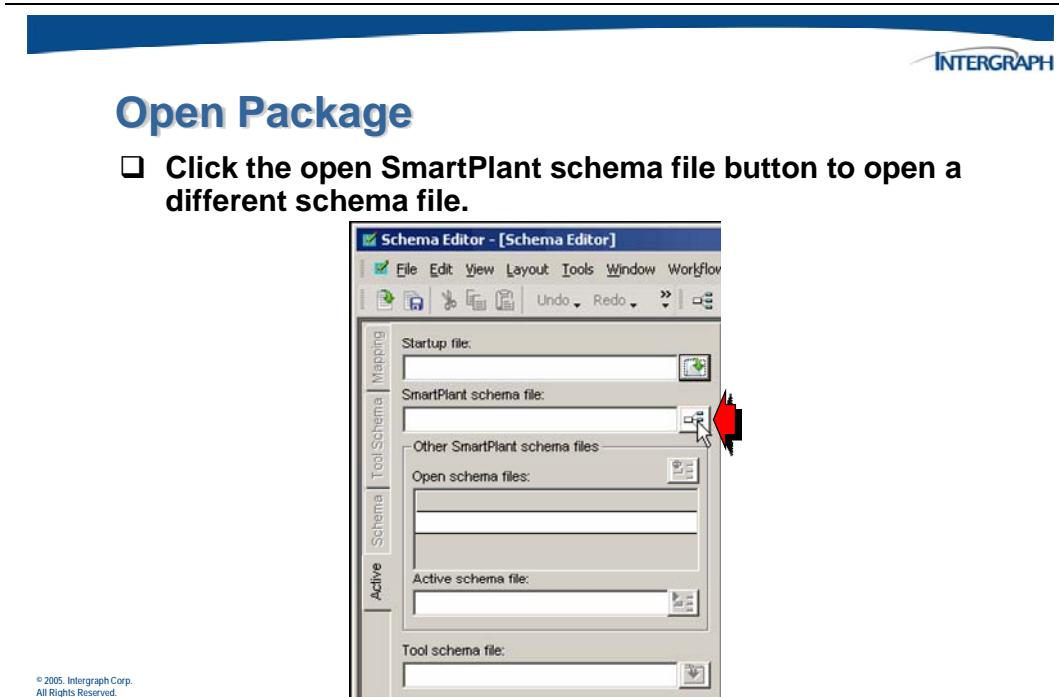
- From the menu bar, click the **File > Exit** command.



2.9 Open a Package

Packages are used to organize saved UML diagrams. Selecting a saved view from the *UML View* field displays that UML diagram.

You can view existing UML views by listing the saved **Packages** from the **Package** field and then selecting the view that you want to see in the UML View.

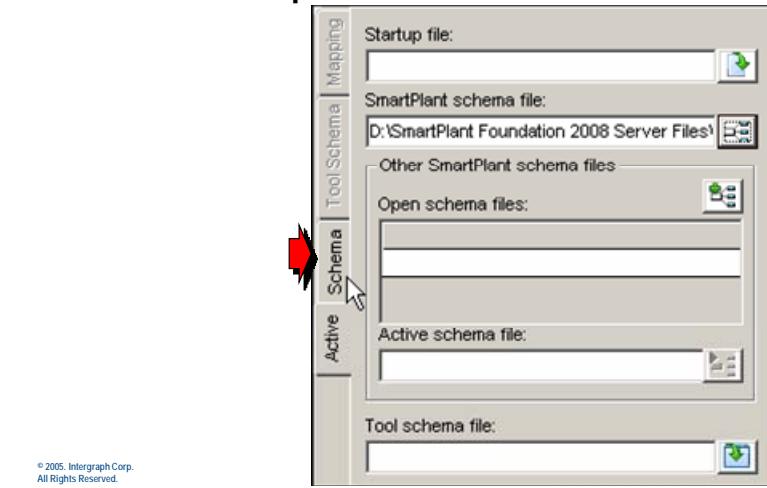


Browse to the **SmartPlant Foundation 2008 Server Files|Web_Sites|SPF42|EFSchema|04.02** folder to find the XML file to be opened.



Open Package

- Open the **Schema** tab to view the base class tree and search options.

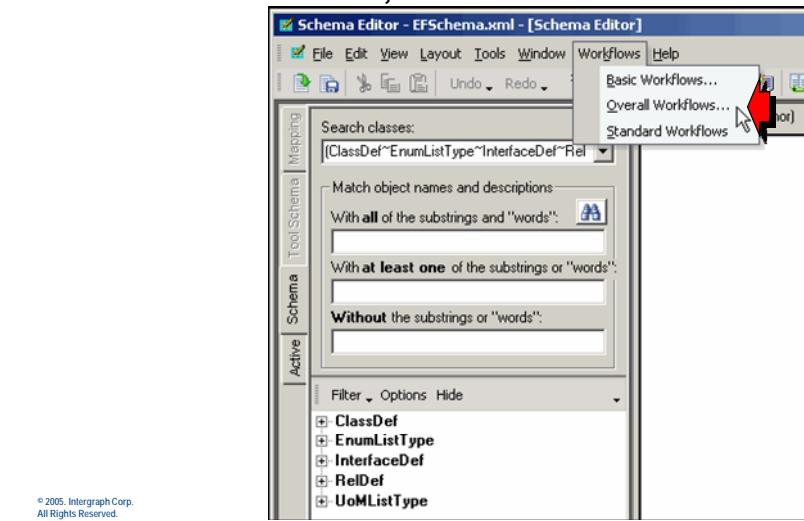


Next, switch from the *Standard Workflows* display to the *Overall Workflows* dialog.



Open Package

- From the menu, click **Workflows > Overall Workflows**.

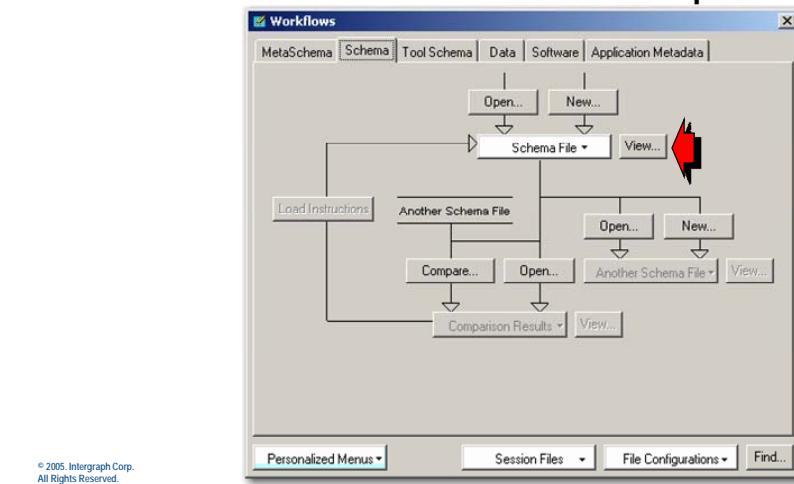


Make sure the **Schema** tab is selected when the *Workflows* dialog displays.



Open Package

- ❑ In the **Workflows** dialog box, click the **View** button beside the **Schema File** button to view the open schema file.

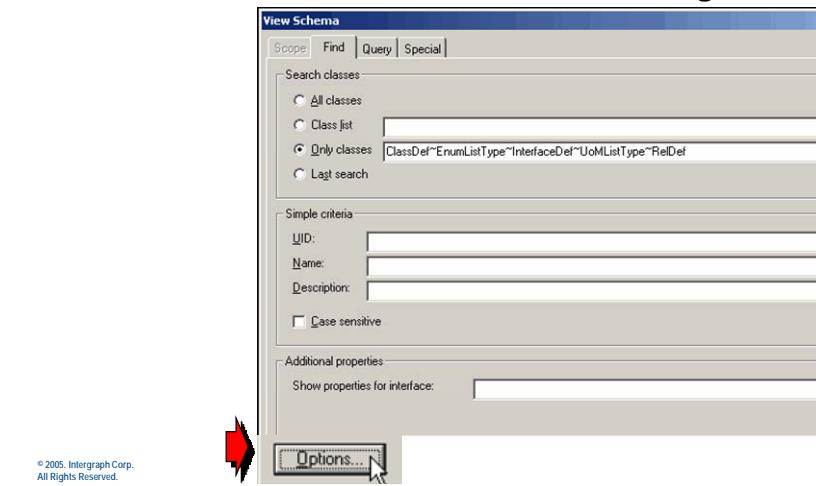


Choose a view option for displaying a saved UML view.

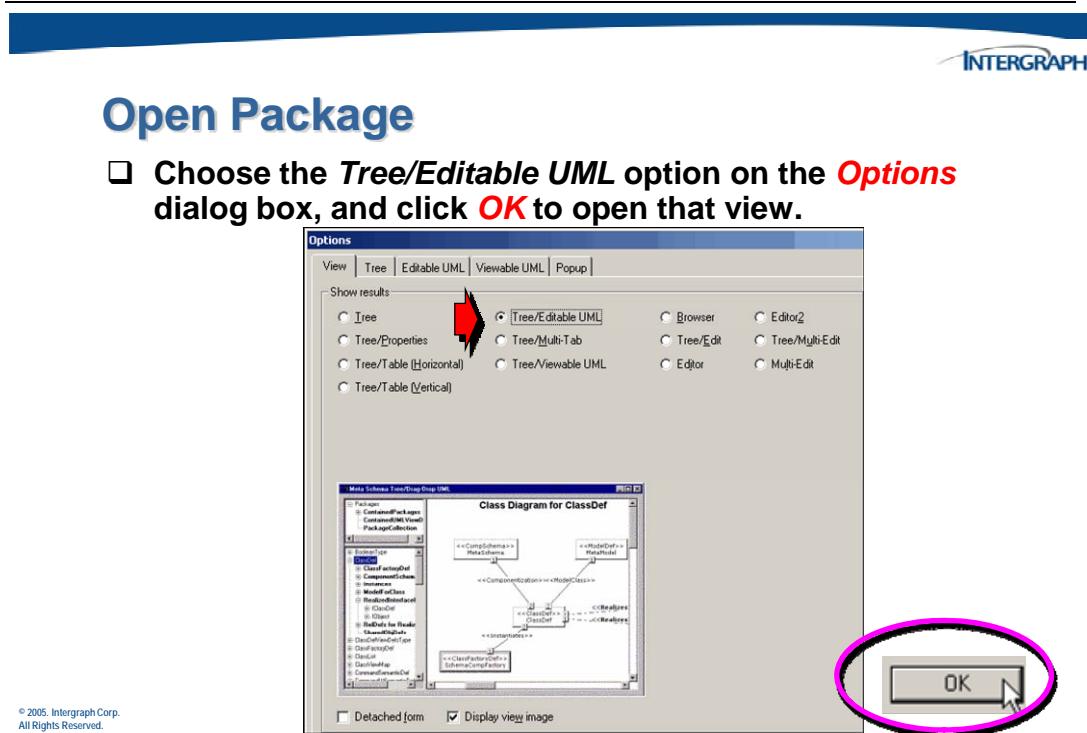


Open Package

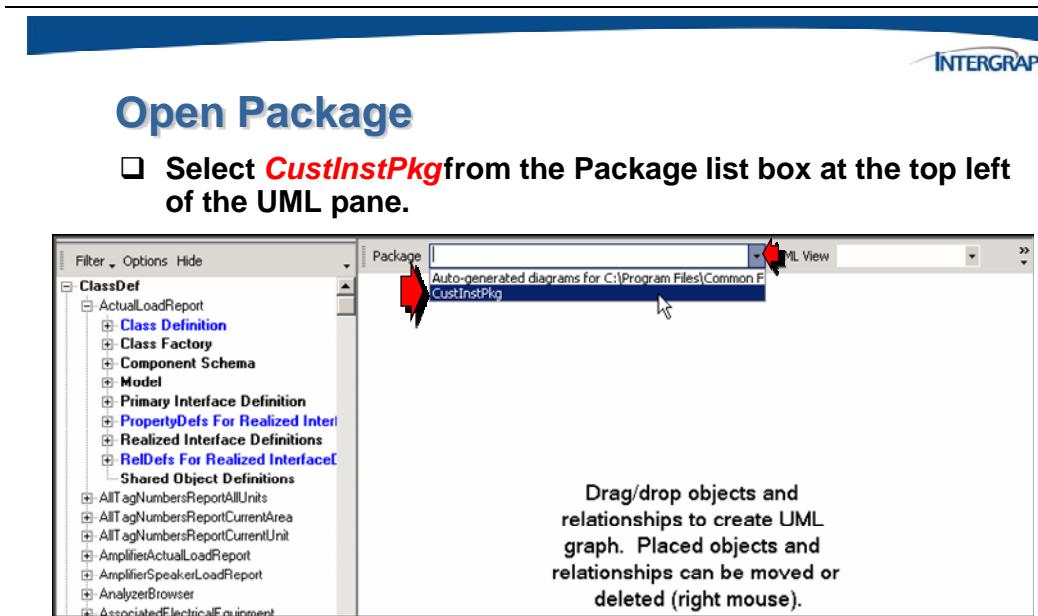
- ❑ To select a view option, click the **Options** button at the bottom left of the **View Schema** dialog box.



In order to see the saved packages and views, use the ***Tree/Editable UML*** view option.



You will be taken from the ***Options*** dialog back to the ***View Schema*** dialog.
You will be looking for any of the custom UML packages/views that you have defined.



Select a saved view that you wish to display.

INTERGRAPH

Open Package

□ Select **CustInstrumentView** from the UML View list box at the top right of the UML pane.

Drag/drop objects and relationships to create UML graph. Placed objects and relationships can be moved or deleted (right mouse).

© 2005, Intergraph Corp.
All Rights Reserved.

The displayed view changes to the custom saved UML view.

INTERGRAPH

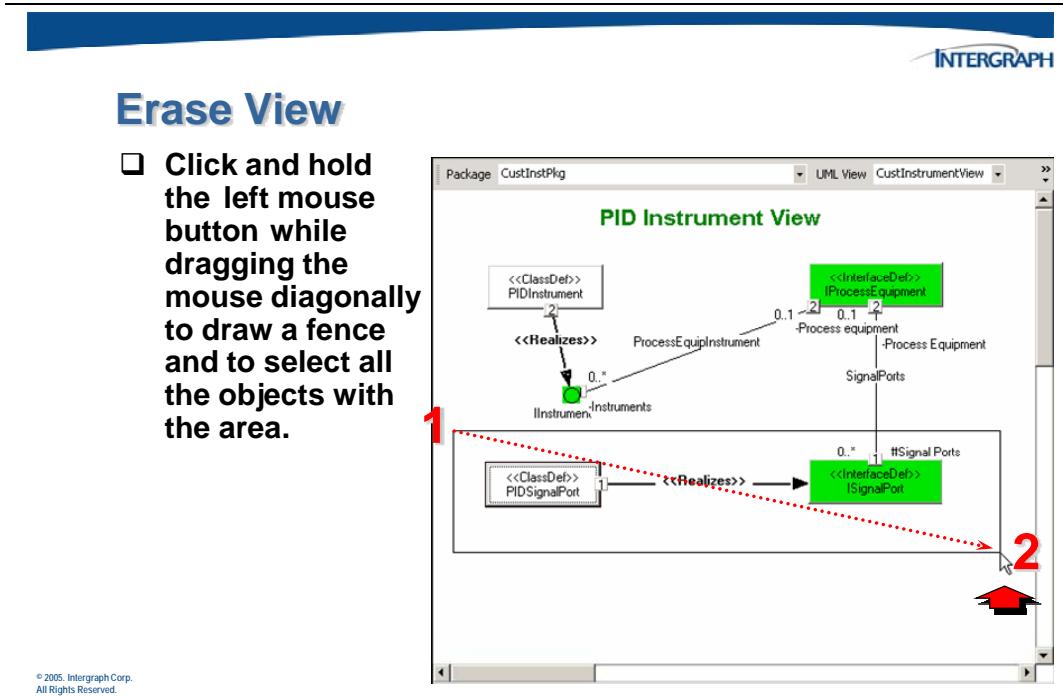
Open Package

The view changes to the saved **PID Instrument View**.

© 2005, Intergraph Corp.
All Rights Reserved.

2.10 Erasing the View

The **Erase** command on the shortcut menu will allow you to remove a portion or all of the objects displayed in a view. Before selecting the **Erase** command, select the objects that you want to erase by placing a rectangle around them.



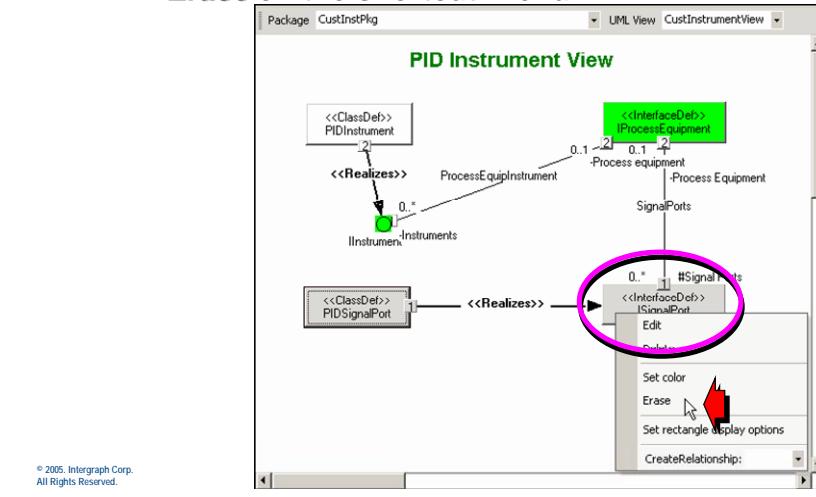
As you drag the mouse diagonally a dynamic rectangle will be displayed.

All of the objects completely inside of the dynamic rectangle will be selected.



Erase View

- Right-click a highlighted object in the view, and then click **Erase** on the shortcut menu.

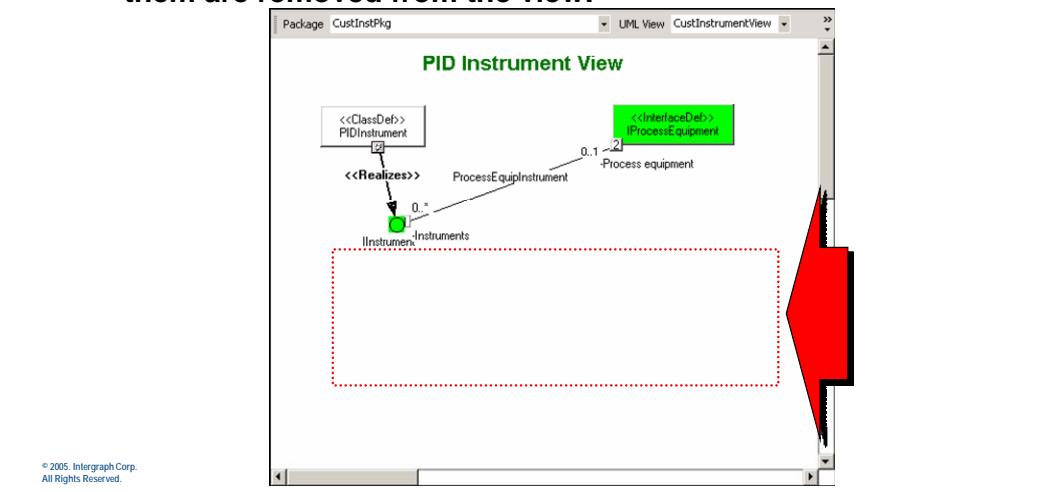


The selected objects will be erased but will remain in the schema.



Erase View

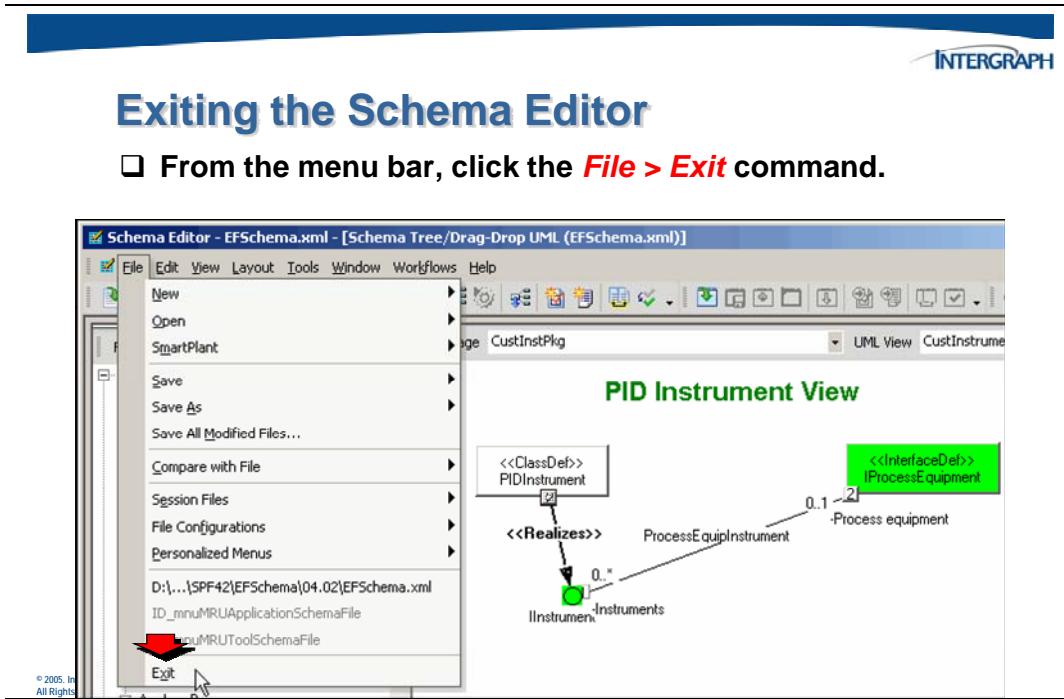
The selected objects and any relationships associated with them are removed from the view.



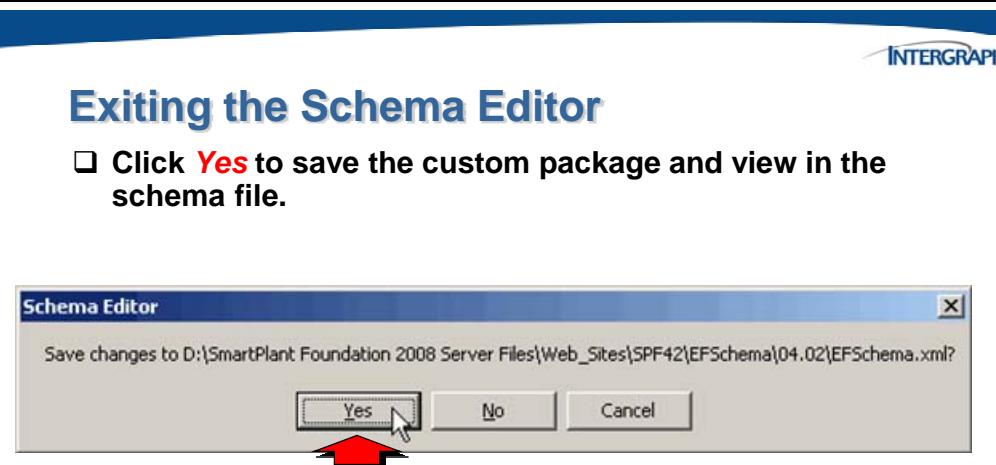
Note: You can also clear the view by right-clicking in the background of the view and clicking **New UML view definition** on the shortcut menu.

2.11 Exiting the Schema Editor

Once the view windows have been closed, use the **File** command on the main menu to **Exit** from the Schema Editor.



You will be prompted whether or not to save any changes that has been made to the extension schema.

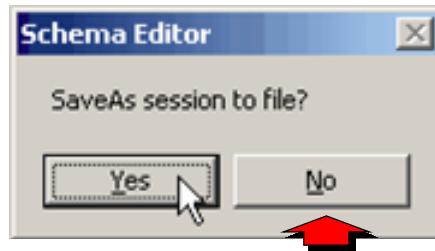


© 2005, Intergraph Corp.
All Rights Reserved.

A *Schema Editor* window will allow you to save the session to a file. We will not save the session file at this time.

Exiting the Schema Editor

- Click **No** when prompted to save the current configuration as a session file.



© 2005, Intergraph Corp.
All Rights Reserved.

2.12 Activity 1 – Using the Schema Editor

Complete **Chapter 2 - Activity 1** in the SPF Modeling and Mapping Activity Workbook.

2.13 Creating a Extension Schema

As part of the **Schema Modeling and Mapping** course, you will create the necessary class, interfaces, properties and relationships in the hands on sessions in chapters 3, 4 and 5. Below is an example of a **Pump** object that will be modeled and eventually implemented.



New Object – The Excel Import Pump



We want to map a class for storage in SPF called:

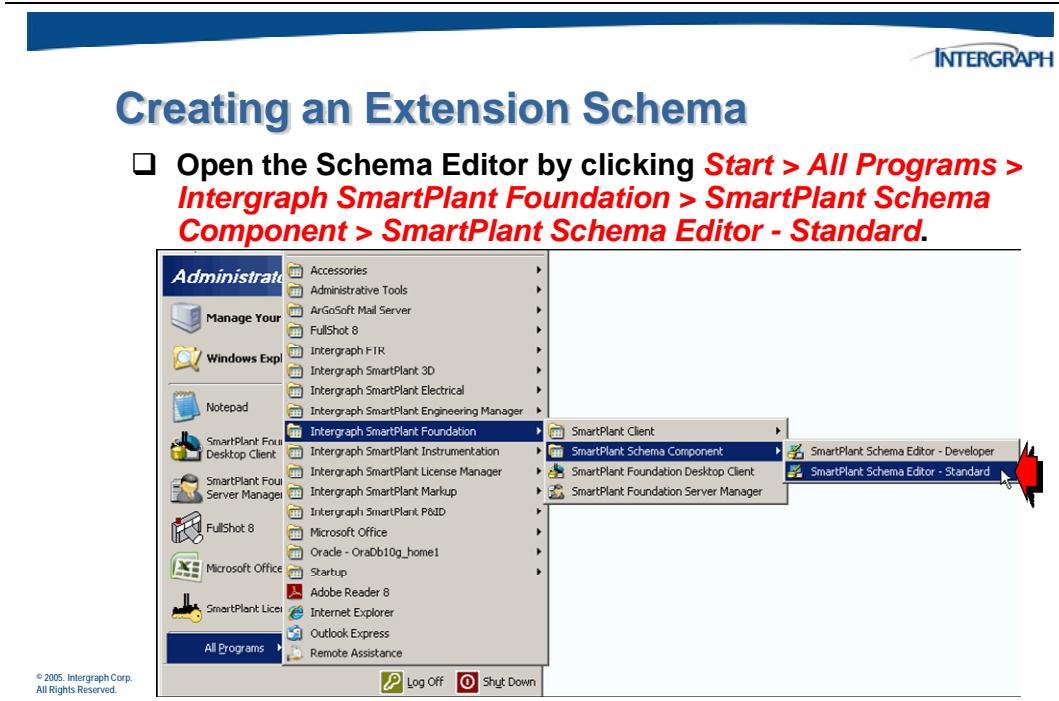
<u>Excel Import Pump</u>	
Name	
Description	
Pump Type	(Picklist)
Tag Prefix	
Tag Sequence Number	
Tag Suffix	
Maximum Pressure	
Maximum Temperature	
Hold Status	(Picklist)

© 2005, Intergraph Corp.
All Rights Reserved.

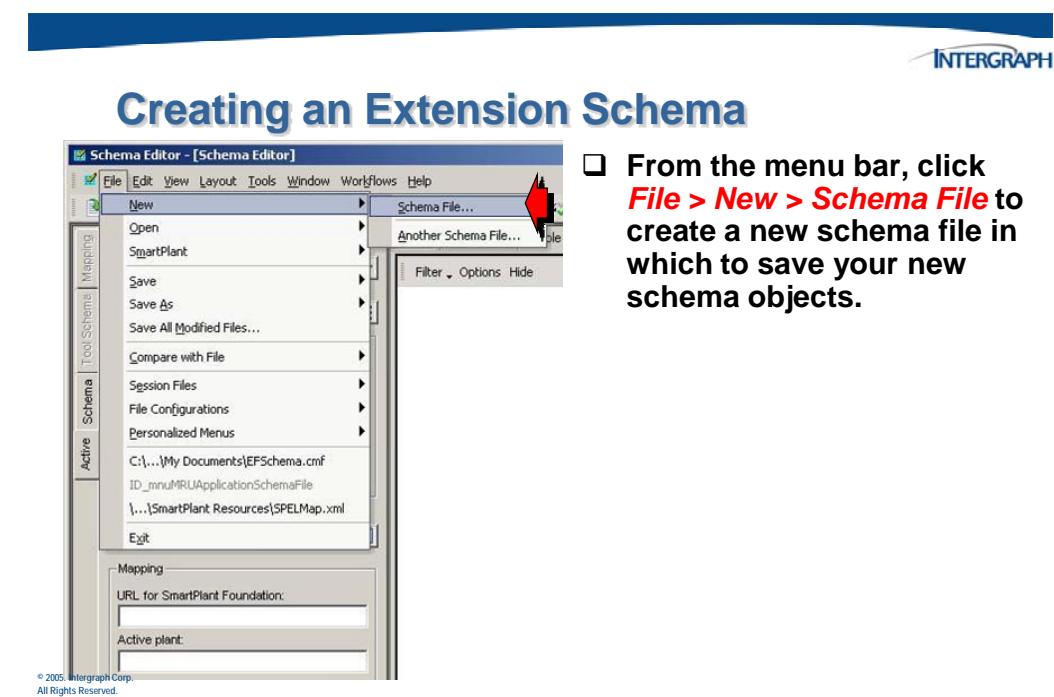
When extending the delivered schema structure, it is recommended that any site specific extensions be made in an “*extension*” schema.

This is done to make it easier to carry site specific extensions forward when you upgrade your SmartPlant Enterprise system. In future releases, just change the configuration of the extension schema file with your custom model extensions to use the new version of the EFSchema.xml file.

Begin by setting up a configuration in the Schema Editor to create extensions to the delivered model. Startup the Schema Editor in the *Standard* mode.



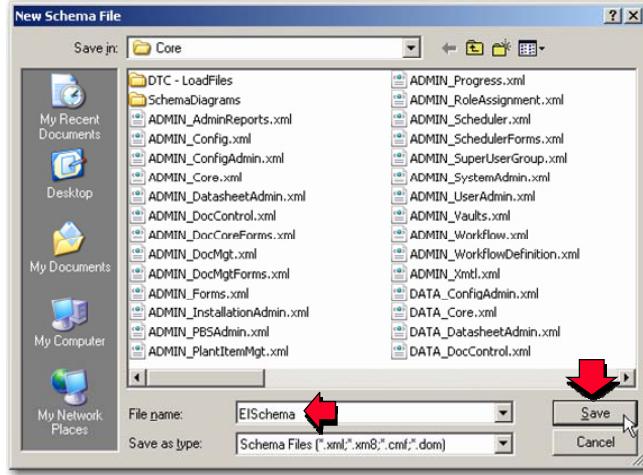
Create new extension file for you modeling changes.



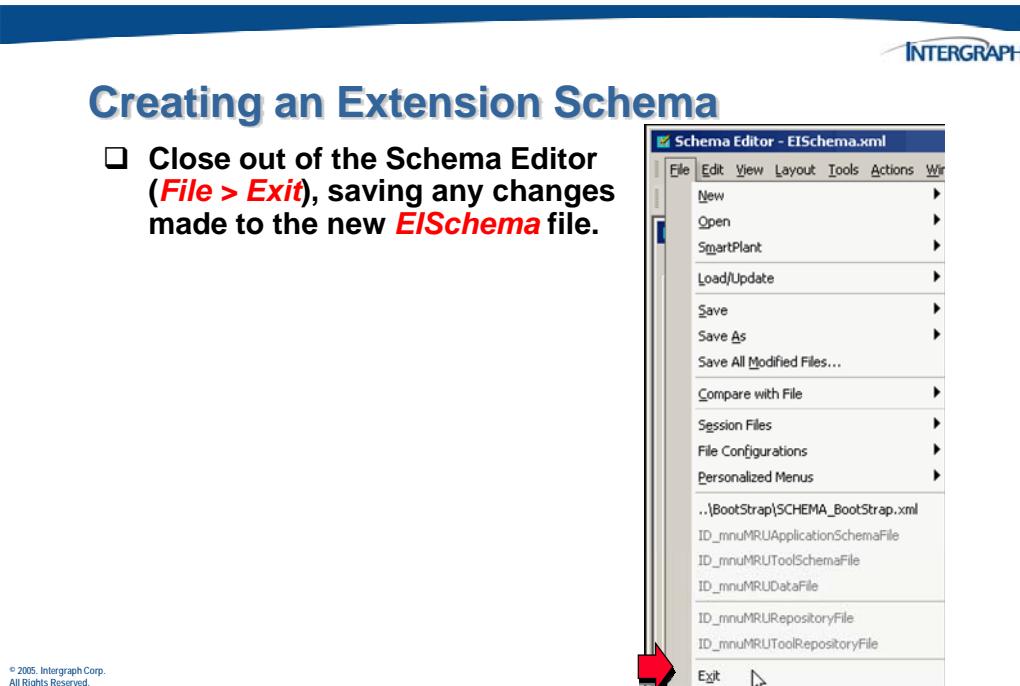
Call the new file EISchema, and place it in the same following folder: **D:\Program Files\SmartPlant\Foundation\2008\Models\Core**.

Creating an Extension Schema

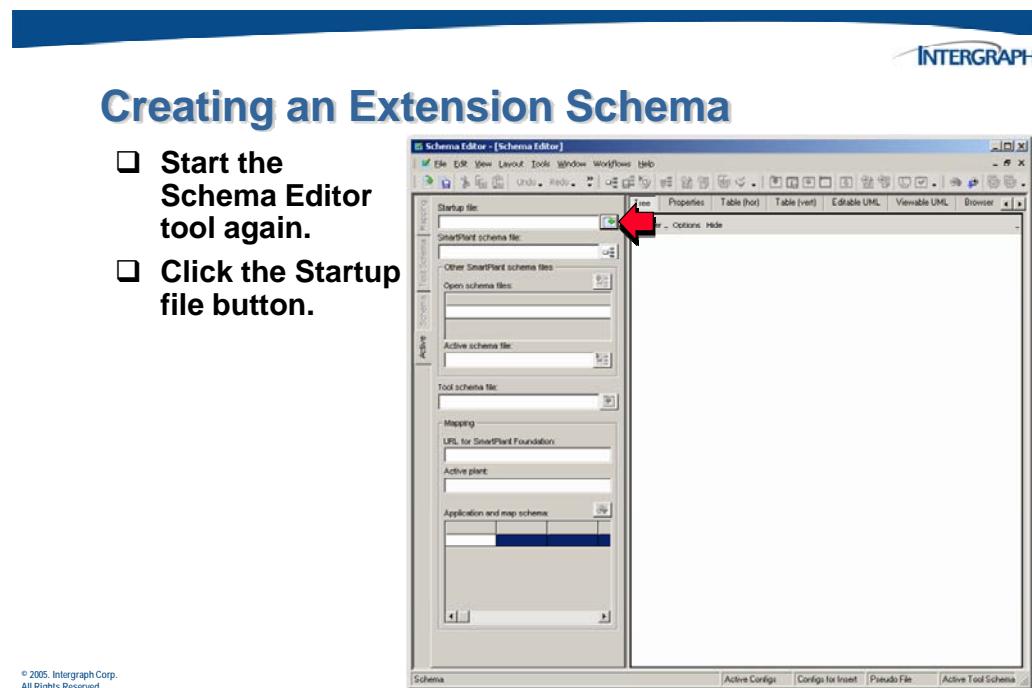
- Provide a name for your new schema file (**EISchema**), and save it in the **Core** directory.



Once you have created the new schema file, close the Schema Editor. We will be opening a number of files, and they must be opened in the correct order. If prompted, save the changes to the EISchema file.



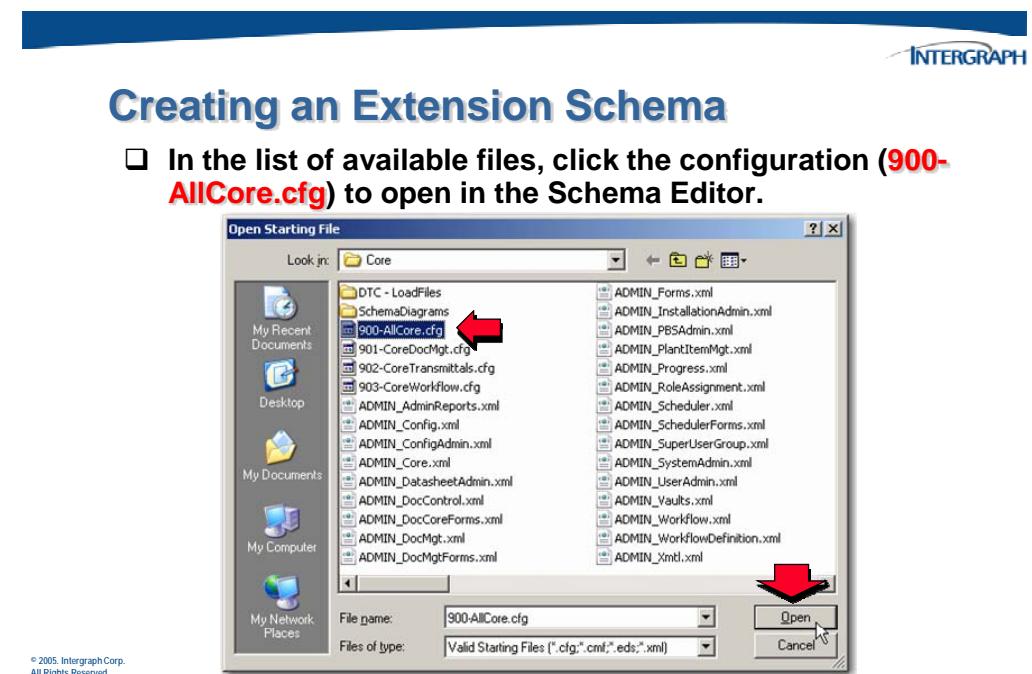
Once the Schema Editor has closed, re-open it as before, in the *standard* mode. Then click the button beside the *Startup file* field.



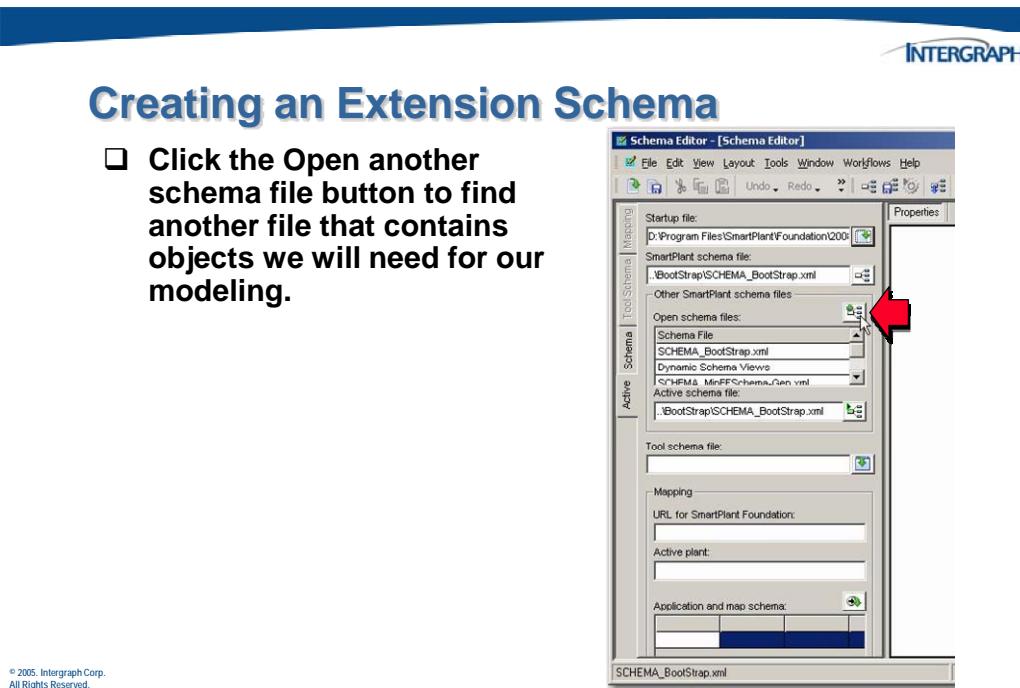
Creating an Extension Schema

- Start the Schema Editor tool again.
- Click the Startup file button.

Select the 900-AllCore.cfg file from the **\Program Files\SmartPlant\Foundation\2008\Models\Core** folder.



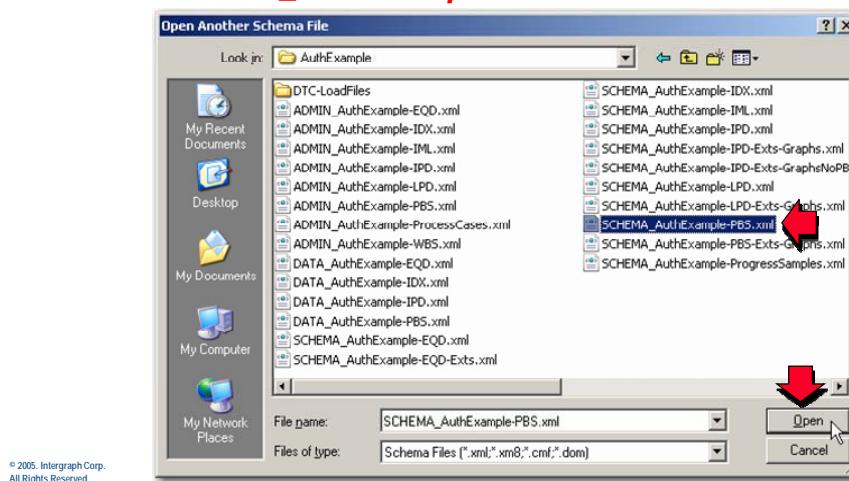
Next, click the button beside the **Open schema files** section to open another schema file.



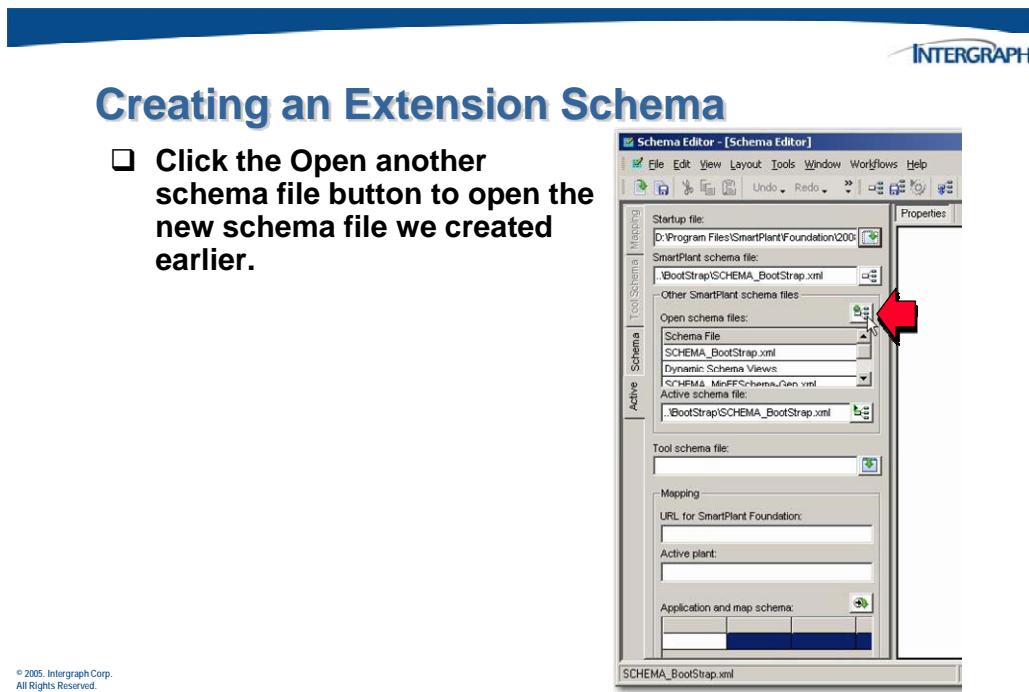
In the **D:\Program Files\SmartPlant\Foundation\2008\Models\AuthExample** folder, find and open the file called **SCHEMA_AuthExample-PBS.xml**.

Creating an Extension Schema

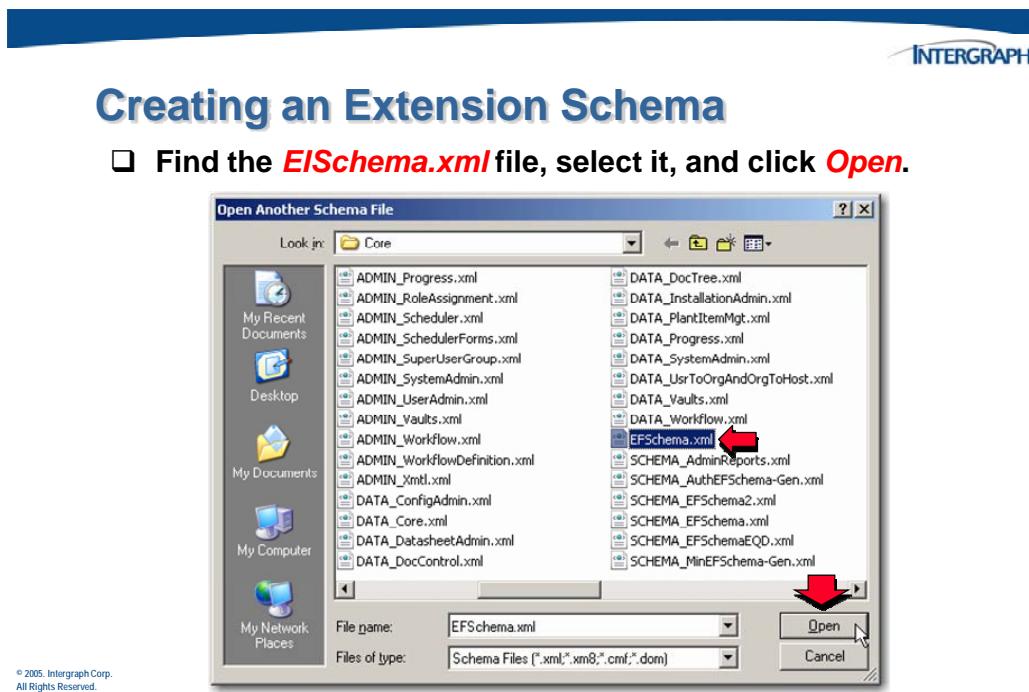
- In the **AuthExample** folder, find and open the **SCHEMA_AuthExample-PBS.xml** file.



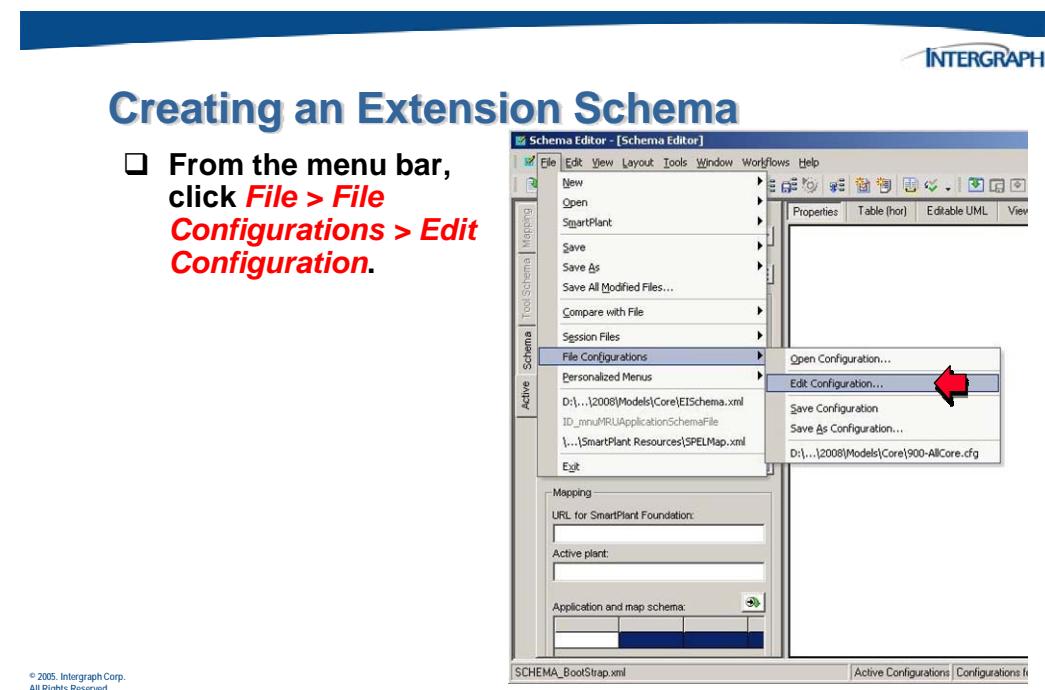
Click the same button again to open another file.



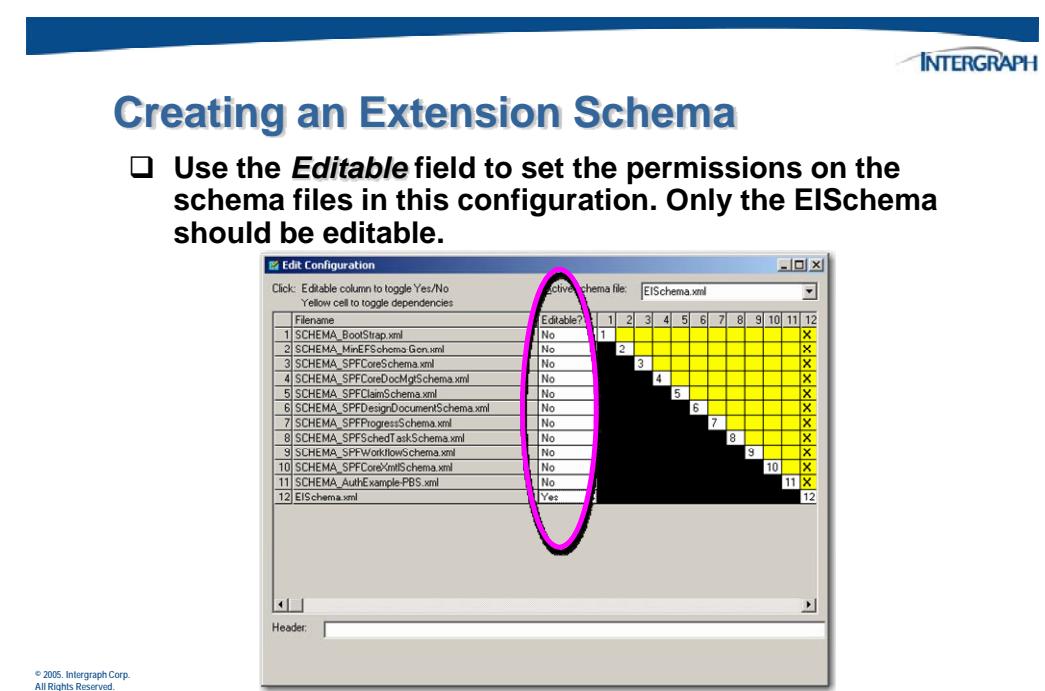
This time, open the new EISchema file you created a moment ago.



Once you have opened the necessary files, you need to create a configuration file that will open all these files and make the EISchema both editable and the active schema.



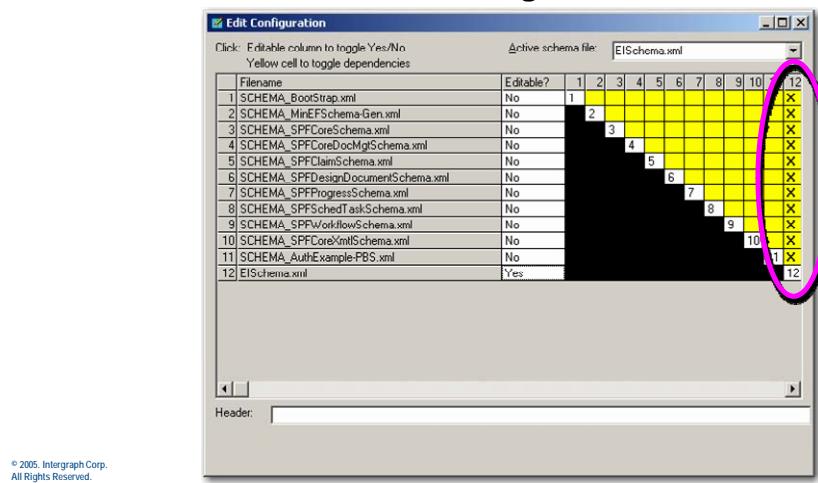
First, make sure that only the EISchema has a Yes in the Editable column. That file is the only one that you should make any changes to.



Next set your dependencies. The EISchema file is dependent on all the others in the list.

Creating an Extension Schema

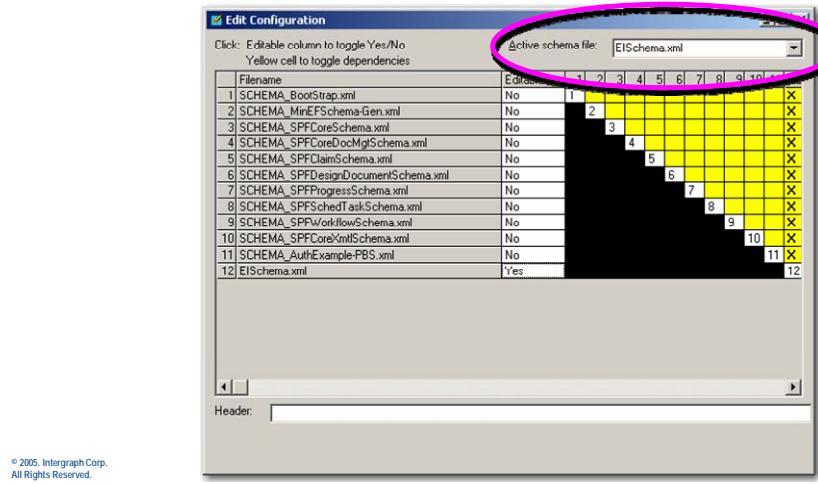
- ❑ Next make sure that the EISchema file is dependent on all the other files in the configuration.



Finally, make sure that the EISchema file is selected in the *Active schema file* field. This selection will tell the system that any new objects should be placed in the EISchema file.

Creating an Extension Schema

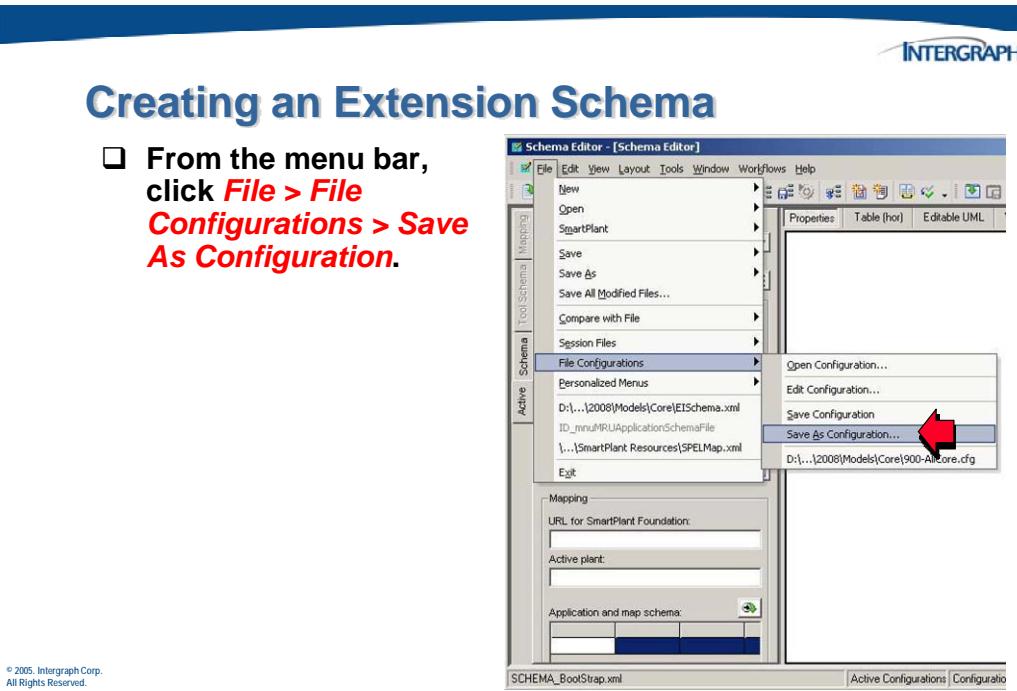
- Finally, make sure that the *EISchema* file is selected as the **Active schema file**, and then close the dialog box.



Save the new configuration file. We don't want to change the 900-AllCore configuration file, so we will create a new one for our purposes.

Creating an Extension Schema

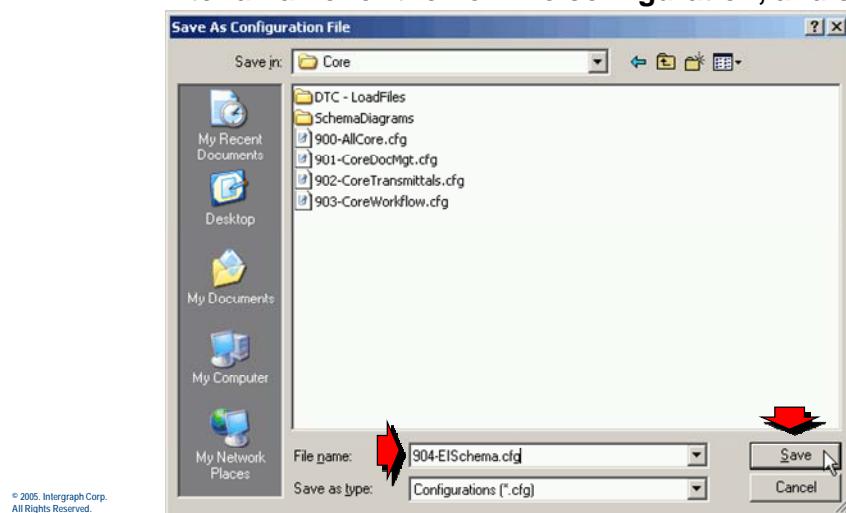
- From the menu bar, click **File > File Configurations > Save As Configuration**.



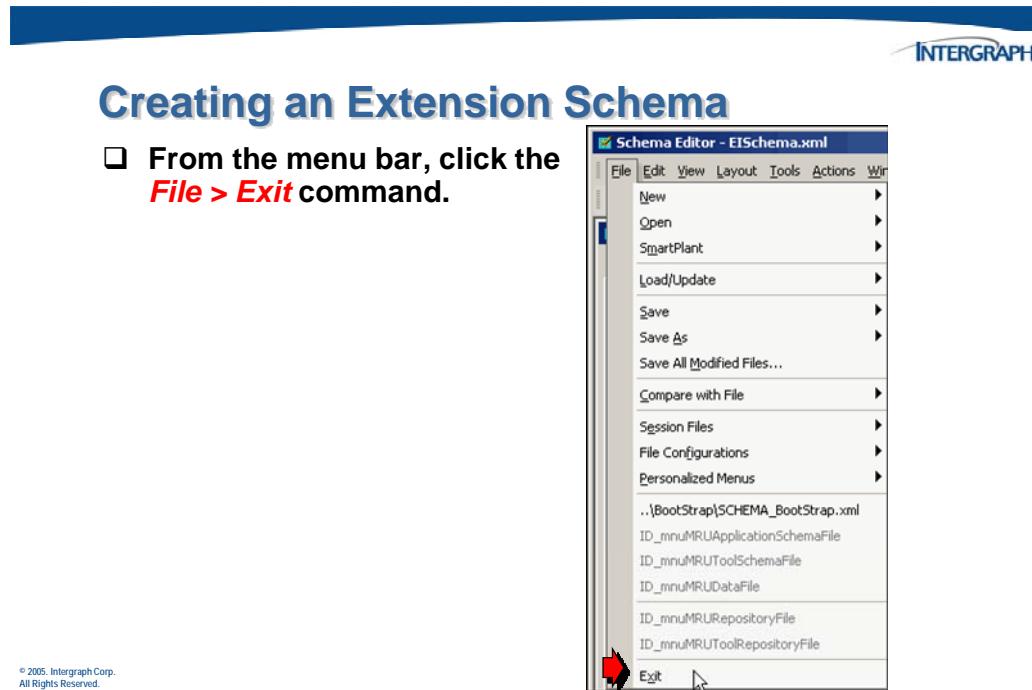
Name the new configuration file **904-EISchema.cfg** and save it into the same directory as the **900-AllCore.cfg**.

Creating an Extension Schema

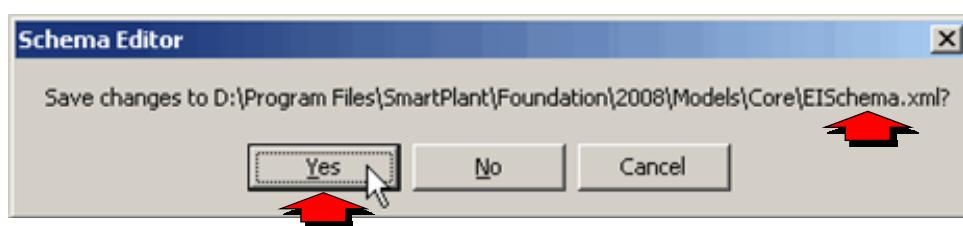
- Enter a name for the new file configuration, and click **Save**.



You can now close the schema editor.



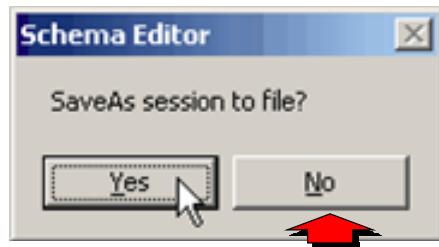
If prompted, choose Yes to save the changes to the EISchema.xml file.



Do not save the session file at this time. The cfg file we just created will open all the files we will need to perform our activities.

Creating an Extension Schema

- ❑ Click **No** when prompted to save the session file information.



2.14 Activity 2 – Creating a Extension Schema

Complete **Chapter 2 - Activity 2** in the SPF Modeling and Mapping Activity Workbook.

C H A P T E R

3

Schema Overview and Modeling New Classes

3. Meta Schema Concepts

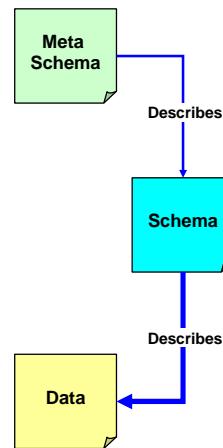
The **Meta Schema** describes the SmartPlant schema and provides the building blocks upon which the SmartPlant schema is built. Because it would be impractical and unwise to write code behind every defined object in the SmartPlant schema, there is code behind every object defined in the meta schema that drives their behavior.



Meta Schema Concepts

The **meta schema** is a set of schema objects that describe the objects in the SmartPlant schema. The meta schema defines the language in which the SmartPlant schema is written.

The SmartPlant schema is a set of schema objects that describe the data that is published by the authoring tools.



© 2005, Intergraph Corp.
All Rights Reserved.

The meta schema classes do not directly describe the **data** that is part of the schema (the SmartPlant schema describes the data). Instead, these classes describe the **classes** in the schema. The meta schema defines the rules of the schema and is the code behind the schema.

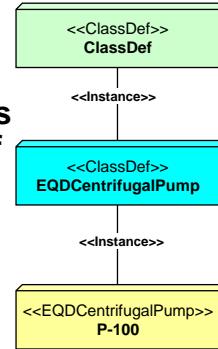
For example, all classes in the schema are instances of the *ClassDef* class in the meta schema. Similarly, all relationship definitions in the schema are of class *RelDef*, which is part of the meta schema.

The meta schema also describes itself. All classes in the meta schema are also of the class *ClassDef*, which is defined as part of the meta schema.

Meta Schema Concepts

In this example, **ClassDef** is the meta schema object that describes the **EQDCentrifugalPump** object in the schema. The EQDCentrifugalPump class in the schema is an instance of ClassDef in the meta schema.

The EQDCentrifugalPump class in the schema describes **P-100**, which is the data. P-100 is an instance of the EQDCentrifugalPump class.



More information on the meta schema classes will be discussed in the following sections of this chapter.

3.1 Model Definitions

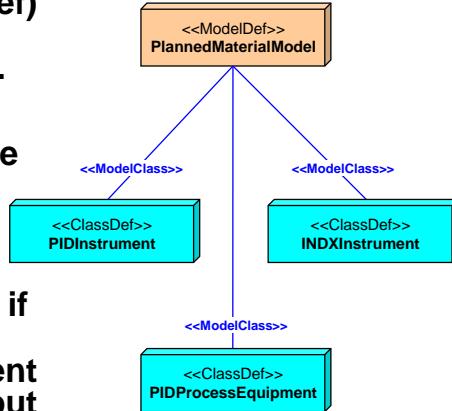
A model is a non-redundant description of an object. For example, a 3D plant model is a non-redundant 3D description of a plant (non-redundant = non-excessive). Model definitions in the SmartPlant schema are roughly based on the POSC-CAESAR model.

Model Definitions

A **model definition** (ModelDef) is the schema information used to describe the model.

The primary function of a ModelDef is to identify those objects that can be shared through a ModelClass relationship.

Objects can only be shared if they exist within the same ModelDef. Objects in different ModelDefs can be related, but cannot be shared.



© 2005, Intergraph Corp.
All Rights Reserved.

The only use for the ModelDef currently is to enforce a rule that ClassDefs must be in the same ModelDef to participate in a *Sharing* relationship.



Model Definitions

ModelDefs define domains of data. There are seven ModelDefs in the SmartPlant schema:

- MetaModel**
- OrganizationModel**
- WorkBreakdownModel**
- FacilityModel**
- PlannedMaterialModel**
- TypicalMaterialModel**
- ActualMaterialModel**

© 2005, Intergraph Corp.
All Rights Reserved.

MetaModel - Used only by the ClassDefs in the meta-schema. You should never use this model for Schema.

OrganizationModel - Like the name implies, this is intended for ClassDefs that are organizational concepts. Examples would be User, Company, Department, Manufacturer, Vendor, etc.

WorkBreakdownModel - For ClassDefs that provide organizations of work. This includes the ClassDefs that you see in the WBS documents, such as Project and Contract, as well as all the document ClassDefs defined in the schema.

FacilityModel - For ClassDefs that represent abstract concepts in a process plant. Examples include Plant, Area, Unit, System, Discipline (although you might put Discipline in the OrganizationModel instead.)

PlannedMaterialModel - For ClassDefs that represent real objects that you plan to purchase to construct your design. This is the “design” arena in which engineering tools like Zygad, SPPID, and SmartPlant Instrumentation work. In a traditional SPF Model, this would be the “tag”.

TypicalMaterialModel - For ClassDefs that represent real objects offered by manufacturers or vendors. When an engineer goes looking for something to fulfill the requirements of his design, he might look through a manufacturer catalog. This is the domain of this model. In a traditional SPF Model, this would be the “model”, or “catalog”, or “standard equipment”.

ActualMaterial - For ClassDefs that represent what is purchased and assembled to construct the plant. In a traditional SPF Model, this would be the “asset”.

Evolution for objects is normally from planned facility (FacilityModel) to planned material (PlannedMaterialModel) to actual material (ActualMaterialModel). Catalog parts are part of the typical material model. Meta schema objects are part of the meta model. The organization model contains objects that are outside the normal set of designed objects; these objects describe the organization rather than the design. The work breakdown model contains object associated with the work being performed rather than the objects for that work.

Note: All ModelDefs are part of the meta schema and are therefore predefined.

3.2 Component Schemas

A **Component Schema** is a subdivision of the complete SmartPlant schema and is a set of class definitions that are used within a specified domain or application area. Class definitions tie a primary interface definition to a component schema. Class definitions and interface definitions will be discussed in more detail later.

The component schema breaks up the SmartPlant schema into manageable chunks that correspond almost one to one to a document type being published. At a minimum, there is a component schema defined for each tool that publishes into SmartPlant. You can extract component schemas from the SmartPlant schema using the Schema Editor.

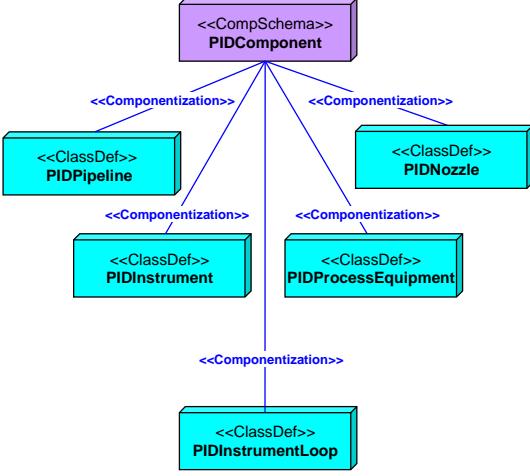
One of the primary purposes of the component schema is performance. When a tool indicates it is going to publish a certain document type, the SmartPlant Client loads only the necessary component schema instead of the entire SmartPlant schema.



Component Schemas

A component schema (CompSchema) is a set of class definitions that are used within a specified domain or application area.

For example, the PIDComponent schema contains all the class definitions that are relevant to P&IDs.



© 2005. Intergraph Corp.
All Rights Reserved.



Component Schemas

Component schemas contain the following:

- The set of class definitions that have a componentization relationship with the component schema**
- The interface definitions realized by those class definitions**
- The property definitions exposed by those interface definitions**
- The property types scoped by those property definitions**

© 2005, Intergraph Corp.
All Rights Reserved.

The component schemas are full of examples of shared objects that appear in more than one component schema. Virtually all of the schema data defined in a component schema, including class definitions, interface definitions, property definitions, and property types, may be shared by one or more other component schemas.

The unique identity of an object is defined by its unique identifier (UID) and does not depend in any way on its classification. Therefore, an object can be classified differently by different components and still be one shared object in the SmartPlant Foundation database. Shared object definitions will be discussed later in this chapter.

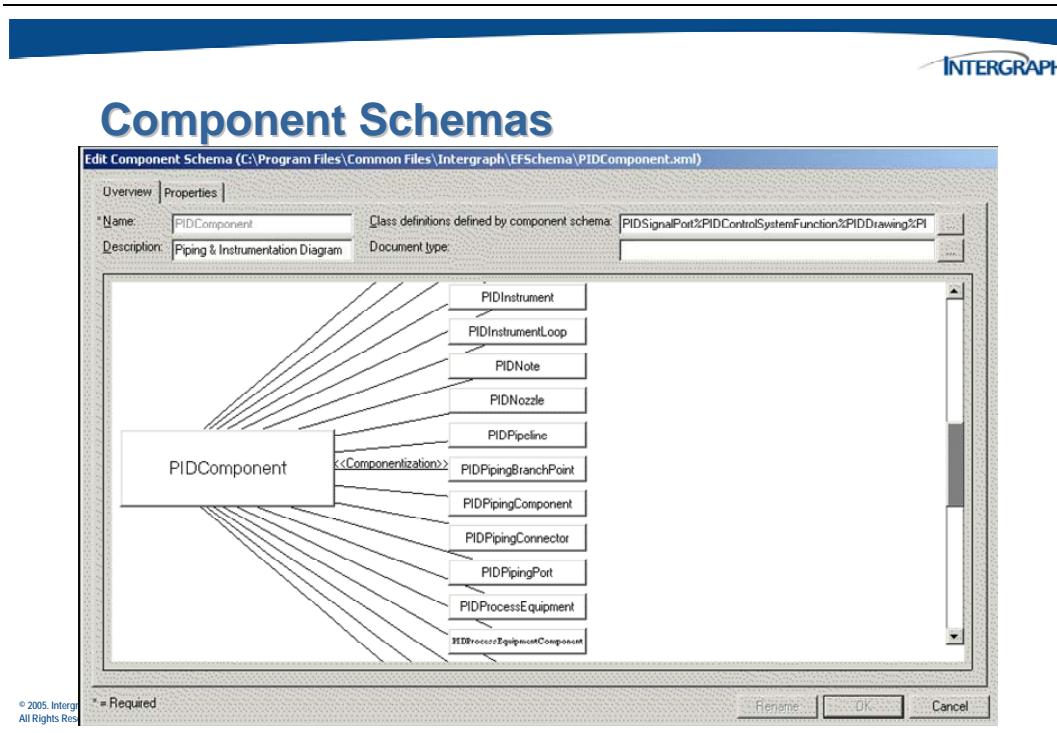
For example, if an object in one component schema is classified as class AA1 and the same object with the same UID is classified as class AA2 by another component schema, then the shared object in the SmartPlant Foundation database contains the information associated with the union of classes AA1 and AA2.

Editing of the component schema should always be done against the SmartPlant schema files from which the component schema files were generated rather than the component schema files themselves (because they can be regenerated).

3.2.1 Properties of a Component Schema

Component schemas have the following properties:

- ❑ **Name** – Specifies the name of the component schema.
- ❑ **Description** – Specifies the description of the component schema.
- ❑ **Class definitions defined by component schema** – List all class definitions that have a componentization relationship with the component schema. Because validation of published XML is done using the component schema instead of the SmartPlant schema, the component schema should contain all class definitions required to publish a particular document type.
- ❑ **Document type** – Specifies which document types the component schema scopes. This setting is currently optional because the SmartPlant Client queries the adapter for this relationship during a publish.



3.3 Interactive Activity – Creating a Component Schema

For the interactive activities in this chapter, we will be creating a new component schema, as well as all the necessary definitions to add some custom attributes to the schema. The following slides will explain the process that you will use throughout this chapter to accomplish this task.

New Object – The Excel Import Pump



Excel Spreadsheet

We want to map a class for storage in SPF called:

<u>Excel Import Pump</u>	
Name	
Description	
Pump Type	(Picklist)
Tag Prefix	
Tag Sequence Number	
Tag Suffix	
Maximum Pressure	
Maximum Temperature	
Hold Status	(Picklist)

© 2005, Intergraph Corp.
All Rights Reserved.

This is an example of **Excel Import Pump** that needs to be implemented. What you see depicted above is a form that will be used to gather the necessary information about the pump.

First, we will need a new **Component Schema** to contain the type of object to be added to the SmartPlant Desktop Client.



New Object – The Excel Import Pump

Excel Spreadsheet

Step 1 – Create a Component Schema to store class defs of objects that will be added through an import from Excel.

Component Schema: SPFExcelImportComponent

Want to map a class EIPump for Storage in SPF:

<u>Excel Import Pump</u>	
Name	
Description	
Pump Type	(Picklist)
Tag Prefix	
Tag Sequence Number	
Tag Suffix	
Maximum Pressure	
Maximum Temperature	
Hold Status	(Picklist)

© 2005. Intergraph Corp.
All Rights Reserved.

Since a new type of object will be created, a new class of object must be defined, which is called a Class Definition.



New Object – The Excel Import Pump

Excel Spreadsheet

Step 2 – Create a class definition for the Excel Import Pump.

Want to map a class EIPump for Storage in SPF:

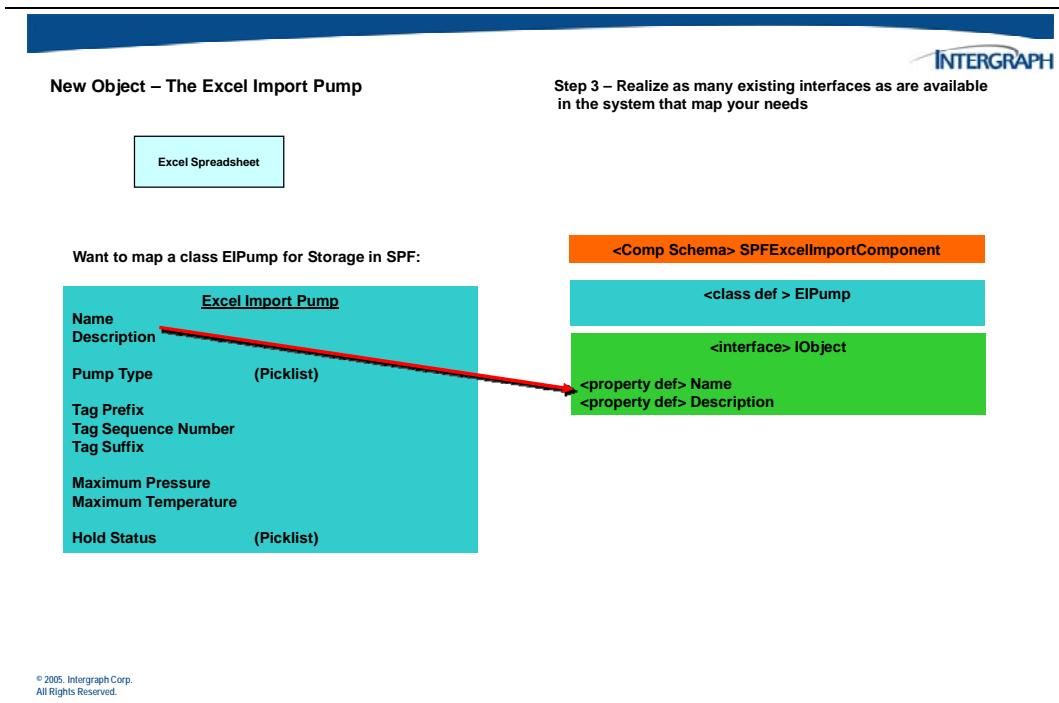
<u>Excel Import Pump</u>	
Name	
Description	
Pump Type	(Picklist)
Tag Prefix	
Tag Sequence Number	
Tag Suffix	
Maximum Pressure	
Maximum Temperature	
Hold Status	(Picklist)

<Comp Schema> SPFExcelImportComponent

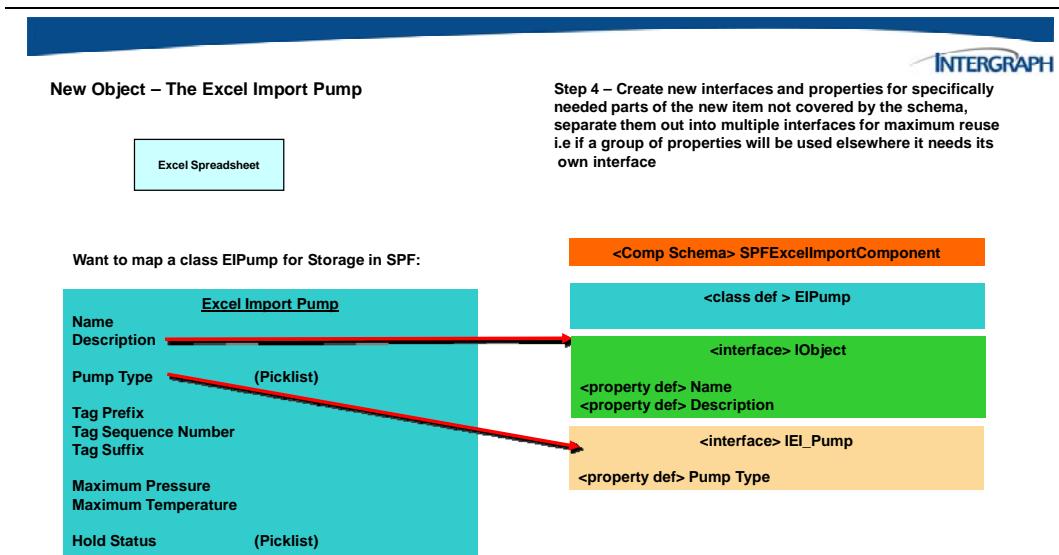
<class def > EIPump

© 2005. Intergraph Corp.
All Rights Reserved.

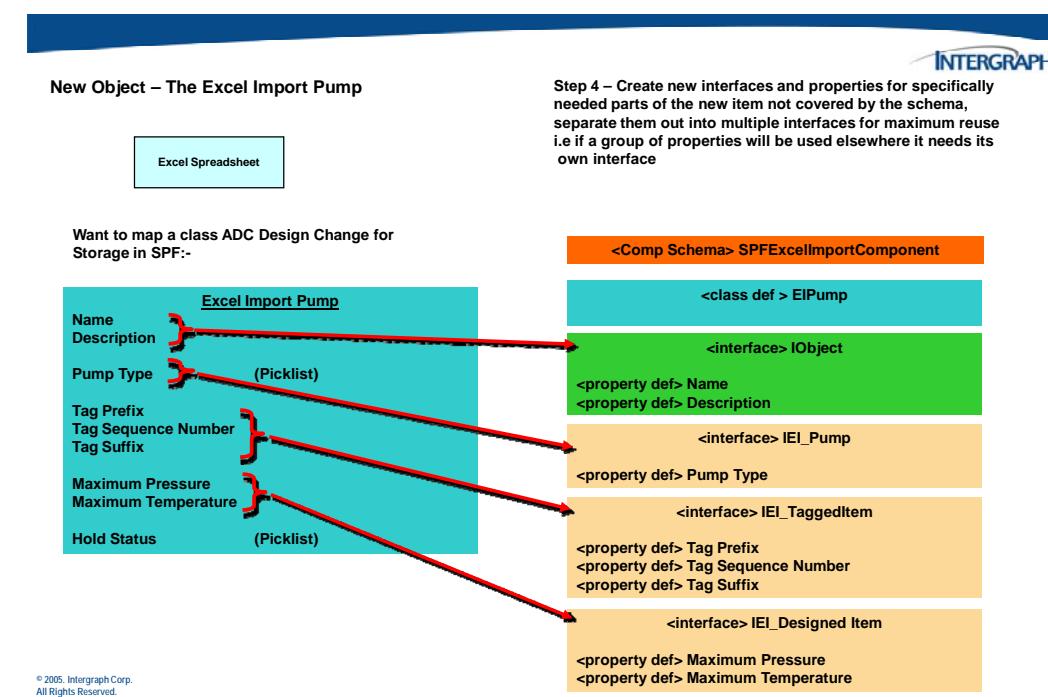
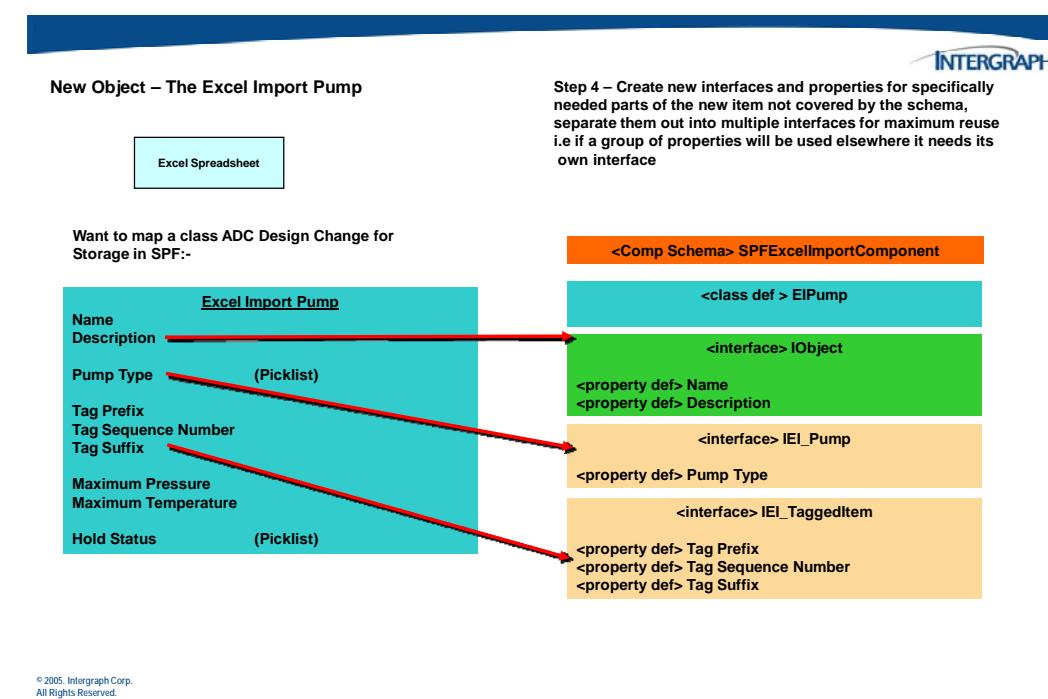
In a couple of instances, the necessary attributes or **properties** needed to provide the necessary information have already been defined in the system and associated with an **interface**. Rather than re-define existing objects, you can reference the existing interface and use its properties.



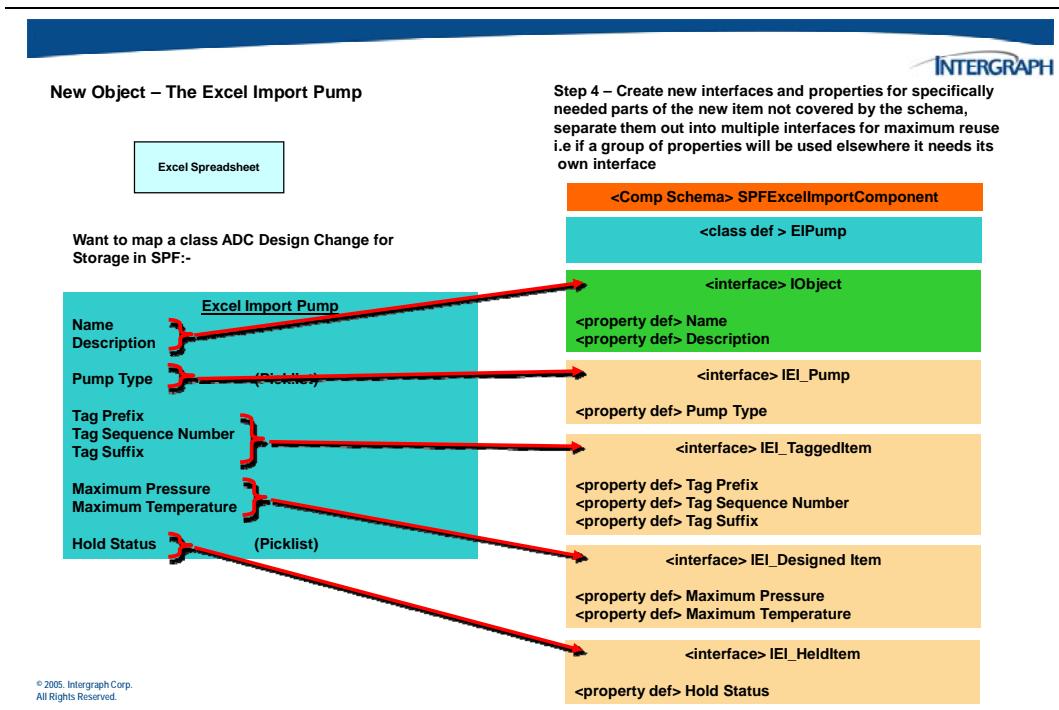
You will also need to define **new** interfaces and properties to complete the mechanism to capture all of the needed information for this new object type.



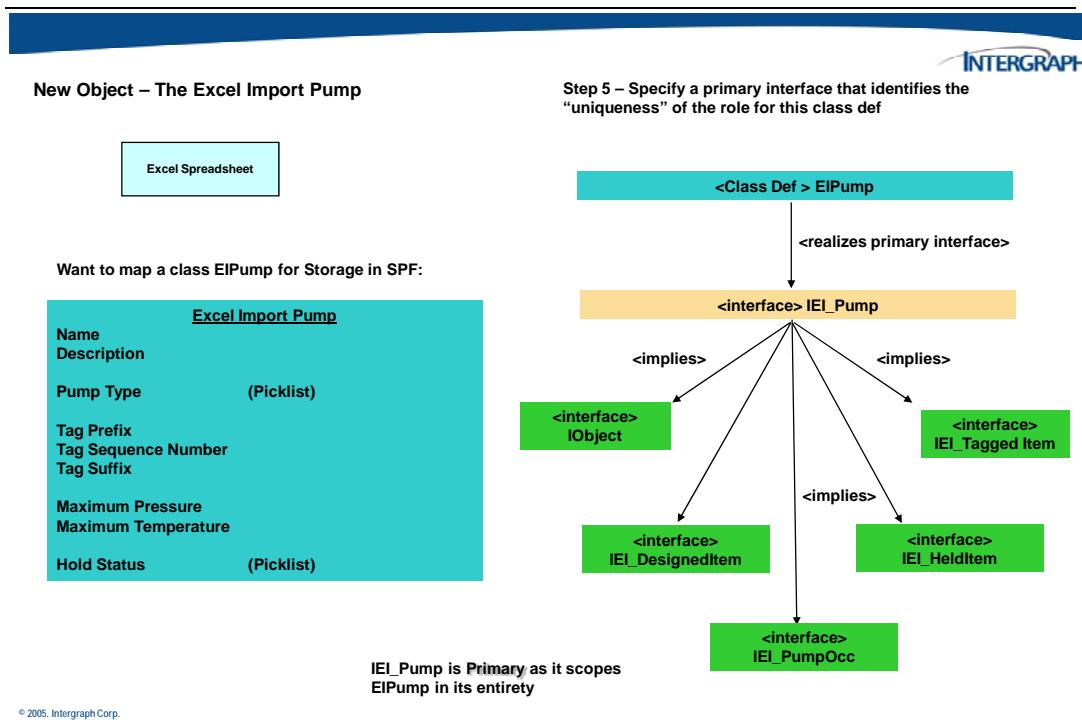
The various properties we need can be spread out across multiple interfaces to make it easier to re-use the interfaces for more than one type of object.



A portion of the needed data can use a picklist or **enumerated list**.



As part of defining a new class (ClassDef), a **primary interface** for the class should be specified.



One of the last steps will be to specify the values used for the picklists.

New Object – The Excel Import Pump

Excel Spreadsheet

Want to map a class EI_Pump for Storage in SPF:

<u>Excel Import Pump</u>	
Name	
Description	
Pump Type	(Picklist)
Tag Prefix	
Tag Sequence Number	
Tag Suffix	
Maximum Pressure	
Maximum Temperature	
Hold Status	(Picklist)

Step 6 – Scope the design status property using a “picklist”, an enum list type and enum enums (entries)

```

classDiagram
    class IEI_Pump {
        <interface>
    }
    class EI_PumpType {
        <property def>
    }
    class PumpTypes {
        <enum list type>
    }
    class Horizontal {
        <enum enum>
    }
    class Vertical {
        <enum enum>
    }

    IEI_Pump <|--> EI_PumpType
    EI_PumpType <|--> PumpTypes
    PumpTypes <|--> Horizontal
    PumpTypes <|--> Vertical
  
```

© 2005, Intergraph Corp.
All Rights Reserved.

New Object – The Excel Import Pump

Excel Spreadsheet

Want to map a class EI_Pump for Storage in SPF:

<u>Excel Import Pump</u>	
Name	
Description	
Pump Type	(Picklist)
Tag Prefix	
Tag Sequence Number	
Tag Suffix	
Maximum Pressure	
Maximum Temperature	
Hold Status	(Picklist)

Step 6 – Scope the design status property using a “picklist”, an enum list type and enum enums (entries)

```

classDiagram
    class IEI_HeldItem {
        <interface>
    }
    class EI_HoldStatus {
        <property def>
    }
    class HoldStatuses {
        <enum list type>
    }
    class Held {
        <enum enum>
    }
    class Released {
        <enum enum>
    }

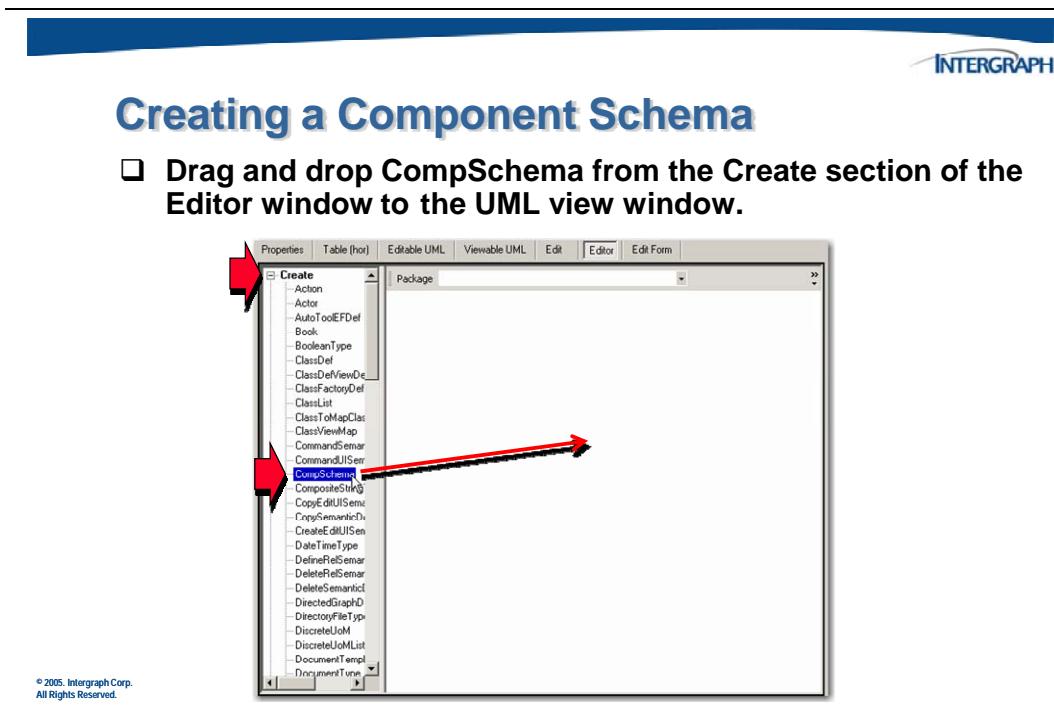
    IEI_HeldItem <|--> EI_HoldStatus
    EI_HoldStatus <|--> HoldStatuses
    HoldStatuses <|--> Held
    HoldStatuses <|--> Released
  
```

© 2005, Intergraph Corp.
All Rights Reserved.

Hands on:

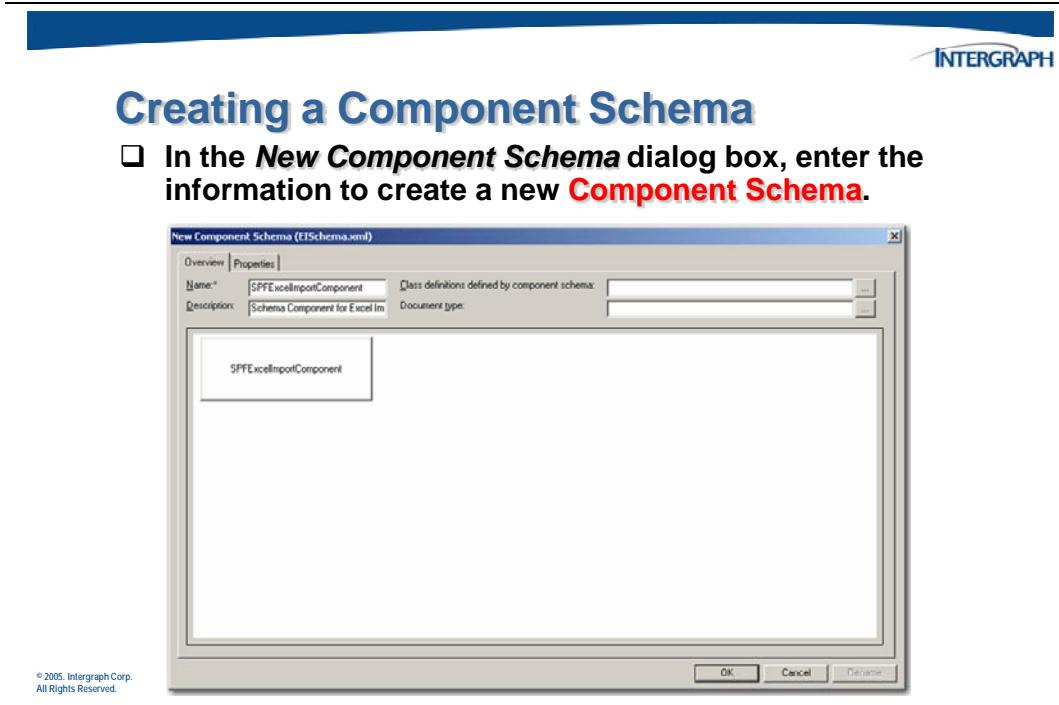
Please follow along with the instructor in creating a new component schema. **Do not close** your Schema Editor window as this activity will continue with additional components.

1. If you are not already logged into your VM session, log on to your operating system as **Administrator** with a password of *spf2008*.
2. Click Start > All Programs > Intergraph SmartPlant Foundation > SmartPlant Schema Component > SmartPlant Schema Editor - Standard to start the Schema Editor.
3. Open the project schema XML file.
 - Click the **File > Open > Configuration File**.
 - Select the **904-EISchema.cfg** file, and click **Open** (Path is D:\Program Files\SmartPlant\Foundation\2008\Models\Core).
 - Use the **Editor** view to perform your modeling tasks.
 - Along the Left side of the window, choose to view the **Schema** tab.
 - Along the top of the window, choose to use the **Editor** viewer.



4. In the Create section of the Editor view, find the CompSchema option. Drag and drop it into the UML view area.

5. Provide the following information for the new component schema:
 - Name – **SPFExcelImportComponent**
 - Description – **Schema Component for Excel Import**



3.4 Class Definitions

Class definitions are typically real-world objects like instruments, equipment, or pipe runs. Class definitions have the following characteristics:

- ❑ Every instance of a class definition is instantiated by a class factory.
- ❑ Every class definition belongs to one and only one component schema.
- ❑ Every class definition has a primary interface definition that defines the set of possible roles (interface definitions) for the class definition. Interface definitions will be discussed in the next section.



Class Definitions

A **class definition (ClassDef)** is a named description of a set of objects that specifies the attributes (data elements) and methods (member functions) for those objects in the schema.

<<ClassDef>>
PIDInstrument

<<ClassDef>>
PIDInstrumentLoop

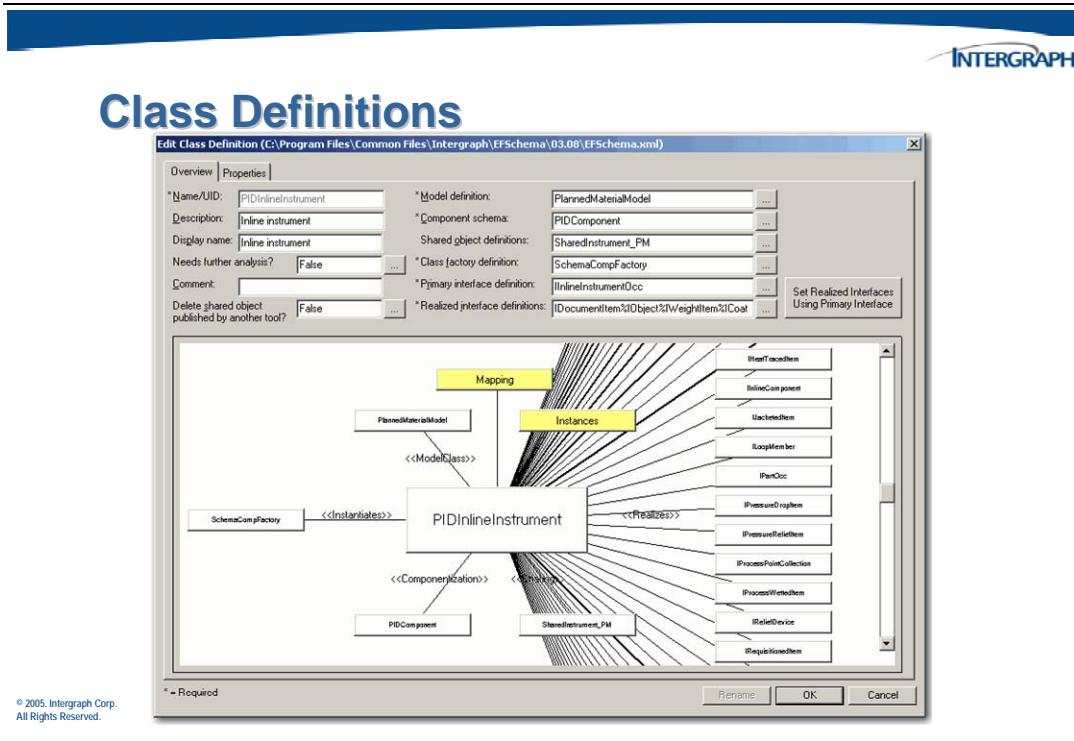
In the schema, ClassDefs can represent physical things, such as pumps and instruments, or conceptual things, such as projects.

ClassDefs offer different roles with which the software can interact. ClassDefs expose or realize their roles through abstract entities called interface definitions.

3.4.1 Properties of a Class Definition

The properties of a class definition include the following:

- Name** – Specifies the name of the class definition.
- Description** – Specifies a description of the class definition.
- Display name** – Specifies the name that you want the user interface to use when displaying the class definition.



- Delete shared object published by another tool** – Specifies what happens when SPF receives a delete instruction for a shared object. SPF either deletes all ClassDefs/UIDs for the shared object or only the ClassDef/UID referenced by the delete instruction. For example, a shared object CV-2001 is published by SmartPlant P&ID as PIDInlineInstrument and by SmartPlant Instrumentation as INDXInstrument. If SmartPlant P&ID publishes a delete instruction and this option is false on the PIDInlineInstrument ClassDef, then the UID for the PIDInlineInstrument object will be terminated in SPF and the INDXInstrument object will remain in the SPF database. If the option is set to true, both UIDs will be terminated in SPF.
- Model definition** – Specifies the model definition to which this class definition belongs. Class definitions can only participate in a sharing relationship if they exist in the same model.

- Component schema** – Specifies the component schema to which the class definition belongs. A component schema is a subdivision of the SmartPlant schema. There is typically one component schema per published document type.
- Shared object definitions** – Specifies the shared object definitions with which this class definition has a sharing relationship. Shared object definitions define the same object in different authoring tools.
- Class factory definition** – Specifies the class factory used to create the class definition. There is currently only one class factory in the SmartPlant schema: SchemaCompFactory.
- Primary interface definition** – Identifies the primary interface definition for the class definition. The primary interface definition defines the set of possible roles for a ClassDef and should imply everything that is known about the ClassDef.
- Realized interface definitions** – Identifies all interface definitions realized by the class definition.
- Set Realized Interfaces Using Primary Interfaces** – Displays the **Identify Realized Interfaces Using Primary Interface** dialog box. This dialog box allows you to select which optional interfaces implied by the class definition's primary interface that you want the class definition to realize.

3.5 Interactive Activity – Creating a Class Definition

Next we will create the class def to represent our new type of object.

The screenshot shows a software interface for creating a new object. At the top left, it says "New Object – The Excel Import Pump". At the top right, there is an "INTERGRAPH" logo. Below the title, there is a button labeled "Excel Spreadsheet". The main area is divided into two sections. On the left, under the heading "Want to map a class ElPump for Storage in SPF:", there is a table with the following columns:

Excel Import Pump	
Name	
Description	
Pump Type	(Picklist)
Tag Prefix	
Tag Sequence Number	
Tag Suffix	
Maximum Pressure	
Maximum Temperature	
Hold Status	(Picklist)

On the right, under the heading "<Comp Schema> SPFExcelImportComponent", there is a table with the following columns:

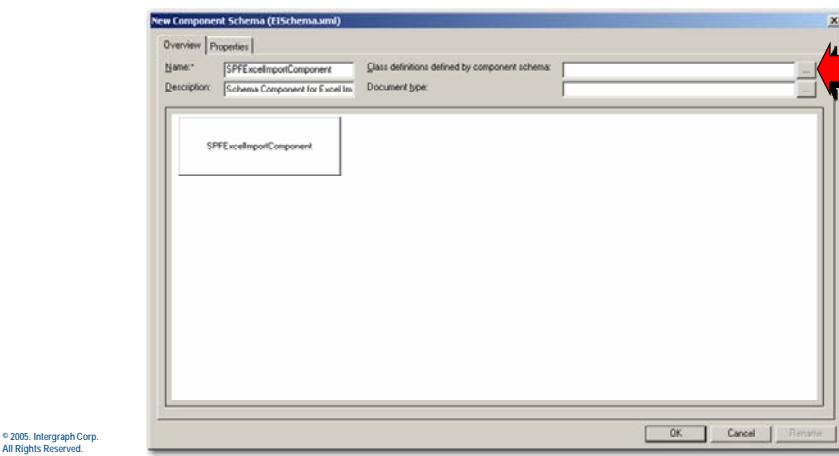
<class def > ElPump

At the bottom left of the interface, there is a small note: "© 2005. Intergraph Corp. All Rights Reserved."

6. From the form where you are creating your component schema, click the browse button beside the ***Class definitions defined by component schema*** field.

Creating a Class Definition

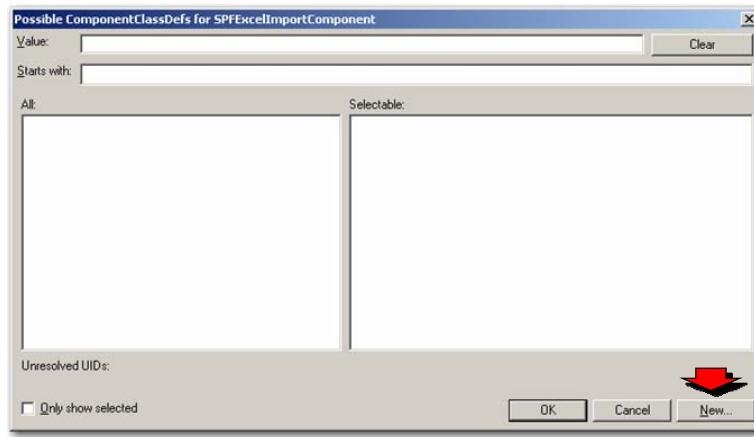
- Choose the browse button to display the list of class definitions that can be associated with this Component Schema.



7. As there currently are not class defs that do not already belong to a component schema, the list is empty. We need to create a new class def. Click the **New** button at the bottom.

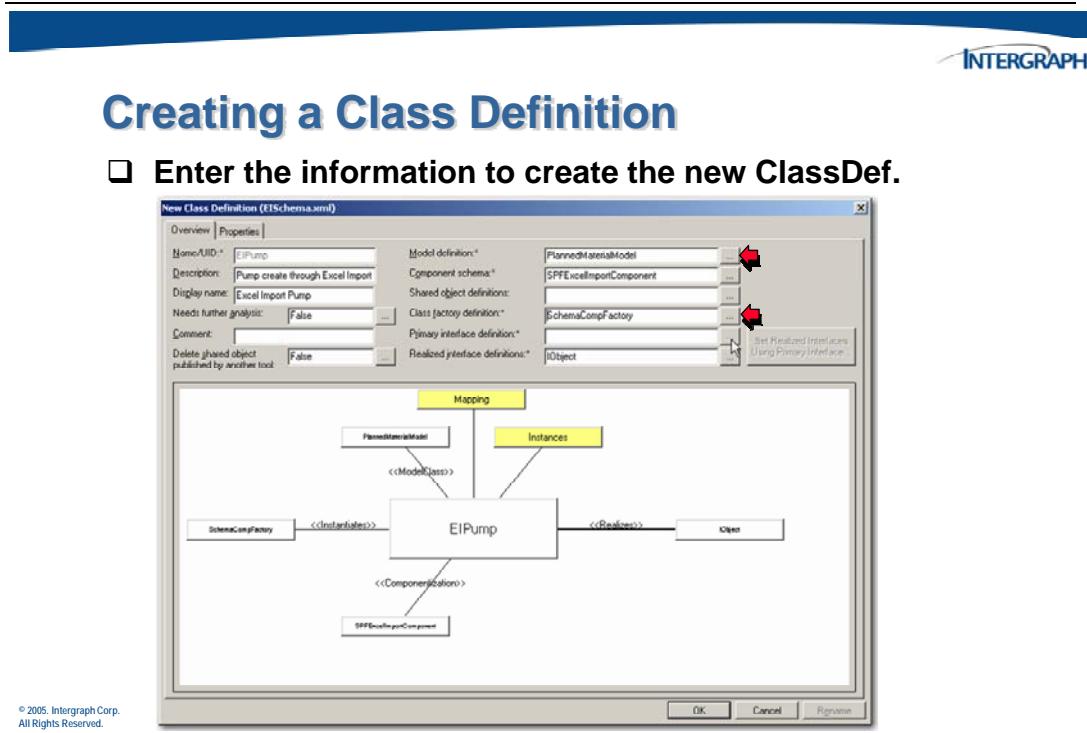
Creating a Class Definition

- Click the **New** button from the **Possible ComponentClassDefs for SPFExcelImportComponent** dialog box to create a new class definition

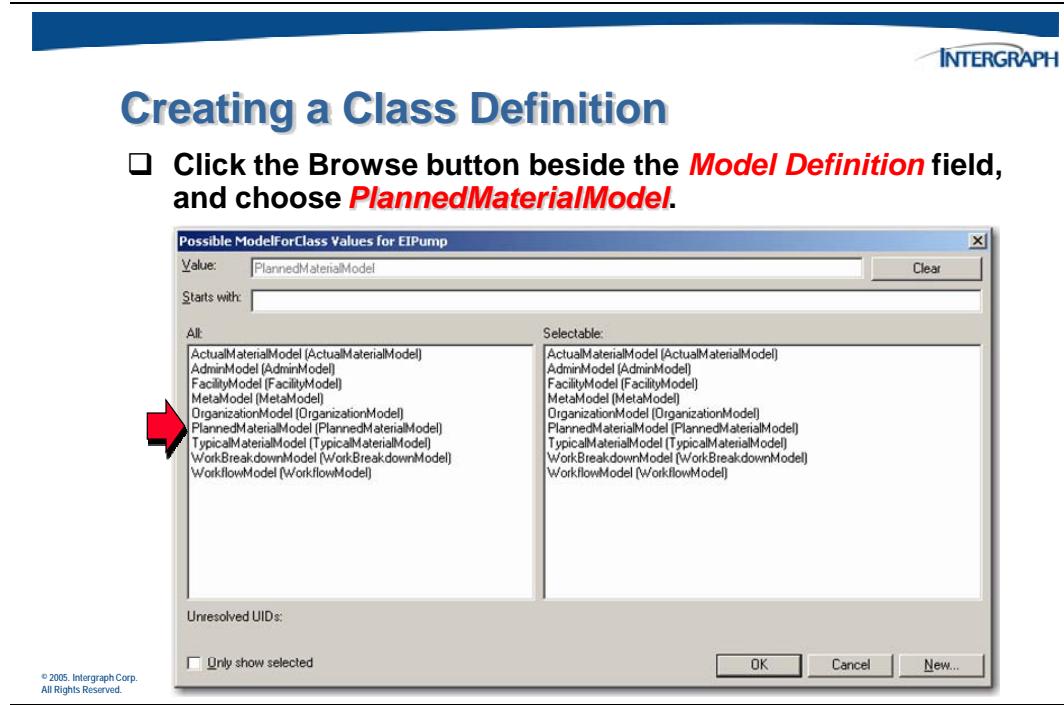


8. Provide the following information for the new class def:

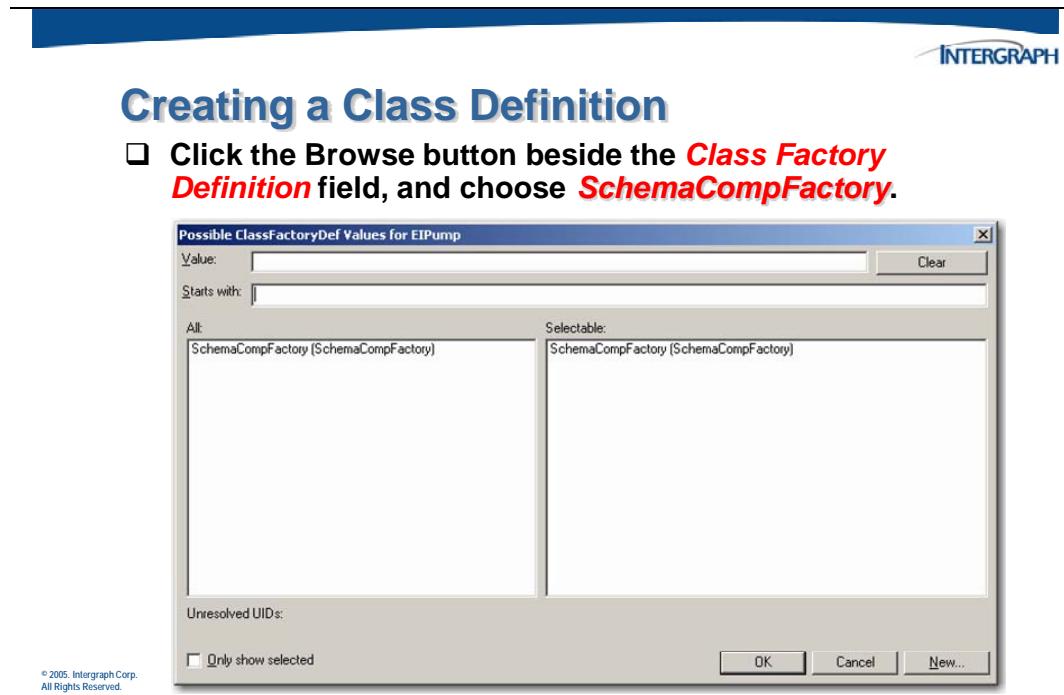
- Name – **EIPump**
- Description – **Pump created through Excel Import**
- Display name – **Excel Import Pump**



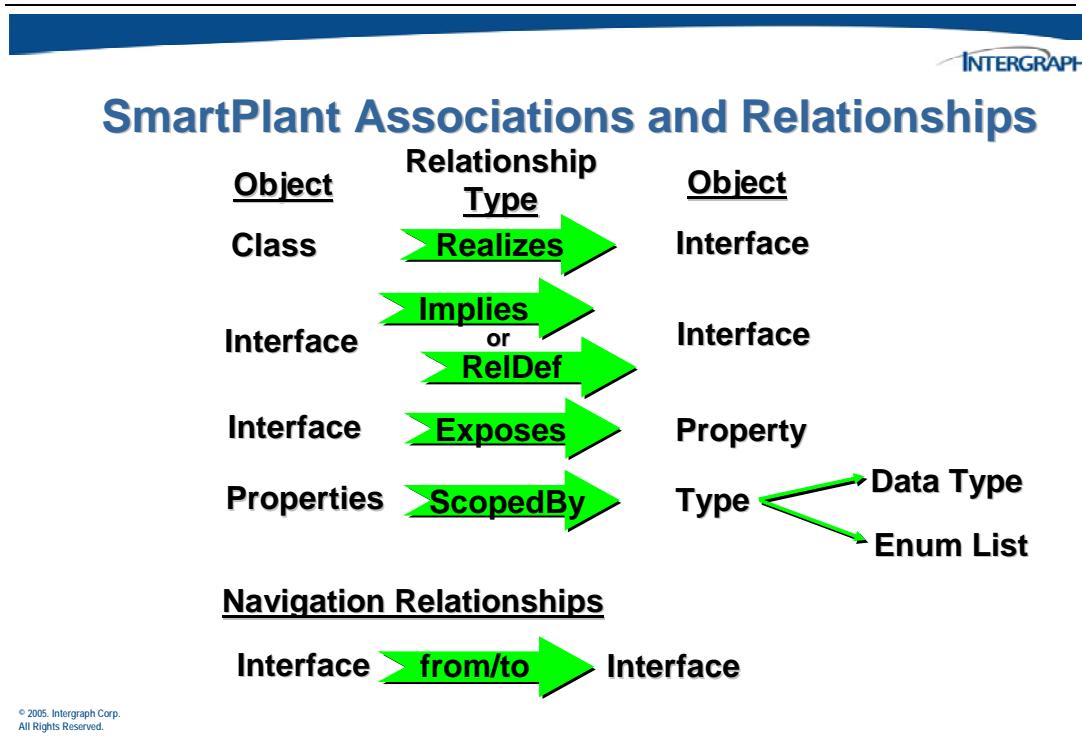
9. Click the browse button beside the **Model Definition** field, and choose the *PlannedMaterialModel* option from the list.



10. Click the browse button beside the **Class Factory Definition** field, and choose the *SchemaCompFactory* option from the list.



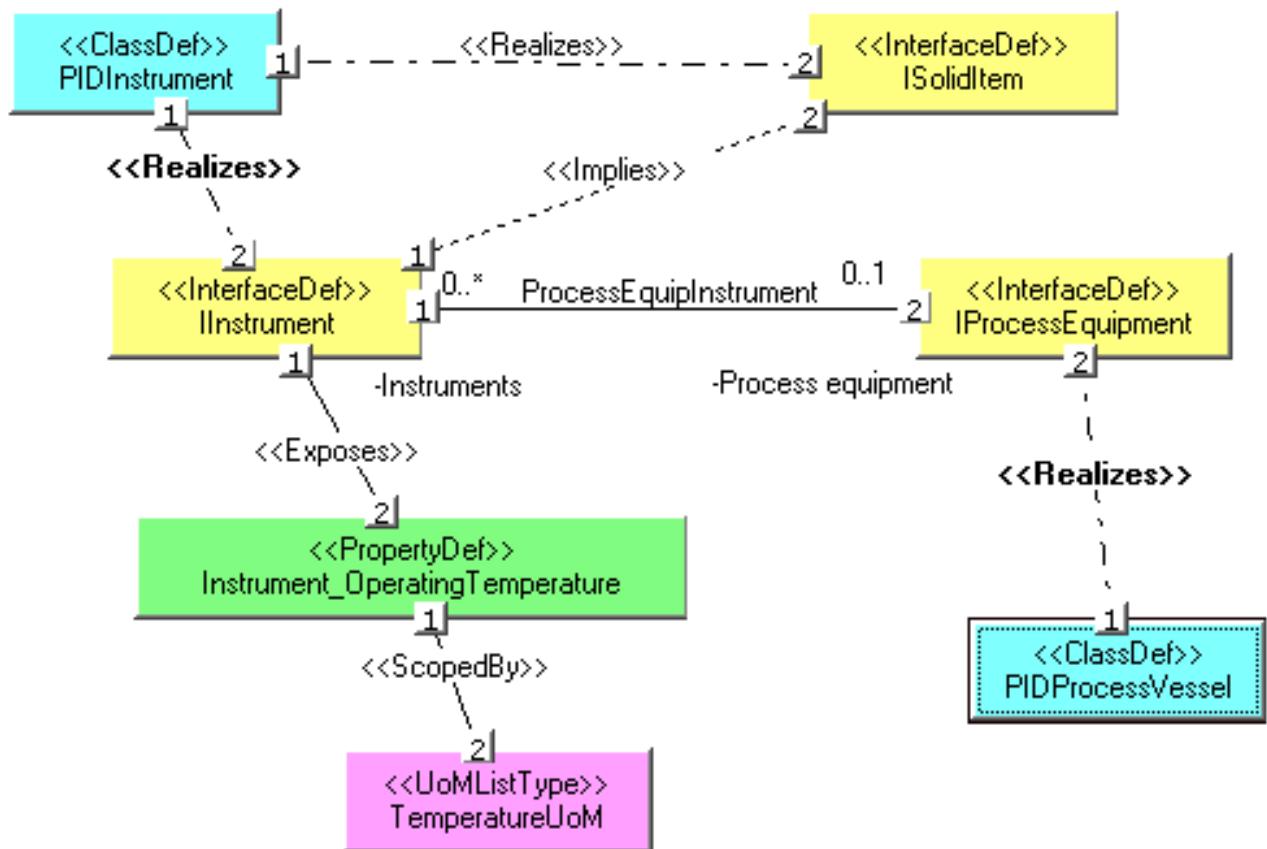
The following graphic is a reminder of the terms that will be important to us as we learn about the relationships between the various pieces of the schema.



A reminder from the previous chapter that the two types of relationships used from interface to interface are:

- ❑ RelDefs – these are user defined relationships defined by the data modeler and are used to associate interfaces that are realized by different class definitions (ClassDef's).
- ❑ Implies – this type of relationship is thought of as a system default relationship. It is used primarily for inheritance from one interface to another interface where both are realized by the **same** ClassDef.

The following example shows how these different types of relationships are used.



3.6 Interface Definitions

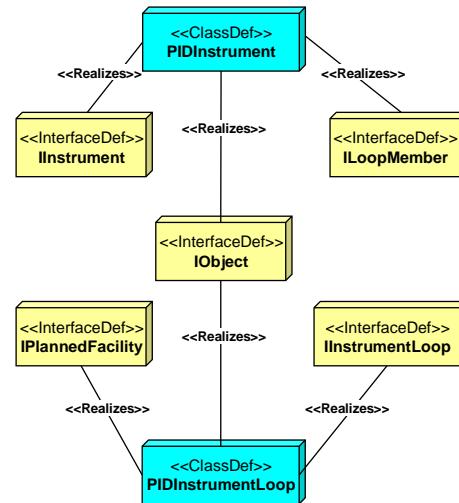
An **Interface Definition** (InterfaceDef) is the most important concept in the meta schema. An interface definition is a named collection of property definitions. Interface definitions expose the property definitions for class definitions. Every interface definition is realized by one or more class definitions. By sharing specific interface definitions, class definitions can also share property definitions, but not the data associated with the properties.

Interface Definitions

An interface definition (InterfaceDef) represents a "role" for a class definition.

Different class definitions can share the same InterfaceDefs, and therefore, the same roles.

For example, all ClassDefs in the schema realize the IObject interface.



© 2005. Intergraph Corp.
All Rights Reserved.

Just as in the real world, the role defines both the properties and relationships of an object. Some interface definitions are defined to carry properties. Some are defined to carry relationships. Others may be defined merely to indicate a role.

Note: SPF uses InterfaceDefs to define what methods are allowed on an object and what workflows are appropriate. These special relationships are not available in the Schema Editor, only within SPF Desktop Client.

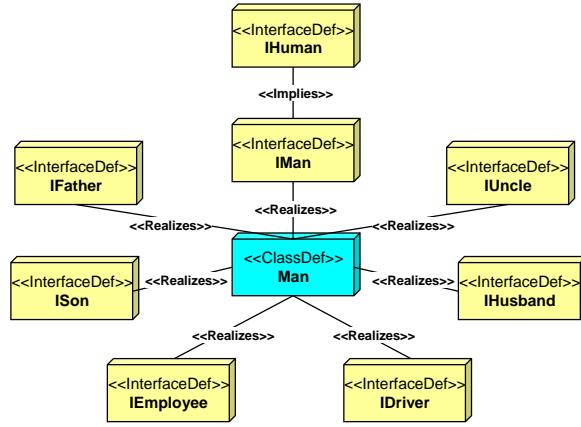
3.6.1 Role Example

The following example looks at the behavior of a class def and interfaces in a generic, non-industry example.

Interface Definitions - Role Example

A class definition can offer up many **roles** to the world through its interface definitions.

For example, a "Man" can expose many roles, such as "Father", "Son", "Uncle", "Husband", "Employee", "Driver", and so on.



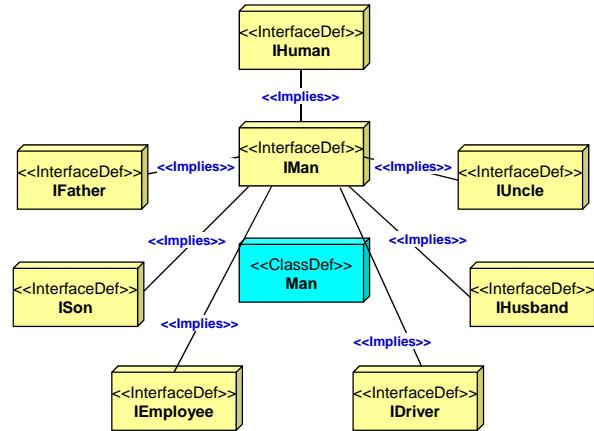
The primary interface will imply all of the interfaces that are realized by the class def.



Interface Definitions - Role Example

A class definition can offer up many **roles** to the world through its interface definitions.

For example, a "Man" can expose many roles, such as "Father", "Son", "Uncle", "Husband", "Employee", "Driver", and so on.



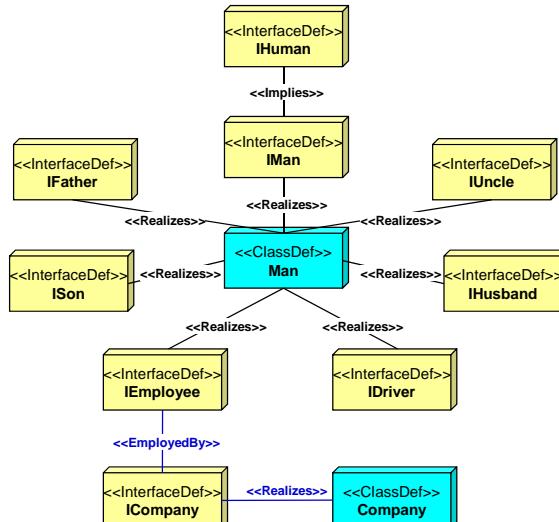
© 2005. Intergraph Corp.
All Rights Reserved.

Associations between classes are made by creating relationships between interfaces.



Interface Definitions - Role Example

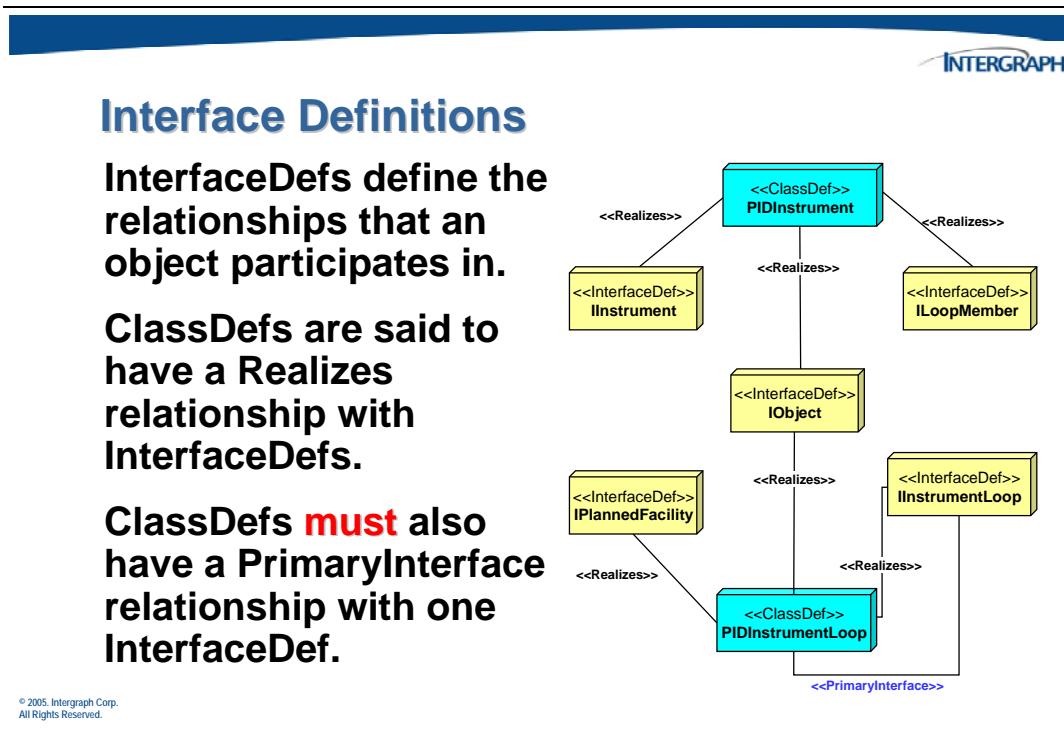
When a company wants to interact with a Man, the company does not care that he is a son or an uncle. The company only cares that the "Man" class exposes the "IEmployee" role for the company to interact with.



© 2005. Intergraph Corp.
All Rights Reserved.

3.6.2 Interface Definitions and Relationships

There are two relationships defined between ClassDef and InterfaceDef. The first (and most common) is the “**Realizes**” relationship. We say that a ClassDef *Realizes* an InterfaceDef. The Realize relationship may be required or optional. If it is required, the Schema Component will insist that any instance of that ClassDef must realize an instance of the required InterfaceDef.



© 2005, Intergraph Corp.
All Rights Reserved.

The second relationship defined between ClassDef and InterfaceDef is the “**Primary Interface**” relationship. The primary interface definition defines the set of possible roles for a class definition and should imply everything that is known about that class definition. Primary interface definitions, rather than class definitions, actually define the objects in the schema. Class definitions fulfill a subset of the roles defined by the primary interface definition.

The most relevant purpose for the **Primary InterfaceDef** is to define the full and complete “roleness” of an object. It defines what an object “is” and everything that is known about it. The *IEquipmentOcc* defines the what a piece of equipment is in the schema and the *IInstrumentOcc* defines what an instrument is. This definition is completed through the use of the **Implies** relationships. When you consider the complete hierarchy of Implies under a Primary InterfaceDef, you know what that object “is”.

This definition of “roleness” is used in several ways in SmartPlant:

1. **Validation** – ClassDefs may not realize any InterfaceDef outside of the implies hierarchy of the primary InterfaceDef.
2. **Sharing** – ClassDefs defined as shared must have the same primary InterfaceDef or must imply the same. PFDProcessEquipment and PIDProcessEquipment have the same primary InterfaceDef and are defined as shared. EQDReactorVessel does not have the same primary InterfaceDef, but it does imply the same; i.e. **IReactorVesselOcc implies IEquipmentOcc**. Therefore, EQDReactorVessel is defined as shared with PFDProcessEquipment and PIDProcessEquipment.
3. **Mapping** – It is possible to map the SmartPlant Schema into the tool Schema by ClassDef. But to do so, you might end up mapping the same interfaces and properties over and over. For example, if your application retrieved equipment tags, you would want to retrieve PFD, P&ID, and EQD. To map by ClassDef, you would end up mapping to PFDProceeeEquipment, PIDProcessEquipment, and dozens of EQD* ClassDefs. If you map to IEquipmentOcc, you only do so once and can retrieve any object which realizes that InterfaceDef.

So to summarize primary interface definitions:

1. A primary interface definition is a concept of “roleness” so for example, *IProcessEquipementOcc* outlines all process equipment but *PFDProcessEquipment* does NOT realize all of its hierarchy only the interfaces it needs for the class.
2. ALL realize relationships between a classdef and interfaces are scoped by the primary interface def AND ITS CHILD IMPLIES relations downwards.
3. An implies or primary interface concept is for modelling and mapping “a schema concept” and bears no relation to instantiation in SPF (only realizes does this).
4. An interface must be realized to be used on a form.

One of the interface definitions that identify a role is **IObject**. This is the most common interface in the system and has the properties of *unique identifier*, *a name*, and a *description*. **Every class definition in SmartPlant must realize IObject**.

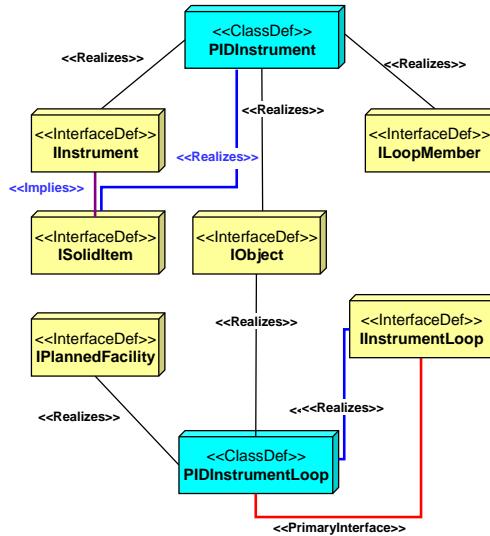
There are no relationships between class definitions, only between interface definitions. That is because *only the interface definitions are exposed to the outside world*. The underlying reason for modeling class definitions and interface definitions instead of class definitions and relationships is because of the complexity of the relationships that can be formed.

In a traditional class data model, relationships go directly from one class definition to another. When the class hierarchy gets moderately deep, however, the relationships become very hard to understand. The consequence is that the modeler moves the relationships “up” in the hierarchy, which causes ambiguity in defining just what the relationship represents. By grouping similar characteristics together and exposing them as an abstract entity called an interface definition, it is easier to maintain precision in the relationships.

Interface Definitions

InterfaceDefs can imply other InterfaceDefs. If an InterfaceDef implies another InterfaceDef, then any ClassDef that realizes the first InterfaceDef can also realize the implied InterfaceDef.

For example, IInstrument implies the ISolidItem. Therefore, any ClassDef, such as PIDInstrument, that realizes IInstrument must also realize ISolidItem.



© 2005, Intergraph Corp.
All Rights Reserved.

There is a relationship defined between InterfaceDefs called “**Implies**”. If an InterfaceDef such as *IFather* implies another interface such as *IMale*, then any object that has the *IFather* interface must also have the *IMale* interface. The Schema Component enforces these rules.

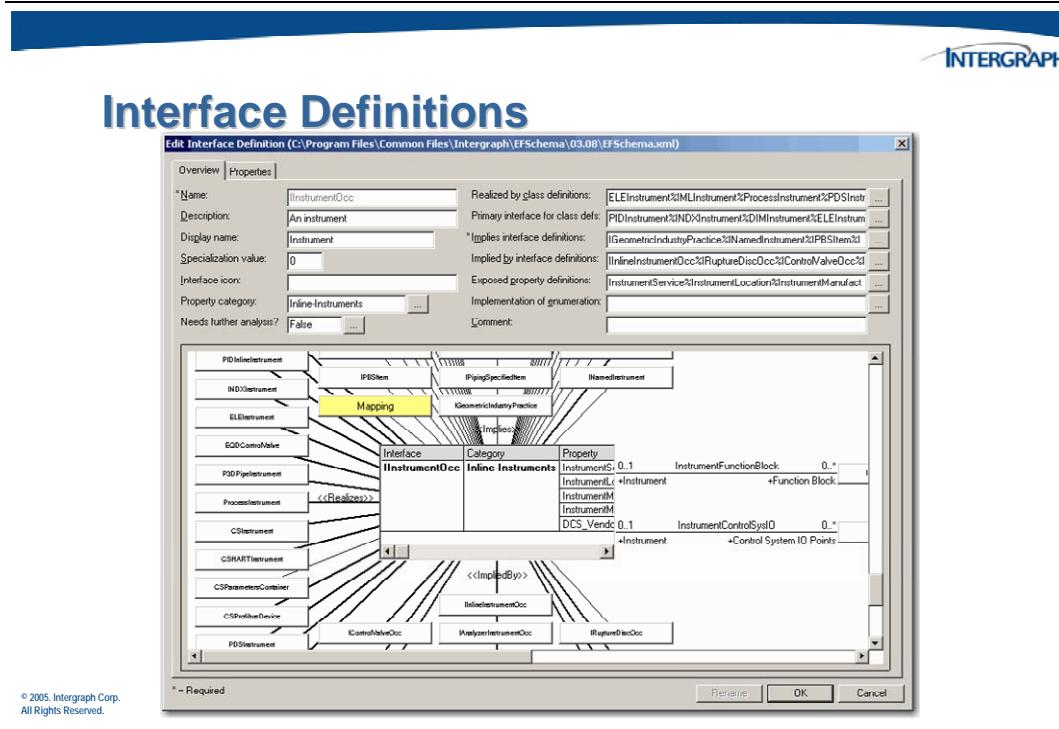
The set of interface definitions directly or indirectly implied by an interface definition defines the “role” for that interface definition. The Implies relationships between InterfaceDefs, also referred to as implied interfaces, also defines the schema hierarchy, which allows you to navigate through the schema almost endlessly.

In the schema, every interface definition, except for *IObject*, should imply the *IObject* interface definition.

3.6.3 Properties of an Interface Definition

Interface definitions have the following properties:

- Name** – Specifies the name of the interface definition.
- Description** – Specifies a description for the interface definition.
- Display name** – Specifies the name that you want the user interface to use when displaying the InterfaceDef.

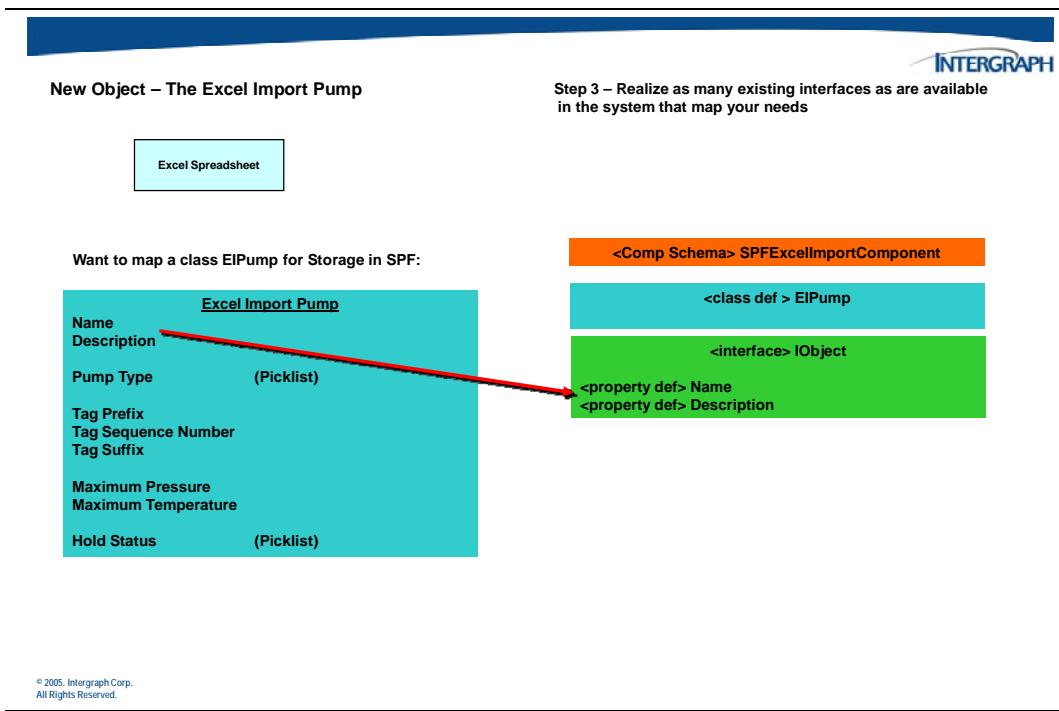


- Specialization value** – Determines the extent to which this interface identifies the object. The higher the number, the more specific the interface is for defining the object. The primary interface for a class definition should have the highest specialization value. An interface definition should always have a higher specialization value than any of the interface definitions that it implies. However, this is not currently checked during validation.
- Interface icon** – Specifies the icon that you want to use to represent this InterfaceDef in the Schema Editor user interface.
- Property category** – Defines the property category for properties exposed by this interface. The property category helps organize properties in the **Properties** window in SmartPlant Foundation and SmartPlant P&ID.
- Realized by class definitions** – Lists the class definitions that realize this InterfaceDef.

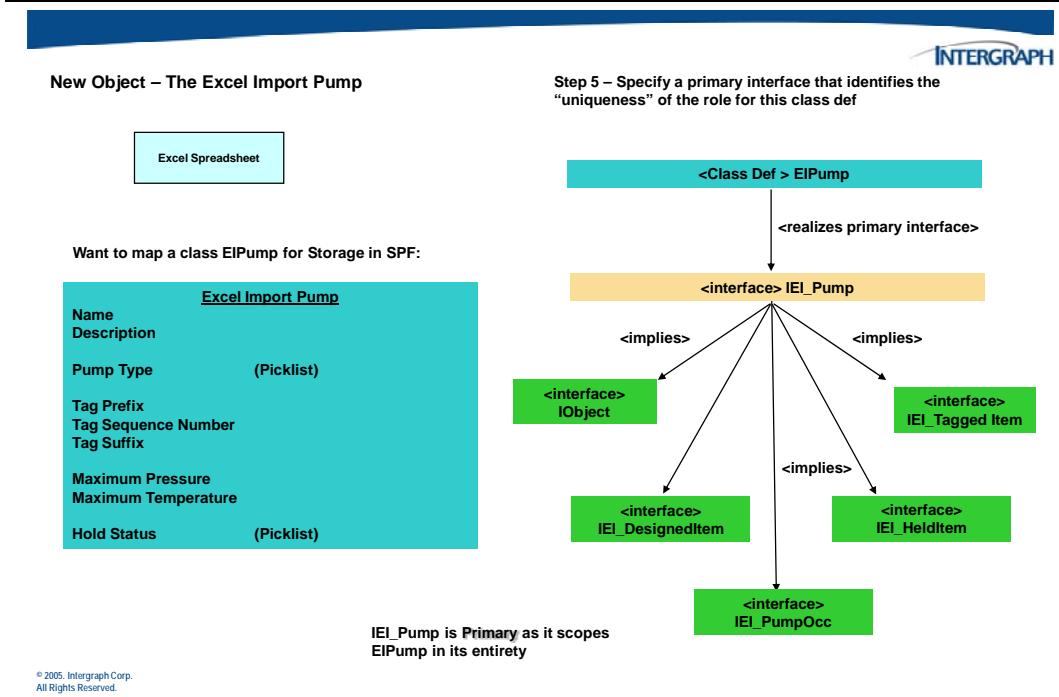
- ❑ **Primary interface for class defs** – Lists the class definitions for which this InterfaceDef is the primary interface. Primary interfaces should imply everything that is known about a particular class definition.
- ❑ **Implies interface definitions** – Lists all the InterfaceDefs that this InterfaceDef implies. All class definitions that realize this InterfaceDef must also realize all the InterfaceDefs this InterfaceDef implies.
- ❑ **Implied by interface definitions** – Lists all other InterfaceDefs that imply this one.
- ❑ **Exposed property definitions** – Lists all property definitions that this InterfaceDef exposes.
- ❑ **Implementation of enumeration** - Defines a relationship between an enumerated entry that represents a classification hierarchy and the primary interface of the objects the enumerated entry classifies. For example, a primary interface, such as IPIDDrawing, should have a relationship with the P&ID enumerated entry from the DocCategories enumerated list.

3.7 Interactive Activity – Creating Interface Definitions

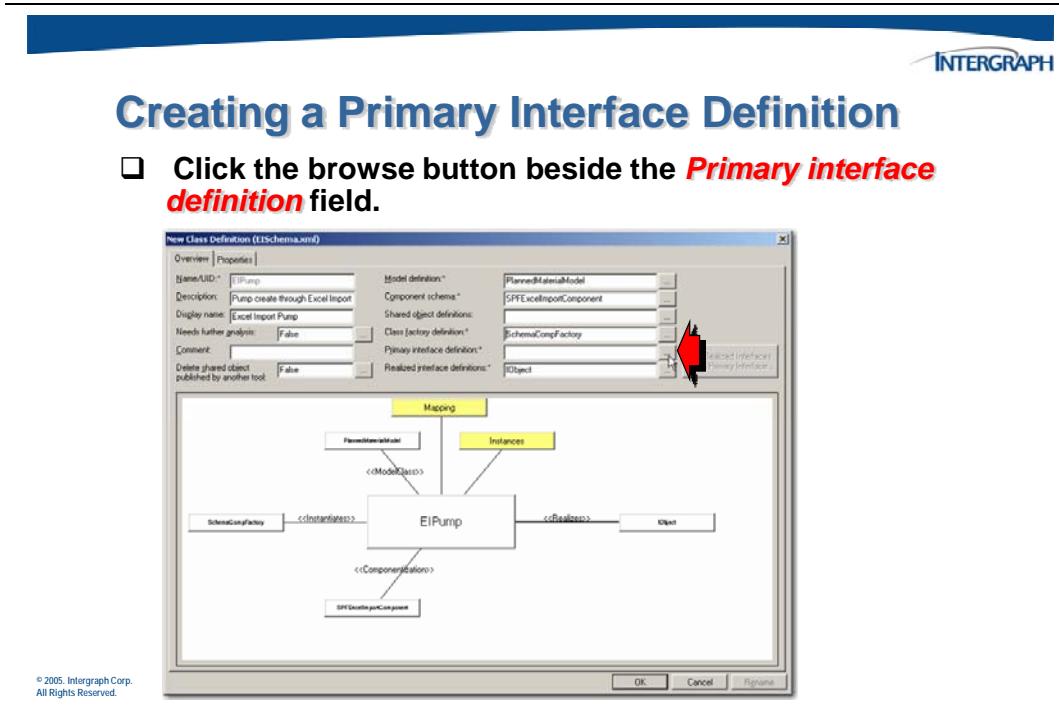
Next we will create an interface that will act as the primary interface of our new class def.



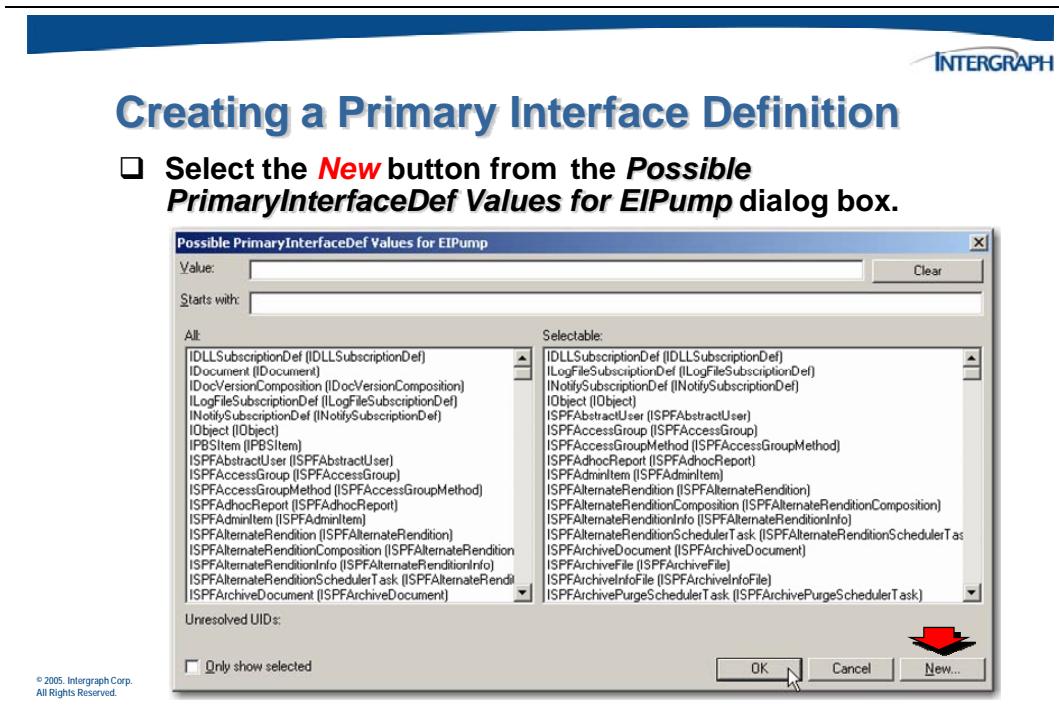
Eventually, we will create implies relationships between our primary interface and each of the other interfaces realized by our class def.



- Click the browse button beside the **Primary interface definition** field on the **New Class Definition** form.

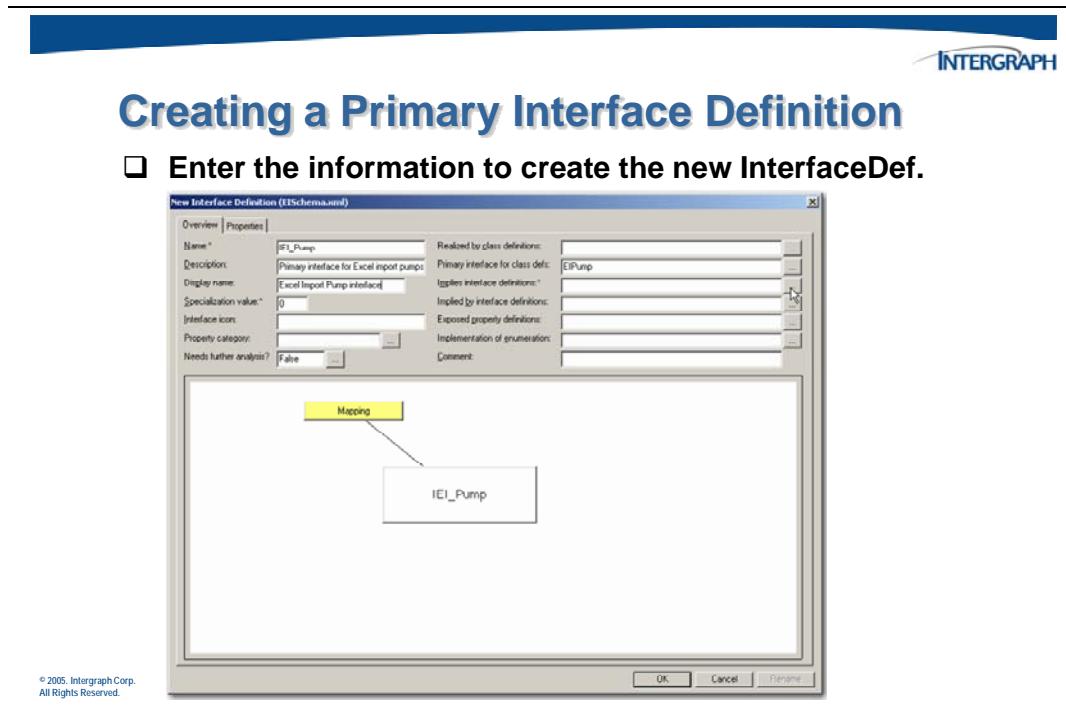


12. The interface we want to use is not the list because it does not exist yet. Click the *New* button.

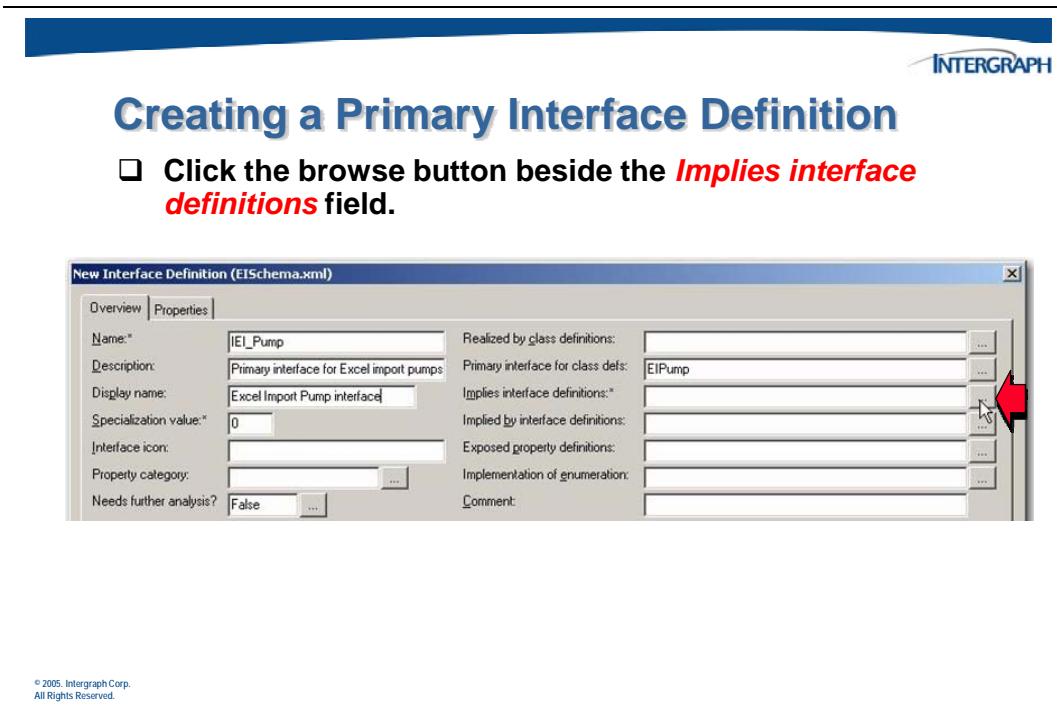


13. Provide the following information for the new interface:

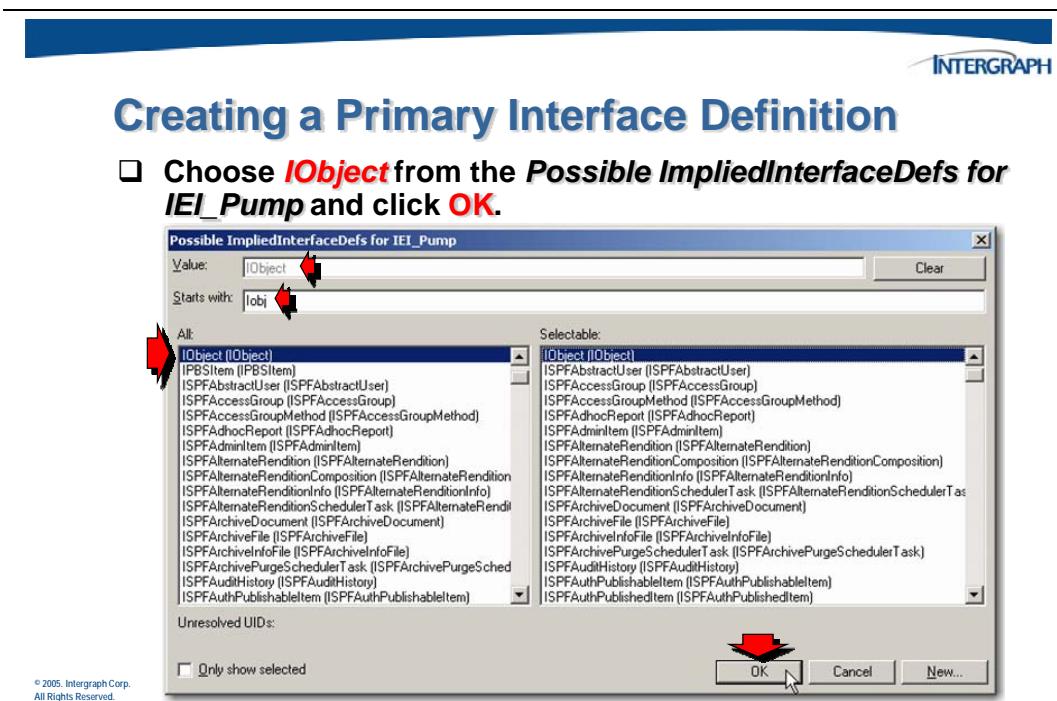
- Name – **IEI_Pump**
- Description – **Primary interface for Excel import pump**
- Display name – **Excel Import Pump interface**



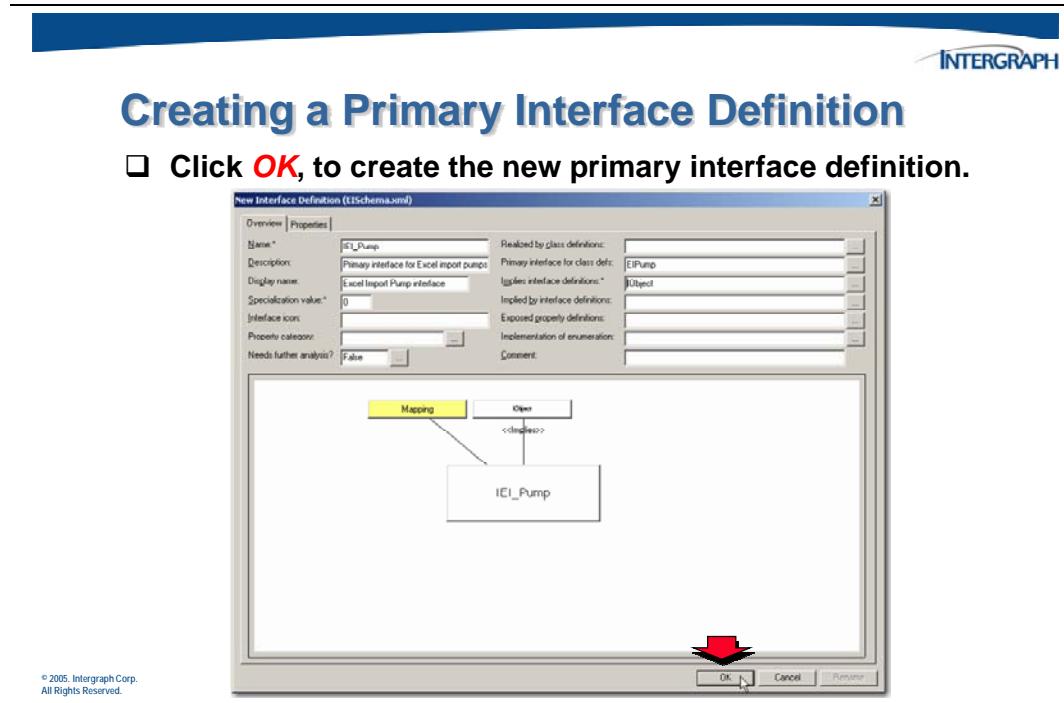
14. The ***Implies interface definitions*** field is a required field. Click the browse button to find an interface to be implied by the new interface.



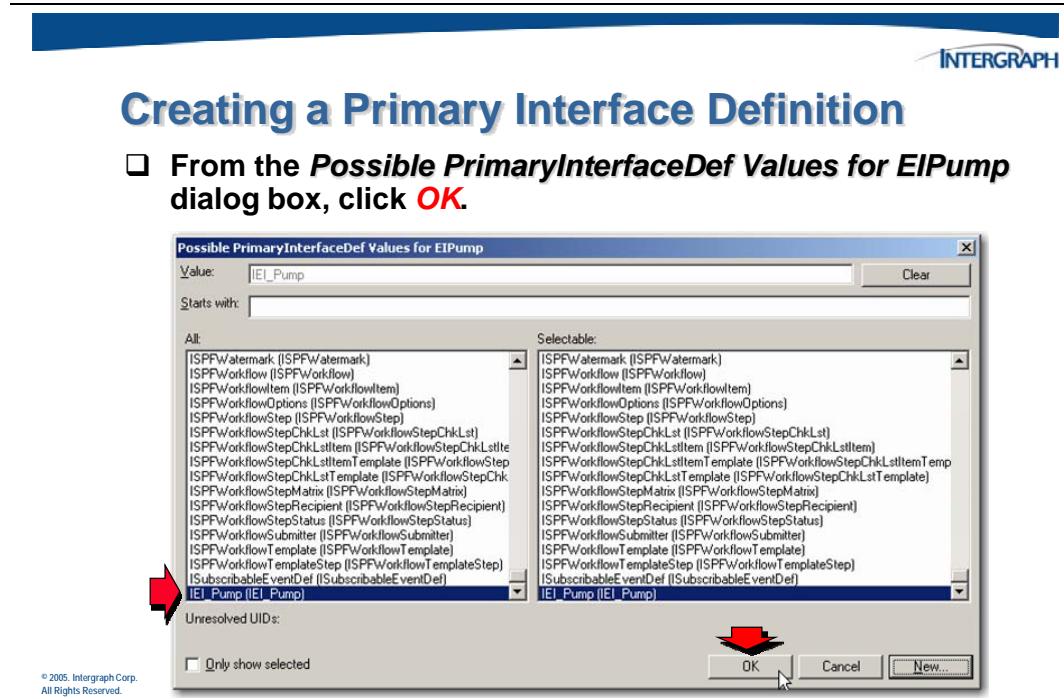
15. Eventually, this interface will imply all the interface realized by our new class def, but at this point, the only one of those interfaces that exists is IObject.



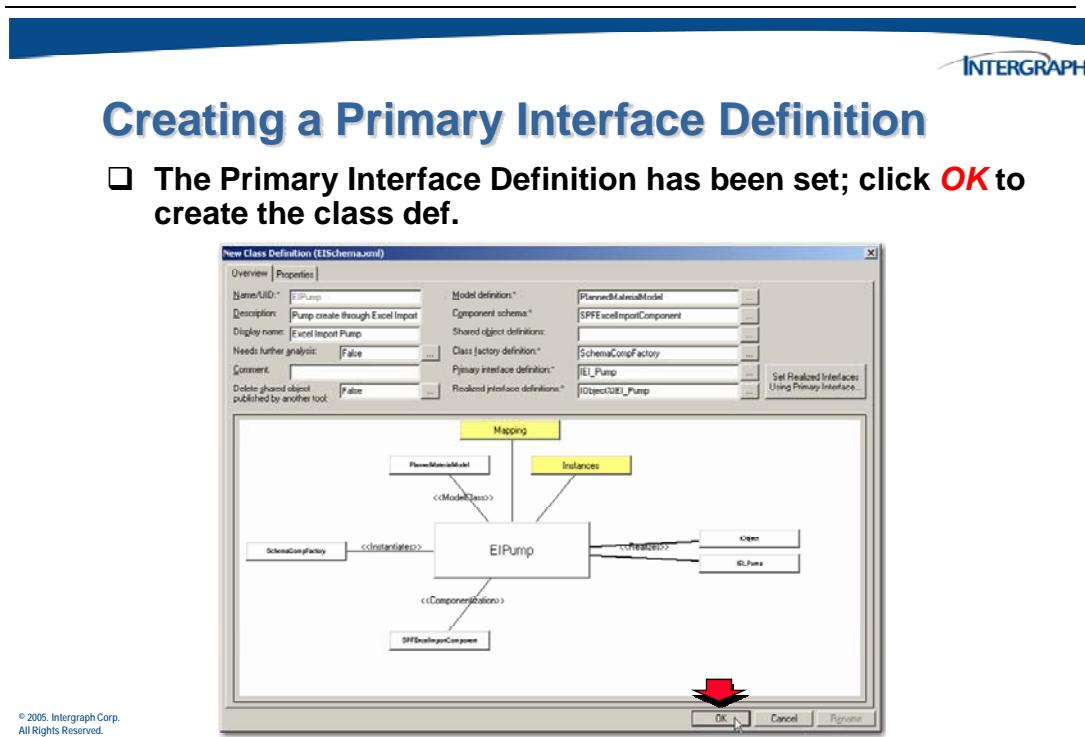
16. This is enough information to create the interface. We can provide more information later. Click **OK** to create the interface.



17. The new interface is already selected in the *Possible PrimaryInterfaceDef Values for EIPump* dialog box. Click **OK** to accept the selection.



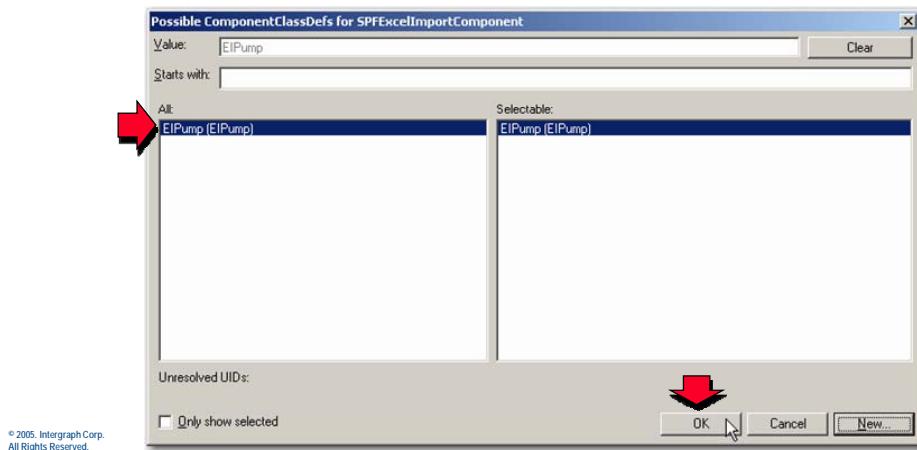
18. The new interface name now appears as the *Primary interface definition* field.
All required information has been provided. Click **OK** to create the class def.



19. The new class def is selected automatically in the *Possible ComponentClassDefs for SPFEExcelImportComponent* dialog box. Click **OK** to accept that selection.

Creating a Component Schema

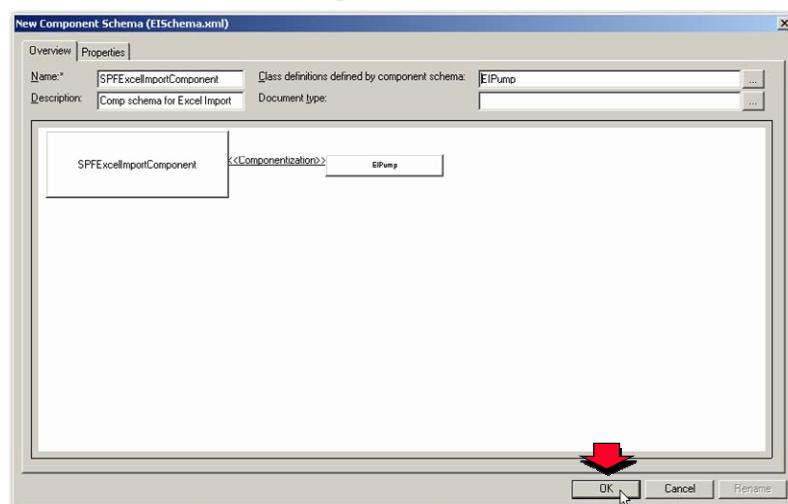
- From the *Possible ComponentClassDefs for SPFEExcelImportComponent* dialog box, choose **EIPump**, and click **OK**.



20. Click **OK** to finish creating the new component schema.

Creating a Component Schema

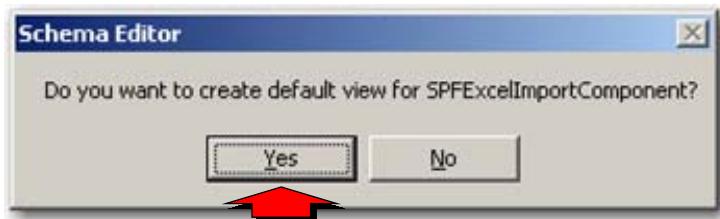
- To create the new **Component Schema**, click **OK**.



21. When prompted, choose to create a default UML view of the component schema.

Creating a Component Schema

- To create a default UML view for the **SPFExcelImportComponent** component schema, click **Yes**.

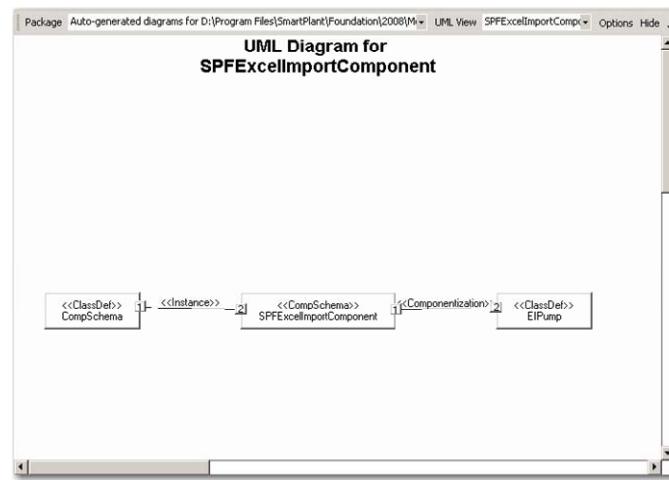


© 2005, Intergraph Corp.
All Rights Reserved.

The component schema and its first-level relationship will appear in the UML view.

Creating a Component Schema

A default UML diagram for the **SPFExcelImportComponent** is created.



© 2005, Intergraph Corp.
All Rights Reserved.

22. Change the search criteria to the *base classes* and refresh the display. Find the new *EIPump* class def in the **ClassDef** section. Choose the **Table (Hor)** view and review the interface associated with the class def and the properties exposes by those interfaces.

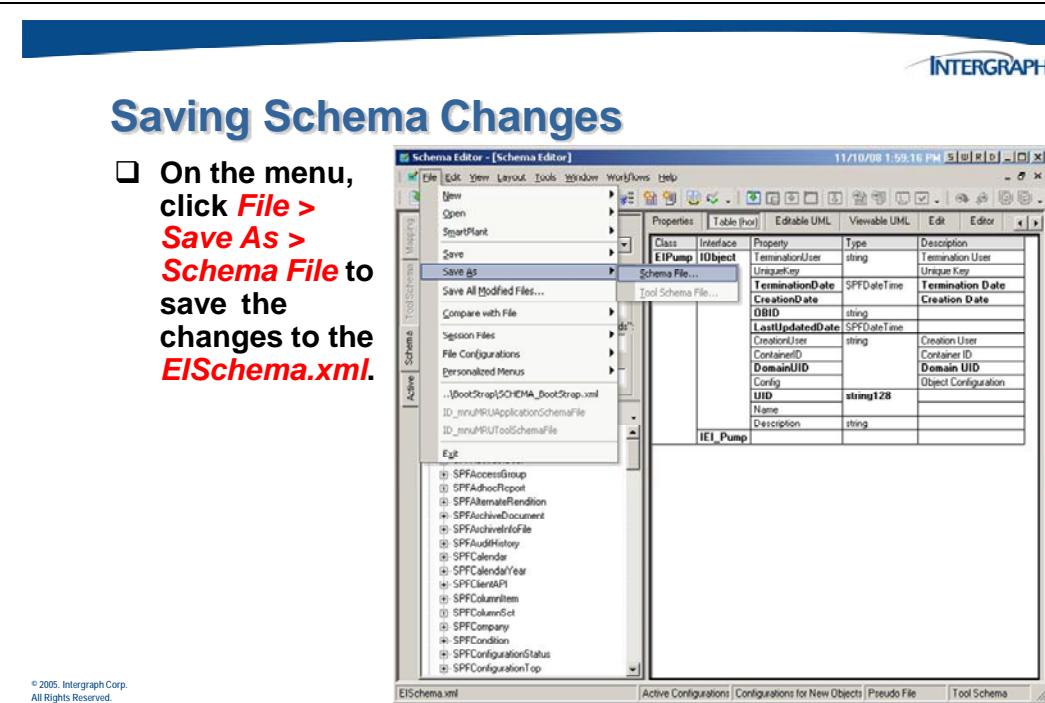
Class	Interface	Property	Type	Description
EIPump	IObject	TerminationUser	string	Termination User
		UniqueKey		Unique Key
		TerminationDate	SPFDatetime	Termination Date
		CreationDate	SPFDatetime	Creation Date
		OBID	string	
		LastUpdatedDate	SPFDatetime	
		CreationUser	string	Creation User
		ContainerID		Container ID
		DomainUID		Domain UID
		Config		Object Configuration
		UID	string128	
		Name	string	
		Description	string	

ClassDef

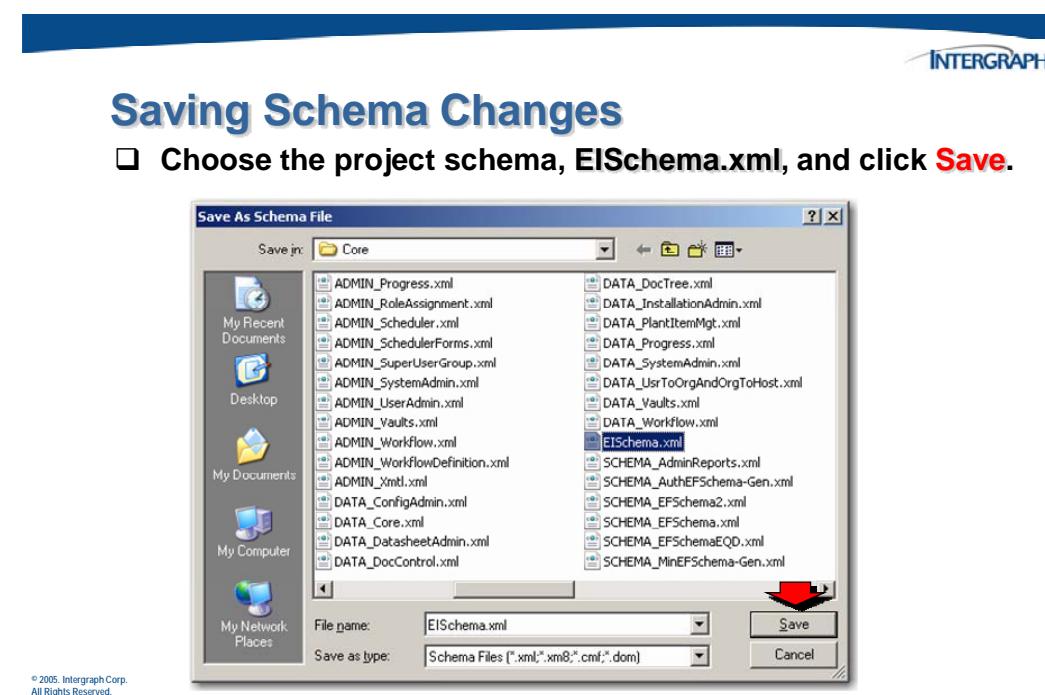
- EIPump
- SPFAuditAuditUser
- SPFAuditAuditGroup
- SPFAuditAuditReport
- SPFAuditAuditTrendline
- SPFAuditArchiveDocument
- SPFAuditArchiveFile
- SPFAuditHistory
- SPFCalendar
- SPFCalendarYear
- SPFClientAPI
- SPFColumnItem
- SPFColumnSet
- SPFCompany
- SPFCondition
- SPFConfigurationStatus
- SPFConfigurationTop

© 2005, Intergraph Corp.
All Rights Reserved.

23. Save the changes you made to the *EISchema* file. Click **File > Save As > Schema File**.



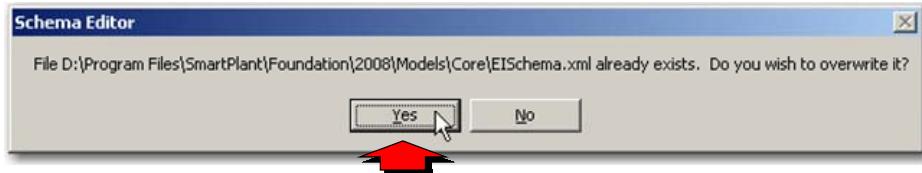
24. Find and select the *EISchema.xml* file.



25. When prompted, choose to overwrite the file.

Saving Schema Changes

- Click **Yes** to save the changes to the project schema.



3.8 Activity 1 – Reviewing Schema Concepts

Complete the Chapter 3 – Activity 1 in the SmartPlant Foundation 2008 (4.2) Modeling and Mapping Activity workbook.

3.9 Activity 2 – Creating Objects and Relationships with Schema Editor

Complete the Chapter 7 – Activity 2 in the SmartPlant Foundation 2008 (4.2) Modeling and Mapping Activity workbook.

4

C H A P T E R

Creating Properties, Enumerated Lists and Relationships

4. Creating Properties, Enumerated Lists, and Relationships

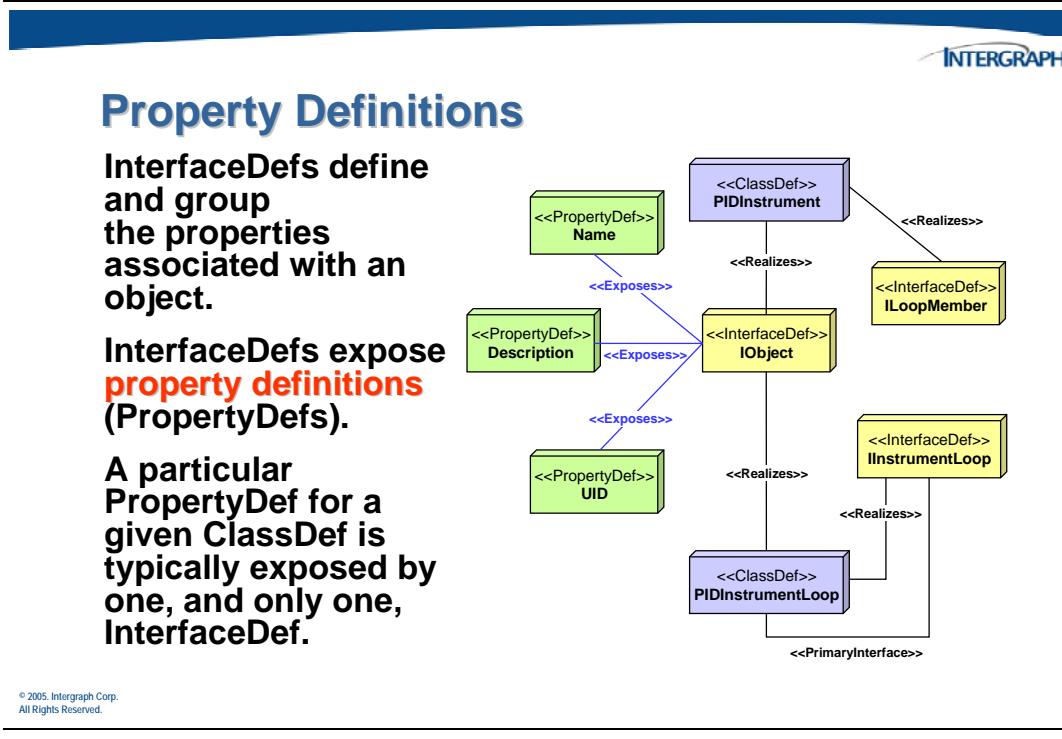
This chapter covers creating attributes, or properties. Properties can be added to the interfaces that have been created to define a “role” for the associated class definition.

In some cases, there may be a requirement to use “picklists” from a user interface. Picklist, or list boxes, can be accomplished by creating an enumerated list type that contains entries called enum enums.

Finally, this chapter will cover the ability to create additional relationships, or relationship definitions, between custom interfaces and existing interfaces.

4.1 Property Definitions

All ***property definitions*** for an object are exposed through its interface definitions and never directly by the object, or class def. The property definitions that apply to a particular interface definition are defined by the *Exposes* relationship between objects of type InterfaceDef and objects of type PropertyDef.



For example, the IObject interface definition exposes, among other properties, the following basic properties:

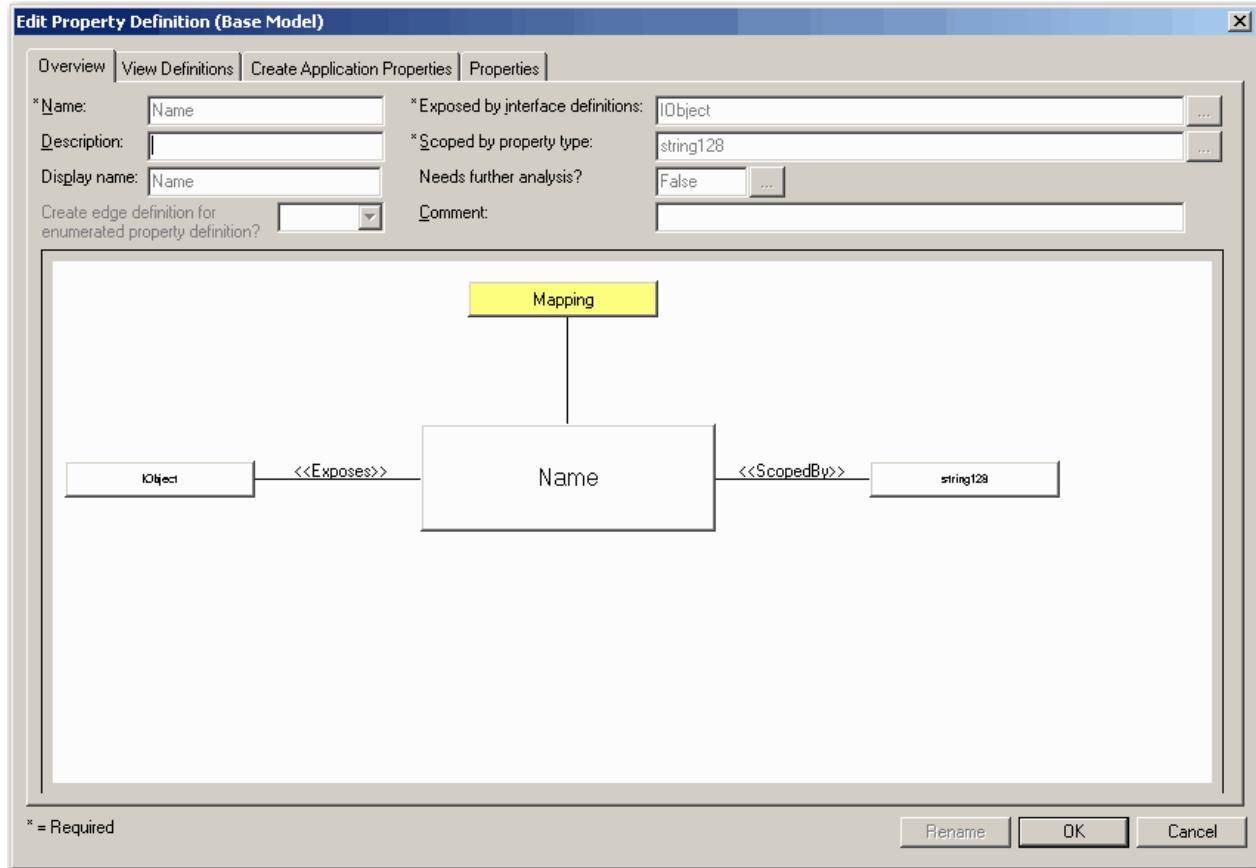
- UID** - Unique identifier for the object. This identifier is only required to be unique within the SmartPlant.
- Name** - Name of the object
- Description** - Description of the object

Some interface definitions exist to hold properties, and other interface definitions exist to form relationships. Some interface definitions are defined for neither properties nor relationships – they are defined only for a role.

4.1.1 Properties of a Property Definition

Property definitions have the following attributes in the schema:

- Name** – Specifies the name of the PropertyDef.
- Description** – Specifies a description for the PropertyDef.
- Display name** – Specifies the name that you want the user interface to use when displaying the PropertyDef.



- Exposed by interface definitions** – Lists the interface definition that exposes this PropertyDef. PropertyDefs are exposed by one and only one InterfaceDef.
- Scoped by property type** – Specifies the property type that scopes this PropertyDef. You can select from standard property types, like string, double, integer, Boolean, and so on, or you can select an enumerated list or unit of measure list to scope this property.

4.2 Property Types

Each property type is actually a class definition in the meta schema that realizes IPropertyType, which allows it to be a scoping property type for property definitions.

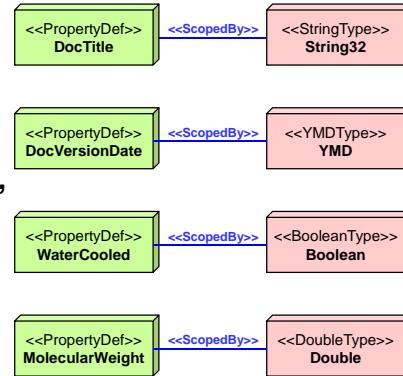


Property Types

Property types define the set of possible values for a PropertyDef.

The ScopedBy relationship definition specifies the property type that defines acceptable values, or scopes, a particular property definition.

Every property definition is scoped by one and only one property type. All properties of that property definition must be of that property type.

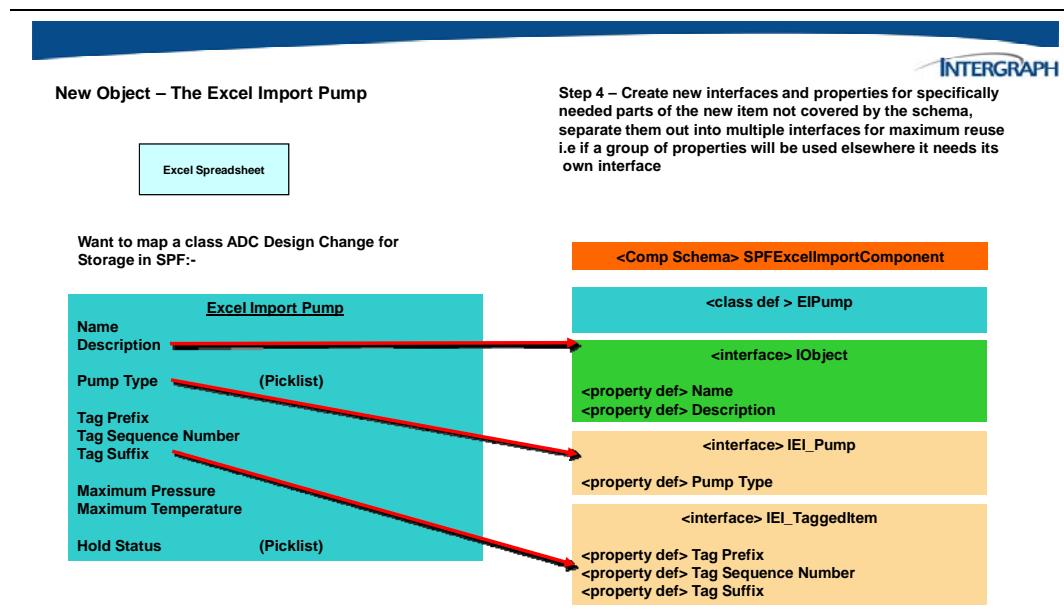
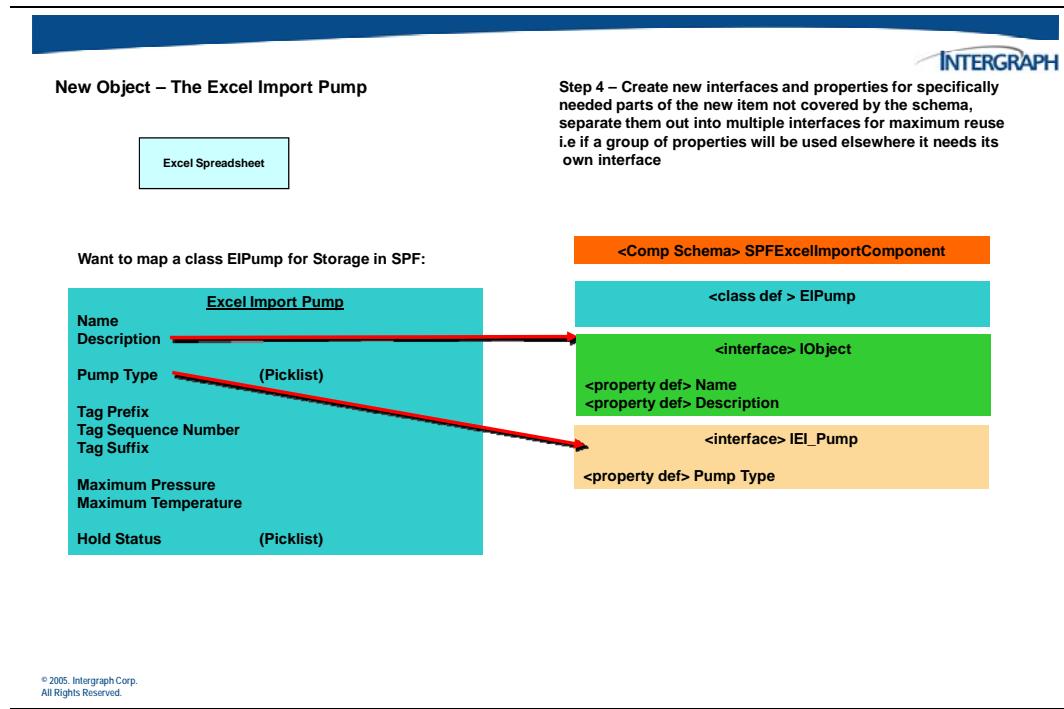


© 2005, Intergraph Corp.
All Rights Reserved.

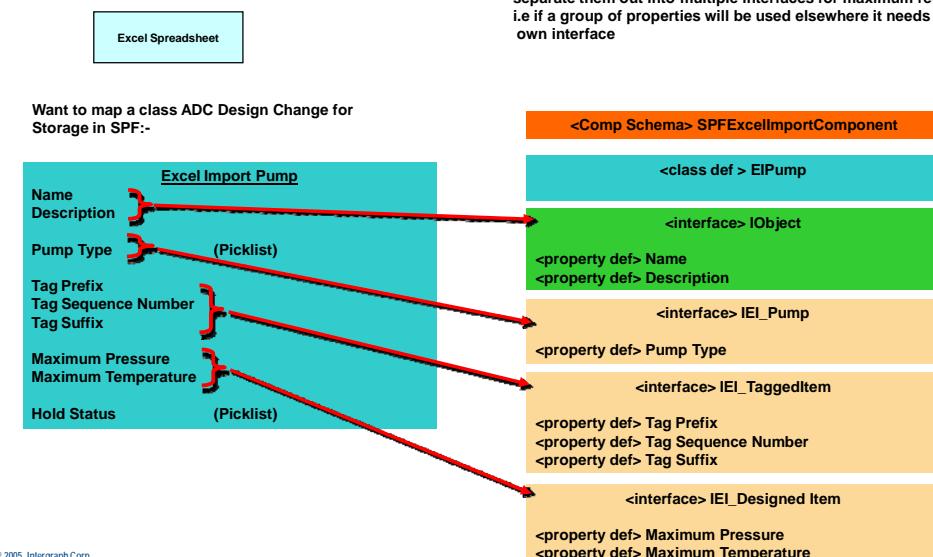
Basic varieties of property types in the SmartPlant schema include the following:

- Boolean** – Specifies that the property can have only one of two values: True or False.
- Double** – Specifies that the property is a double-precision floating point number.
- String** – Specifies that values for the property can contain alphanumeric strings with some punctuation, such as periods (.), underbars (_), question marks (?), and so on.
- YMD** – Specifies that the property value is a date (year, month, day).
- EnumListType** – Specifies that property values are defined by an enumerated list.
- UoMList** – Specifies that values for the property can be a numeric value (double) combined with a unit of measure from the appropriate unit of measure list, along with some optional criteria.

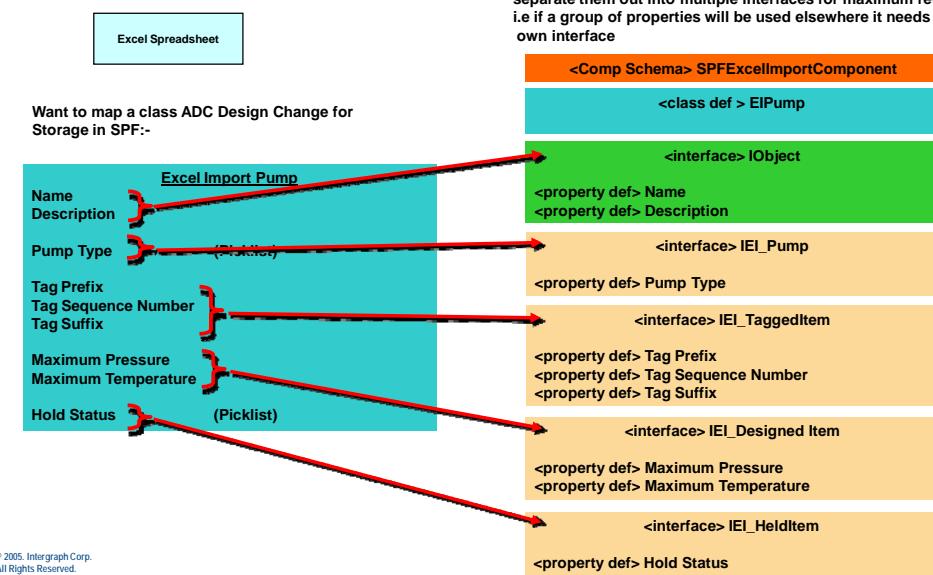
4.3 Interactive Activity – Creating Property Definitions



New Object – The Excel Import Pump



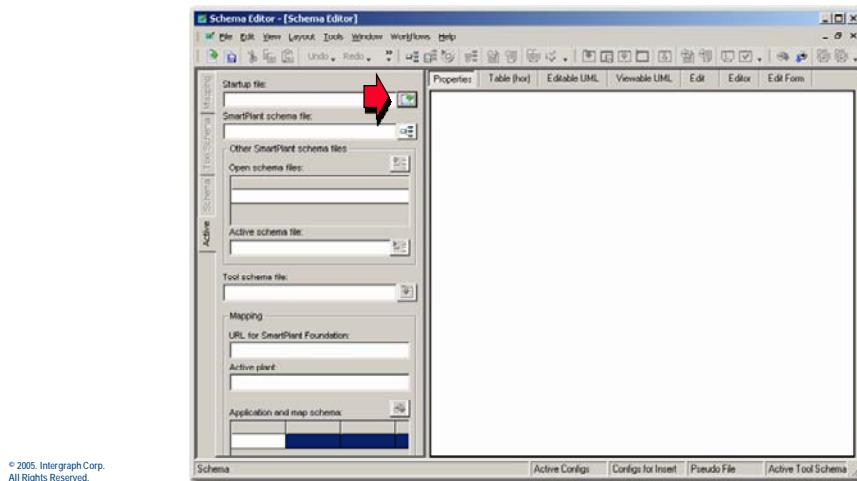
New Object – The Excel Import Pump



1. Open the Schema Editor, and click the *Open Starting File* button to find your configuration file to open.

Create a Property Definition

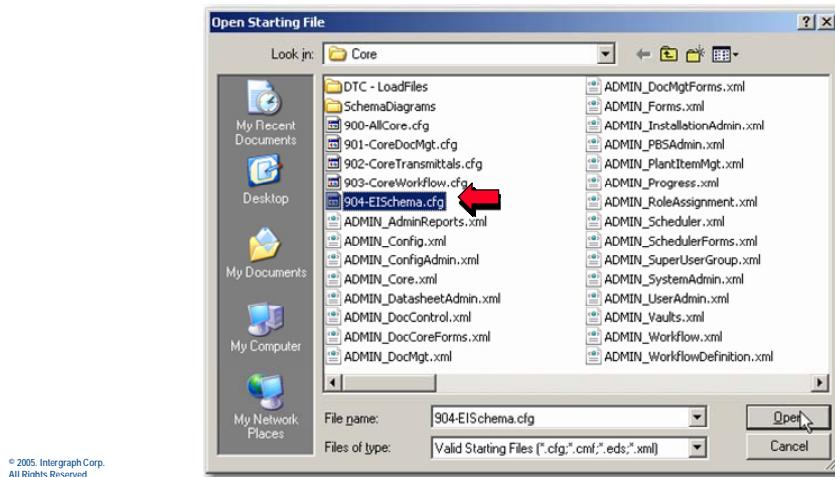
- Open the Schema Editor, and click the *Open Starting File* button.**



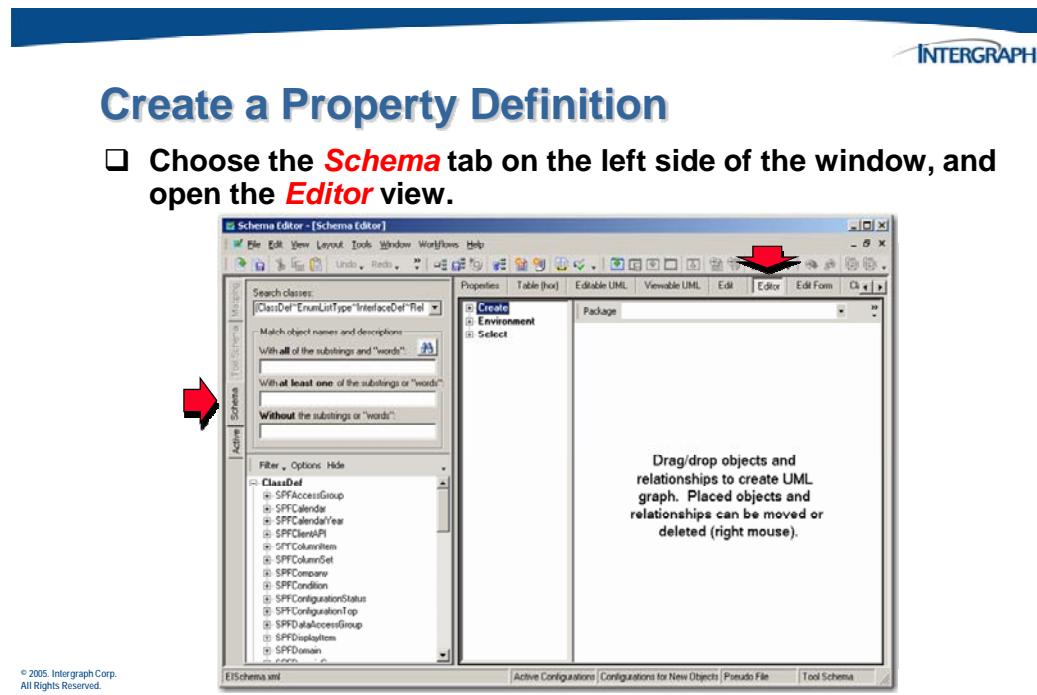
2. Find the **904-EISchema.cfg** file, and click the *Open* button.

Create a Property Definition

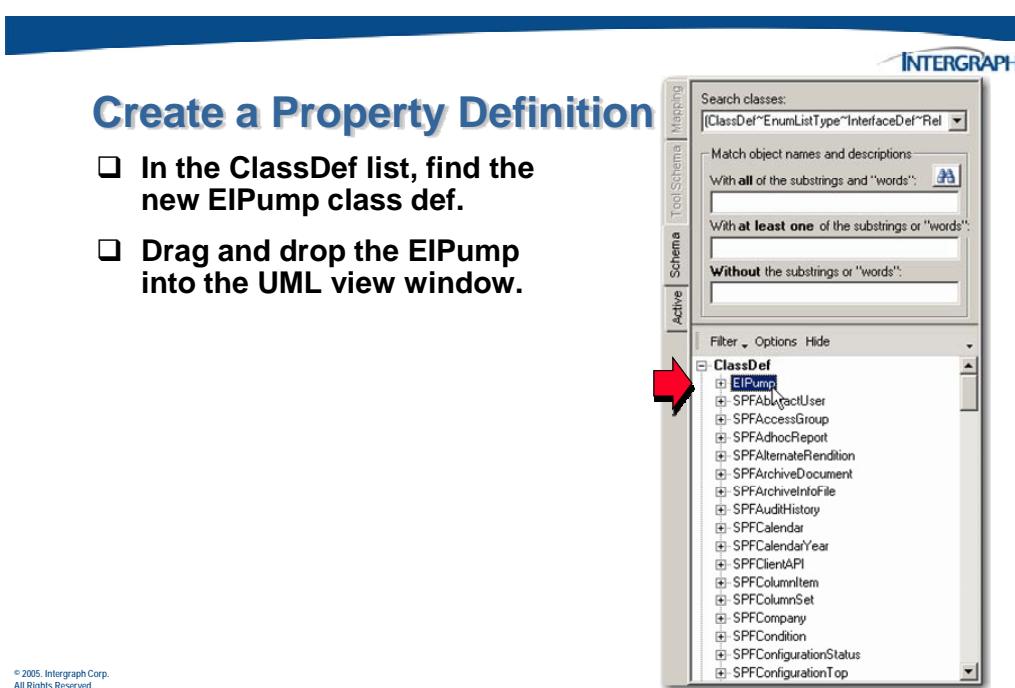
- In the *Open Starting File* dialog, select the **904-EISchema** configuration, and then click *Open*.**



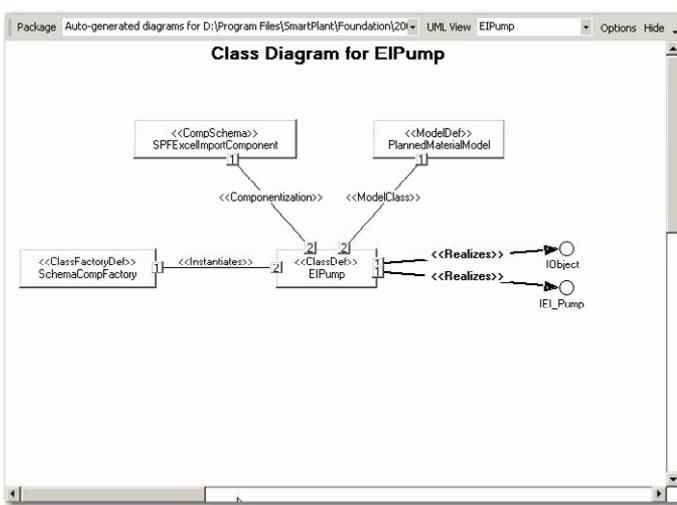
3. Using the tabs along the left-hand side of the window, open the **Schema** tab. Then using the tabs along the top of the window, open the **Editor** view. **Note:** You may need to select an object in the tree (bottom of left-hand pane) to populate the **Create**, **Environment**, and **Select** lists in the **Editor** view.
-



4. From the **ClassDef** list, find the **EIPump**, and drag and drop it into the UML window.



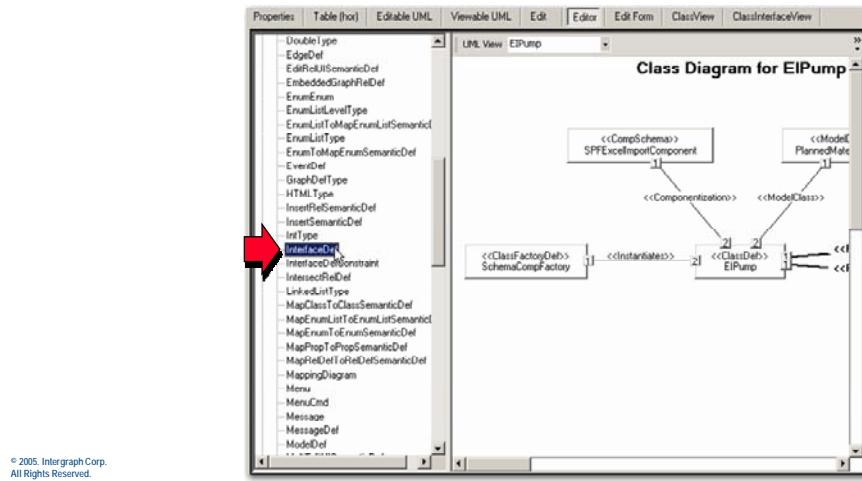
5. When prompted, click **Yes** to create a default view of the class def.



6. From the *Create* list, find *InterfaceDef*. Drag it into the UML window.

Create a Property Definition

- In the *Create* list, find *InterfaceDef*, and drag and drop it in the UML window.



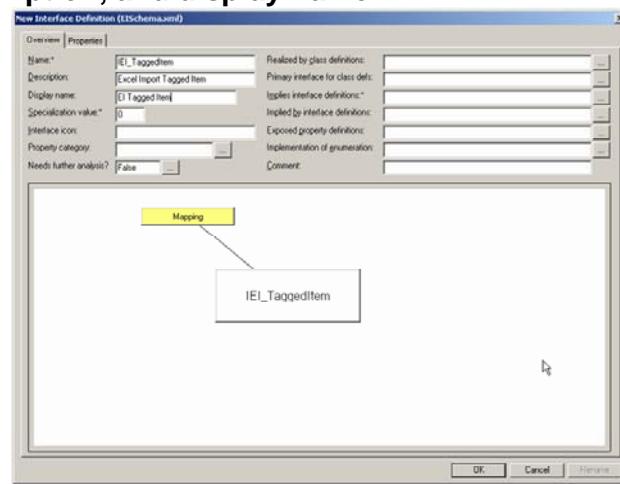
7. Create a new interface with the following information.

- Name** – IEI_TaggedItem
- Description** – Excel Import Tagged Item
- Display Name** – EI Tagged Item



Create a Property Definition

- Create a new *IEI_TaggedItem* interface. Provide a name, description, and display name.**

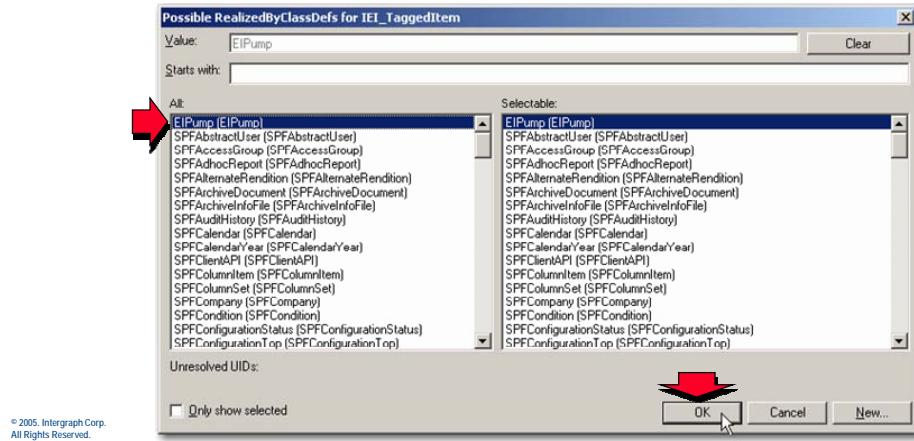


© 2005, Intergraph Corp.
All Rights Reserved.

8. Click the browse button beside the ***Realized by class definitions*** field, and choose the ***EIPump*** class def from the list. Click ***OK*** to close the dialog box.

Create a Property Definition

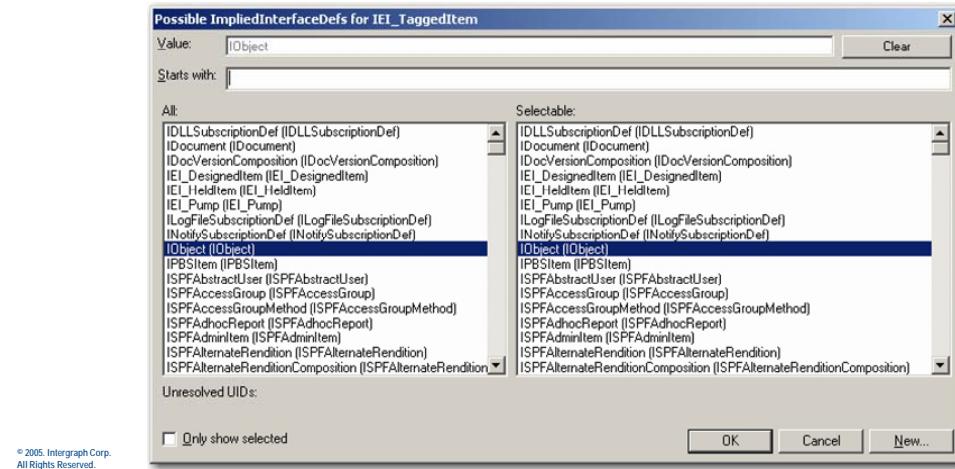
- Click the browse button beside the ***Realized by class definitions*** field, and choose the ***EIPump*** class def. Click ***OK***.



9. Click the browse button beside the ***Implies interface definitions*** field, and choose ***IObject*** from the list of interfaces. Click ***OK*** to close the dialog box.

Create a Property Definition

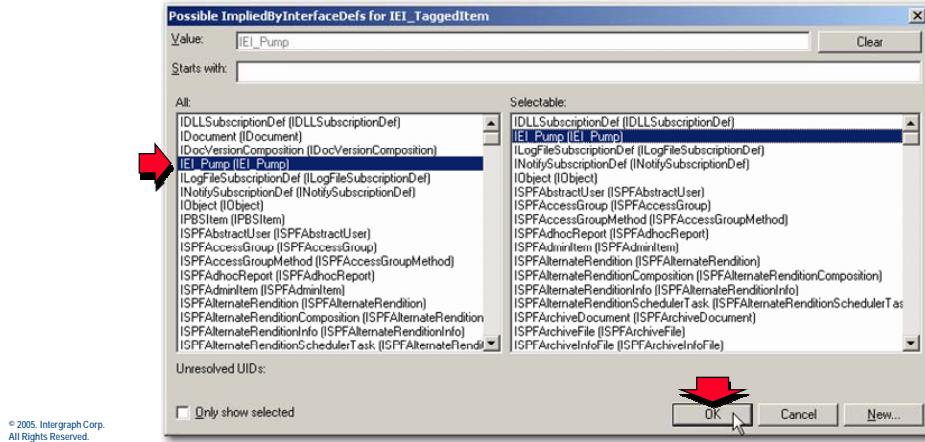
- Click the browse button beside the ***Implies interface definitions*** field, and choose the ***IObject*** interface. Click ***OK***.



10. Click the browse button beside the ***Implied by interface definitions*** field, and choose the ***IEI_Pump*** interface from the list. Click ***OK*** to close the dialog box.

Create a Property Definition

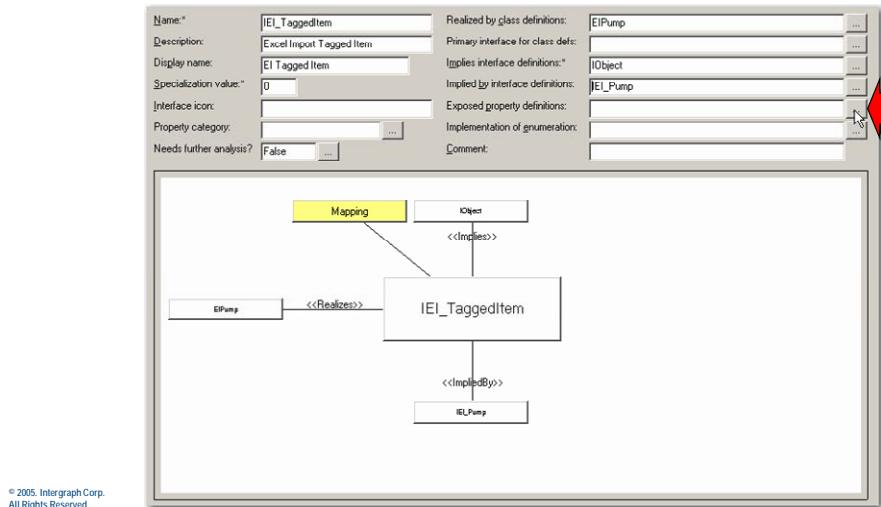
- Click the browse button beside the ***Implied by interface definitions*** field, and choose the ***IEI_Pump*** interface. Click ***OK***.



11. Next, click the browse button beside the ***Exposed property definitions*** field.

Create a Property Definition

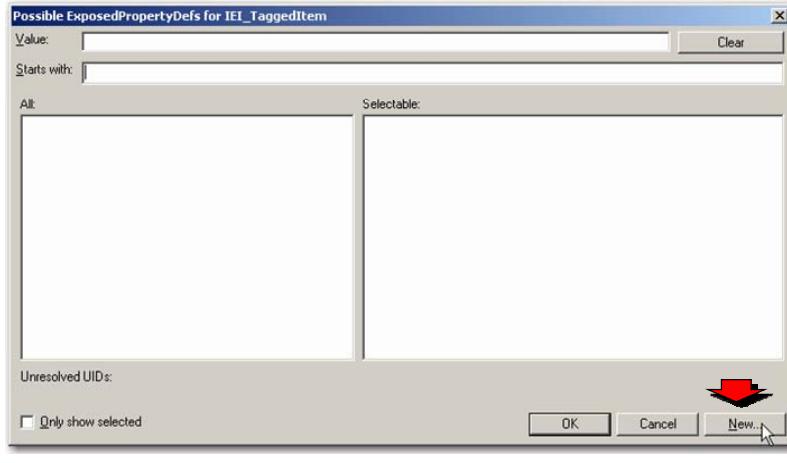
- Click the ***Exposed property definitions*** browse button.



12. The list of available properties is empty. Click the **New** button to create a new property.
-

Create a Property Definition

- ❑ Click the **New** button on the **Possible ExposedPropertyDefs for IEI_TaggedItem** dialog box.



© 2005, Intergraph Corp.
All Rights Reserved.

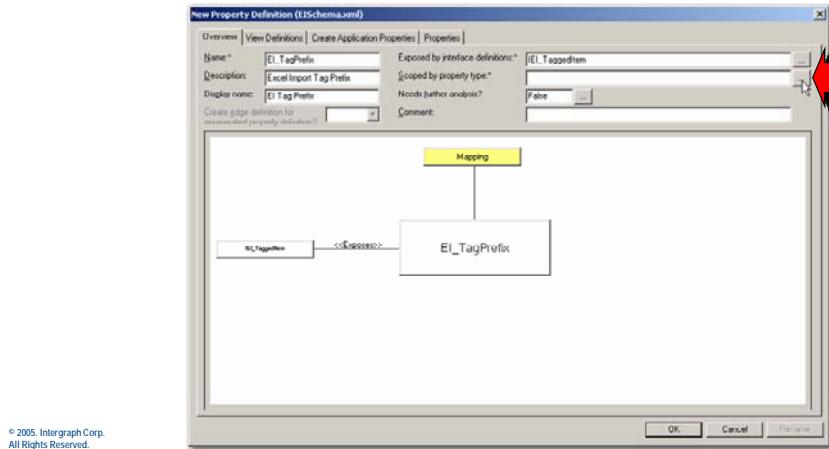
13. Create a new property with the following information:

- Name** – EI_TagPrefix
- Description** – Excel Import Tag Prefix
- Display name** – EI Tag Prefix
- Exposed by interface definition** – IEI_TaggedItem (This property will be populated automatically.)

14. Click the browse button beside the **Scoped by property type** field.

Create a Property Definition

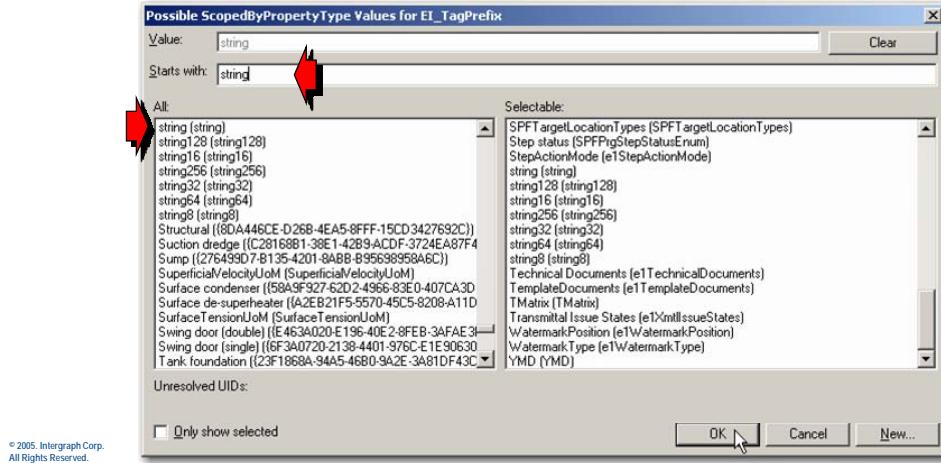
- Provide a name, description, and display name for the property, and then click the **Scoped by property type** browse button.



15. Start typing “string” in the Starts with field until you can see the string value in the list. Choose the **string (string)** option.

Create a Property Definition

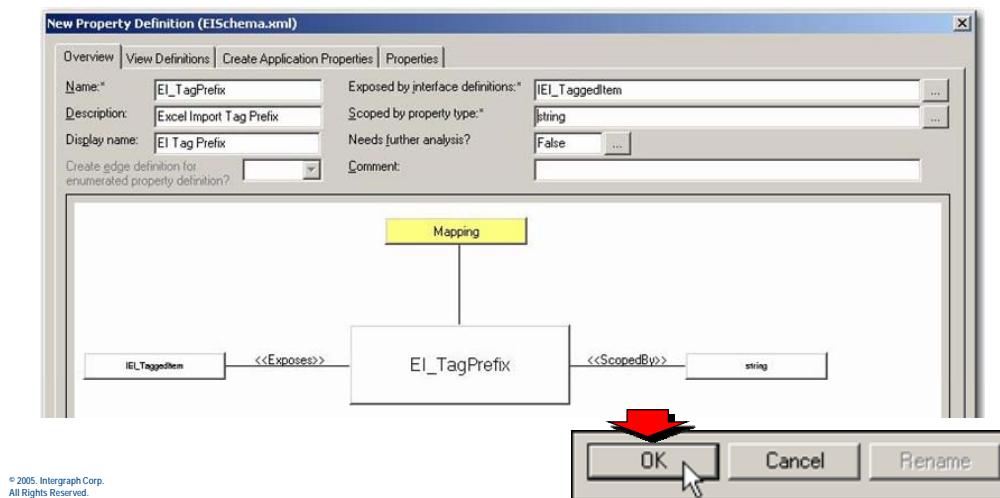
- Enter the first several characters of the type, then select the type of data for the new property.



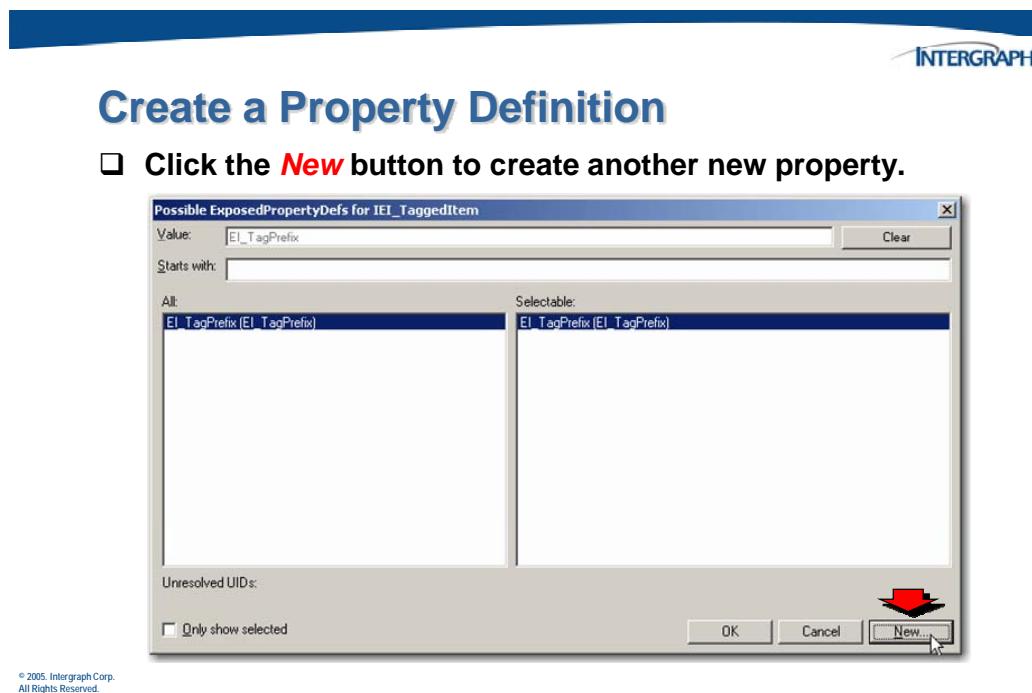
16. Once you have defined the new property, click **OK** to create it.

Create a Property Definition

- Click **OK** to add the new **EI_TagPrefix** property.



17. The new property appears in the list of properties. Click **New** to create another new property.
-



© 2005, Intergraph Corp.
All Rights Reserved.

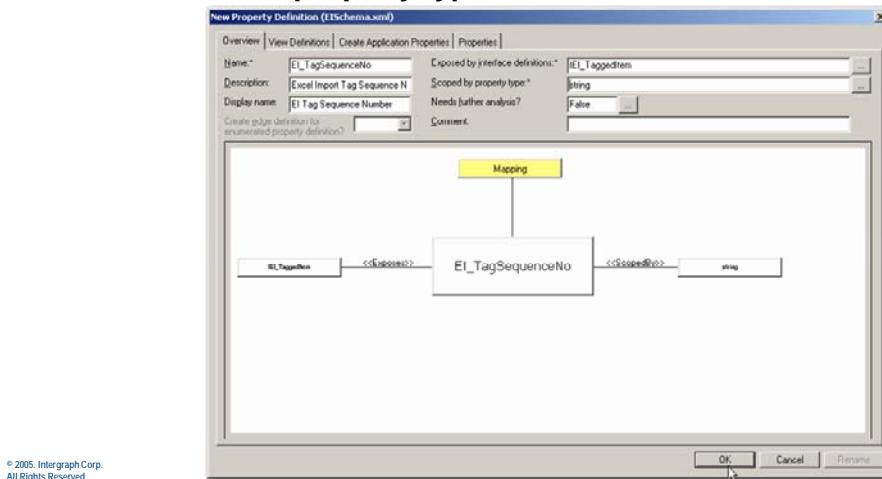
18. Create a new property with the following information:

- Name** – EI_TagSequenceNo
- Description** – Excel Import Tag Sequence Number
- Display name** – EI Tag Sequence Number
- Exposed by interface definition** – IEI_TaggedItem (This property will be populated automatically.)
- Scoped by property type** – string

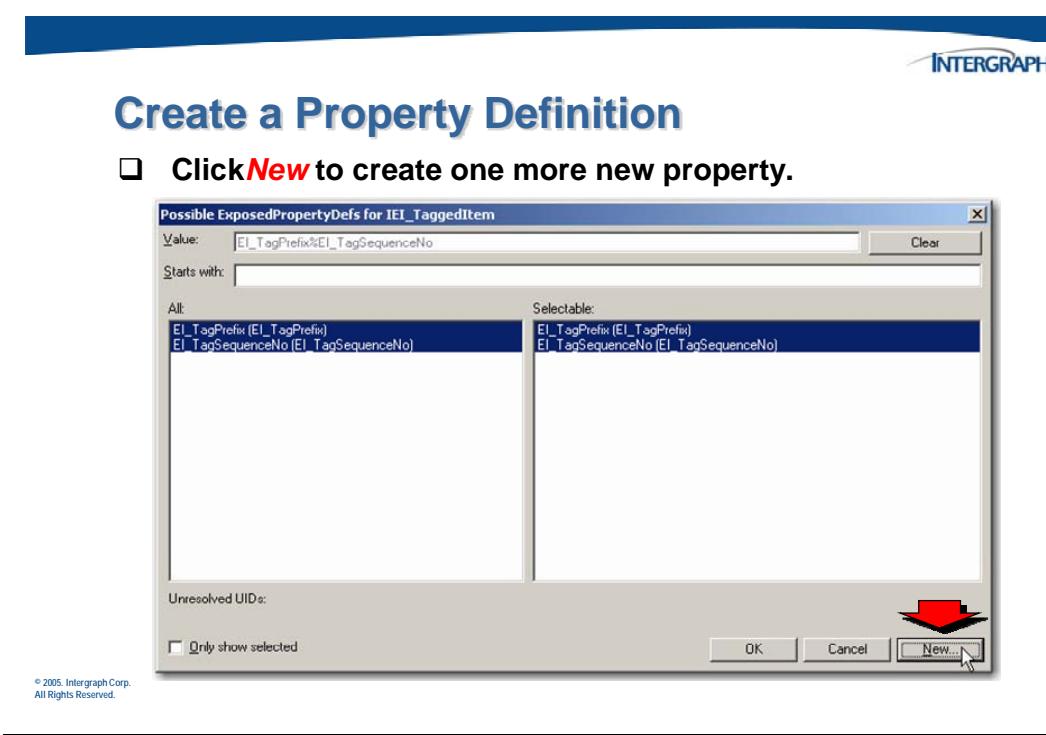
19. Click **OK** to create the new property.

Create a Property Definition

- For the new property, provide a name, description, display name, and property type. Click **OK**.



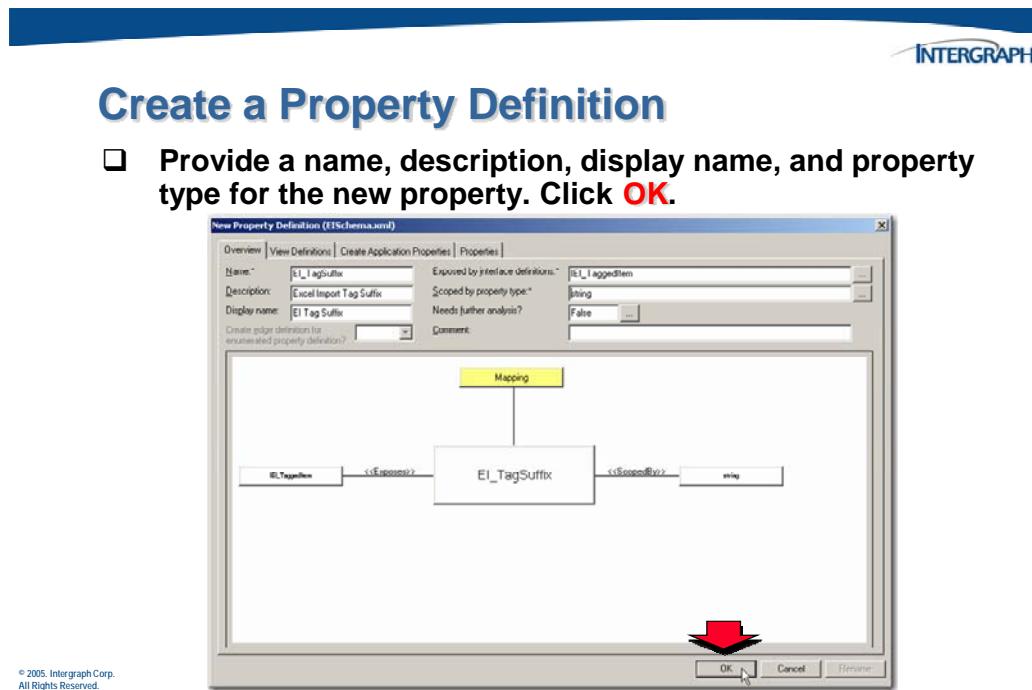
20. The new property is added to the list of available properties. Click **New** to create another new property.



21. Create a new property with the following information:

- Name** – EI_TagSuffix
- Description** – Excel Import Tag Suffix
- Display name** – EI Tag Suffix
- Exposed by interface definition** – IEI_TaggedItem (This property will be populated automatically.)
- Scoped by property type** – string

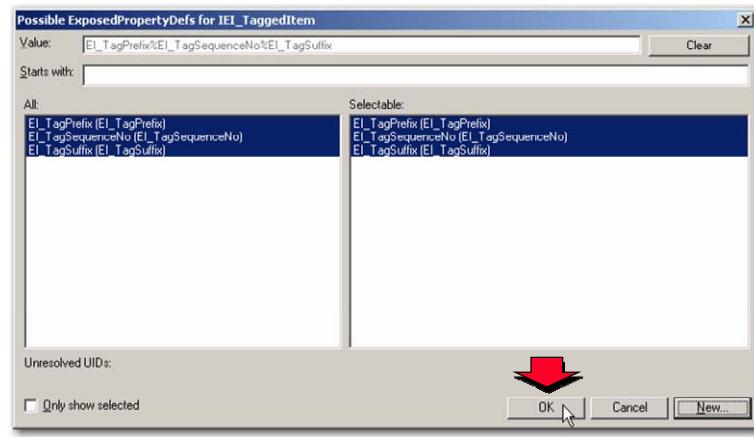
22. Click **OK** to create the new property.



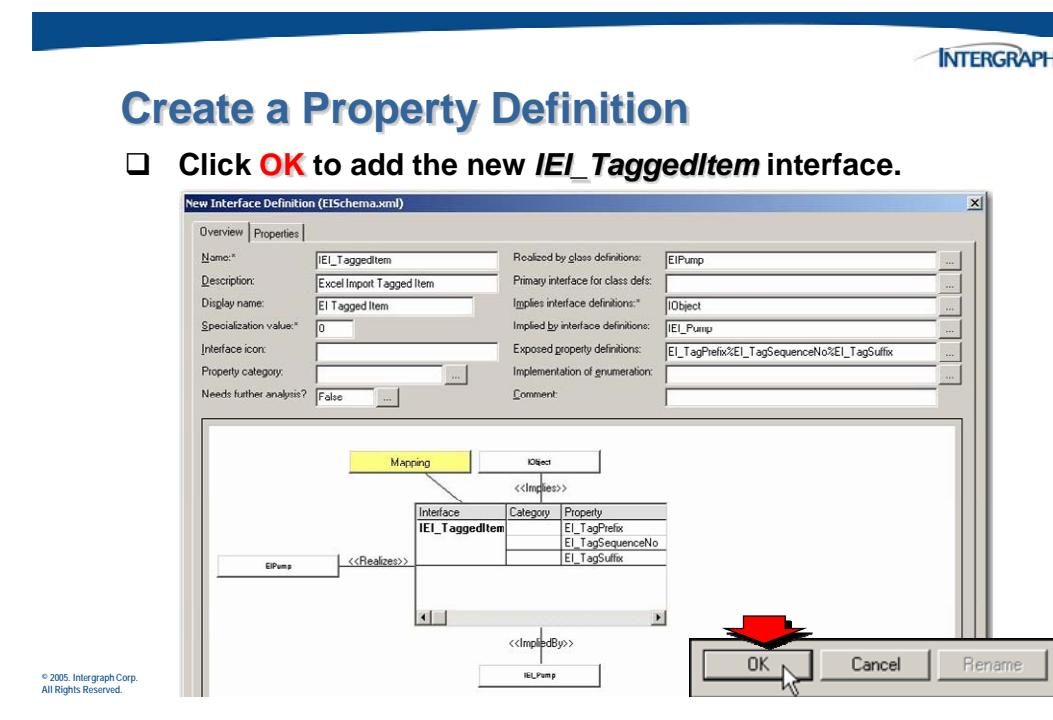
23. When all three of the new properties have been created, make sure they are all selected in the list, and click **OK**.

Create a Property Definition

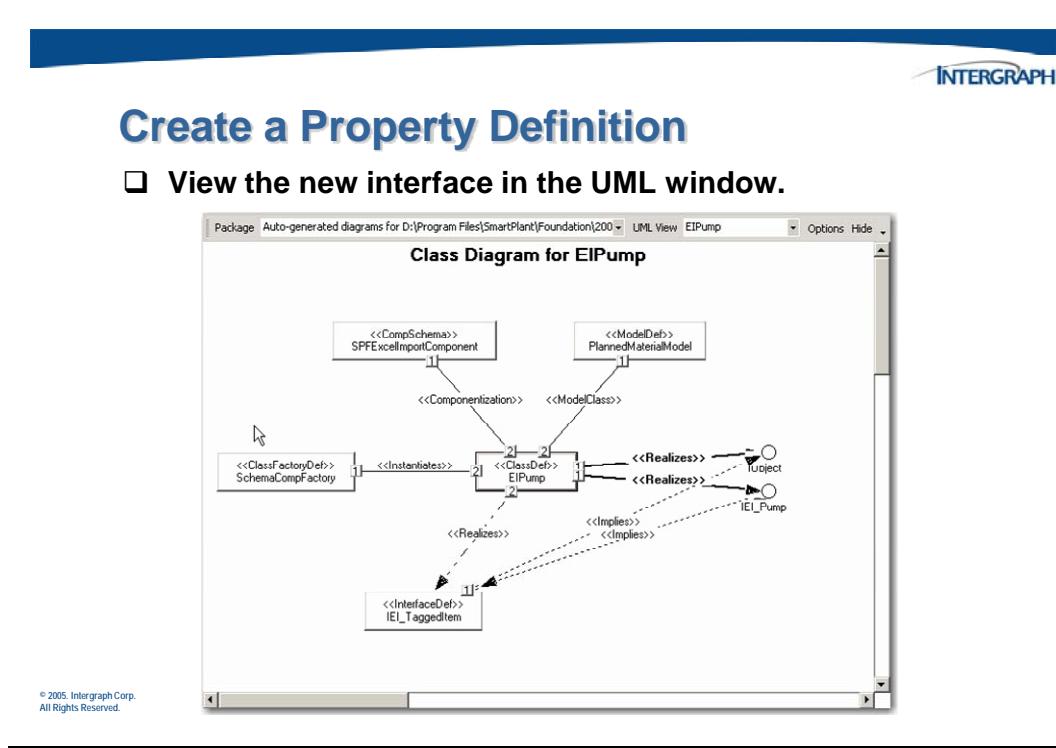
- Make sure all the new properties are selected, and then click **OK** to associate them all with the *IEI_TaggedItem* interface.



24. You will see the three new properties on the form where you provided information about the new interface. Click **OK** to finish creating the **IEI_TaggedItem** interface definition.



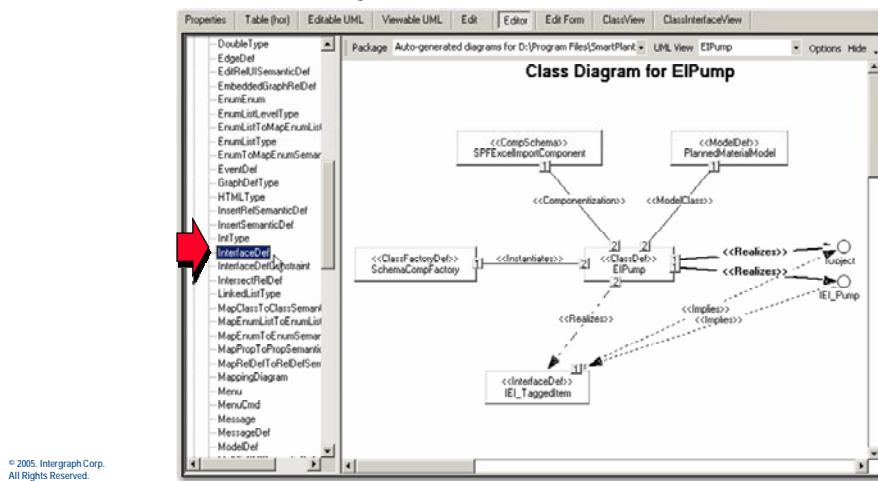
25. The new interface and its relationships with the other items already displayed, is added to the UML display.



26. Create another interface to expose more custom properties. Drag the **InterfaceDef** option from the **Create** section of the **Editor** view into the UML view.

Create a Property Definition

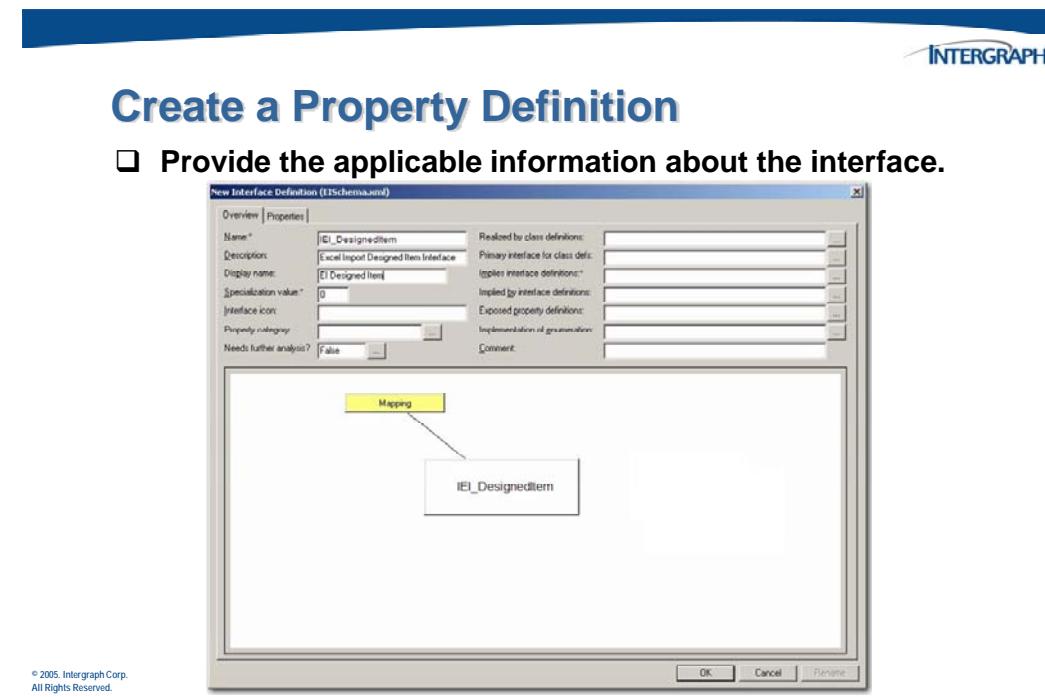
- Create another interface by dragging and dropping the **InterfaceDef** object from the **Create** section to the UML view.



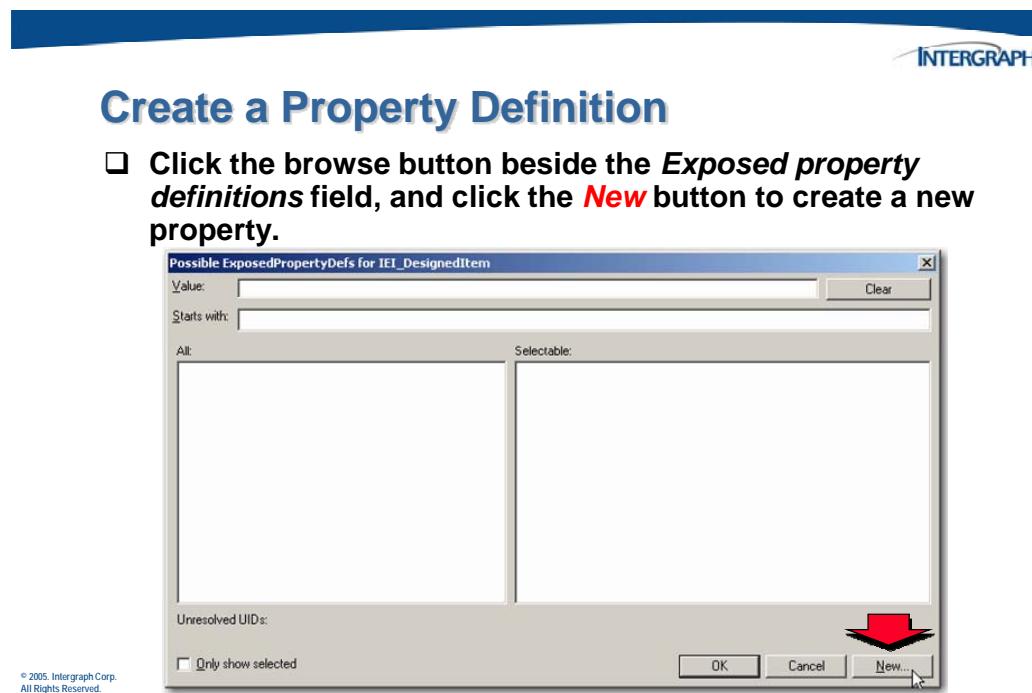
27. Create a new interface with the following information:

- Name** – IEI_DesignedItem
- Description** – Excel Import Designed Item Interface
- Display name** – EI Designed Item
- Realized by class definitions** – EIPump
- Implies interface definitions** – IObject
- Implied by interface definitions** – IEI_Pump

28. Click the browse button beside the **Exposed property definitions** field.



29. No properties are available for selection. Click the *New* button to create a new property.
-



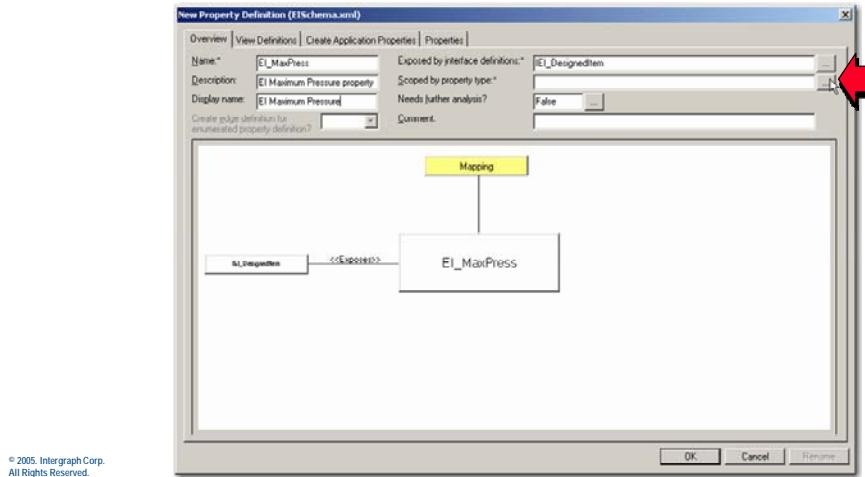
30. Create a new property with the following information:

- Name** – EI_MaxPress
- Description** – EI Maximum Pressure Property
- Display name** – EI Maximum Pressure
- Exposed by interface definition** – IEI_DesignedItem (This property will be populated automatically.)

31. Click the browse button beside the **Scoped by property type** field.

Create a Property Definition

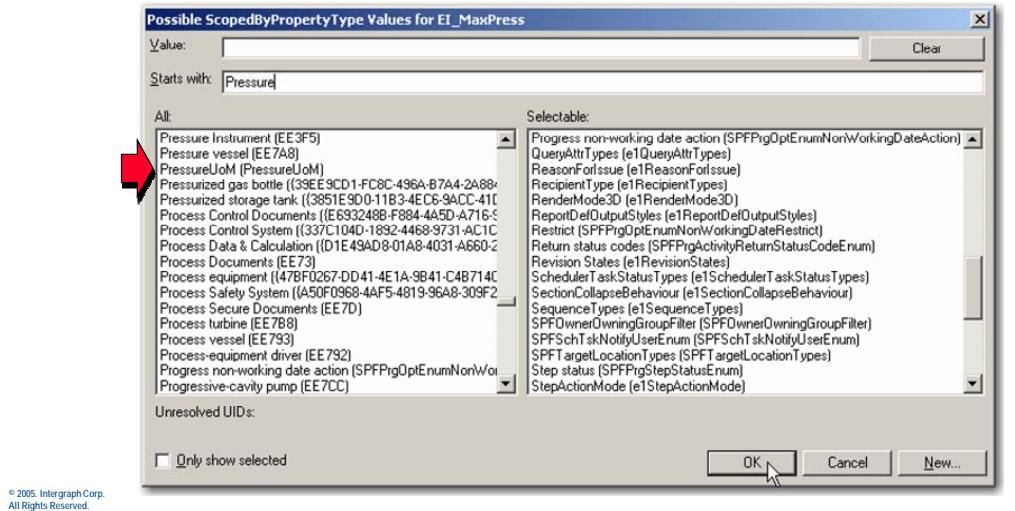
- Provide a name, description, and display name for the new property. Then choose a property type.



32. From the list find the **PressureUoM** property type, and click on it.

Create a Property Definition

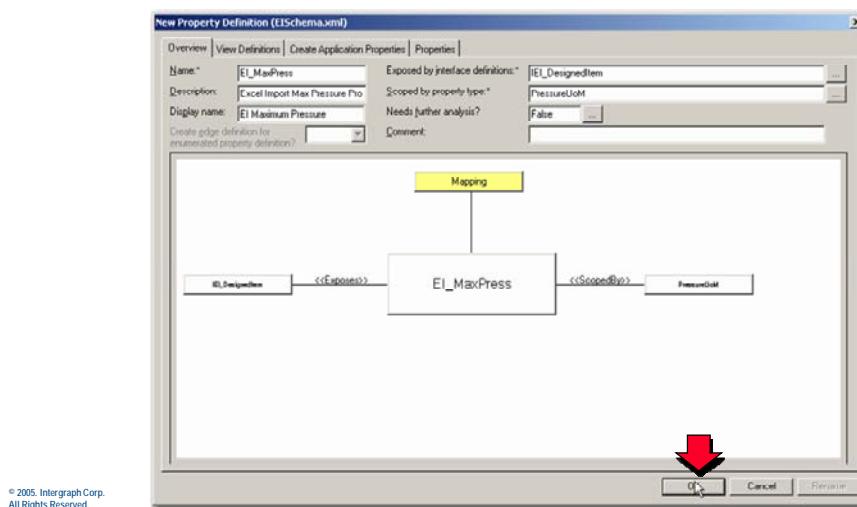
- Choose the **PressureUoM** property type for the new property.



33. Click **OK** to create the new property.

Create a Property Definition

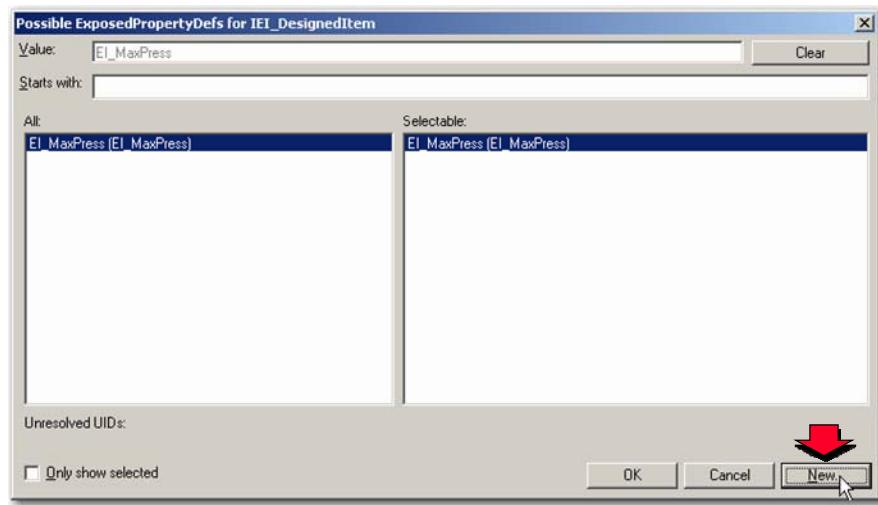
- Once you have finished defining the property, click **OK**.



34. Click the **New** button, again, to create another new property.

Create a Property Definition

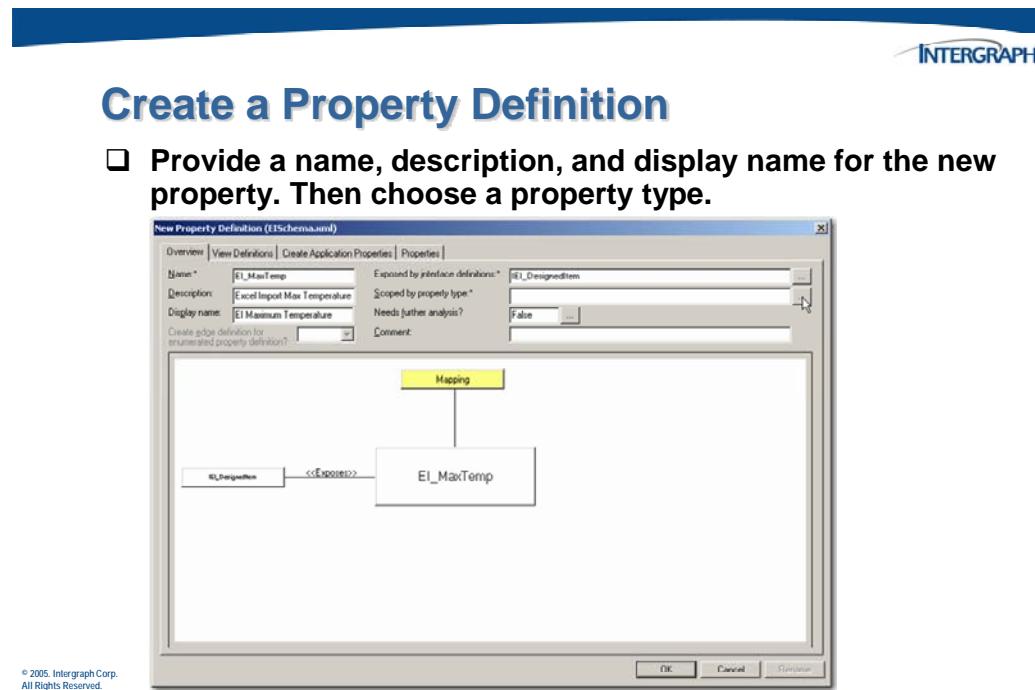
- Click the **New** button to create another property.



35. Create a new property with the following information:

- Name** – EI_MaxTemp
- Description** – Excel Import Max Temperature
- Display name** – EI Maximum Temperature
- Exposed by interface definition** – IEI_DesignedItem (This property will be populated automatically.)

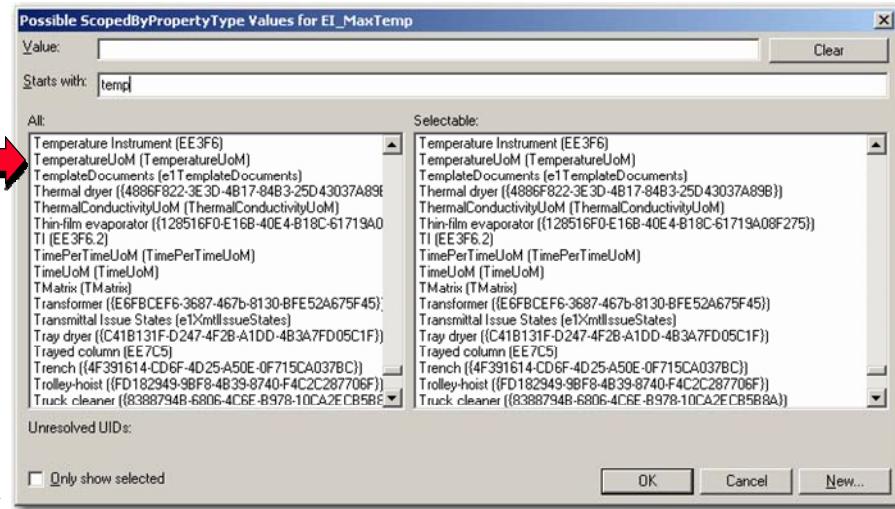
36. Click the browse button beside the **Scoped by property type** field.



37. From the list of property types, find and click on the **TemperatureUoM** option.

Create a Property Definition

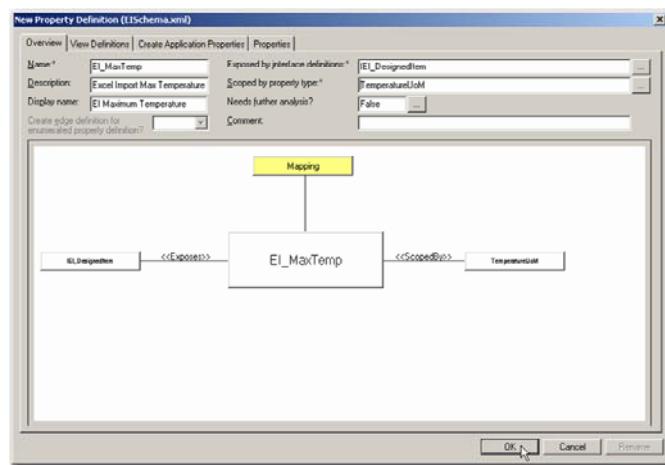
- Choose the **TemperatureUoM** property type.



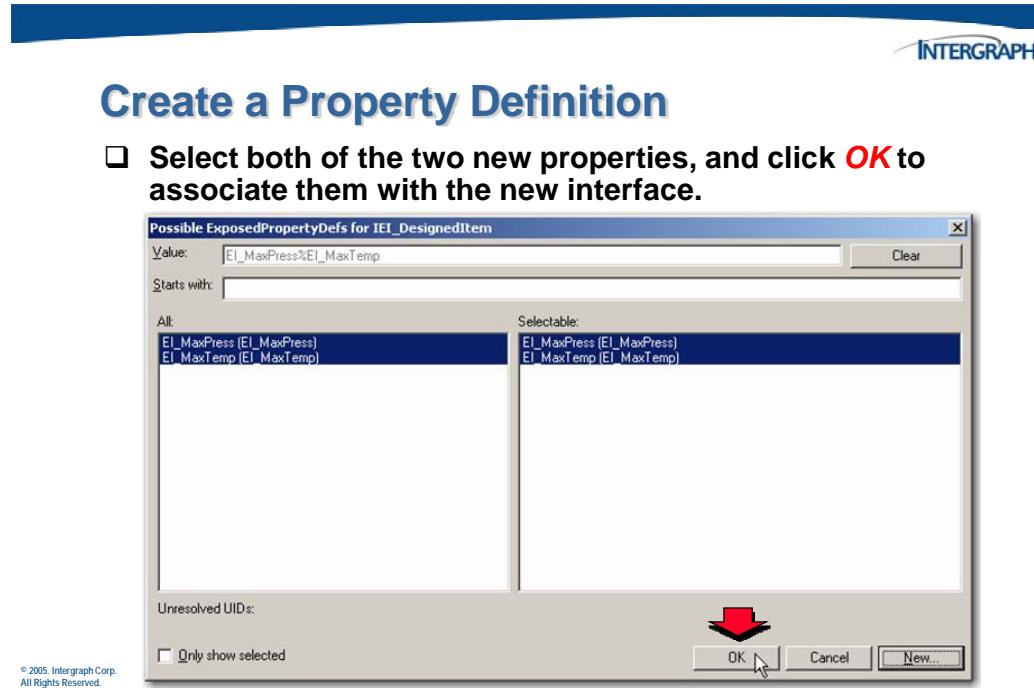
38. Click **OK** to create the new property.

Create a Property Definition

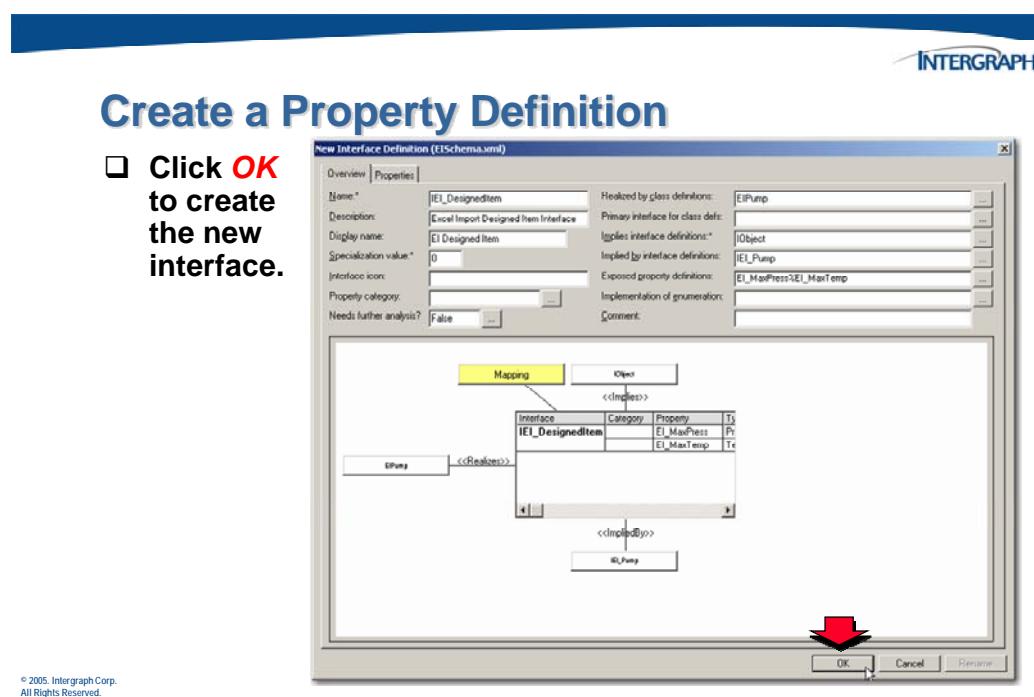
- Once you have finished defining the **EI_MaxTemp** property, click **OK**.



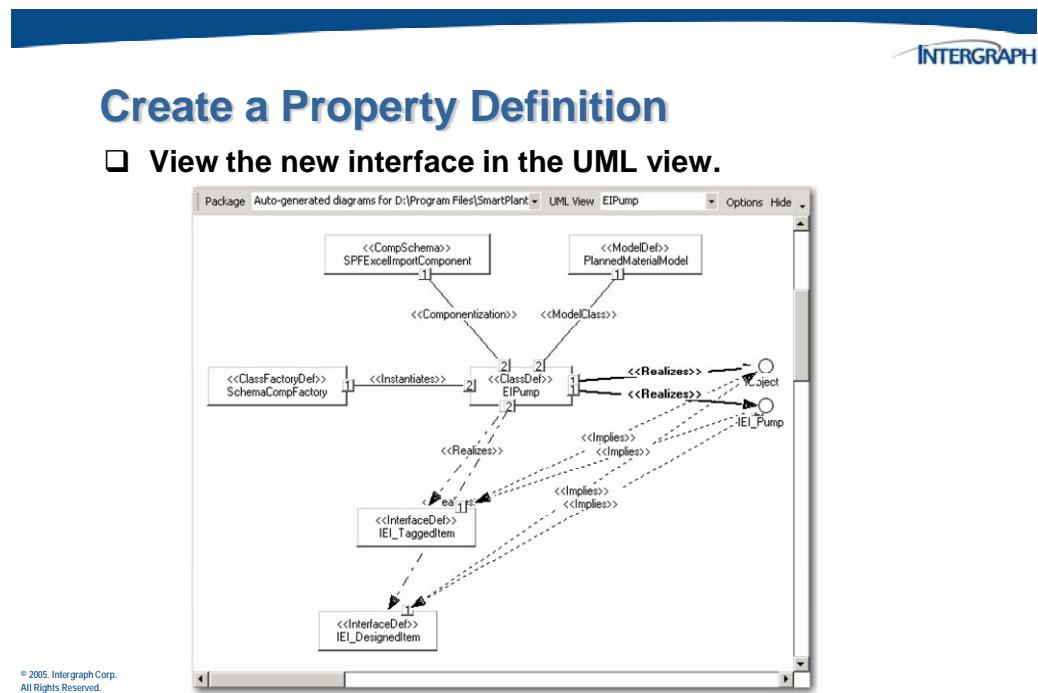
39. Make sure both new properties are selected, and then click **OK**.



40. Click **OK** to create the **IEI_DesignedItem** interface definition.



41. The new interface will appear in the UML view.



4.4 Enumerated List Type

An *enumerated list* (EnumListType) is one of the property types that can scope property definitions. Enumerated lists are sometimes called picklists. Each enumerated list contains one or more enumerated entries (EnumEnum). Each enumerated entry represents a possible value for a property scoped by that enumerated list.

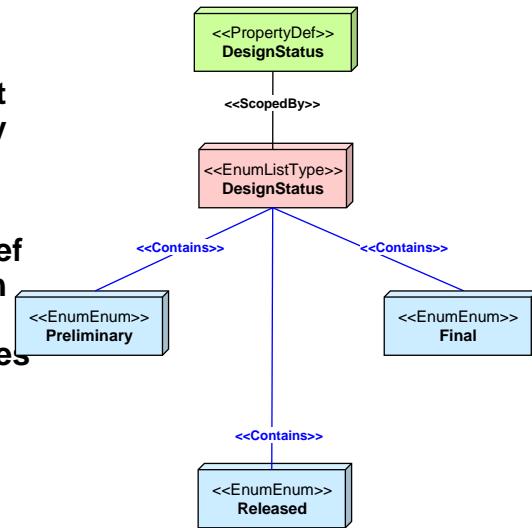


Property Types - Enumerated List Type

Property definitions of the enumerated list type

(EnumListType) have a list
of possible string property
values defined for them in
an enumerated list.

Any value for a PropertyDef
of this type must match an
entry in the list of
enumerated property values
defined for the property
type.



© 2005, Intergraph Corp.
All Rights Reserved.

Enumerated lists may be combined to form an enumeration hierarchy. When this happens, a special property type (enumerated list level type) is used to describe all of the enumerations at a given level in the enumeration hierarchy.



Property Types - Enumerated List Type

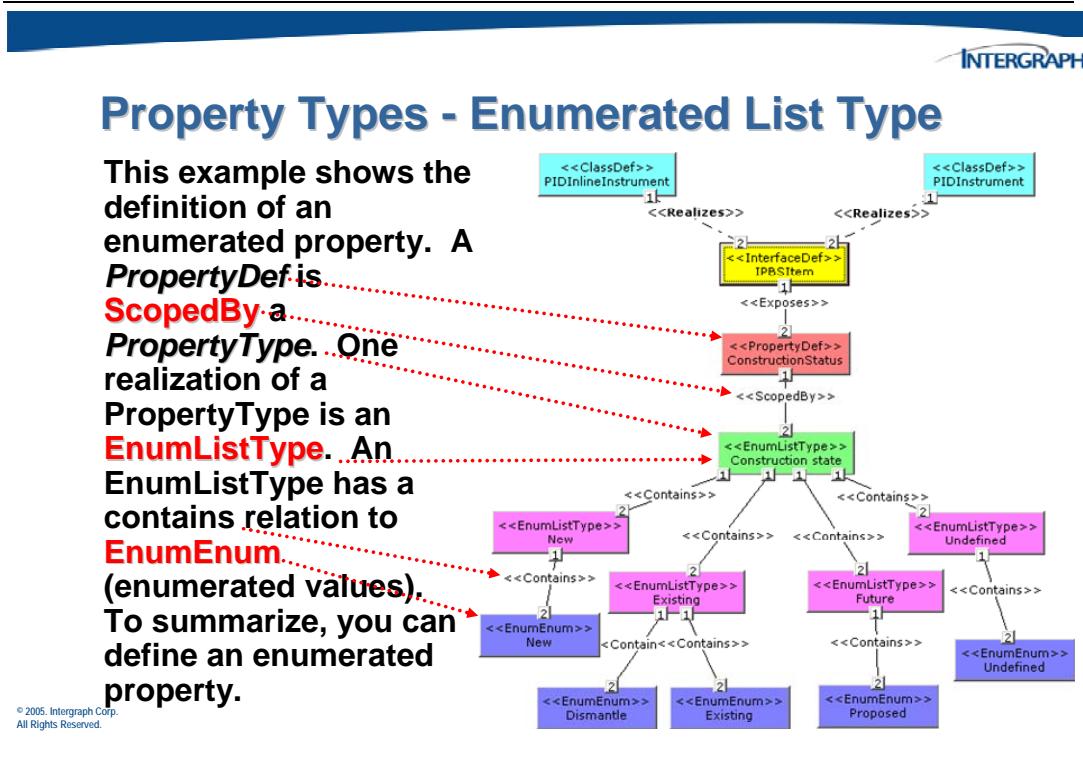
Enumerated lists can be combined to form an enumeration hierarchy.

All nodes in the hierarchy that are `EnumListTypes` are enumerated entries for the `EnumListType` above them.

All `EnumListType` nodes must contain `EnumEnums` (list entries) or other `EnumListTypes` (enumerated lists).

For example, the typing of equipment consists of a multi-level hierarchy, where each level down the hierarchy provides a more specific definition of the equipment. Every branch node in a hierarchy is itself an enumerated list, as well as an enumerated value. Only the topmost node in the hierarchy is not an enumerated value (just an enumerated list) and only the leaf nodes in the hierarchy are not enumerated lists (just enumerated values).

The following UML diagram presents the SmartPlant schema model for enumerated lists and enumerated entries.



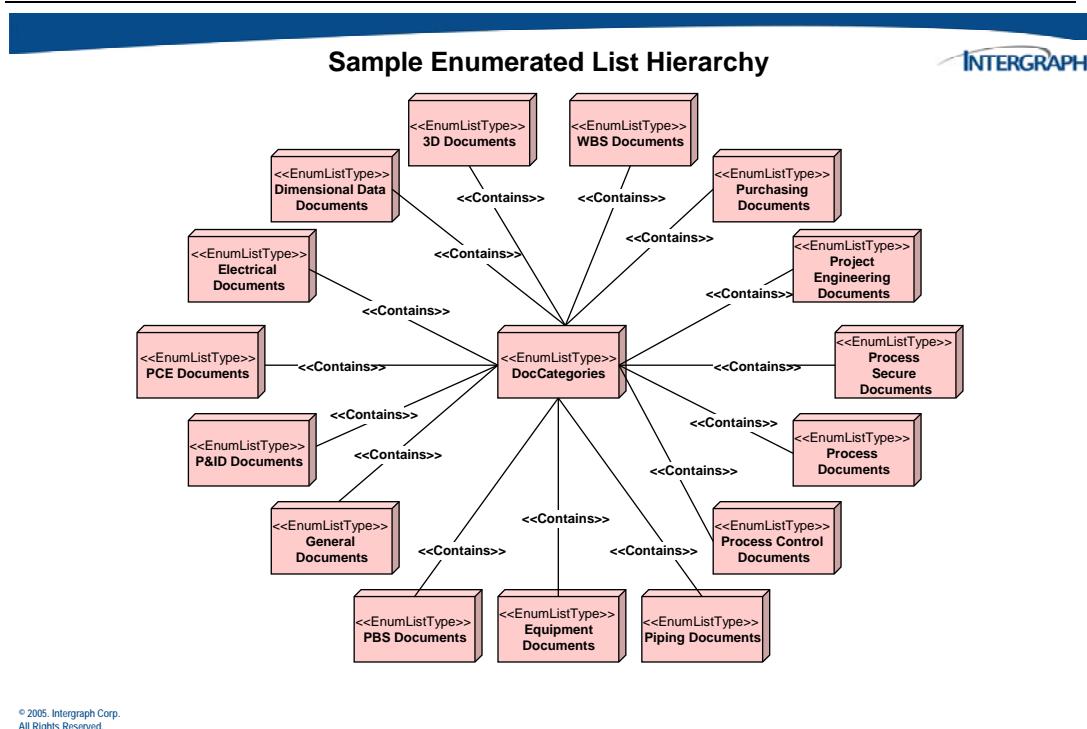
The **EnumListType** class definition is used for enumerated list property types. **EnumListType** realizes **I.PropertyType**. **EnumListType** also realizes **IEnumListType**, the interface definition that supports enumerated list behavior.

IEnumListType has a **Contains** relationship definition to **IEnumEnum**. Instances of this relationship are used to indicate the enumerated entries contained within the enumerated list.

Leaf nodes within an enumerated list hierarchy are instances of the **EnumEnum** class definition. This class definition realizes **IEnumEnum**, the interface definition that supports enumerated entry behavior, and **IOObject**, the interface that contains the short (**Name**) and long (**Description**) text values for the enumerated entry.

Branch nodes within an enumerated list hierarchy are instances of the **EnumListType** class. **EnumListType** has an optional realizes relationship to **IEnumEnum**, which means that an enumerated list may or may not be an enumerated entry as well. For branch nodes within an enumerated list hierarchy, the **EnumListType** object will realize this optional interface.

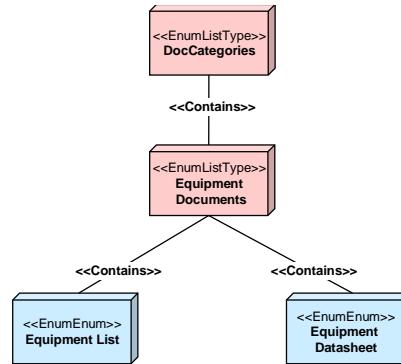
The topmost node in an enumerated list hierarchy is an instance of the `EnumListType` class definition. This instance does not realize `IEnumEnum`, since it is not an enumerated entry.



© 2005, Intergraph Corp.
All Rights Reserved.

Each `EnumListType` in the `DocCategories` enumerated list either contains enumerated list entries (`IEnumEnum`) or other nested lists (`EnumListType`). An example of each of these cases appears below.

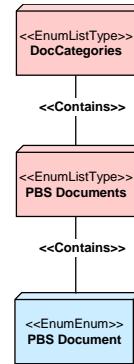
Sample Enumerated List Hierarchy



© 2005. Intergraph Corp.
All Rights Reserved.

The Equipment Documents EnumListType contains two enumerated list values:
Equipment List and Equipment Datasheet.

Sample Enumerated List Hierarchy



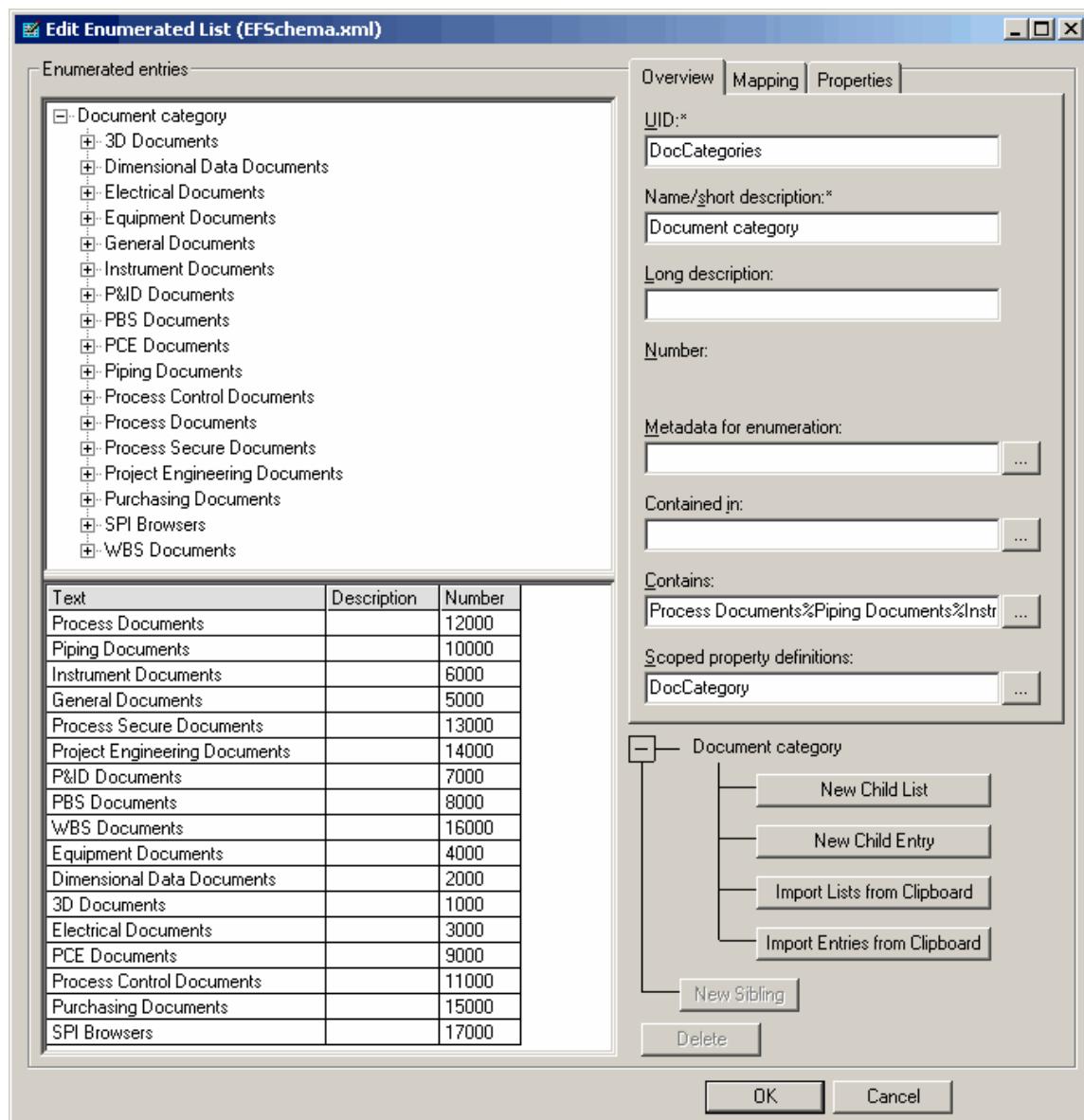
© 2005. Intergraph Corp.
All Rights Reserved.

The PBS Documents EnumListType contains an EnumEnum called PBS Document.

4.4.1 Properties of an Enumerated List Type

The following options are available when you create or edit an enumerated list type:

- Enumerated entries** – Displays the enumeration hierarchy as you add child lists and entries to the main enumerated list.
- UID** – Displays the unique identifier for the enumerated list. The UID is automatically assigned by the Schema Editor.
- Short description** – Specifies a short description for the enumerated list. The short description is used as the name for the enumerated list throughout the Schema Editor.
- Long description** – Specifies a long description for the enumerated list.



- Number** – Specifies a number that can be used as a secondary key to identify the enumeration.
- Metadata for enumeration** – Defines a relationship between the enumerated entry and another object or objects in the schema that provide more information about that enumeration. This is frequently used to relate enumerations to interface definitions for type hierarchies.
- Contained in** – Identifies the names of other enumerated lists that contain this enumerated list.
- Contains** – Identifies the names of other enumerated lists and enumerated list entries that this list contains.
- Scoped property definitions** – Identifies the property definitions that have a ScopedBy relationship with this enumerated list type.
- New Child List** – Creates a new child list (EnumListType) under the selected node in the enumeration hierarchy. When you select an enumerated entry and create a new child list under it, it becomes an EnumListType object.
- New Child Entry** – Creates a new child enumerated list entry (EnumEnum) under the selected enumerated list in the hierarchy.
- Import Lists from Clipboard** – Imports enumerated lists that you have copied from a spreadsheet to the Clipboard into the Schema Editor. When you import lists, the Schema Editor tries to map the contents of the Clipboard to the appropriate enumerated list properties based on the imported values. You can change this mapping manually in the **Import from Clipboard** dialog box.
- Import Entries from Clipboard** - Imports enumerated list entries that you have copied from a spreadsheet to the Clipboard into the Schema Editor. When you import entries, the Schema Editor tries to map the contents of the Clipboard to the appropriate enumerated entry properties based on the imported values. You can change this mapping manually in the **Import from Clipboard** dialog box.
- New Sibling** – Creates a new enumerated list or enumerated list entry at the same level as the list or entry selected in the **Enumerated entries** tree.
- Delete** – Deletes the selected node in the **Enumerated entries** tree.

4.4.2 Unit of Measure List Types

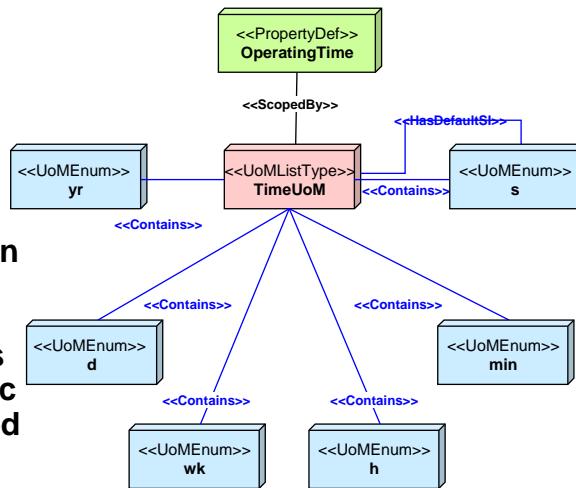
The unit of measure (UoM) list type is specialization of the enumerated list type (IUoMListType implies IEnumListType). A unit of measure list contains unit of measure enumerated values (UoMEnum). A UoMEnum contains two conversion factors: A (multiplier) and B (addend). These conversion factors are used to convert the value to SI units.



Property Types - Unit of Measure List Type

The unit of measure list type (UoMListType) is a special type of enumerated list with additional semantic information.

Each enumerated entry in the UoM list has conversion factors to convert from the SI units for that list to the specific units of measure selected by the user.



© 2005. Intergraph Corp.
All Rights Reserved.

For example, the given unit (x) is multiplied by A and then B is added to that value to determine the SI value ($y = Ax + B$).

NOTE: SI stands for International System of Units and is the modern metric system of measurement. Long the language universally used in science, the SI has become the dominant language of international commerce and trade. SI is the basis used for the conversion of UoM's.

A UoMListType is a composition of enumerated unit of measure entries. One entry in every unit of measure list has a relationship of HasDefaultSI with the UoMListType, which means that this entry is the default SI unit of measure for the list. For example, in the TimeUoM list, "s" (seconds) is the default SI unit of measure for time.

A unit of measure list may have an associated enumerated list that identifies the possible conditions for that unit of measure list. For example, the pressure unit of measure list has

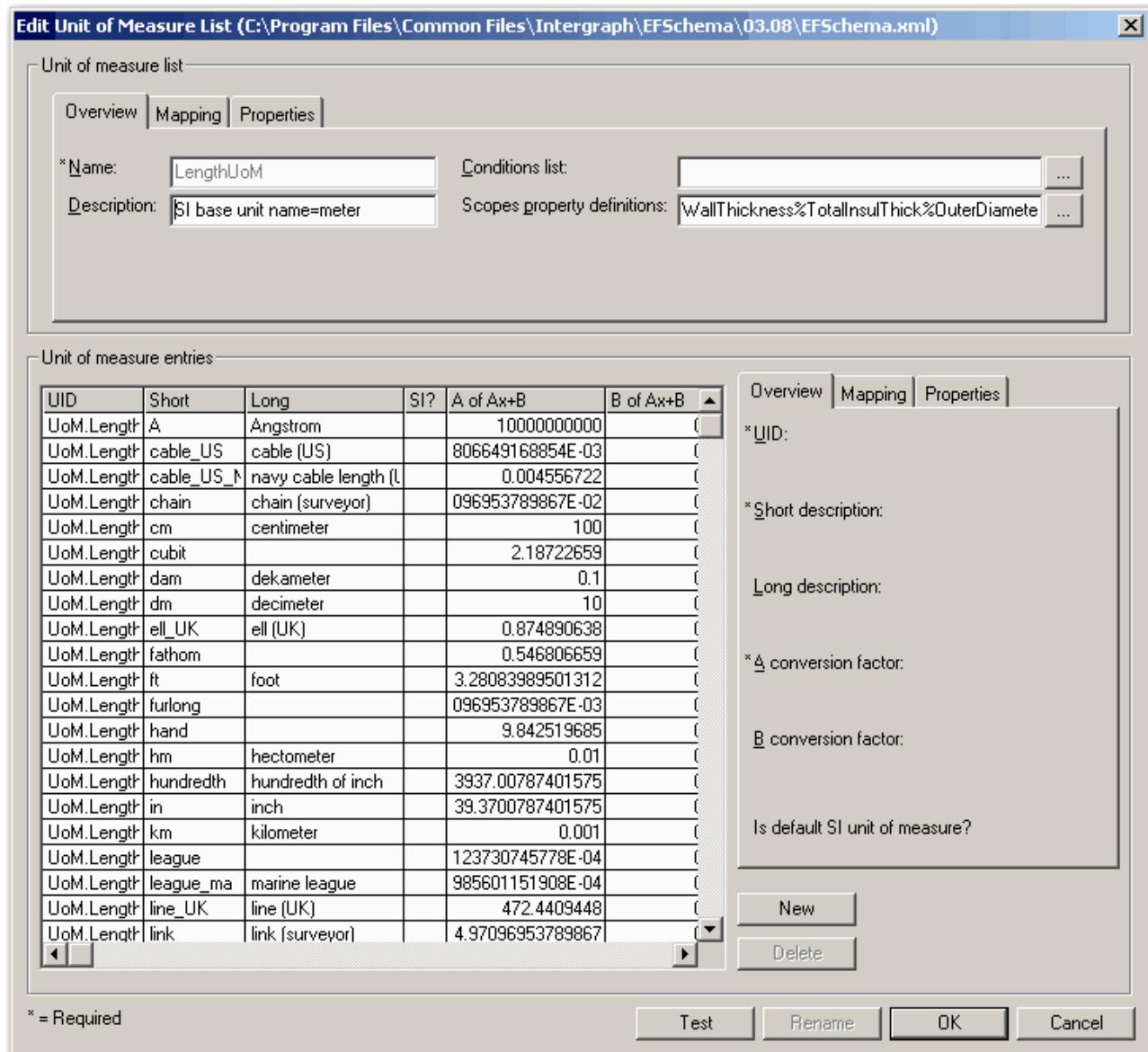
an associated enumerated list that identifies whether the pressure value is absolute or gauge (@ Gauge).

A PropertyDef that is typed as unit of measure has a double precision floating point value in the SI unit for that list. It can also have a preferred display unit of measure, a number of significant digits, and uncertainty.

4.4.3 Properties of a Unit of Measure List Type

The following options are available when you create a new enumerated list type:

- Name** – Specifies the name of the unit of measure list.
- Description** – Specifies a description for the unit of measure list
- Conditions list** – Identifies the enumerated list that contains the possible conditions for a property scoped by this unit of measure list. For example, the pressure unit of measure list uses a conditions list that identifies whether the pressure value is absolute or gauge.

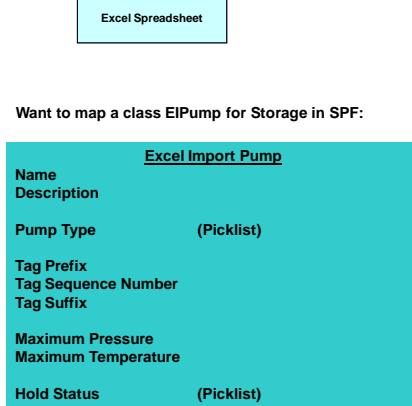


- Scopes property definitions** – Lists the property definitions that are scoped by this unit of measure list.

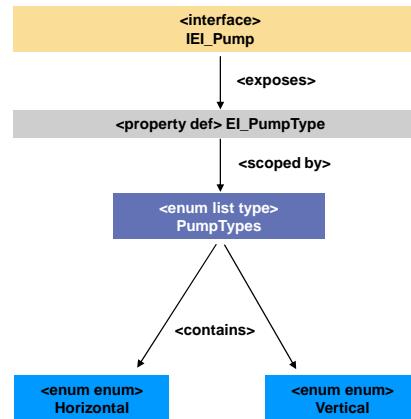
- Unit of measure entries** – Displays the unit of measure entries for this list in tabular format.
- UID** – Displays the unique identifier for the selected entry in the list. The UID is automatically assigned by the Schema Editor, but you can change it.
- Short description** – Specifies a short description for the unit of measure list entry. The short description is used as the name for the entry throughout the Schema Editor.
- Long description** – Specifies a long description for the unit of measure list entry.
- A conversion factor** – Specifies the multiplier used to convert from this unit of measure to SI units. Conversions are done using $y = Ax + B$, where x is the value in this unit of measure and y is the value in SI units.
- B conversion factor** – Specifies the addend used to convert from this unit of measure to SI units. Conversions are done using $y = Ax + B$, where x is the value in this unit of measure and y is the value in SI units.
- Is default SI unit of measure?** – Specifies whether this unit of measure is the default SI unit of measure for this list.
- New** – Adds a new unit of measure entry to the list.
- Delete** – Deletes the selected unit of measure entry from the list.
- Test** – Allows you to test conversions among units of measure in this list using the A and B conversion factors defined for each unit of measure and any specified conditions for each unit of measure.

4.5 Interactive Activity – Creating an Enumerated List Property

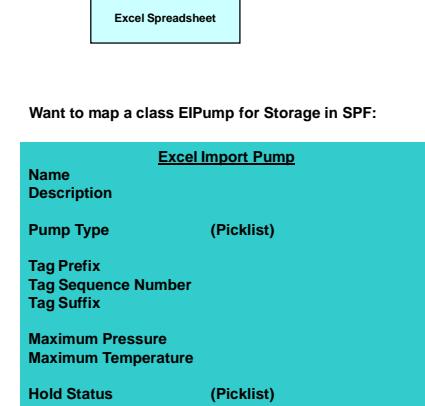
New Object – The Excel Import Pump



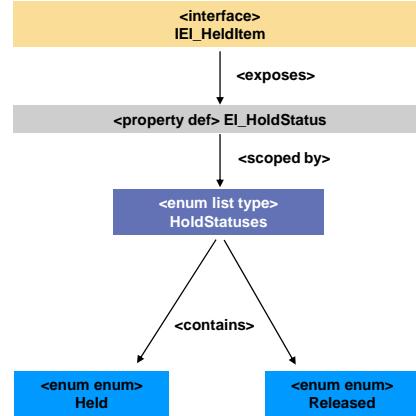
Step 6 – Scope the design status property using a “picklist”, an enum list type and enum enums (entries)



New Object – The Excel Import Pump



Step 6 – Scope the design status property using a “picklist”, an enum list type and enum enums (entries)

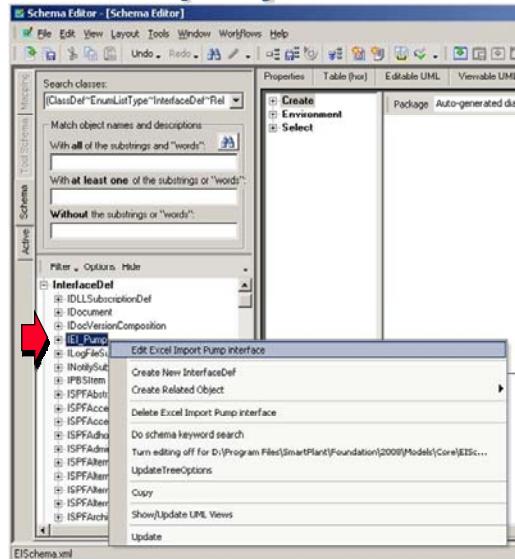


© 2005, Intergraph Corp.
All Rights Reserved.

1. In the tree, in the left-hand pane, find the **IEI_Pump** interface definition.

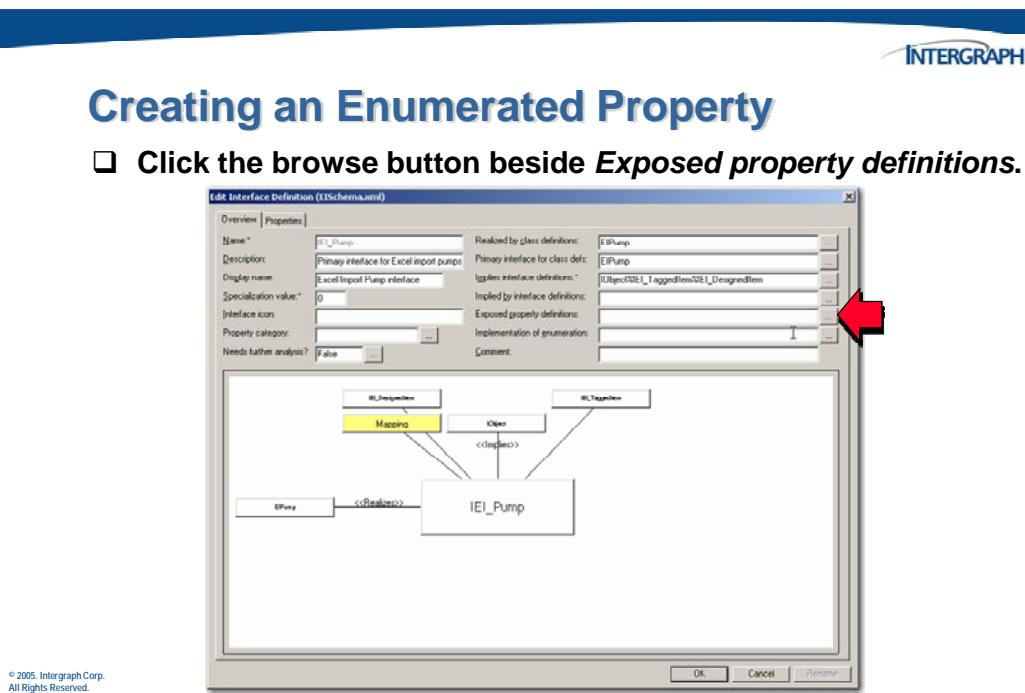
Creating an Enumerated Property

- In the tree on the left, find the **IEI_Pump** interface, right click and choose the **Edit** command.



© 2005, Intergraph Corp.
All Rights Reserved.

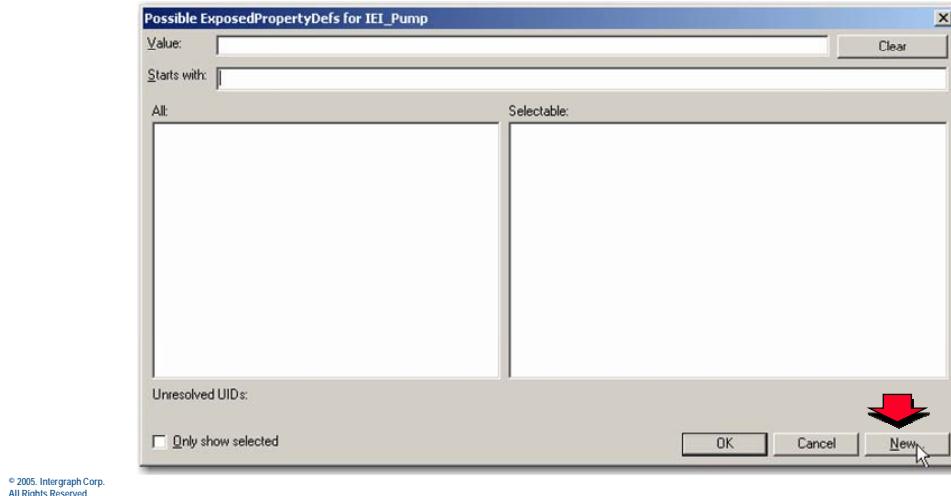
2. Click the browse button beside the *Exposed property definitions* field.



3. Click the *New* button to create a new property definition.

Creating an Enumerated Property

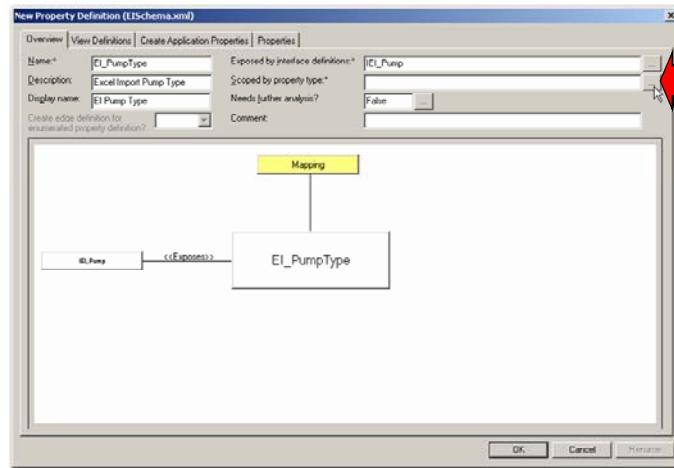
- Click the **New** button to create a new property.



4. Create a new property with the following information:
 - Name** – EI_PumpType
 - Description** – Excel Import Pump Type
 - Display name** – EI Pump Type
 - Exposed by interface definition** – IEI_Pump (This property will be populated automatically.)
5. Click the browse button beside the **Scoped by property type** field.

Creating an Enumerated Property

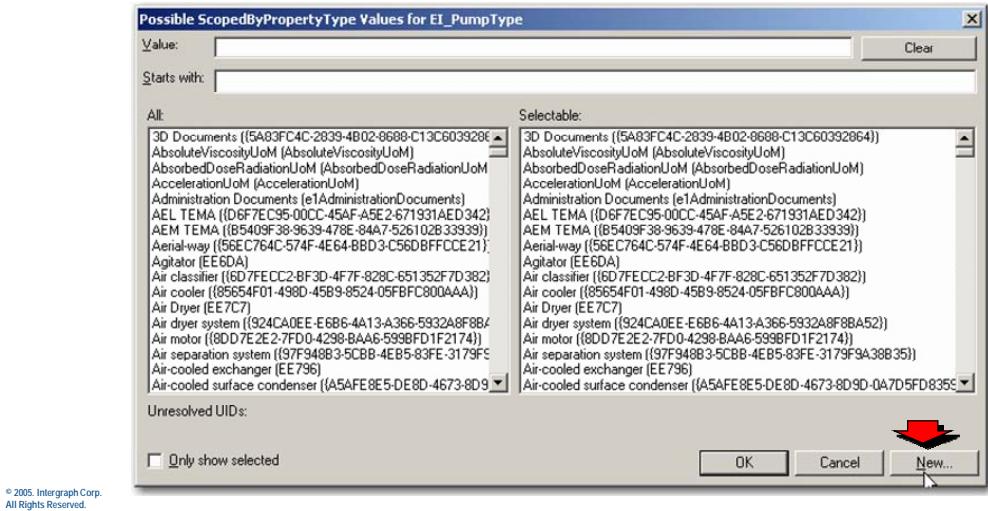
- Provide the information, and then click the **Scoped by property type** button.



6. We need to create a enumerated list type. Click the **New** button.

Creating an Enumerated Property

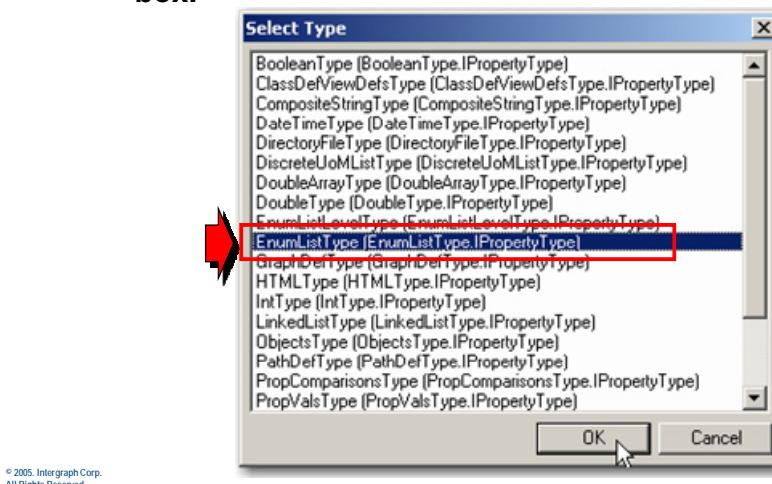
- Click the New button to create a new property type.



7. From list of property types that can be created, find and click on **EnumListType**.

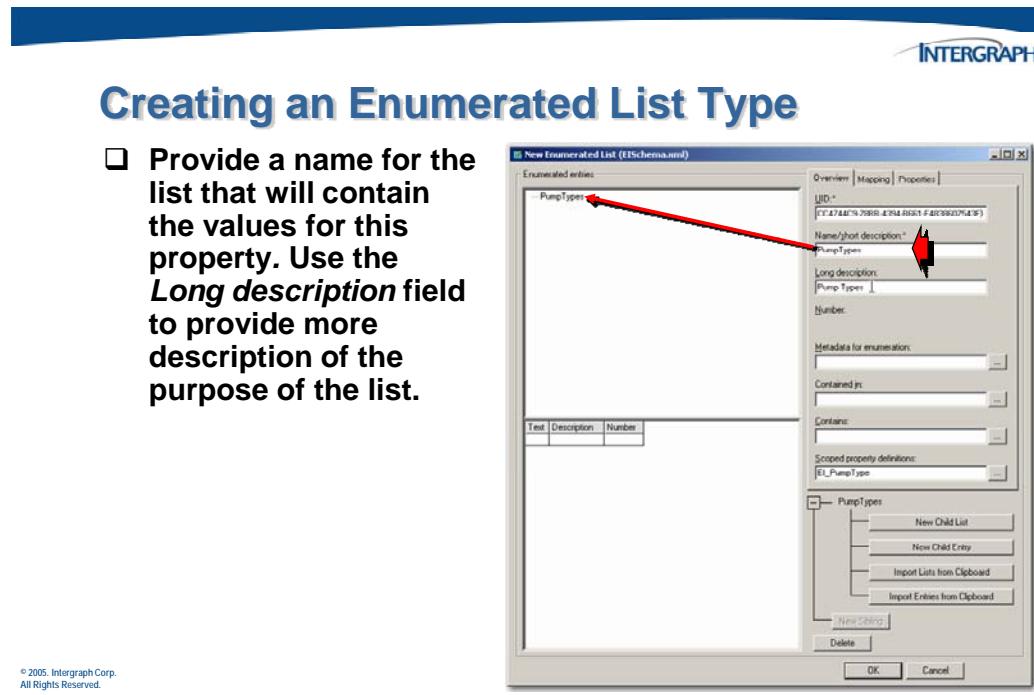
Creating an Enumerated List Type

- Select the **EnumListType** entry from the **Select Type** dialog box.

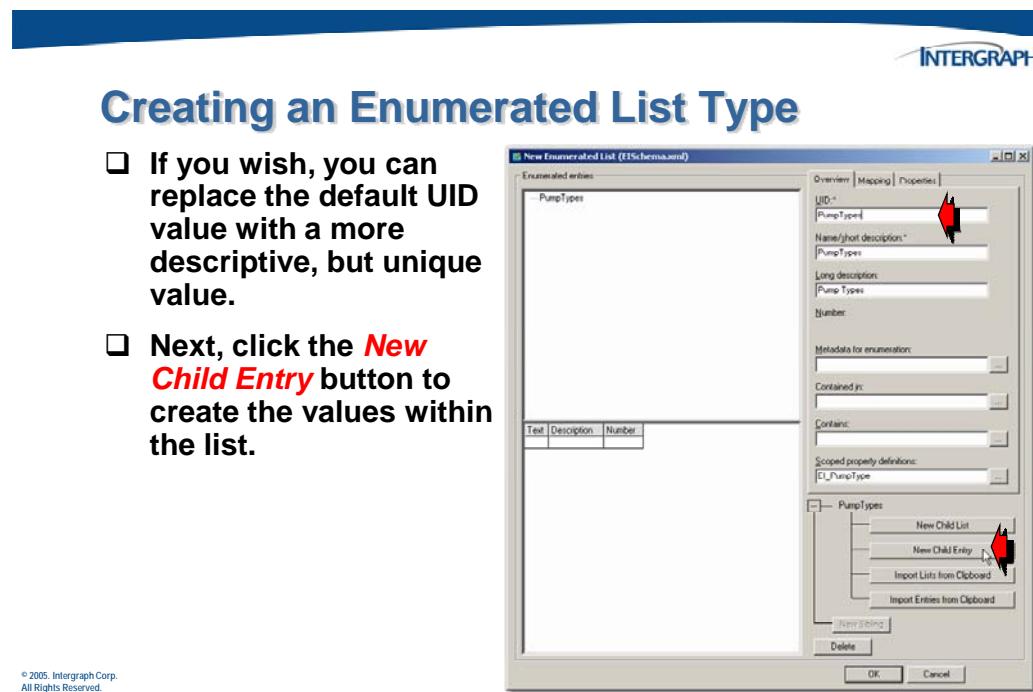


8. Provide a name and description for the new list.

- Name** – PumpTypes
 - Long description** – Pump Types
-

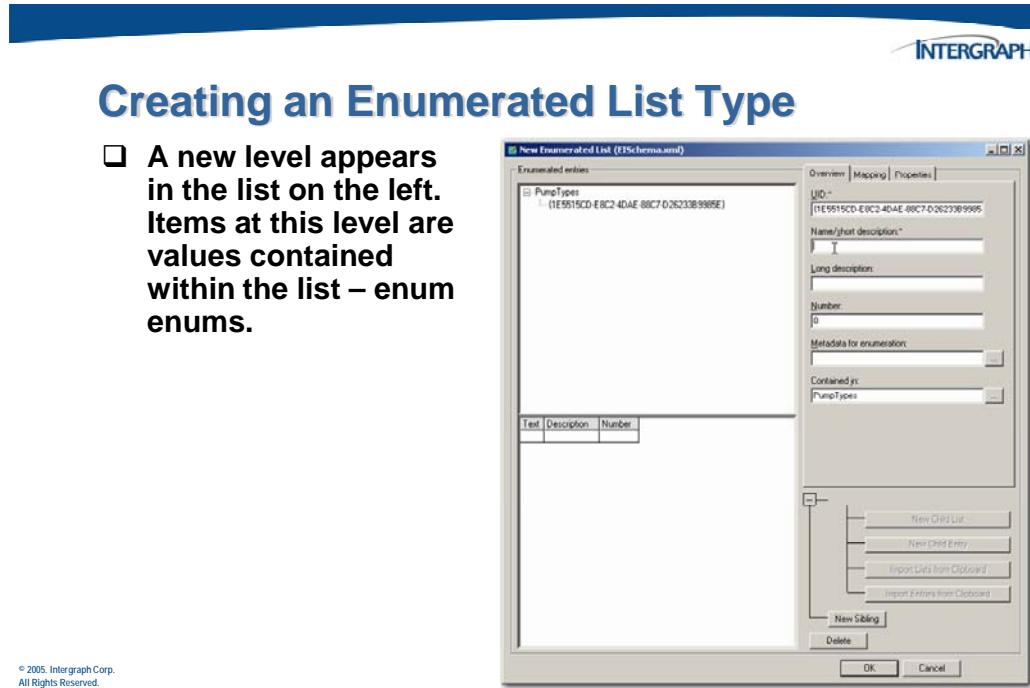


9. If you want, you can also replace the default **UID** value with a unique, but more user-friendly value. This value will be used throughout the software to indicate the list that scopes the property.
 - UID -- PumpTypes**
10. When you have finished providing information about the list itself, you are ready to start creating the values in the list. Click the **New Child Entry** button.



© 2005, Intergraph Corp.
All Rights Reserved.

11. A new object appears in the tree, as a child of the new list.
-



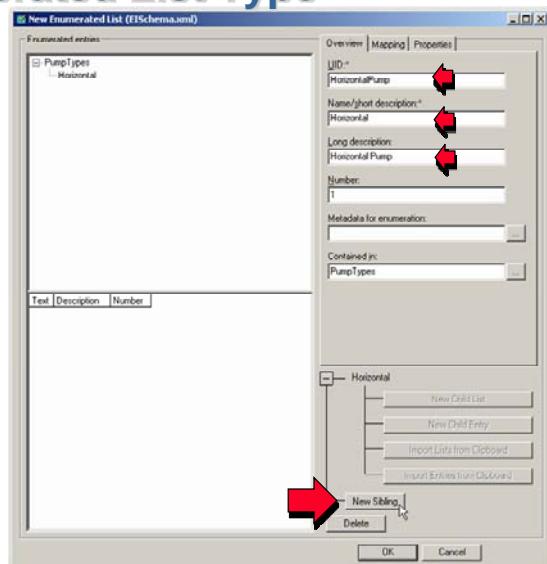
12. Provide a name, description, and unique UID for the list value, and then click **New Sibling** to create another. The number field is optional, but may be used to specify the order of the values in the list box.

- Name – Horizontal
- Long description – Horizontal Pump
- UID -- HorizontalPump
- Number -- 1

NOTE: The number value for list entries is used specifically by SmartPlant 3D when publishing and retrieving the property values. This use will be covered in the mapping section of this course.

Creating an Enumerated List Type

- Provide a name and long description for the first enum enum. – *Horizontal*.**
- If you wish, replace the *UID* with a unique, user-friendly value.**
- Click the *New Sibling* button to create another value in the list.**



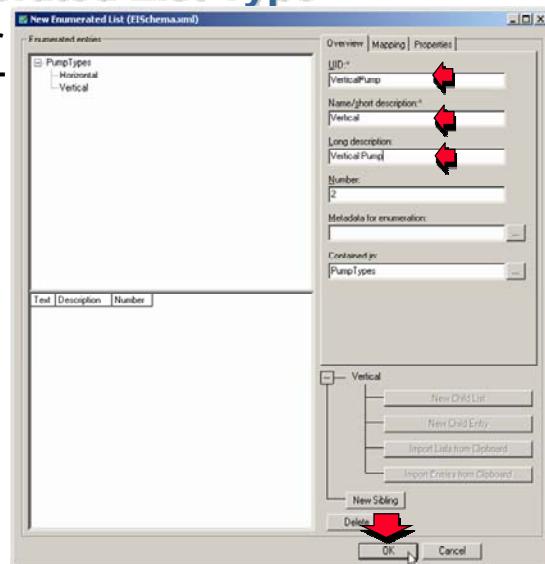
13. Create a second value for the list with the following information:

- Name – Vertical
- Long description – Vertical Pump
- UID -- VerticalPump
- Number – 2 (This value will appear automatically, if you gave the previous item a value of 1.)

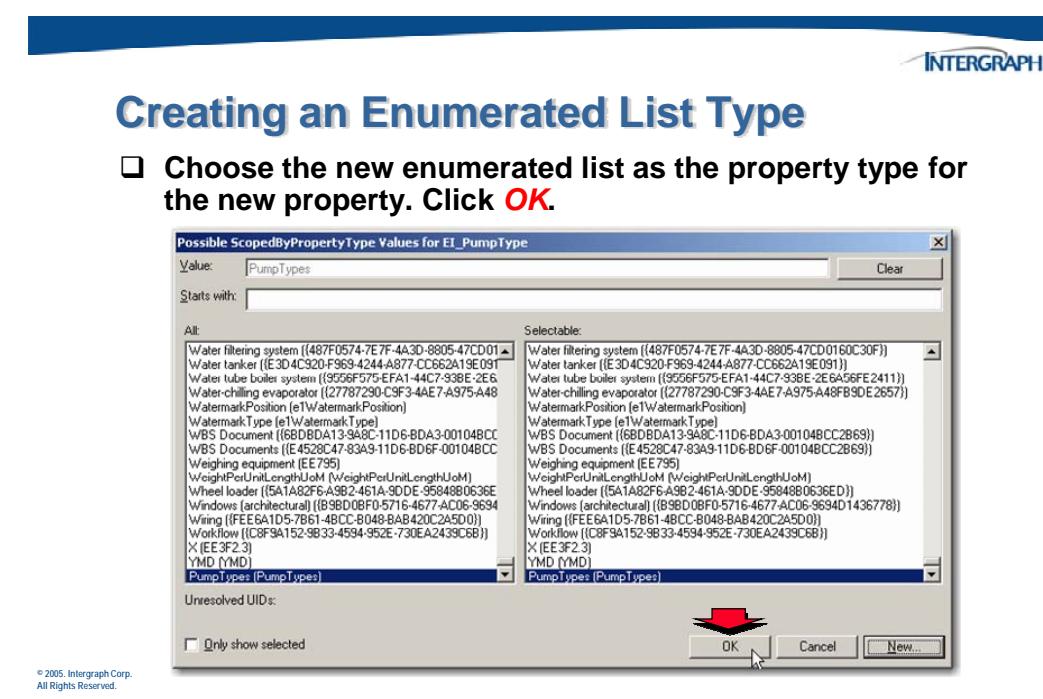
14. Click OK to create the list.

Creating an Enumerated List Type

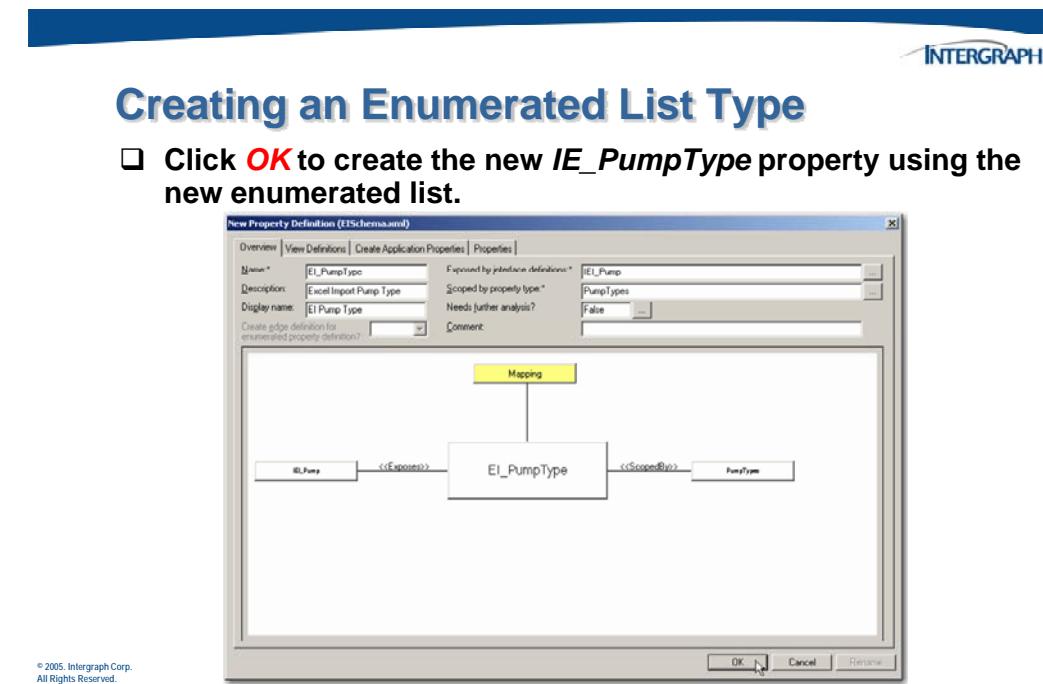
- Provide information for the other enum enum – Vertical.
- When you are done, click OK.



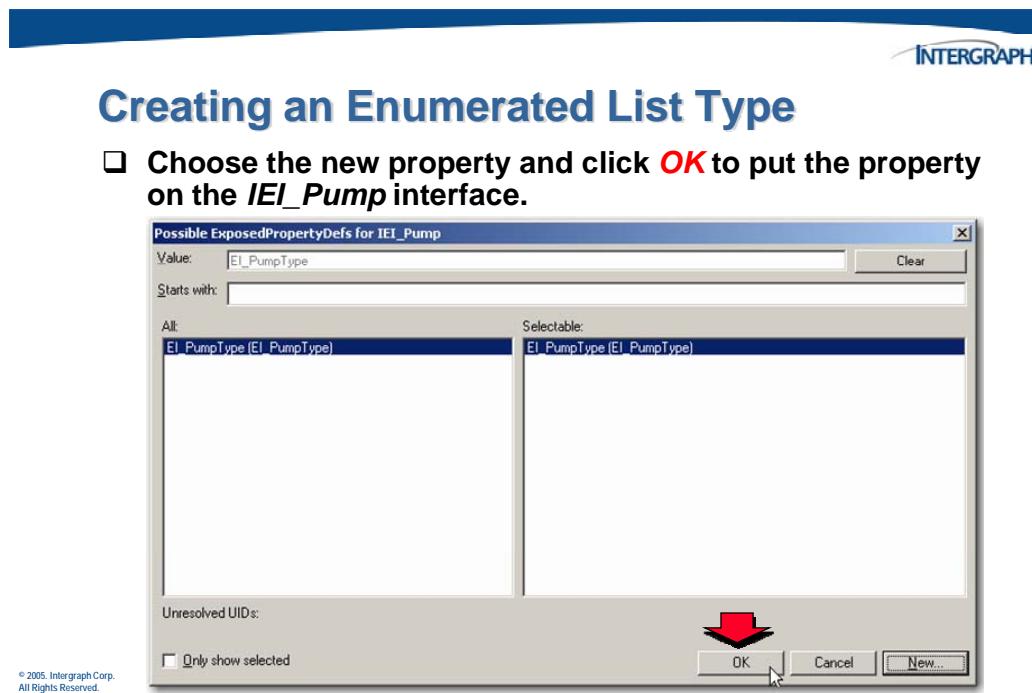
15. The new list now appears as a option in the list of property types. It should be selected automatically. Click **OK** to choose it.



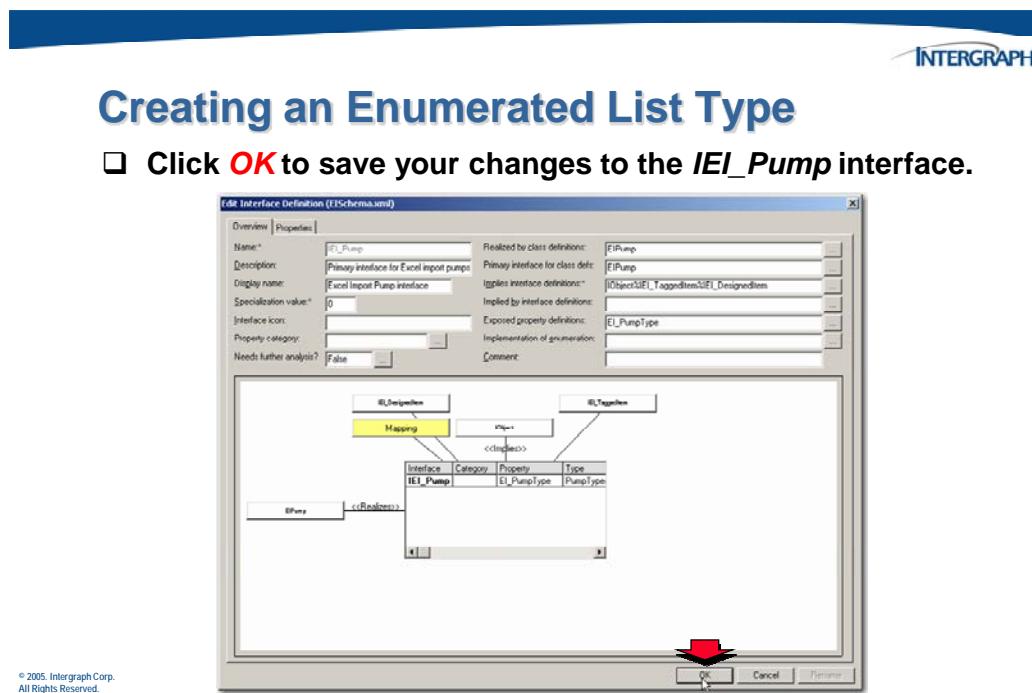
16. Click **OK** to create the new property.



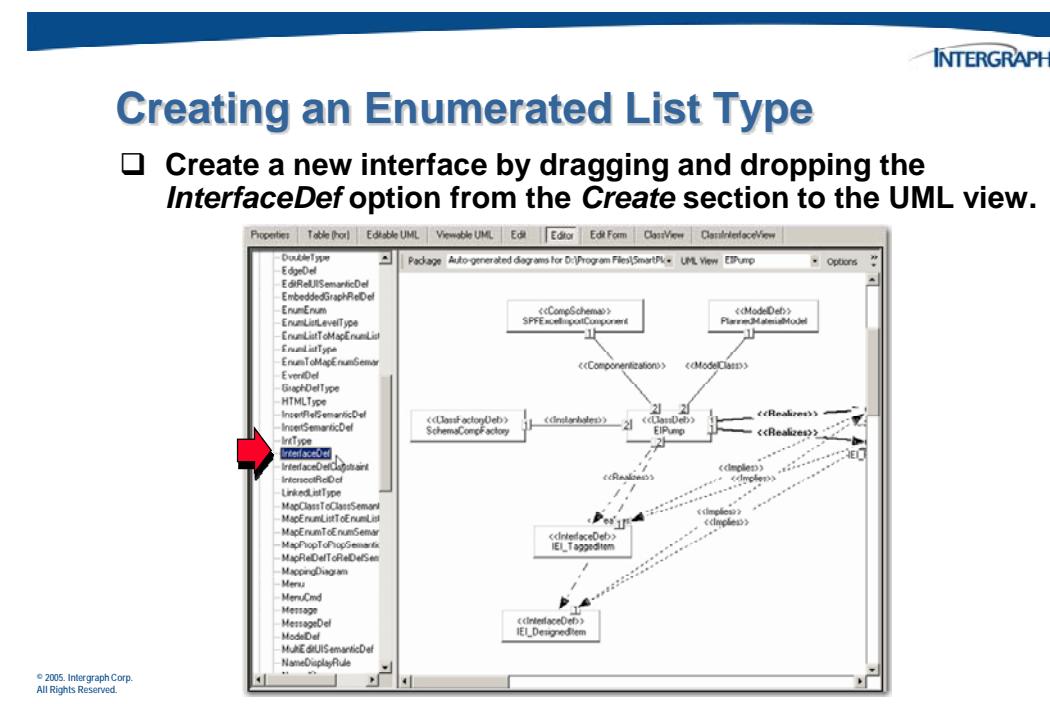
17. With the new property selected, click **OK**.



18. Click **OK**, again, to save the changes you made to the **IEI_Pump** interface.



19. Next, create a new interface that will also have a property defined by an enumerated list. Find the **InterfaceDef** option in the **Create** section of the **Editor** view, and drag and drop it into the UML view.



20. Provide the following information about the new interface.

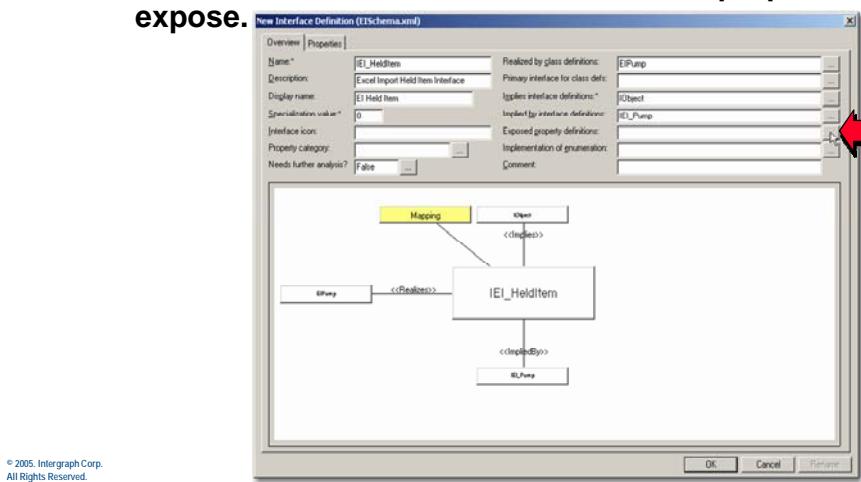
- Name** – IEI_HeldItem
- Description** – Excel Import Held Item Interface
- Display name** – EI Held Item
- Realized by class definitions** – EIPump
- Implies interface definitions** – IObject
- Implied by interface definitions** – IEI_Pump

21. Click the browse button beside the *Exposed property definitions* field.



Creating an Enumerated List Type

- Provide the required information about the new interface, and then click the browse button to find properties to expose.**

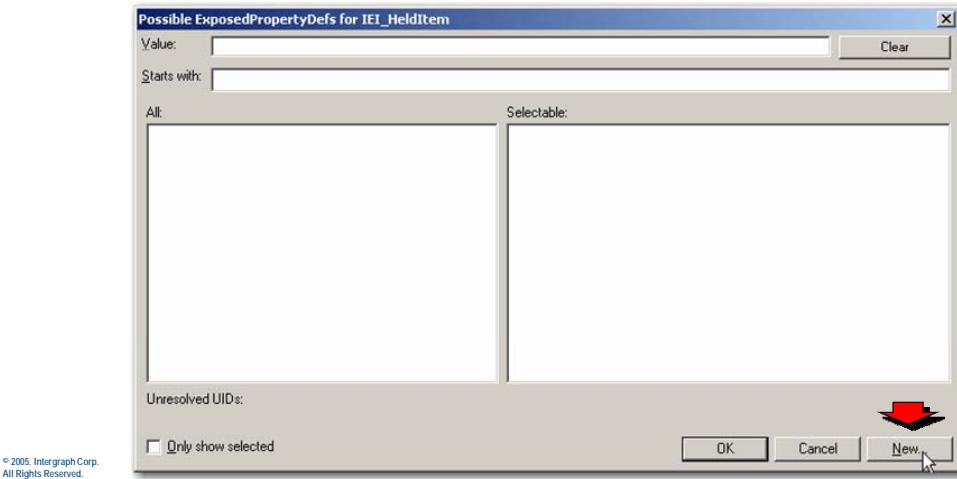


© 2005, Intergraph Corp.
All Rights Reserved.

22. Click the **New** button to create a new property.

Creating an Enumerated List Type

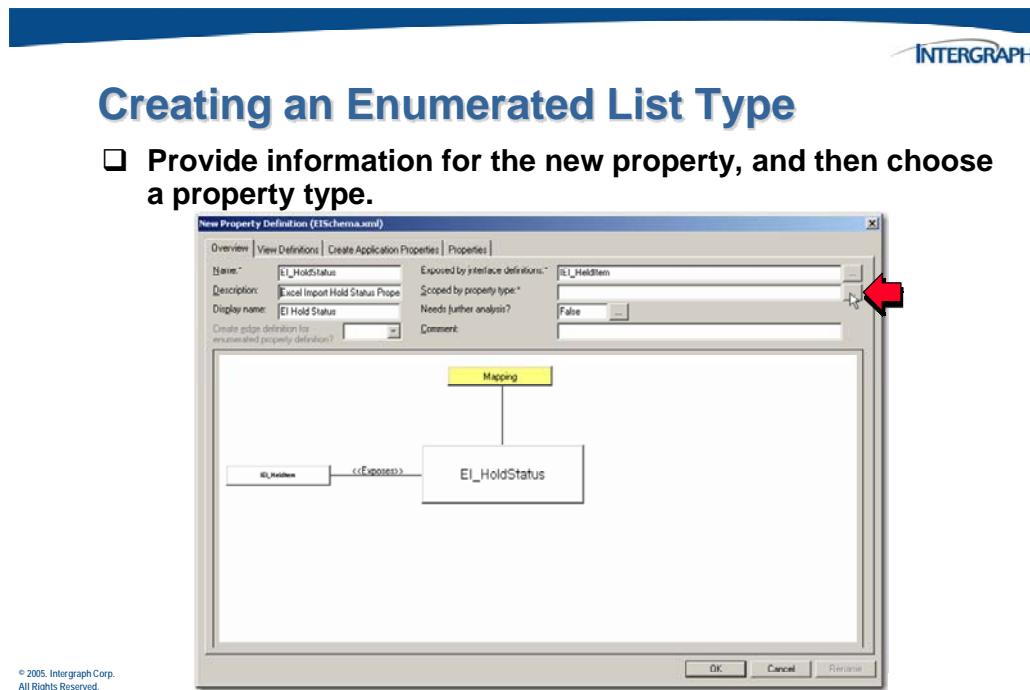
- Click the **New** button to create a new property to expose on this new interface.



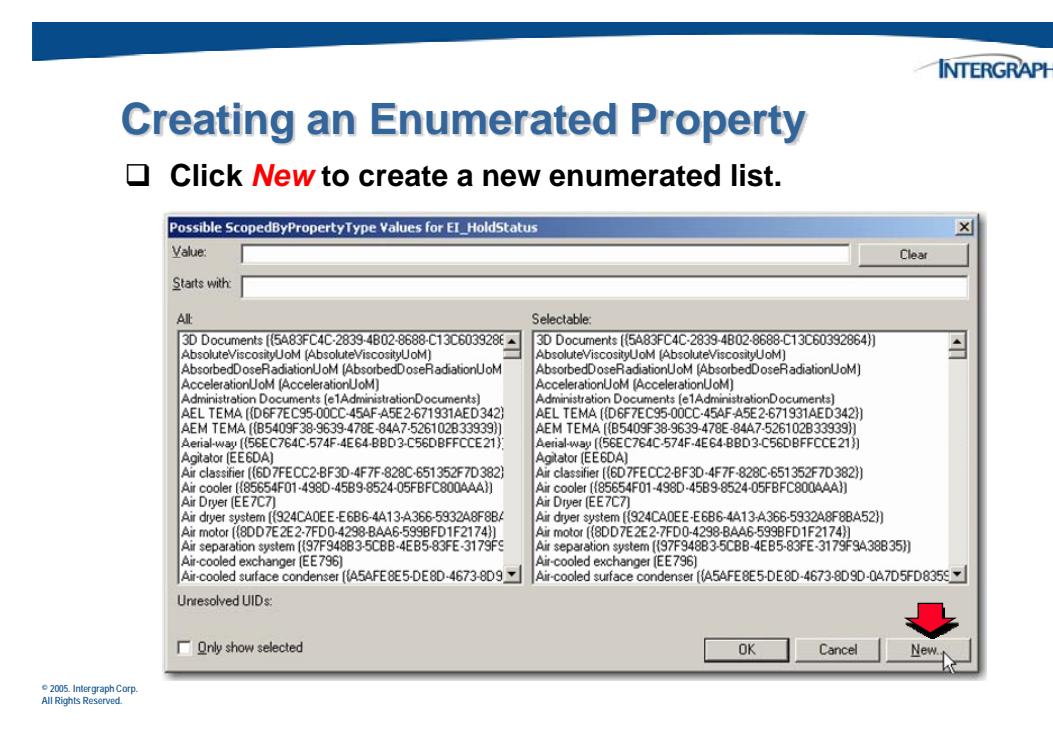
23. Create a new property with the following information:

- Name – EI_HoldStatus
- Description – Excel Import Hold Status Property
- Display name – EI Hold Status
- Exposed by interface definition – IEI_HeldItem

24. Click the browse button beside the **Scoped by property type** field.



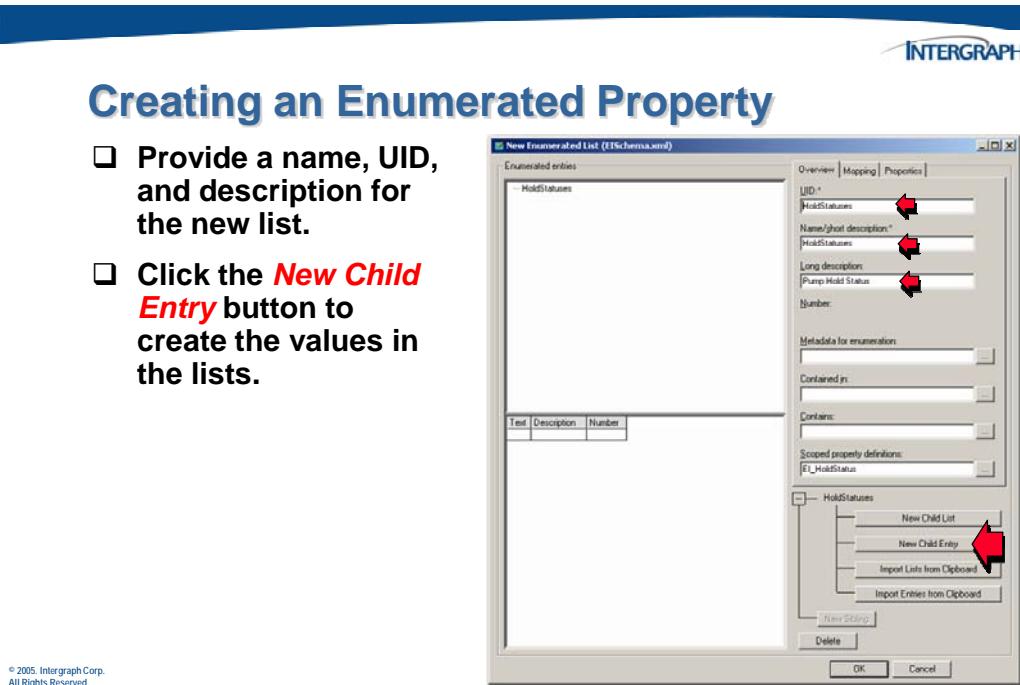
25. Click the **New** button to create a new property type. When prompted, choose the **EnumListType** as the new kind of property type you will be creating.



26. Create a new list with the following information:

- Name** – HoldStatuses
- Description** – Pump Hold Status
- UID** -- HoldStatuses

27. Click **New Child Entry** to create the first value in the list.



© 2005, Intergraph Corp.
All Rights Reserved.

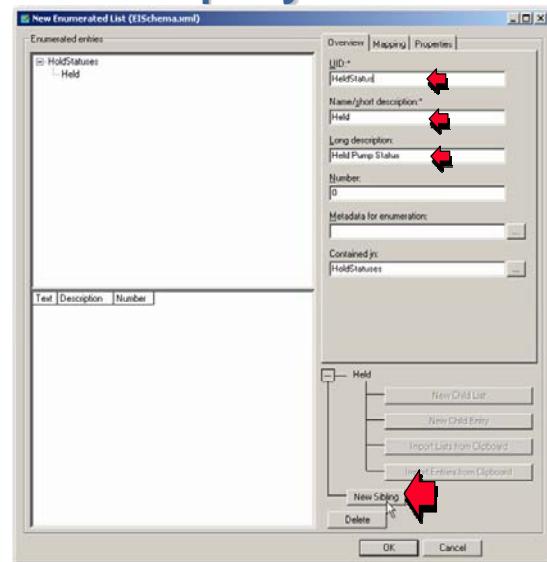
28. Create a list value with the following information:

- Name** – Held
- Description** – Held Pump Status
- UID** -- HeldStatus

29. Click the *New Sibling* button to create another list value.

Creating an Enumerated Property

- Provide a name, description, and UID for the first enum enum in the list.**
- Click *New Sibling* to create another list value.**



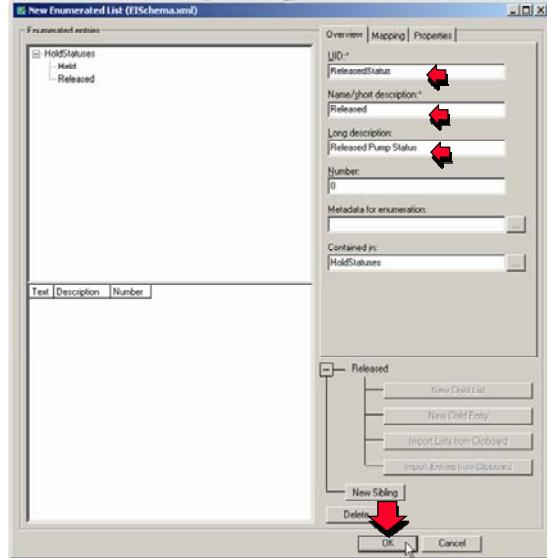
30. Create a second list value with the following information:

- Name** – Released
- Description** – Released Pump Status
- UID** -- ReleasedStatus

31. Click **OK** to create the new enumerated list.

Creating an Enumerated Property

- Provide a name, description, and UID for the next enum enum in the list.**
- When you are done creating list values, click the **OK** button to close this dialog.**

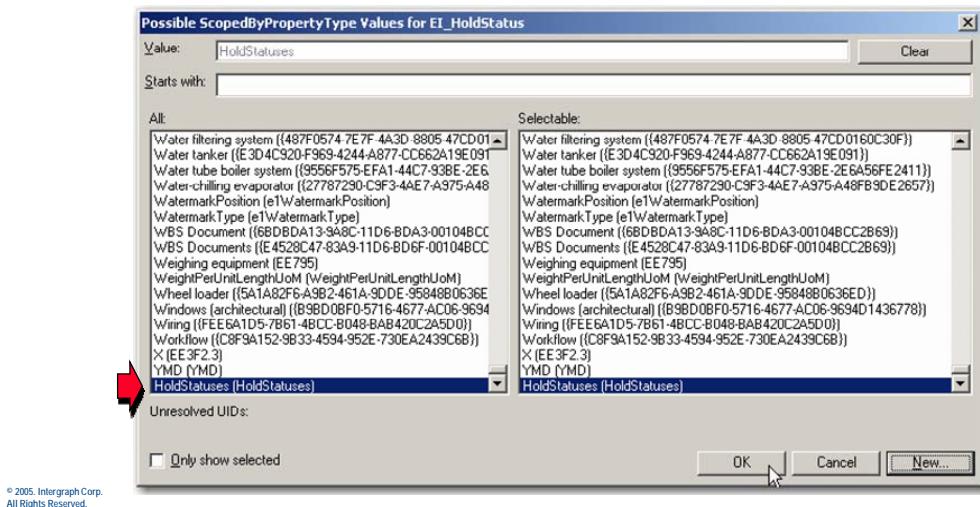


© 2005, Intergraph Corp.
All Rights Reserved.

32. From the list of property types, find the new enumerated list. It should be selected automatically. Click **OK** to choose it.

Creating an Enumerated List Type

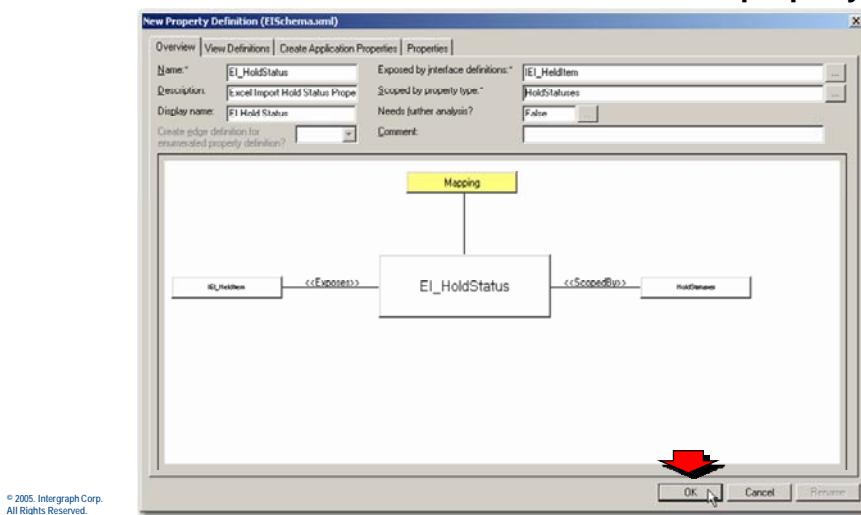
- Choose the new enumerated list as the property type.



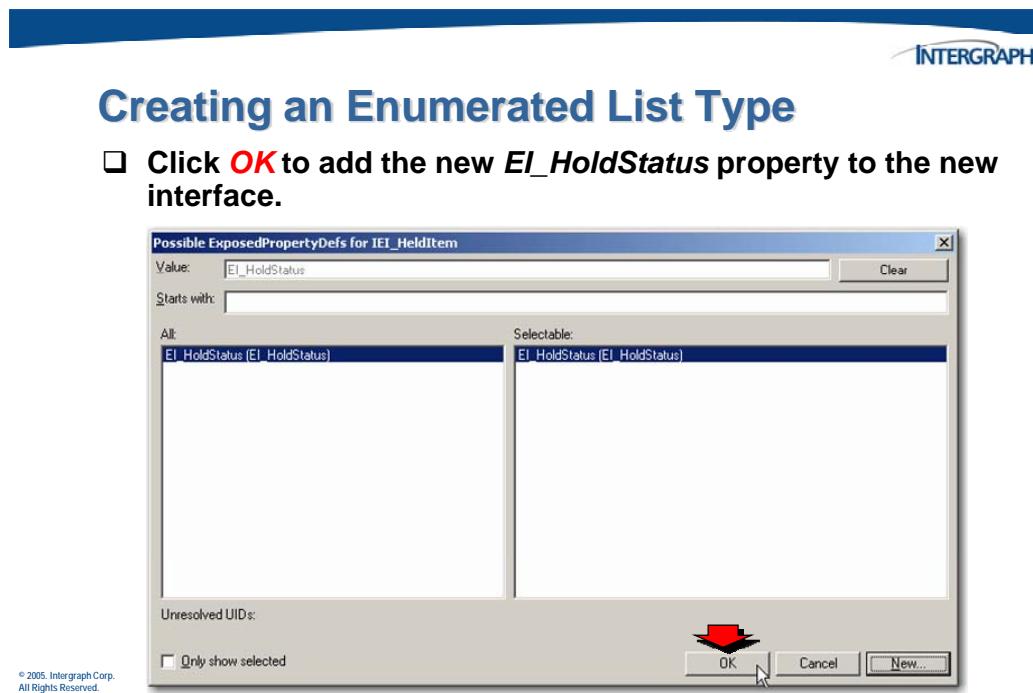
33. Click **OK** to create the new property.

Creating an Enumerated List Type

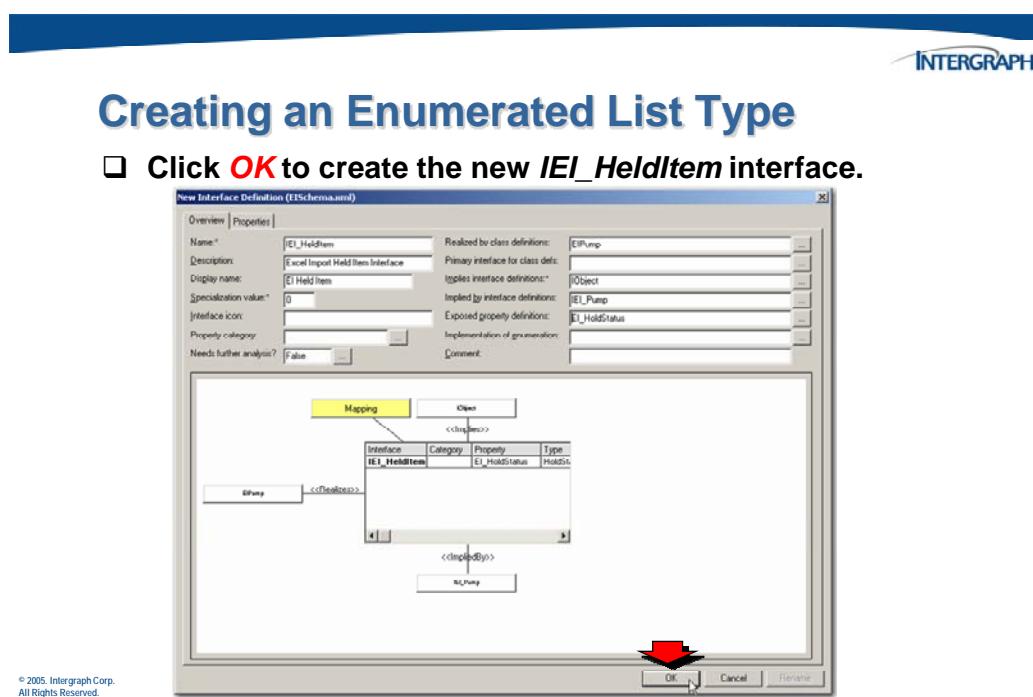
- Click **OK** to create the new *EI_HoldStatus* property.



34. The new property will be selected. Click ***OK***.



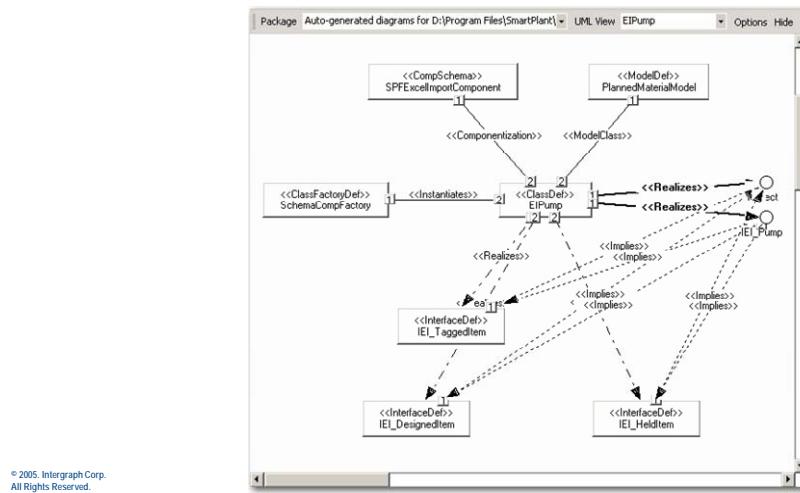
35. Click ***OK*** to create the new interface definition.



36. The new interface will appear in the UML window.

Creating an Enumerated List Type

- ❑ View the new interface in the UML window.



Review of the EIPump shows that we have now modeled all the requirements for our new class of pump.

Creating an Enumerated List Type

- ❑ From the tree, find the *EIPump* and view it with the **Properties** view. Compare the list to the requirements

The screenshot shows the Schema Editor with the EIPump object selected. The left pane displays the properties of EIPump, and the right pane shows the properties of the IEI_Pump interface. Red arrows point from specific properties in the left pane to their corresponding definitions in the right pane.

Property	Type	Description
TerminationUser	string	Termination User
User	string	User ID
TerminationDate	SPFDateTime	Termination Date
CreationDate	SPFDateTime	Creation Date
OID	string	
LastUpdatedDate	SPFDateTime	
ContainerID	string	Creation User
DomainID	string	Container ID
Config	string	Domain UID
UID	string128	Object Configuration
PumpType	string	
IEI_Pump	object	
IEI_PumpType	PumpTypes	Excel Import Pump Type
IEI_TagPrefix	string	Excel Import Tag Prefix
IEI_SequenceNo	string	Excel Import Tag Sequence Number
IEI_HeldStatus	string	Excel Import Hold Status
IEI_DesignatedItem	object	
IEI_TagSequenceNo	string	Excel Import Tag Sequence Number
IEI_MaxTemp	SPFDouble	Excel Import Max Temperature Property
IEI_MaxPress	SPFDouble	Excel Import Max Pressure Property
IEI_HoldStatus	HoldStatuses	Excel Import Hold Status Property

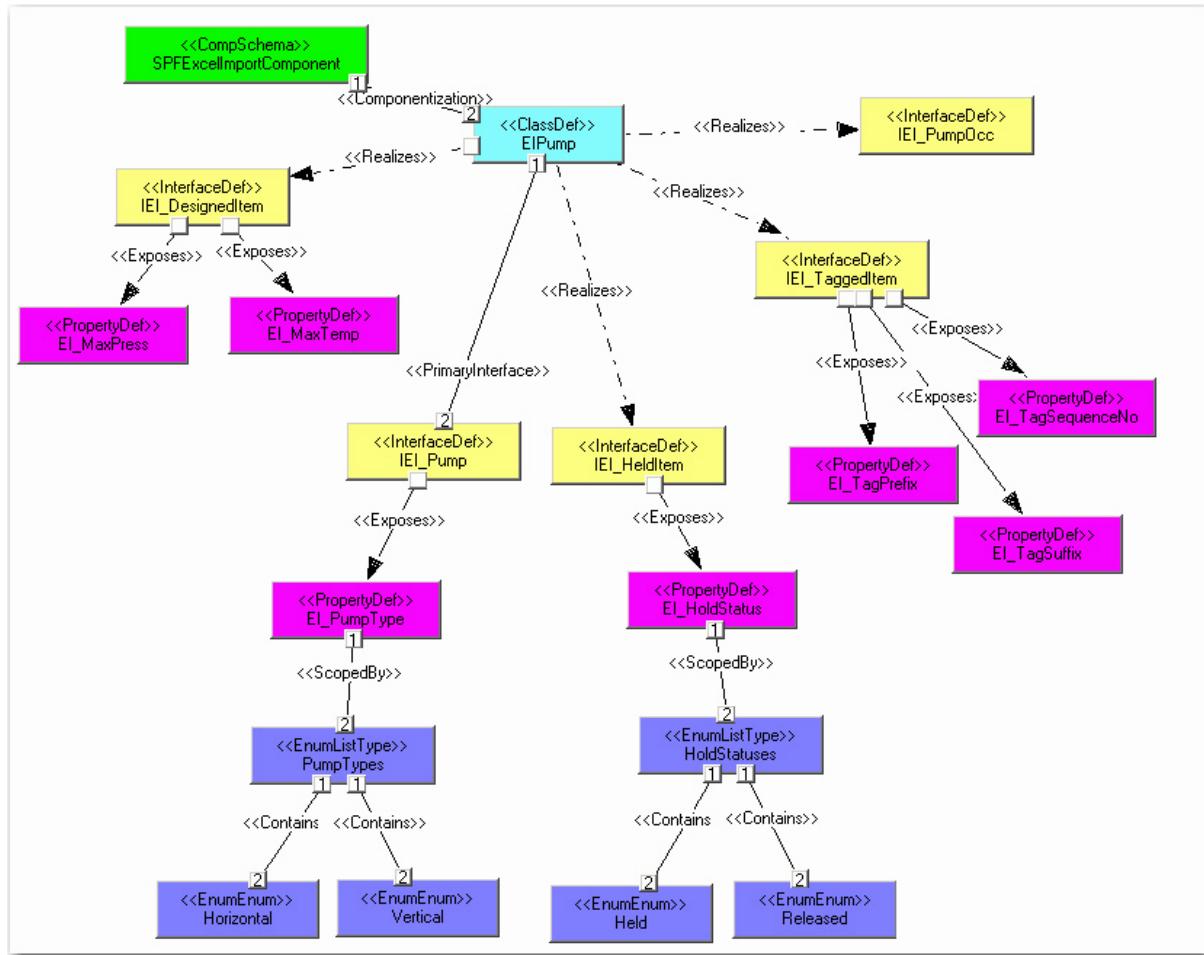
Properties View (Left):

- Name (text input)
- Description (text input)
- Pump Type (dropdown labeled '(Picklist)')
- Tag Prefix (text input)
- Tag Sequence Number (text input)
- Tag Suffix (text input)
- Maximum Pressure (text input)
- Maximum Temperature (text input)
- Hold Status (dropdown labeled '(Picklist)')

Properties View (Right):

- TerminationUser (string)
- User (string)
- TerminationDate (SPFDateTime)
- CreationDate (SPFDateTime)
- OID (string)
- LastUpdatedDate (SPFDateTime)
- ContainerID (string)
- DomainID (string)
- Config (string)
- UID (string128)
- PumpType (string)
- IEI_Pump (object)
- IEI_PumpType (PumpTypes)
- IEI_TagPrefix (string)
- IEI_SequenceNo (string)
- IEI_HeldStatus (string)
- IEI_DesignatedItem (object)
- IEI_TagSequenceNo (string)
- IEI_MaxTemp (SPFDouble)
- IEI_MaxPress (SPFDouble)
- IEI_HoldStatus (HoldStatuses)

The following graphic is an illustration of all the pieces modeled so far in Chapters 3 and 4.



4.6 Relationship Definitions

All relationship definitions in the schema are of class ***RelDef***, which is part of the meta schema. A RelDef has two ends, **End1** and **End2**. The positive direction for navigating this relationship is from End1 to End2. The reverse direction is from End2 to End1. A relationship can only exist between objects that support (realize) the interface definitions at each end of the relationship definition.

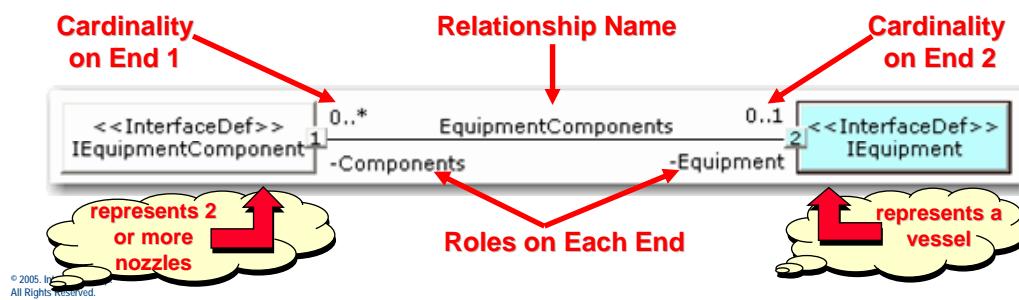


Relationship Definitions

Relationship definitions (***RelDef***) are associations between interface definitions. A RelDef will define two object instances that fulfill the roles on each end of the relationship.

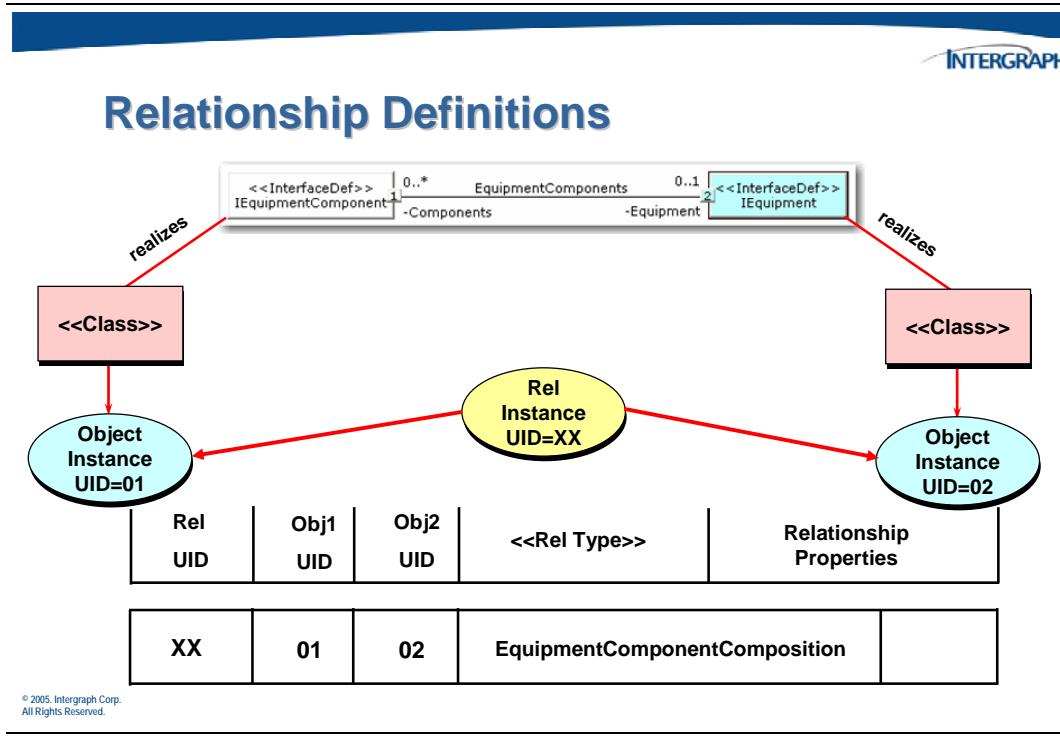
Relationship definitions also define cardinalities on each end of the relationship.

The following is an example of a *relationship* showing equipment components that can be associated with equipment e.g. nozzles on a vessel.



Cardinalities provide the constraints for relationships by defining the minimum and maximum numbers of participants at each end of a relationship. For example, in the EquipmentComponentComposition relationship there can be zero to many (0..*) objects that have the role of Equipment Components but only zero or one (0..1) object that can have the role of Equipment in each instance of the relationship. For example there could be many nozzles and some may be associated to a vessel but that nozzle instance would not be connected to more than one (i.e. multiple) vessel.

When something is published, a Rel (relationship) is an instantiation of a RelDef.



Relationship Definitions

Engineering tools can publish objects at different times.

Users can publish nozzles ahead of equipment.

The software has to decide “can this relationship be resolved”? An example would be P&ID could publish nozzles first and then vessels are published later. The software would suppress errors because there is no harm in doing this since the vessels CAN be published at a later time.

When defining a relationship definition, End 1 and End 2 interfaces depends on which way users are going to query.

The locality of reference, which is shown as part of the role name, shows how this relationship will be instantiated:

- “I’ve got to publish.”
- “You have to publish.”
- “Someone will publish someday.”



Relationship Definitions

0..*	EquipmentComponents	0..1
-Components		-Equipment

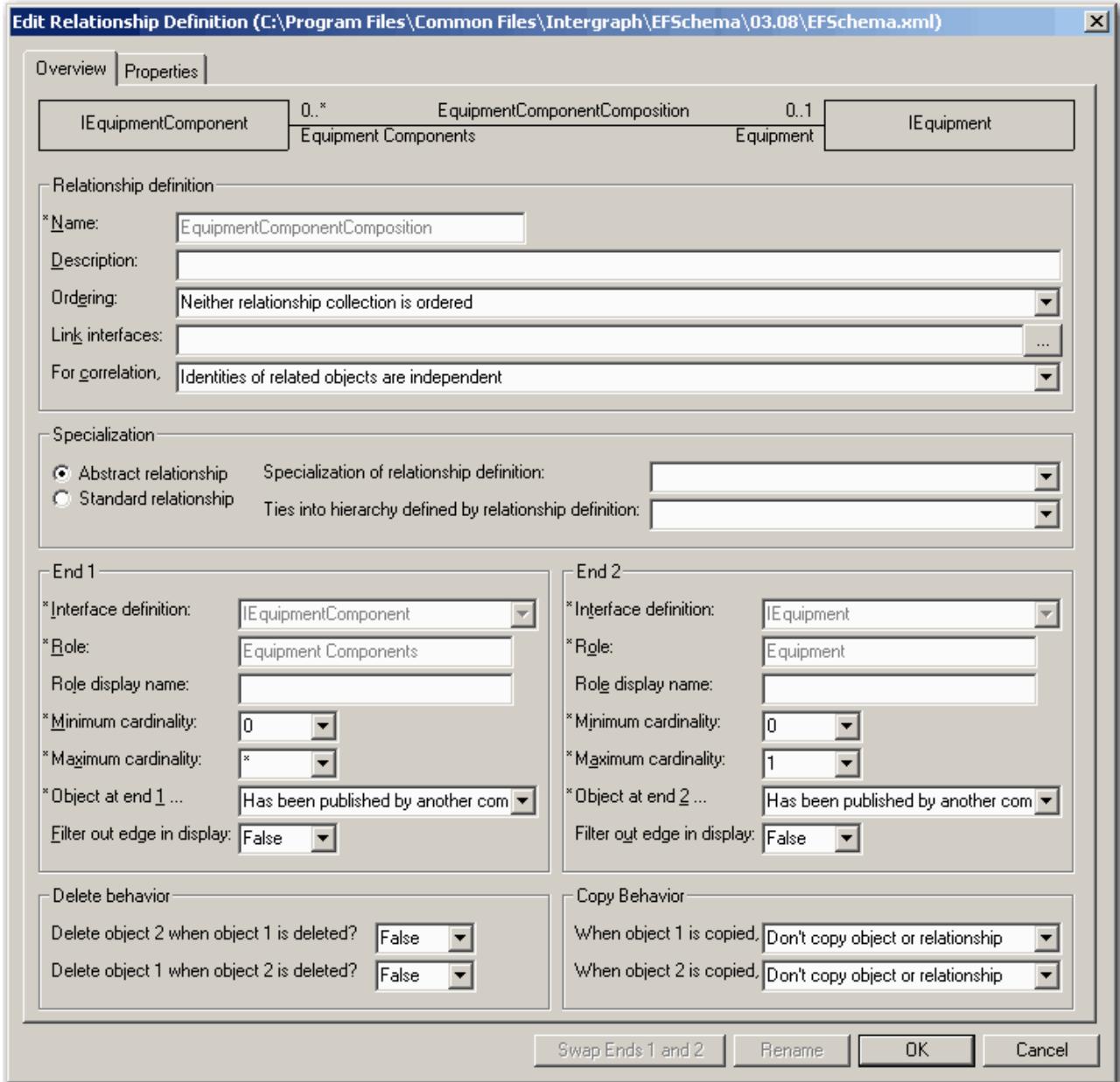
Locality of Reference

- has been published by another component
- + must be in the same container as object 2
- # must be published by the same tool

4.6.1 Properties of Relationship Definitions

When you create or edit a relationship definition in the Schema Editor, you can define the following options:

- Name** – Specifies the name of the relationship definition.
- Description** – Provides a more detailed description of the relationship definition.



- Ordering** – Specifies whether the relationship is ordered (optional). If objects in a relationship are ordered, they can be ordered from end 1 to end 2 or from end 2 to end 1, but not both. Ordering relationships is important when the

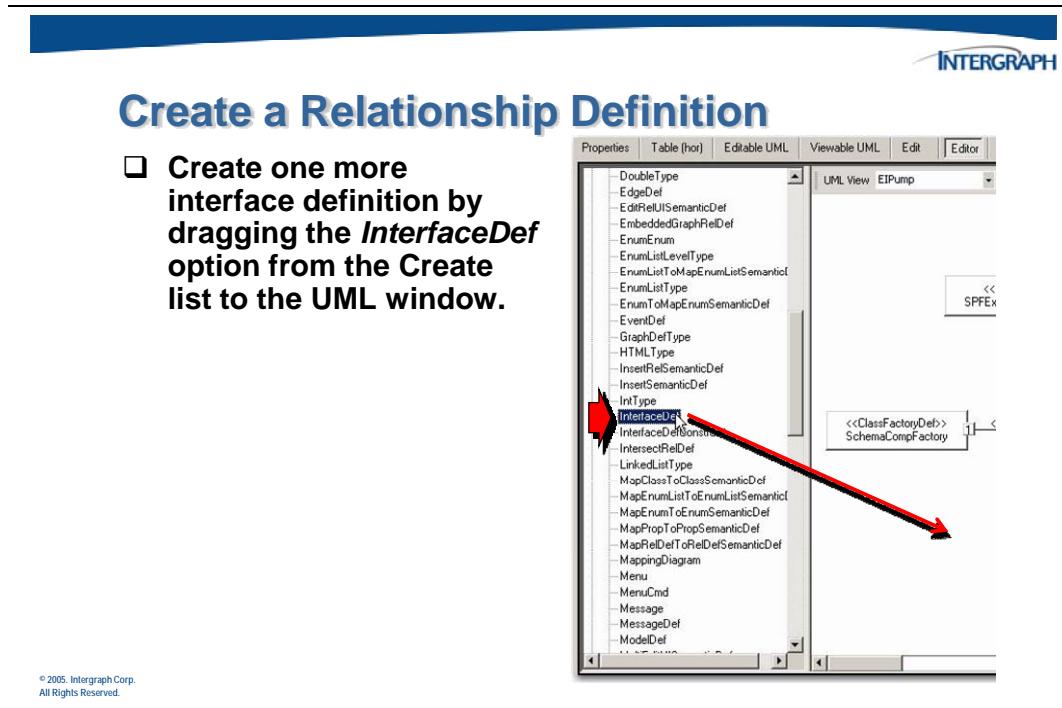
sequence of the related objects is important. For example, the order of the piping ports for a piping connector is important in understanding the logical connectivity. Conditional ordering is not currently supported in SmartPlant.

- Abstract relationship** – Indicates that the relationship definition is abstract. An abstract relationship cannot be directly instantiated (no Rel objects can use the abstract relationship definition). Abstract relationship definitions are used for generic traversal operations. An abstract relationship definition must be specialized by one or more concrete (standard) relationship definitions. (Note: **Do NOT** publish abstract relationships.)
- Standard relationship** – Indicates that the relationship definition is a standard relationship. Standard relationships are also called concrete relationships. Standard or concrete relationship definitions can be used for instantiated relationships. A relationship definition can be either abstract or standard, but not both.
- Specialization of abstract relationship definition** – Specifies the name of the abstract relationship definition of which this standard relationship definition is a specialization. Any standard (concrete) relationship that is a specialization of an abstract relationship is also treated as an instance of that abstract relationship. For example, the PipingEnd1Conn, PipingEnd2Conn, and PipingTapOrFitting concrete relationship definitions are all specializations of the Connections abstract relationship definition. Therefore, a connection at end 1 is a PipingEnd1Conn relationship and also a Connections relationship (one Rel that is a specialization). A connection at end 2 is a PipingEnd2Conn and also a Connections relationship; a tap/fitting connection is a PipingTapOrFitting relationship and also a Connections relationship.
- End 1**
 - **Interface definition** – Identifies the interface definition at end 1 of the relationship definition.
 - **Role** – Specifies the name of the role for end 1 of the relationship.
 - **Minimum cardinality** – Specifies the minimum number of participants allowed on end 1 of the relationship. Possible values are 0, 1, and 2.
 - **Maximum cardinality** – Specifies the maximum number of participants allowed on end 1 of the relationship. Possible values are 1, 2, or * (many).
 - **Object at end 1** – Specifies whether hanging relationships are allowed for the object at end 1 of the relationship. Objects can be required to be located in the same container (typically a document) as the other object in the relationship, to be published by the same authoring tool as the other object, or to be published by another component.
 - **Filter out edge in display** – Specifies whether you want to see the relationship edge automatically created from end 1 to end 2 of this relationship in the Schema Editor user interface. If this option is set to **True**, user access for this edge definition can be controlled in the SPF client by user group security.
- End 2**

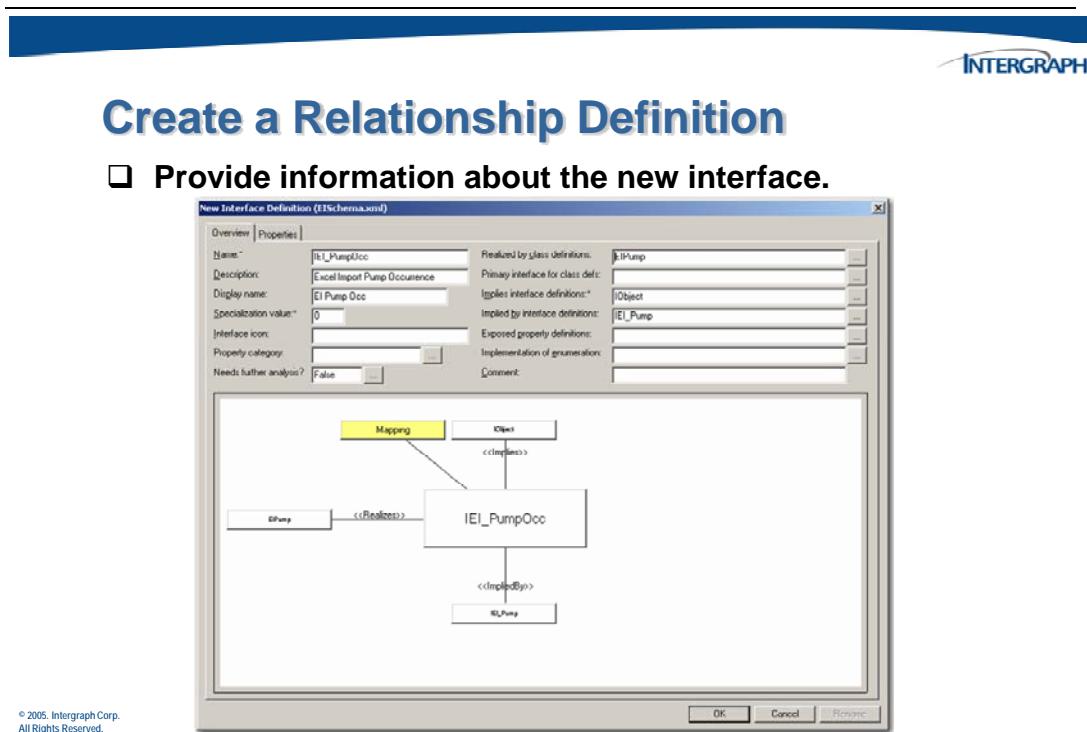
- **Interface definition** – Identifies the interface definition at end 2 of the relationship definition.
 - **Role** – Specifies the name of the role for end 2 of the relationship.
 - **Minimum cardinality** – Specifies the minimum number of participants allowed on end 2 of the relationship. Possible values are 0, 1, and 2.
 - **Maximum cardinality** – Specifies the maximum number of participants allowed on end 2 of the relationship. Possible values are 1, 2, or * (many).
 - **Object at end 2** – Specifies whether hanging relationships are allowed for the object at end 2 of the relationship. Objects can be required to be located in the same container (typically a document) as the other object in the relationship, to be published by the same authoring tool as the other object, or to be published by another component.
 - **Filter out edge in display** – Specifies whether you want to see the relationship edge automatically created from end 2 to end 1 of this relationship in the Schema Editor user interface. If this option is set to **True**, user access for this edge definition can be controlled in the SPF client by user group security.
- Delete object 2 when object 1 is deleted?** – Indicates whether you want the software to delete the object on end 2 of the relationship when the object on end 1 is deleted.
- Delete object 1 when object 2 is deleted?** – Indicates whether you want the software to delete the object on end 1 of the relationship when the object on end 2 is deleted.
- When object 1 is copied** – Specifies what you want the software to do when the object on end 1 of the relationship is copied. You can choose whether to copy the object on end 2 as well, copy the object and relationship, leaving the relationship pointing to the same object that the first relationship pointed to, or not to copy the object or relationship at all.
- When object 2 is copied** – Specifies what you want the software to do when the object on end 2 of the relationship is copied. You can choose whether to copy the object on end 1 as well, copy the object and relationship, leaving the relationship pointing to the same object that the first relationship pointed to, or not to copy the object or relationship at all.

4.6.2 Interactive Activity – Creating Relationship Definitions

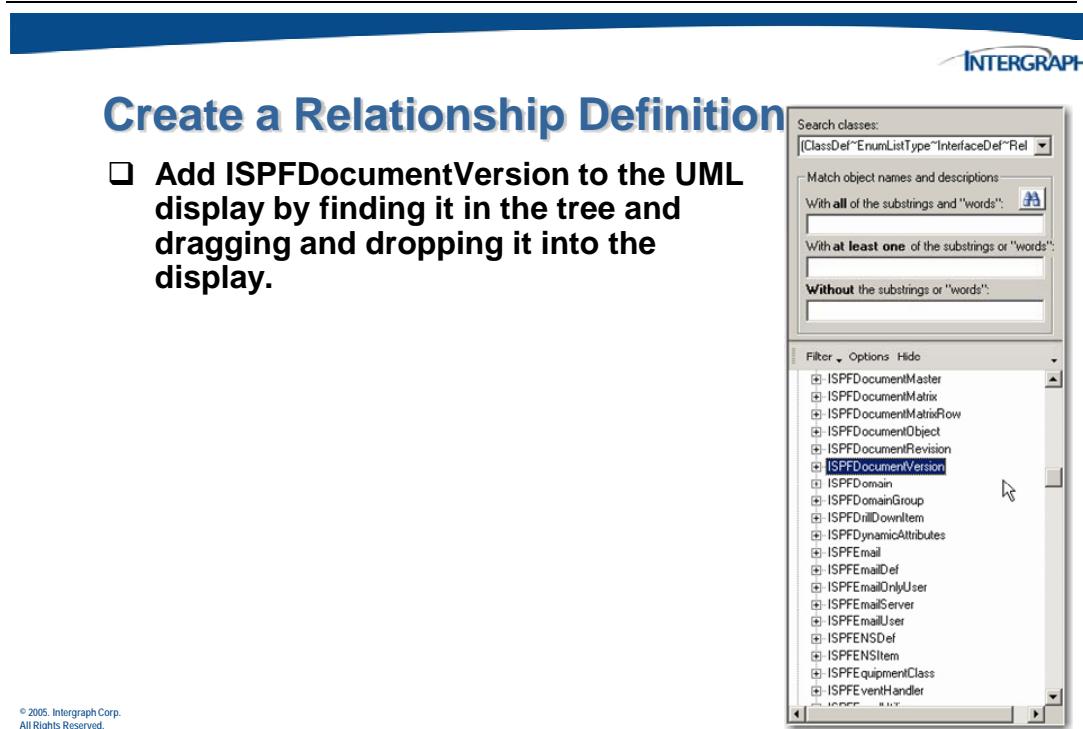
1. Create a new interface. Drag and drop the *InterfaceDef* option from the *Create* section of the *Editor* view into the UML window.



2. Create a new interface with the following information:
 - Name** – IEI_PumpOcc
 - Description** – Excel Import Pump Occurrence
 - Display name** – EI Pump Occ
 - Realized by class definition** – EIPump
 - Implies interface definitions** – IObject
 - Implied by interface definitions** – IEI_Pump
3. Click **OK** when you are done completing the required fields.



-
4. Add the **ISPFDocumentVersion** interface to the UML view. Find the interface in the tree on the left-hand side of the window, and drag and drop it into the UML window.



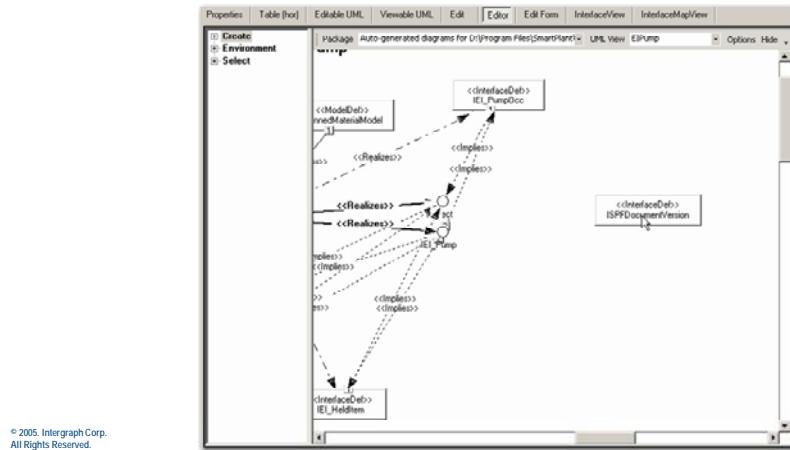
© 2005, Intergraph Corp.
All Rights Reserved.

The interface will appear in the UML view, but at this point, it has no relationships to any of the other items in the display.

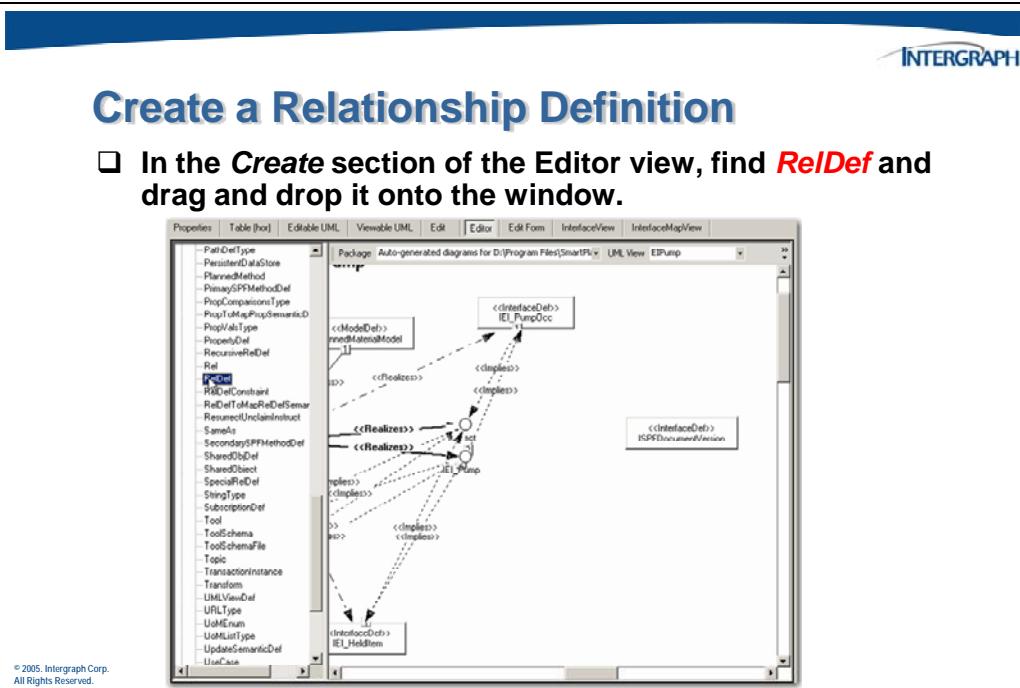


Create a Relationship Definition

- ❑ ISPFDocumentVersion has no relationships with any of the items currently displayed, therefore, it will have no connectors in the UML window.



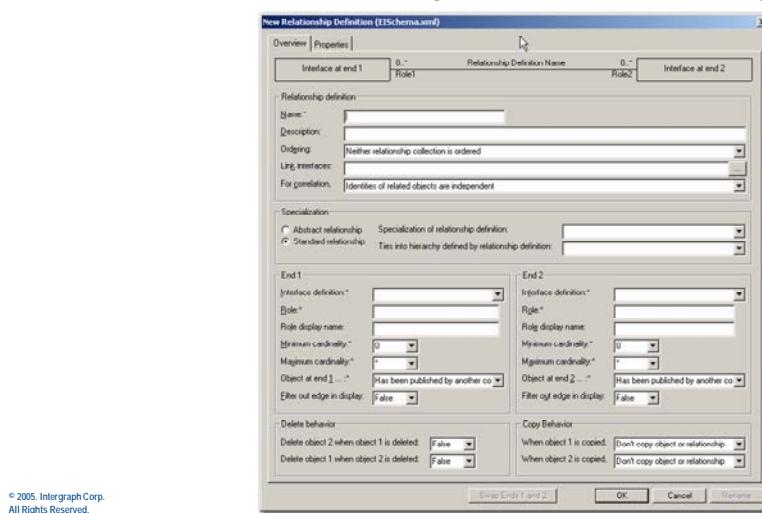
5. In the *Create* section of the the *Editor* view, find the *RelDef* option. Drag and drop it into the UML display.



6. The *New Relationship Definitions* form will appear.

Create a Relationship Definition

- The *New Relationship Definition* form will appear.



7. Name the new reldef: ***DesignDocumentVersionToPump***.
-



Create a Relationship Definition

- In the top portion of the form, provide a name for the new relationship definition. This relationship will be between our new class of pump and SPF document versions. The name of the relationship should indicate that.

A screenshot of a Windows-style dialog box titled "Relationship definition". It contains five input fields:

- Name:
- Description:
- Ordering:
- Link interfaces: ...
- For correlation:

8. Provide information about each end of the relationship definition:

End 1

- Interface definitions – IEI_PumpOcc
- Role – Pumps (Excel)
- Role display name – Pumps (Excel)
- Minimum cardinality – 0
- Maximum cardinality – *

End 2

- Interface definitions – ISPFDocumentVersion
- Role – Document Versions (Design)
- Role display name – Document Versions (Design)
- Minimum cardinality – 0
- Maximum cardinality – *

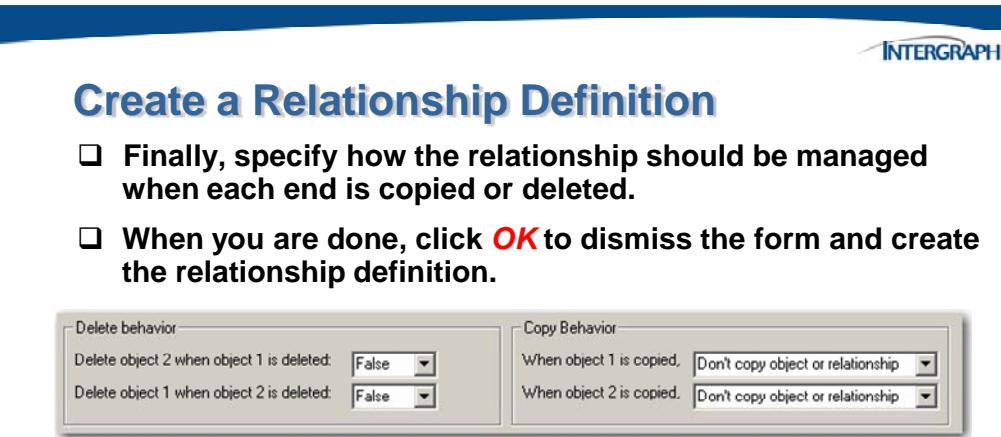


Create a Relationship Definition

- In the middle section of the form, provide information about the two ends of the relationship. First select the interfaces that the relationship will join, then complete additional information for each of the two ends.

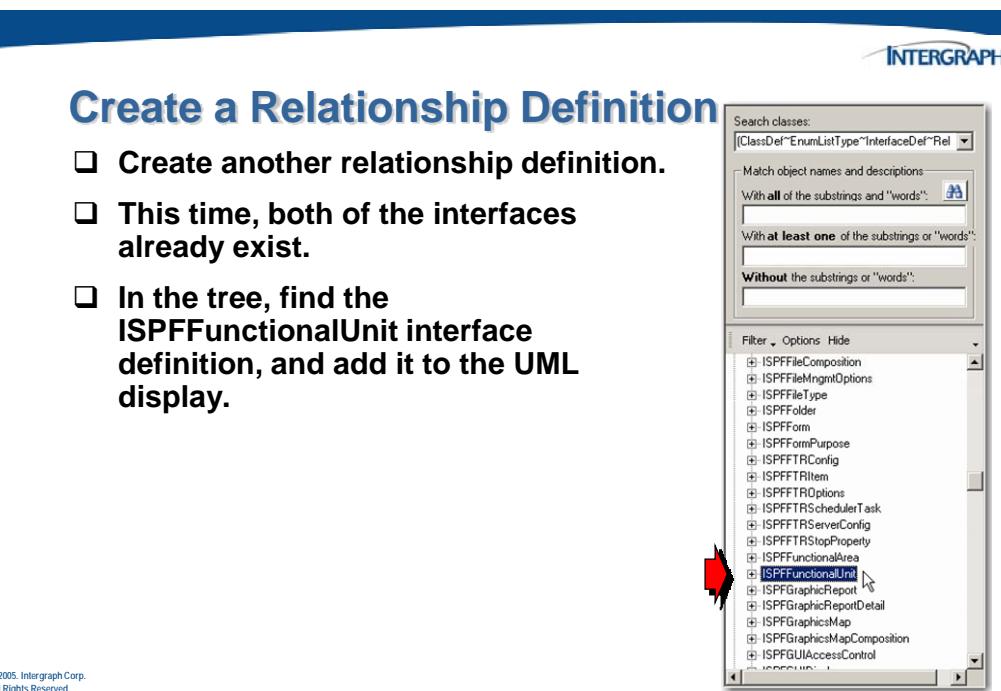
End 1 Interface definition: * <input type="text" value="IEI_PumpOcc"/> Role: * <input type="text" value="Pumps (Excel)"/> Role display name: <input type="text" value="Pumps (Excel)"/> Minimum cardinality: <input type="text" value="0"/> Maximum cardinality: <input type="text" value="*"/> Object at end 1 ... *: <input type="text" value="Has been published by another co"/> Filter out edge in display: <input type="text" value="False"/>	End 2 Interface definition: * <input type="text" value="ISPFDocumentVersion"/> Role: * <input type="text" value="Document Versions (Design)"/> Role display name: <input type="text" value="Document Versions (Design)"/> Minimum cardinality: <input type="text" value="0"/> Maximum cardinality: <input type="text" value="*"/> Object at end 2 ... *: <input type="text" value="Has been published by another co"/> Filter out edge in display: <input type="text" value="False"/>
---	---

9. Click **OK** to create the relationship definition.



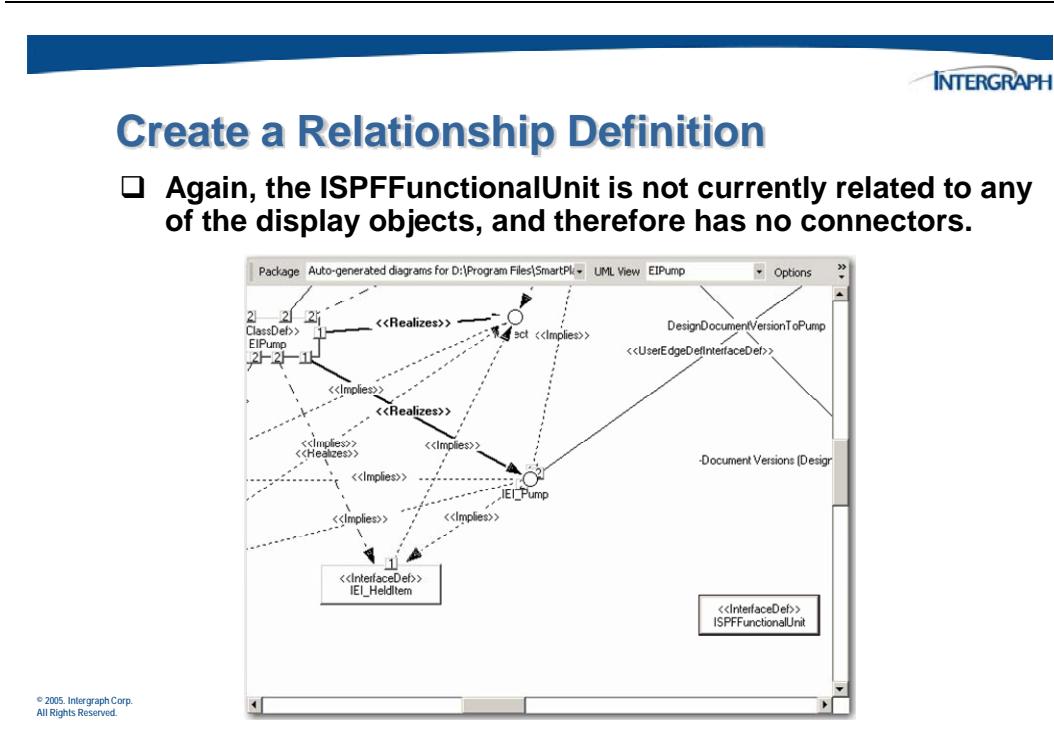
© 2005, Intergraph Corp.
All Rights Reserved.

10. To create the next relationship, find the **ISPFFunctionalUnit** interface definition in the tree. Drag and drop it into the UML window.



© 2005, Intergraph Corp.
All Rights Reserved.

The interface has no relationships with any of the other items currently displayed in the UML window.



11. To create the new relationship, drag and drop the **RelDef** option from the **Create** section of the **Editor** view into the UML window.

12. Create a relationship definition with the following information:

- Name -- ExcelPumpFunctionalUnit

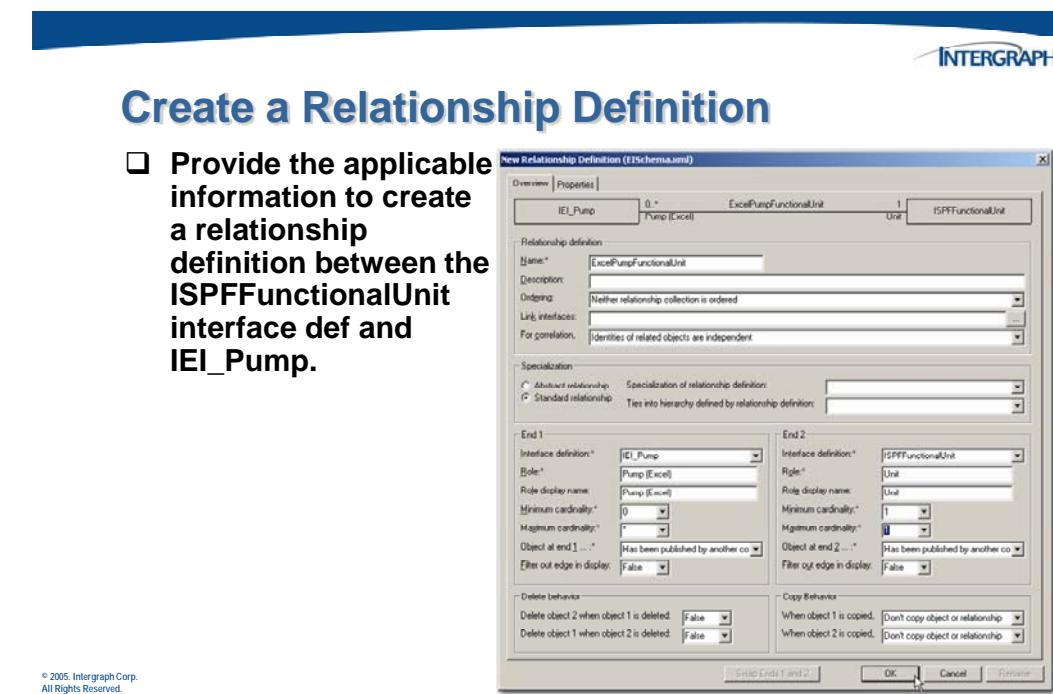
End 1

- Interface definitions – IEI_Pump
- Role – Pumps (Excel)
- Role display name – Pumps (Excel)
- Minimum cardinality – 0
- Maximum cardinality – *

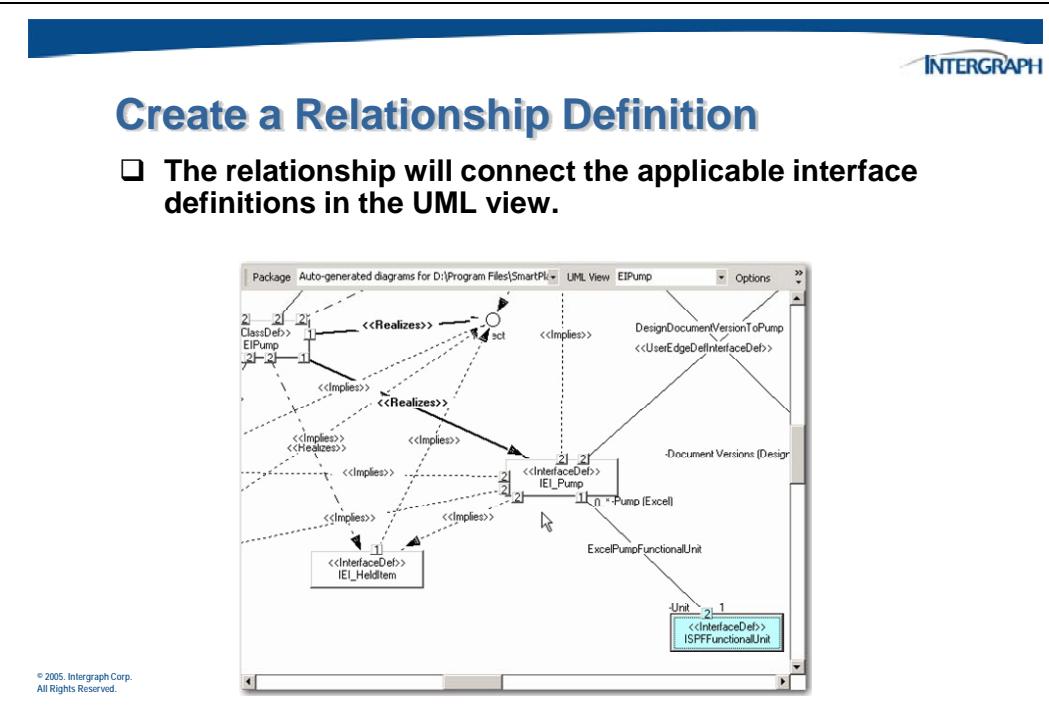
End 2

- Interface definitions – ISPFFunctionalUnit
- Role – Unit
- Role display name – Unit
- Minimum cardinality – 1
- Maximum cardinality – 1

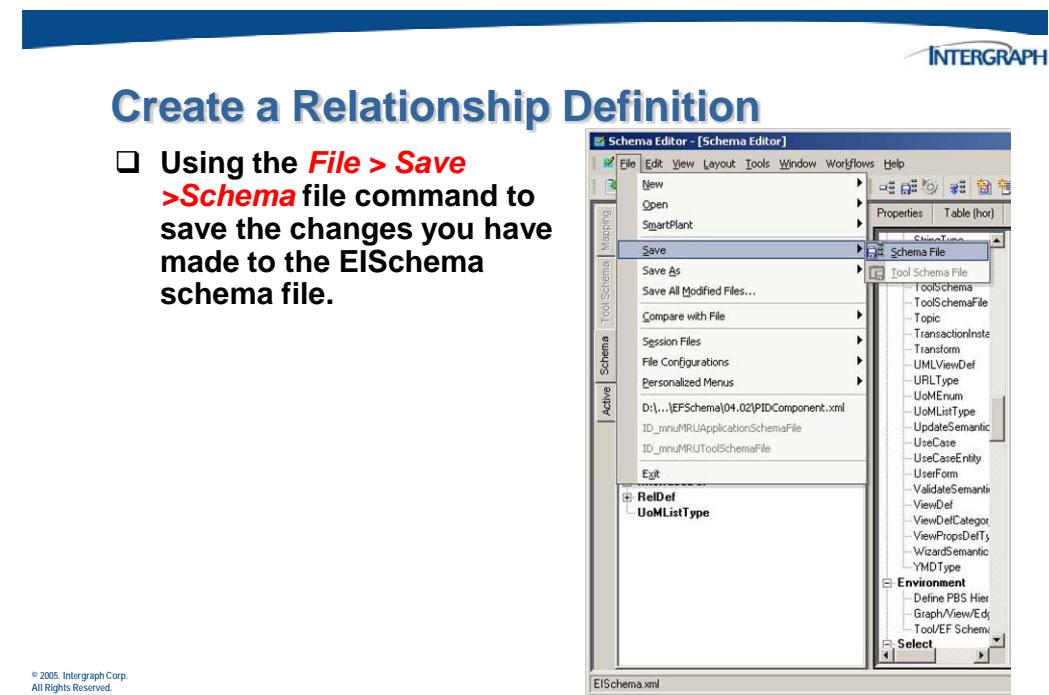
13. Click OK to create the relationship.



14. The new relationship will appear between the applicable interfaces in the UML view.



15. Save the changes you made to the EISchema file.



4.7 Shared Object Definitions

A shared object definition is used to group together similar classes that define the same object in different domains. For example, in a *Process Flow Diagram* (PFD) a user places a pump called P100 and publishes the document.

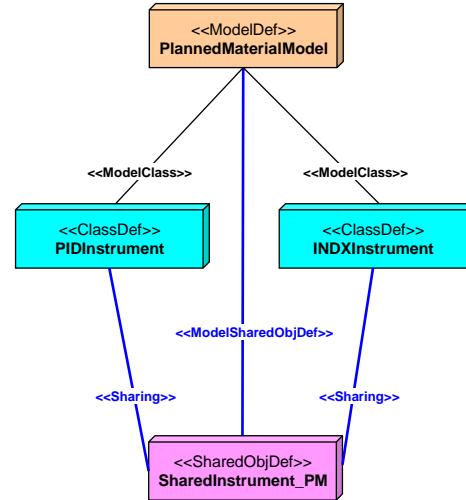


Shared Object Definitions

Shared object definitions (`SharedObjDef`) are used to group together similar classes that define the same object in different domains.

In concurrent engineering, two different tools can create and publish the same object. For example, instruments can be created and published in both SmartPlant P&ID and SmartPlant Instrumentation.

ClassDefs that can be shared have a Sharing relationship with SharedObjDefs.



© 2005, Intergraph Corp.
All Rights Reserved.

At the front end of the engineering process, not a lot of information is known about the pump other than the following:

- Some kind of pumping function is required.
- The pump is connected to streams.
- The streams have a certain fluid code running through them.
- The pump needs to pump at a certain volume.

When the PFD is defined, the user does not know how the pumping function with these general requirements will be accomplished. That information is left up to the P&ID portion of the workflow. In the PFD, the user sees some interfaces and properties of the pump, but not everything that defines the pump in SmartPlant.

In SmartPlant P&ID, engineers start adding value to the design by adding more and more information, such as the type of pump needed and so on. When SmartPlant P&ID publishes the document containing the same pump (P100) as part of the workflow, the users see more interfaces because the pump object is being enriched as it moves through the design process. When the engineer updates the Equipment Data Sheet in Zygad, there are thousands of properties describing the same object, further enriching it.

In each tool, the pump (P100) is published with a different class definition, including PFDProcessEquip, PIDProcessEquip, and EQDCentrifugalPump. However, a *SharedObjDef* indicates that these three classes definitions all define the same object in SmartPlant. The three class definitions coexist in a sharing relationship. The *SharedObjDef* collects information to indicate this sharing relationship. If you look at the *realizes* relationship for each class definition, you can see many of the same properties because as the object moves from one tool to another, properties are updated and the information is enriched.

When an object is created in the PFD, it has a unique identifier for the object (UID). For example, PFD UID AA1 (P100) gets published. Then, SmartPlant P&ID retrieves P100, some additional work is done, and it gets published as PID UID AA2 (P100).

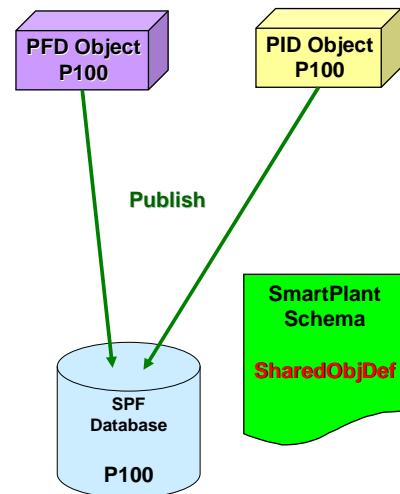
The two publishes establish a “*same as*” relationship. The same as relationship indicates that object AA1 is the same as object AA2.



Shared Object Definitions

In SmartPlant Foundation, interfaces and properties that are shared are overlaid (merged).

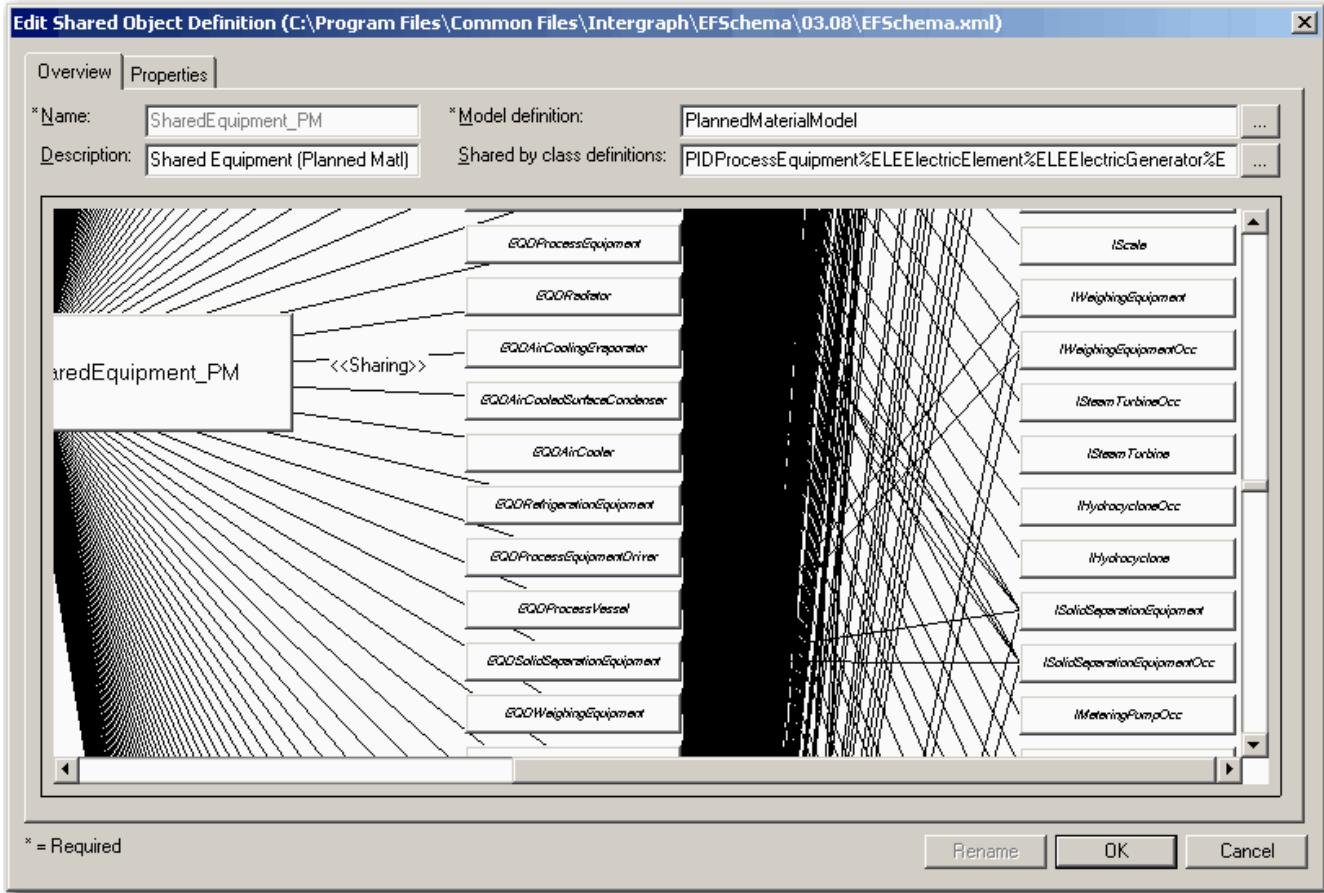
Interfaces that are not shared are appended to the object in the SPF data database.



4.7.1 Properties of a Shared Object Definition

Shared object definitions have the following properties:

- Name** – Specifies the name of the shared object definition.
- Description** – Specifies the description of the shared object definition.



- Model definition** – Specifies the model definition to which the shared object definition belongs. Objects can only be shared if they exist in the same model.
- Shared by class definitions** – Specifies the class definitions that have a sharing relationship with this SharedObjDef.

4.8 Activity – Creating Properties, Enumerated Lists and Relationships

Complete the Chapter 4 – Activity 1 in the SmartPlant Foundation 2008 (4.2) Modeling and Mapping Activity workbook.

5

C H A P T E R

Creating EdgeDefs, Graph Defs and View Defs

5. Creating EdgeDefs, Graph Defs, View Defs and Class View Maps

You will continue to add components to your data model in this chapter. This chapter will cover the object to object navigation functionality that is used by the SmartPlant Foundation Desktop client. For discussion purposes, we will refer to these object types as navigation objects.

The ability to navigate from one object in the model to another object in the model can be determined by **Edge** definitions. This allows you to specify which interface to start the navigation, and which direction to use to see information associated with a different interface.

Once these navigation edges have been defined, they can be grouped together to form a Directed **Graph** definition. These are a connected network of edge definitions.

Finally a **View** definition is a way of filtering lists of properties that will be available in the user interface.

These object types are **not required** in order to create data in the database. They will be used only by SmartPlant Foundation once they have been loaded to the SPF Admin database.

5.1 Edge Definitions

Edge definitions are used to combine multiple relationships into a single definition and have the following properties:

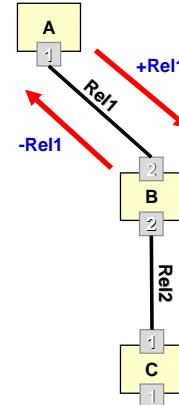
- A starting interface definition
- A path definition that specifies the relationships to traverse from the starting interface definition along related interface definitions and the direction to traverse those relationships (+ or -)



Edge Definitions

Edge definitions (EdgeDef) are single or multiple relationship definitions with direction. In the schema, an edge definition is used to traverse from a starting object to related objects.

Two implicit edges exist for every relationship in the schema. One that is positive (going from end 1 to end 2 of the relationship), and one that is negative (going from end 2 to end 1).



© 2005, Intergraph Corp.
All Rights Reserved.

- Discrimination criteria that can be applied to objects at the end of the path
- Position criteria that can select a specific object from the set of objects that satisfy the discrimination criteria at the end of the path

When you create an interface definition in the Schema Editor, the software automatically creates a corresponding edge definition. This edge definition is given a UID and name that is a combination of + and the UID for the interface definition. When you create a relationship definition in the Schema Editor, two edge definitions exist: one for traversing from end 1 of the relationship to end 2 and the other for traversing from end 2 to end 1.

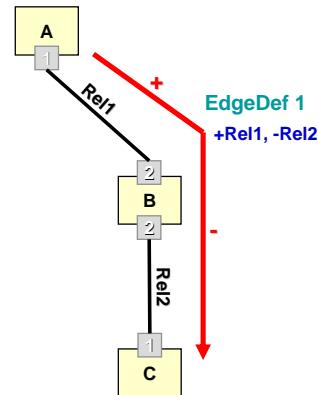
In addition to these automatically generated edge definitions (implicit EdgeDefs), you can explicitly define edge definitions for a variety of purposes in the Schema Editor. Explicit edge definitions may be used to traverse across multiple relationships, to select only certain objects from the destination end, or to do multiple relationship traversal and selective discrimination.



Edge Definitions

EdgeDefs collect one or more relationship edges.

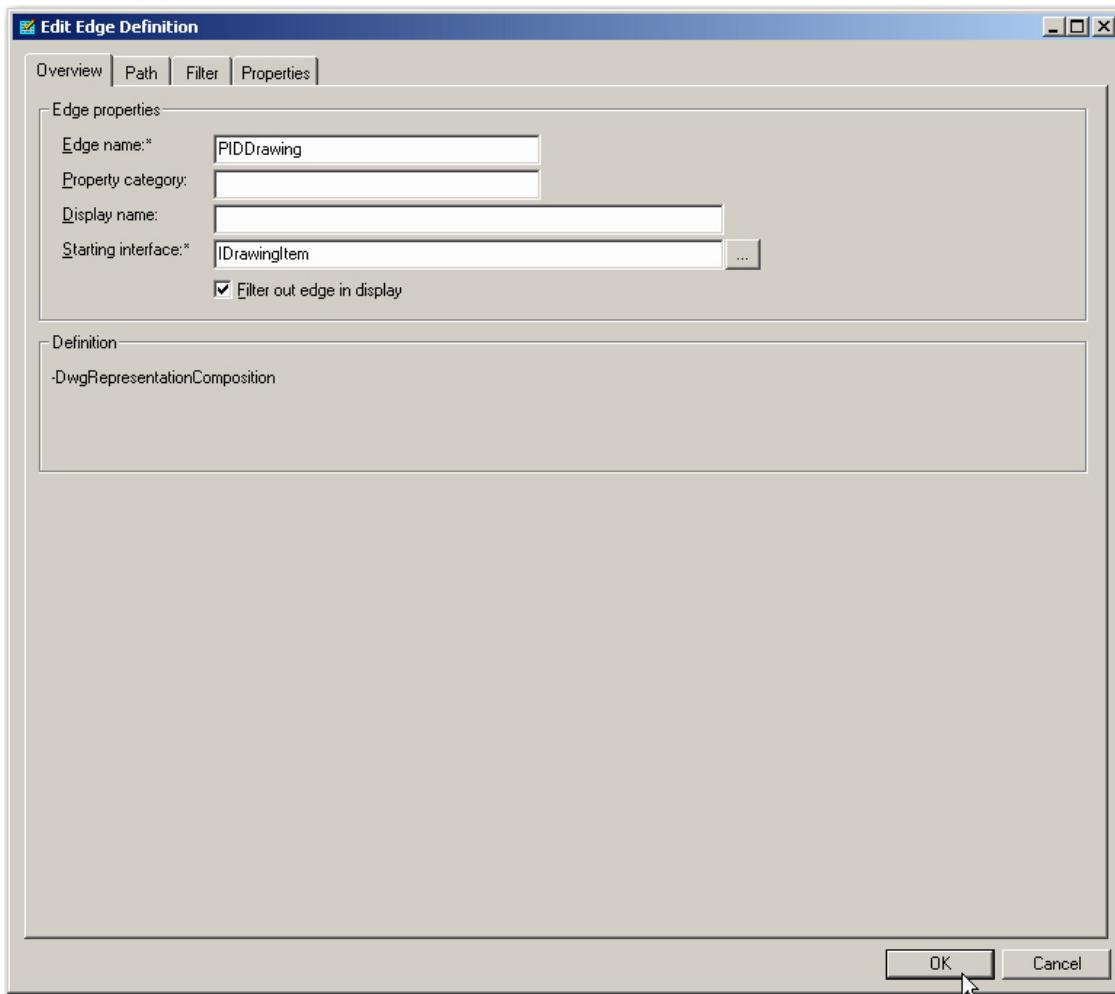
EdgeDefs can specify discrimination criteria, including position, to help you filter the objects at the end of the path defined by the EdgeDef.

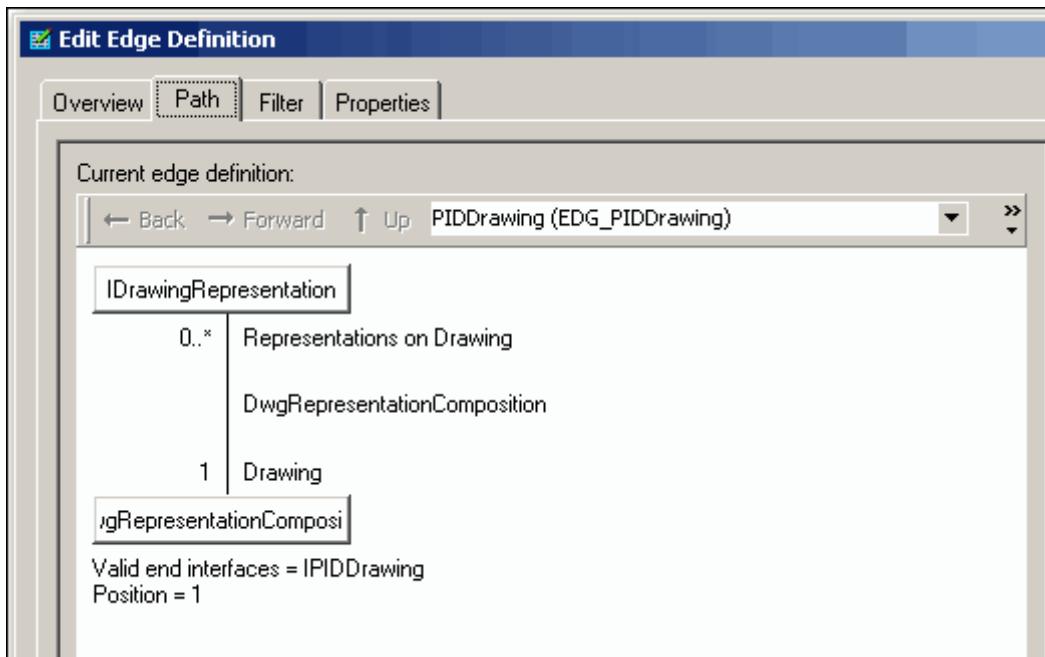


5.1.1 Properties of an Edge Definition

Edge definitions have the following properties:

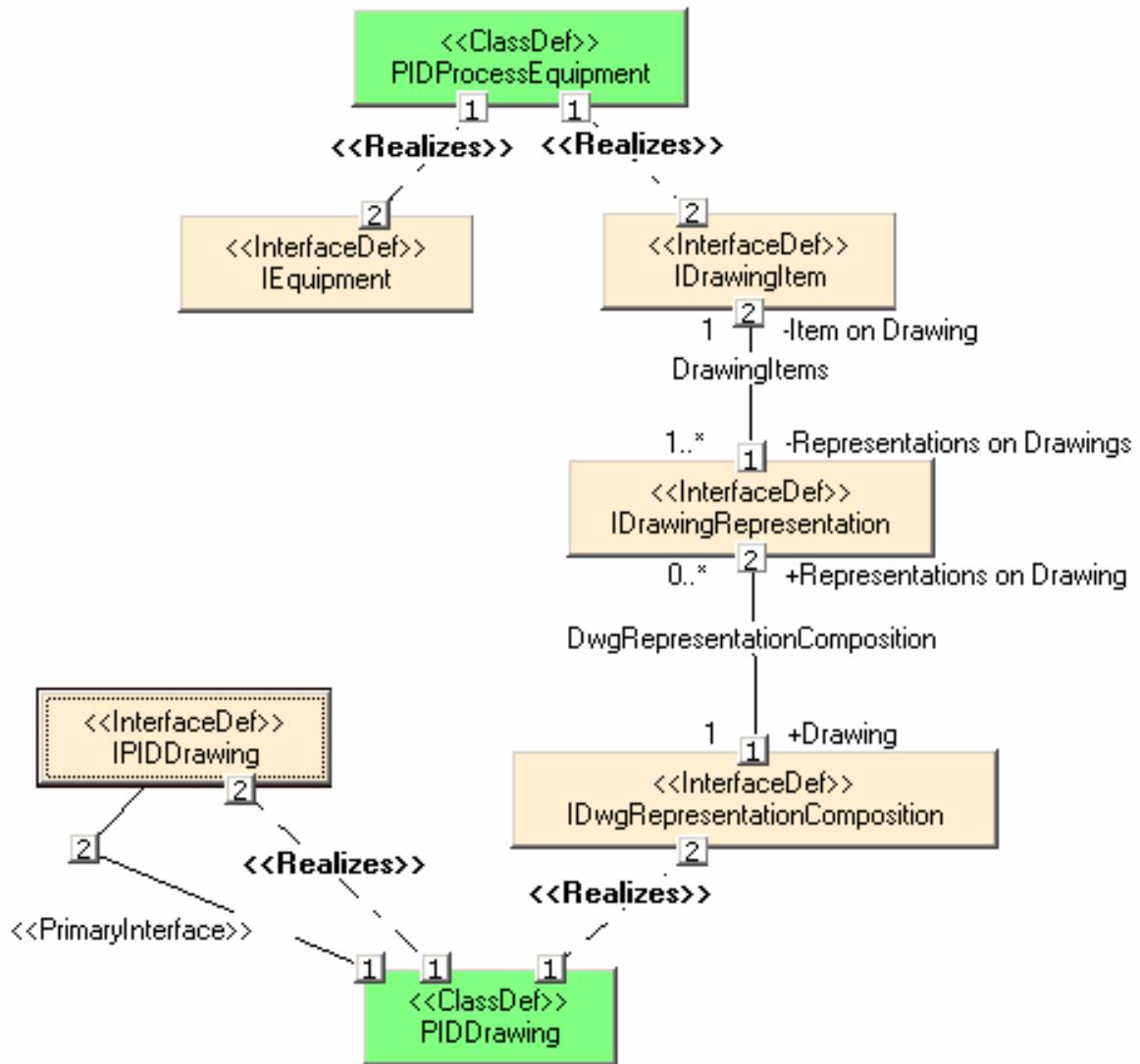
- ❑ **Edge name** – Specifies the name of the edge definition.
- ❑ **Property category** – Defines the property category for properties contained in this edge definition. The property category helps organize properties in the **Properties** window in SmartPlant Foundation and SmartPlant P&ID.
- ❑ **Display name** – Specifies the name that you want the user interface to use when displaying the edge definition.
- ❑ **Starting interface** – Allows you to select the starting interface when you create a new edge definition.
- ❑ **Filter out edge in display** – Specifies whether you want to see the edge definition in the Schema Editor user interface. If this option is checked (4) which is True, user access for this edge definition can be controlled in the SPF client by user group security.





- ❑ **Path** – Allows you to select the relationship edges that you want to include in the edge definition. The set of edge definitions that appear here are the set of interface definitions that apply to the end interface definition. As you click nodes in the tree, the path for the edge definition updates beside the **Path** label at the bottom of the dialog box. The UI allows you to include interface definitions that are implied by the end interface definition in your edge definition. Every edge definition starts at an interface definition and traverses through one or more relationship definitions to another interface definition at the other end. When you define a path for an edge definition, the set of edge definitions that appear in the **Current edge definition** window are the set of interface definitions that apply to the end interface definition. For example, if you traverse the relationship definition from equipment to equipment components, you will see the edge definitions that apply to the role of equipment components. However, if you want to continue the path using nozzles, which are a specialization of equipment components, then you can select INozzleOcc from this list. Selecting INozzle for example, will allow you to continue the path using the more specific nozzle edge definitions.

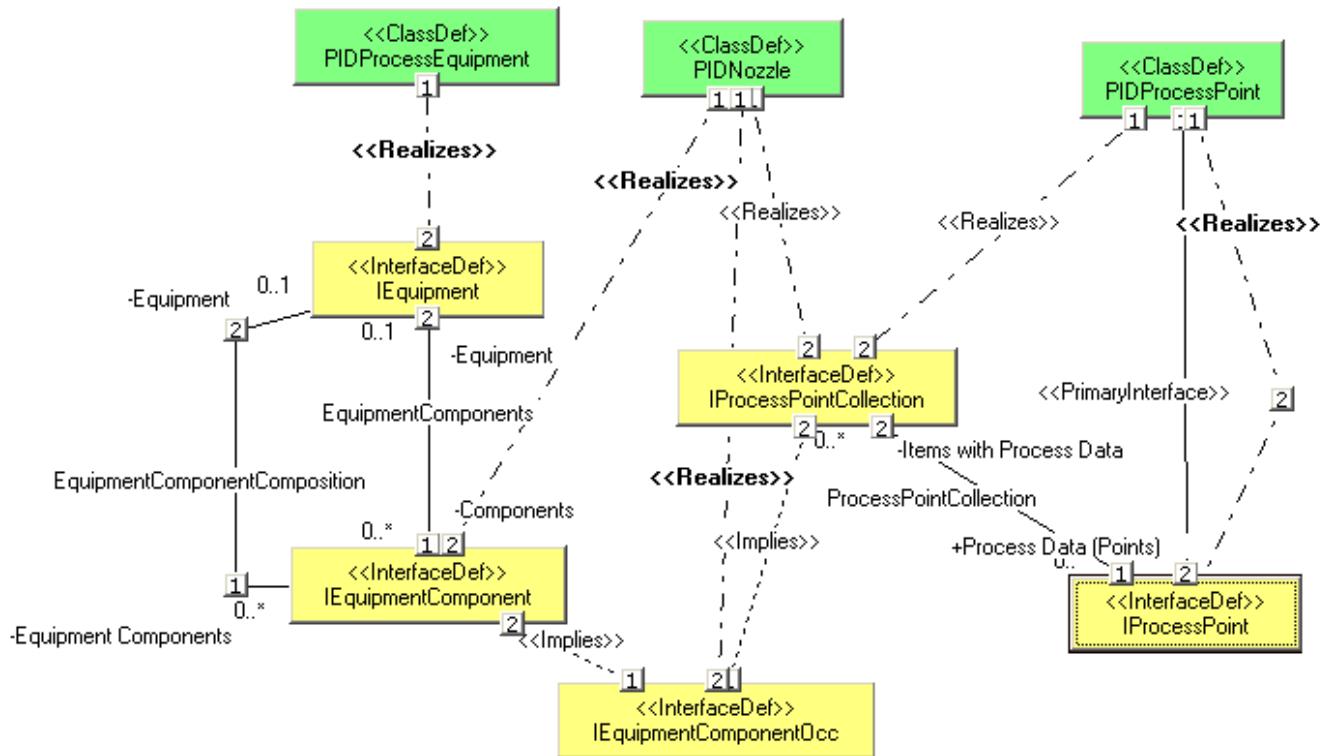
The following UML diagram shows an EdgeDef that navigates from a piece of equipment to the P&ID on which the piece of equipment appears:



In SmartPlant Foundation, edge definitions are custom relationship expansions that appear on the shortcut menu when a user right-clicks an object that exposes the relationship referenced by the edge definition. Instead of requiring the user to go directly from Object A to Object B in the tree view, an edge definition can be created to allow the user to traverse through multiple intermediate objects and display an object at the end of the relationship that meets specific criteria. For example, an edge definition could be created to expand to only the first nozzle on a vessel.

System administrators might also create an edge definition to traverse directly from a plant to a unit. In SmartPlant Foundation, units are directly related to functional areas. However, a custom edge definition would allow the user to expand the plant to see the units without expanding a functional area first.

The following UML diagram can be used to show what an edge definition might look like:



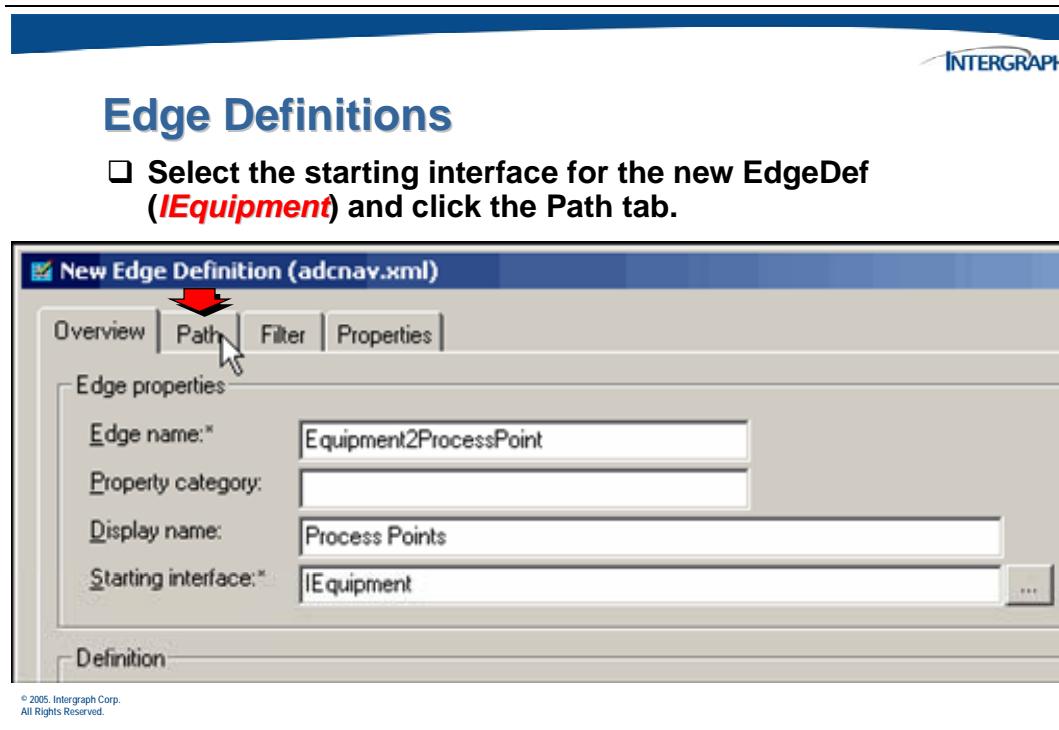
The next few pages provide an example of an Edge Definition.

The first step is to access the **New Edge Definition** form, this can be done in a number of ways, including dragging and dropping the **EdgeDef** option from the **Create** section of the **Editor** view into the UML window.

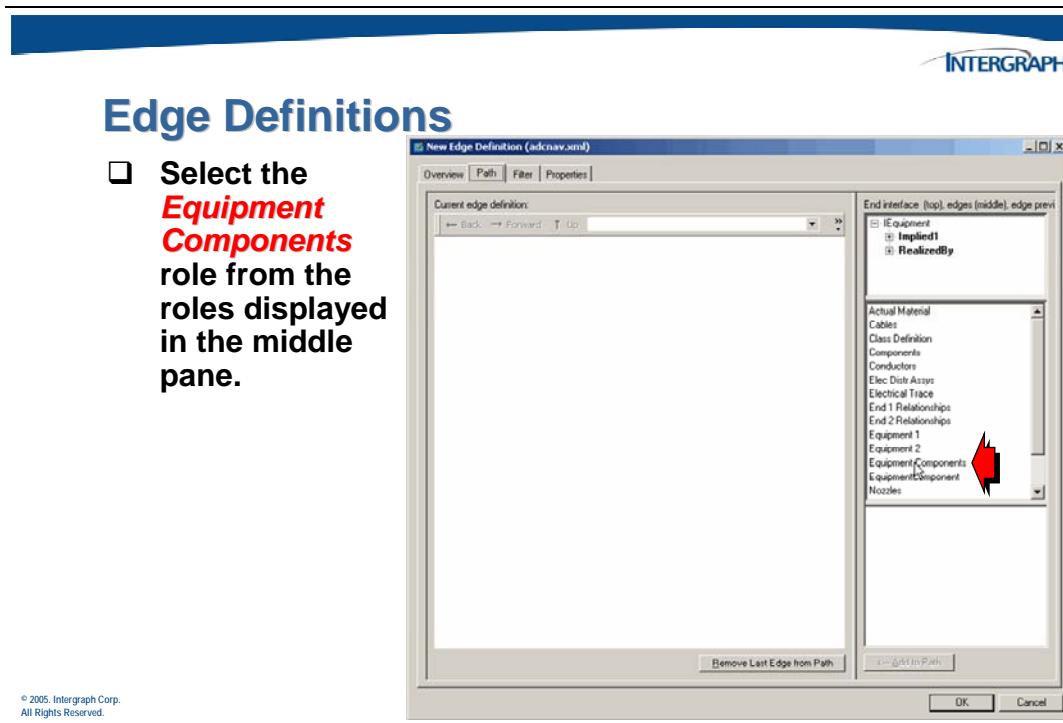
On the form, provide a name for the edge and a display name to describe how the edge will appear on an object's shortcut menu.

Choose the starting interface which will represent objects at the starting point.

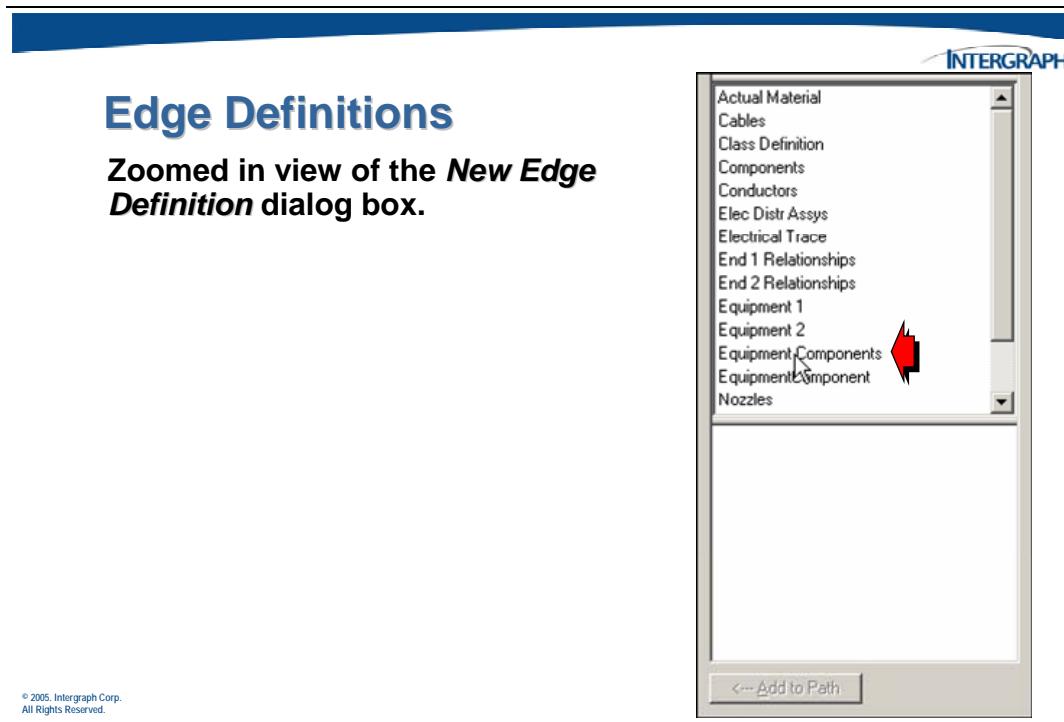
The **Filter out edge in display** check box (not shown below) should be selected if you want to put security on this command on the shortcut menu.



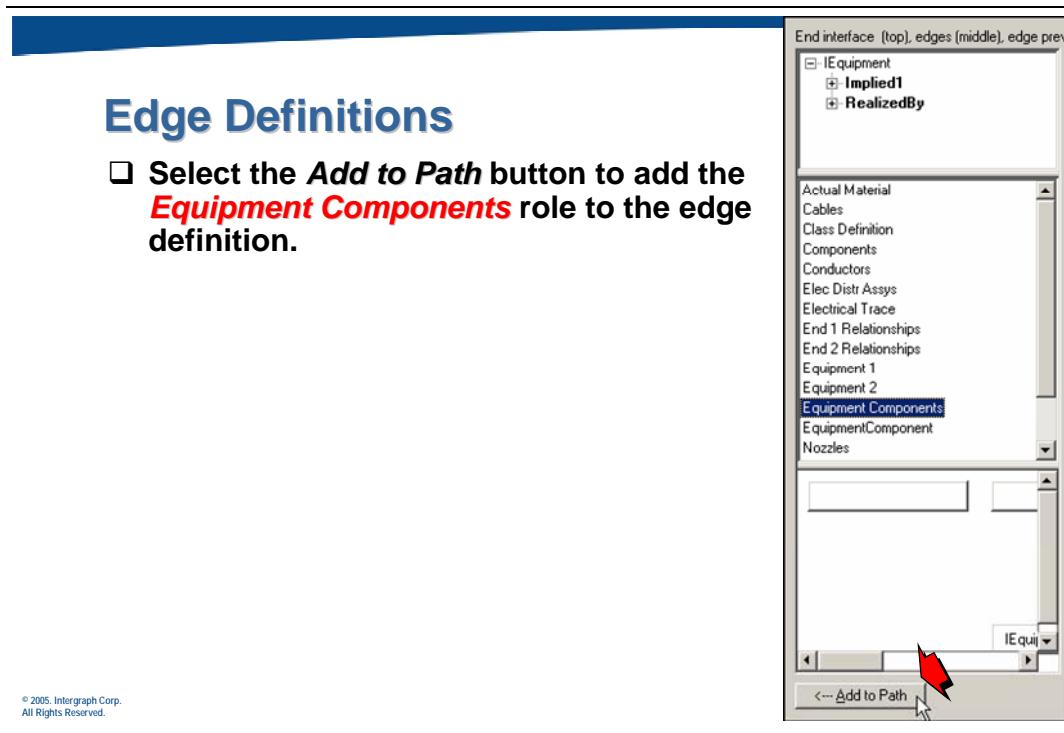
From the **Path** tab, use the right-hand side of the window to determine a path. The top pane shows you the current positions (in this case the starting point) and allows you to navigate implies relationships. The middle pane shows the roles on the end of reldefs available to the interface where you are currently located, and the bottom pane will display the relationships used to navigate to the roles selected in the middle pane.



Below is a closer look at the roles displayed in the middle pane.



Click on the *Equipment Components* role in the middle pane, and click *Add to Path*.



The path to the selected role appears in the large window on the left.

The screenshot shows the "Edge Definitions" dialog box. On the left, under "Current edge definition:", there is a tree view of relationships:

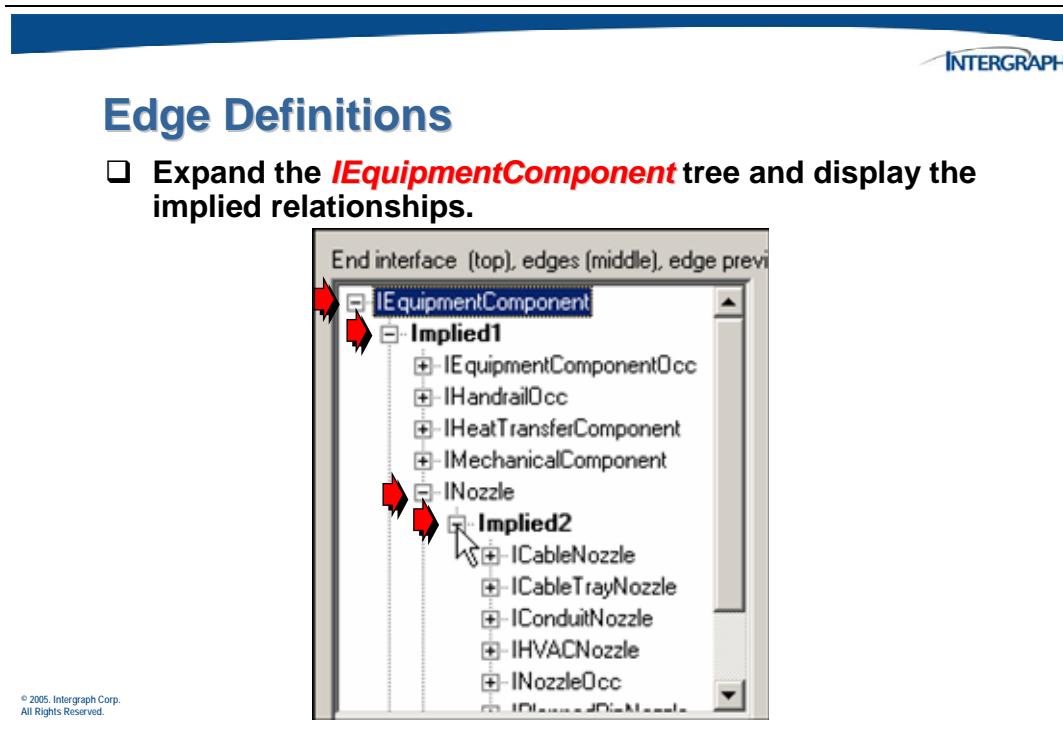
- IEquipment (Root node)
- 0..1 Equipment
- EquipmentComponentComposition
- 0..* Equipment Components
- IEquipmentComponent (Leaf node)

A red arrow points from the "Equipment Components" node towards the right pane. On the right, under "End interface (top), edges (middle), edge previ", there is a list of roles:

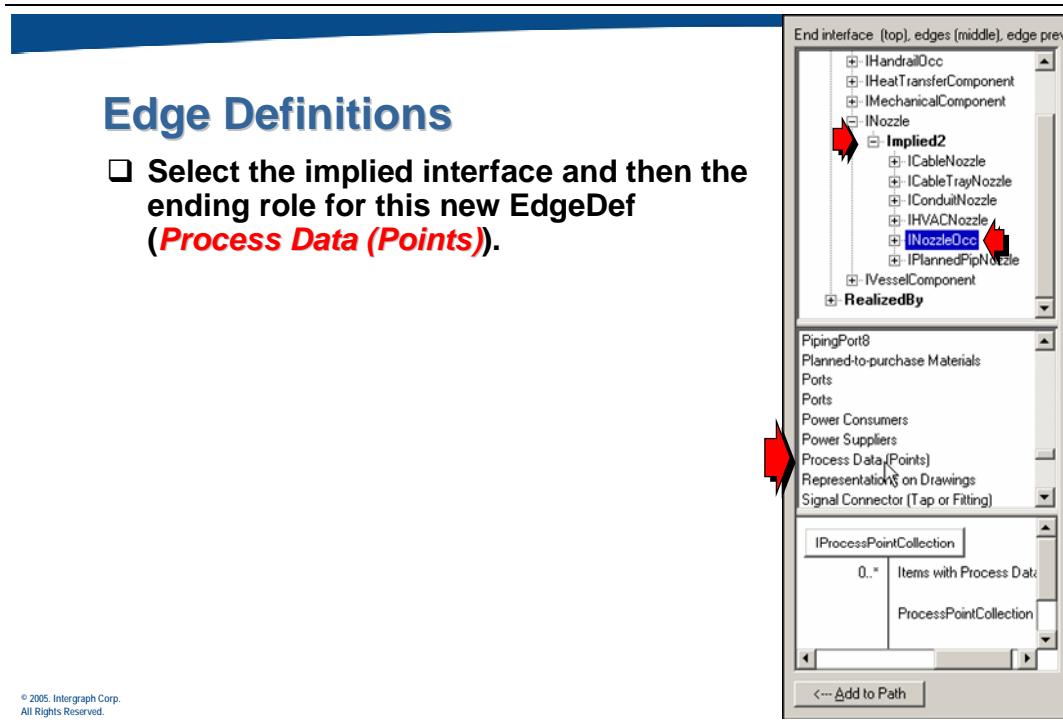
- IEquipmentComponent
 - Implied1
 - RealizedBy
- Actual Material
- Cables
- Class Definition
- Component Assembly** (This item is highlighted in blue.)
- Component Assembly
- Components of Assembly
- Components of Assembly
- Conductors
- Elec Distr Assys
- Electrical Trace
- End 1 Relationships
- End 2 Relationships
- Equipment
- Equipment

At the bottom left of the dialog box, it says "© 2005 All Rights Reserved".

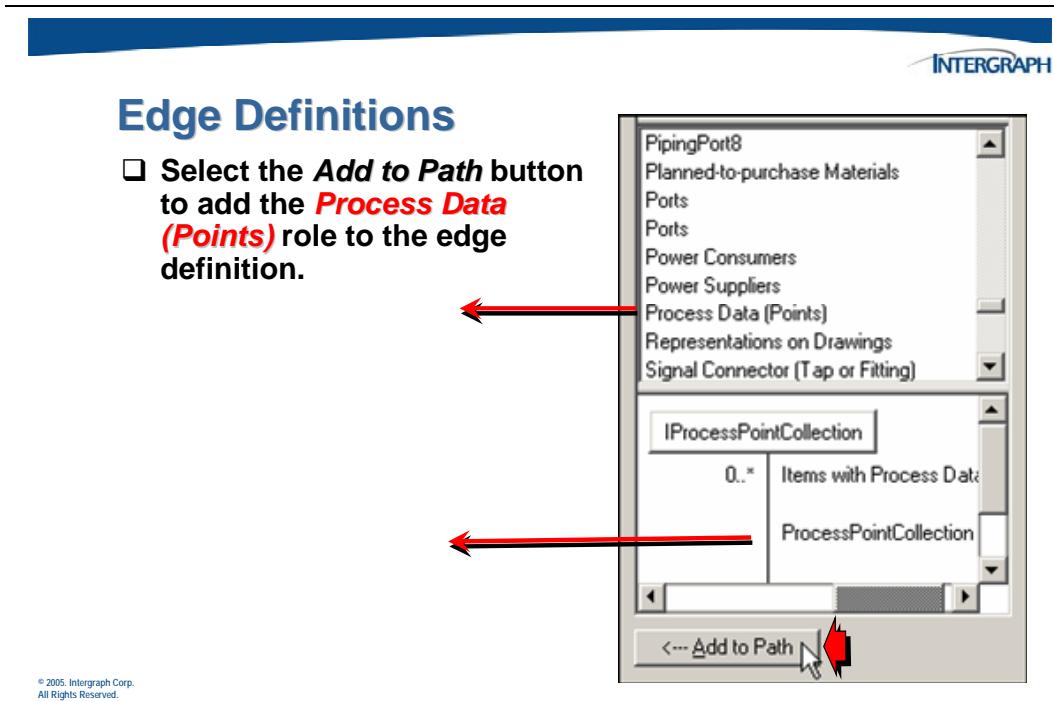
The top right pane will now show the current location (the interface on the other end of the reldef displayed in the path window). You may also navigate using implies relationships, as illustrated below.



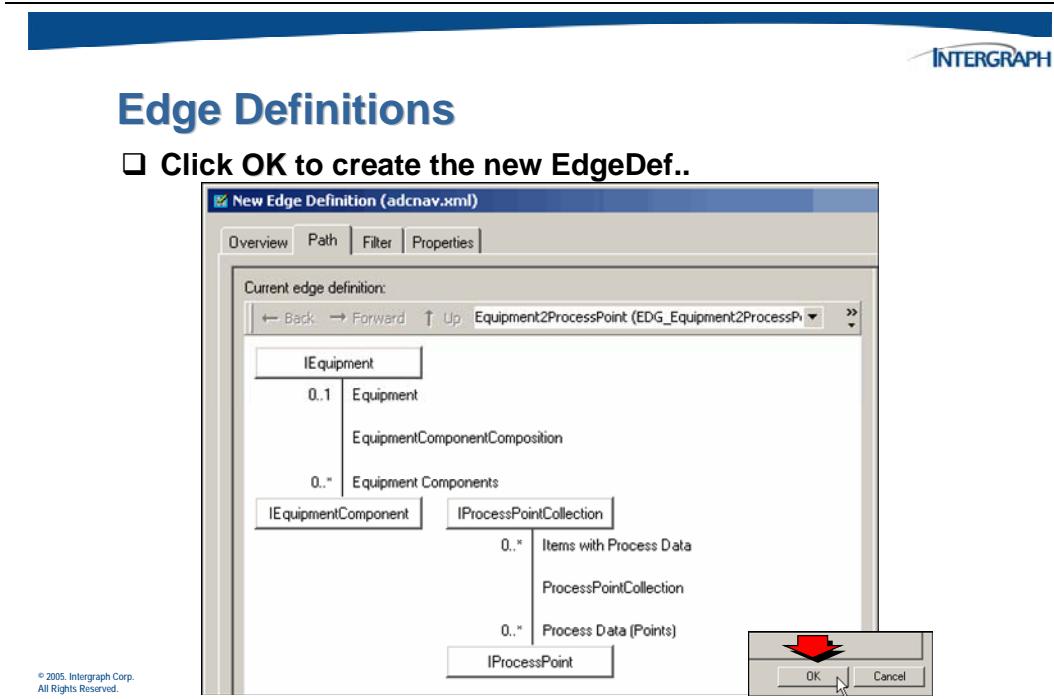
Drill down in the tree to locate the *INozzleOcc* implied relationship.



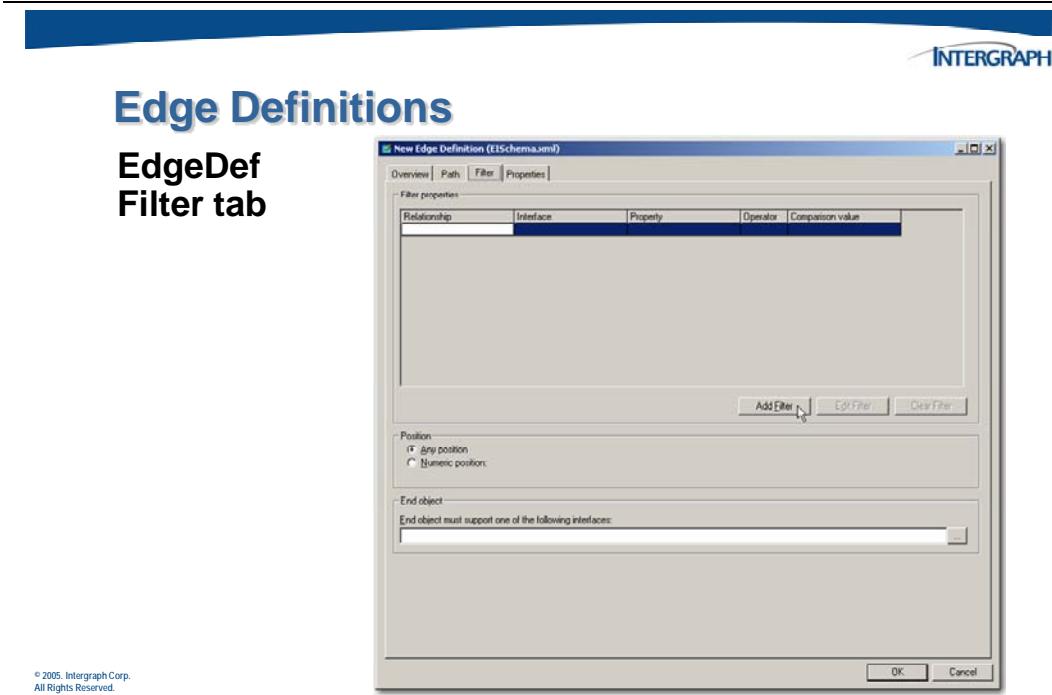
The middle pane now shows roles at the other end of reldefs from the interface selected in the top pane. Choose the role you want, and click the **Add to Path** button.



The path appears on the left. Gaps in the path represent navigation through implies relationships.



You can also define discrimination criteria for edge definitions using the options on the **Filter** tab.



- ❑ **Property** – Allows you to select properties exposed by the starting interface to use as criteria to limit the objects at the end of the path for the edge definition.
- ❑ **Operator** – Specifies whether you want objects at the end of the path for the EdgeDef to be equal to (=), less than (<), greater than (>), or not equal to (<>) the value in the **Comparison value** box.
- ❑ **Comparison value** – Specifies the value that you want to use with the specified comparator to limit objects at the end of the edge definition. If you select a property in the **Property** list that has an enumerated list associated with it, you can click the ? button here to select a value from the enumerated list to use as the comparison value.
- ❑ **Any position** – Specifies that objects at the end of the path for the edge definition can be in any position.
- ❑ **Numeric position** – Specifies that objects at the end of the path for the edge definition must be in a particular numeric position. For example, if you want to return the fifth nozzle on a piece of equipment, the numeric position would be 5.
- ❑ **End object must support the following interfaces** – Allows you to further define the objects at the end of the path for the edge definition by specifying an interface that end objects must support. For example, if you want to create an EdgeDef to display the P&ID an object is on, you could select IPIDDrawing in this list to make sure that only P&IDs and not other objects that participate in the DwgRepresentationCompostion relationship are returned by the EdgeDef.

5.2 Interactive Activity – Creating Edge Definitions

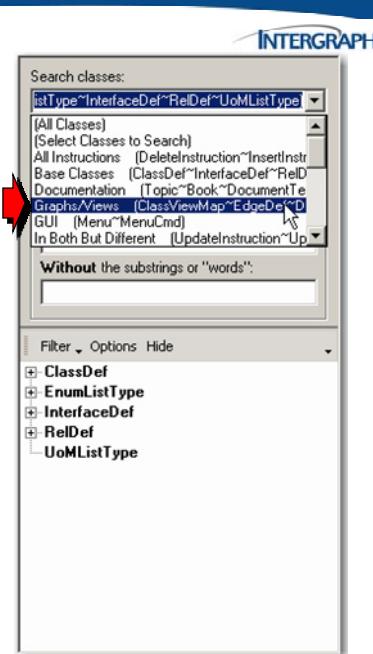
The following interactive activities guide you through the process of creating your own edge definitions for use with the schema objects created in the previous activities.

The first step is to re-filter the tree view to display the objects we will be creating.

1. From the *Search classes* field on the left side of the window, find the *Graphs/Views* object.

Creating Edge Definitions

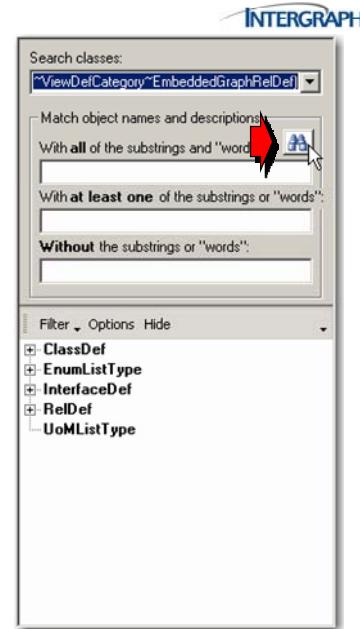
- In order to see the new edge definitions we will create, we need to modify the filter on the tree view.
- From the *Search classes* field, choose the *Graphs/Views* option.



2. With the new selection made, click the **Find** button below the field to refresh the view of the tree.

Creating Edge Definitions

- Once you have changed the selection, click the find button to refresh the tree display.

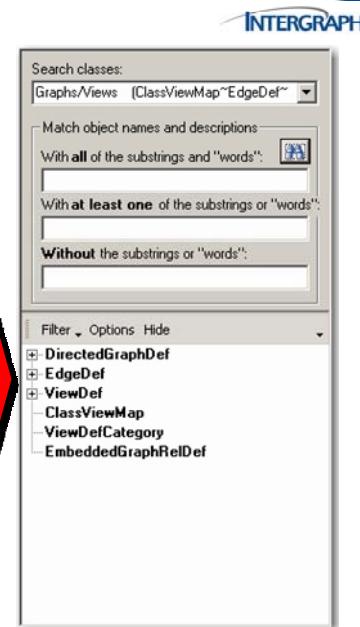


© 2005, Intergraph Corp.
All Rights Reserved.

The tree now displays DirectedGraphDefs, EdgeDefs, and ViewDefs.

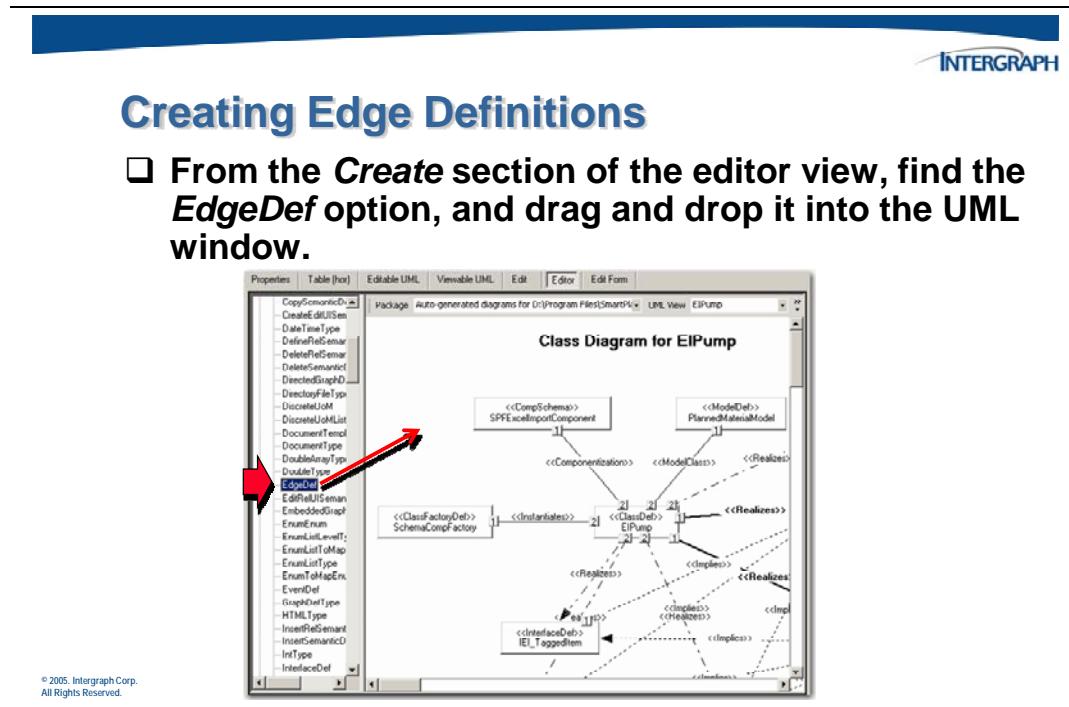
Creating Edge Definitions

- The tree will now display the directed graph defs, edge defs, and view defs that are part of the current configuration file.



© 2005, Intergraph Corp.
All Rights Reserved.

3. Find the **EdgeDef** option in the **Create** list, and drag and drop it into the UML window.



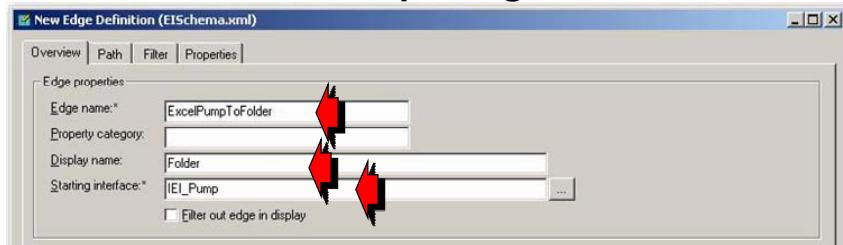
4. Provide the following information for the new edge def:

- Edge name** – ExcelPumpToFolder
- Display name** – Folder
- Starting interface** – IEI_Pump

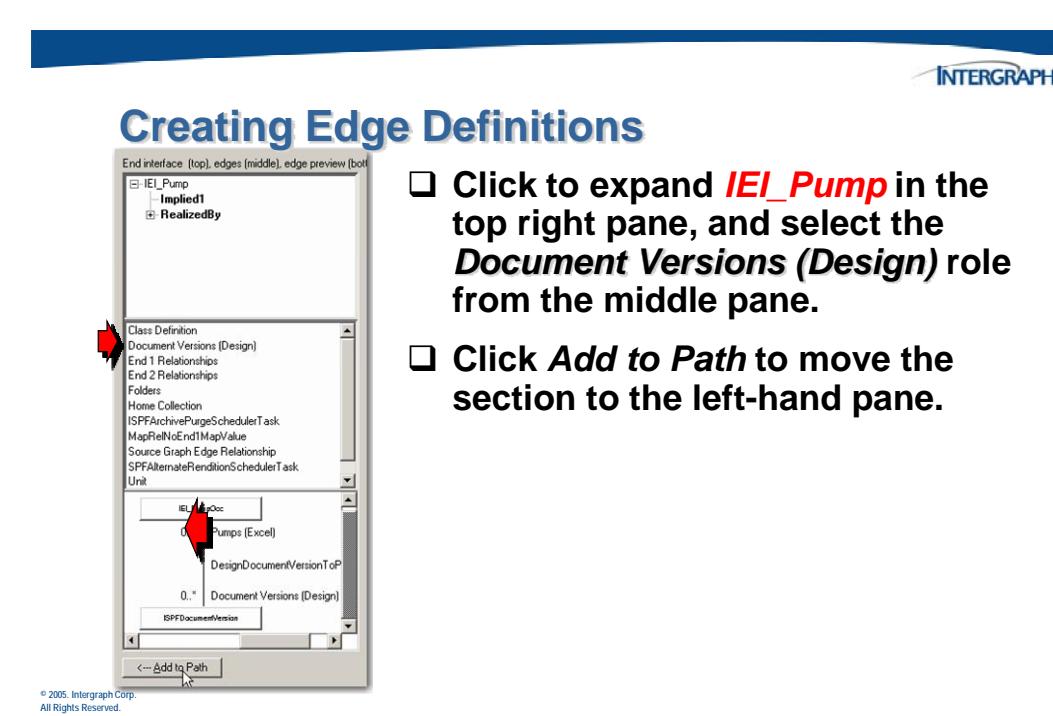
5. When these fields have been created, go to the **Path** tab.

Creating Edge Definitions

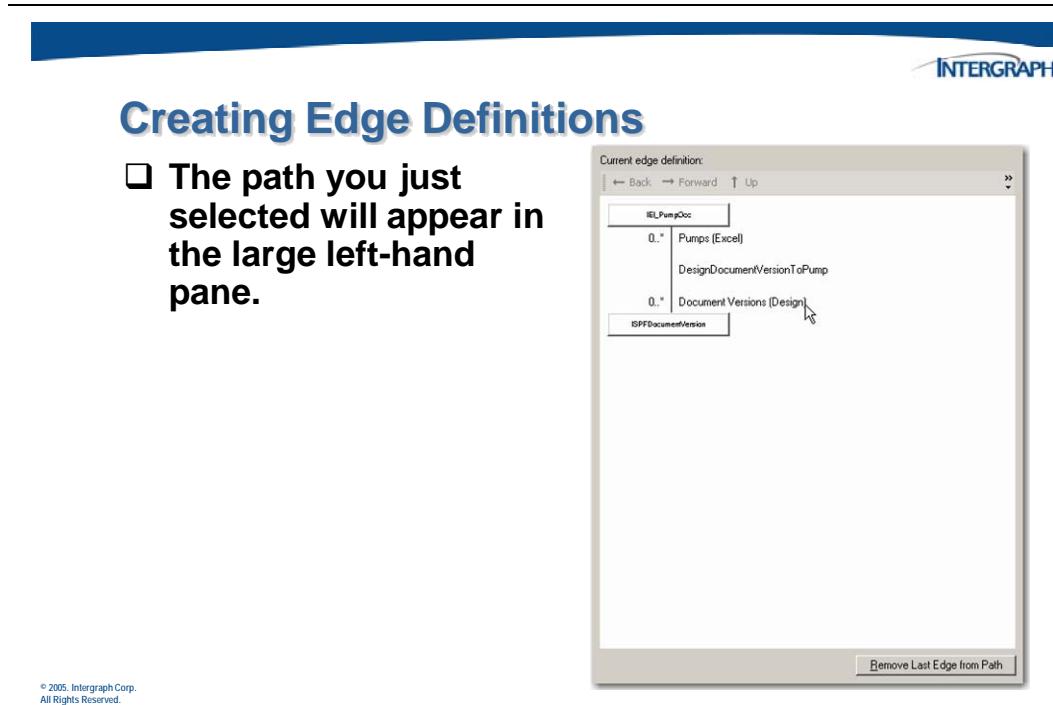
- Enter an **Edge name** and **Display name**, and then click the **Starting interface** browse button. When this information is complete, go to the **Path** tab.



6. In the middle pane, find the ***Document Versions (Design)*** role. Select it, and click the ***Add to Path*** button.



The path will appear in the large pane on the left-hand side of the window.



7. From the middle pane, find the **Folders** role. Select it, and click the **Add to Path** button.

The screenshot shows the 'Creating Edge Definitions' interface. In the top left, there's a tree view with nodes like 'ISPFDocumentVersion', 'Implied1', and 'RealizedBy'. The main pane displays a relationship diagram with entities like 'ISPFDocumentVersion', 'ISPFObject', 'SPFFolderObject', and 'ISPFFolder'. A red arrow points to the 'Folders' entity. At the bottom right of the main pane is a button labeled '<-- Add to Path'. The bottom left of the interface has a copyright notice: '© 2005, Intergraph Corp. All Rights Reserved.'

Next, choose the **Folders** role from the middle pane.

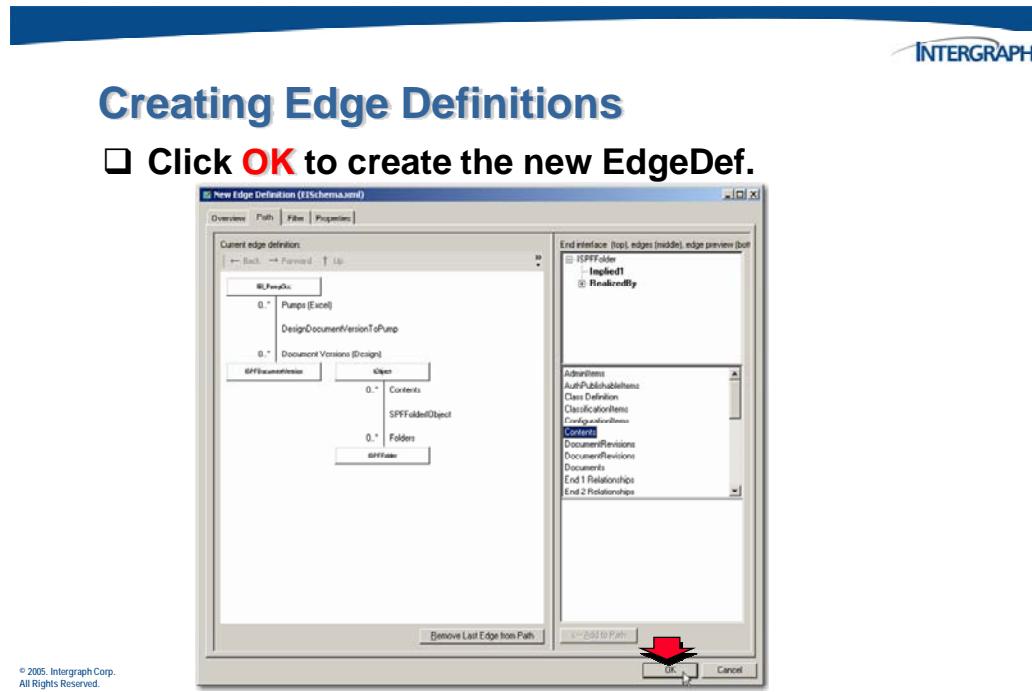
The second step of the path will also appear in the window on the right.

The screenshot shows the 'Creating Edge Definitions' interface. The top part is identical to the previous screenshot. The bottom part shows a larger pane on the right containing a detailed relationship diagram. This diagram includes entities like 'ISPLPumpDoc', 'ISPFDocumentVersion', 'ISPFObject', 'SPFFolderObject', and 'ISPFFolder'. It also shows relationships such as 'Pumps (Excel)' and 'DesignDocumentVersionToPump'. At the bottom right of this pane is a button labeled 'Remove Last Edge from Path'. The bottom left of the interface has a copyright notice: '© 2005, Intergraph Corp. All Rights Reserved.'

The second section of the path will appear in the large left-hand pane.

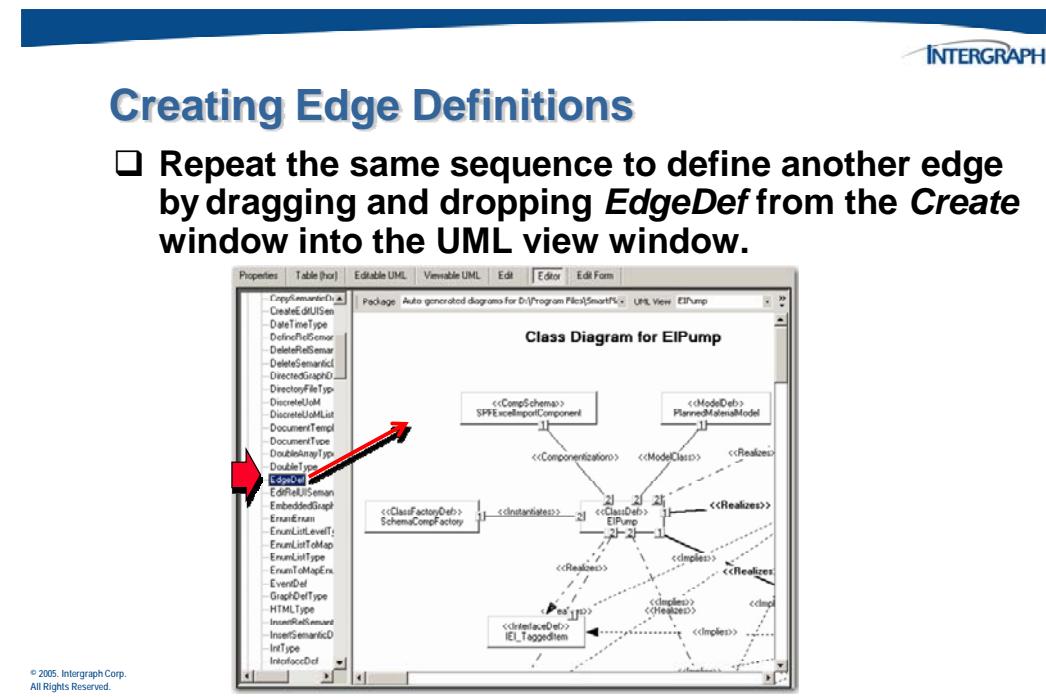
Sections do not appear connected are actually paths using implies and implied by relationships.

-
8. Click **OK** to create the new edge definition.



Next, we will create another edge definition. This edge def, however, will include a filter to display only objects on the end that meet a specified criteria

9. Create another edge def by dragging and dropping the *EdgeDef* from the *Create* section of the *Editor* view into the UML view.



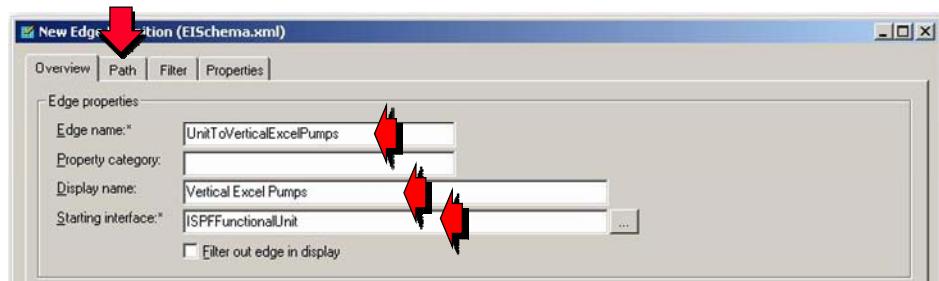
10. Create a new edge def with the following information:

- Edge name** – UnitToVerticalExcelPumps
- Display name** – Vertical Excel Pumps
- Starting interface** – ISPFFunctionalUnit

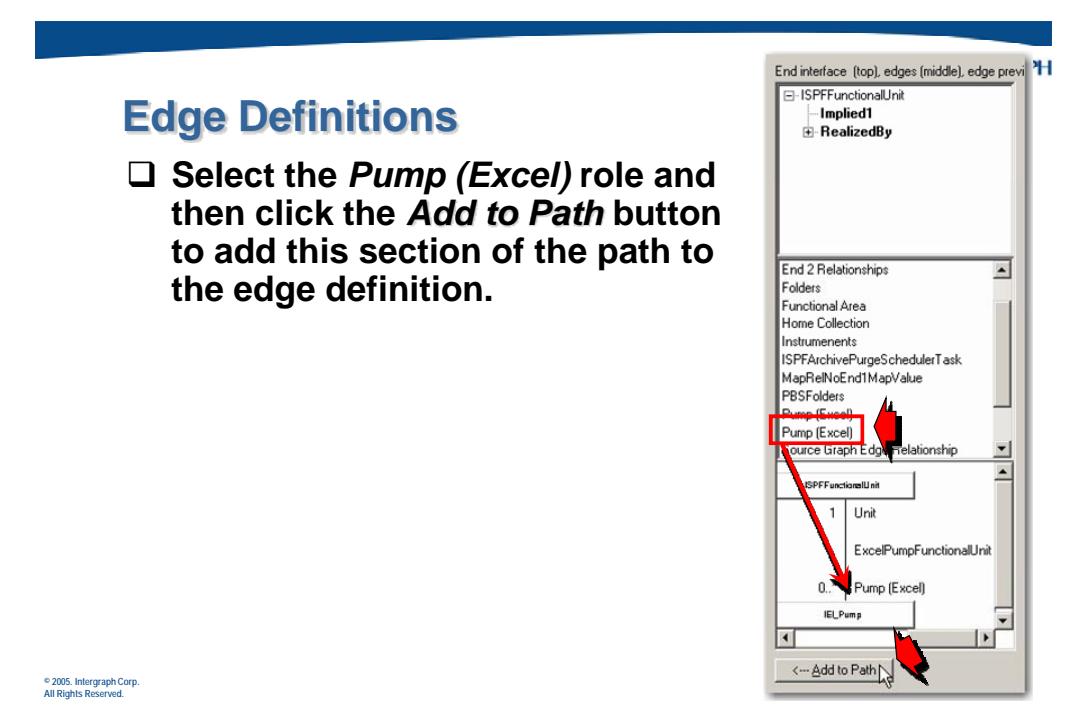
11. When you have provided the required information, go to the **Path** tab.

Creating Edge Definitions

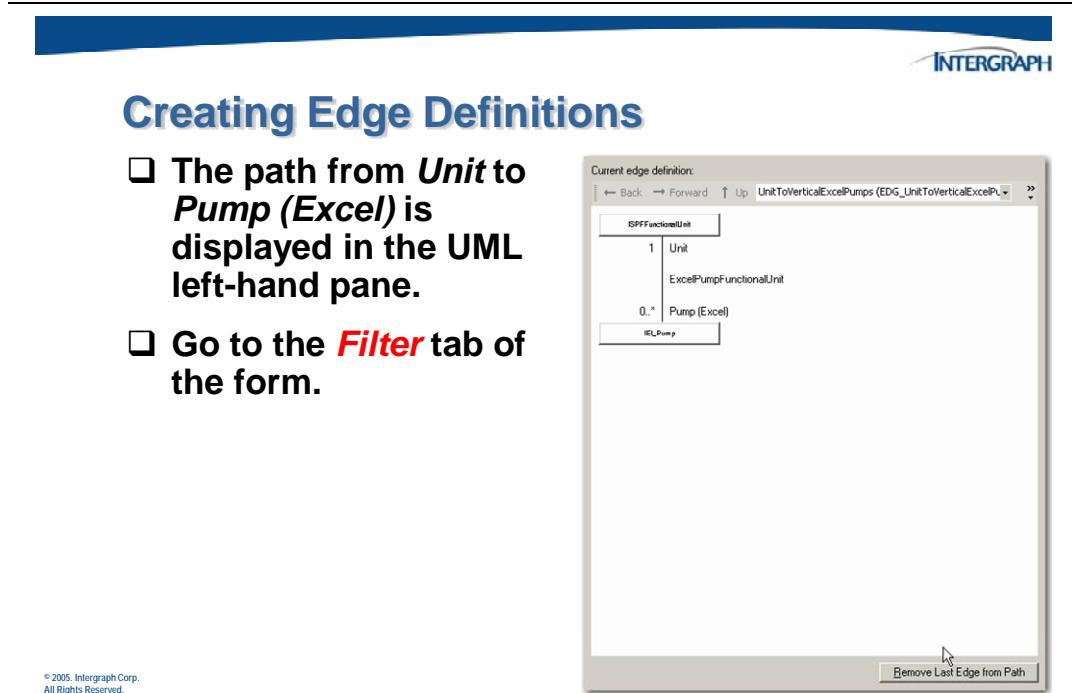
- Enter a name, display name, and starting interface for the next edge. Then select the **Path** tab.



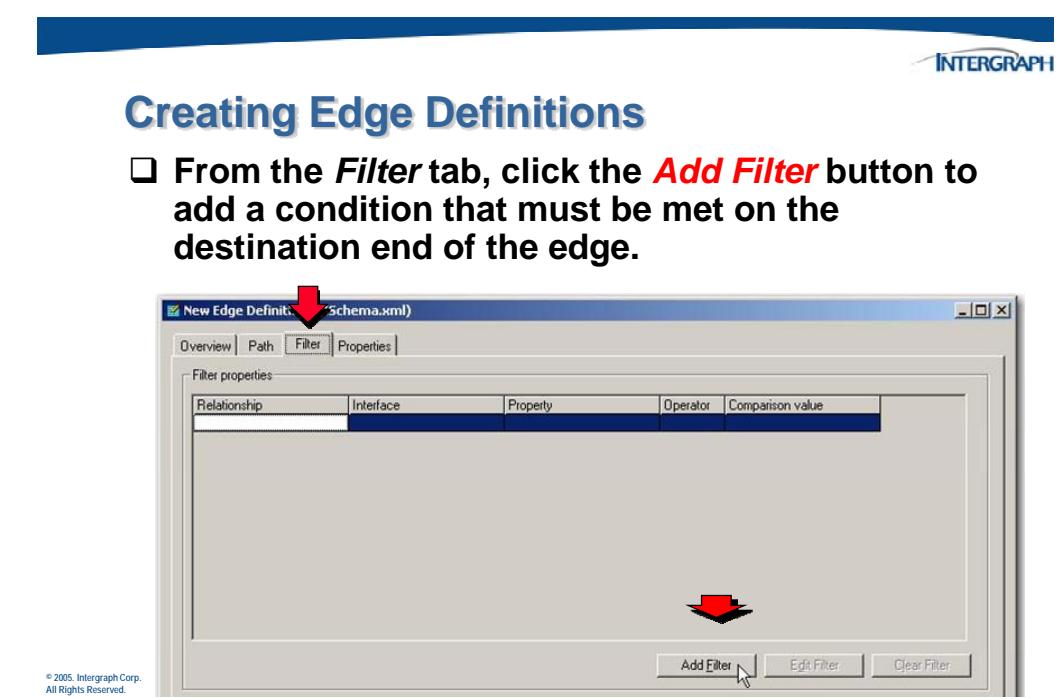
12. From the list of roles in the middle pane, find **Pump (Excel)**. Select it, and click the **Add to Path** button.



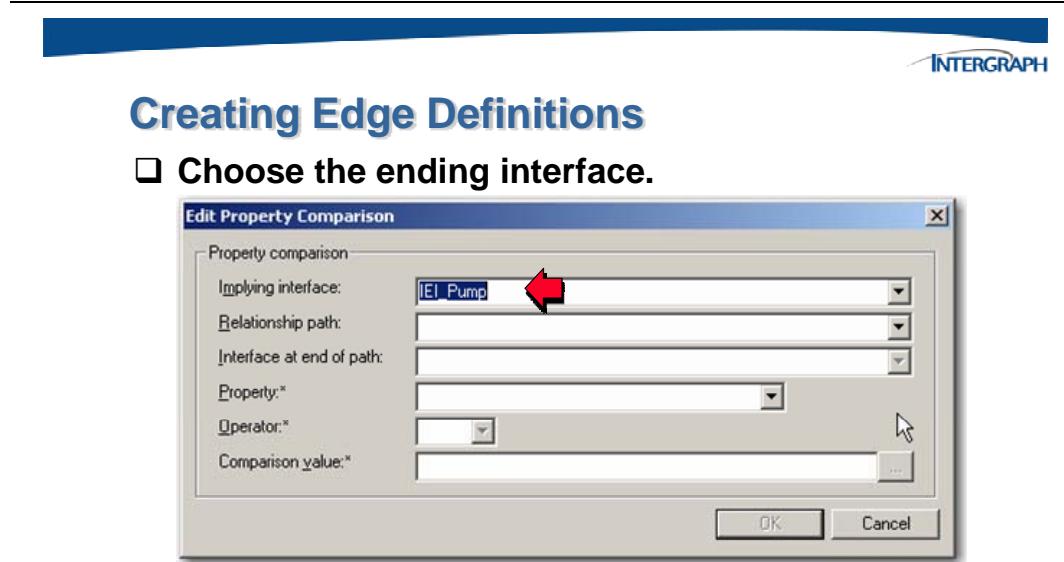
13. The path will appear in the large window on the left-hand side of the form. Next, go to the **Filter** tab.



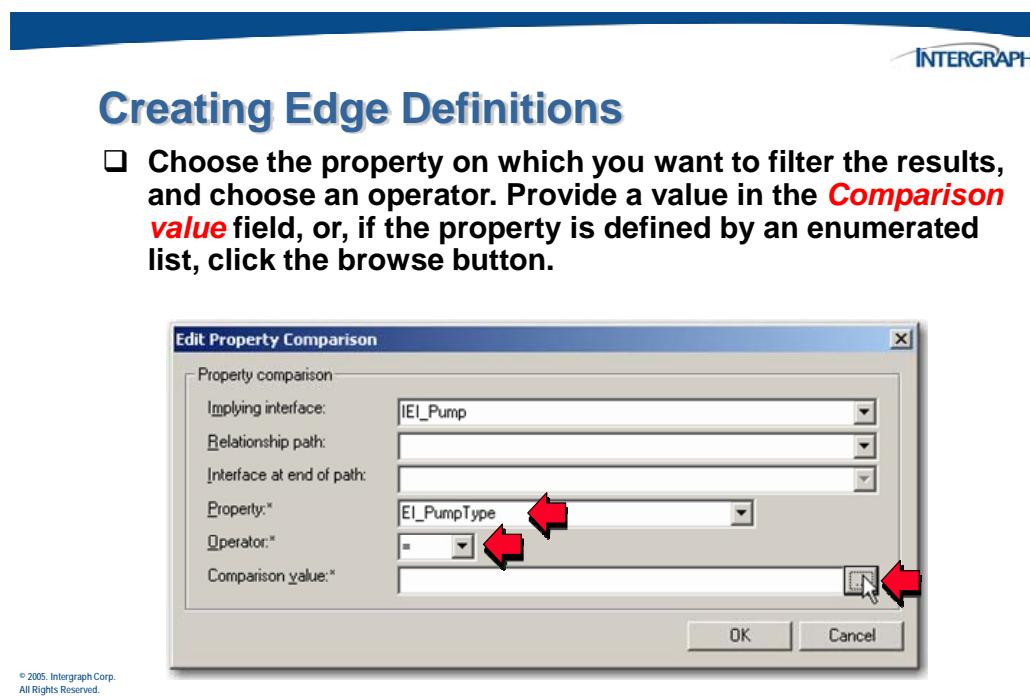
14. From the **Filter** tab, we will specify a criteria that pumps (the end point) must meet in order to be returned by this edge def. Click the **Add Filter** button.



15. Choose **IEI_Pump** as the starting interface. Remember, IEI_Pump is the primary interface for our EIPump class def.



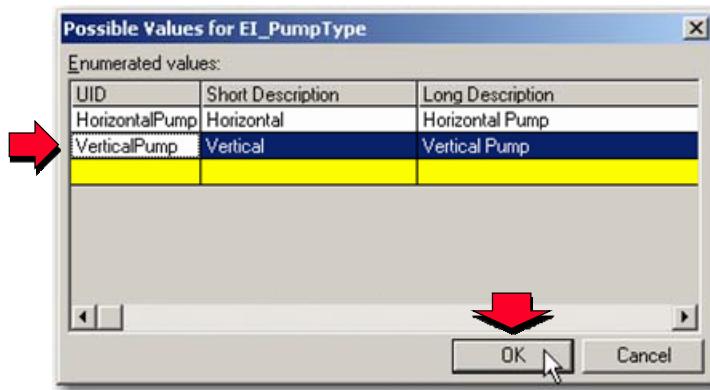
16. Specify the property on which you want to filter. We will filter on the type of the pump, so choose the ***EI_PumpType*** property def.
 17. Choose the ***Operator*** to be used in the criteria, and then click the browse button beside the ***Comparison value*** field.
-



18. Choose the value that must be set for the ***EI_PumpType*** property of a pump in order for it to be returned by the edge definition. As our example is for a command to return vertical pumps, choose the **Vertical** option, and click **OK**.

Creating Edge Definitions

- Choose the applicable enum enum from the table of enumerated list values, and then click **OK**.

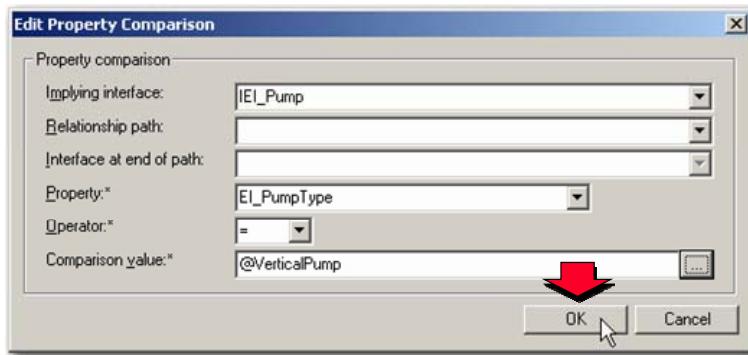


© 2005, Intergraph Corp.
All Rights Reserved.

19. When you have finished creating the criteria for the edge definition, click **OK**.

Creating Edge Definitions

- Review the filtering criteria to be placed in the EdgeDef, and then click **OK**.



© 2005, Intergraph Corp.
All Rights Reserved.

20. Click **OK** to finish creating the new edge definition.

5.3 Graph Definitions

Graph definitions (GraphDef), also called Directed Graph Definitions in the schema, are a connected network of edge definitions starting at a particular interface definition.

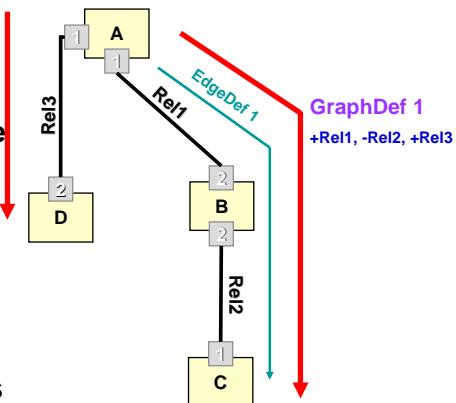
Graph Definitions

Graph definitions (GraphDef) are a set of edge definitions with structure.

Each GraphDef starts at an interface definition and branches out from that interface definition to related interface definitions.

Any edge that is tied to the starting interface definition or to any interface definition implied by that interface definition can be used as part of the GraphDef.

It is not necessary to create EdgeDefs for GraphDefs unless you need to apply criteria.



© 2005, Intergraph Corp.
All Rights Reserved.

Graph definitions include the following:

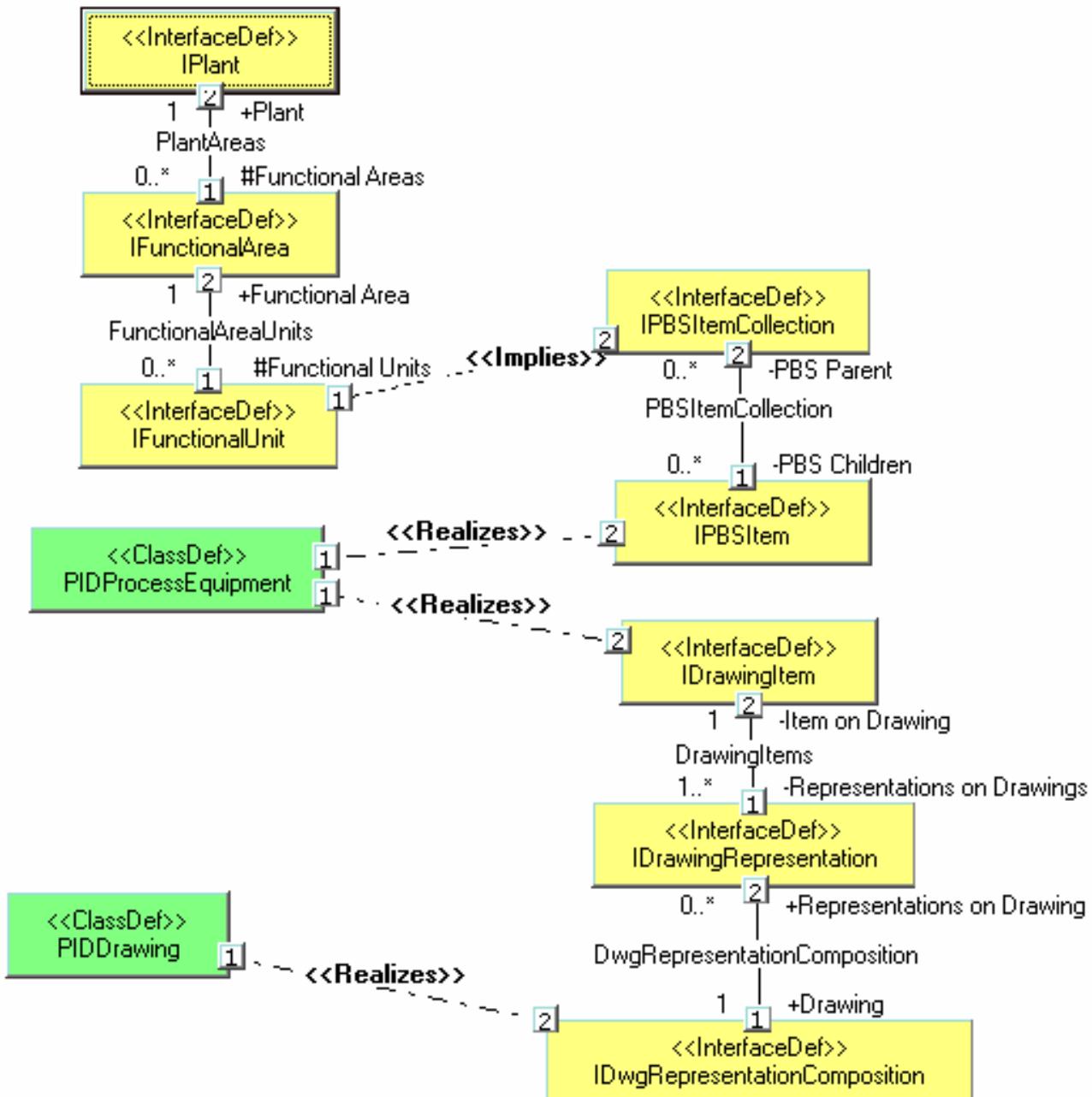
- A relationship to its starting interface definition
- A definition of the directed graph

When you create a GraphDef, explicit EdgeDefs are not required unless you need to apply criteria. One example is the Equipment to Nozzle relationship. You must turn the one-to-many relation into many one-to-one relationships using the position criteria. Another example is the Equipment to Drawing relationship. If you want to put include the P&ID in the GraphDef, you need to navigate that relationship, but you must apply criteria to get only P&IDs instead of all drawings.

In SmartPlant Foundation, graph definitions are used to define default expansions and alternate expansions for class definitions. They are also used to define datasheets for SmartPlant. If the name of a class definition is the same as the name of a graph definition, the graph definition is automatically used to define the default expansion for the class definition. For example, when you expand an object in the tree view, the graph definition with the same name as the class to which the object belongs defines the default expansion that you see.

You can also define alternate expansions for class definitions. For alternate expansions to show up on the shortcut menu when you right-click an object in the SmartPlant Foundation client, you must create a new method that uses the GetObjsbyGraphDefName client API and references the appropriate graph definition. Default expansions created using graph definitions show up in the method section of the shortcut menu that appears when you right-click an object that exposes the starting interface defined in the graph definition.

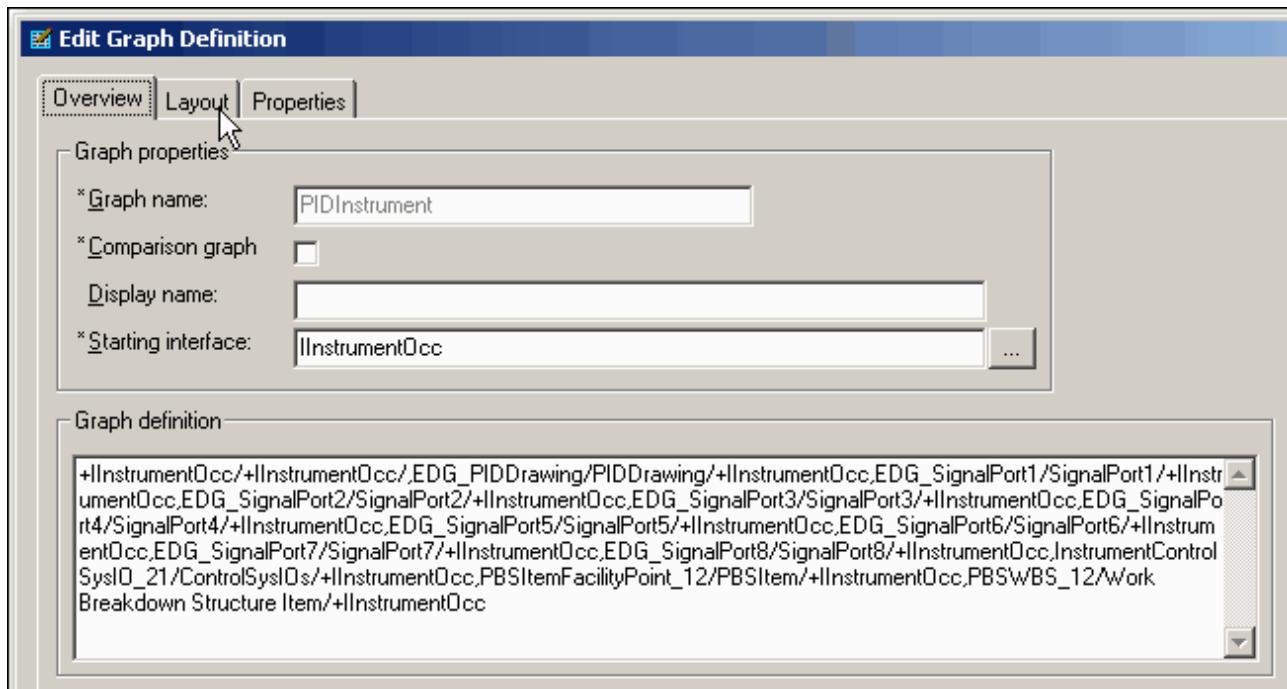
The following UML diagram shows a GraphDef that goes through the plant, area, and unit to a piece of equipment and then to the P&ID on which the equipment appears.

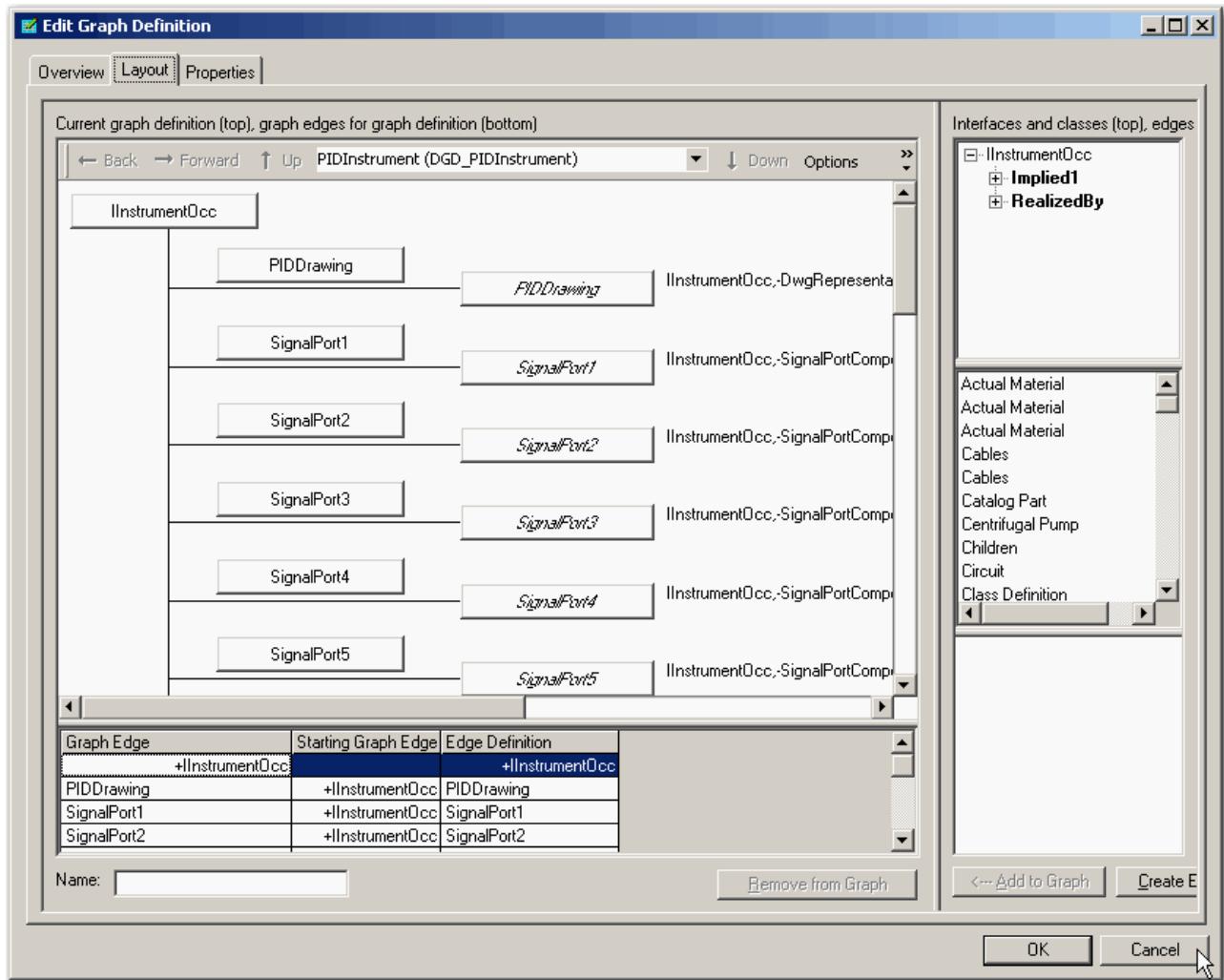


5.3.1 Properties of a Graph Definition

The following options are available when you create or edit a graph definition:

- ❑ **Graph name** – Specifies the name of the graph definition.
- ❑ **Comparison graph** – Specifies whether this graph definition is used for comparison. A comparison graph definition is a special type of graph definition that is intended for performing comparisons between the two sets of data described by the comparison graph definition.
- ❑ **Display name** – Specifies the name that you want the user interface to use when displaying the graph definition.
- ❑ **Starting interface** – Defines the starting interface for the graph definition. The starting interface determines which edge definitions are available to include in the graph definition. When you are editing a graph definition, you cannot modify the starting interface definition.



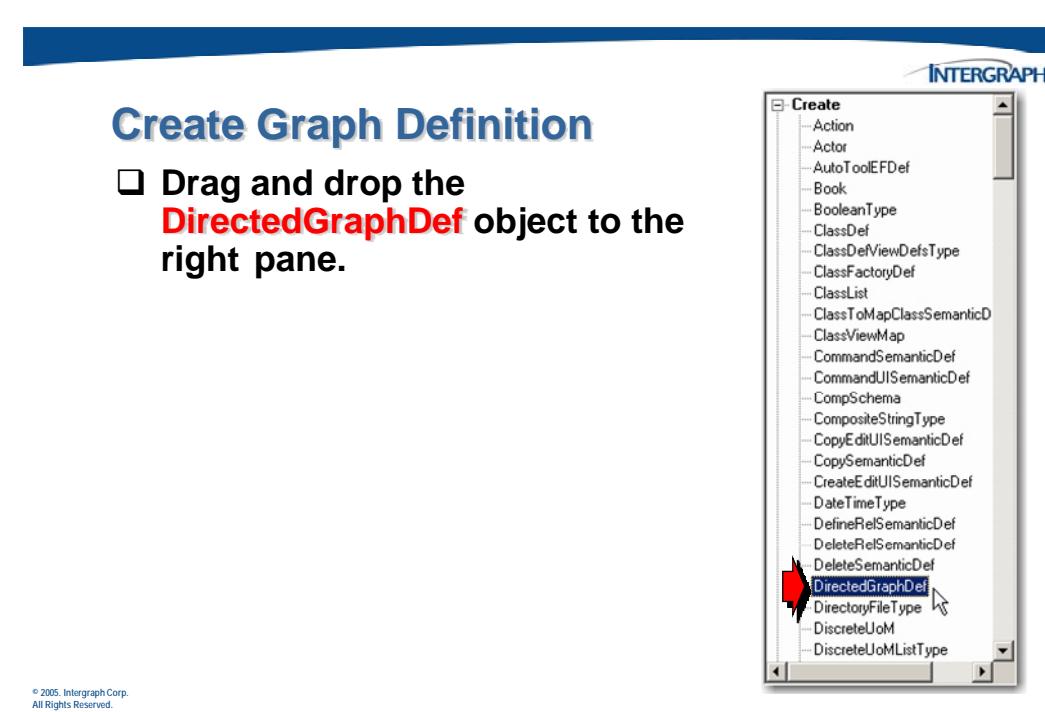


- Layout** – Displays a tree containing all the possible relationship edges that can be expanded from the selected interface definition and all the interfaces that it implies. To add an edge definition to the graph definition, you can click it in the tree view, and then click **Add to Graph**.
- Add to Graph** – Adds the selected edge to the graph definition.
- Current graph definition** – Displays the current graph definition in a tree view.
- Graph definition** – Displays the current graph definition in UML format. Relationships have a + or – in front of them to indicate direction, followed by the relationship name. Commas separate relationships traversed in the GraphDef.

5.4 Interactive Activity – Creating Graph Definitions

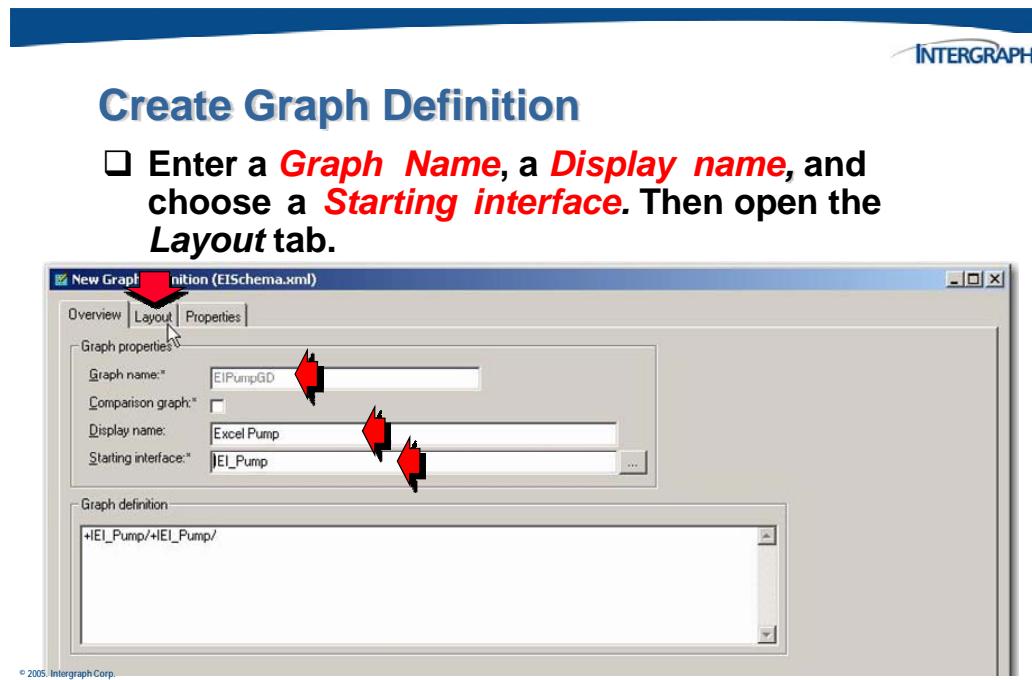
In this interactive activity, we will create a new graph definition for use with our new EIPump class definition.

1. To create a new graph definition, we will find the **DirectedGraphDef** option from the **Create** section of the **Editor** view, and drag and drop it into the UML window.



© 2005, Intergraph Corp.
All Rights Reserved.

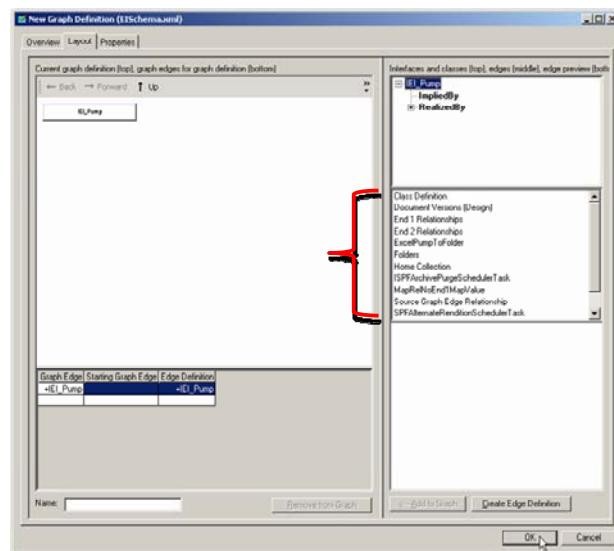
2. Provide the following information for the new Graph def:
 - Graph name** – EIPumpGD
 - Display name** – Excel Pump
 - Starting interface** – IEI_Pump
 3. When you have finished providing information about the new graph def, go to the **Layout** tab.
-





Create Graph Definition

- Use the roles and edge defs in the middle pane of the right-hand side to add paths to the Graph Def.

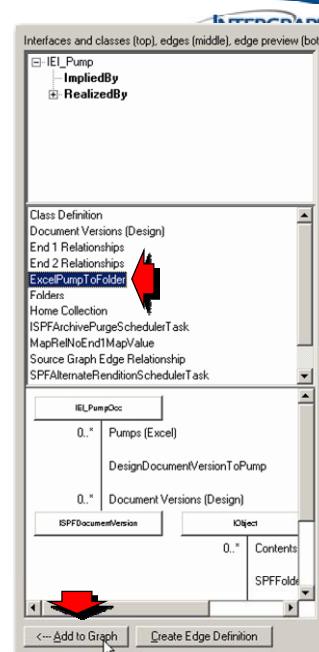


© 2005, Intergraph Corp.
All Rights Reserved.

4. Click the **ExcelPumpToFolder** edge def in the middle pane, and click the **Add to Graph** button.

Create Graph Definition

- Select the **ExcelPumpToFolder** edge def in the middle pane, and click **Add to Graph** to add this edge to the GraphDef.

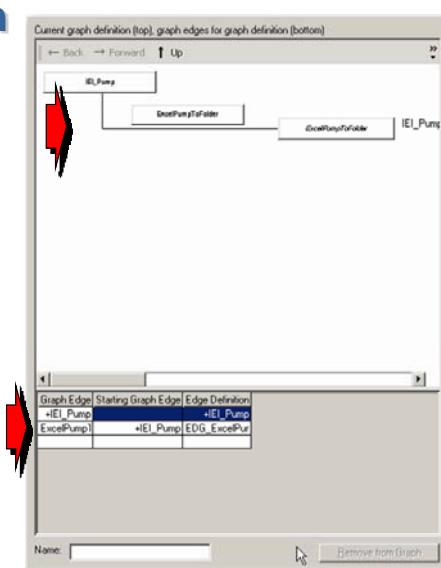


© 2005, Intergraph Corp.
All Rights Reserved.

5. The relationship or edge def will appear in both the top and bottom panes on the left-hand side of the menu.

Create Graph Definition

- The edge appears in both the graphical display at the top and the tabular display on the bottom of the right-hand side pane of the window.

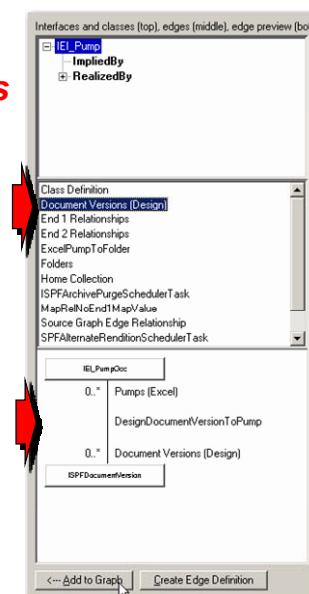


© 2005. Intergraph Corp.
All Rights Reserved.

6. From the middle pane, select the **Document Versions (Design)** role. Click the **Add to Graph** button to add to the path on the left.

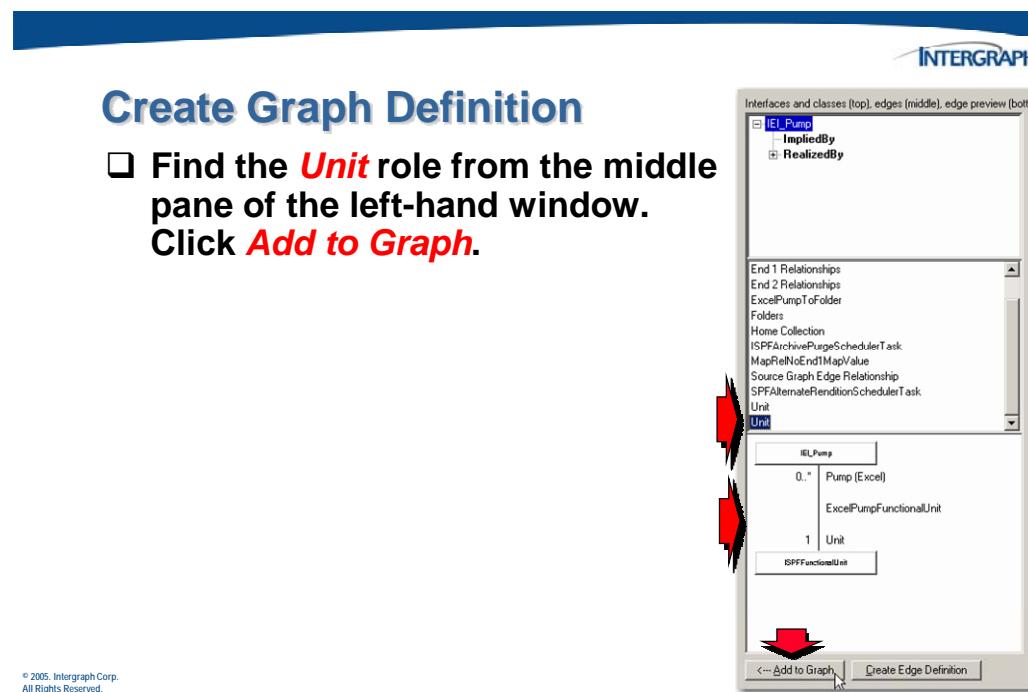
Create Graph Definition

- Next, find the **Document Versions (Design)** role from the middle pane of the left-hand window. Click **Add to Graph**.

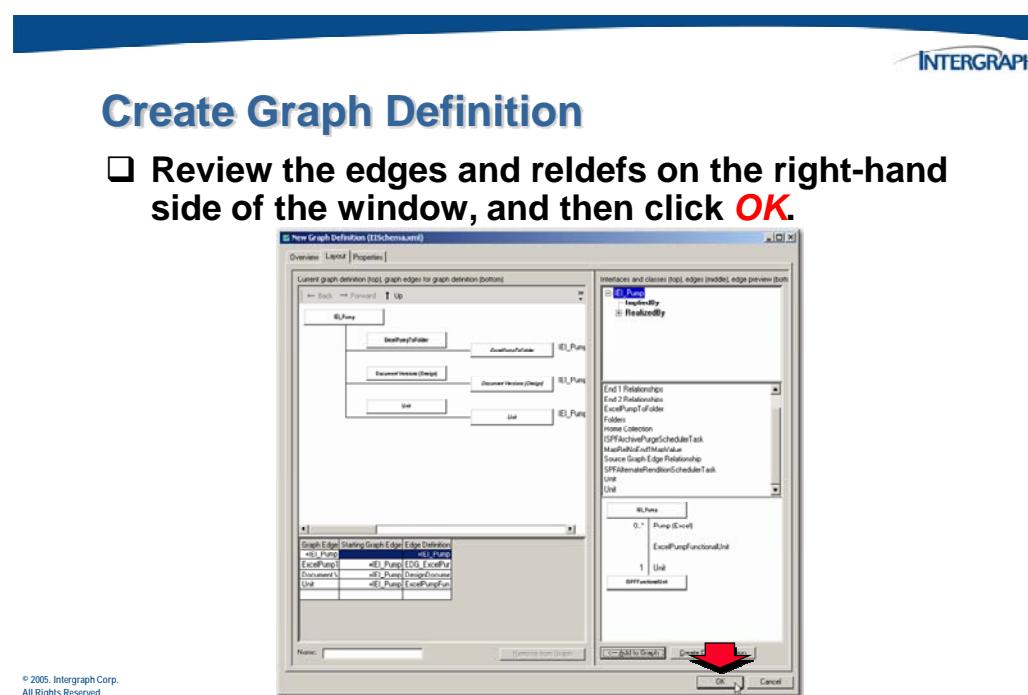


© 2005. Intergraph Corp.
All Rights Reserved.

7. Finally, select the ***Unit*** role, and use the **Add to Graph** button to add it to the path.



8. Click **OK** to create the graph definition.



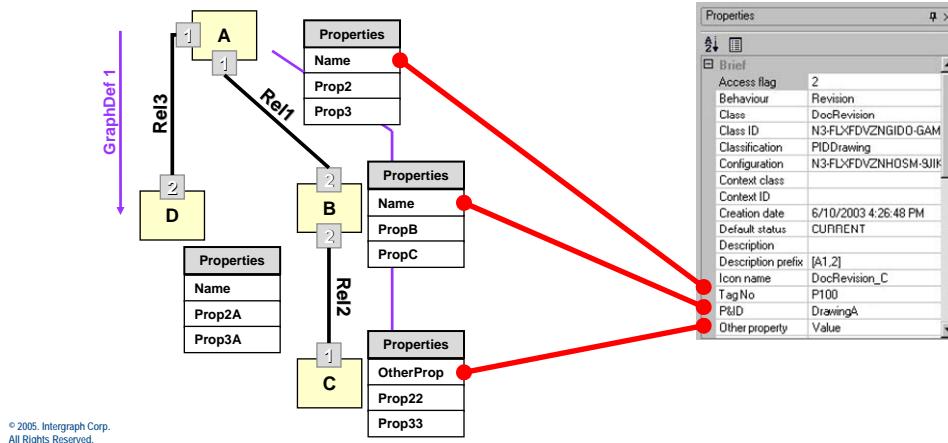
5.5 View Definitions

While schema-based depictions of the data are useful to users who are familiar with the underlying schema, other users need more user-oriented views of the data for it to be meaningful.



View Definitions

View definitions (ViewDef) are properties extracted from possible properties that a graph definition exposes. A view definition is used to provide a different view of data from that provided by the underlying schema.





View Definitions

ViewDefs are SmartPlant equivalents of relational database views. A relational database view is a combination of joins and projects where the joins are used to retrieve the desired objects and the projects are used to determine which properties (columns) to include and what to call those properties.

© 2005. Intergraph Corp.
All Rights Reserved.

SmartPlant view definitions consist of the following:

- A relationship to its starting interface definition
- An identification of the directed graph definition that applies
- A definition of the projection of property definitions from the directed graph definition

A view definition is based on a directed graph definition and, therefore, like the directed graph definition, has a relationship to its starting interface definition. In actuality, this interface definition is always the same interface definition as that for its directed graph definition. The directed graph definition for the view definition defines the set of edge definitions that will be traversed when the view corresponding to the view definition is created.

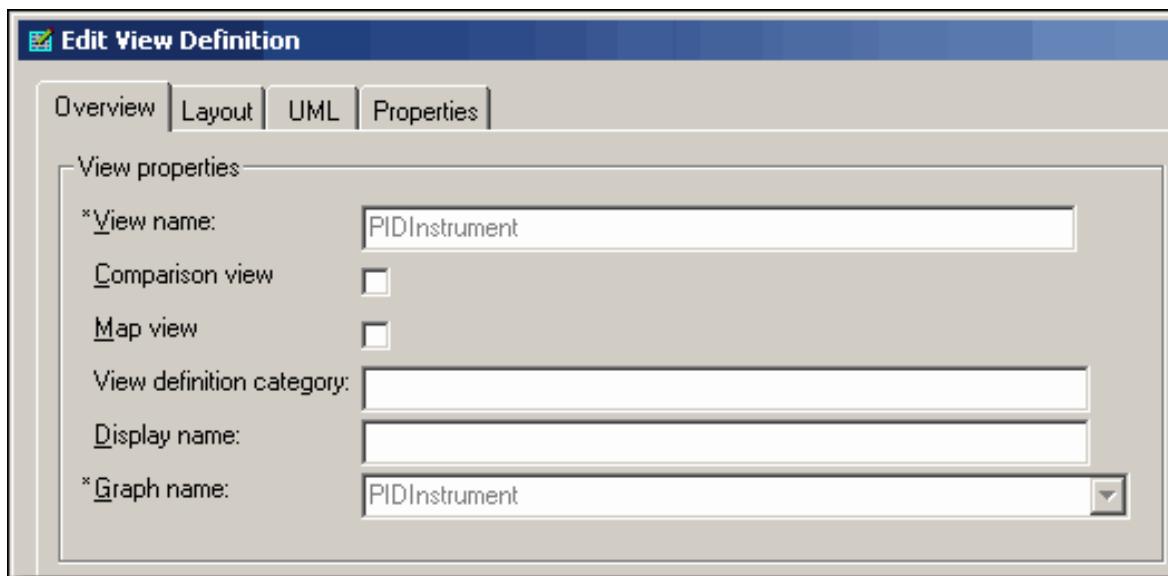
In SmartPlant Foundation, view definitions are used to define what users see in the **Properties** window when they select an object in the client. When you create ad-hoc reports in the SmartPlant Foundation client, you can also select the view definition that you want to use as the basis for the report as well as the properties from that view definition that you want to include in the report.

You can also use view definitions to create alternate views for class definitions.

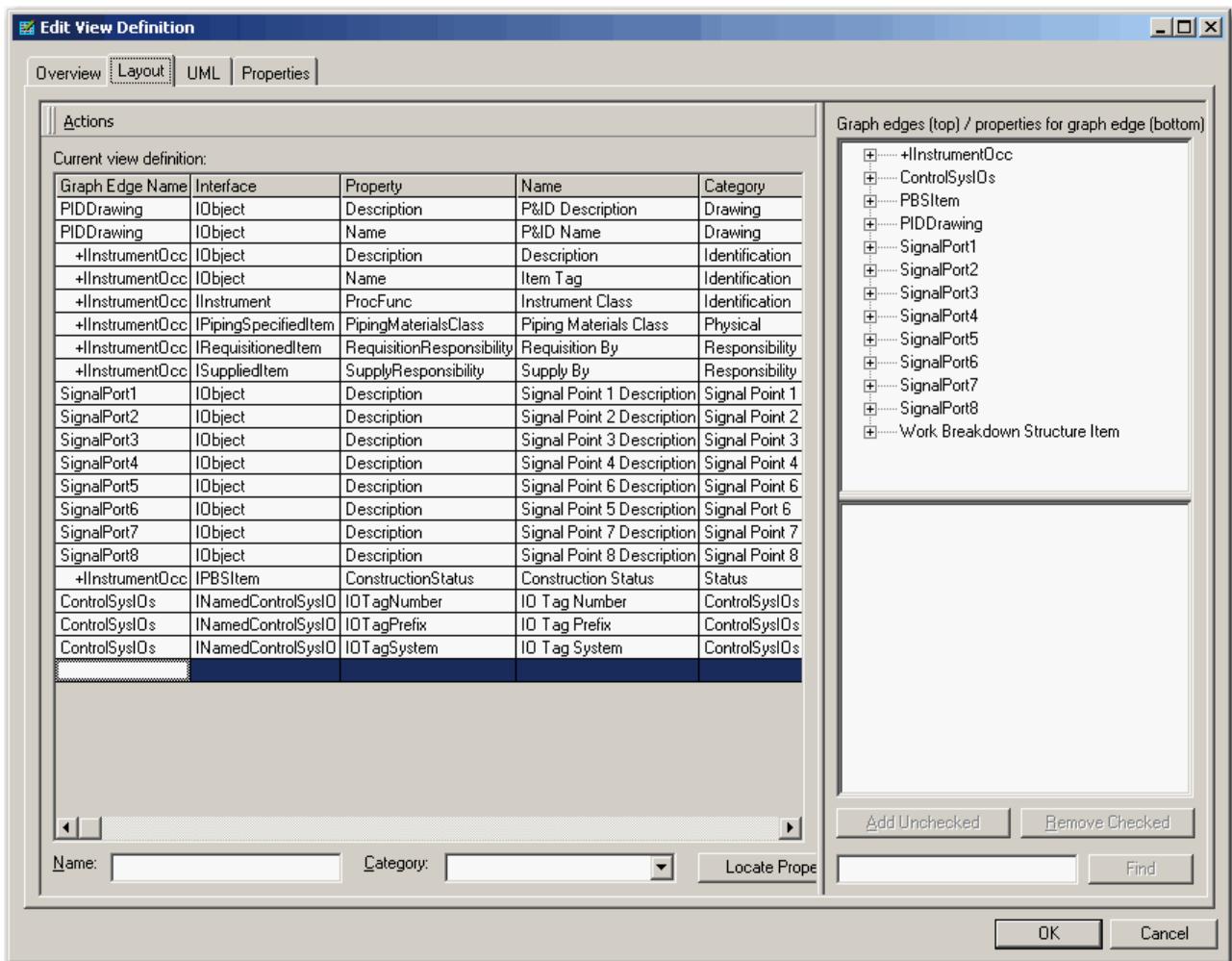
5.5.1 Properties of a View Definition

The following options are available when you create or edit a view definition:

- ❑ **View name** – Specifies the name of the view definition.
- ❑ **Comparison view** - Specifies whether this view definition is used for comparison. A comparison graph definition is a special type of view definition that is intended for performing comparisons between the two sets of data described by the comparison view definition.
- ❑ **Map view** – Specifies whether or not the view definition is to be used for defining mapping between applications and the SmartPlant schema.
- ❑ **View definition category** – Currently not used
- ❑ **Display name** - Specifies the name that you want the user interface to use when displaying the view definition.
- ❑ **Graph name** – Specifies the name of the graph definition that this view definition contains.



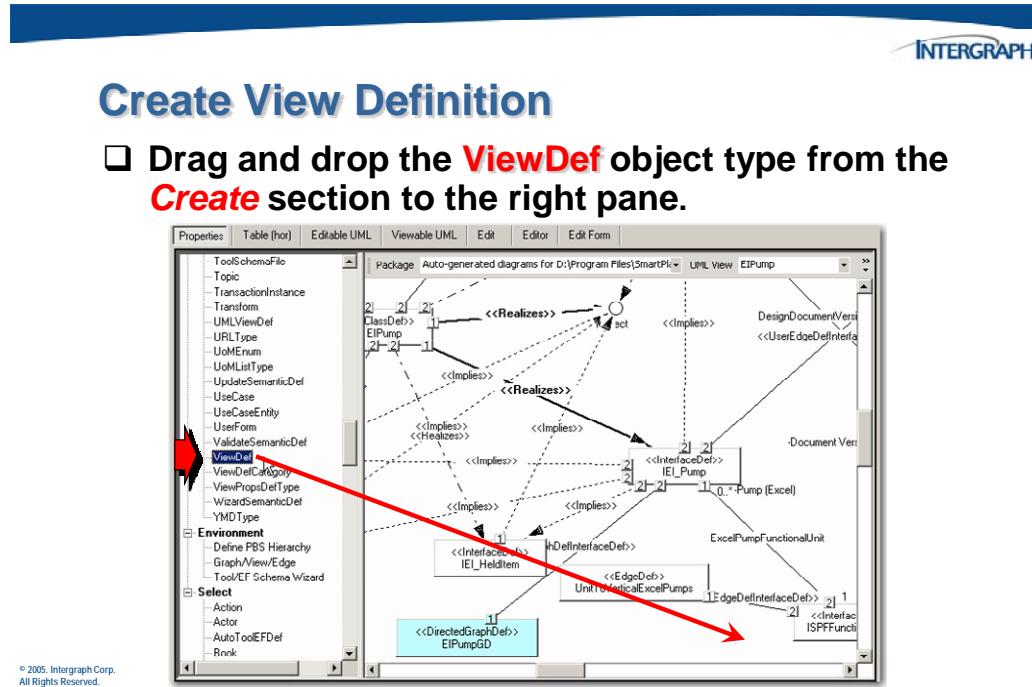
- ❑ **Current view definition** - Displays the current view definition in text format. You can edit the view definition manually in this box.
- ❑ **Name** – Specifies the name of the selected property when it appears in the view definition. After you select the property that you want to add you can type a new display name here.



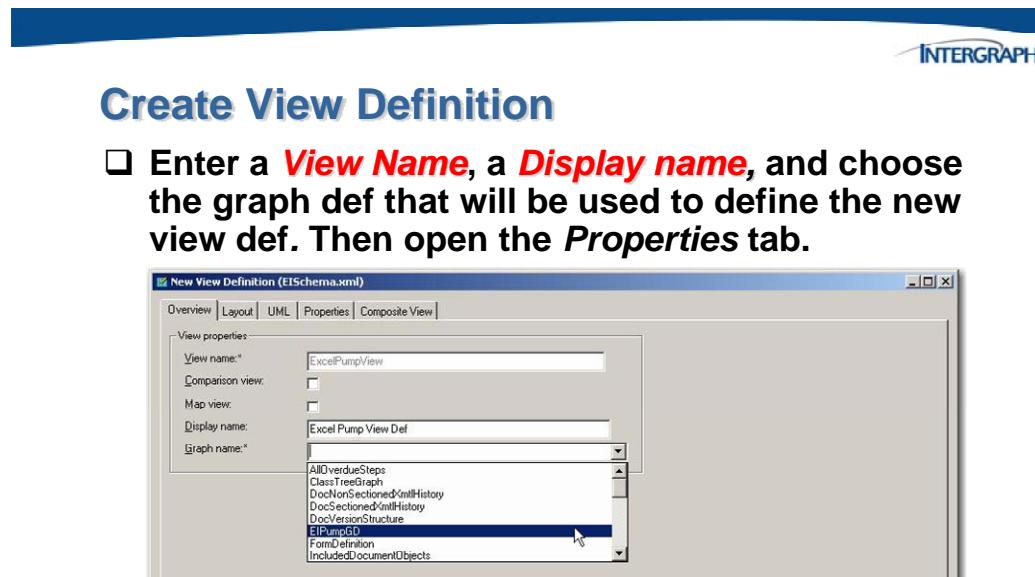
5.6 Interactive Activity – Creating View Definitions

This interactivity activity demonstrates the process of creating a new view definitions for use with the new EIPump class def.

1. Create a new view definition by finding the ***ViewDef*** option from the ***Create*** section of the ***Editor*** view and dragging and dropping that option into the UML view.



2. Create a new view definition using the following information:
 - View name** – ExcelPumpView
 - Display name** –Excel Pump View Def
 - Graph name** – EIPumpGD
3. When you have finishes providing the required information, go to the **Layout** tab of this dialog box.

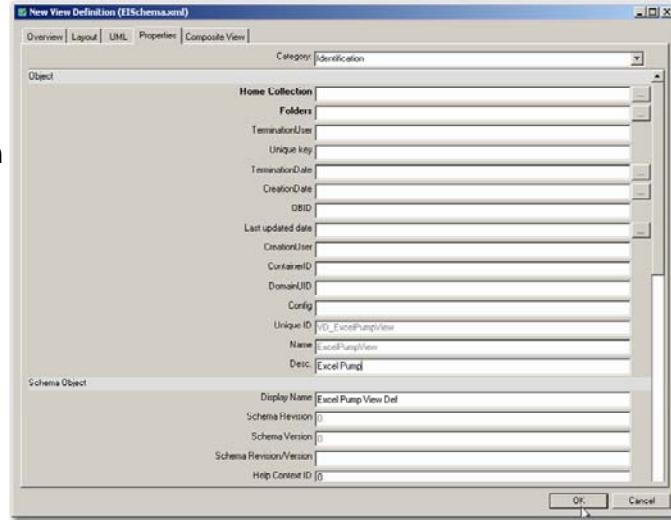




Create View Definition

- If you will be using this view definition to run reports in the Desktop Client, be sure to provide a **Description** on the **Properties** tab. Then continue to the **Layout** tab.

© 2005, Intergraph Corp.
All Rights Reserved.



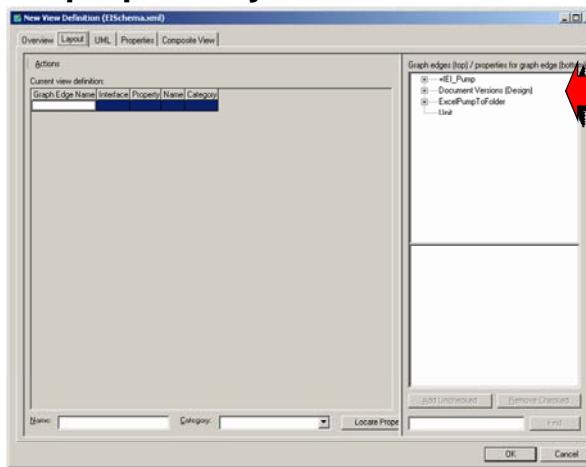
Note: If you do not provide a description for the view def, as illustrated above, the view def will appear as a blank link in the New Report dialog box in the Desktop Client.



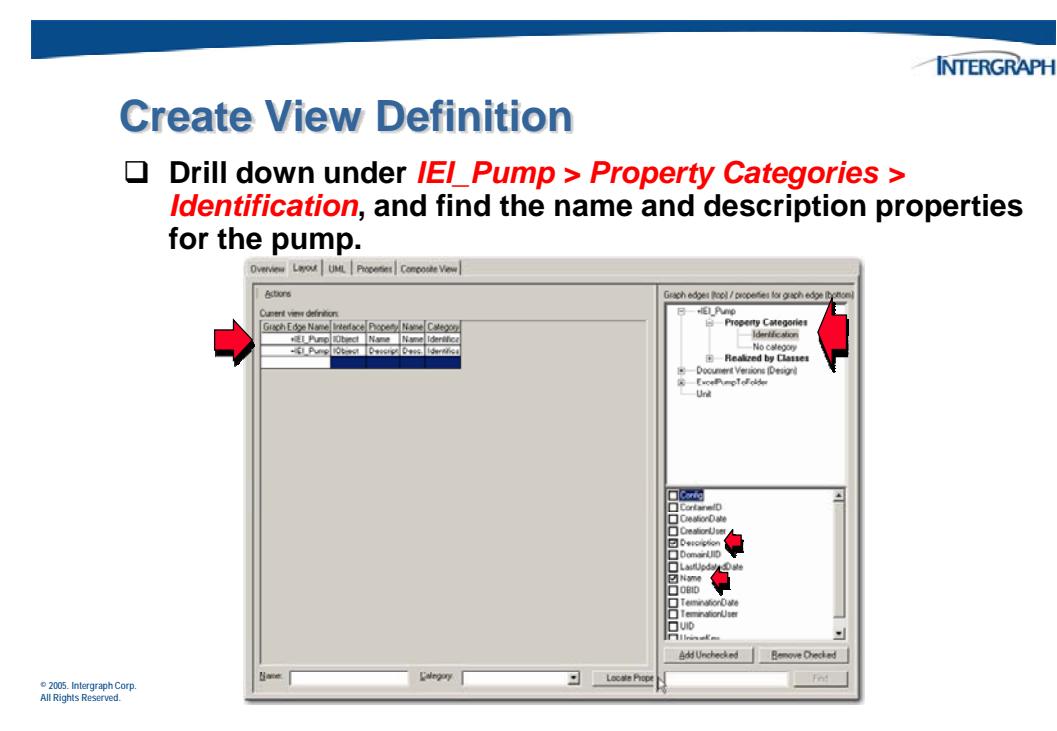
Create View Definition

- Use the list of interfaces, rel defs, and edge def to find the properties you want to add to the view def.

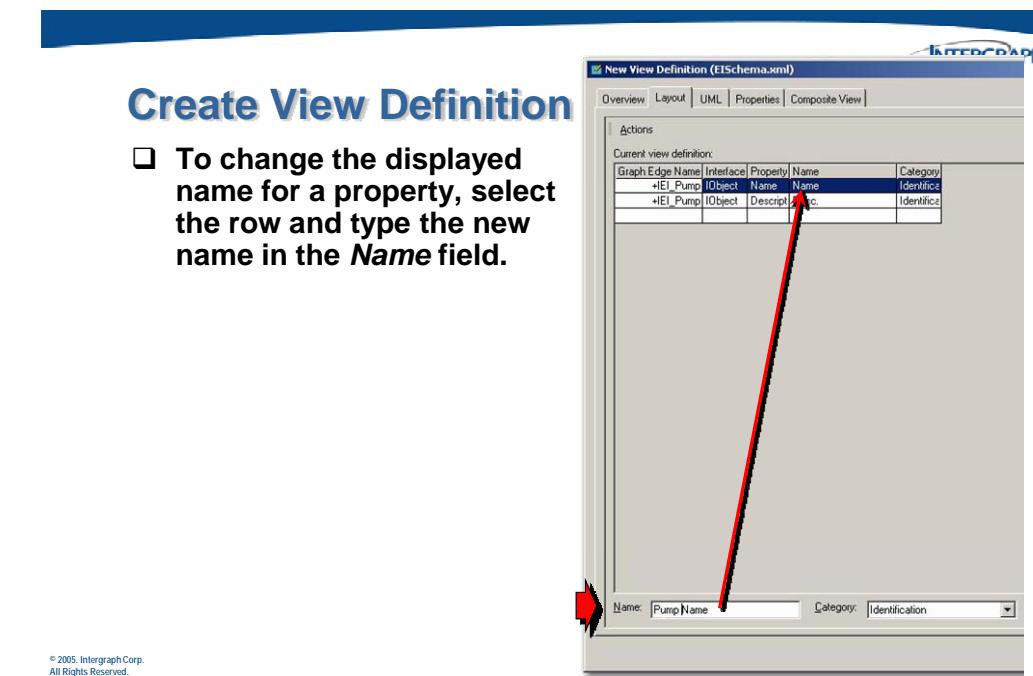
© 2005, Intergraph Corp.
All Rights Reserved.



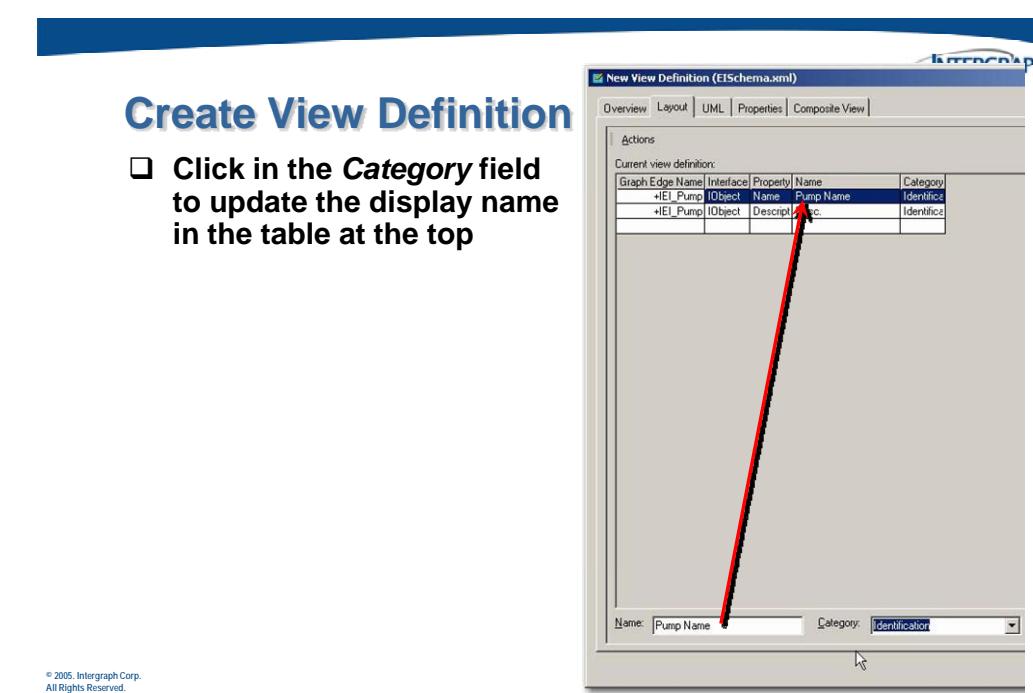
4. Expand the **IEI_Pump** interface definition in the top, right-hand pane. Select the Identification property category to populate the bottom pane with a list of properties.
5. Activate the check boxes for the **Name** and **Description** fields to add those properties to the view definition.



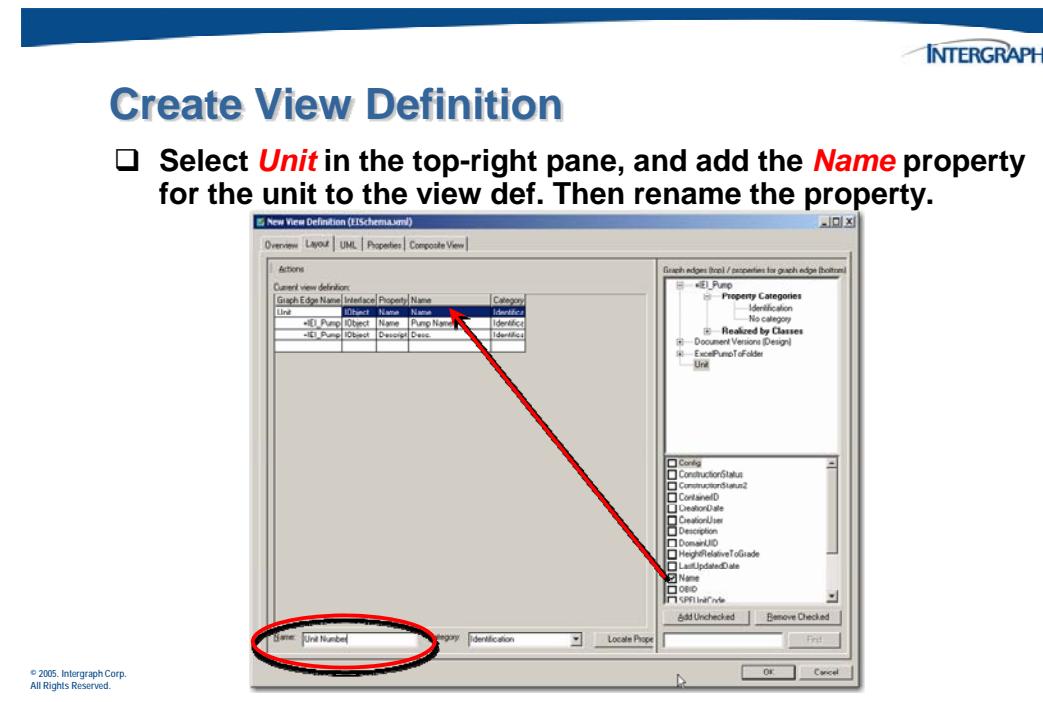
6. To change the display name of a property, select the row with the applicable property, and modify the value in the **Name** field at the bottom of the left-hand side of the dialog box.



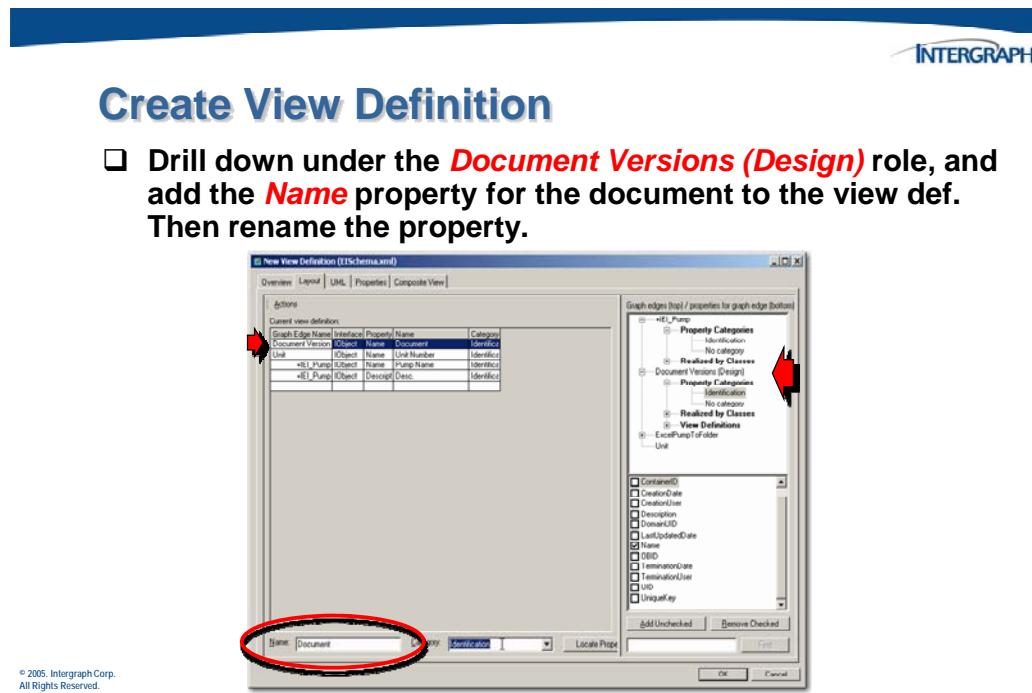
7. Click the **Category** field to apply the name change to the property.



8. Select **Unit** in the top right-pane, and add the **Name** property to the view def. Rename the property to reflect that the property is the name of the unit associated with the pump, not the pump itself.



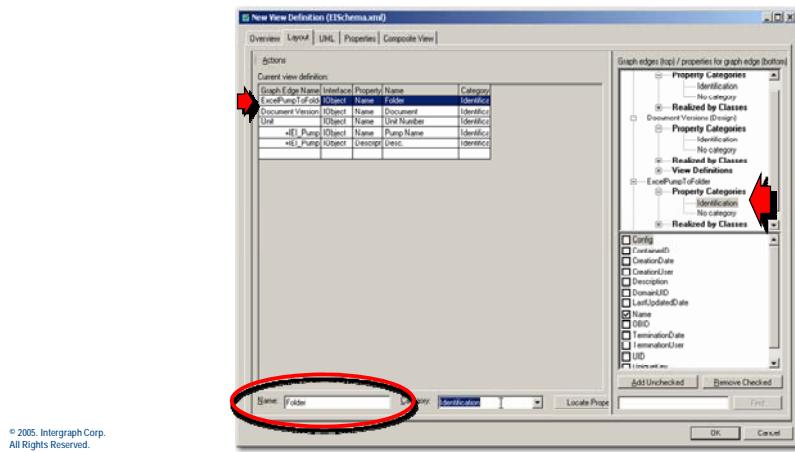
9. Expand the **Document Versions (Design)** role in the top, right pane, and add the **Name** property. Modify the name of the property to reflect that this is the name of a document version associated with the pump.
-



10. Expland the **ExcelPumpToFolder** edge definition, and find the **Name** property to add to the view def. Modify the display name of the property to indicate that this is the name of the folder associated with the pump.

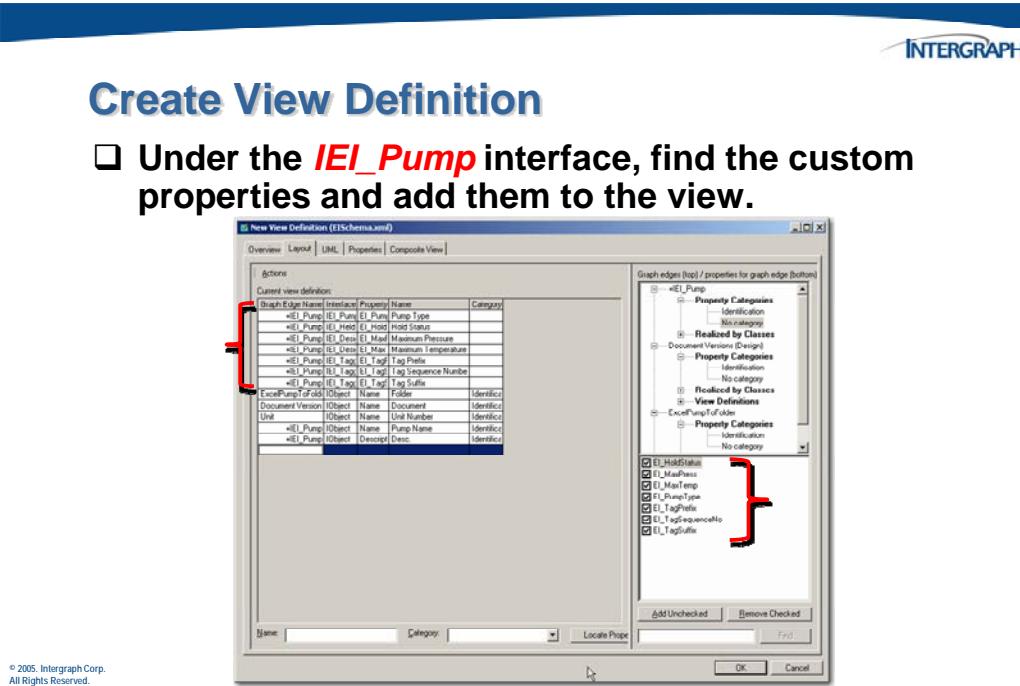
Create View Definition

- ❑ Expand the **ExcelPumpToFolder** edge def and find the **Name** property for the folder. Add it to the view def and rename it.

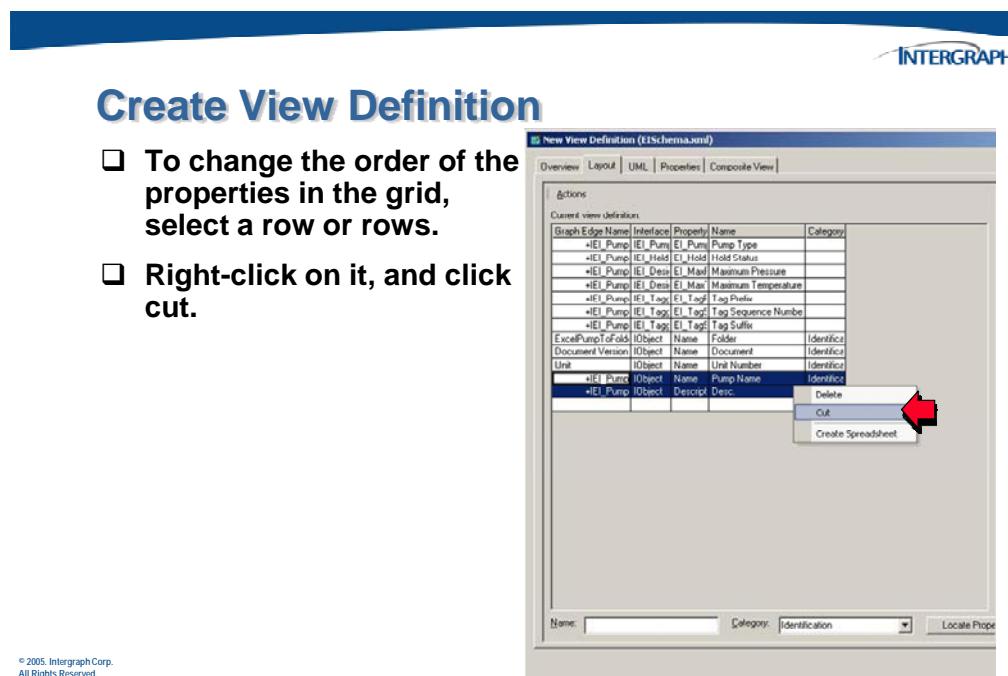


© 2005, Intergraph Corp.
All Rights Reserved.

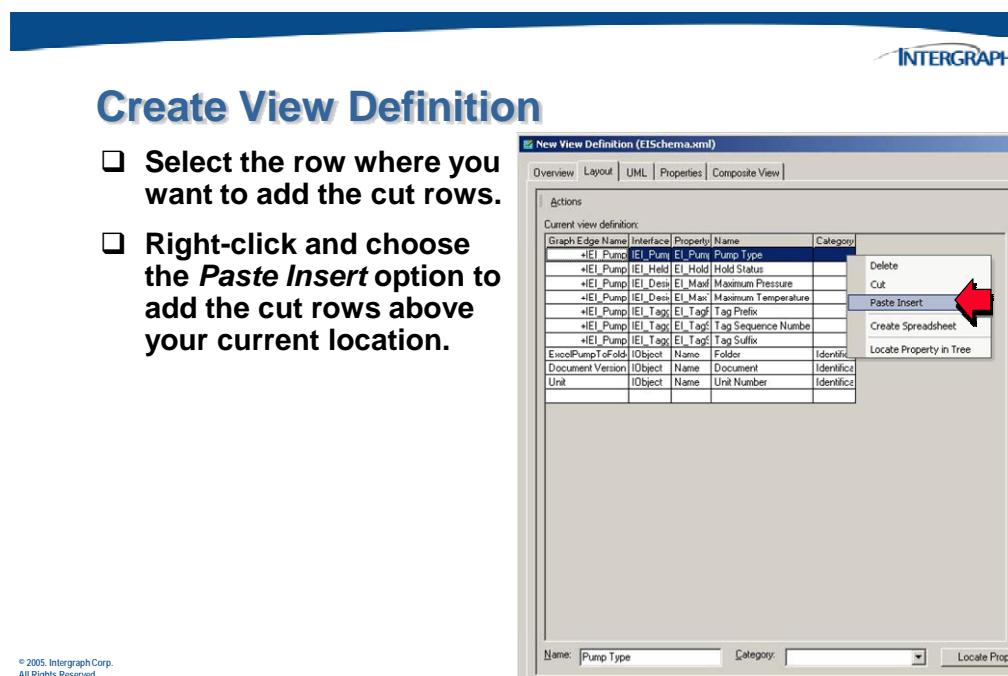
11. From the *No category* property category under the ***IEI_Pump*** interface, find the custom properties you created to describe the new pumps. Add the custom properties to the view def and modify their names, as necessary.



12. To change the order of the properties in this new def, select the row or rows that you want to move. Right-click on the selection, and use the **Cut** command.



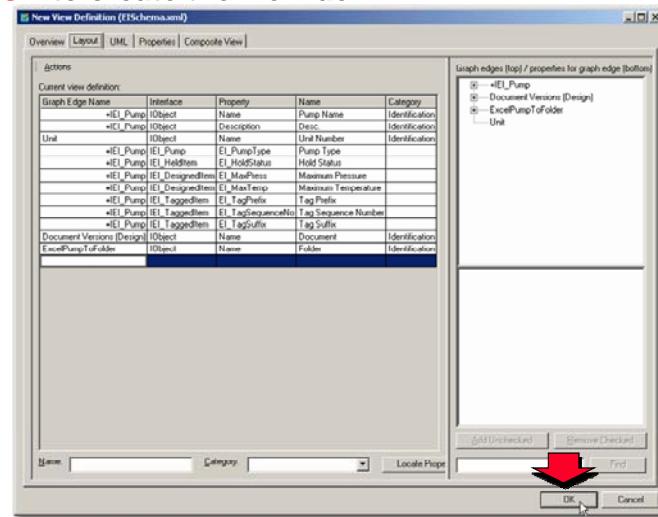
13. Select the row above which you want to place the cut rows, right-click, and choose the **Paste Insert** command.



14. At this point you can click **OK** to create the view definition. Alternately, if you want, you may first click on the **UML** tab to review the view def in a different format before clicking **OK**.

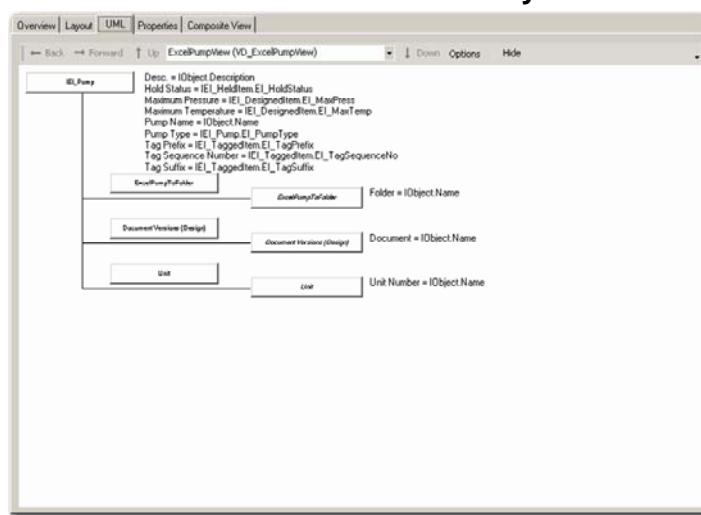
Create View Definition

- Click **OK** to create the view def.



Create View Definition

- You can review the UML view before you click **OK**.



5.7 Class View Maps

In the schema, **class view maps** associate class definitions with view definitions. After the SmartPlant schema is loaded into the SmartPlant Foundation system administration database, class view maps can be used to define user access to different views in the SmartPlant Foundation client in SmartPlant Foundation SmartPlant Administration. The class view map specifies the default view definitions the software should use for a set of class definitions.



Class View Maps

A **Class View Map** (**ClassViewMap**) is a collection of **ClassDefs** and **ViewDefs** in the schema that has related user groups in SmartPlant Foundation.

ClassViewMaps define the default **ViewDefs** the software should use for a set of **ClassDefs**.

Relationships with SPF user groups specify which view the software uses in the *Properties* window or in an ad-hoc report when a particular user selects a **ClassDef** in the SPF client.

Associations between SPF user groups and ClassViewMaps are defined in SPF SmartPlant Administration.

© 2005, Intergraph Corp.
All Rights Reserved.

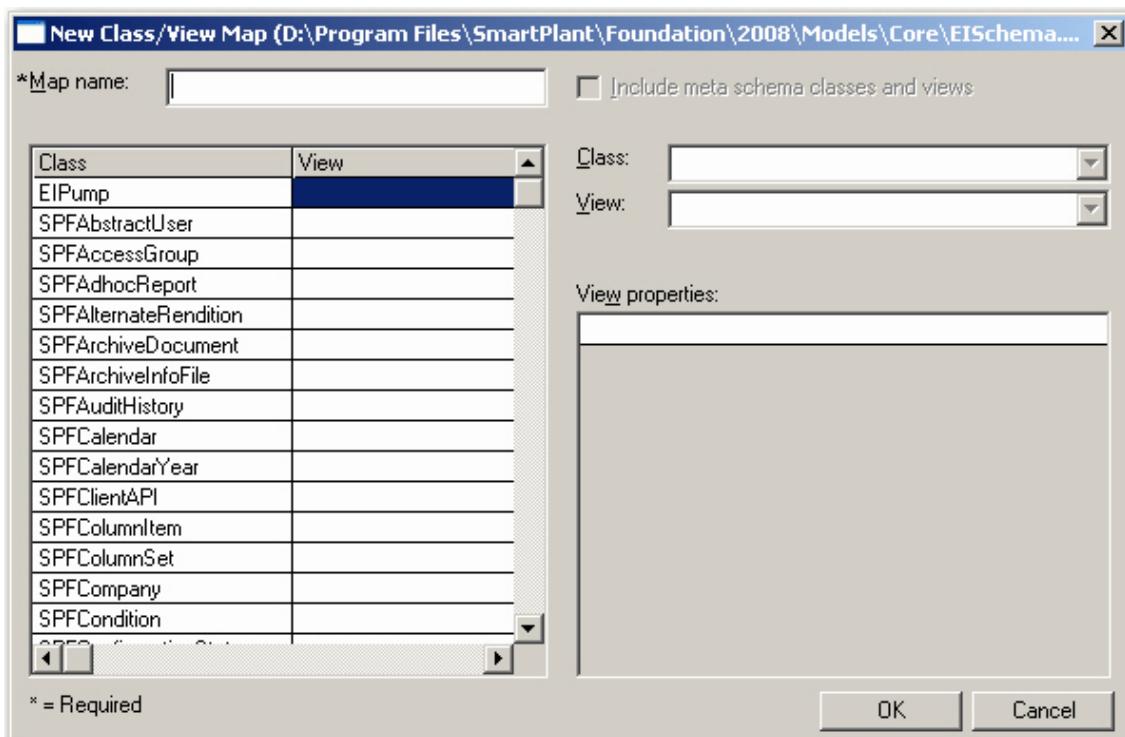
Defining the class view map is especially important for shared class definitions because it allows you to use the same view definition for multiple class definitions, including those objects that are shared across tools. By using the same view definition for multiple class definitions, users see the same presentation of information whether they are looking at the originally published class or a class from another tool for the shared object.

For example, if SmartPlant P&ID publishes an object of the PIDProcessEquipment class and Zyqad publishes an object of the EQDCentrifugalPump class, the shared object will have a different class in SmartPlant Foundation. To allow users to see the same view of the shared object regardless of which tool published it, the same view definition could be associated with both the class definitions.

5.7.1 Properties of a Class View Map

The following options are available when you create or edit a graph definition:

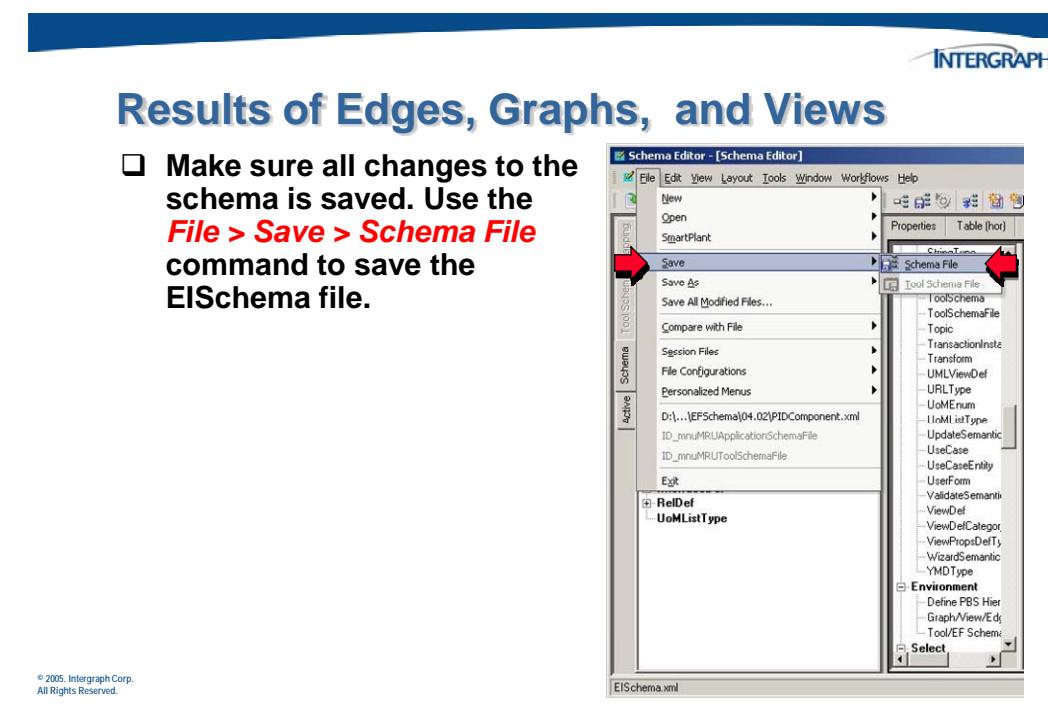
- Map name** – Specifies the name of the class view map.
- Class/View Table** - Displays a list of all class definitions and the view definitions associated with them. To select a class definition to map, you can click it in this table or in the **Class** box. You can also change the view mapping for a class definition by selecting the class definition in this table and making changes in the **Class** and **View** boxes.



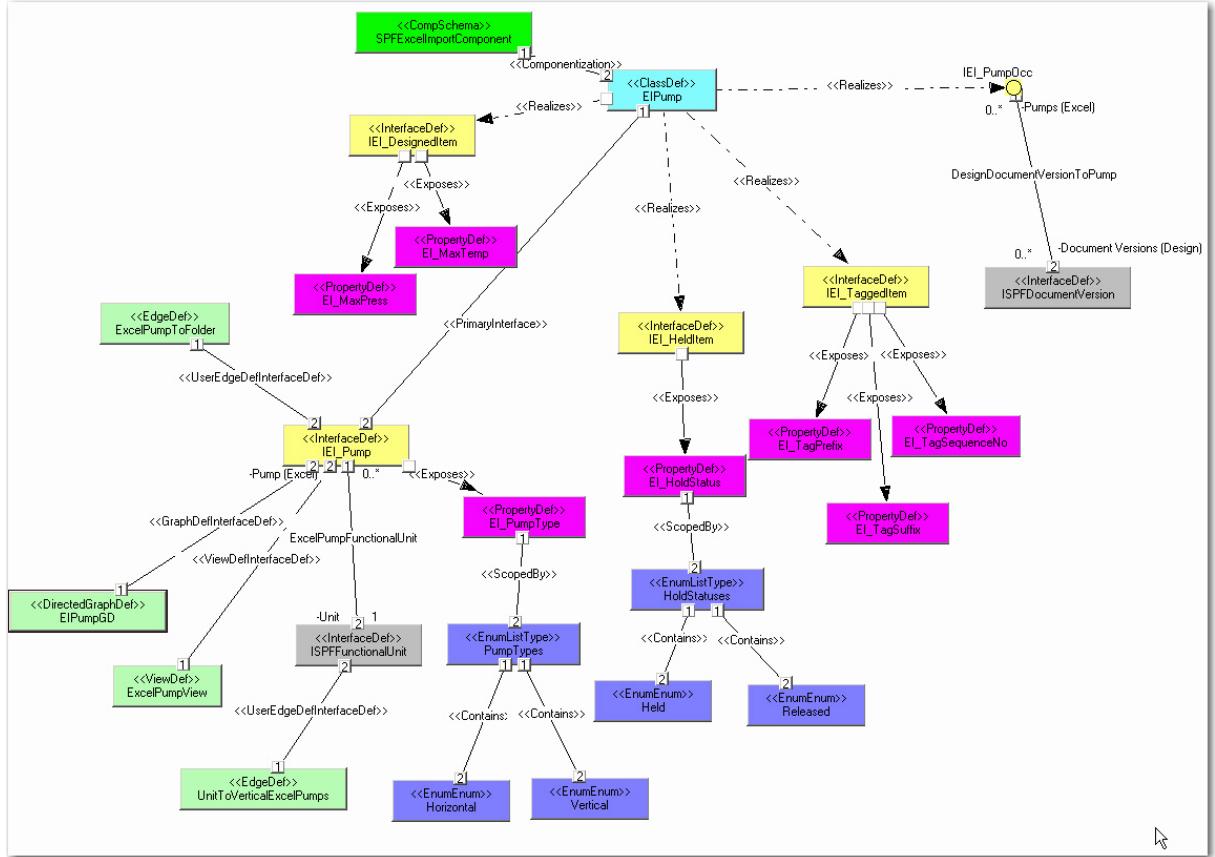
- Include meta schema classes and views** – Indicates whether you want to include classes and views from the meta schema in your class view map. Including meta schema classes and views are only used inside the Schema Editor. They are useful when you want to use class view maps to define the set of comparison view definitions to use for the set of corresponding class definitions or to define the view that is displayed in the Table view in the Schema Editor for each class definition.
- Class** – Specifies the class definition to which you want to map a view definition. You can select a class definition in this list or click the class definition in the table on the left side of the dialog box to automatically fill in the **Class** box.
- View** – Specifies the view definition that you want to associate with the class definition selected in the **Class** box.
- View properties** – Displays a list of all properties associated with the view selected in the **View** list.

5.7.2 Save the Changes to the Schema File

Be sure to save the changes you have made to the your schema file. Use the **File > Save > Schema File** command to save the changes to the EISchema file.



The UML diagram below shows all the changes that have been made to the schema.



6

C H A P T E R

Viewing and Finding Data

6. Viewing and Finding Data

Many sessions in the Schema Editor are typically performed to view the data files and not the schema files. When you want to view data files, you must open a schema file before you can open the data file. To view data organized by schema classes, it is often useful to view the data in terms of the schema.



Viewing a Data File

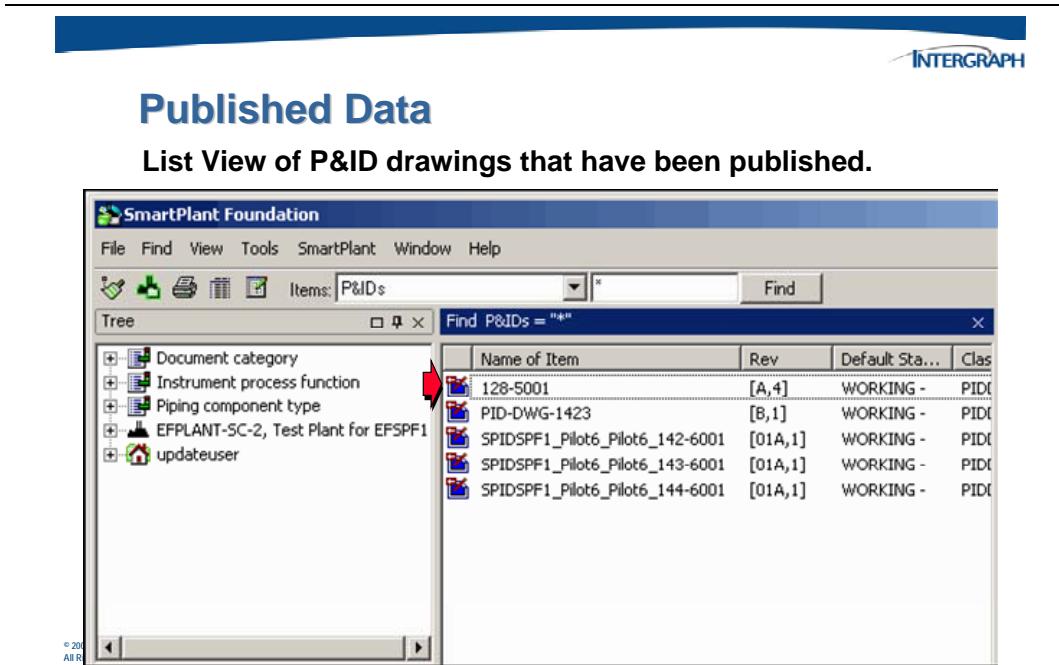
Many of the same views that are useful for viewing the schema are also useful for viewing data files.

Some of the most valuable data views include:

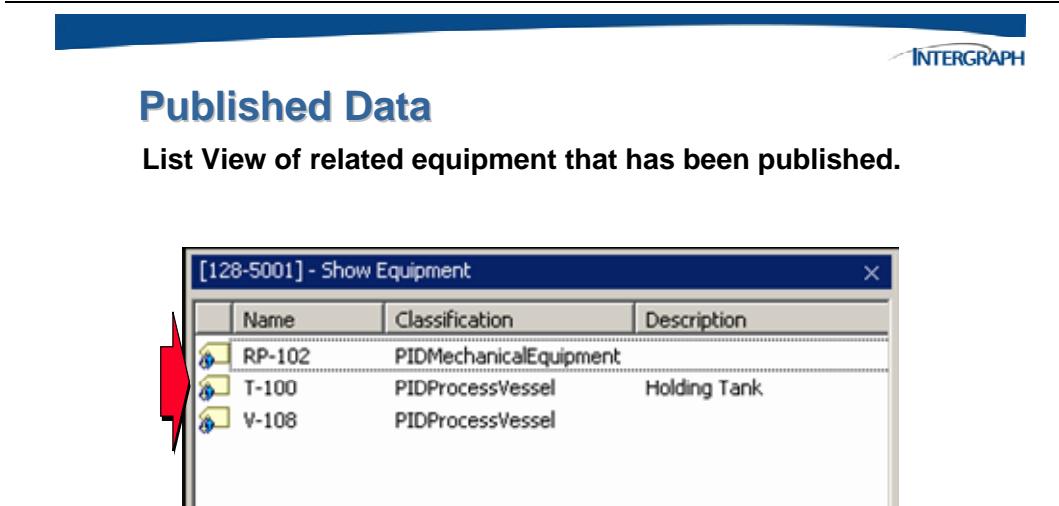
- Data Tree/Viewable UML View**
- Data Tree/Table View**
- Data Tree/Properties View**
- Data Tree View**

The data that can be viewed by the schema editor is the contents of the XML container that gets created as a result of performing a **Publish** operation from one of the engineering authoring tools.

For example, if drawing **128-5001** gets published from SmartPlant P&ID, the instantiation of the objects can be viewed in the SmartPlant Foundation *Desktop Client* browser.



Using the **Show Equipment** command will show the published equipment objects.



Using the **Show Instruments** command will show the published instrument objects.

Published Data

List View of related instruments that have been published.



Name	Classification	Description
ABV-128HV	PIDInlineInstrument	
SA-305	PIDInstrument	
SV-305	PIDInstrument	

© 2005, Intergraph Corp.
All Rights Reserved.

Using the **Show Lines** command will show the published pipeline objects.

Published Data

List View of related pipelines that have been published.



Name	Classification	Description
PH-101	PIDPipeline	
PH-104	PIDPipeline	

© 2005, Intergraph Corp.
All Rights Reserved.

6.1 XML Files Overview

All documents that are published into SmartPlant or retrieved from SmartPlant are in XML format. These XML files must conform to the structure defined by the SmartPlant schema.



XML Files

When a set of objects, grouped by document, are published or retrieved, an XML file container is used to house those objects in XML format.

Each publish includes at least data and metadata containers and may also contain an instructions (tombstones) container.

In XML files, the <class> elements identify objects of a particular class. Within each <class> element, there are the interfaces for that class. The <interface> elements expose the properties for the class. The property type for each property defines the acceptable values for that property in the XML file.

The following is sample data from a P&ID data container:

```
- <Container CompSchema="PIDComponent" Scope="Data" SoftwareVersion="03.08.00.12"
  SchemaVersion="03.08.00.04 Plus29922 MINUS Edge" Plant="EFPLANT-SC-2" Project=""
  DocUID="4CECEEA6B99D4490A557D9BFC4134435" DocName="128-5001" Revision="A"
  Version="4" ToolID="SMARTPLANTPID" ToolSignature="AABQ">
- <PIDDrawing>
  <IObject UID="4CECEEA6B99D4490A557D9BFC4134435" Name="128-5001"
    Description="" />
  <IDocument DocCategory="P&ID Documents" DocTitle="" DocType="P&ID" DocSubtype="" />
  <IDocVersionComposition />
  <IDwgRepresentationComposition />
  <IPIDDrawing />
  <ISchematicDwg />
</PIDDrawing>
- <PIDInlineInstrument>
  <IObject UID="04992AEB25CA489AA8BAA8242683B5A9" Name="ABV-128HV" />
  <IInlineInstrument />
  <IInstrument ProcFunc="@EE3F8" InstrumentType3="@EE425" InstrumentType="@EE3F8.1" />
  <IInstrumentOcc />
  <IPBSItem ConstructionStatus="@NewConstruction" ConstructionStatus2="@{78398AB4-
    9F3D-11D6-BDA7-00104BCC2B69}" />
  <IPipingPortComposition />
  <IPlannedMatl />
  <IDrawingItem />
  <IInlineInstrumentOcc />

<IPressureReliefItem />
<IProcessPointCollection />
<ISignalPortComposition />
<IDocumentItem />
<IElecPowerConsumer />
<IPart />
<INoteCollection />
<INamedInstrument InstrFuncModifier="ABV" InstrLoopSuffix="" InstrTagPrefix="Unit1"
  InstrTagSequenceNo="128" InstrTagSuffix="HV" MeasuredVariable="" />
<IPipeCrossSectionItem NominalDiameter="4 in" />
<IPipingSpecifiedItem PipingMaterialsClass="ABV" />
<IIInsulatedItem InsulCompositeMatl="@{12EB29A1-B69C-11D6-BDB9-00104BCC2B69}" />
<IReliefDevice OrificeLetter="@{96C5801B-9C2D-4E74-94CD-50890AAC614F}" />
<IInlineComponent IsFlowDirectional="False" />
</PIDInlineInstrument>
- <Rel>
  <IObject UID="PBS-04992AEB25CA489AA8BAA8242683B5A9-N3-FMCUHPQRGJCA-
    RDAWMMF3" />
  <IRel UID1="04992AEB25CA489AA8BAA8242683B5A9" UID2="N3-FMCUHPQRGJCA-
    RDAWMMF3" DefUID="PBSItemCollection" />
</Rel>
- <PIDInstrument>
  <IObject UID="9F1EFE60E26D47B6B4A8D0D98FF444E8" Name="SA-305" />
  <IInstrument />
```

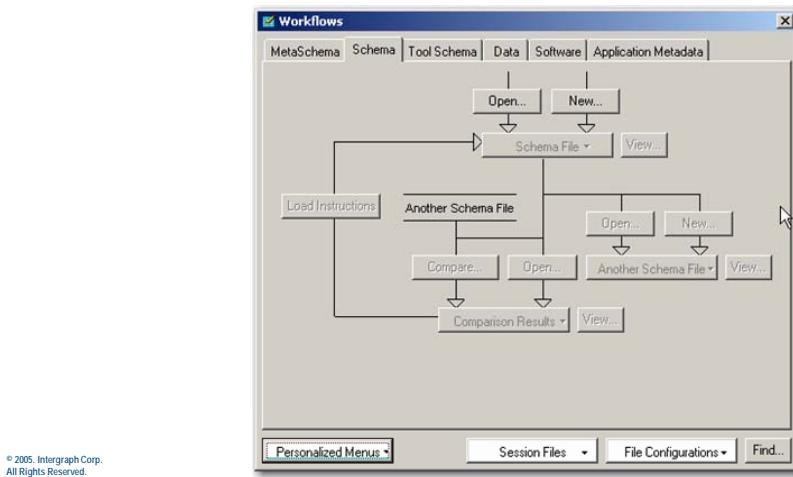
6.2 Opening and Viewing a Data File

After you open a schema file using a saved configuration, you can open an XML data file. However, before you can open a data file, **you must open a schema**. Open the Schema Editor, and open the Overall Workflow.

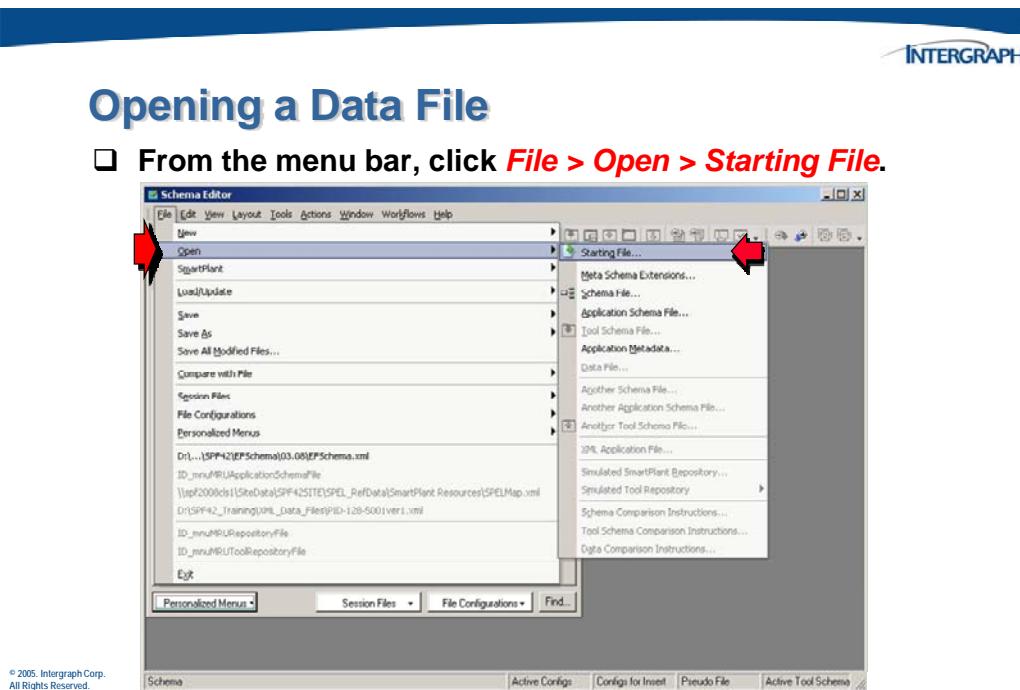


Opening a Data File

- Open the Schema Editor and open the *Overall Workflow*, from the Workflows menu.



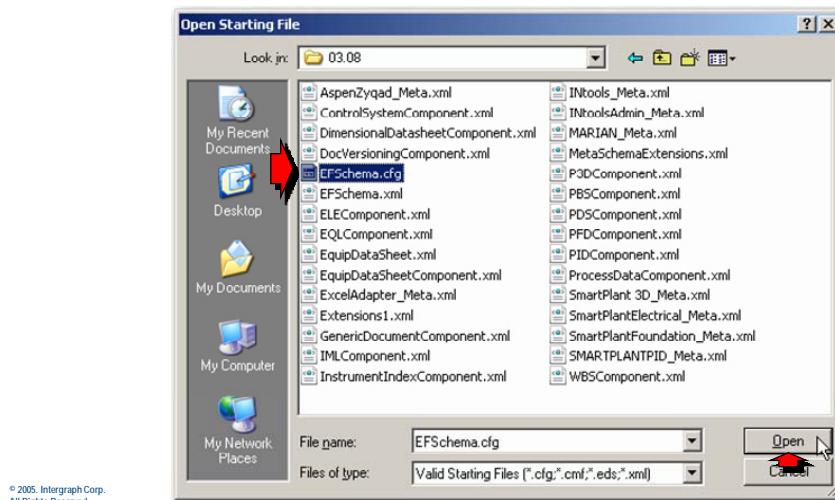
Using the **File > Open > Starting File** command, open the **EFSchema.cfg** file.



This file can be found in the following location: **D:\SmartPlant Foundation 2008 Server Files\Web_Sites\SPF42\EFSchema\03.08**.

Opening a Data File

- Open the EFSchema.cfg file.

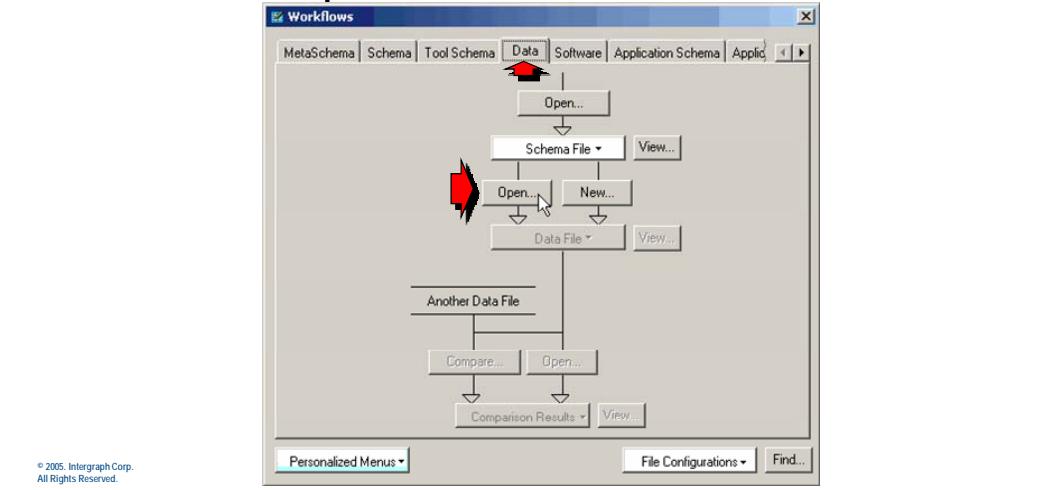


From the **Data** tab, open the data file to view.



Opening a Data File

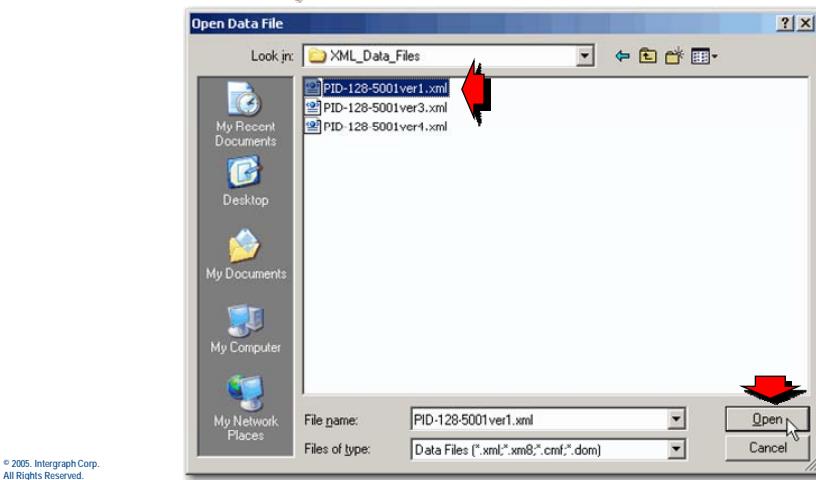
- On the **Data** tab, click the **Open** button above the Data File button to open an **XML** data file.



Find the sample files delivered in **D:\SPF_Training\XML_Data_Files**. In the examples in this chapter, a published XML data file has been copied and renamed to make finding the file easier. In reality, the XML files to be viewed using the schema editor can be found in an SPF vault, but they are encrypted and renamed. We will see more about decrypting them for viewing in a later chapter.

Opening a Data File

- In the **Open Data File** dialog box, select an XML data file and click **Open**.



6.2.1 Data File Tree/UML View

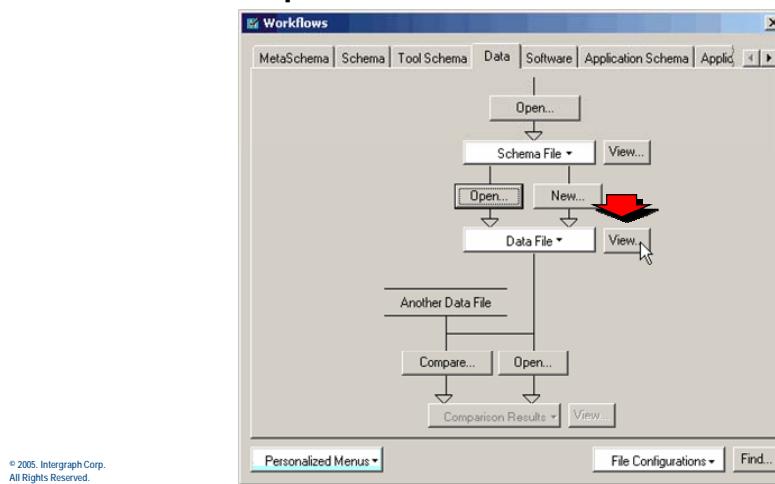
If you remember from the previous chapter, *Using the Schema Editor*, the **Tree/Viewable UML** view is a combination of the standard tree view with a UML view. The UML view provides a graphical representation of the relationships and relationship definitions involving the object selected in the tree view. In the UML view, you can click any object to center the UML around the selected object to view its relationships and relationship definitions.

First, make sure a schema file has been opened.



Viewing a Data File – Tree/UML View

- From the **Workflows** dialog box, click the **View** button to view the open data file.

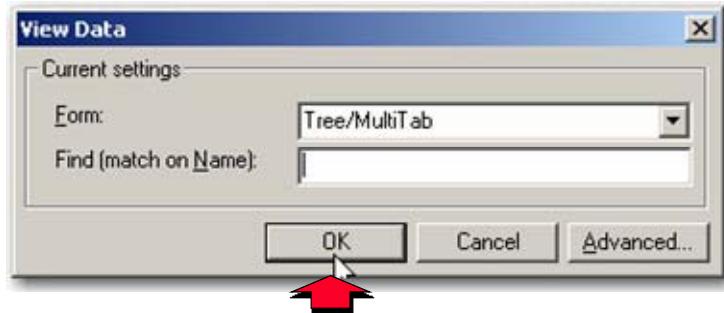


Click **OK** to display all the information in the data file.



Viewing a Data File – Tree/UML View

- In the **View Data** dialog box, choose a view, and then click **OK**.



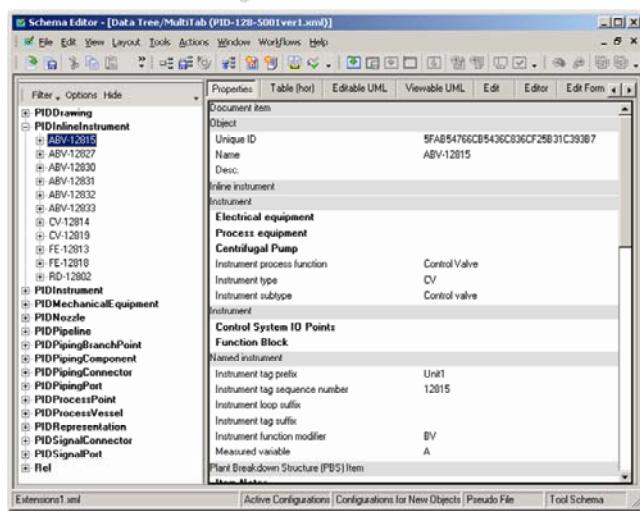
© 2005, Intergraph Corp.
All Rights Reserved.

View the published objects in the tree.



Viewing a Data File – Properties View

- The tree displays objects that were published in the document. Select objects and use the views along the top to see different perspectives of the object.



© 2005, Intergraph Corp.
All Rights Reserved.

Select the object to be viewed into the right pane.

Viewing a Data File – Tree/Viewable UML

Click the + next to **PIDInlineInstrument** to view the related data.

© 2005. Intergraph Corp.
All Rights Reserved.

Display the object in the Viewable UML view.

Viewing a Data File – Tree/Viewable UML

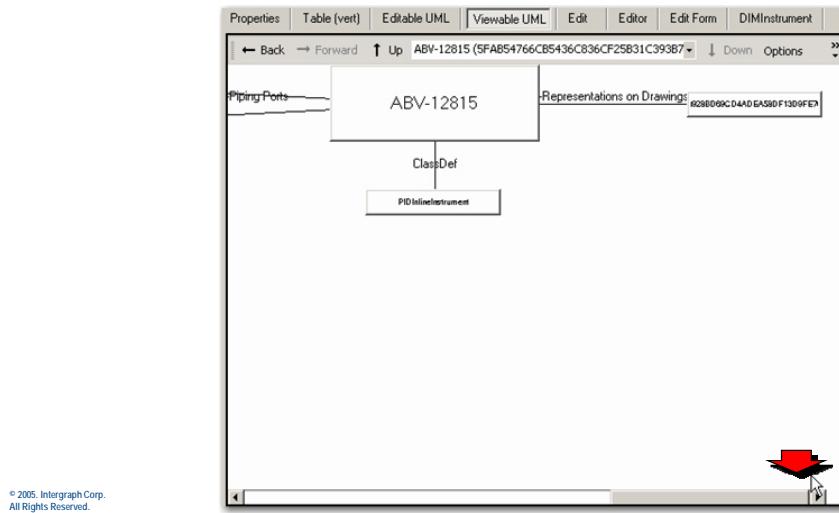
The view will change to show **ABV-12815** and its related components.

© 2005. Intergraph Corp.
All Rights Reserved.

Scroll right to see additional information associated with this object.

Viewing a Data File – Tree/Viewable UML

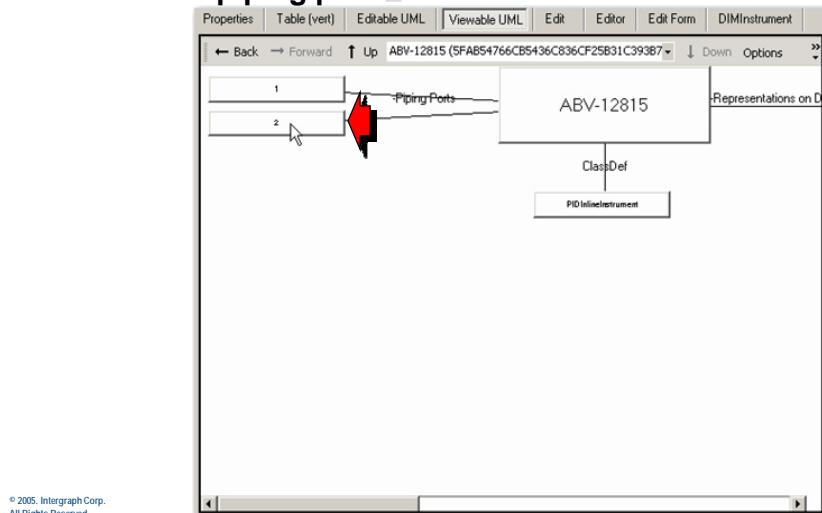
- Use the scroll bar to scroll right and see the related objects.



Click any of the related objects in the UML view to display the details for that object.

Viewing a Data File – Tree/Viewable UML

- Click piping port **2** to view the details for this component.



Continue to drill down in the tree view to see more data.

Viewing a Data File – Tree/Viewable UML

Click the + next to **ABV-12815** to view the related data.

The screenshot shows a software interface with a title bar 'INTERGRAPH'. Below it is a tree view of data classes. A red arrow points to the '+' sign next to the class 'ABV-12815'. The tree structure includes:

- PIDDrawing
- PIDInlineInstrument
- ABV-12815
 - Cables
 - Centrifugal Pump
 - Class Definition** (highlighted in blue)
 - Conductors
 - Connection Items
 - Connections
 - Control System IO Points
- Drawings
 - Elec Distr Assys
 - Electrical equipment
 - Electrical Trace
 - Function Block
 - Instrument Loop
 - Item Notes
 - Item Notes
 - Owner
 - Part
 - Parts
 - Parts
- Piping Ports
 - Power Consumers
 - Power Suppliers
 - Process Data (Points)
 - Process equipment
 - Process Vessel
 - Process Vessel
 - Process Vessel
 - Signal Ports
- ABV-12827

© 2005, Intergraph Corp.
All Rights Reserved.

Viewing a Data File – Tree/Viewable UML

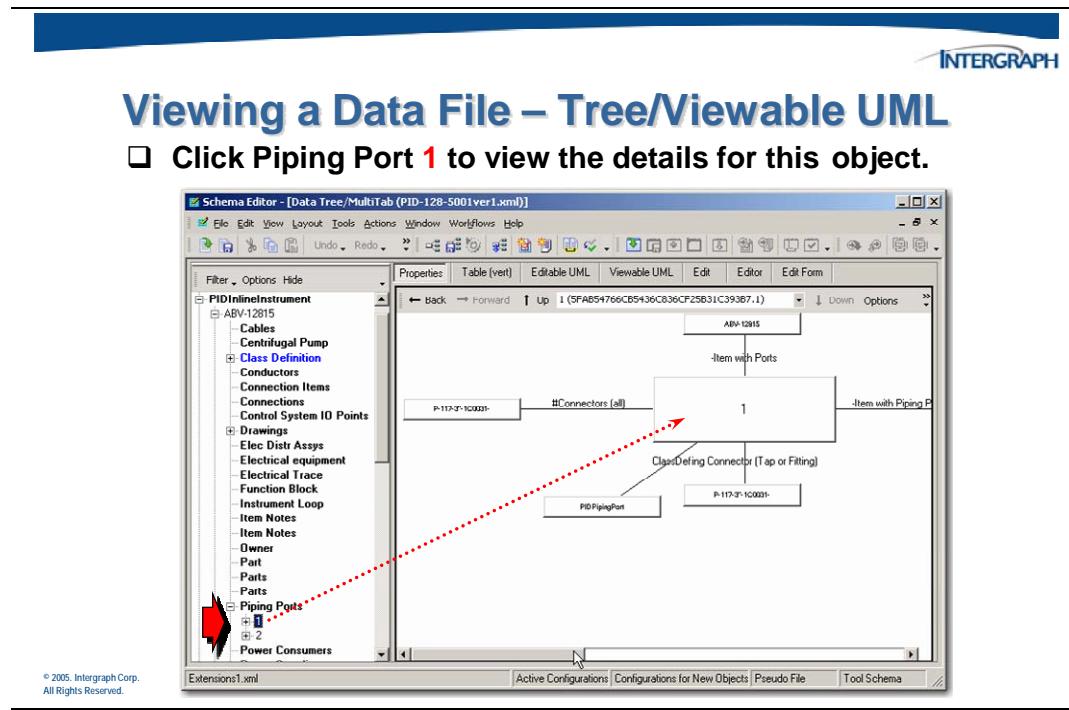
Continue to drill down by clicking the + next to **Piping Ports** to view related data.

The screenshot shows a software interface with a title bar 'INTERGRAPH'. Below it is a tree view of data classes. A red arrow points to the '+' sign next to the class 'Piping Ports'. The tree structure includes:

- Select other classes
- Options
- End 1 Relationships
- End 2 Relationships
 - Function Block
 - Instrument Loop
 - Item Notes
 - Item Notes
 - Owner
 - Part
 - Parts
 - Parts
- Piping Ports
 - Power Consumers
 - Power Suppliers
 - Process Data (Points)
 - Process equipment
 - Process Vessel
 - Process Vessel
 - Process Vessel
 - Signal Ports
- Source Graph Edge Rela
- PIDNozzle
- PIDPipeline
- PIDPipingConnector
- PIDDimension

© 2005, Intergraph Corp.
All Rights Reserved.

Select the object to be viewed in the right pane.



6.2.2 Data Tree/Table View

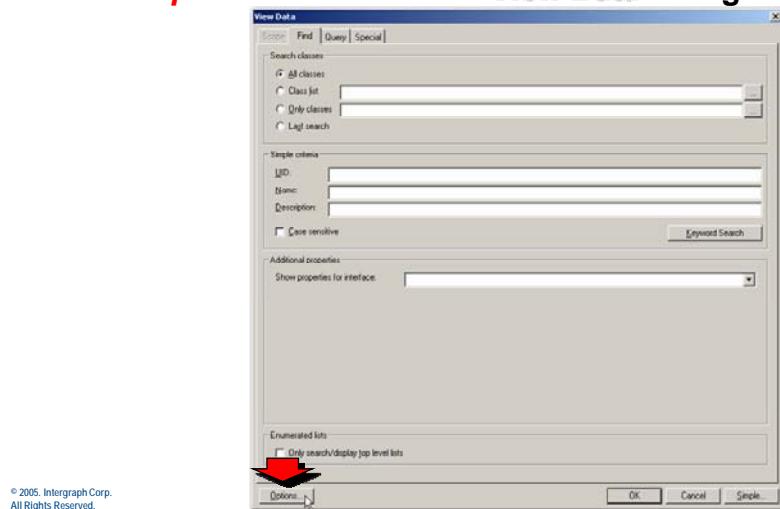
Like the Schema Tree/Table view, the Data **Tree/Table** view allows you to traverse relationships in the tree and see information for items selected in the tree view in a table format.

There are two table views, *vertical* and *horizontal*. Published data is best viewed with the *vertical* table, but that view is not activated by default. To activate it, click the Options button on the *View Data* dialog box.



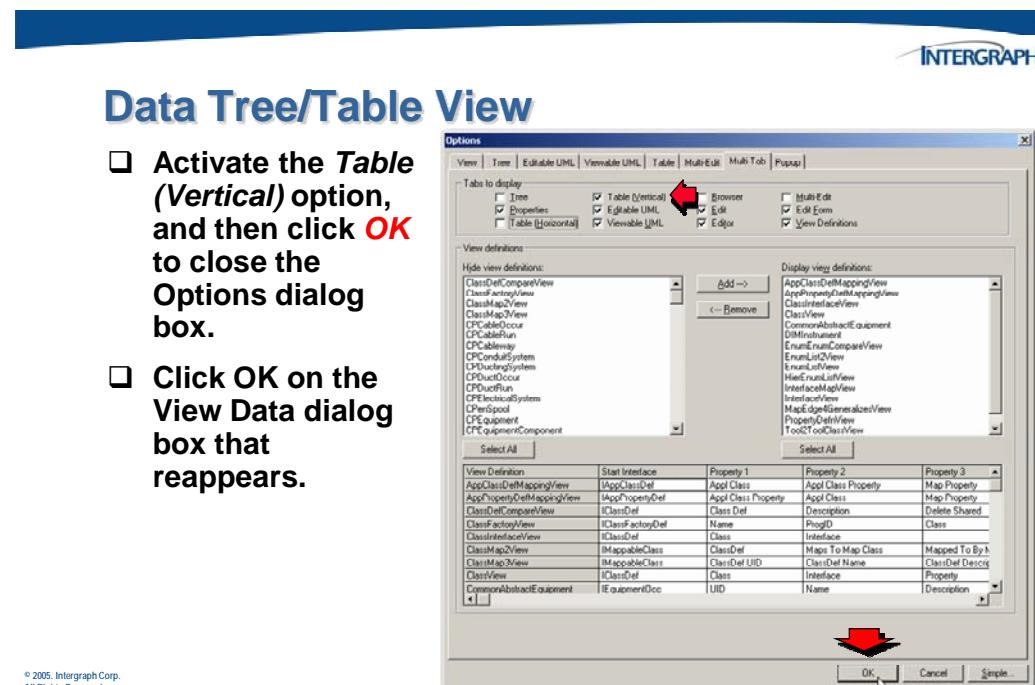
Data Tree/Table View

- To add the *Tree/Table Vertical* view to the tabs, click the **Options** button on the *View Data* dialog box.

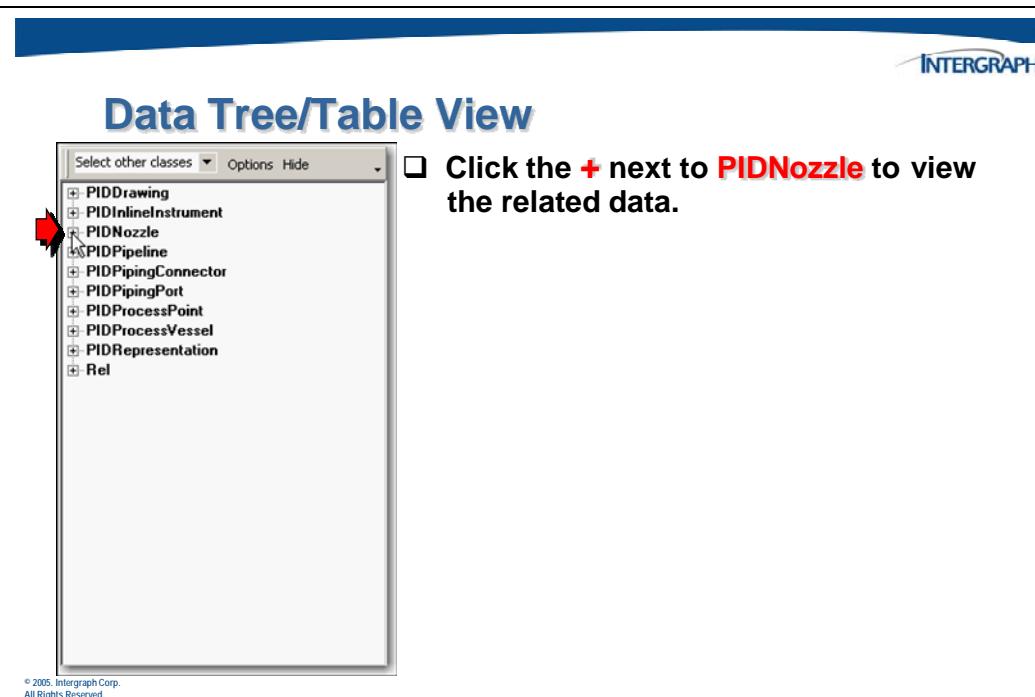


© 2005, Intergraph Corp.
All Rights Reserved.

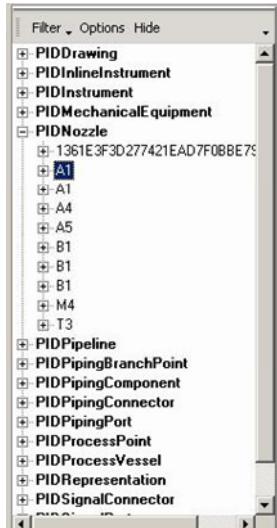
Once the **Options** dialog appears, activate the **Table (Vertical)** display. If you wish, you may also de-activate the **Table (Horizontal)** view, but this is not necessary. Click **OK** to close the **Options** dialog box, and again to close the **View Data** dialog box.



The *Data Tree/Table* window appears.

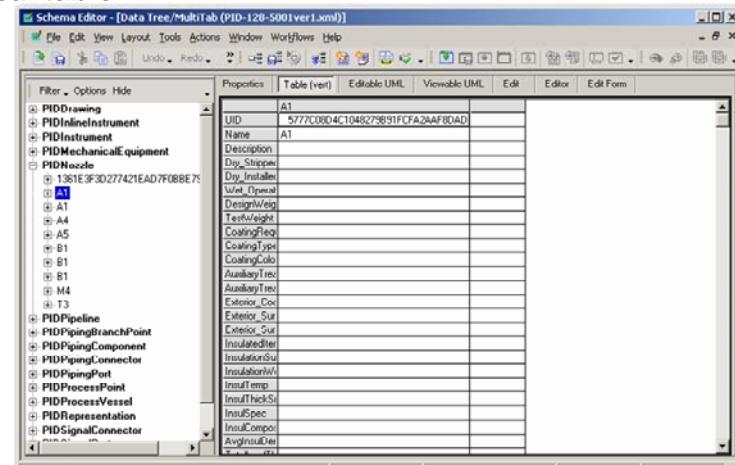


To traverse the relationships in the tree view, expand the nodes.



Click **A1** to view the details in the right pane.

The vertical table displays row headers along the left side of the table and data in vertical columns. In the following example, a **Vertical** table display is used.



	Properties
A1	UID Name Description Dry_Stripper Dry_Installer Vel_Operat DesignWeig TextWeight CoatingReq CoatingType CoatingColo AusakayTire AusakayTire Exterior_Cox Exterior_Sur Exterior_Sur InsulatedSur InsulationSur InsulationW InnlTemp InnlThickS InnlSpec InnlCompo AvgInsula

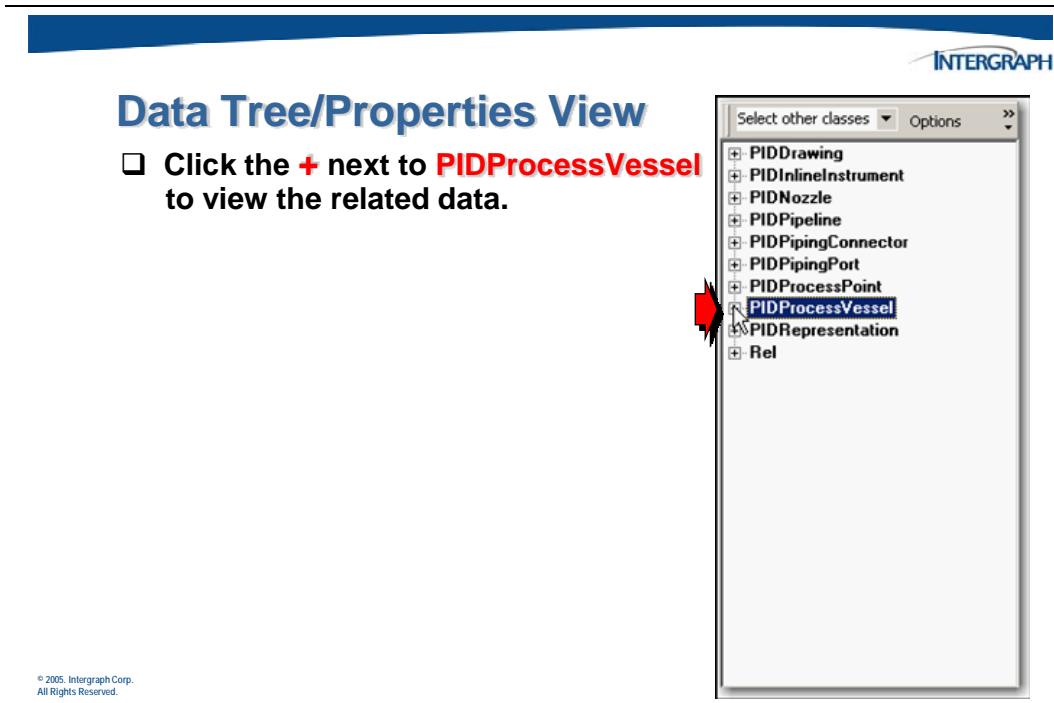
6.2.3 Data Tree/Properties View

The **Tree/Properties** view combines the same tree view used elsewhere in the Schema Editor with a properties view that emulates display of properties in the right pane in SmartPlant Foundation. This properties view identifies each interface for the selected object with text on a shaded background. Following each interface are the properties exposed by that interface and their values for the selected object.

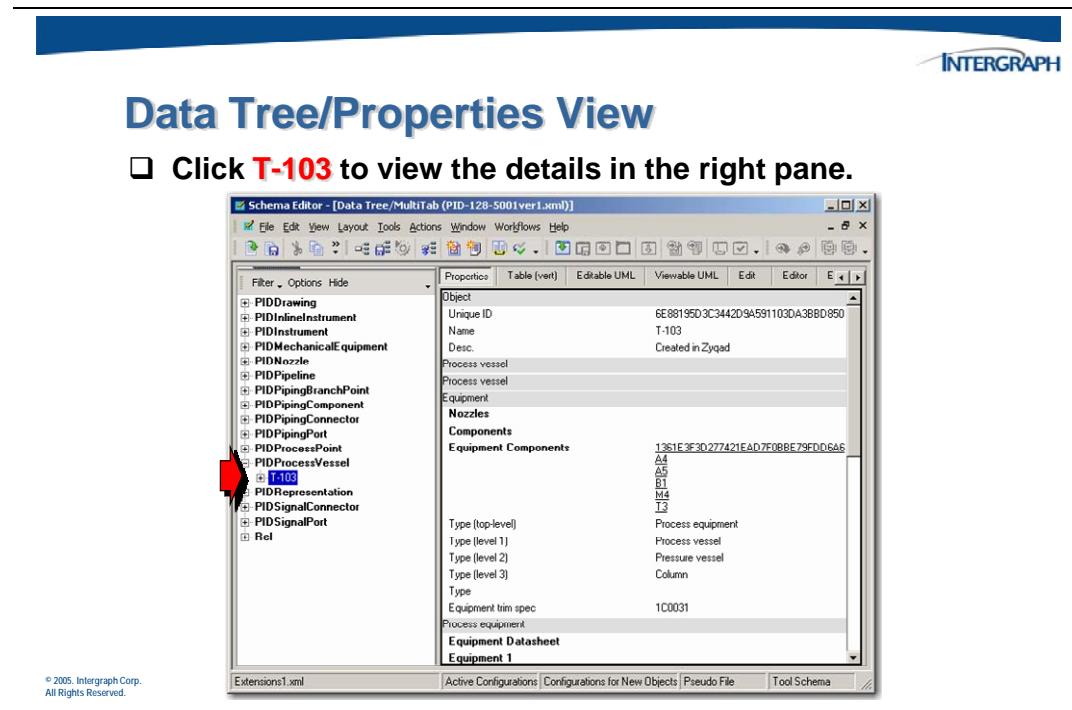
The properties view is currently the only view in the Schema Editor that explicitly converts enumerated property values from the UID of the enumerated entry to the name associated with that entry. Other views display the UID for the enumerated entry as it would appear in the XML file, such as @EE407.

The **Tree/Properties** view provides the most useful information when you use it to view data, instead of the schema, tool schemas, or meta schema. If you select an object in the tree view, you can see its interfaces in the properties view.

From the tree, find an object to view. Drill down to expand the tree to see other objects.



Select the object that you want to see in the **Properties** view.



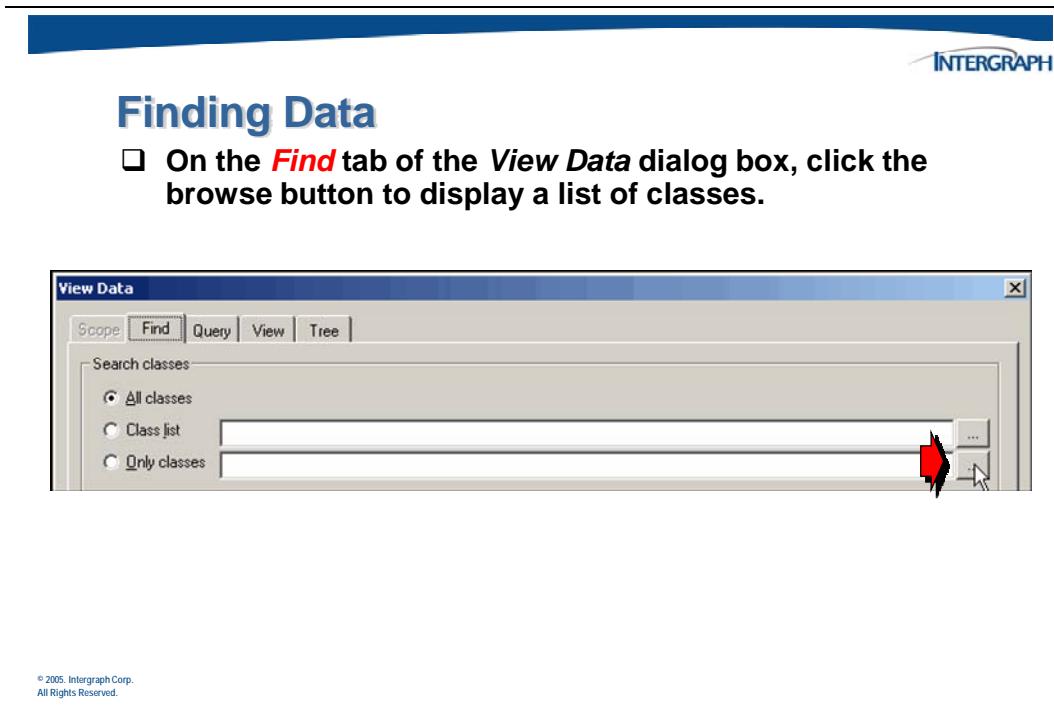
6.3 Finding Data Objects

In the Schema Editor, you can locate particular objects in data files by defining search criteria for the object that you want to find, including the name and description of the object and its UID.

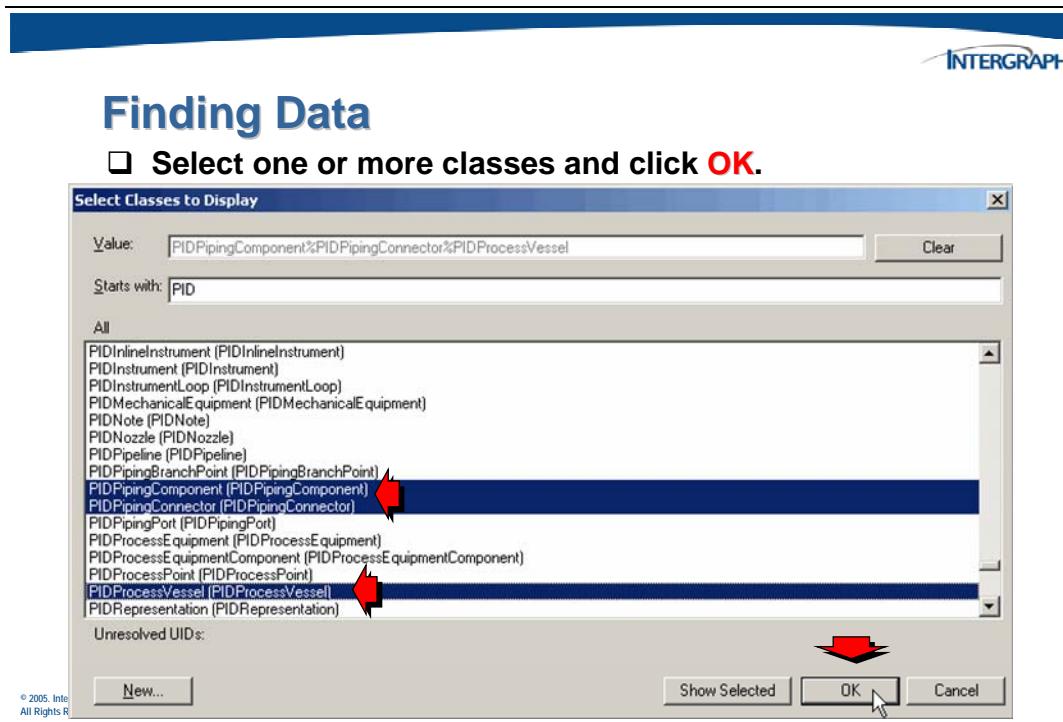
6.3.1 View Data Find Tab

To find an object in the Schema Editor, choose either the **Edit > Find** menu command or use the **Find** tab in the *View Data* dialog box. This allows you to define search criteria and to limit the scope of your search to the data files. Using the *Find* dialog box is especially useful if you know part of the UID, name, or description of the object that you want to find.

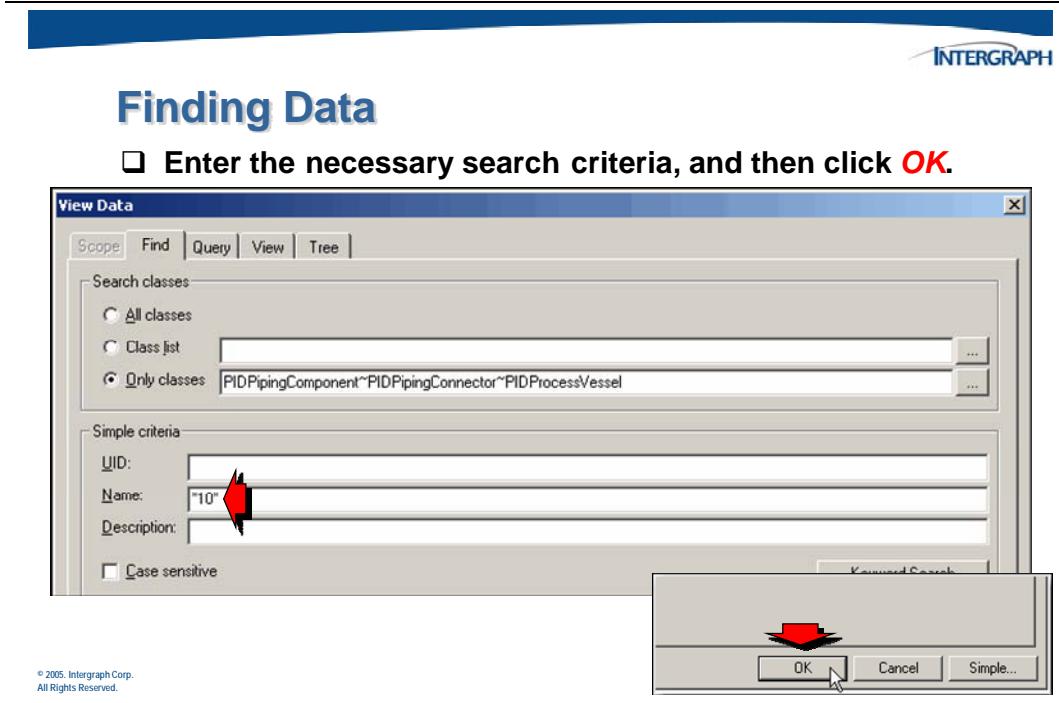
In the following example, click the *Data File View* button first, then click the **Find** tab in the *View Data* dialog box.



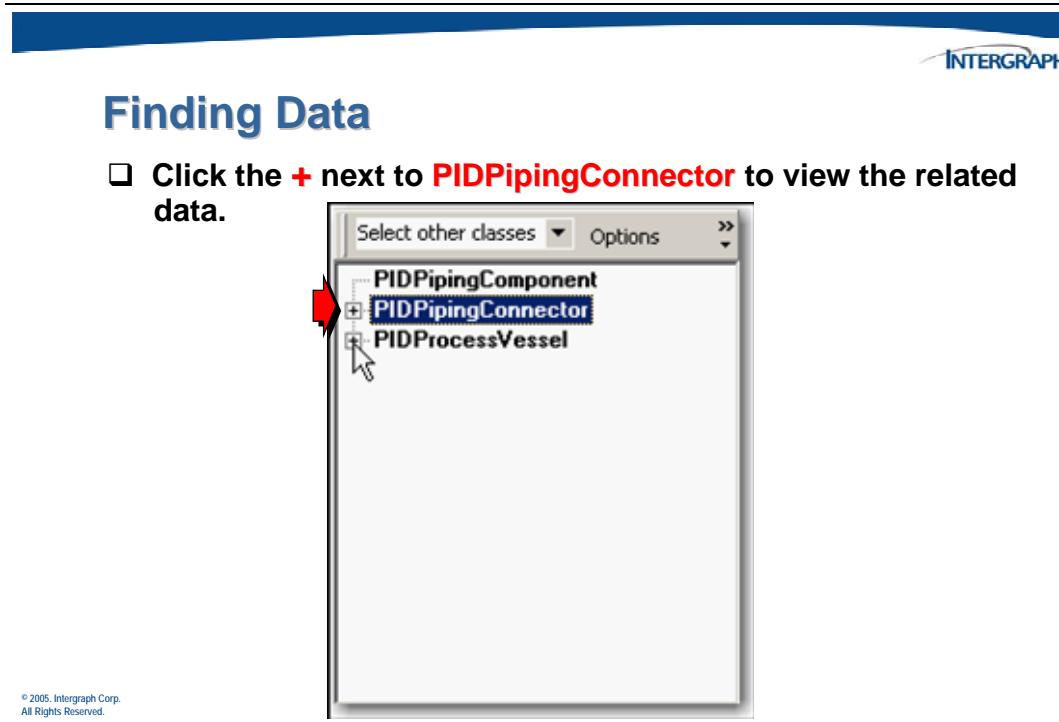
The *Select Classes to Display* dialog box appears.

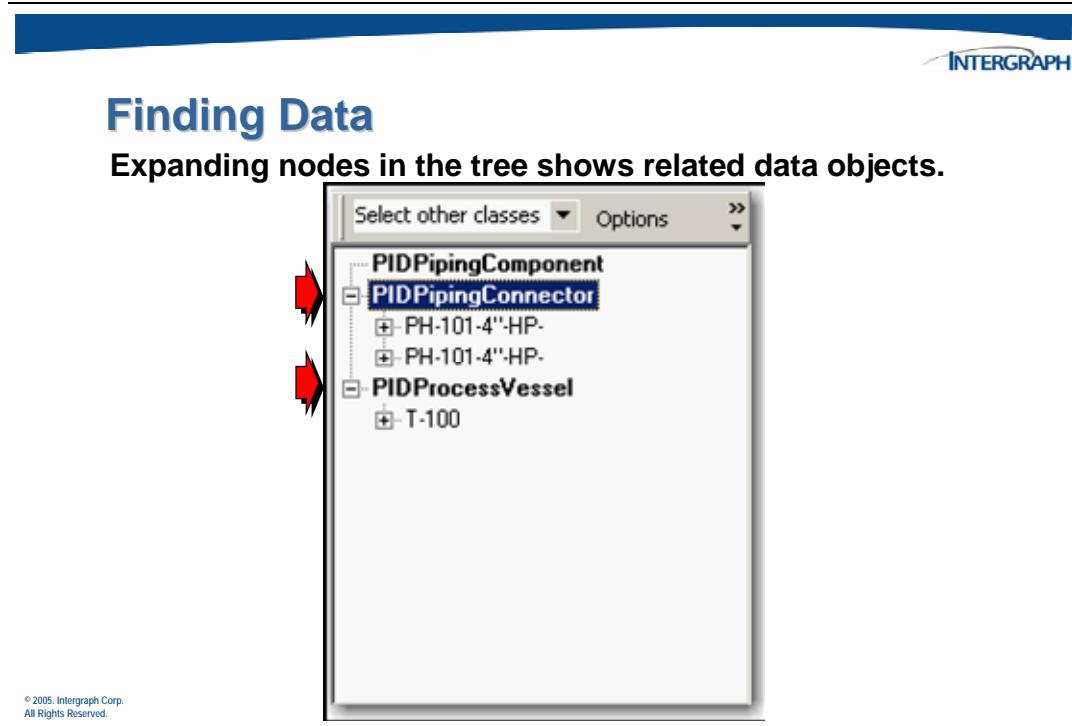


In this example, the software searches for all *PIDPipingComponent*, *PIDPipingConnector* and *PIDProcessVessel* objects that have *10* in the *Name*.

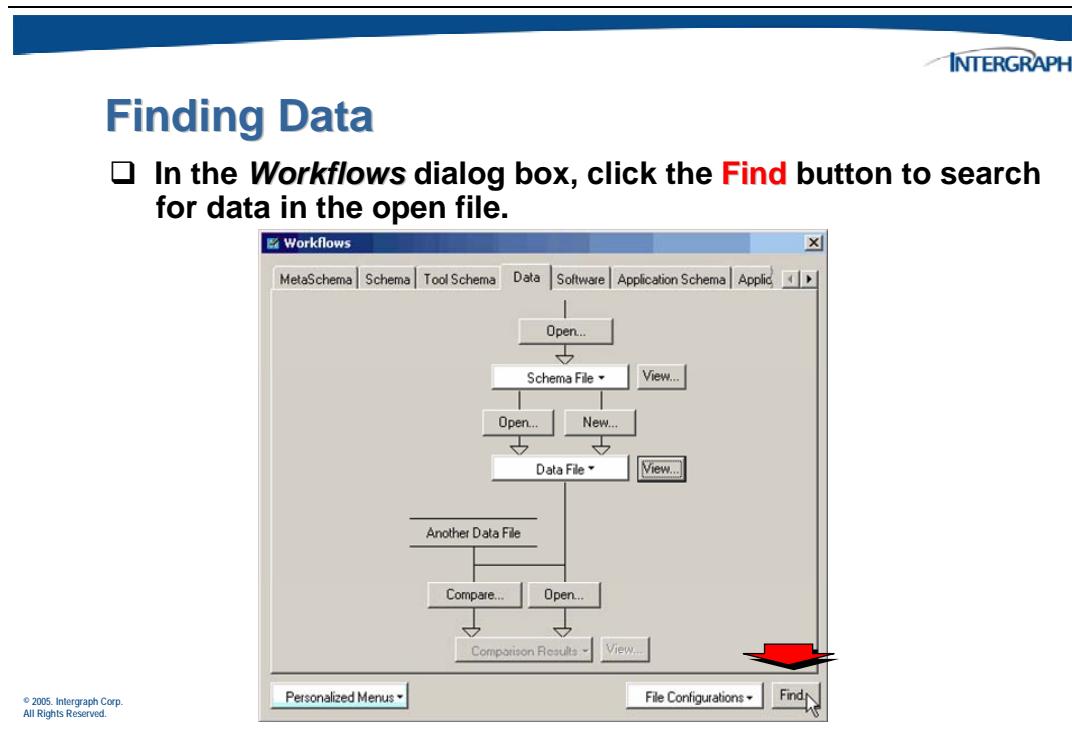


A Data Tree/Properties view displays the results of the search. Expand the tree to see related data.





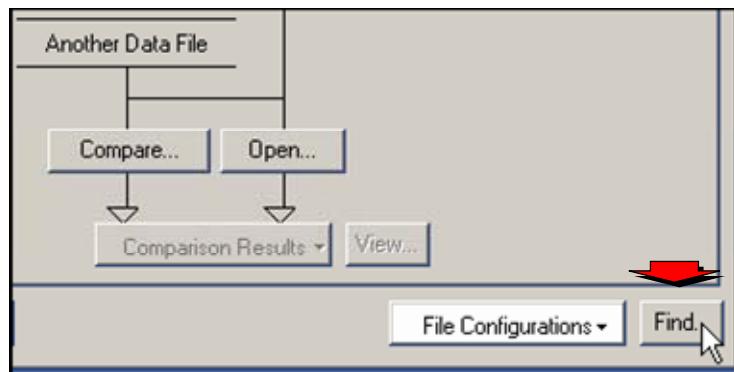
In the following example, the **Find** button in the *Workflows* dialog box is used to find objects.





Finding Data

Zoomed in view of the **Workflows** dialog box.



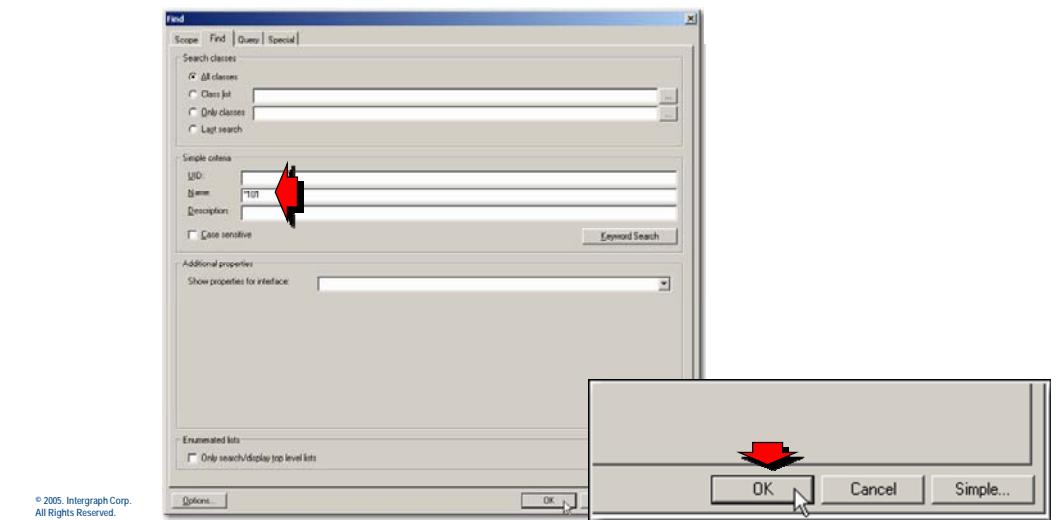
© 2005, Intergraph Corp.
All Rights Reserved.

On the **Find** tab and enter the search criteria to locate the desired data.



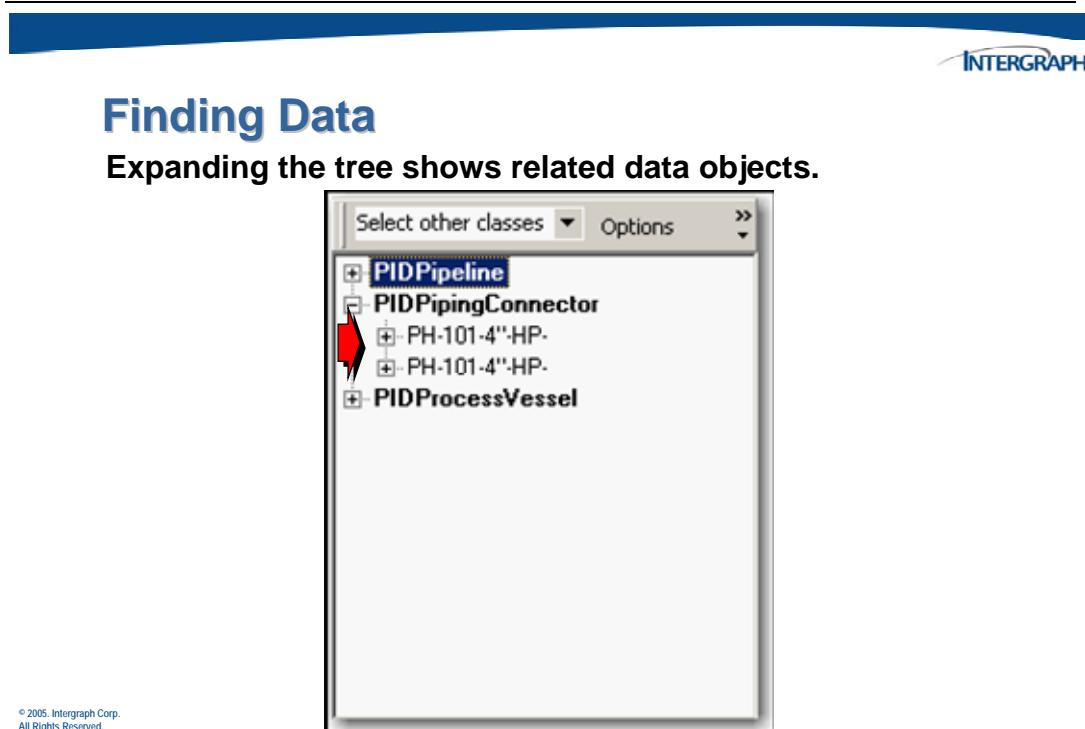
Finding Data

- Enter the necessary search criteria, and then click **OK**.



© 2005, Intergraph Corp.
All Rights Reserved.

In this example, the software searches for all objects that have ***10*** in the *Name* from *All classes*. A Data Tree/Properties view displays the results of the search.



6.4 Activity – Viewing and Finding Data

Complete the Chapter 6 – Activity 1 in the SmartPlant Foundation 2008 (4.2) Modeling and Mapping Activity workbook.

7

C H A P T E R

Introduction to Schema Mapping

7. Introduction to Schema Mapping

Schema Mapping provides a mechanism for the software to convert authoring tool data, described by the *tool map schema*, to SmartPlant data, described by the *SmartPlant schema*, and SmartPlant data back to tool data. Each tool that integrates with SmartPlant supplies the mapping between that tool and the SmartPlant schema for both publish and retrieve operations. The mapping is stored in the authoring tool map schema.

Mapping allows authoring tools to externalize the definition of mapping to isolate authoring tools from changes in the SmartPlant schema. This mapping defines correlations to the SmartPlant schema strictly using UIDs, which should not change.

In the Schema Editor, you can modify how authoring tool data maps into and out of the SmartPlant schema by modifying the authoring tool mapping. Common modifications to existing tool mapping are the extension of ***tool enumerated lists*** that are then mapped to enumerated lists in the SmartPlant schema and the addition of custom properties.

In previous releases, mapping from the “tool schema” to the SmartPlant schema was done based on the underlying SmartPlant schema objects (class, interface, property, and edge definitions). This required the person doing the mapping to be intimately familiar with that part of the overall SmartPlant schema. Feedback from SmartPlant users indicated that this was a burdensome requirement, in that it required significant effort for an individual to attain the appropriate level of knowledge of the SmartPlant schema.

To reduce the effort and knowledge required of the tool integrator defining the mapping, a simpler, view-based approach is now available. This presents the user with a “business object” view of the schema that is much easier to understand and utilize.

To simplify the mapping and to enable the definition of “transformations” associated with the mapping, the GUI was enhanced to use a more graphical (versus tabular) approach to the mapping. The overall complexity of the previous user interface was replaced by a more intuitive, graphical approach.



Overview of SmartPlant Mapping

The Schema Editor allows you to perform mapping between *Tool Map Schemas* and the *SmartPlant Schema*:

- The user interface used for mapping was specifically designed to allow users to create mapping relationships, or even to extend the SmartPlant Schema as necessary, in an easy-to-use window.
- It incorporates ViewDefs and existing mapping relationships to suggest possible SmartPlant Schema changes.

© 2008, Intergraph Corp.
All Rights Reserved.



Overview of SmartPlant Mapping

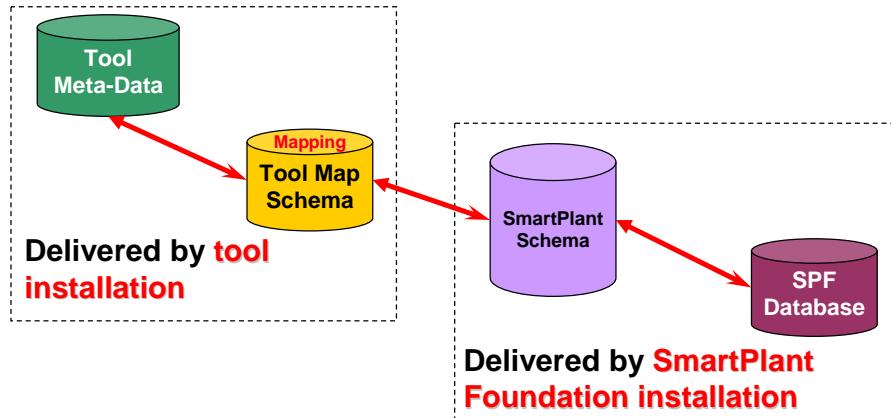
- From within the Schema Editor tool, you can access the Metadata Adapters that allow you to synchronize your tool database structure and the applicable tool map file.
- Special features even allow you to automate the process of creating and mapping new properties in the SmartPlant Schema.

© 2005, Intergraph Corp.
All Rights Reserved.



Mapping with the Schema Editor

Any changes can be added to the tool meta-data (authoring tool data base), then the tool map schemas and SmartPlant schemas, and finally mapped in the **tool** map schema.



© 2005, Intergraph Corp.
All Rights Reserved.



Mapping with the Schema Editor

Every class in the tool map schema that is to be published to SmartPlant maps to a **class** or **interface definition** in the SmartPlant schema.

Properties that are of interest to upstream applications are also mapped to the appropriate properties in the SmartPlant schema.

© 2005, Intergraph Corp.
All Rights Reserved.



Mapping with the Schema Editor

In each tool map schema, the **MapClassToClass** and **ClassToMapClass** relationship definitions identify the mapping from the tool's class definition to a corresponding object in the SmartPlant schema and the mapping from the SmartPlant schema to the tool schema.

The mapping in one direction may not necessarily match the mapping in the other direction.

© 2005, Intergraph Corp.
All Rights Reserved.



Mapping with the Schema Editor

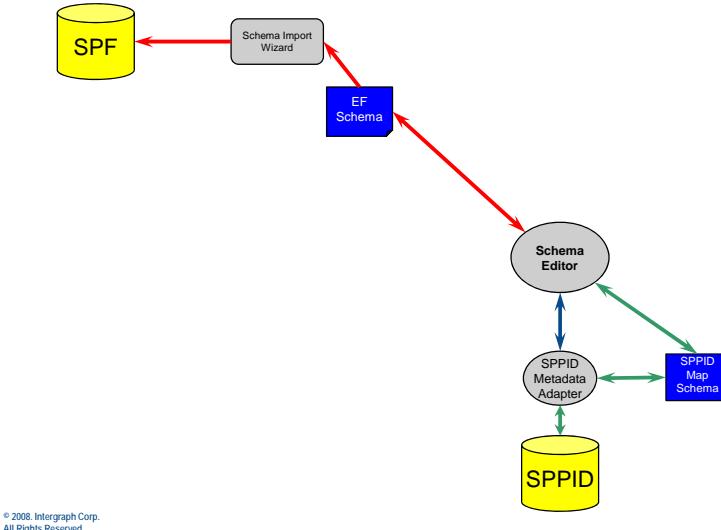
The mapping between property definitions in the tool schema and their counterparts in the SmartPlant schema is done using the **MapPropertyToProperty** and **PropertyToMapProperty** relationship definitions.

If a property definition is enumerated or has units of measure, it has a relationship of type **MapPropertyMapEnumList** that relates it to the appropriate enumerated list or unit of measure list.

© 2005, Intergraph Corp.
All Rights Reserved.

Data originating from an authoring tool's database (metadata) can now be manipulated from the Schema Editor using a **metadata adapter** delivered with each individual authoring tool.

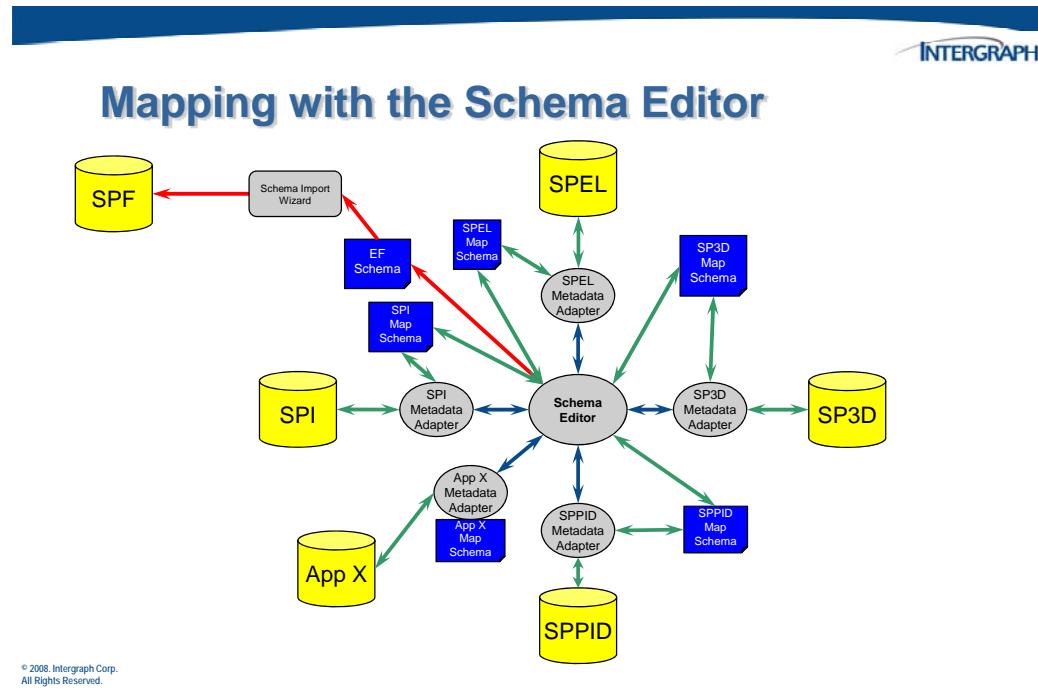
Mapping with the Schema Editor



The metadata adapter can be used to read the data definitions in the tool meta-schema. If new definitions are found in the authoring tool database, the metadata adapter can extract these new definitions and automatically add them to the tool map schema. This is called synchronizing the tool meta-schema and the tool map schema and occurs when the Schema Editor connects to SmartPlant and the tool map schema is opened.

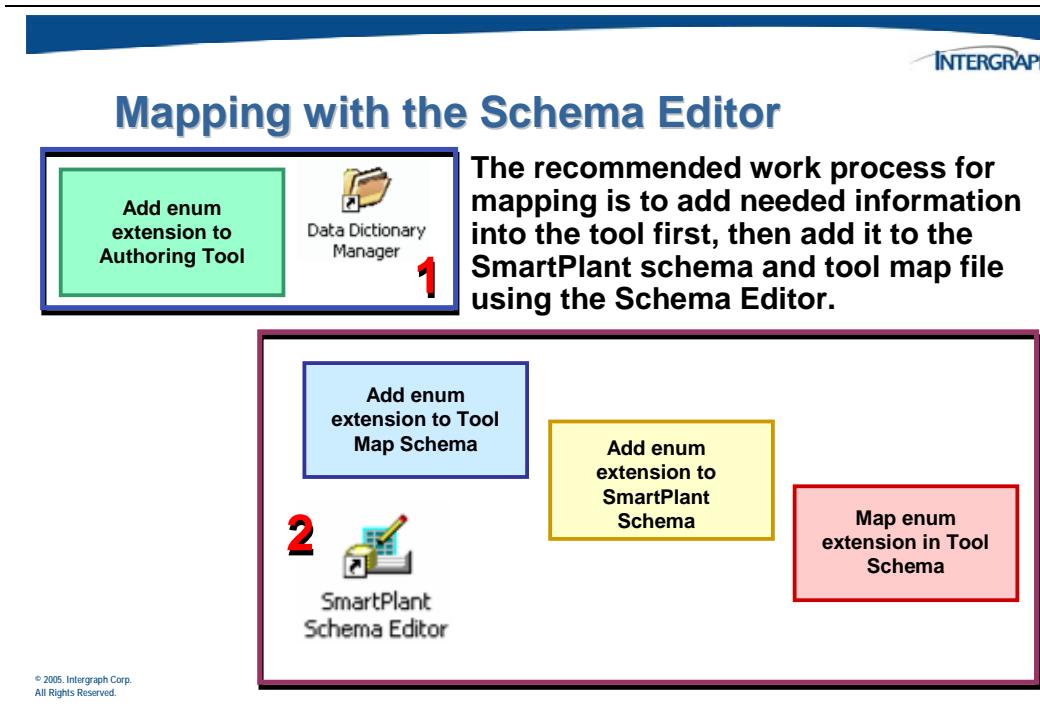
After the tool meta-schema and tool map schema have been synchronized, new definitions can be added to the SmartPlant schema (EFSchema) and then mapped via the Schema Editor.

Every tool uses a metadata adapter as illustrated by the figure below.

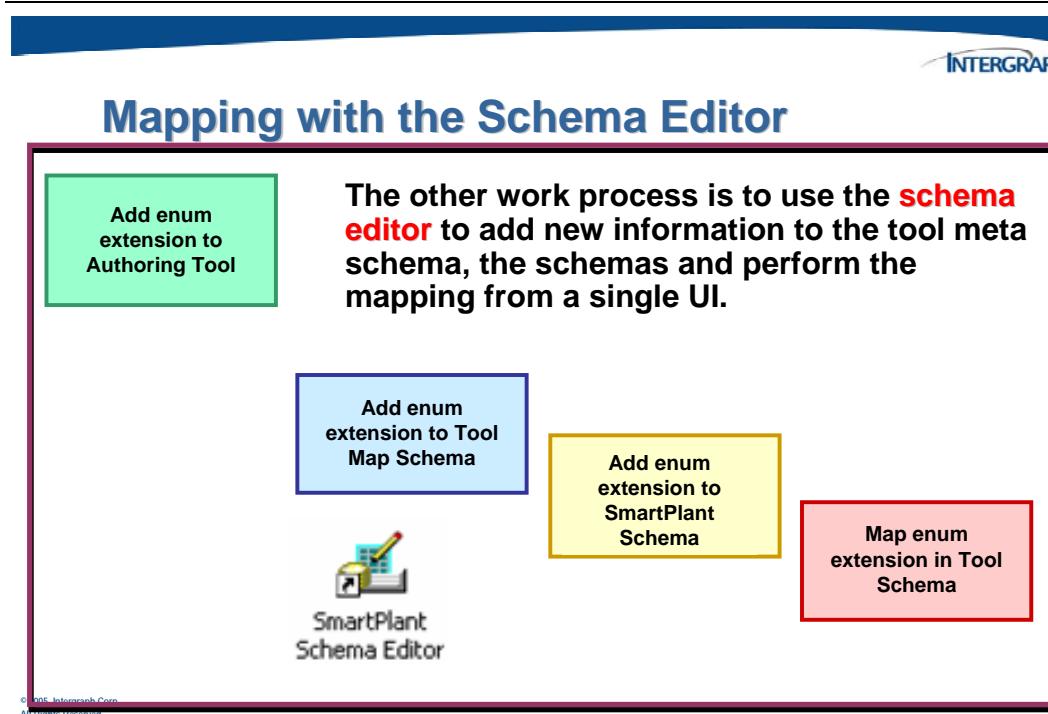


7.1 Mapping Process Options

Mapping data from the authoring tool to SmartPlant and back requires one of the following work processes. In some cases, extensions to the authoring tool already exist. To complete this process, the same extensions have to be integrated throughout the schemas.



Another possibility is to use the schema editor and the metadata adapter to add any new needed definitions to the schemas and the tool meta-schema from a single user interface, the *Schema Editor Map Environment*.



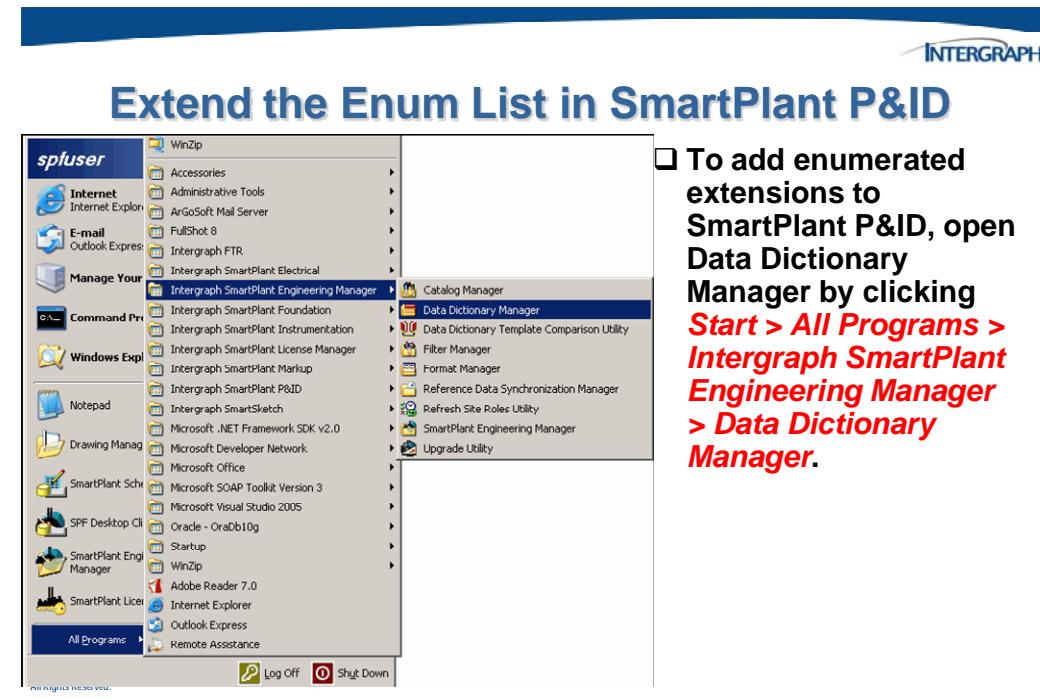
Configurable mapping, in which the authoring tool supplies a tool schema that defines the mapping between the tool schema and the SmartPlant schema and back, is not required for all tools that participate in SmartPlant. Authoring tool developers can hardcode the tool mapping in the tool adapter, although doing so is not recommended. The example in this chapter is only relevant for tools that implement configurable mapping.

7.2 Mapping Example – Extending an Enumerated List

The most common extension made to the SmartPlant Schema is the addition of new enum enums for existing enumerated lists that are mapped for publishing to or retrieving from an authoring tool. This section illustrates an example of extending an existing select list in the SPPID authoring tool and mapping the change for publish to SmartPlant Foundation.

7.2.1 Extending the Authoring Tool Enumerated List

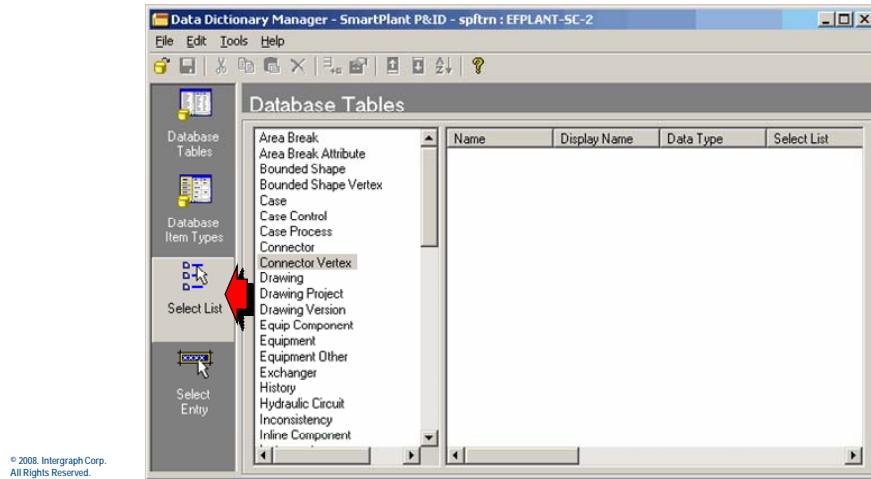
The procedure for adding data definitions to authoring tools differs from tool to tool. The following example describes the procedure for extending an existing *Select List* (enumerated list) in the SmartPlant P&ID data model using SmartPlant Data Dictionary Manager, delivered with SmartPlant Engineering Manager. For more information about adding properties to SmartPlant P&ID, see the SmartPlant Data Dictionary Manager documentation.



The Data Dictionary will appear as illustrated below.

Extend the Enum List in SmartPlant P&ID

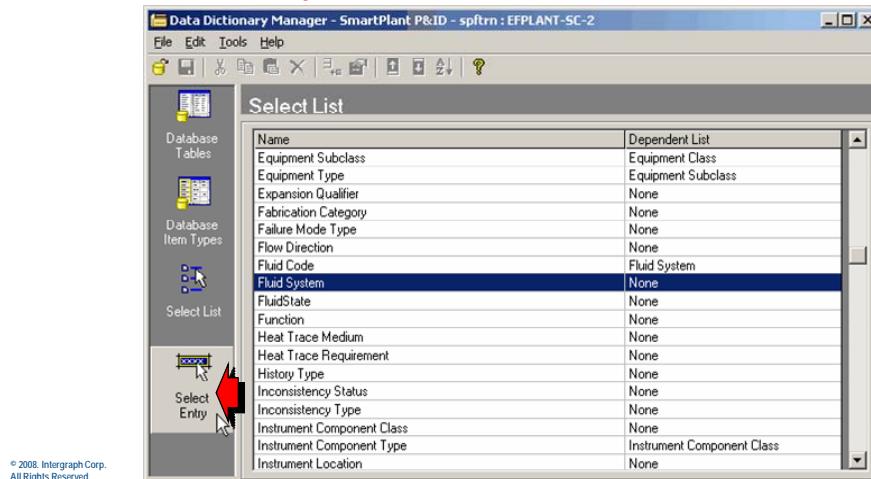
- To view the existing Select Lists, click the **Select List** button on the left side of the window.



The list of defined Select Lists appears.

Extend the Enum List in SmartPlant P&ID

- To define a new entry for an existing select list, click the **Select Entry** button.

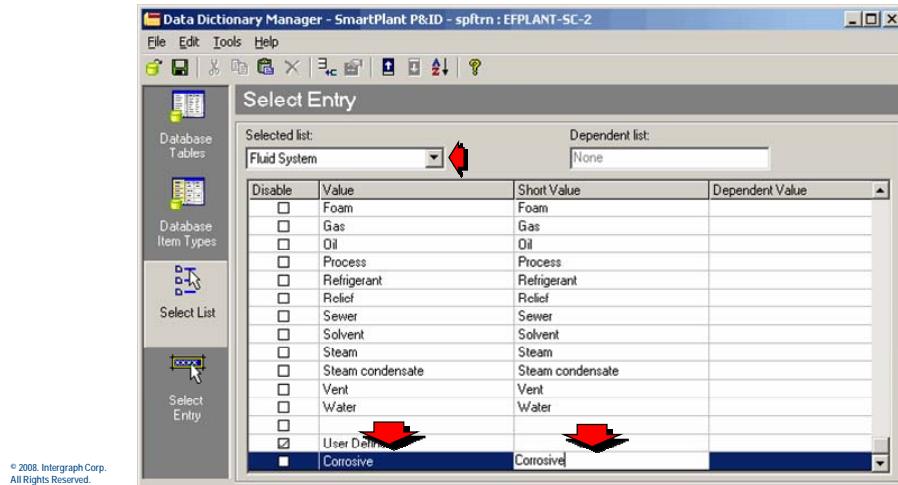


In the **Select Entry** list, scroll to the bottom of the list to the empty row, and add an entry. Type **Corrosive** as the **Value** and **Short Value**.



Extend the Enum List in SmartPlant P&ID

- Choose the **Fluid System** existing select list in the **Selected list** box and enter the new value.

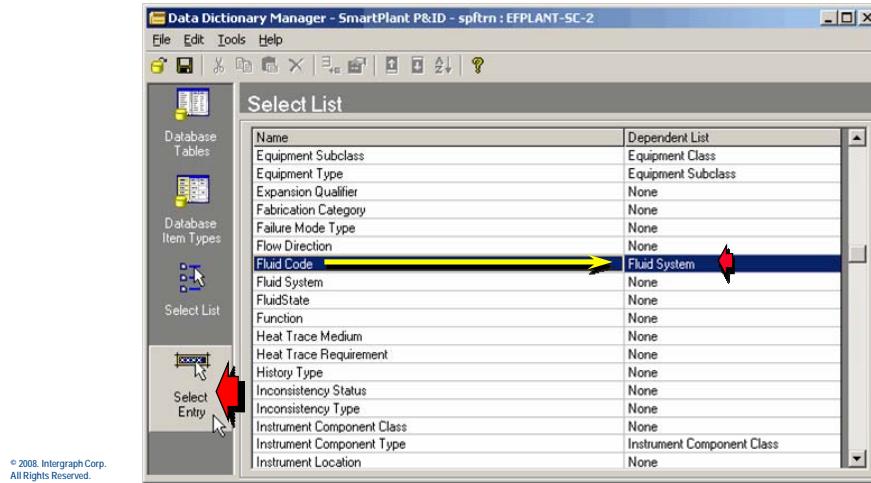


© 2008, Intergraph Corp.
All Rights Reserved.

The **Fluid Code** select list is dependent upon the **Fluid System** select list. Next, add new fluid code values that will be available when the fluid system is **Corrosive**.

Extend the Enum List in SmartPlant P&ID

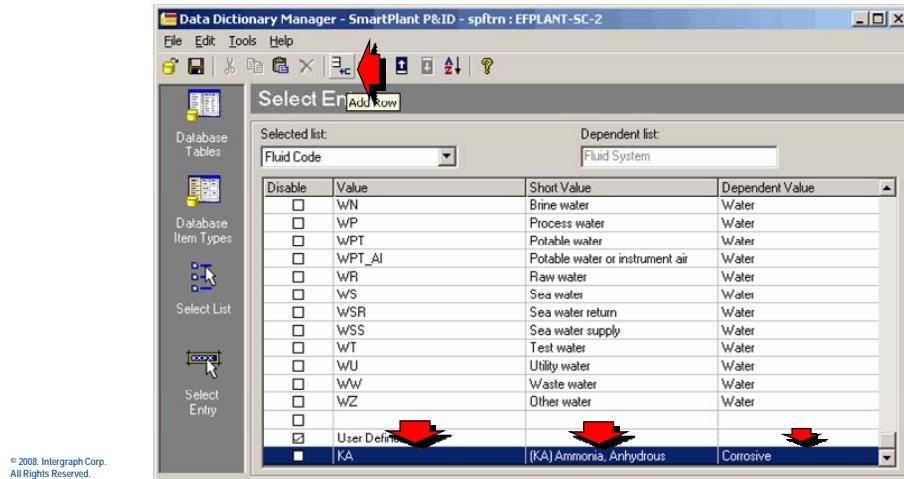
- To define a new entry for the next dependent select list, click the **Select Entry** button once more.



Select the **Fluid Code** select list and open the **Select Entries** view. Scroll to the bottom of the window to the empty row and add the new fluid code values.

Extend the Enum List in SmartPlant P&ID

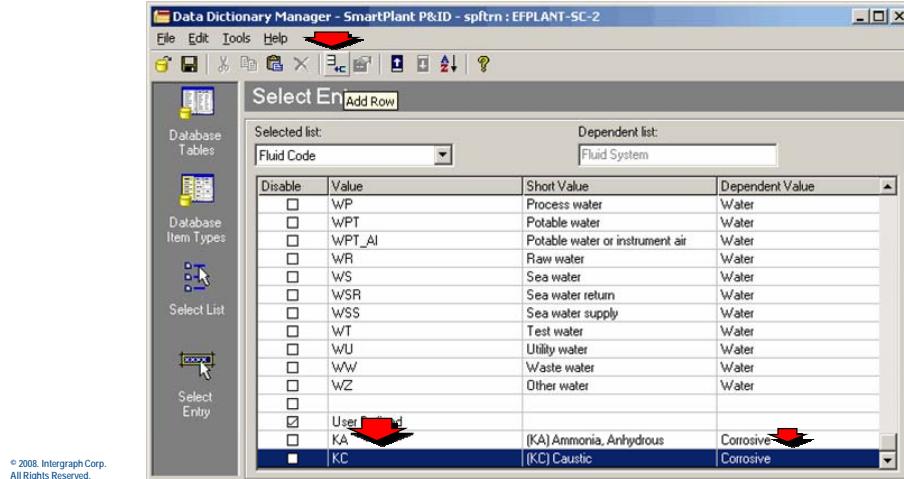
- Enter new **Value** and **Short Value** for the new **Fluid Code** entry then click the **Add Row** button to add another entry.



Click the **Add Row** button, and then add the next fluid code value.

Extend the Enum List in SmartPlant P&ID

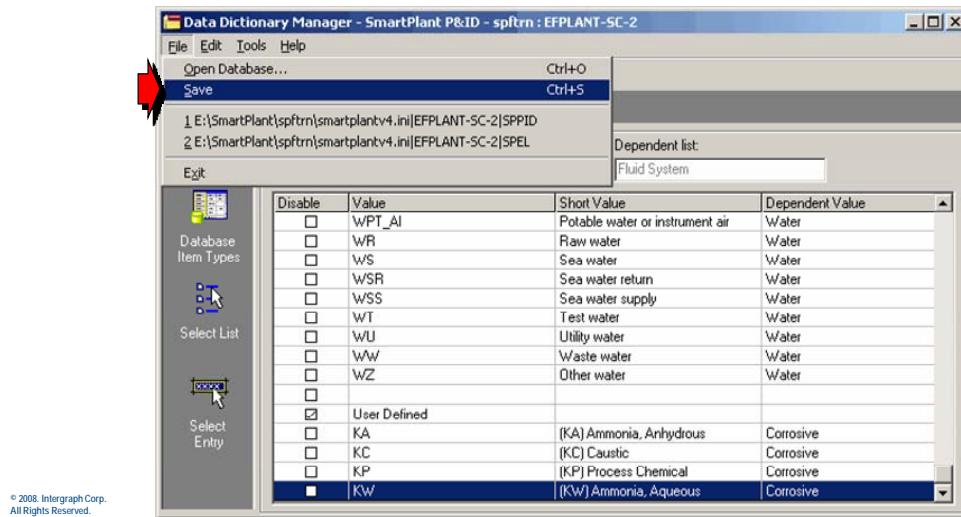
- On the toolbar, click the **Add Row** button to add another select entry.



When you finish adding your new fluid codes, save your changes.

Extend the Enum List in SmartPlant P&ID

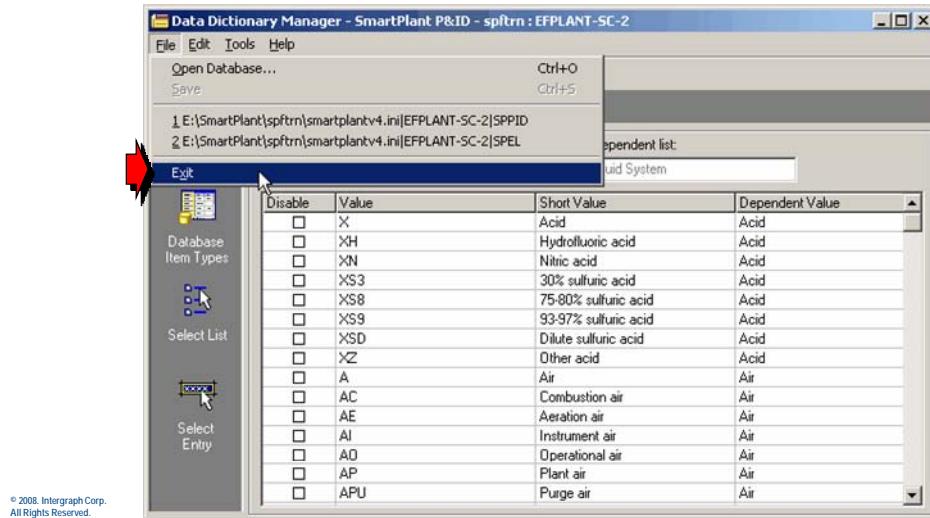
- From the menu, select **File > Save**.



Once the changes have been saved, close the Data Dictionary Manager.

Extend the Enum List in SmartPlant P&ID

- Select **File > Exit** from the menu.



7.2.2 Launching the CMF File

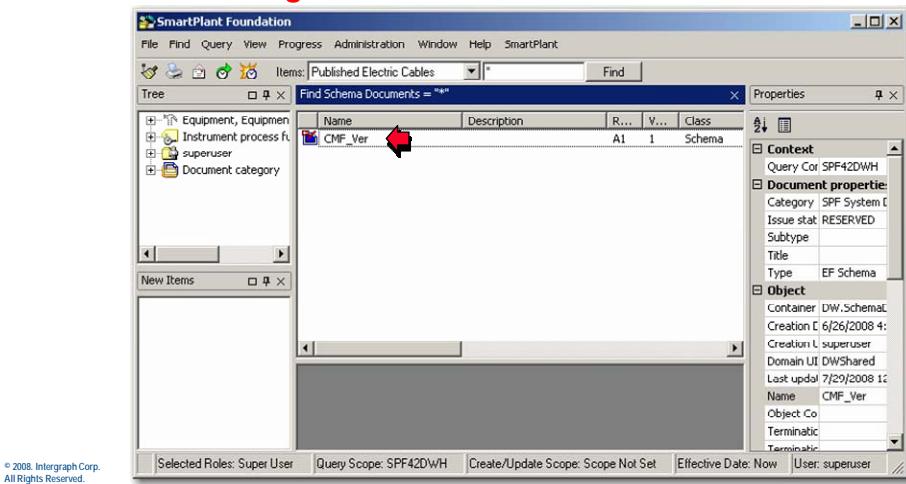
Before you can map a property, or in our current example, a select list value, from the tool schema to the SmartPlant Schema and back, you must use the Schema Editor to add the change to the SmartPlant schema.

The integration schema file is now stored in SmartPlant Foundation for change management purposes. Before you can change that file, you need to find it in the Desktop Client application, and check it out to make changes.

Find the schema, stored as a CMF file, using the ***Find > Integration > Schema Documents*** command.

Synchronizing the Tool Data and Map File

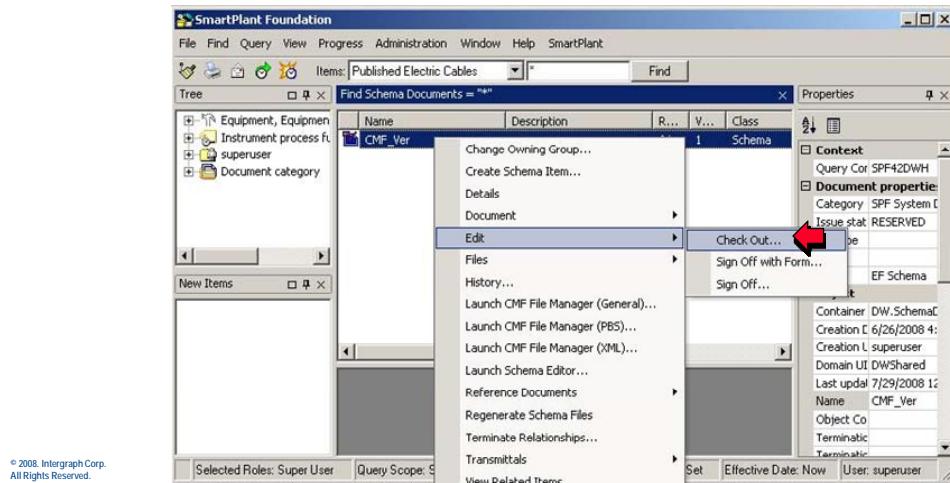
- Launch the Desktop Client and find the CMF file, using the *Find > Integration > Schema Document* command.**



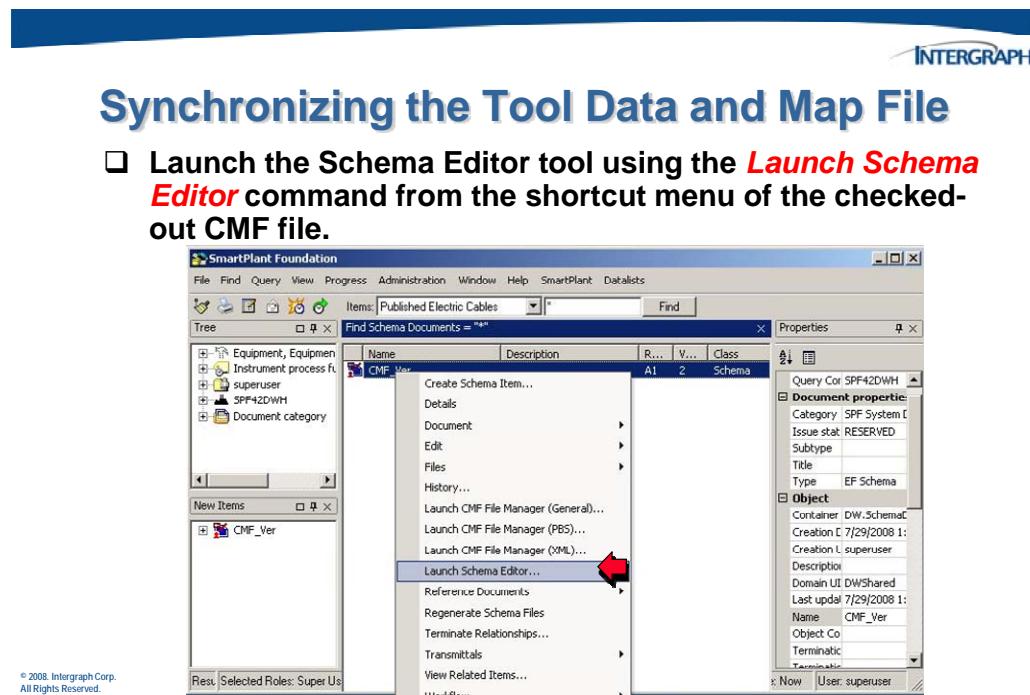
Once you find the file, use the standard **Edit > Check Out** command from the shortcut menu.

Synchronizing the Tool Data and Map File

- ❑ Using the **Edit > Check out** command from the shortcut menu, check out the CMF file.



Once you have checked out the file and copied the CMF file locally, right-click the checked out version of the CMF file and use the **Launch Schema Editor** command to open the Schema Editor, with the applicable connection information for this site.



This command opens the CMF file in the schema editor using a new session file. You can save the session file at any time. Once it is saved, you can close the Schema Editor and reopen it using that session file until you check the CMF file back into SPF.

When you launch the session file in the Schema Editor, you reopen that tool with the same connections and interfaces active as when you saved the file. For example, using the **Launch Schema Editor** command opens the schema editor, but it also connects to the appropriate schema files for the site in which you are working. Reopening the session file reconnects to that schema.

The **Set Active Configuration** dialog appears. The CMF file contains a number of different schemas. You must indicate which of these schemas you want to open and work with.

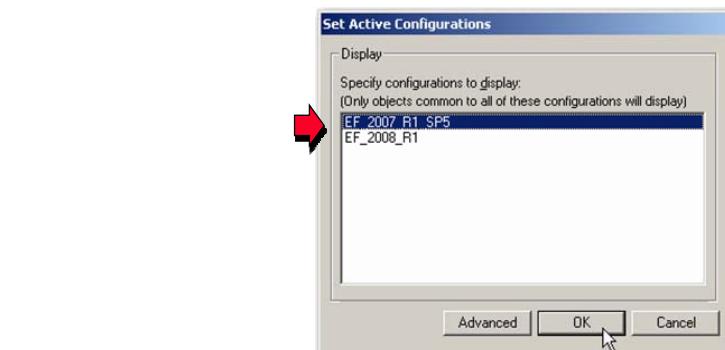
Since we are using tools that are version 2007 SP5, we want to open the 2007 SP5 schema rather than the 2008 version of the schema.

Note:

- You may select multiple schemas, if you wish, but remember that you will see only those objects that exist in ALL the schema selected. Objects that appear in some but not all of the selected schemas will not be displayed by the software.
- By default, the software saves all your changes to all the SmartPlant Schemas. If you want to change that for some reason, you can click the **Advanced** button and deactivate one or more schemas to which the changes will be made, or you will be prompted where to save changes when the save occurs. However, it is recommended that you save your changes to all schemas, as this will simplify later upgrades.

Synchronizing the Tool Data and Map File

- Indicate the schema you want to work in while using the Schema Editor.

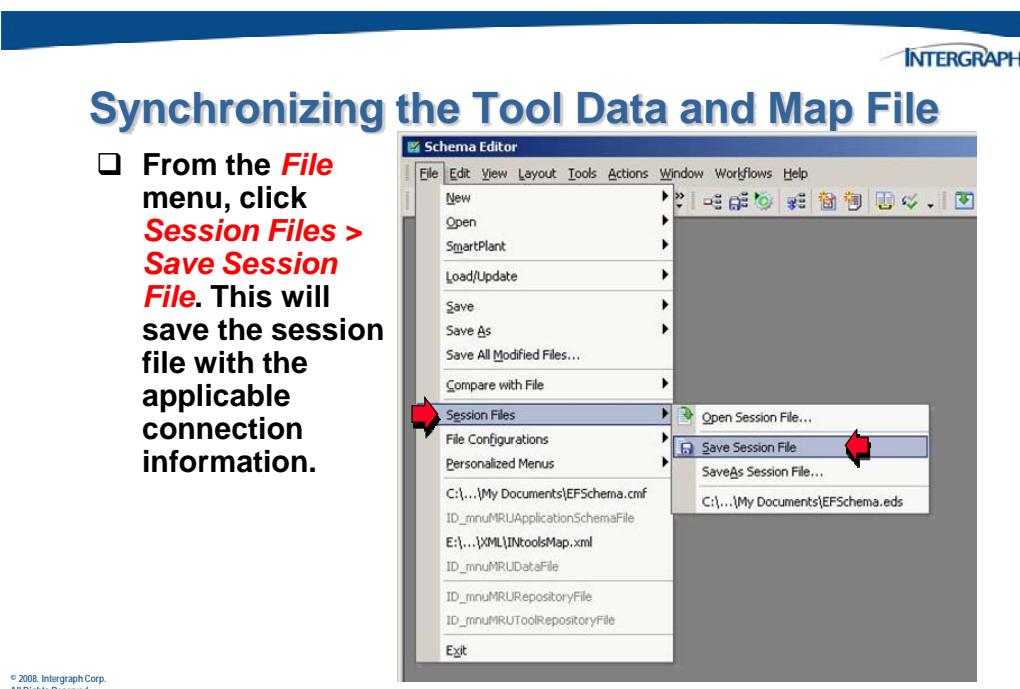


© 2008 Intergraph Corp.
All Rights Reserved.

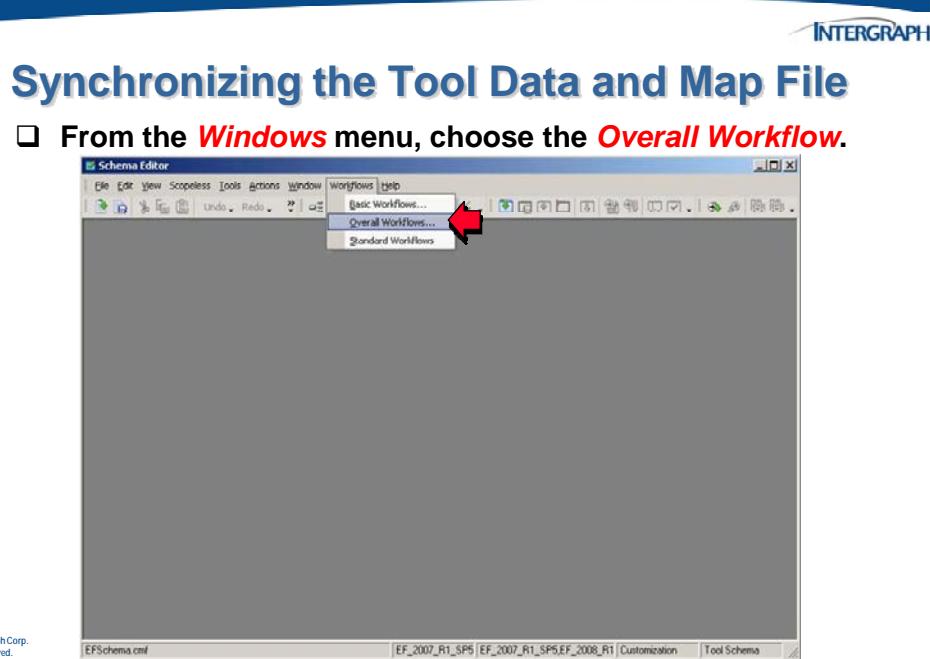
Once you have launched the Schema Editor tool, you may close the SmartPlant Foundation Desktop Client if you wish.

When the Scheme Editor application opens, click **File > Session Files > Save Session File** to save the session file with the following configuration. In the future, until you check the CMF file back into SPF, you will be able to launch the Schema Editor and use this session file to connect to the appropriate schema files without having to launch the file from the Desktop Client again.

In addition to the information necessary to connect the Schema Editor tool to the site's CMF file, the session file also contains information for connecting to the metadata adapters and tool map files for authoring tools with plants registered with plants in the site from which you launched the CMF file.



From the **Workflows** menu, choose the **Overall Workflows** option.

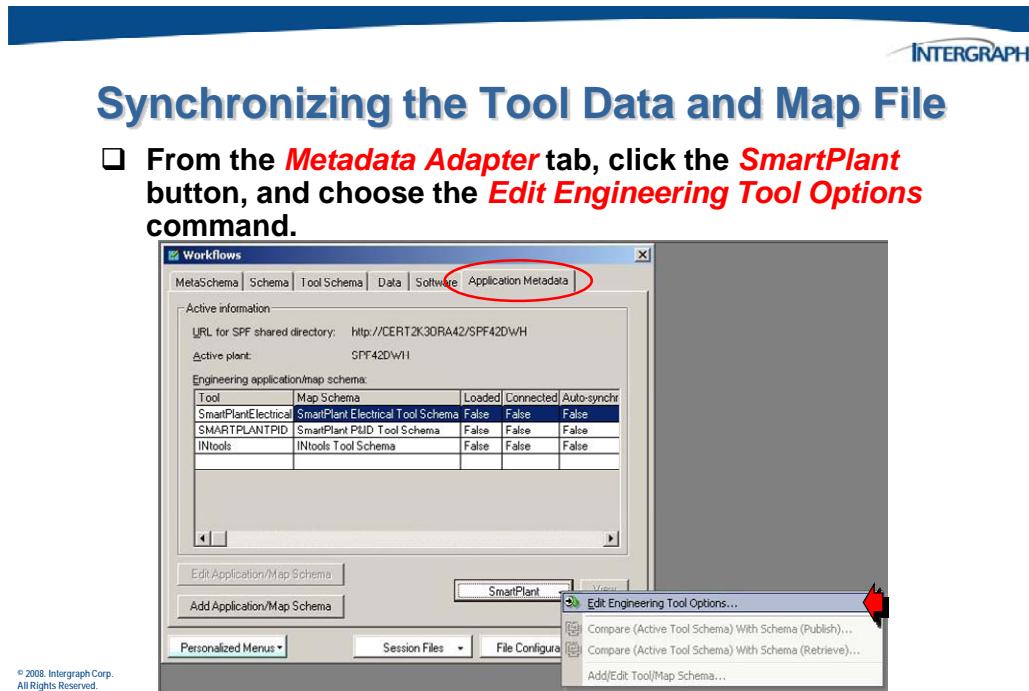


7.2.3 Synchronize the Tool Data and Map File

Along the top row of tabs, click the *Application Metadata* tab.

On this tab, there is a list of all the applications that have plants registered to this SmartPlant Schema. Information for this list is included in the session file.

Once there, click the SmartPlant button and choose the *Edit Engineering Tool Options* command.



Dismiss the warning message that tells you about applications for which there are not metadata adapters.

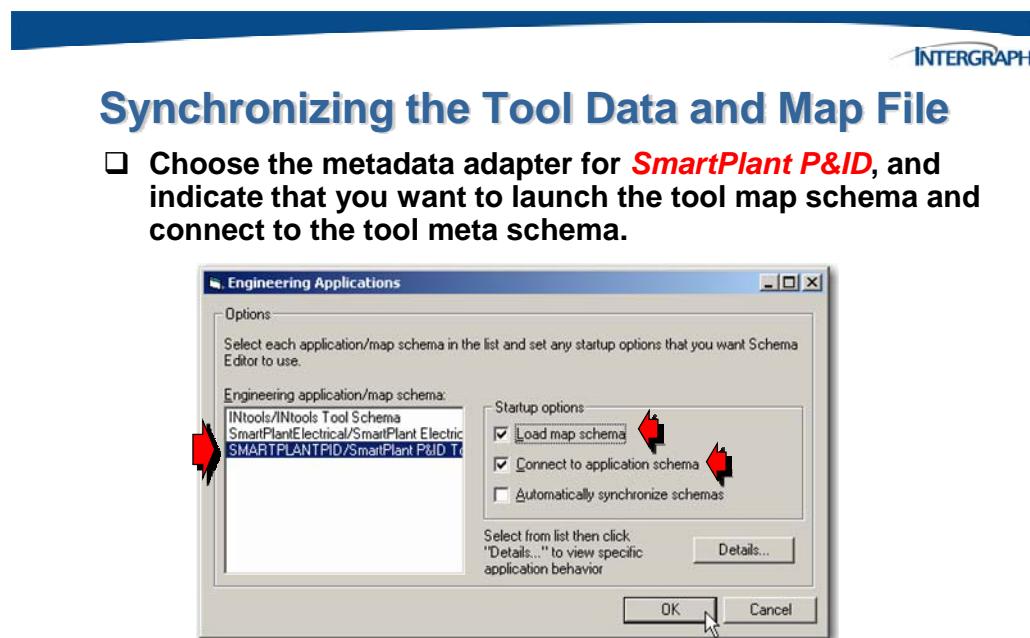
Synchronizing the Tool Data and Map File

- A warning message indicates tools that do not have metadata adapters. Click **OK** to continue.



On the **Engineering Applications** dialog box, a list indicates the tool map schemas and metadata adapters that are available for this SmartPlant site. Choose the tool for which you want to open the metadata adapter. Also check the **Load map schema** check box to indicate that you want to load the tool's map file. The **Connect to application schema** check box is automatically activated for you. This option connects to the tools database structure to compare it with the tool map file.

You may activate the **Automatically synchronize schemas** check box if you want the metadata adapter to synchronize the tools metaschema (database structure) and tool map schema without providing you with a list of the changes prior to making the changes.

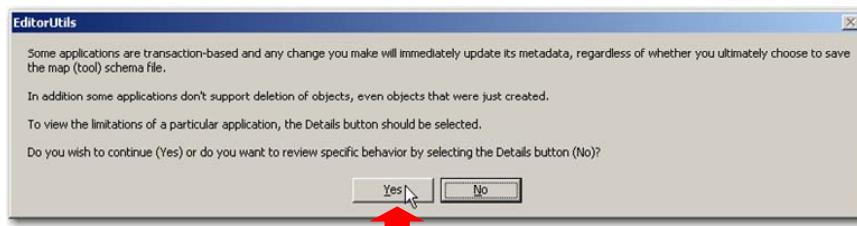


© 2008, Intergraph Corp.
All Rights Reserved.

The following informational dialog box appears.

Synchronizing the Tool Data and Map File

- The following message appears to warn you that the metadata adapter is capable of making changes immediately, without having to it is possible to make manually save the tool map schema. It also warns you that it is possible to make changes to the tool schema using the metadata adapter.
- Click **Yes** to continue.

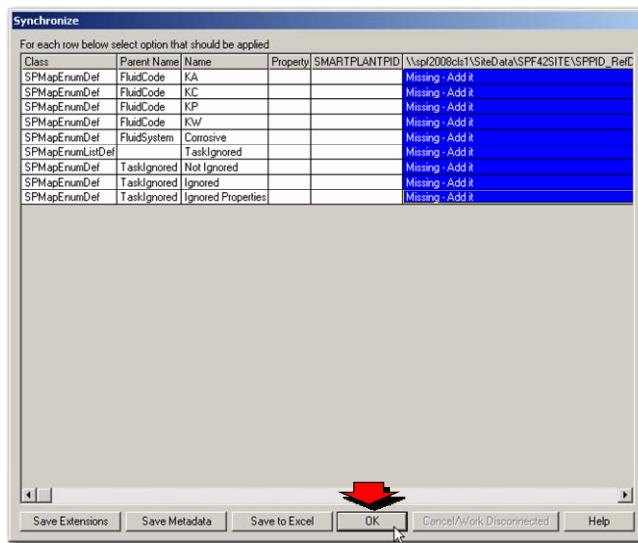


© 2008, Intergraph Corp.
All Rights Reserved.

The **Synchronize** dialog box shows the differences found between the tool meta schema and the tool map schema. Click OK to synchronize the tool schema and the map file.

Synchronizing the Tool Data and Map File

- On the **Synchronize** screen, click **OK** to push the changes you made in the Data Dictionary tool to the SPPID Tool Map Schema.



© 2008, Intergraph Corp.
All Rights Reserved.

Synchronize Form Columns:

Class – the class definition (ClassDef) for the object that is different.

Name – the name of the object that is different.

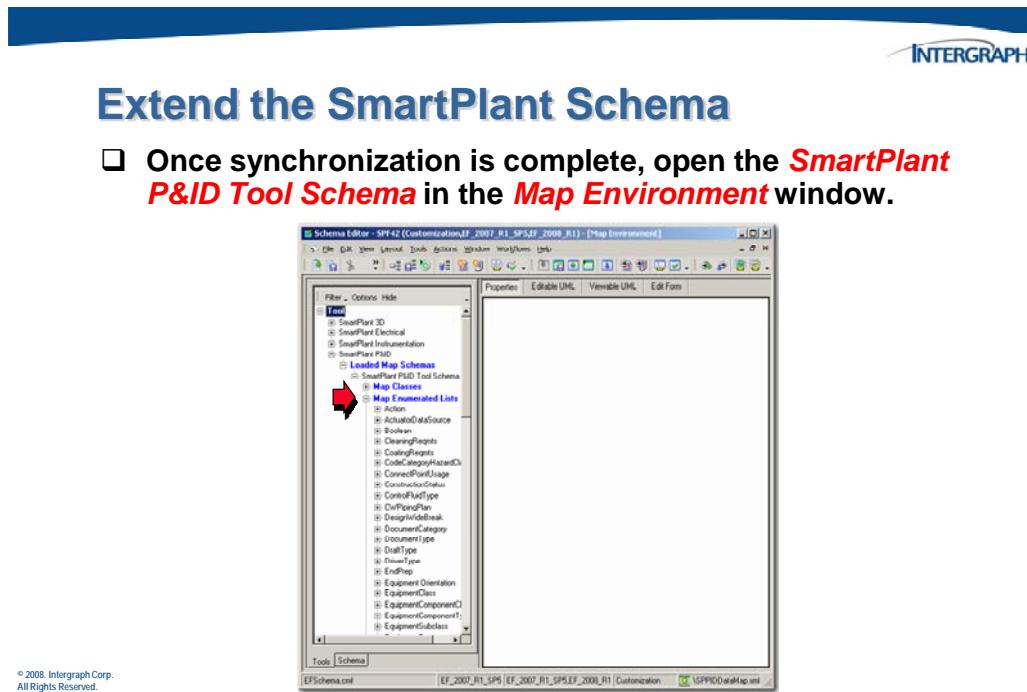
Property – if the object exists in both map containers, but one or more properties are different form one container to the other, then the name of a property that is different will appear in this column.

Tool metaschema – the tool’s database structure or data structure. If an object is found in the tool map file that is not in the tool’s database, the software may provide the option to create the new object in the tool’s database.

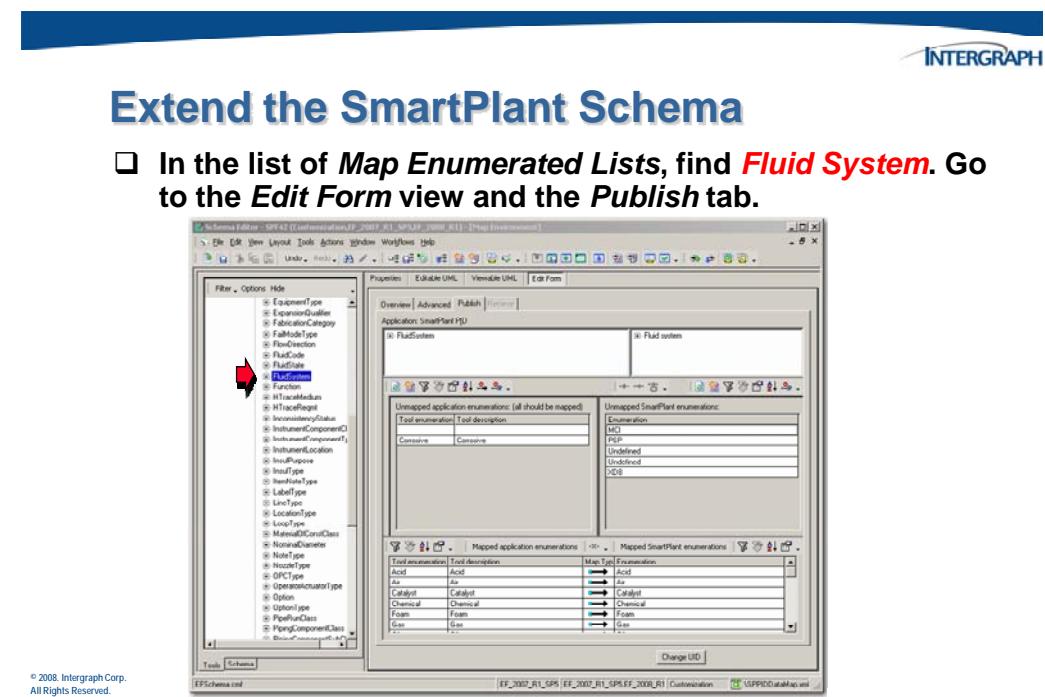
Tool map file – the tool’s map file. If an object is found in the tool’s database that is not found in tool map file, the software provides the option to create the new object in the tool map file.

7.2.4 Extending an Existing Enumerate List

Once the synchronization is complete, the **Map Environment** appears. This interface will display the tool map file using the **Tool** tab or the SmartPlant schema on the **Schema** tab. Use the tree on the left side of the display to drill down under SmartPlant P&ID to the **Map Enumerated Lists** heading. This section includes the selects lists defined in the Data Dictionary Manager.



Find the **Fluid System** list, and select it. Click the **Edit Form** tab, and then click the **Publish** tab.



The **Publish** tab has four controls:

Upper left tree – shows tool map schema information.

Upper right tree – shows SmartPlant schema information. Typically, this is an on-the-fly-generated view def created to assist mapping by showing properties of related interfaces.

Middle window – shows unmapped properties from the tool map file (left) and the SmartPlant schema (right).

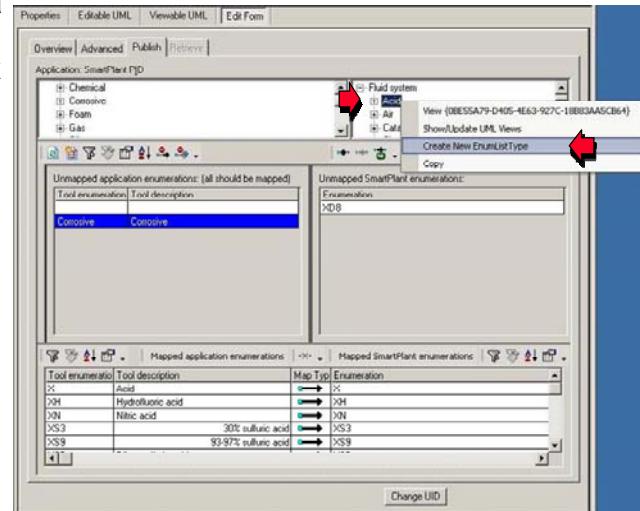
Lower window – shows properties that have been mapped for the tool. The property name as it comes from the tool is displayed on the left, and the name of the SmartPlant schema property that it is mapped to is displayed on the right. An arrow between the two properties points from left to right to indicate the direction of the mapping.

The **Retrieve** tab contains the same controls that perform the same functions as those on the **Publish** tab. However, on the **Retrieve** tab, the arrows in the lower window point from right to left (from SmartPlant to the tool).

Choose a value of the **Fluid System** list on the SmartPlant side of the dialog box. Right-click the value, and choose the **Create EnumListType** command to create a new list.

Extend the SmartPlant Schema

- Right-click on a fluid system value, and click **Create New EnumListType** to create a new enumerate list in the SmartPlant Schema.

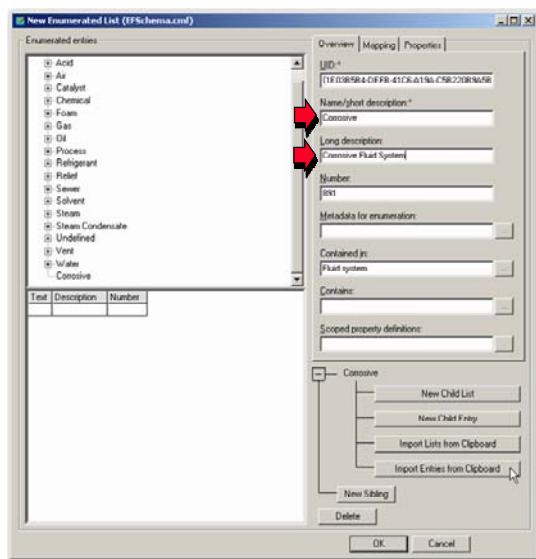


© 2008, Intergraph Corp.
All Rights Reserved.

Create the new Corrosive list, providing a name and long description for the list.

Extend the SmartPlant Schema

- Create the new **Corrosive** EnumListType.



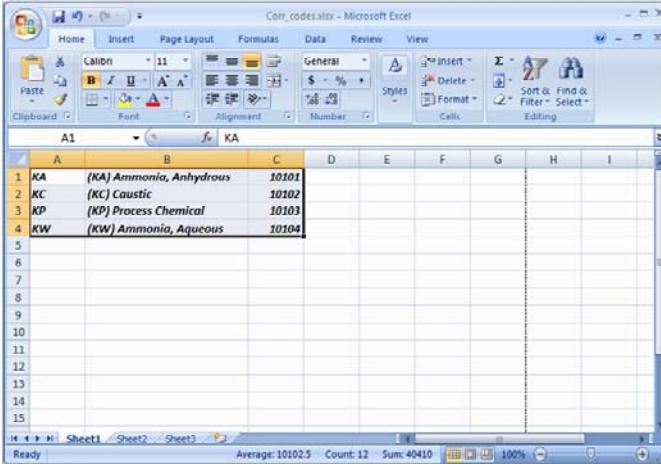
© 2008, Intergraph Corp.
All Rights Reserved.

You can create the list entries (enum enums) manually, or you can copy the applicable information to the clipboard and use the **Import Entries from Clipboard** button to automate the process.



Extend the SmartPlant Schema

Open the Excel spreadsheet that contains the information for the new fluid codes. Copy the information for the codes.

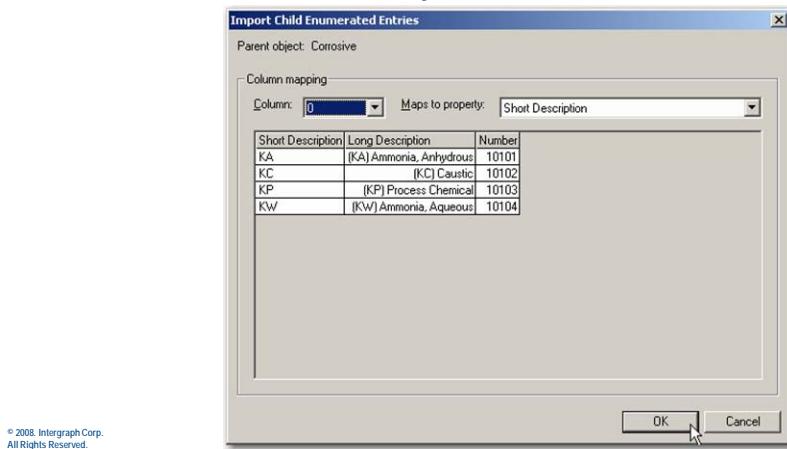


The screenshot shows a Microsoft Excel spreadsheet titled "Corr_codes.xlsx - Microsoft Excel". The table has three columns: A, B, and C. Column A contains codes KA, KC, KP, and KW. Column B contains descriptions like "(KA) Ammonia, Anhydrous", "(KC) Caustic", "(KP) Process Chemical", and "(KW) Ammonia, Aqueous". Column C contains numerical values 10101, 10102, 10103, and 10104 respectively. The table is selected, and the status bar at the bottom shows "Ready", "Average: 10102.5", "Count: 12", and "Sum: 40410".

Verify that the entry information was imported correctly.

Extend the SmartPlant Schema

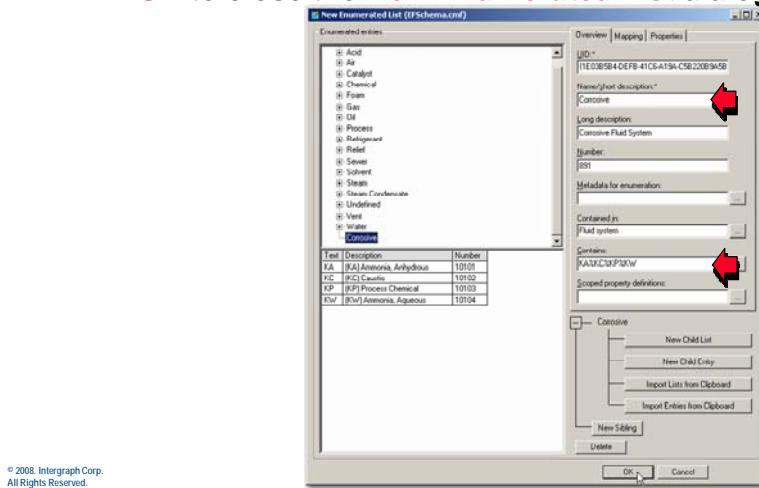
- ❑ Click the **Import Entries from Clipboard** button on the **New Enumerated List** dialog. The system will create the new entries automatically. Click **OK**.



Verify that the entries were added to the Corrosive value.

Extend the SmartPlant Schema

- ❑ Confirm that the entries were created properly, and click **OK** to close the **New Enumerated List** dialog box.



7.2.5 Mapping the Enumerate List

After you have created the new list and entries in the SmartPlant Schema, you are ready to map the list for publish from or retrieval to the authoring tool using the **Map Environment**.

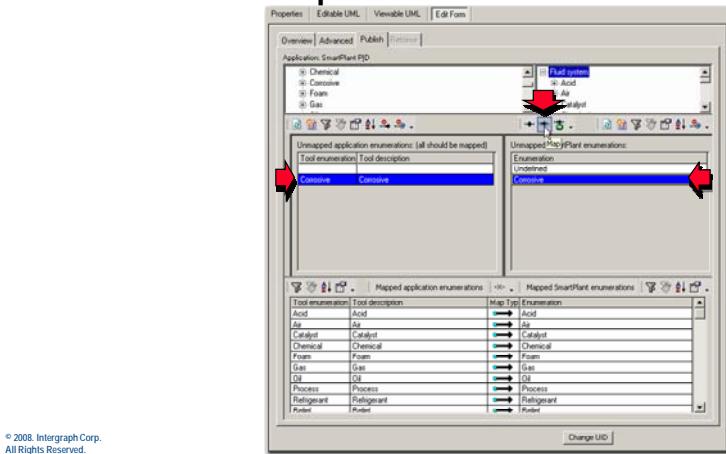
Now that you have created the list in the SmartPlant schema, you should see the Corrosive option on both sides of the middle pane. You can now select the properties on each side and map them, using the **Map** button.

Be sure you select the Fluid System node in the top right-hand pane.



Map the New Select List Value

- Select **Fluid System** on the SmartPlant Schema side, and find the value **Corrosive** on both sides. Click the **Map** button to map the two.



The following charts show you the mapping icons that are available in the middle pane and define their use.



Map the New Select List Value

Click To



Refresh – refreshes the display of the map properties



Create application property – displays the create/edit form for an application property (SPMapPropertyDef) and allows user to add an application property to the map schema



Filter application properties – allows the user to specify criteria that filters the set of displayed application properties in the unmapped (middle) control



Unfilter application properties – allows the user to unfilter the application properties

© 2008, Intergraph Corp.
All Rights Reserved.



Map the New Select List Value

Click To



Add/delete displayed properties – allows user to specify properties (columns) to be displayed in the unmapped control



Sort map properties – allows user to specify sort criteria to be used for ordering unmapped application properties



Ignore application properties – allows user to identify application properties to ignore



Set unmapped map properties – allows user to identify map properties that won't be mapped

© 2008, Intergraph Corp.
All Rights Reserved.



Map the New Select List Value

Click To



Auto-create – automates the process of creating a new property.



Map – creates a mapping relationship between two objects selected in the window.



Auto-map – provides suggestions of existing objects that you might want to map to a selected item.

© 2008, Intergraph Corp.
All Rights Reserved.

Confirm that the mapping was created successfully by checking the bottom window.



Map the New Select List Value

Confirm that the **Corrosive** fluid system is mapped.



Tool enumeration	Tool description	Map Typ	Enumeration
Solvent	Solvent	→	Solvent
Steam	Steam	→	Steam
Steam condensal	Steam condensate	→	Steam Condensate
Vent	Vent	→	Vent
Water	Water	→	Water
Corrosive	Corrosive	→	Corrosive

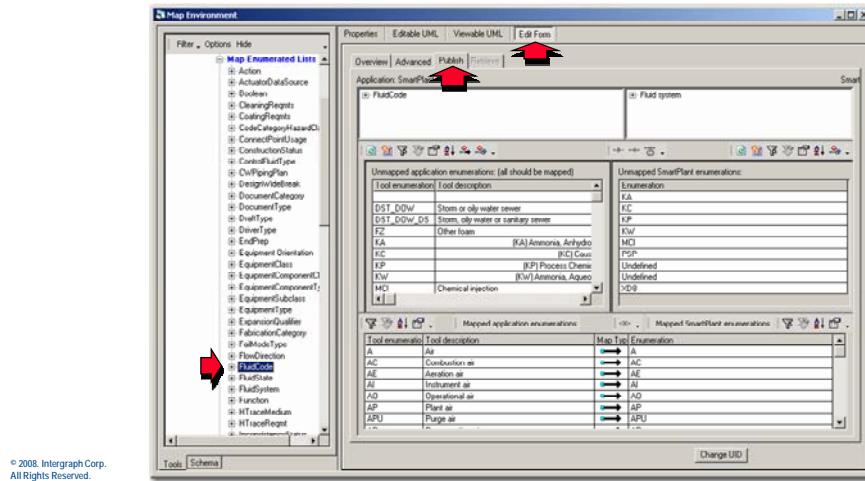
© 2008, Intergraph Corp.
All Rights Reserved.

While the previous steps mapped the Corrosive value for the Fluid System list, we still need to map the new values for the Fluid Code list.

From the **Map Environment**, scroll up the SmartPlant P&ID **Map Enumerated Lists** to the **Fluid System** option. Again, open the **Edit Form** view, and go to the **Publish** tab.

Extend the SmartPlant Schema

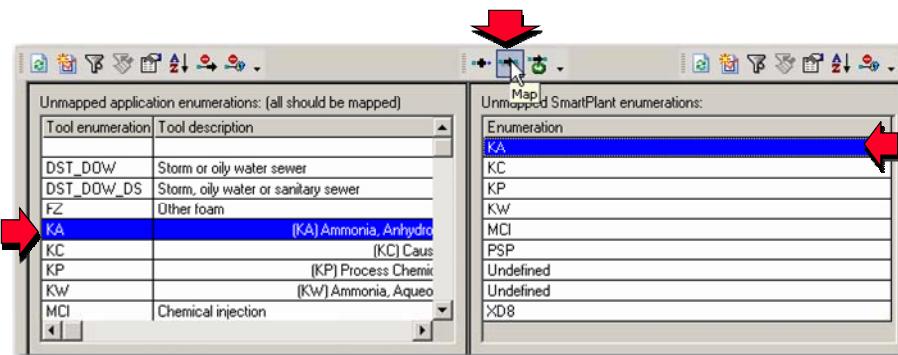
- Find the **Fluid Code** map enumerated list. From the **Edit Form** view, open the **Publish** tab.



Select the new values on each side and use the map button to map them.

Map the New List Values

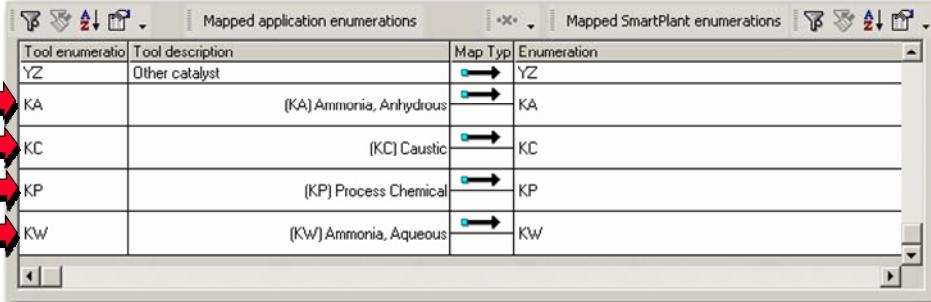
- Choose a new fluid code value from the tool map file and the matching value in the SmartPlant Schema. Click the **Map** button to create a mapping relationship. Repeat for each of the new fluid codes.



Repeat that last step for each value, as each enum enum must be mapped one at a time. Confirm that all the fluid codes are mapped.

Map the New List Values

- Confirm that the fluid codes were mapped.



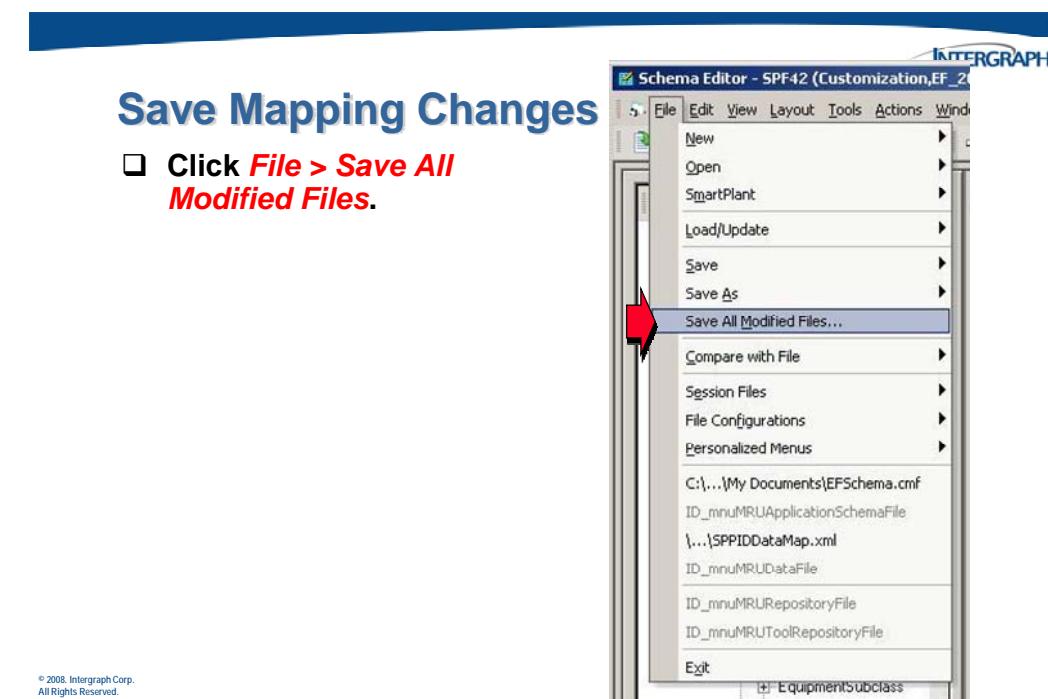
Tool enumeration	Tool description	Map Typ	Enumeration
YZ	Other catalyst	→	YZ
KA	(KA) Ammonia, Anhydrous	→	KA
KC	(KC) Caustic	→	KC
KP	(KP) Process Chemical	→	KP
KW	(KW) Ammonia, Aqueous	→	KW

7.2.6 Saving the Schema Changes

At this point, the additions to the SmartPlant Schema and the mapping information are stored in memory and need to be saved to the applicable files.

Note:

- The **File > Save All Modified Files** command prompts you to save the changes to all schema files modified; however, it does not save the changes to the session file.
- You should not save the session file when the Schema Editor is connected to the a tool map file, unless you want to open that map file whenever you open the Schema Editor and use that session file.
- You should not open multiple tool map files at the same time.



You are first prompted to save the changes to the CMF file. These changes include the addition of the new enum list type and enum enums to the SmartPlant Schema, so you click **Yes** to save the changes.

Save Mapping Changes

- Verify the tool map schema name for the mapping changes and choose **Yes**.



© 2008, Intergraph Corp.
All Rights Reserved.

A warning message appears if you have not run validation on your changes. Validation will be discussed in a later section of this guide. For now, click **Yes** to continue.

You are next prompted to save the changes to the P&ID map file. The changes to this file include the additions made by the metadata adapter during the synchronization process and the mapping information. Click **Yes** to continue.

Save Mapping Changes

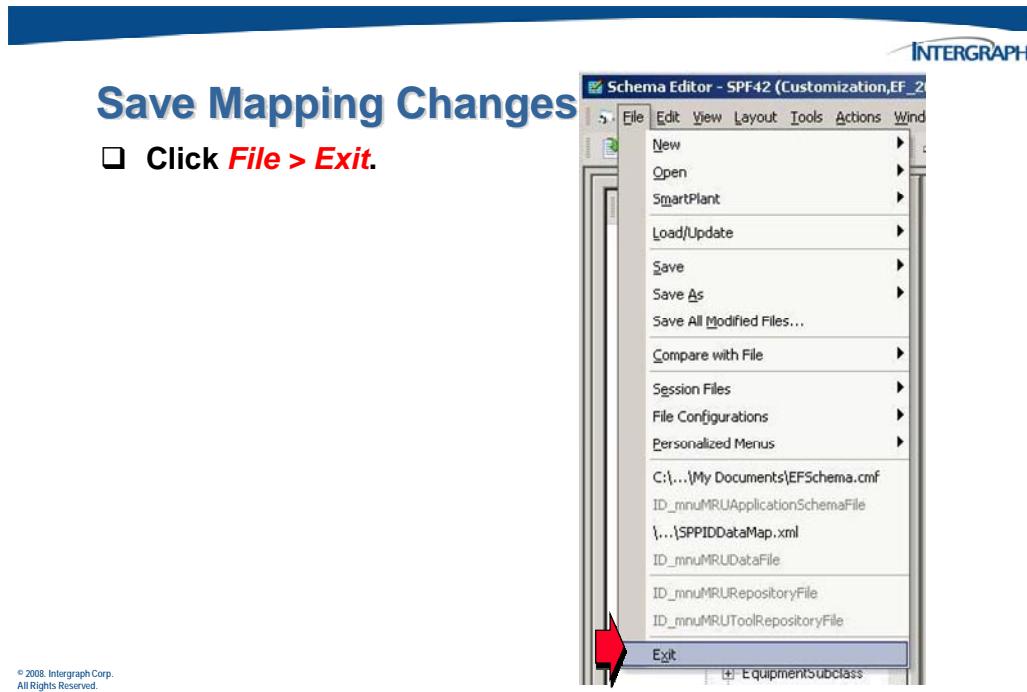
- Verify the CMF file for the property addition and choose **Yes**.



© 2008, Intergraph Corp.
All Rights Reserved.

Again, a warning message appears if you have not run validation on your changes. Validation will be discussed in a later section of this guide. For now, click **Yes** to continue.

Close the Schema Editor.



7.3 Planning Changes to the SmartPlant Schema

Before you start extending the SmartPlant Schema to hold new information from authoring tools, you should spend some time planning your changes. First, determine what types of changes will be needed:

- Will you be publishing new classes of objects?
- Will you be sharing custom properties for existing objects?
- Do you need to add custom values to existing enumerate lists?

The way you will extend the SmartPlant Schema will be determined by what changes you need to make. For example, we have just seen an example of extending an existing list.

However, if you want to create whole new classes of objects, you need to plan how to model the new class – what interfaces should it realize, what properties should be available, what objects should it have relationships to.

If you are extending an existing enumerate list, you need to find the existing list in the SmartPlant Schema, and if you are adding custom properties to be shared, you need to analyze the schema to determine what interface should expose the properties.



Schema Analysis

- In the remaining chapters, we will be adding custom properties to SmartPlant Enterprise tools and creating mapping to get that information into SPF.**
- Before we can add our new property to the SmartPlant schema, we must decide what interface should expose our property. We have two choices:**
 - Existing/Delivered Interface
 - Custom Interface (recommended)

In the following chapters, we will see how to extend the SmartPlant Enterprise tools to publish and retrieve custom properties. Before we can do that, we will need to determine the best way to model those new properties in the SmartPlant Schema.

Two basic options are available: we can add the custom properties to existing interfaces or we can create custom interfaces for our properties.

In the first option, we would need to find an interface that is either already realized by the class definitions for the objects we want to use the properties, or that can be added to any of these class defs.



Schema Analysis

- Existing/Delivered Interface*** –find an existing interface that is already realized by all the class definitions that you want to have access to this property.

Method:

1. Create a list of all the objects that should have this property and find the correlating class defs. Remember that an object shared across multiple tools will have multiple class defs: For example, an instrument from SPPID is a PIDInstrument, but an instrument from SPI is an INDXInstrument.
2. Compare the interfaces realized by each of these class defs until you find an interfaces common to all. Add the custom property to that shared interface.

With the second option, we can create a custom interface for our new properties and can then add it to all the class defs to which we want to add those properties.



Schema Analysis

- New Interface** –create a custom interface just for the custom properties you are adding to the system.

Method:

1. Create a list of all the objects that should have this property and find the correlating class defs. Remember that an object shared across multiple tools will have multiple class defs: For example, an instrument from SPPID is a PIDInstrument, but an instrument from SPI is an IDXInstrument.
2. Create a custom interface and make sure that it is realized by all the class defs from the list above.

Note: You will also want to make sure that the primary interfaces of those class defs *imply* your new interface.

© 2008. Intergraph Corp.
All Rights Reserved.

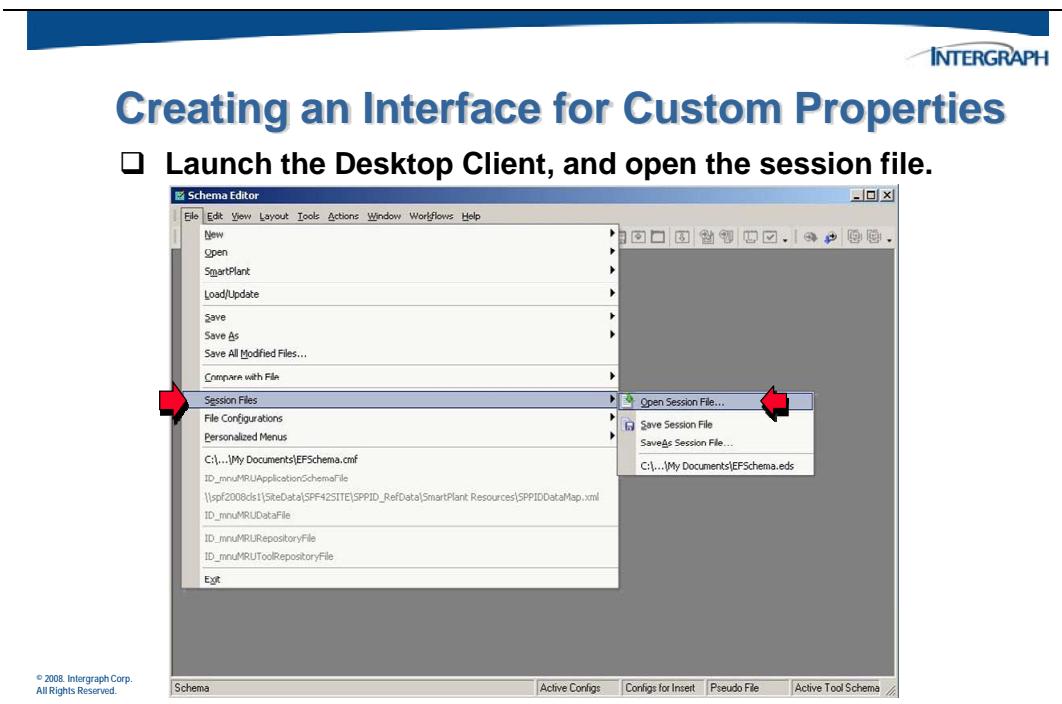
For our activities, we will use the second option and create a custom interface with the sole purpose of exposing our new properties. For this method to work, we need to make sure that our interface is realized by all the class defs of objects that will use this property and also be implied by all the primary interfaces of those class defs.

7.4 Creating a Custom Interface for New Properties

When expanding the SmartPlant Schema for integration items, we must launch the CMF file that is stored in the SmartPlant Foundation database.

If you still have the CMF checked out of the Desktop Client from the previous example, you need only launch the Schema Editor application and open the saved session file. However, if you do not currently have the CMF file checked out, you must first do so as we saw in the previous section.

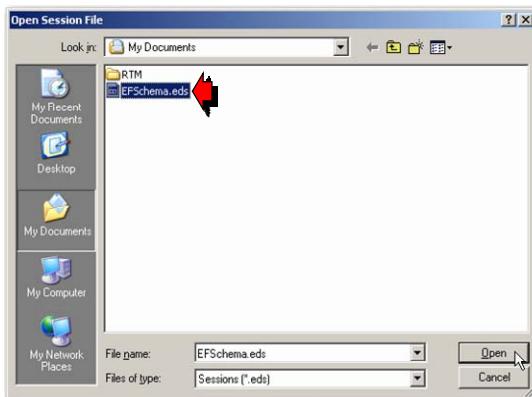
Since we have not yet checked in our CMF file, we will continue making changes. Launch the Schema Editor tool and use the **File > Session Files > Open Session file** command.



Find the session file that you save when you first launched the CMF file from the Desktop Client.

Creating an Interface for Custom Properties

- Find the **EFSchema.edt** file in your *My Documents* folder, and open it.

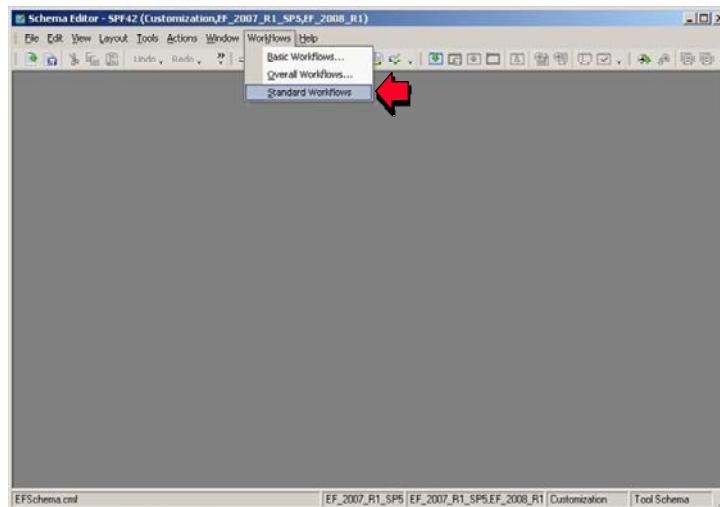


© 2008, Intergraph Corp.
All Rights Reserved.

Use the **Workflows** menu to choose the user interface you want to use.

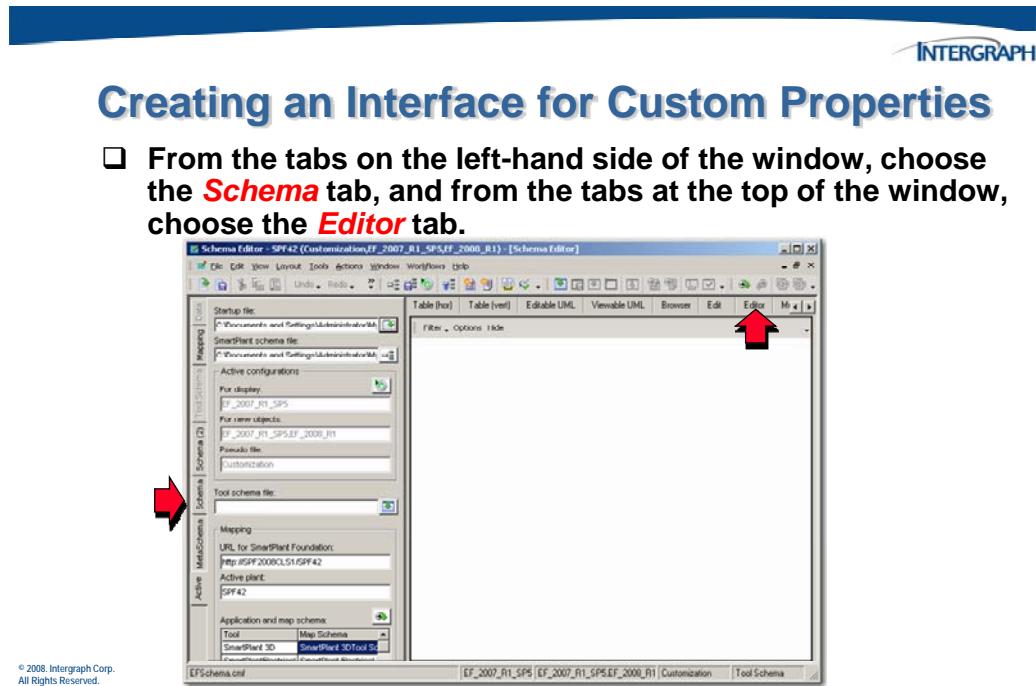
Creating an Interface for Custom Properties

- From the **Windows** menu, choose the **Standard Workflow**.

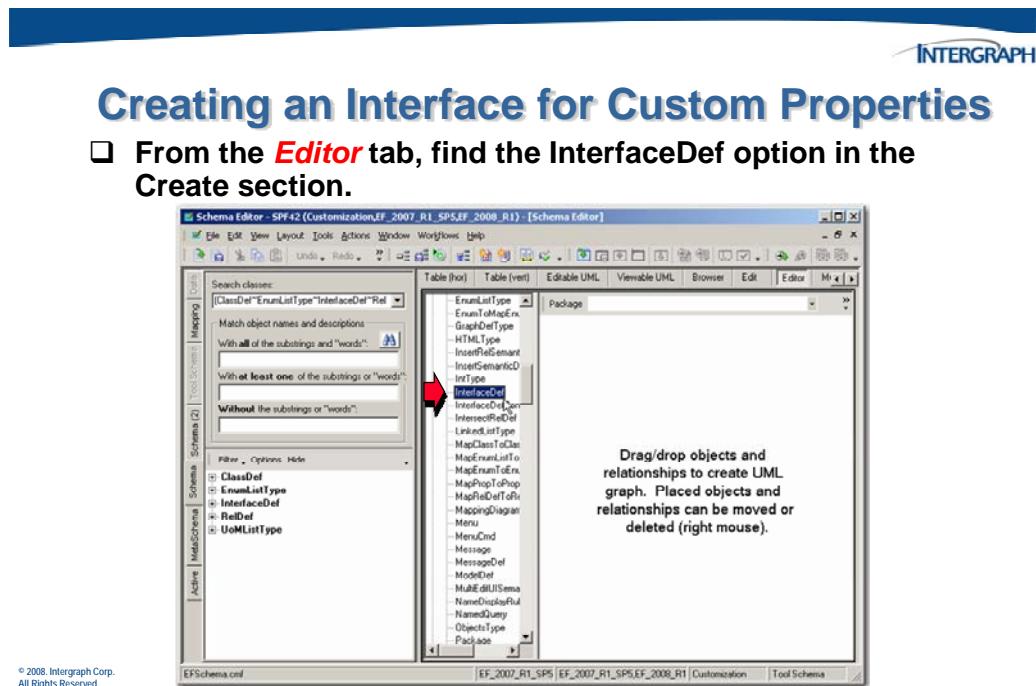


© 2008, Intergraph Corp.
All Rights Reserved.

From the *Standard Workflow*, click the open the *Editor* view of the *Schema* tab.



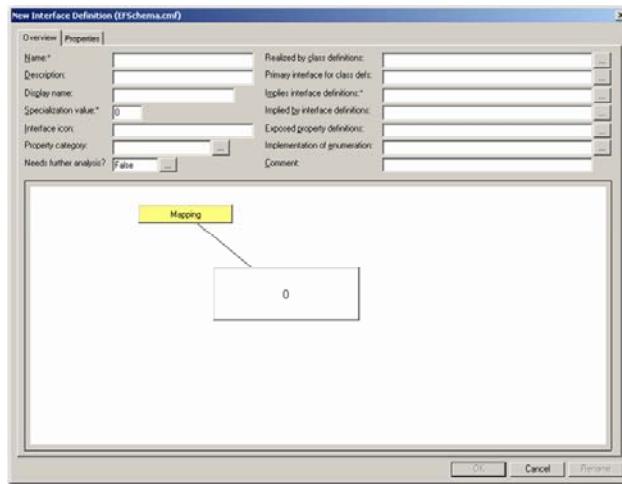
Open the *Create* section of the *Editor* view.



Create a new *interface definition*.

Creating an Interface for Custom Properties

- Drag and drop the *InterfaceDef* option into the work area.
Create a new interface for custom properties.

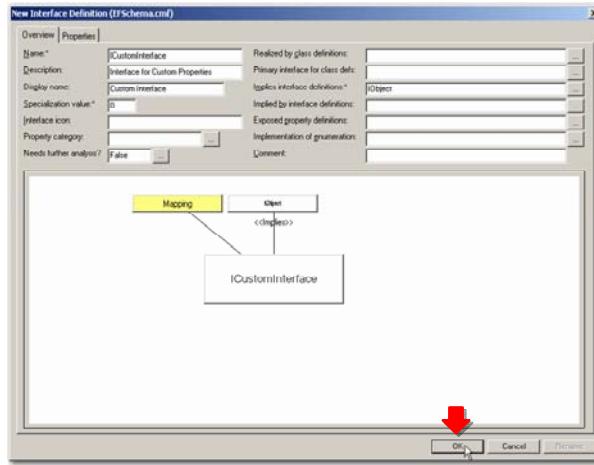


For now, we only want to create the new interface. Later we will make sure it is realized by the class defs using the properties and implied by the appropriate interfaces. Provide the necessary information to create the interface – a name and implied interface definition (IObject).



Creating an Interface for Custom Properties

- Click **OK** to save your changes to the **ICustomInterface**.
- Save your changes to the **Schema** file and the **Session** file.



Save your changes to the file and save the session file so that you can connect with it again later.

7.5 SmartPlant Foundation Mapping Spreadsheets

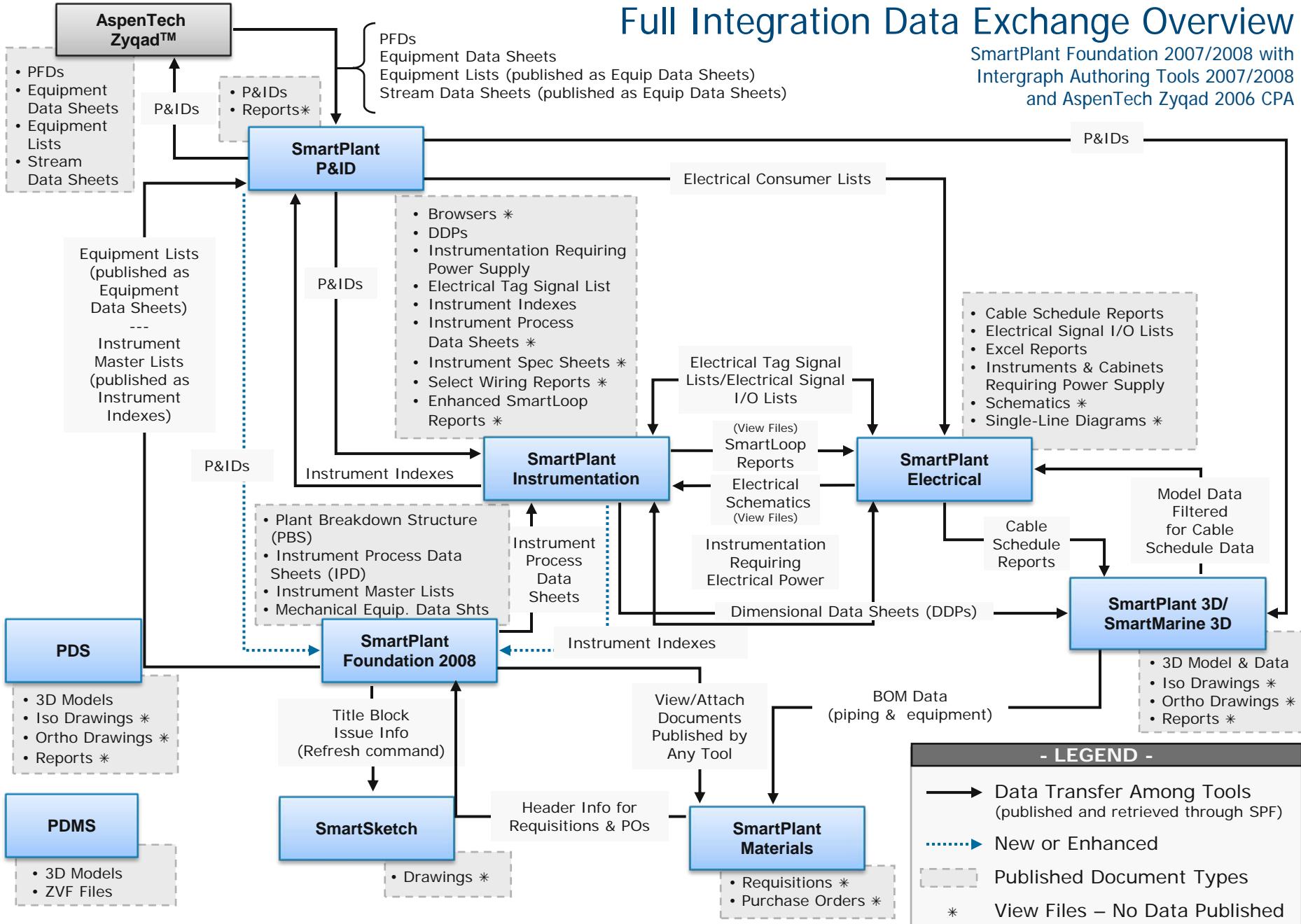
SmartPlant Foundation mapping spreadsheets are available on Intergraph's eCustomer (<http://crmweb.intergraph.com>) under:

Download Software Updates > Products > SmartPlant Foundation > Technical Notes and Documentation

If you don't have an eCustomer login, you can request one at <https://crmweb.intergraph.com> then click on the *I'm a new user* link, fill out the form, and your eCustomer login will be provided.

Full Integration Data Exchange Overview

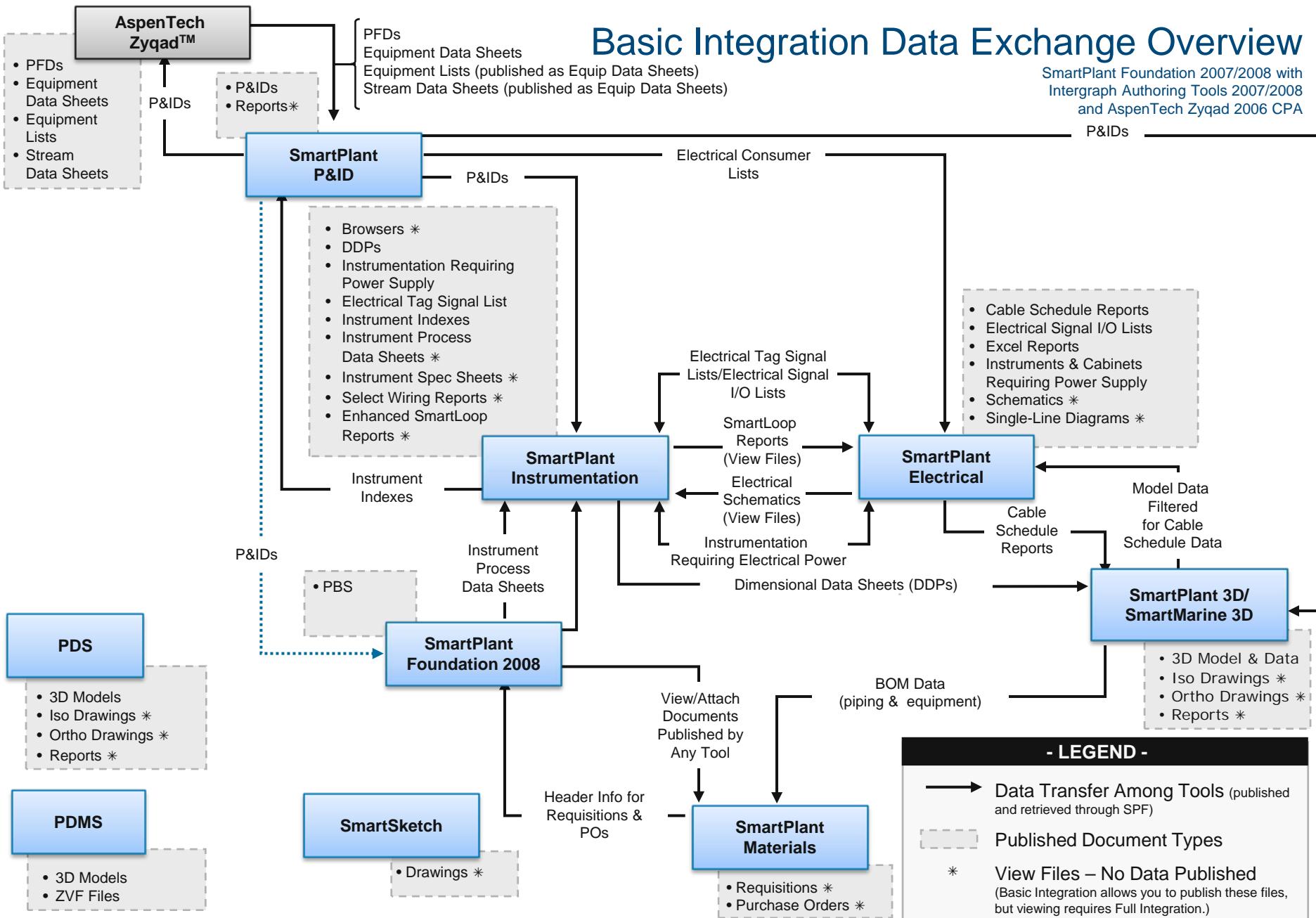
SmartPlant Foundation 2007/2008 with
Intergraph Authoring Tools 2007/2008
and AspenTech Zyqad 2006 CPA



NOTE: This diagram represents default, out-of-the-box data exchange.
Configuration allows additional documents and data to be published and retrieved.

Basic Integration Data Exchange Overview

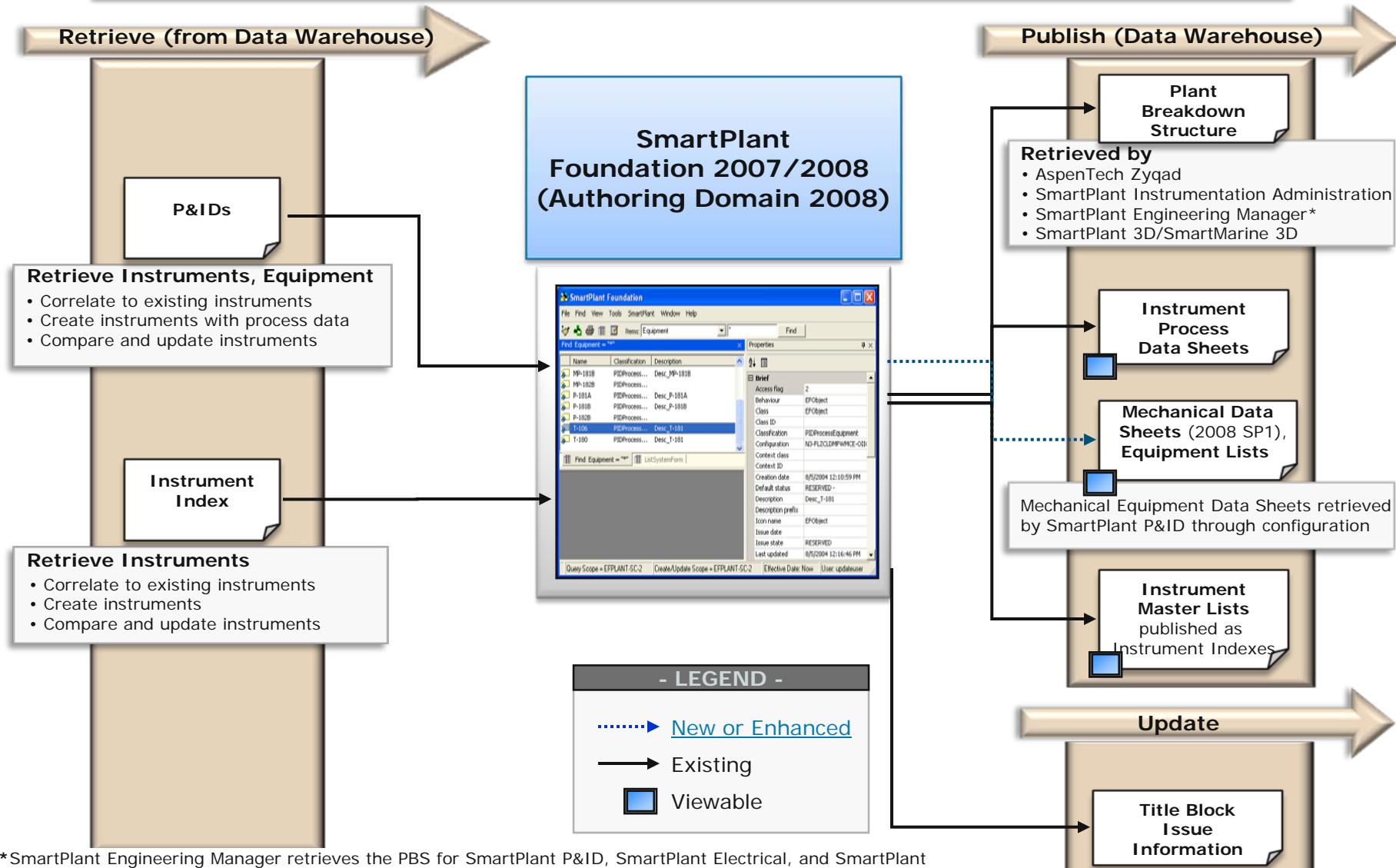
SmartPlant Foundation 2007/2008 with
Intergraph Authoring Tools 2007/2008
and AspenTech Zygad 2006 CPA



NOTE: This diagram represents default, out-of-the-box data exchange. Configuration allows additional documents and data to be published and retrieved.

SmartPlant Foundation Data Exchange Example (Authoring Environments in 2008)

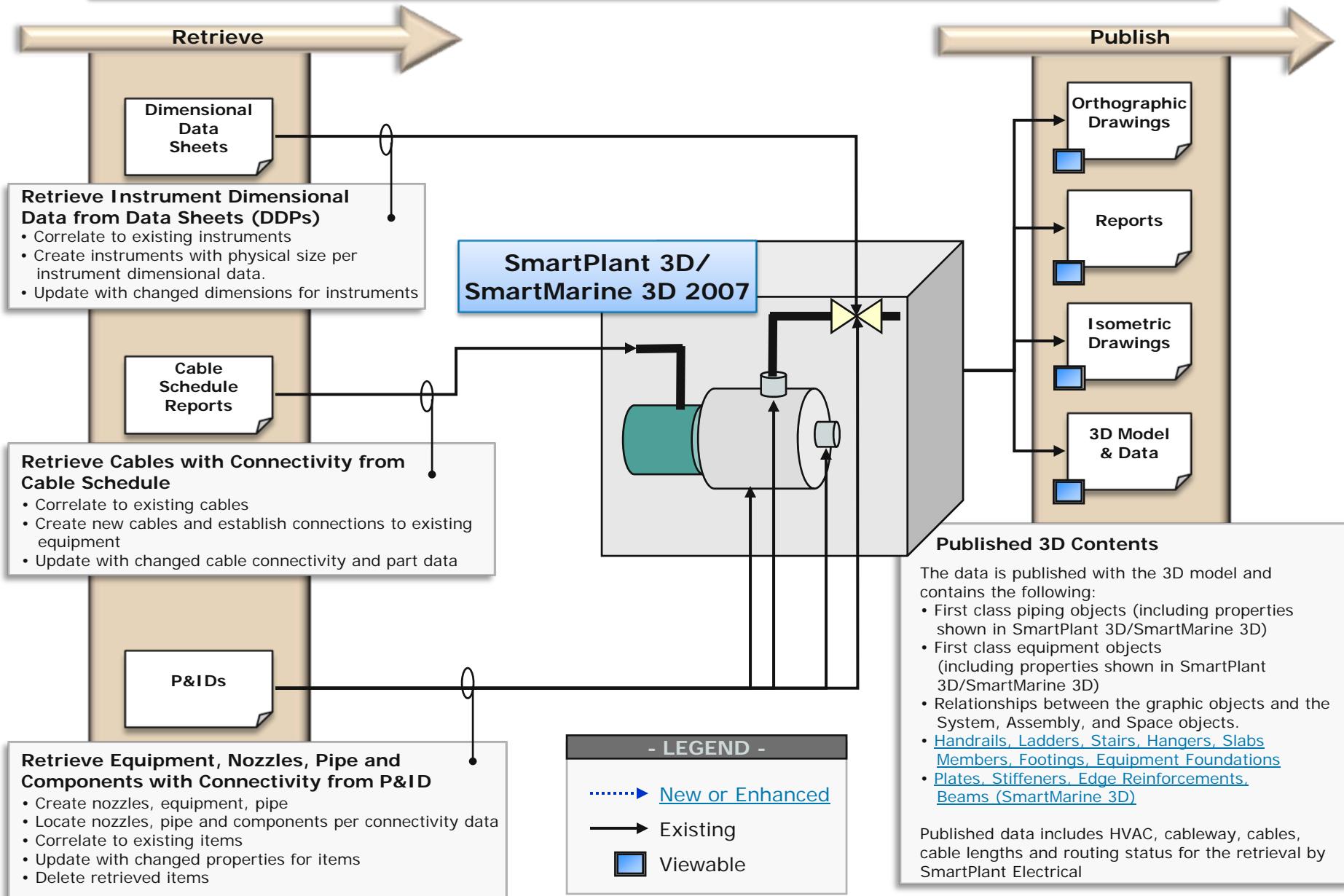
SmartPlant Foundation publishes and retrieves the following types of data and documents.



*SmartPlant Engineering Manager retrieves the PBS for SmartPlant P&ID, SmartPlant Electrical, and SmartPlant Instrumentation. However, SmartPlant Instrumentation Administration can retrieve the PBS for SmartPlant Instrumentation if you are not using SmartPlant Engineering Manager.

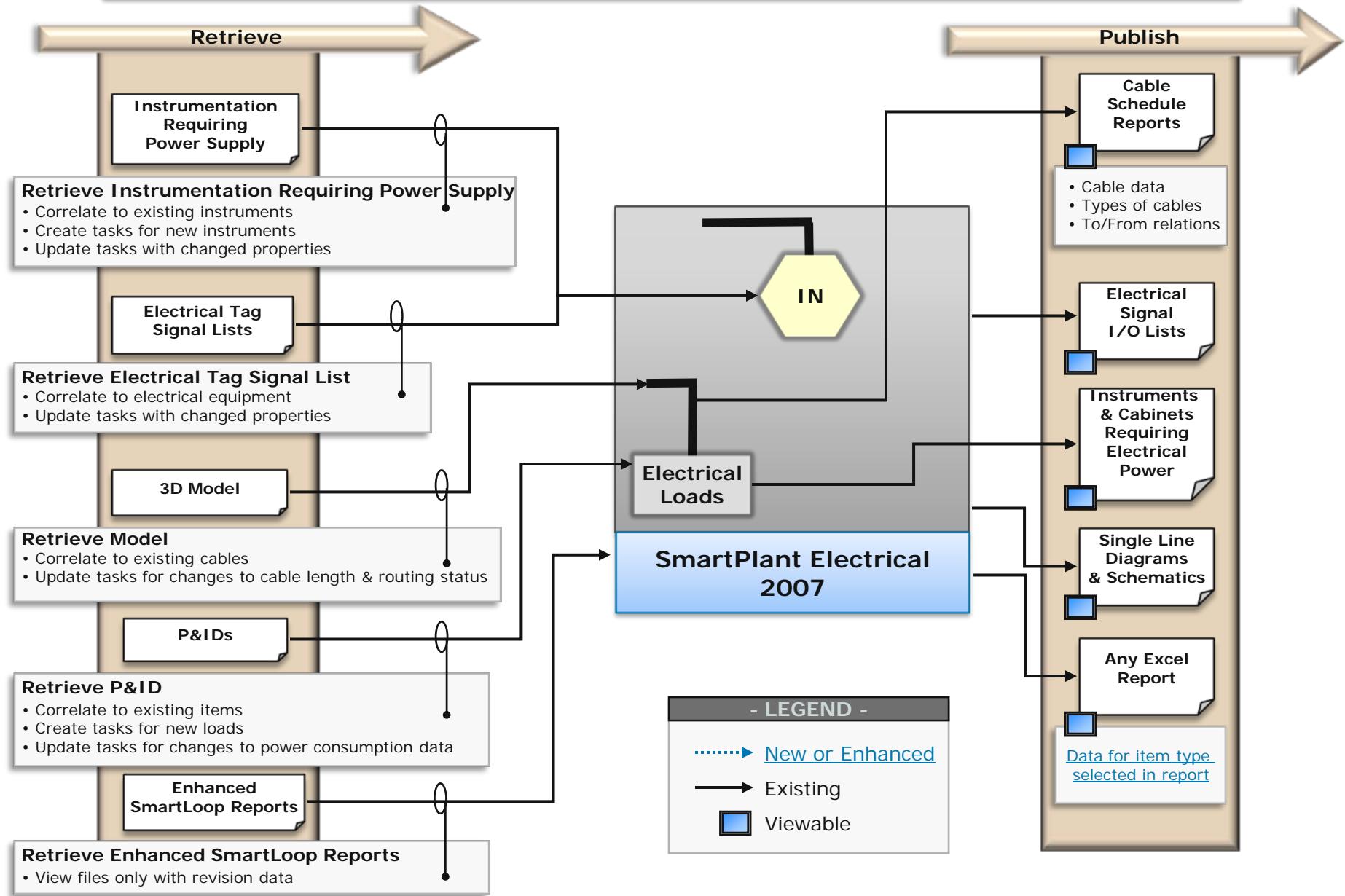
SmartPlant 3D/SmartMarine 3D Data Exchange Example

SmartPlant 3D/SmartMarine 3D publish and retrieve the following types of data and documents.



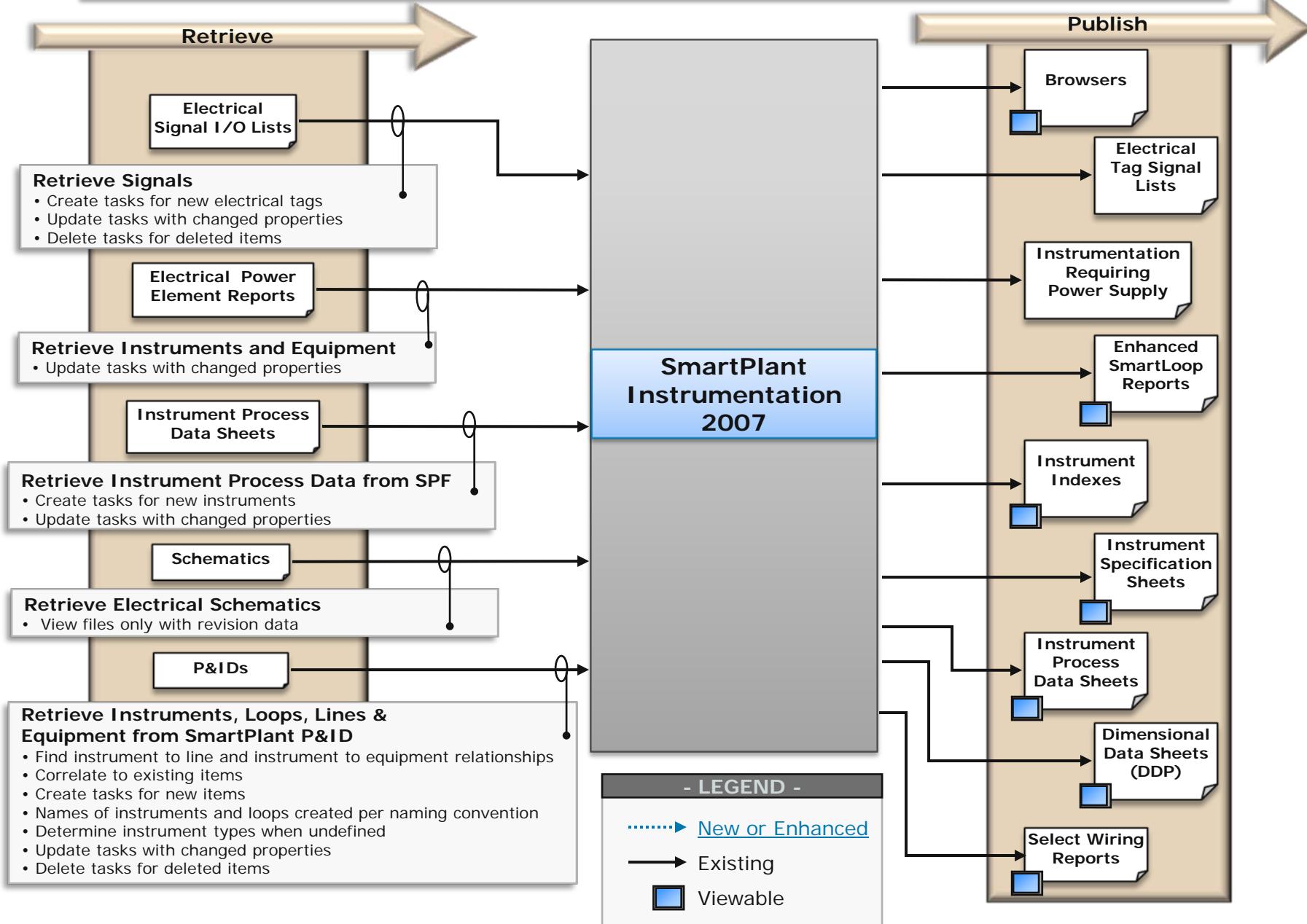
SmartPlant Electrical Data Exchange Example

SmartPlant Electrical publishes and retrieves the following types of data and documents.



SmartPlant Instrumentation Data Exchange Example

SmartPlant Instrumentation publishes and retrieves the following types of data and documents.



SmartPlant P&ID Data Exchange Example

SmartPlant P&ID publishes and retrieves the following types of data and documents.

Retrieve



Retrieve Instruments and Loops from Instrument Index

- Correlate to existing items
- Create tasks for new instruments and loops
- Create tasks can automatically place onto placement zone in drawing
- Update tasks with changed instrument and loop properties
- Delete tasks when instruments and loops are deleted

Retrieve Equipment Components from Data Sheets

- Correlate to existing items
- Create tasks for new equipment components
- Create tasks will automatically place onto parent
- Update tasks with changed equipment comp. properties
- Delete tasks when equipment comps are deleted



N1
N2

M-100

Retrieve Equipment from Data Sheets

- Correlate to existing items
- Create tasks for new equipment
- Create tasks can automatically place into placement zone in drawing
- Update tasks with changed equipment properties
- Delete tasks when equipment are deleted

Retrieve Streams from PFD

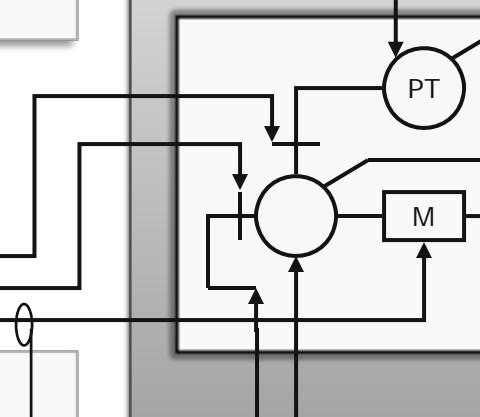
- Create tasks as new pipe runs
- Update tasks with changed stream properties
- Delete tasks remove stream data and correlation, pipe runs remain on P&ID



Retrieve Equipment from PFD

- Correlate to existing items
- Create tasks for new equipment
- Create tasks can automatically place into placement zone in drawing
- Update tasks for changes with changed equipment properties
- Delete tasks when equipment are deleted

SmartPlant P&ID 2007



Expand classes and data published for equipment

Publish

Publish specific instrument type only when known



M

Publish electrical consumption data

Published P&ID Contents

The published document contains the following types of objects:

- Equipment
- Equipment Components
- Nozzles
- Pipe Runs
- Piping Components
- Instruments
- Instrument Loops
- Signal Runs

In addition, the connectivity relationships are published for these objects.

- LEGEND -

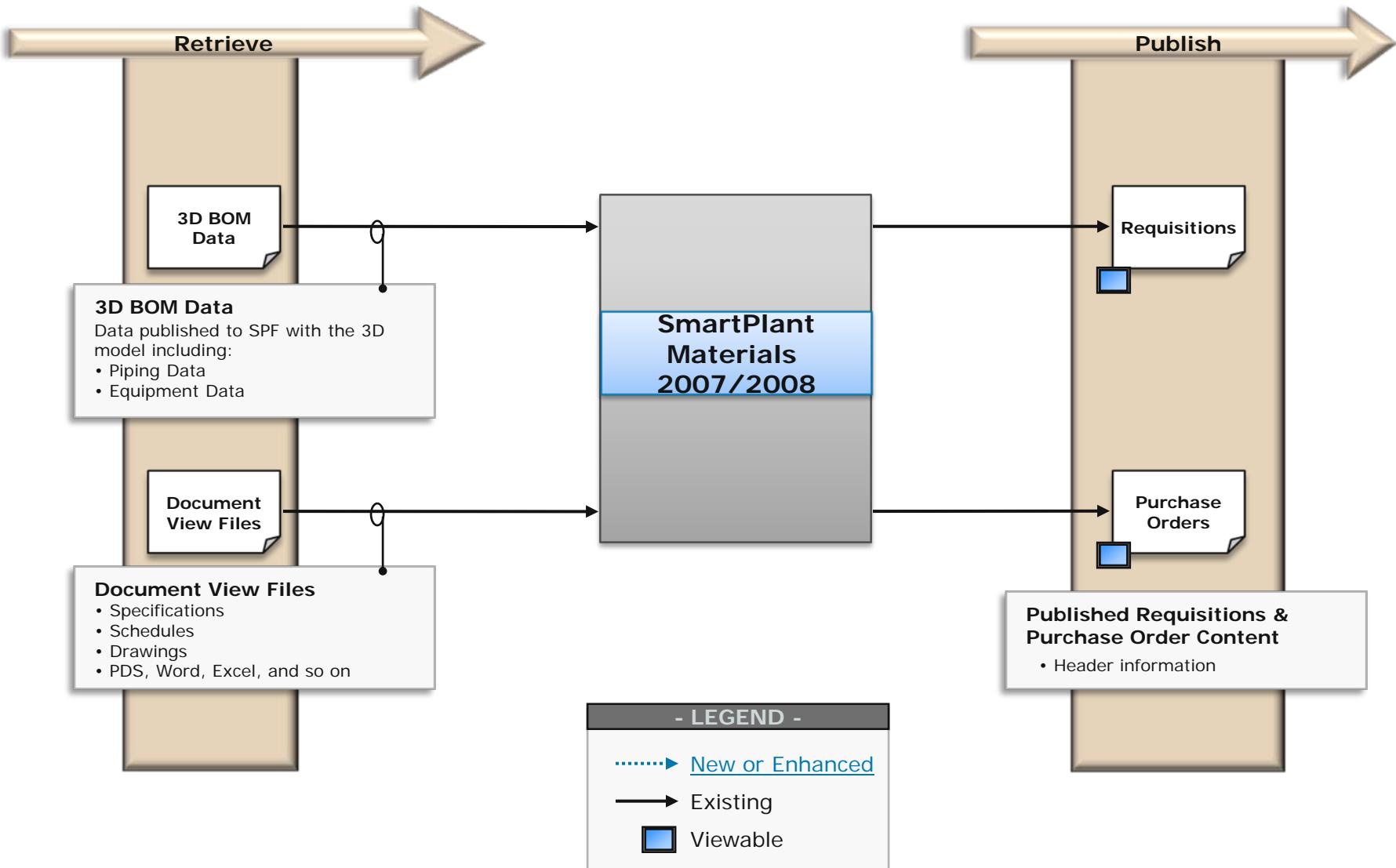
New or Enhanced

Existing

Viewable

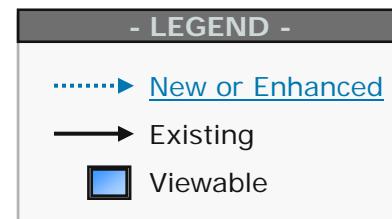
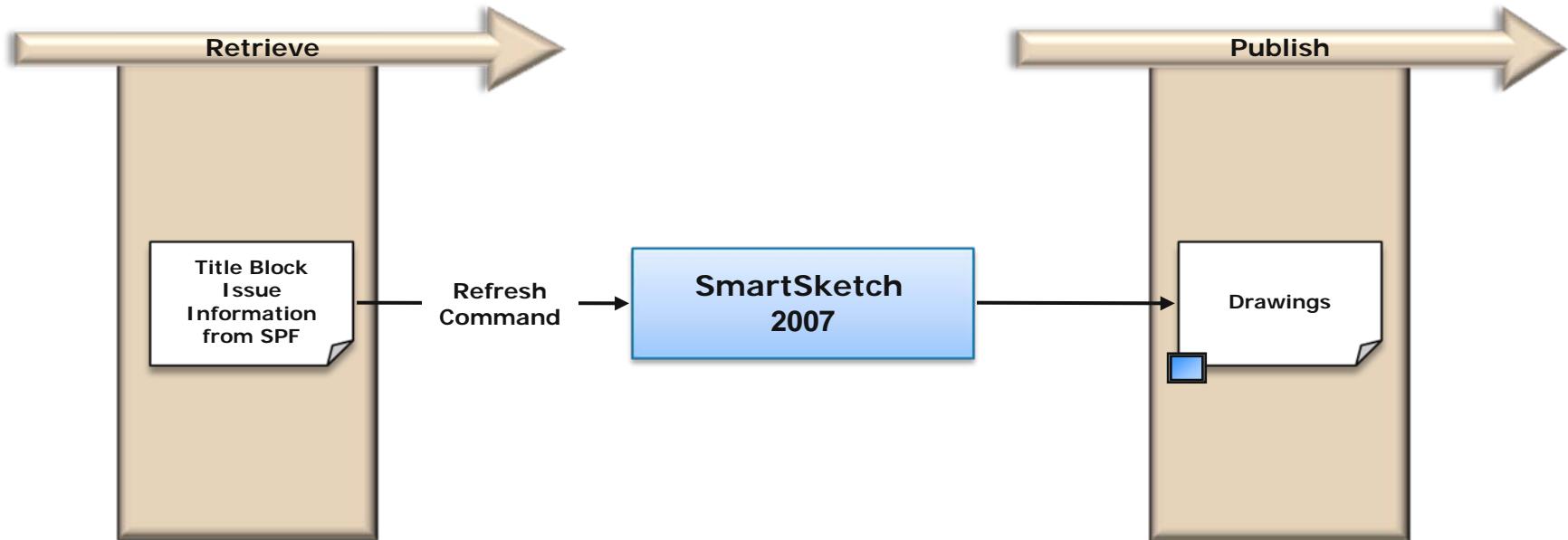
SmartPlant Materials Data Exchange Example

SmartPlant Materials publishes and retrieves the following types of data and documents.



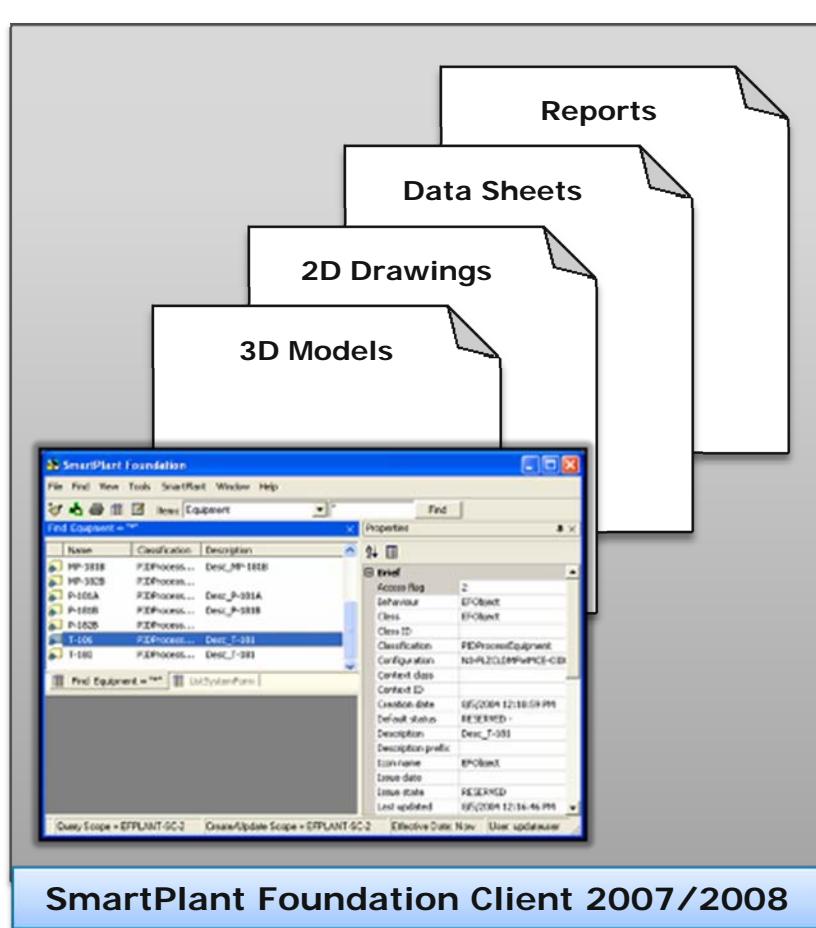
SmartSketch Data Exchange Example

SmartSketch publishes and retrieves the following types of data and documents.



SmartPlant Review Information Exchange Example

Information flows into and out of SmartPlant Review in the following way when it is used in an integrated environment.

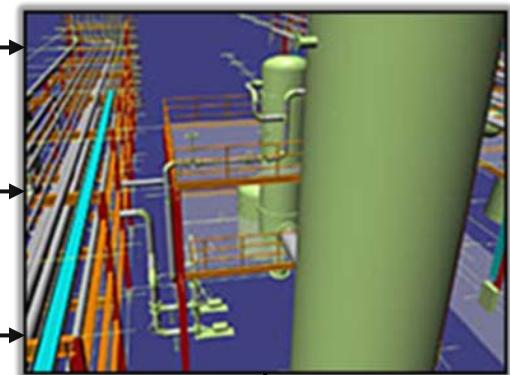


Query SPF for equipment components and display using streaming

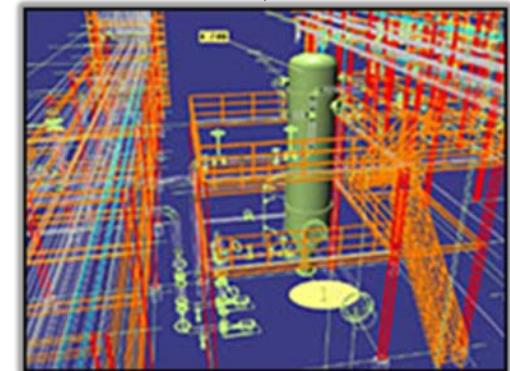
Display properties from SPF

Query SPF properties to build display sets

SmartPlant Review 6.2



Generate Display Sets



- LEGEND -

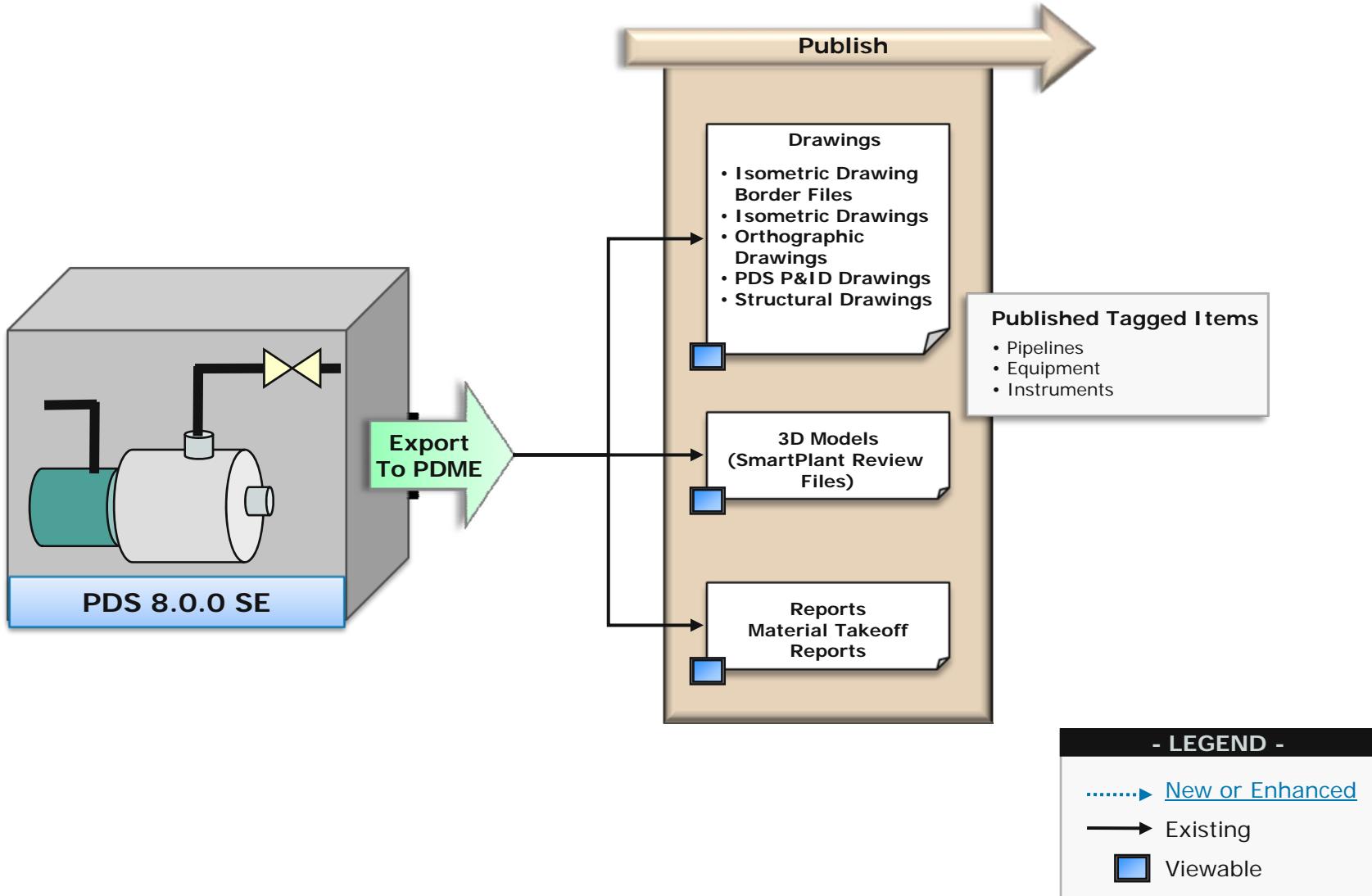
.....> [New or Enhanced](#)

→ Existing

Display Set

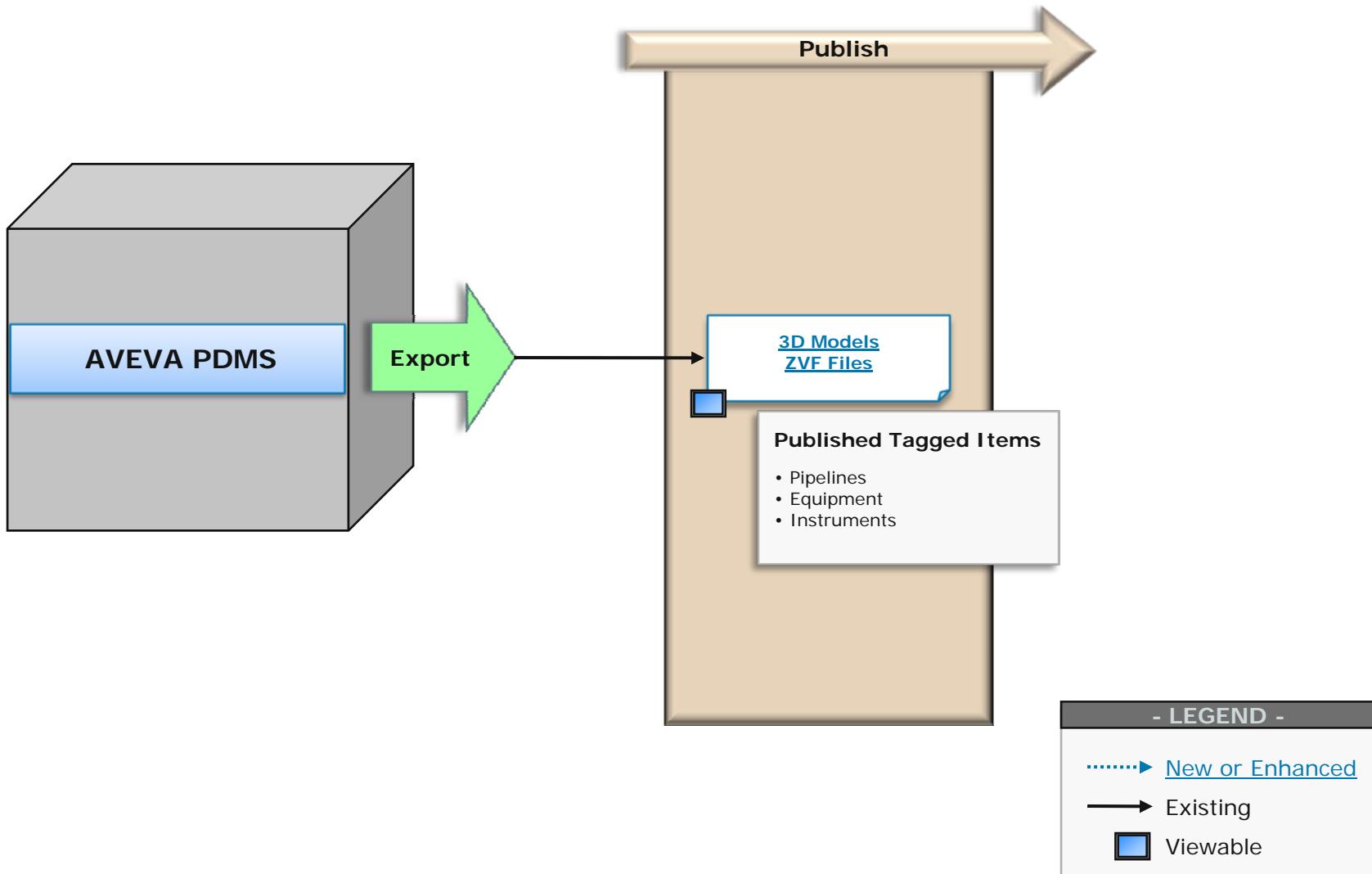
PDS Data Exchange Example

PDS publishes the following types of data and documents via Model Loader.



AVEVA PDMS Data Exchange Example

SmartPlant Model Loader can publish the following data to SmartPlant Foundation after the data is exported from AVEVA PDMS.



7.6 Activity 1 – Extending and Mapping an Existing Enumerated List

Complete the Chapter 7 – Activity 1 in the SmartPlant Foundation 2008 (4.2) Modeling and Mapping Activity workbook.

7.7 Activity 2 – Creating the Custom Interface

Complete the Chapter 7 – Activity 2 in the SmartPlant Foundation 2008 (4.2) Modeling and Mapping Activity workbook.

8

C H A P T E R

Mapping with SmartPlant P&ID

8. Mapping with SmartPlant P&ID

In the last chapter, the basic concepts of mapping were addressed. It was explained how map files are used to help the adapters create and process html files that are published to or retrieved from SmartPlant Foundation and move data between authoring tools and SPF.

This chapter illustrates how this process works using the SmartPlant P&ID tool as an example. Two new properties will be added to the authoring tool and mapped. One property is a simple, textual property type, and the other uses a brand new enumerated list.

Two different methods for making these changes are demonstrated in this chapter.



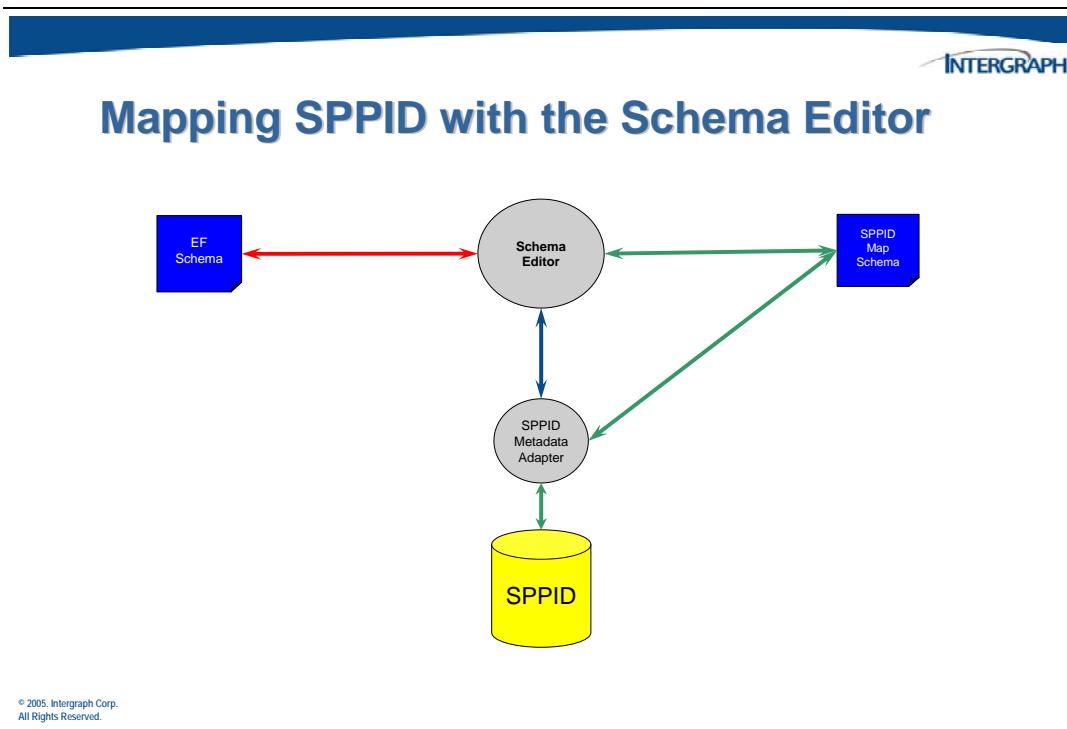
Mapping SPPID with the Schema Editor

There are two methods for extending the delivered schema:

- Make any necessary additions in the SPPID Meta-Schema using the Data Dictionary Manager**
 - this is the recommended process for adding new properties

- Make any changes using the schema editor, which will allow you to add those changes to the SPPID Meta-Schema, the SPPID tool map schema and the SmartPlant Schema**
 - this is the recommended process for adding a new enumerated list

The figure below shows the function of the metadata adapter with the schema editor.



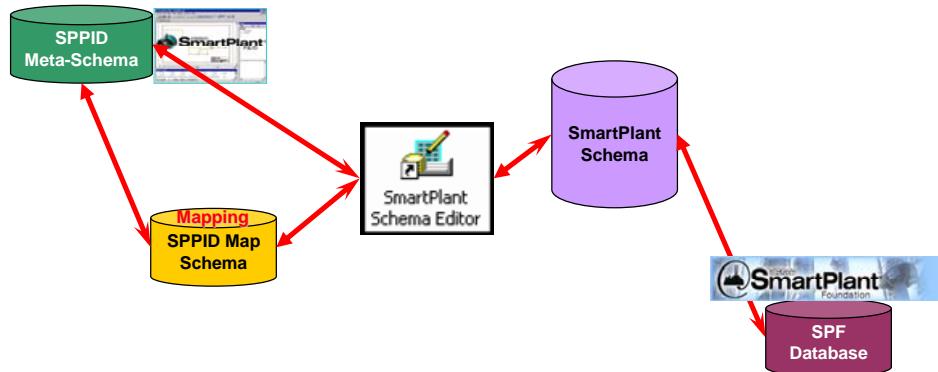
© 2005, Intergraph Corp.
All Rights Reserved.

In this case, the new custom property will first be made in the SmartPlant P&ID application using the Data Dictionary Manager tool. There is a specific reason for approaching the problem in this manner. The property to be added could be used by several different types of objects in SmartPlant P&ID: instruments, instrument loops, nozzles, pipe lines, and so forth. Since the new property will be used as a “global” *Plant Item* item type property, it can be added to the **Plant Item** table in the data dictionary, which makes it available to other classes through inheritance. The *PlantItem* item type is a classification that provides common relationships and attribution for all the types of objects that may exist independently of any other object. This item type is derived from the *ModelItem* class. The subclasses of *ModelItem* include all those types whose existence is not dependent on another.

Once the new property has been added with the Data Dictionary Manager, the schema editor is used to read this new property from the application, also called the tool meta schema. This is accomplished by a part of the software known as the metadata adapter. The task of the metadata adapter is to extract information from the meta schema and automatically update the contents of the tool map schema so that the meta schema and the tool map schema are synchronized.



Mapping SPPID with the Schema Editor



© 2005, Intergraph Corp.
All Rights Reserved.

- Changes are added to the **SPPID Meta schema** via the *Data Dictionary Manager* (new property defined).
- Any changes are automatically added to the **SPPID Map Schema** when the schema editor synchronizes the schema files.
- The same changes are added to the **SmartPlant Schema** using the schema editor *Map Environment*.
- Any unmapped changes to the schemas can be mapped using the schema editor in preparation for publish/retrieve operations.

This allows for the smooth flow of published data from SmartPlant P&ID into the SmartPlant Foundation database.

8.1 Adding Simple Properties to the Tool Meta Schema

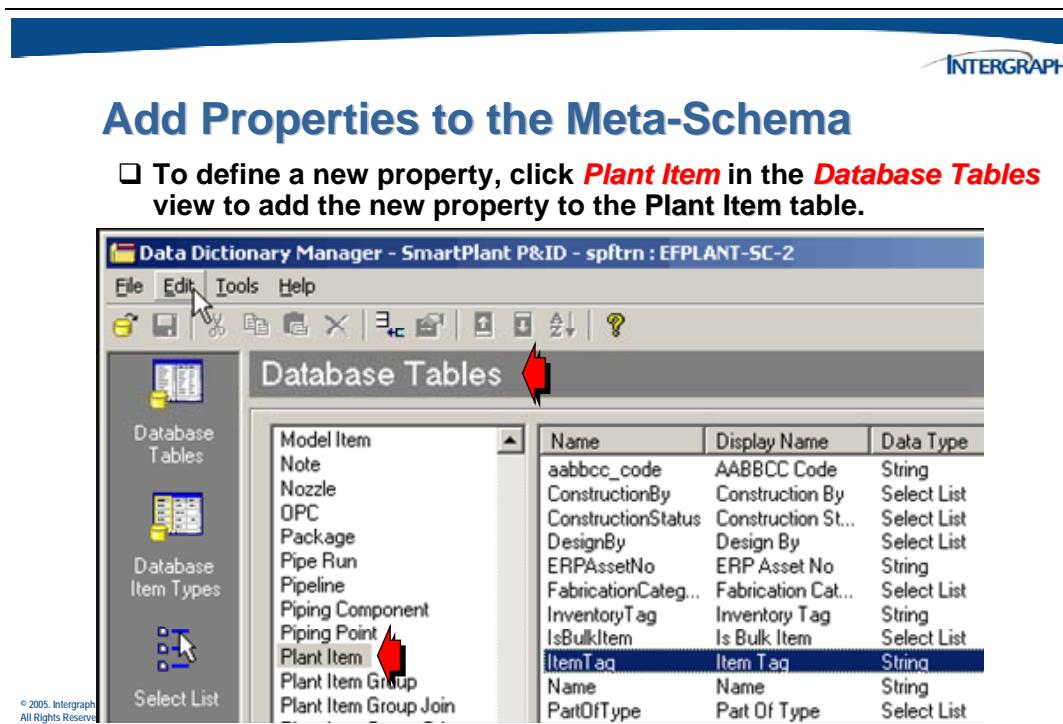
The procedure for adding properties to the authoring tools differs from tool to tool. The following example describes the procedure for adding a new property to the SmartPlant P&ID data model using SmartPlant Data Dictionary Manager (delivered with SmartPlant Engineering Manager). For more information about adding properties to SmartPlant P&ID, see the SmartPlant Data Dictionary Manager documentation.

To add a property to SmartPlant P&ID, open Data Dictionary Manager by clicking **Start > All Programs > Intergraph SmartPlant Engineering Manager > Data Dictionary Manager**.

The following example describes how to add a new property to the SmartPlant P&ID data model. This property includes the following:

- A simple property without an enumerated list

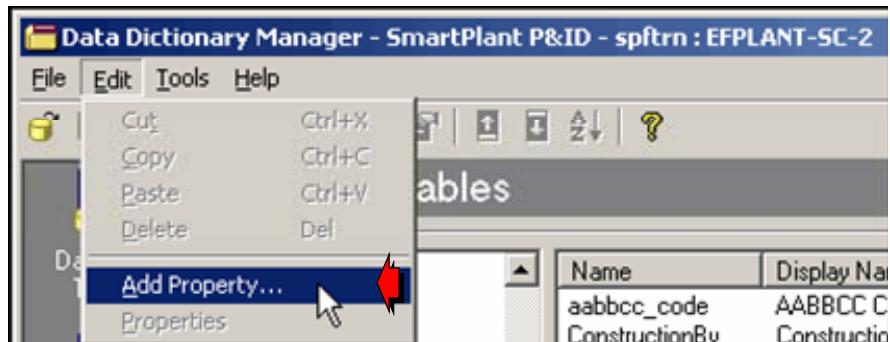
When you open Data Dictionary Manager, the **Database Tables** view is already selected. To select the database table to which you want to add a property, click the name of the table in the list.



After you select the table that you want, you can add a property to that table.

Add Properties to the Meta-Schema

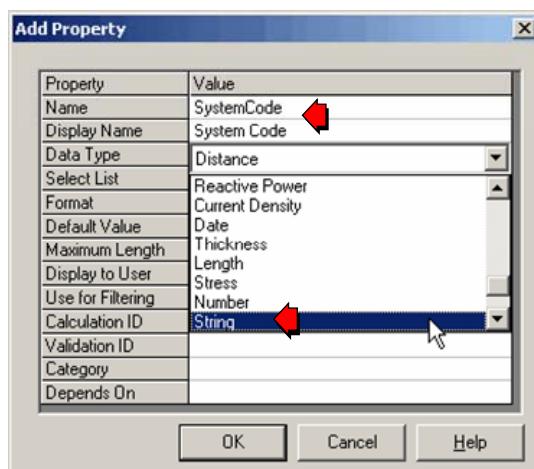
- From the menu, select **Edit > Add Property**.



The **Add Property** dialog box appears. Define information for the new property on the **Add Property** dialog box.

Add Properties to the Meta-Schema

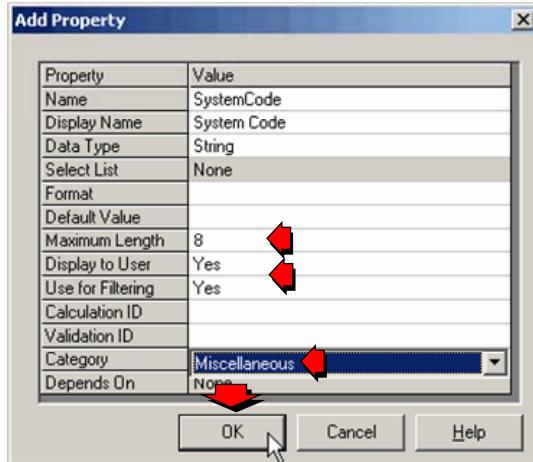
- In the **Add Property** dialog box, define attributes for the new property, including the **Name**, **Display Name**, and **Data Type**.



Add Properties to the Meta-Schema

- Continue to define attributes for the new property, including the **Maximum Length**, **Display**, **Filtering**, and **Category**.

- Click **OK** to add the new property to the Plant Item table.

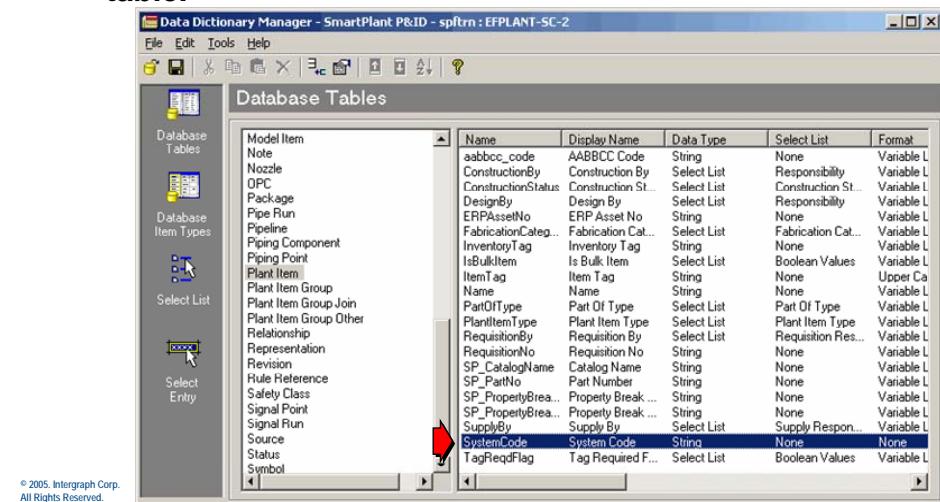


© 2005, Intergraph Corp.
All Rights Reserved.

When you click **OK**, the list of properties for the *Plant Item* table updates with the new property.

Add Properties to the Meta-Schema

The new **SystemCode** property displays in the Plant Item table.



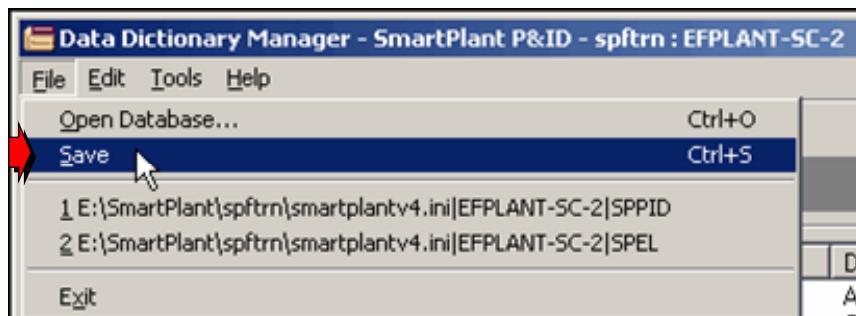
© 2005, Intergraph Corp.
All Rights Reserved.

After you add the property, save your changes to the SmartPlant P&ID meta schema.



Add Properties to the Meta-Schema

- From the menu, select ***File > Save***.



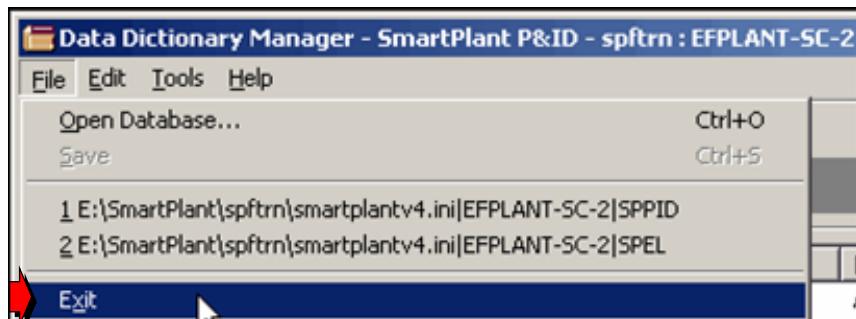
© 2005, Intergraph Corp.
All Rights Reserved.

Click the ***File > Exit*** command to close out of Data Dictionary Manager.



Add Properties to the Meta-Schema

- Select ***File > Exit*** from the menu.



© 2005, Intergraph Corp.
All Rights Reserved.

8.2 Updating the Tool Map Schema

Once the changes are part of the tool's meta schema, you need to make sure the new properties are part of the tool map file. To update the tool map file, use the Metadata adapter, which is accessed through the Schema Editor.



Synchronizing the Tool Data and Map File

Add a Property
to Tool Map
Schema

Add a Property
to SmartPlant
Schema

© 2008 Intergraph Corp.
All Rights Reserved.

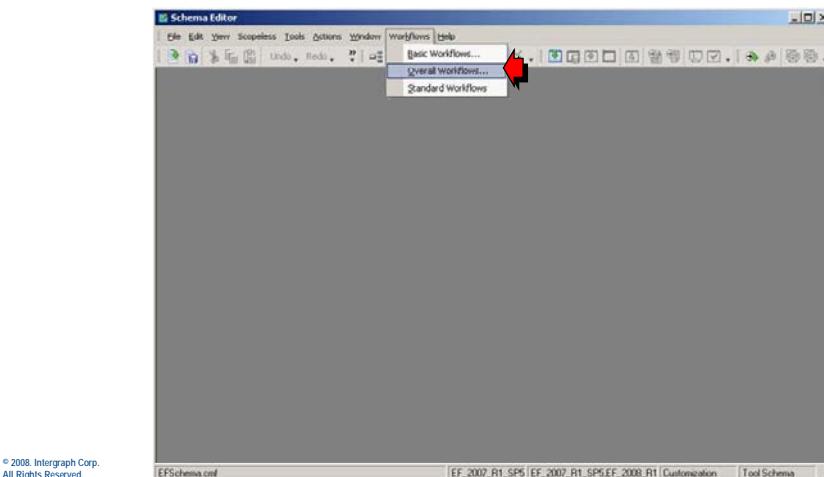
In the previous activity, we checked the CMF file out of SmartPlant Foundation and launched the Schema Editor from that tool. Once it was open, we saved the session file so that we could reopen it again later.

To reopen the session file, open the schema editor application. From the **Workflows** menu, choose **Overall Workflow**.



Synchronizing the Tool Data and Map File

- Launch the Schema Editor and from the **Windows** menu, choose the **Overall Workflow**.

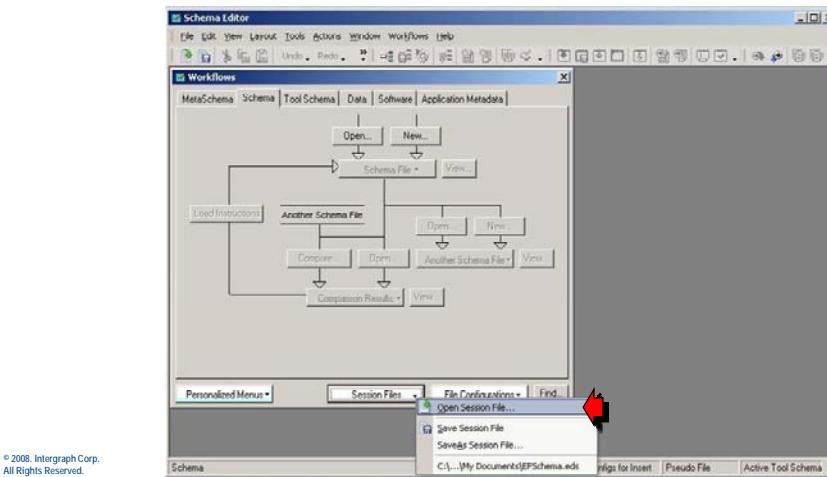


Using the **Session Files** button at the bottom of the window, open the session file you saved early. Typically, this .eds file is stored in the **My Documents** folder.



Synchronizing the Tool Data and Map File

- On the Schema tab, click the **Session Files** button and choose the **Open Session** file command.

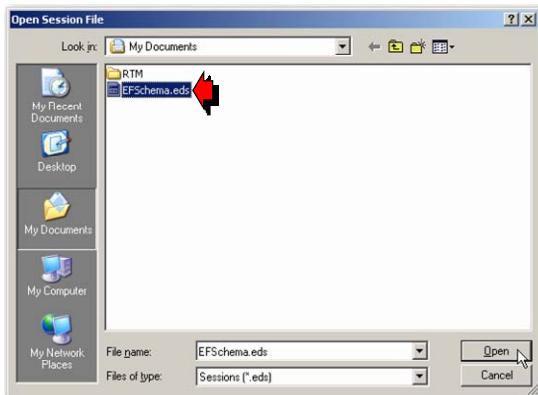


On the *Open* dialog box, find your session file.



Synchronizing the Tool Data and Map File

- Find the *EFSchema.eds* file in your *My Documents* folder, and open it.



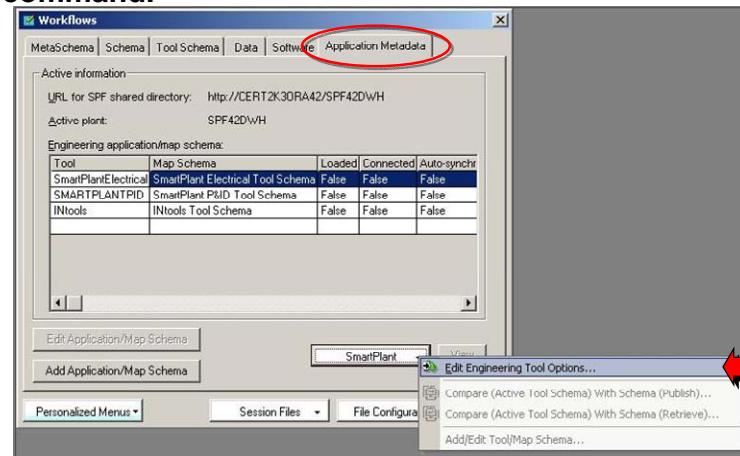
© 2008, Intergraph Corp.
All Rights Reserved.

With the session file open, move to the *Metadata Adapter* tab of the window. Click the *SmartPlant* button, and choose the *Edit Engineering Tools Options* command.



Synchronizing the Tool Data and Map File

- From the *Metadata Adapter* tab, click the *SmartPlant* button, and choose the *Edit Engineering Tool Options* command.



A information message may appear. This message just notifies you whether you have any tools registered with your plant that do not have metadata adapters. Click **OK** to continue.

Synchronizing the Tool Data and Map File

- A warning message indicates tools that do not have metadata adapters. Click **OK** to continue.



© 2008. Intergraph Corp.
All Rights Reserved.

Choose the SmartPlant PID metadata adapter, and turn on the check boxes for loading the map schema (tool map file) and connecting to the application schema (tool meta schema).

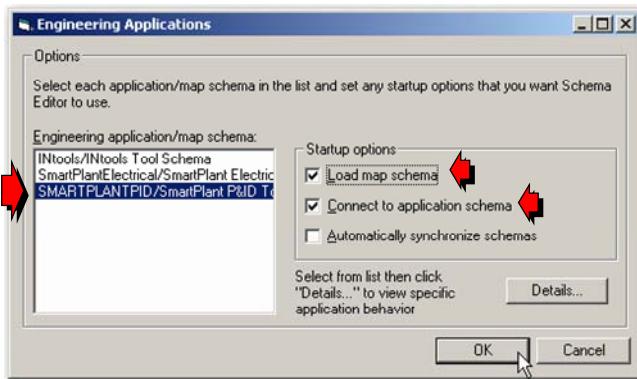
Note:

- If you wish, you can also select the **Automatically synchronize schemas** command to skip the **Synchronize** dialog box. However, with this selection, the software will not show you what changes will be made, and you will not get to choose whether the changes are made to the application schema or the map file.



Synchronizing the Tool Data and Map File

- ❑ Choose the metadata adapter for **SmartPlant P&ID**, and indicate that you want to launch the tool map schema and connect to the tool meta schema.



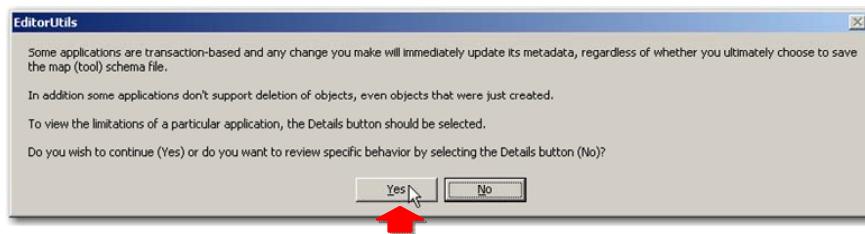
© 2008, Intergraph Corp.
All Rights Reserved.

Another information message appears. This message indicates that the metadata adapter may let you push changes to the tool database without going through the software designed to manage tools database changes.



Synchronizing the Tool Data and Map File

- ❑ The following message appears to warn you that the metadata adapter is capable of making changes immediately, without having to it is possible to make manually save the tool map schema. It also warns you that it is possible to make changes to the tool schema using the metadata adapter.
- ❑ Click **Yes** to continue.



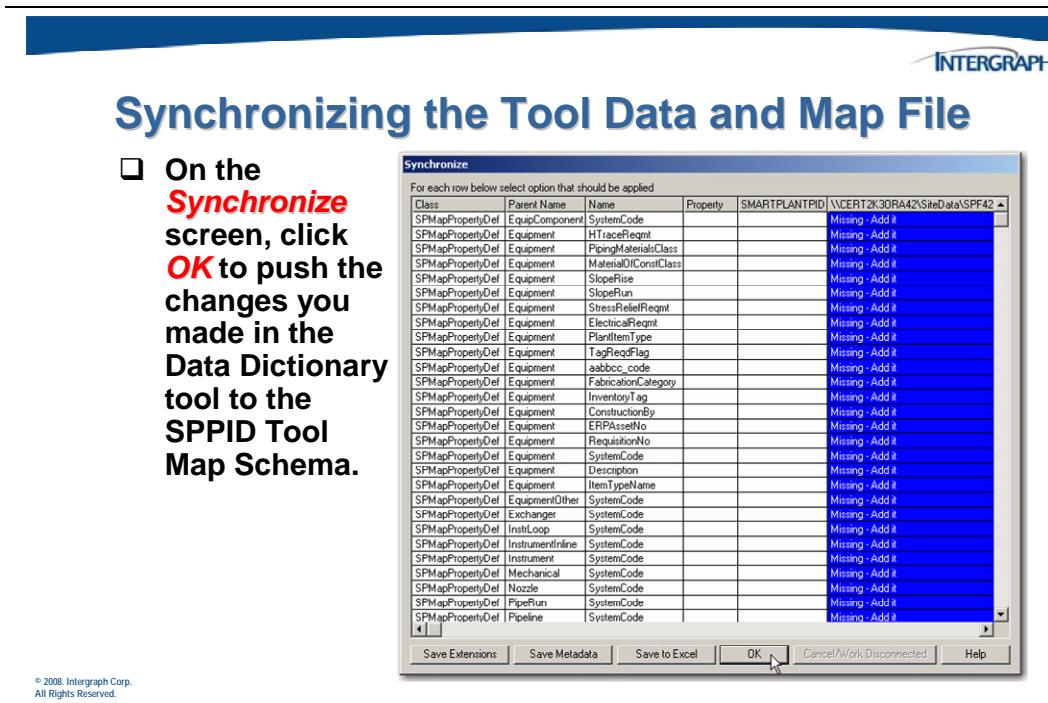
© 2008, Intergraph Corp.
All Rights Reserved.

The **Synchronize** dialog box displays the differences between the tool meta schema (tool database) and the tool map schema (tool map file). Where differences are found, the software shows you the name of the object that exists in only one of the two locations, and it gives you up to two options of what you want to do to synchronize.

Two columns represent the database and the map file. When an object is found in the database, you can choose whether to add it to the map file or to remove it from the database. Similarly, if the object is found in the database but not the map file, you can choose to add it to the database or remove it from the map file.

For our example, the metadata adapter finds the System Code property in the tool database, but not in the map file. We will choose the options to add the new properties to the map file.

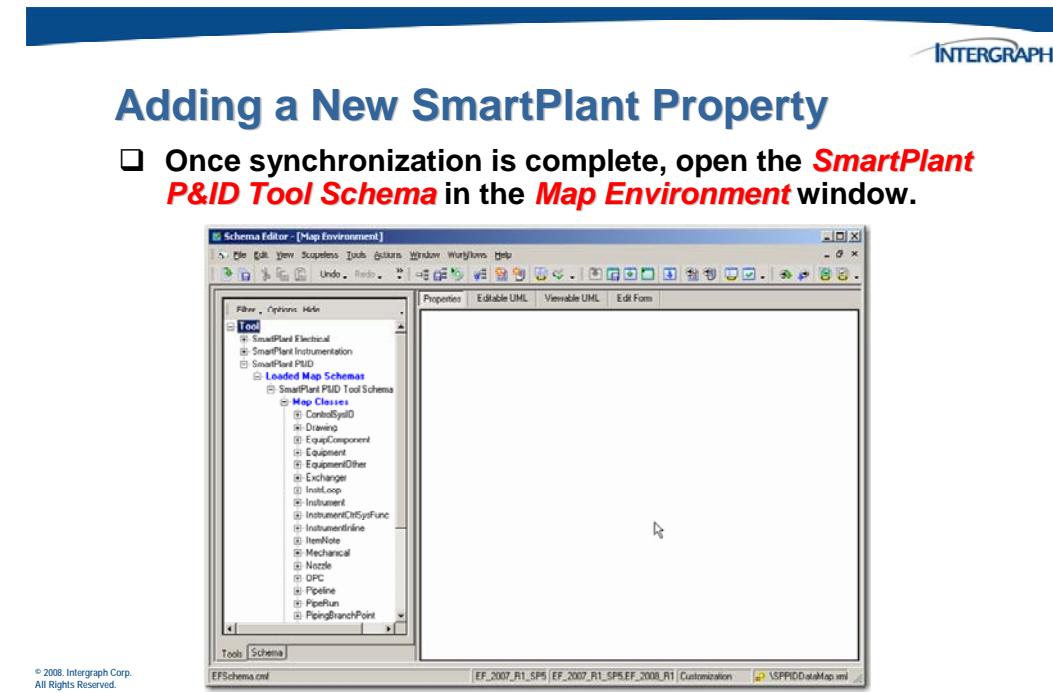
Click **OK** to accept the selections.



8.3 Extend the SmartPlant Schema

Now that the tool database and tool map file are synchronized, we are ready to create the new properties in the SmartPlant Schema so that we have properties on the SmartPlant side to which to map.

In the *Map Environment*, you can drill down beneath the *SmartPlant P&ID* node to view the information in the P&ID map file.



However, we are not ready to set up the mapping. We need to add our new property to our new custom interface, but we first need to make sure that our custom interface is defined so that the properties we expose with it are available to our class defs.



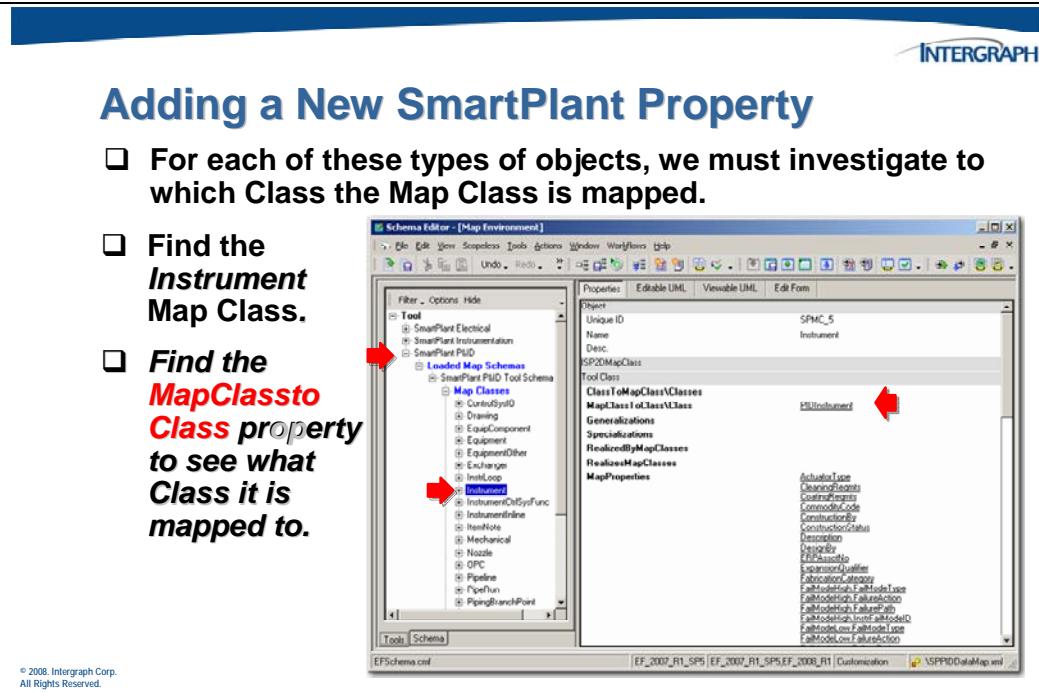
Adding a New SmartPlant Property

- Before we start adding custom properties to the SmartPlant Schema, we should investigate where our custom interface will be needed.
- We want to publish our **System Code** property for the following types of objects from SPPID:
 - **Instruments**
 - **Instruments Loops**
 - **Nozzles**
 - **Piping Components**
 - **Process Equipment Components**
 - **Piping Connectors**
 - **Pipelines**

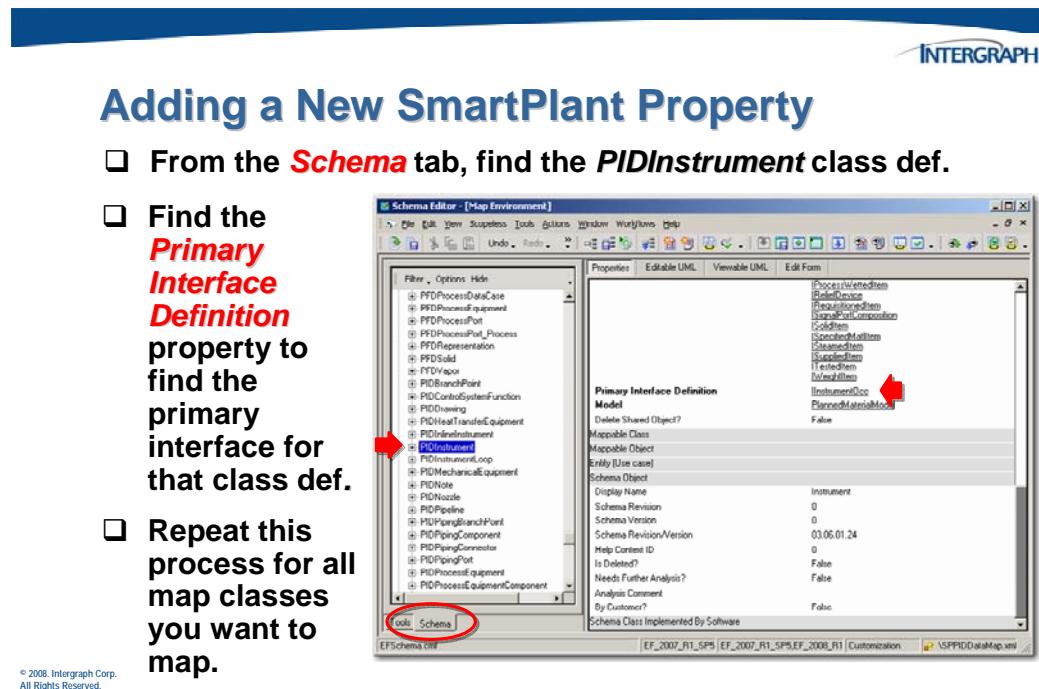
© 2008, Intergraph Corp.
All Rights Reserved.

Once we have a list of the objects that we want to have our property, we need to perform some research in the SmartPlant Schema to determine what class defs define those objects.

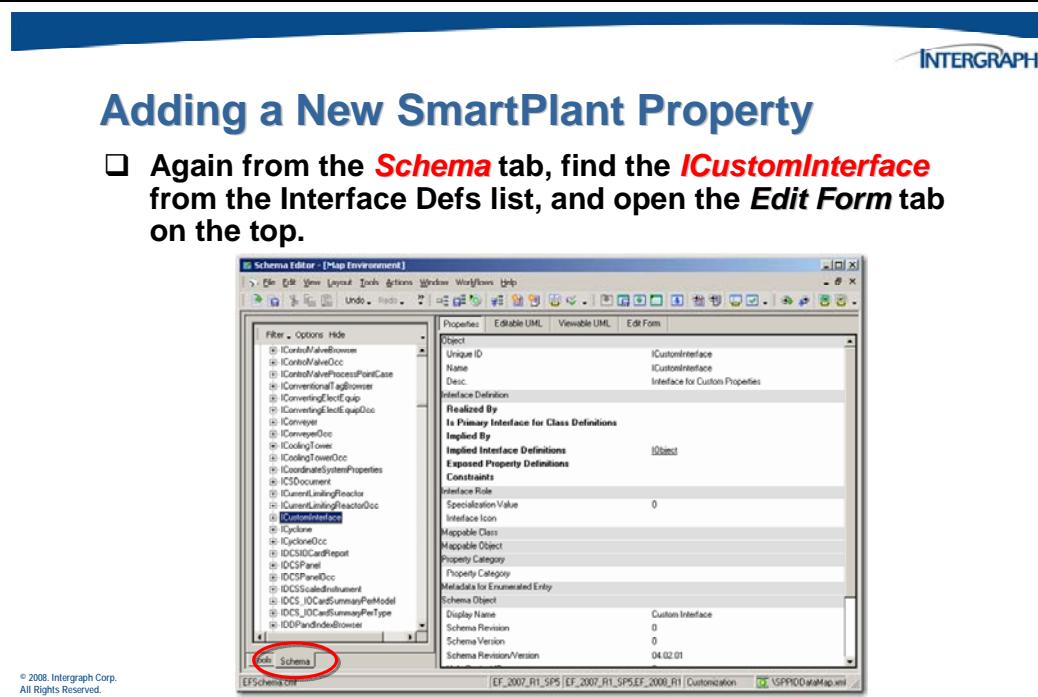
Find the map class of the type of object and, using the **Properties** view, find the SmartPlant class defs to which the map class is mapped.



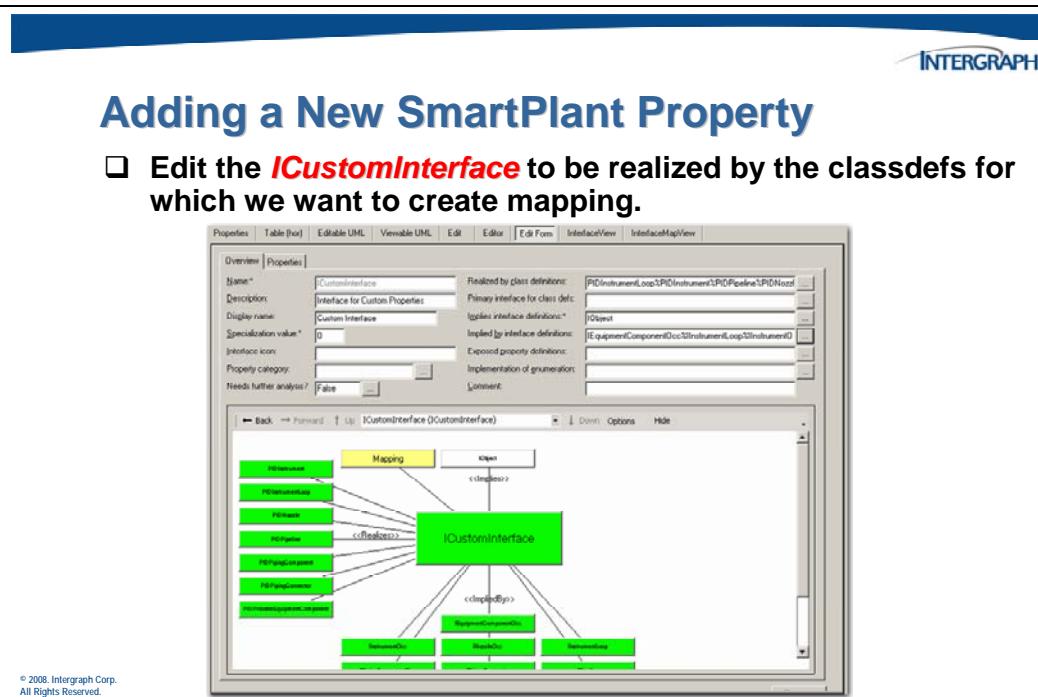
Then, from the **Schema** tab on the left side of the window, find those class defs and determine the primary interface for each applicable class def.



Once you have found the applicable class defs and primary interfaces, we are ready to update the custom interface for our custom properties.



From the **Edit Form** view of the **ICustomInterface** object, add the class defs that should realize the interface and the primary interfaces that should *imply* the interface.



The information below includes the list of relationships to create for the interface.



Adding a New SmartPlant Property

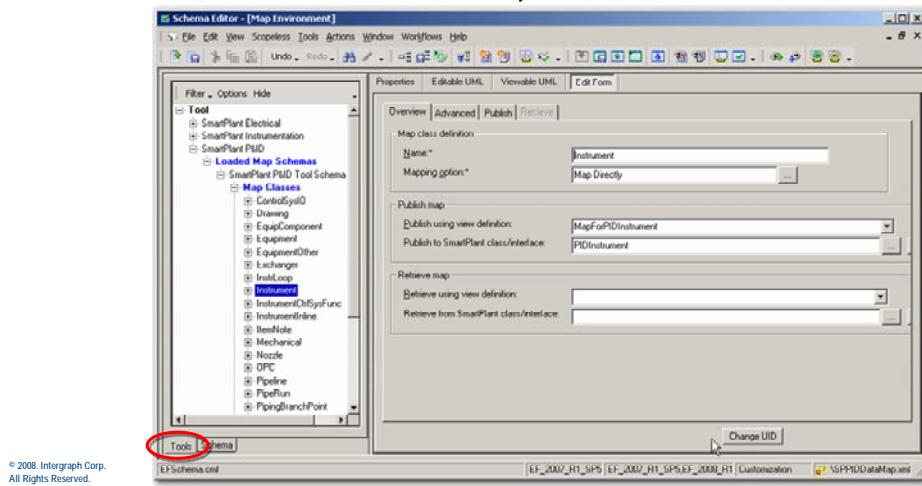
- Edit the interface so that it is realized by the class defs and implied by the primary interfaces of those class defs in the list below:

<u>Class Defs to Realize the Interface</u>	<u>Interfaces to Imply our Interface</u>
PIDInstrument	IInstrumentOcc
PIDInstrumentLoop	IInstrumentLoop
PIDNozzle	INozzleOcc
PIDPipingComponent	IPipingComponentOcc
PIDProcessEquipmentComponent	IEquipmentComponentOcc
PIDPipingConnector	IPipingConnector
PIDPipeline	IPipeline

Once you've made the necessary changes to the custom interface, we are ready to return to the tool map file, but returning to the **Tool** tab of the **Map Environment**. We can actually add the new properties from the same GUI that we use for creating our mapping relationships.

Adding a New SmartPlant Property

- Return to the **Tool** tab and from the **Map Classes** section, find and select **Instrument**, and the select the **Edit Form** tab.



Find the map class for which you want to map the new property. Open the **Edit Form** view for that map class, and go to the **Publish** tab.

Adding a New SmartPlant Property

- Open the **Publish** tab.
- The **System Code** property is available in the PID map schema, but is not in the SmartPlant schema.

© 2008, Intergraph Corp.
All Rights Reserved.

The new property is found in the tool map file, but not in the SmartPlant Schema.

Adding a New SmartPlant Property

- Select the **New Property Definition** button from middle/right side of the dialog.

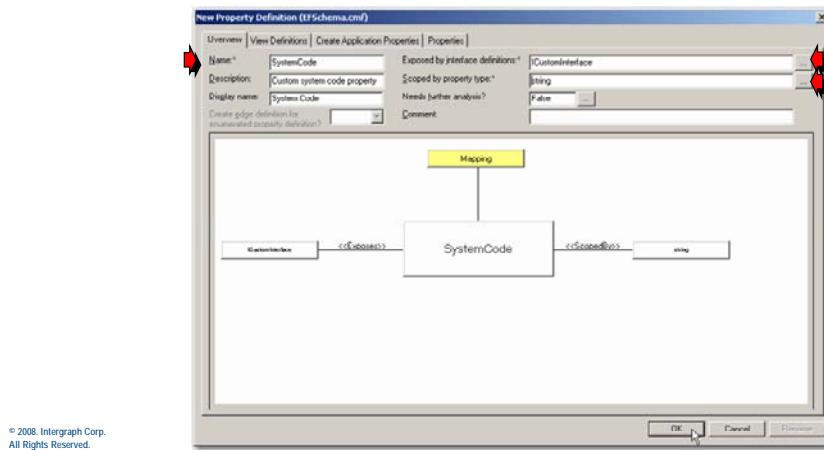
© 2008, Intergraph Corp.
All Rights Reserved.

Create the new SystemCode property, being sure to expose it on the ICustomInterface.



Adding a New SmartPlant Property

- Using the **New Property Definition** dialog box, create a new SmartPlant Schema **PropertyDef**. Be sure to put the property on the **ICustomInterface**.



8.4 Map the New Property for Publish

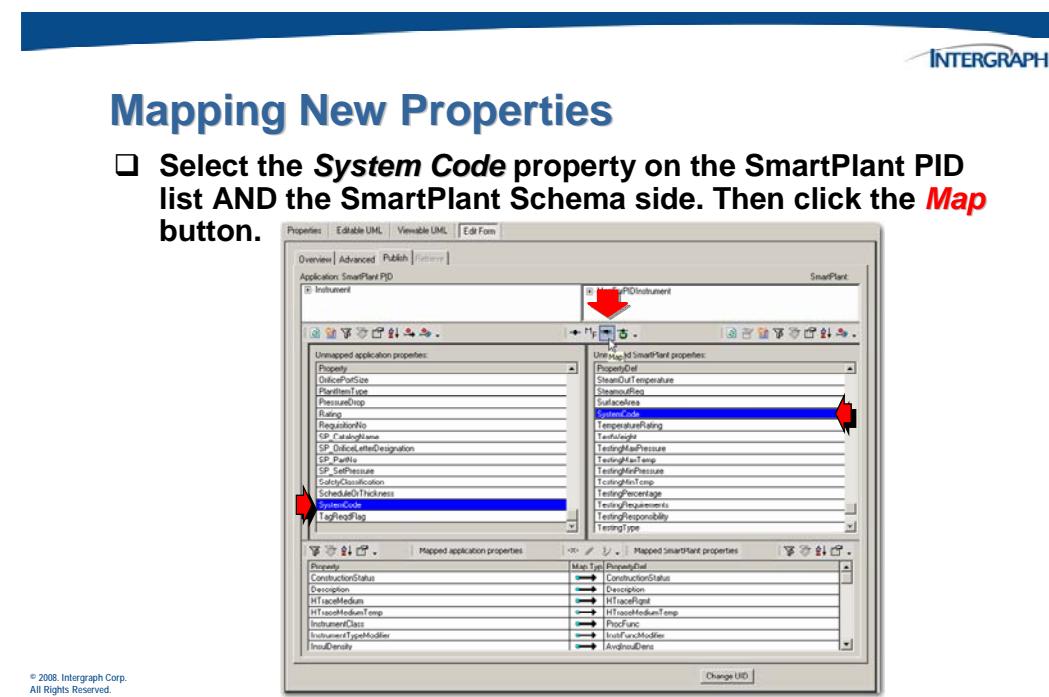
Now that the property exists in both schemas, we are ready to map them.



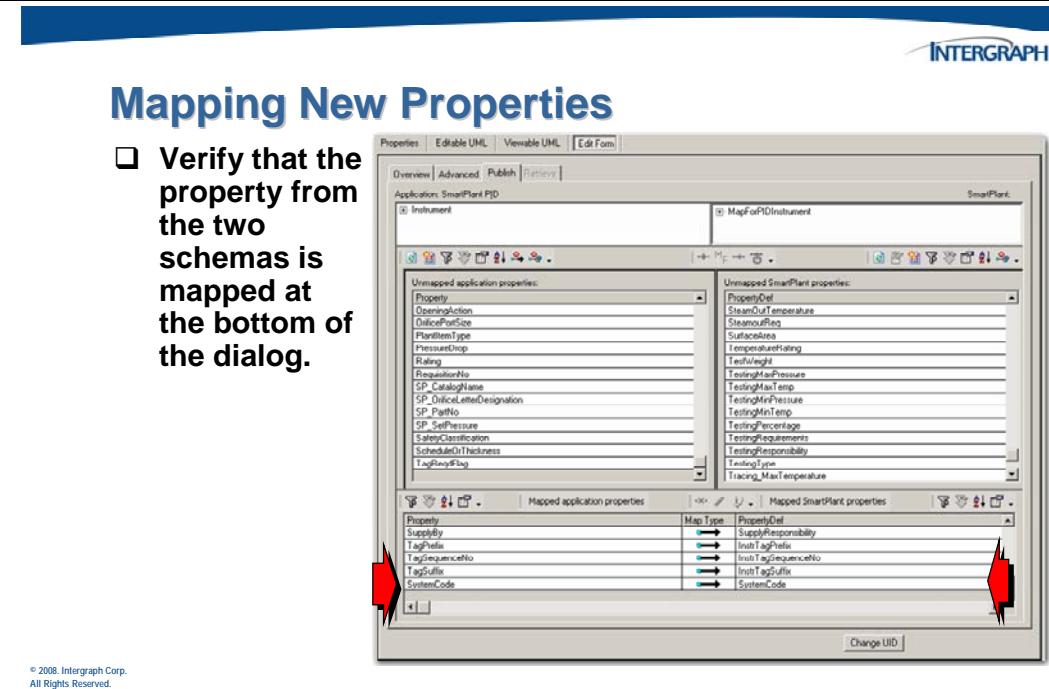
Mapping New Properties

Map Property in
Tool Map
Schema

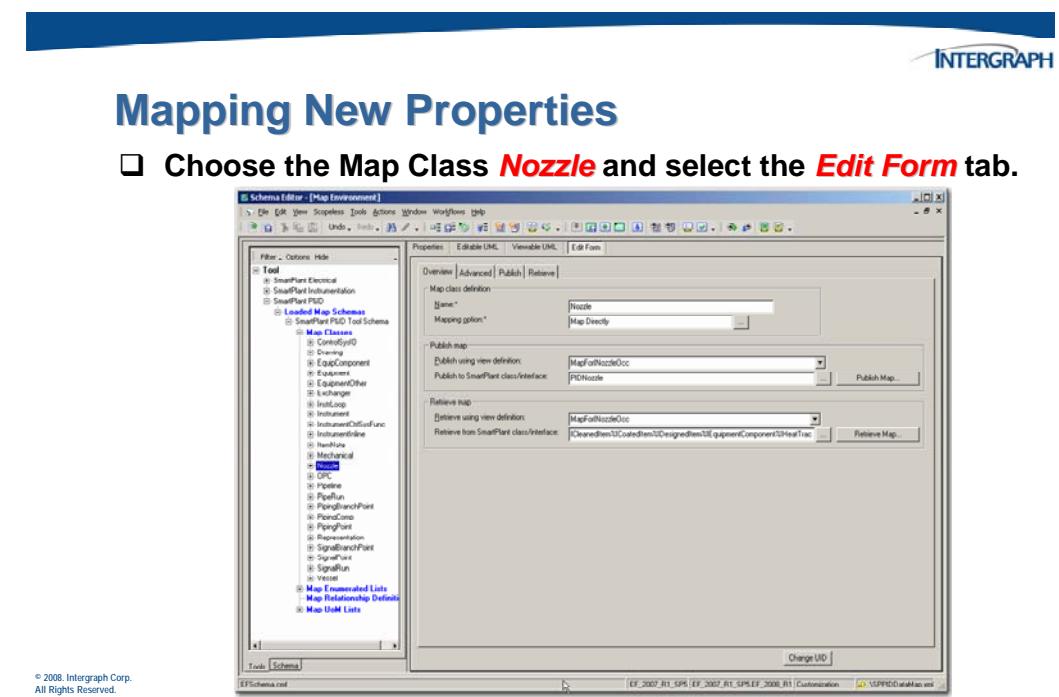
To create a map relationship between two properties, select the property on each side – the tool map file (left) and the SmartPlant Schema (right) – and click the **Map** button.



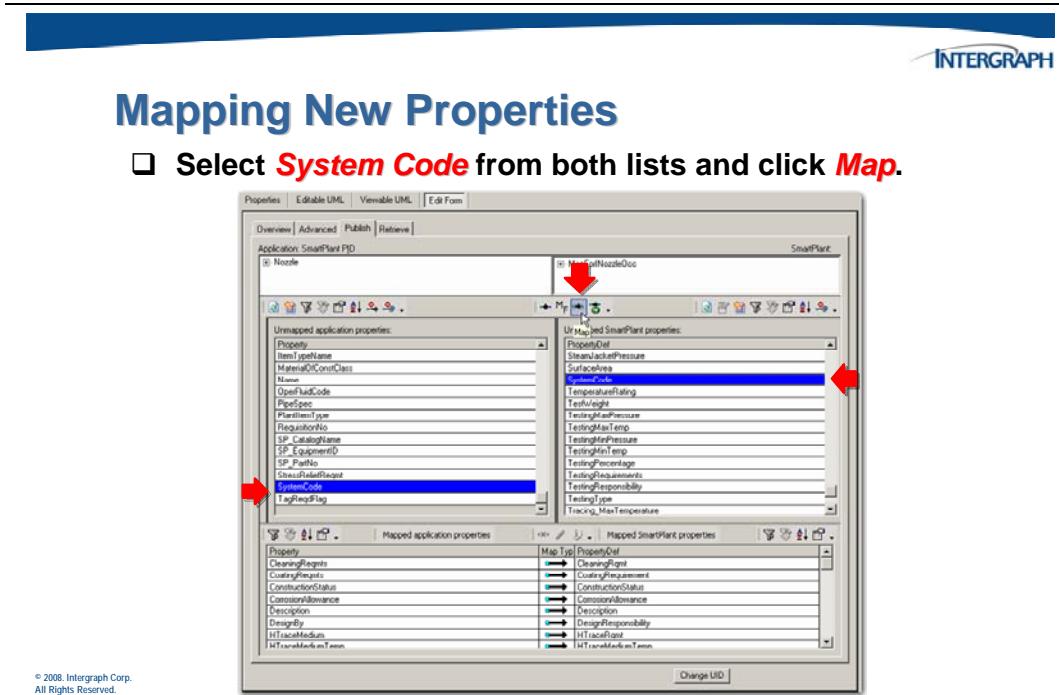
After mapping, check the control at the bottom of the dialog box to confirm the map.



Repeat the steps for additional map classes, such as Nozzle.



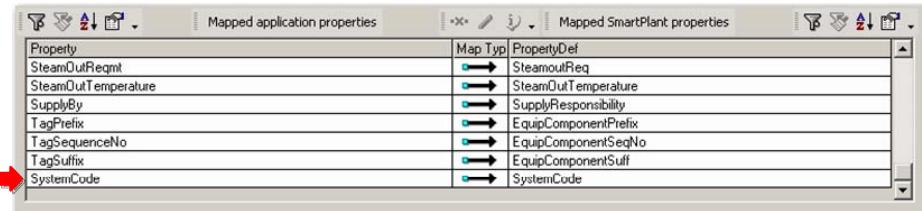
Find the properties on each side, and click the Map button.



Confirm the mapping before repeating the process for other map classes.

Mapping New Properties

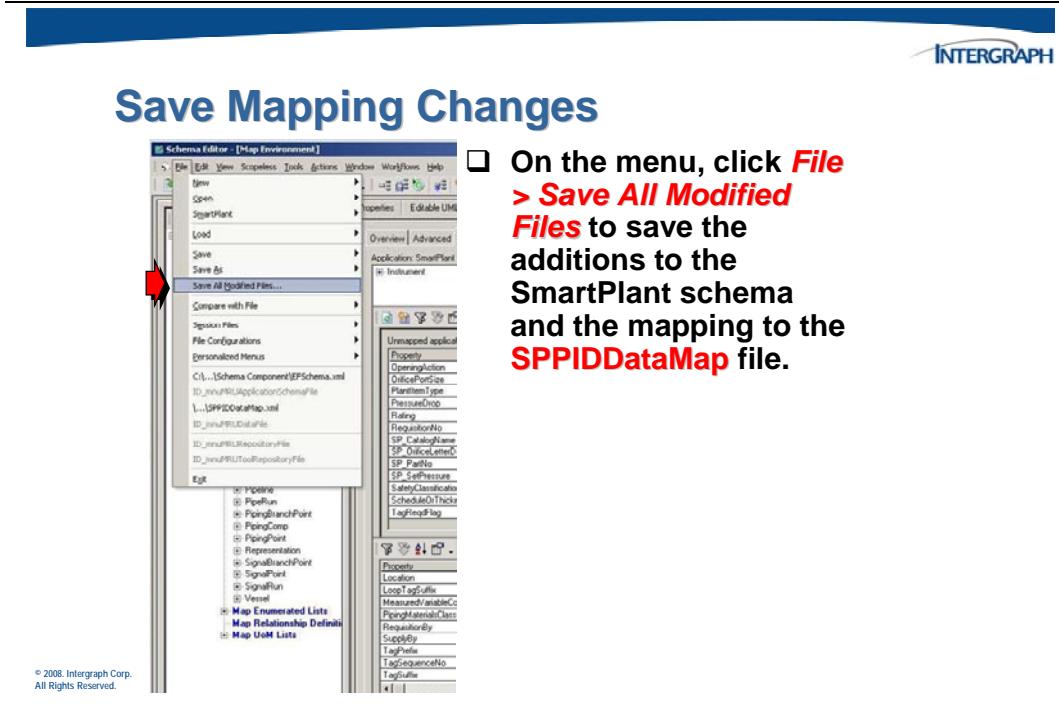
- Verify that the property from the two schemas is mapped at the bottom of the window.



8.5 Save the Schema Changes

Once you have finished mapping the property for all applicable map classes, you need to save the changes to the applicable files. Right now, all changes are stored only in local memory.

The **File > Save All Modified Files** command prompts you to save changes to all the modified schema files, but not the session file.



When prompted, save the changes to the CMF file.

Save Mapping Changes

- Verify the tool map schema name for the mapping changes and choose Yes.



© 2008, Intergraph Corp.
All Rights Reserved.

Save the changes to the SPPID tool map schema.

Save Mapping Changes

- Verify the CMF file for the property addition and choose Yes.



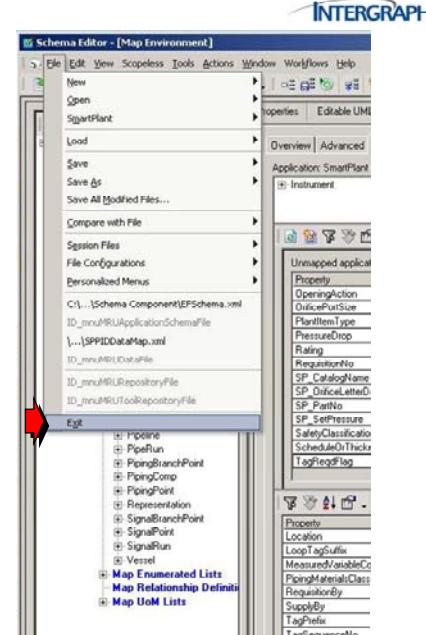
© 2008, Intergraph Corp.
All Rights Reserved.

Once the modified files have been saved, you can close the Schema Editor.

Save Mapping Changes

- On the menu, click **File > Exit** to close out of the schema editor.

© 2008, Intergraph Corp.
All Rights Reserved.



8.6 Activity 1 – Adding and Mapping a Custom Simple Property with SmartPlant P&ID

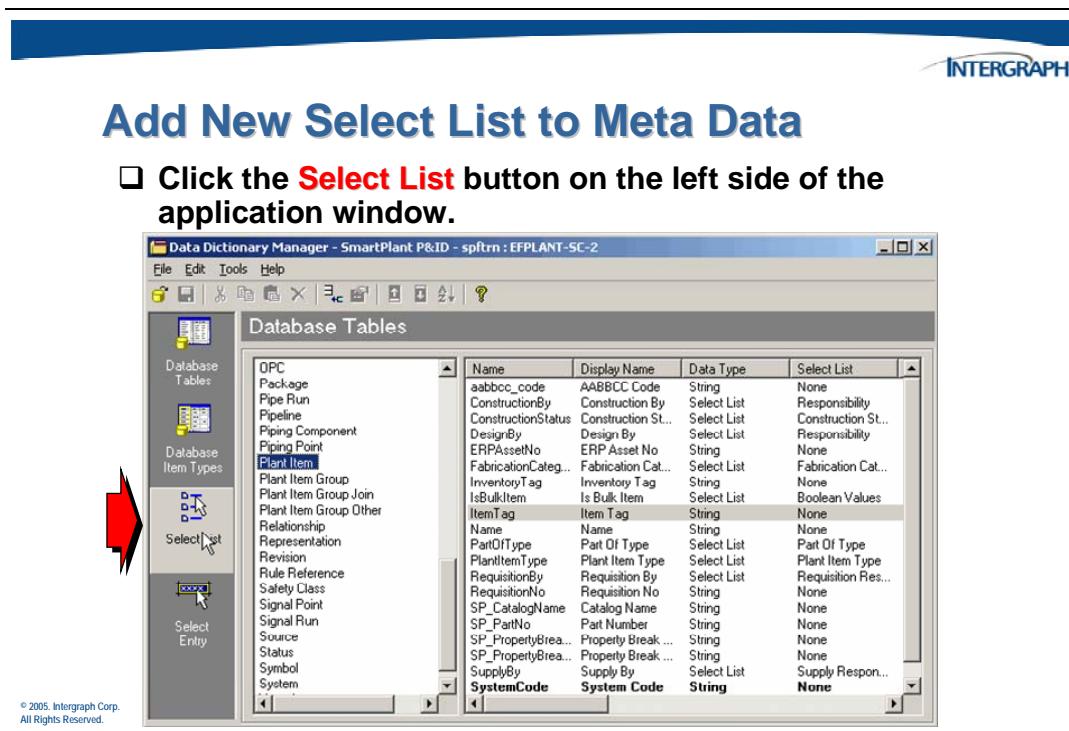
Complete the Chapter 8 – Activity 1 in the SmartPlant Foundation 2008 (4.2) Modeling and Mapping Activity workbook.

8.7 Adding a New Select List/Enum List

In the last section, the task of extending the SmartPlant P&ID meta data/SmartPlant Schema and adding a new simple property was demonstrated. A simple property is a property definition that uses a simple property type such as a string property type.

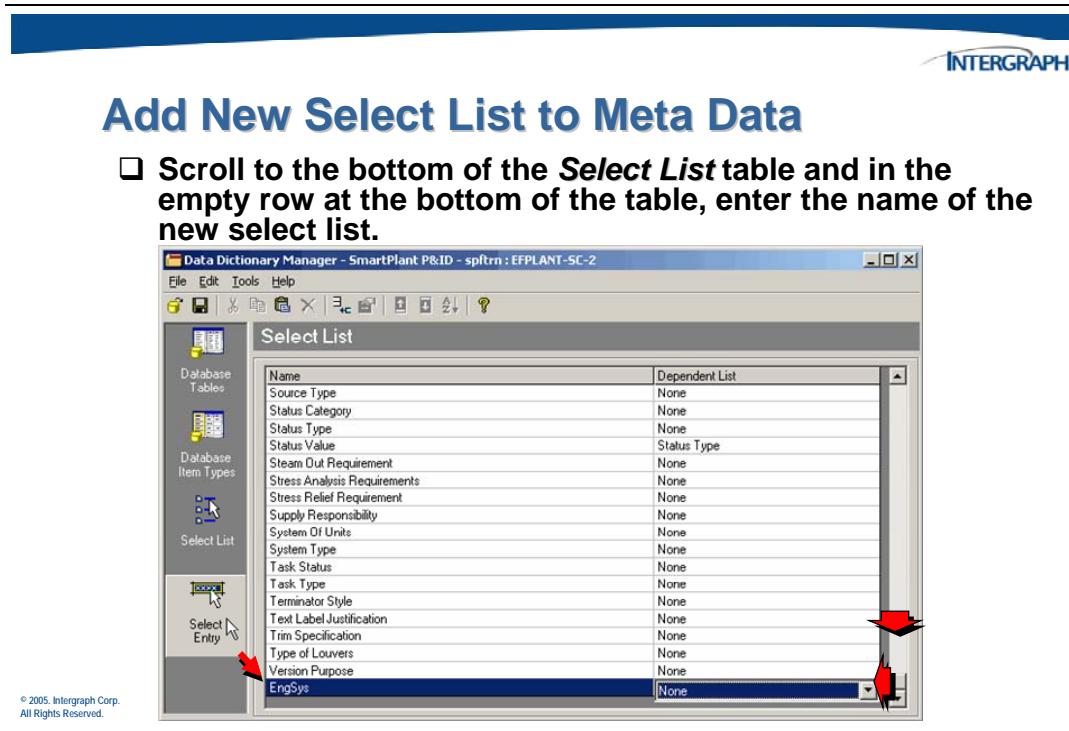
Another property type that is useful in SmartPlant is the **Select List**, which is a drop-down list. This is a more complex type of property. The select list equates to an **Enumerated List** in the tool map schema and the SmartPlant schema.

To add a new property to SPPID that uses a select list, begin by starting the **Data Dictionary Manager**.

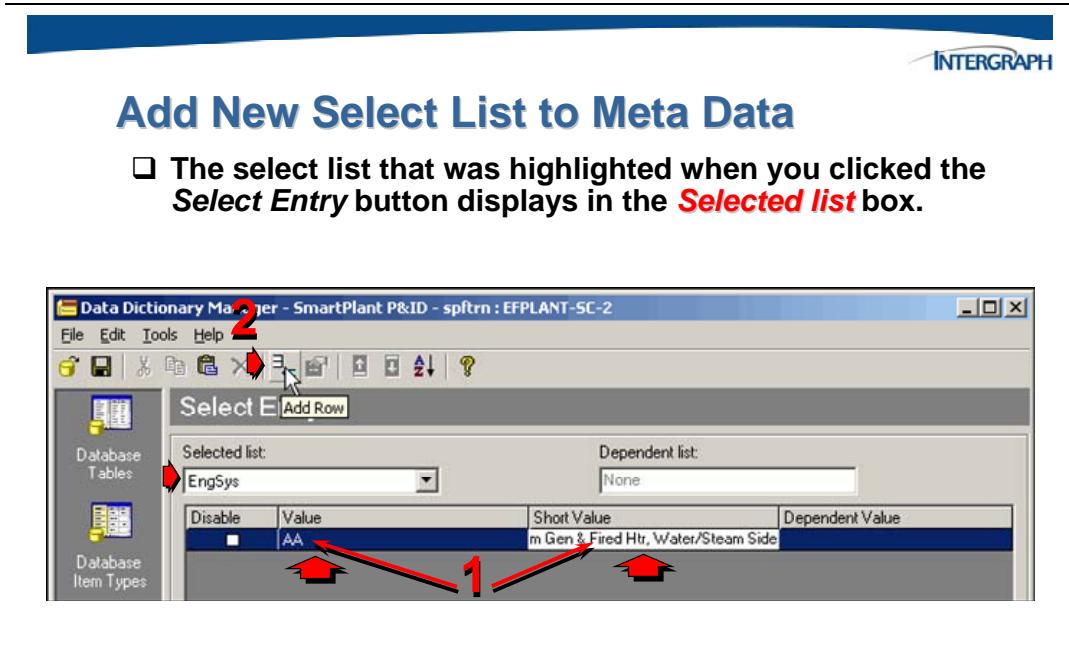


Next, define a property that has a select list associated with it. Before you create the property, create the select list for association with the property.

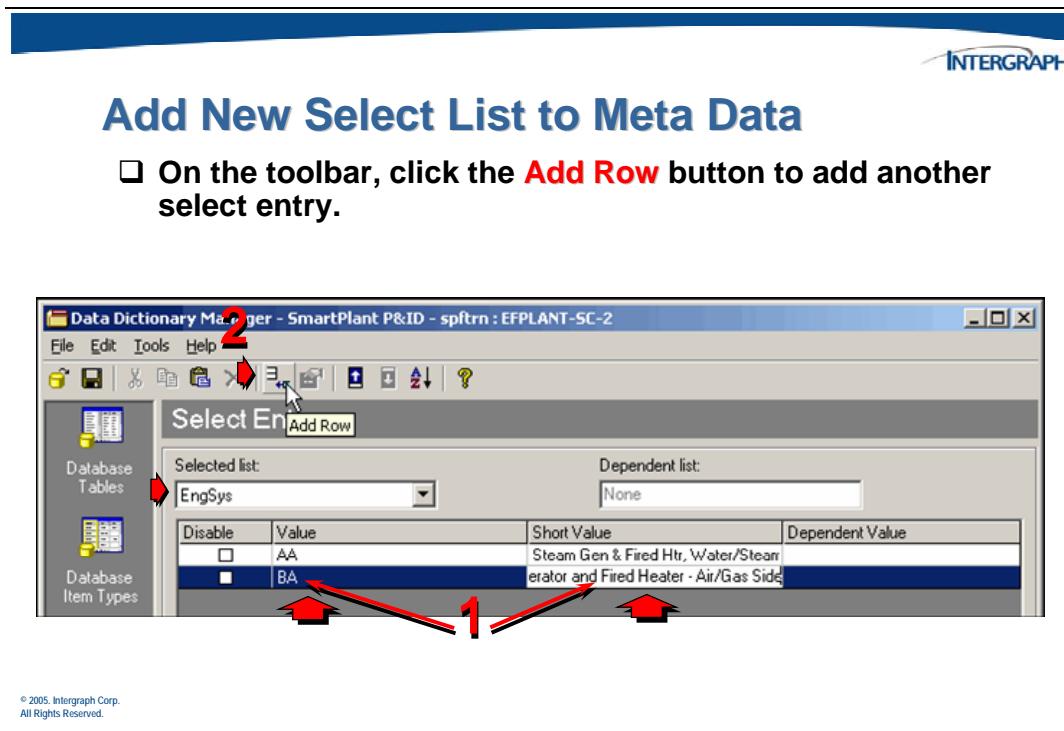
You can add a new select list by typing the name of the list in the blank row at the bottom of the *Select List* table.



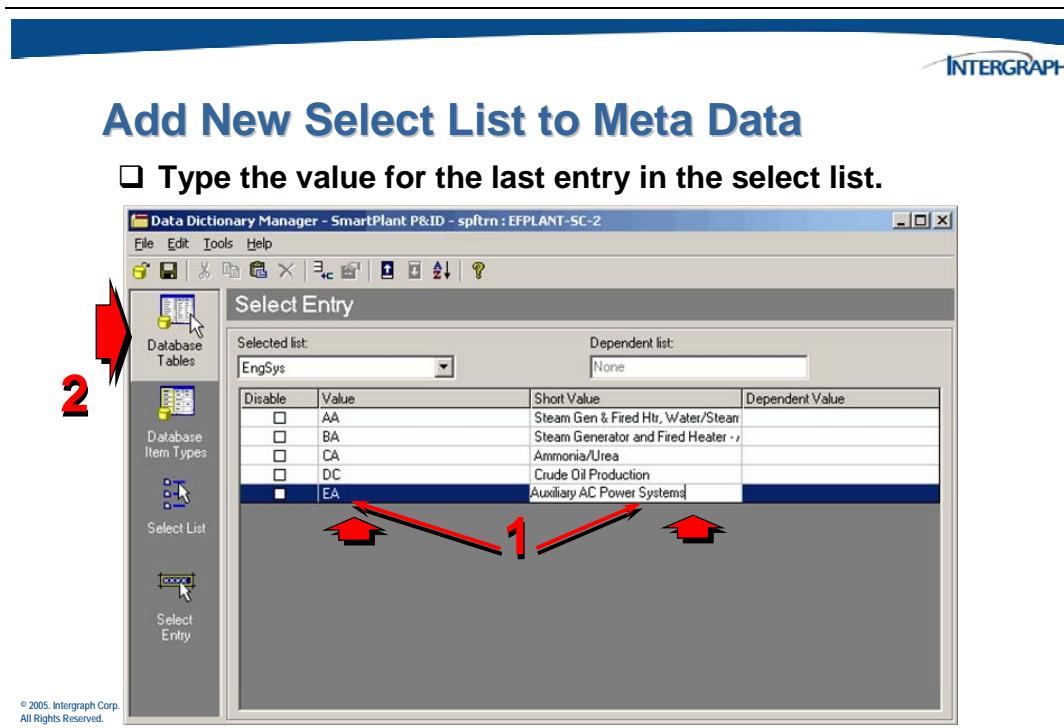
All that is required to define a new select list is the name of the select list. To define entries for the new select list, click the *Select Entry* button. Add the first select entry to the list by typing it in the first row of the table.



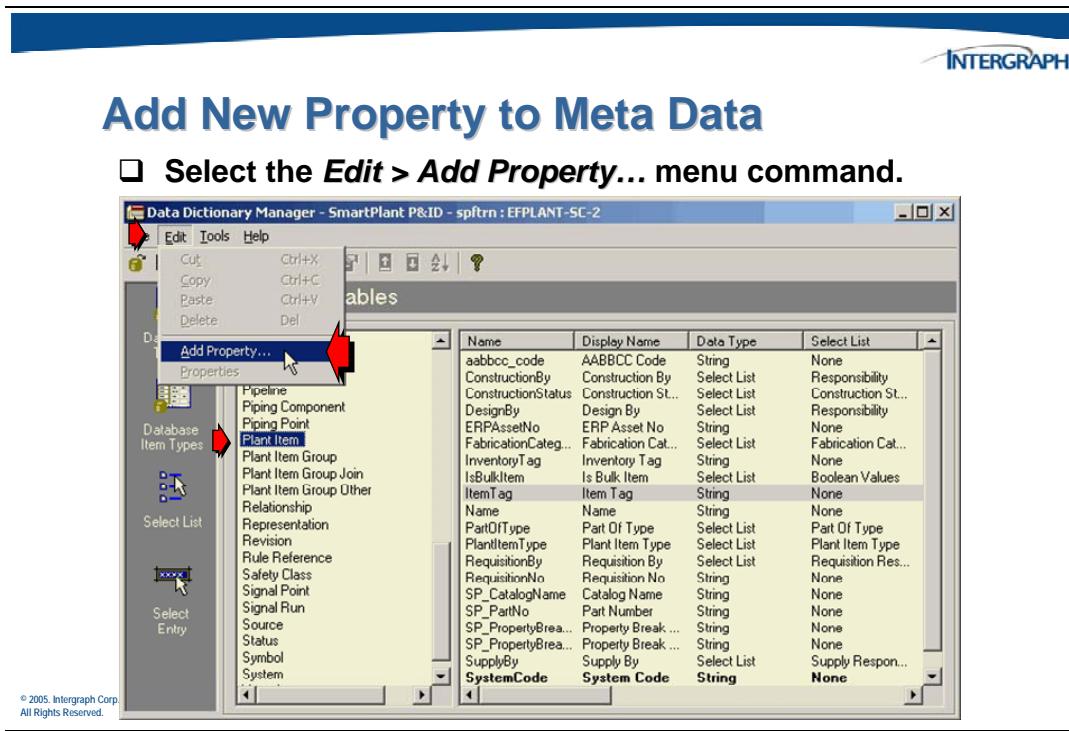
Define a second select entry for the select list.



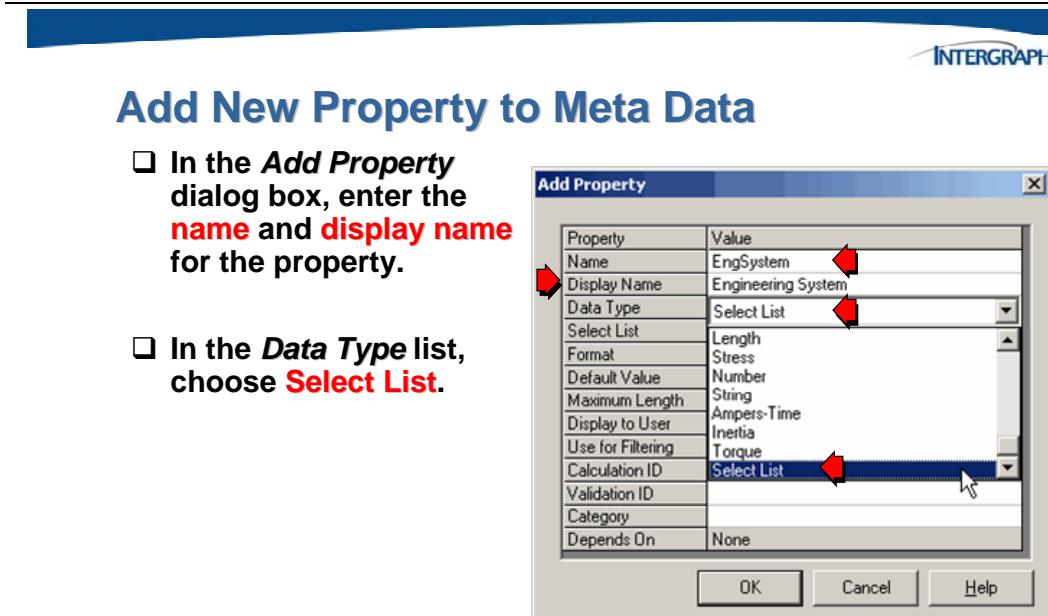
Define the remaining select list entries.



After you add the last entry to the select list, create the property with which you want to associate the select list. To create the property, click the *Database Tables* button then select *Plant Item* in the list of database tables.



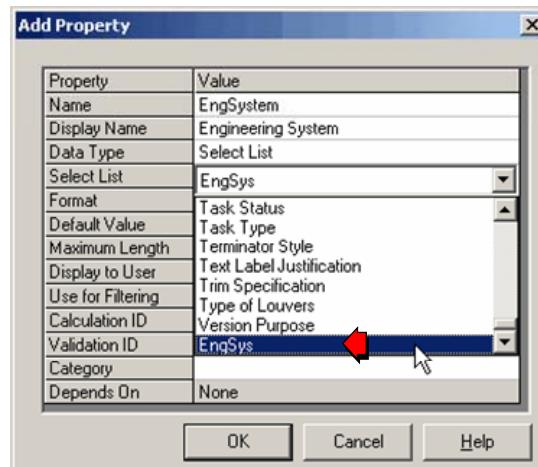
This adds the new **EngSystem** property to the *Plant Item* table. Choosing *Select List* as the data type allows you to select the appropriate list in the *Select List* box.



The new property will use the custom select list, EngSys, defined earlier.

Add New Property to Meta Data

- In the **Select List** list, choose the name of the select list you want to associate with the property.

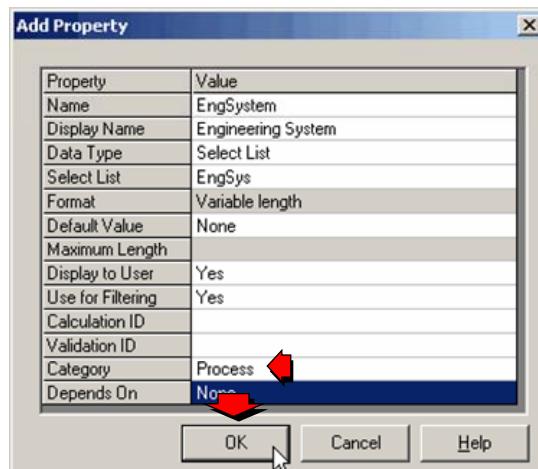


© 2005, Intergraph Corp.
All Rights Reserved.

Finish specifying the new property characteristics.

Add New Property to Meta Data

- In the **Category** list, choose **Process**.
- Click **OK** to save the new property.



© 2005, Intergraph Corp.
All Rights Reserved.

When you click **OK**, the list of properties for the *Plant Item* table updates with the new property.

The screenshot shows the Data Dictionary Manager interface. On the left, there's a tree view with categories like Database Tables, Database Item Types, and Select List. Under Database Tables, 'Plant Item' is selected. A red arrow points to the 'Plant Item' entry in the list. To the right is a table titled 'Database Tables' with columns: Name, Display Name, Data Type, and Select List. The table lists various properties for the Plant Item table, including 'EnoSystem' which has been added. Another red arrow points to the 'Select List' column for 'EnoSystem'.

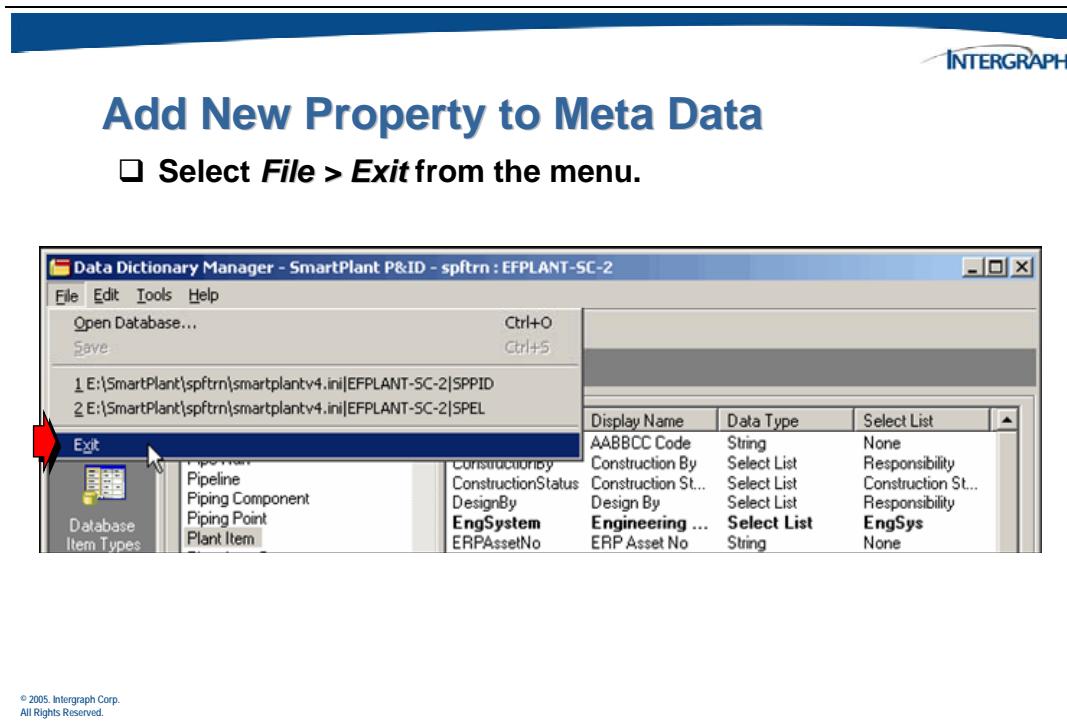
Name	Display Name	Data Type	Select List
aabcc_code	AABCC Code	String	None
ConstructionBy	Construction By	Select List	Responsibility
ConstructionStatus	Construction St...	Select List	Construction St...
DesignBy	Design By	Select List	Responsibility
EnoSystem	Engineering Sv...	Select List	EndSys
ERPAssetNo	ERP Asset No	String	None
FabricationCateg...	Fabrication Cat...	Select List	Fabrication Cat...
InventoryTag	Inventory Tag	String	None
IsBultite	Is Bulk Item	Select List	Boolean Values
ItemTag	Item Tag	String	None
Name	Name	String	None
PartOfType	Part Of Type	Select List	Part Of Type
PlantItemType	Plant Item Type	Select List	Plant Item Type
RequisitionBy	Requisition By	Select List	Requisition Res...
RequisitionNo	Requisition No	String	None
SP_CatalogName	Catalog Name	String	None
SP_PartNo	SP Part No	Part Number	None
SP_PropertyBreak...	SP Property Break...	String	None
SP_PropertyBreak...	SP Property Break...	String	None
SupplyBy	Supply By	Select List	Supply Respons...
SystemCode	System Code	String	None

After adding the property, save the changes to the SmartPlant P&ID meta data.

The screenshot shows the Data Dictionary Manager interface again. This time, the 'File' menu at the top is open, and the 'Save' option is highlighted with a red arrow. The menu also includes 'Open Database...' and 'Exit'. Below the menu, there's a list of recent files and a preview pane showing the 'Database Item Types' table. To the right is a properties table for the Plant Item table, identical to the one shown in the previous screenshot.

Display Name	Data Type	Select List
AABCC Code	String	None
Construction By	Select List	Responsibility
Construction St...	Select List	Construction St...
Design By	Select List	Responsibility
EnoSystem	Select List	EndSys
ERPAssetNo	ERP Asset No	String

Clicking **File > Exit** closes Data Dictionary Manager.

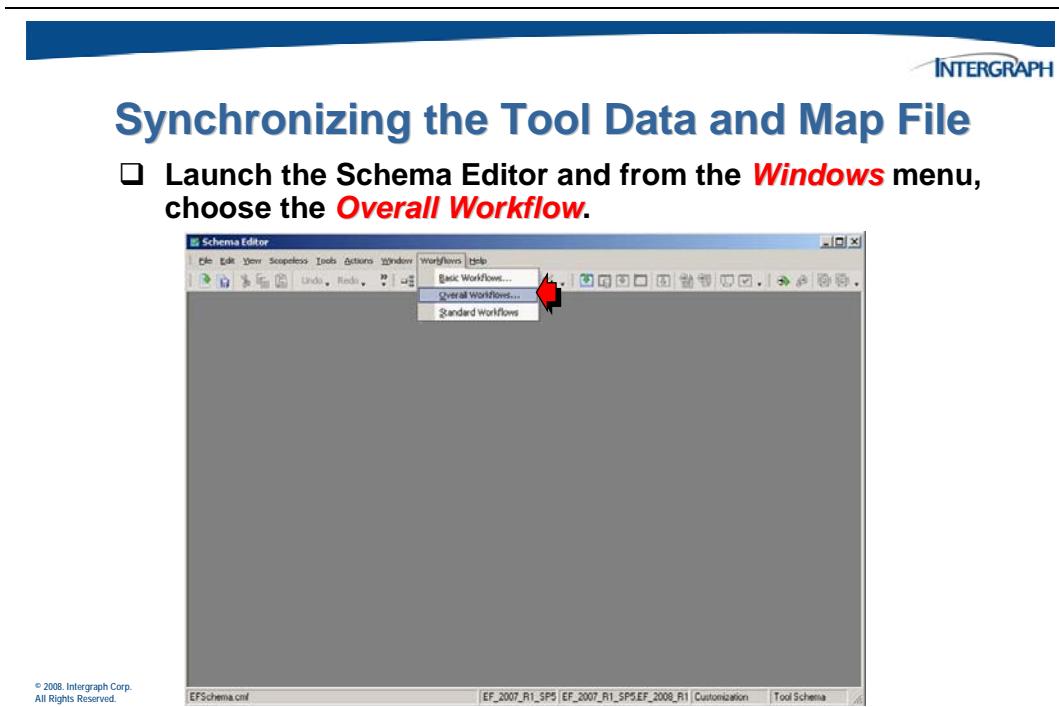


8.8 Extend the Tool Map Schema

After the tool meta schema is extended, you need to synchronize the tool map file to include the new select list and values. For this, use the tool's metadata adapter, available from the Schema Editor.

In this example, we still have our CMF file checked out of the Desktop Client. We will access that file with our saved session file. If you do not currently have your site's CMF file checked out, you must first launch the Desktop Client, find the CMF file, and check it out from the plant you are working in.

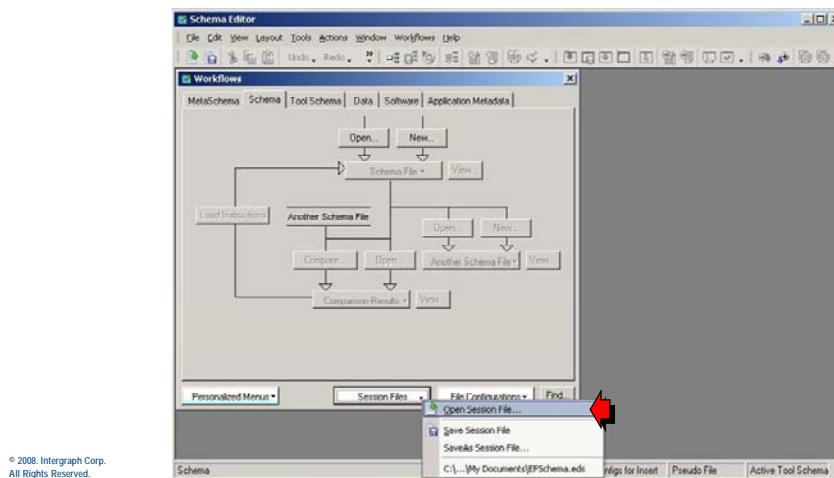
Launch the Schema Editor.



Open the ***Overall Workflow*** and click the ***Session File*** button.

Synchronizing the Tool Data and Map File

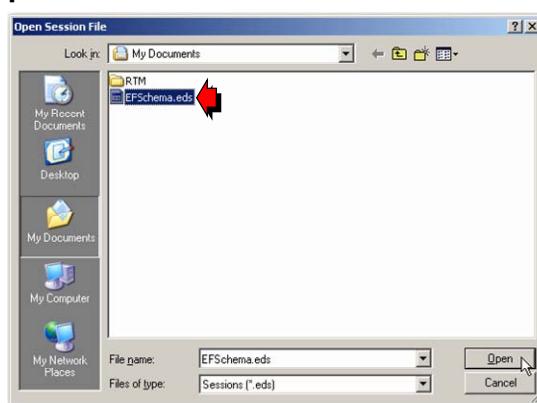
- On the Schema tab, click the ***Session Files*** button and choose the ***Open Session*** file command.



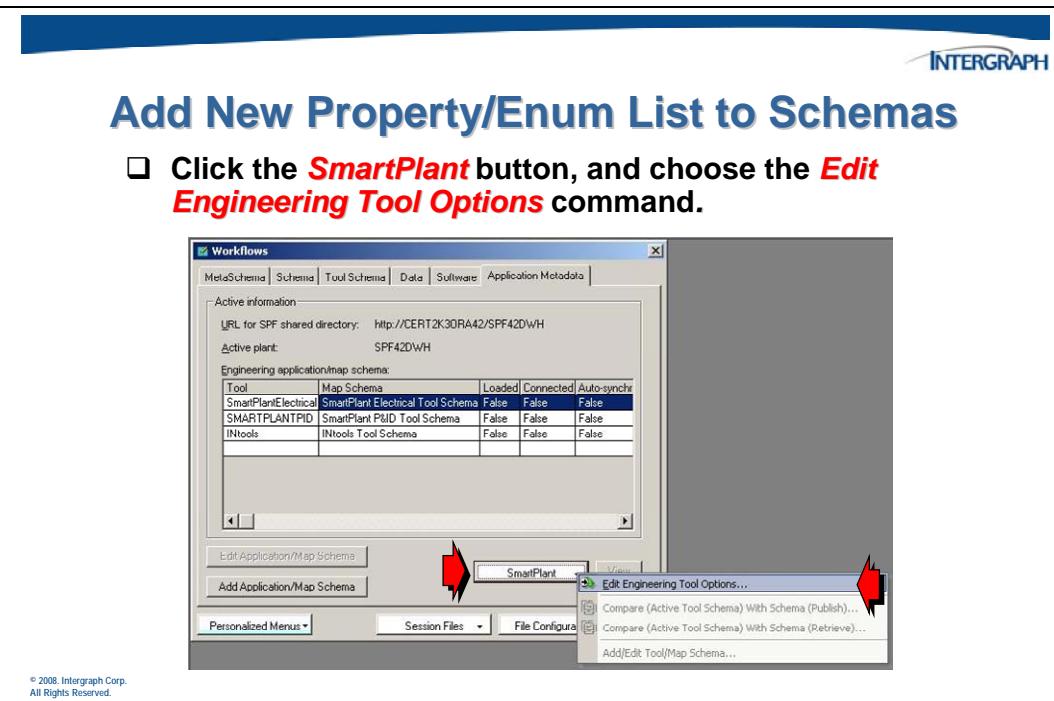
Find your saved session file to connect to the SmartPlant schema for the plant for which you are mapping.

Synchronizing the Tool Data and Map File

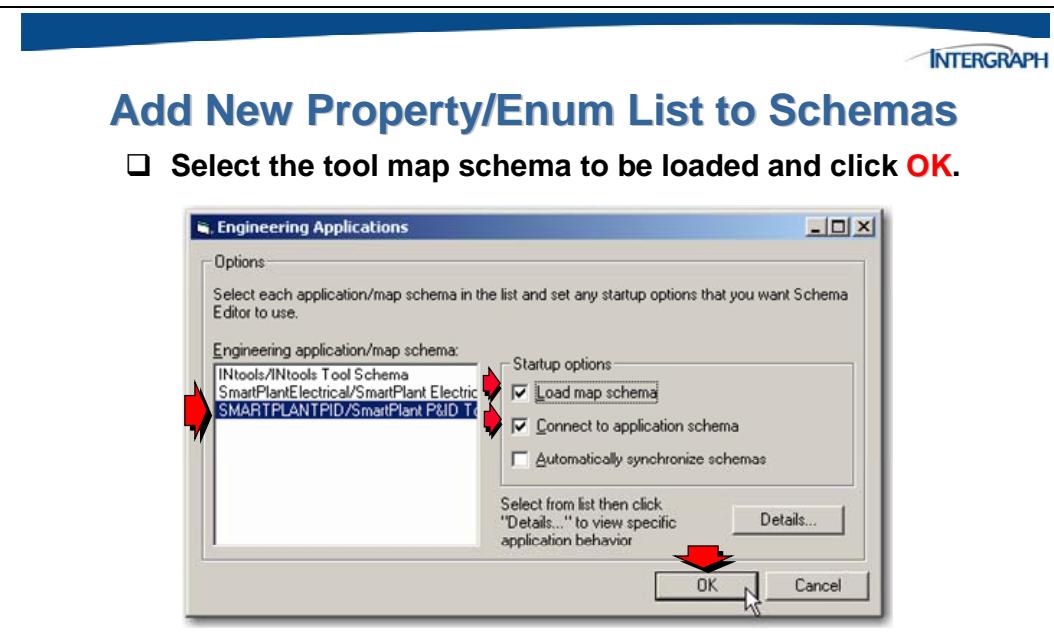
- Find the ***EFSchema.eds*** file in your ***My Documents*** folder, and open it.



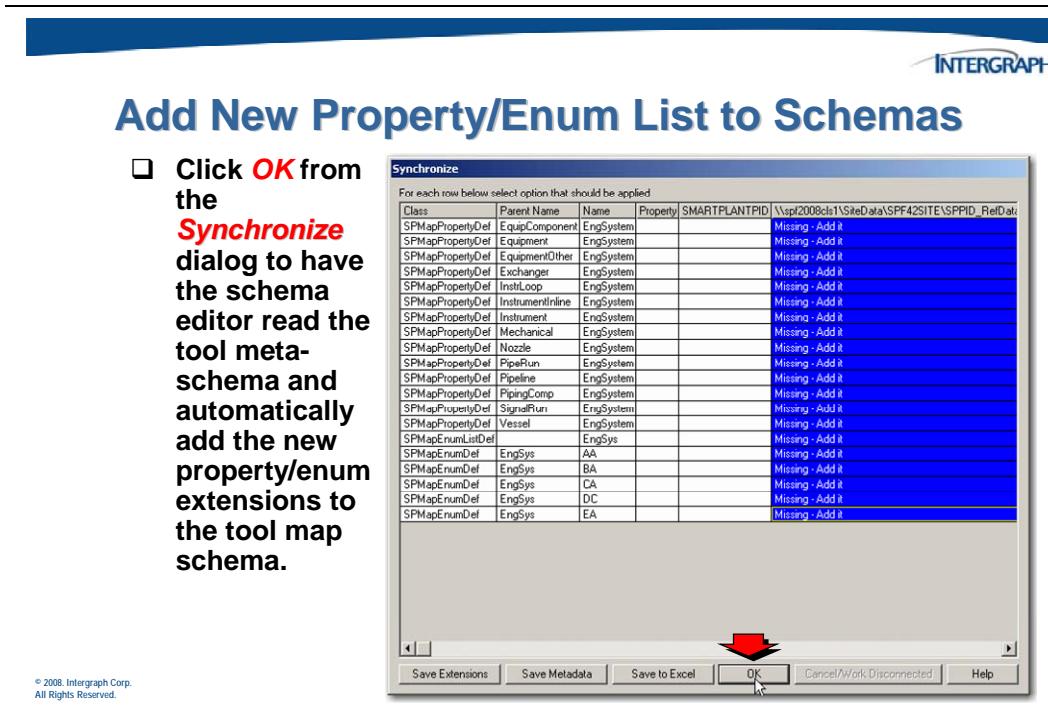
With the session file open, go to the **Application Metadata** tab and click the **SmartPlant** button. On the menu that appears, click the **Edit Engineering Tool Options** command.



Choose to open the SmartPlant P&ID tool map file and connect to the SPPID schema.

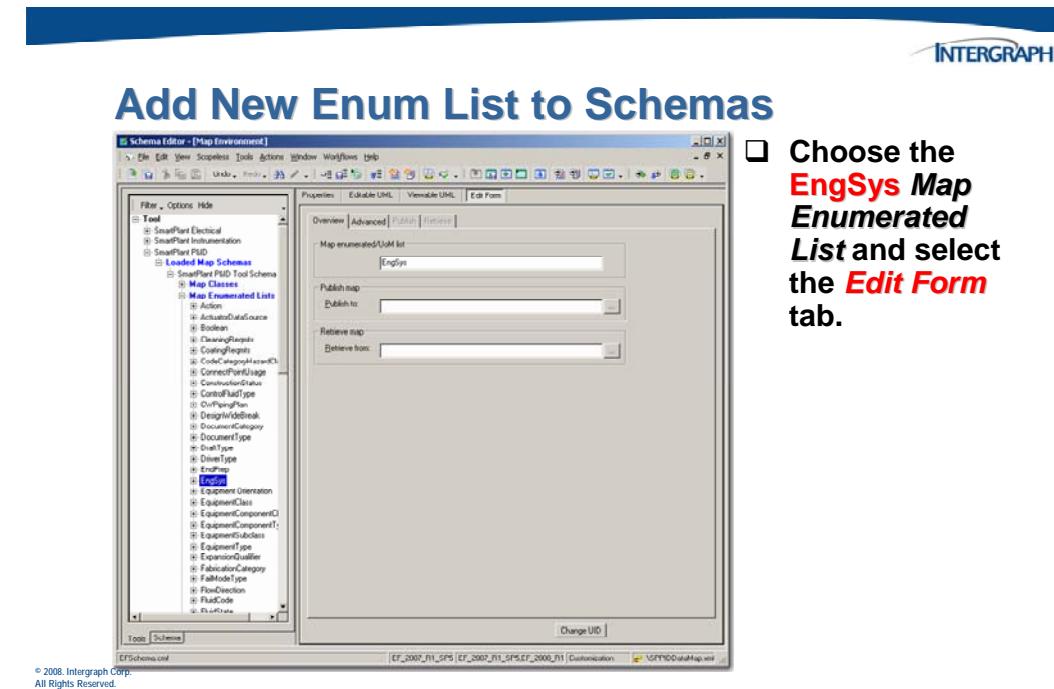


On the **Synchronize** dialog box, push the changes from the Data Dictionary Manager to the SPPID tool map file.



8.9 Extend the SmartPlant Schema

From the **Map Environment**, drill down under SmartPlant P&ID to the map enumerated lists, and find your new select list. You should map the new list before you try to map the new property, because to create the new property in the SmartPlant schema, you need an enum list type to scope the new property.



- Choose the **EngSys Map Enumerated List** and select the **Edit Form** tab.

From the **Edit Form** view, go to the **Advanced** tab.

Add New Enum List to Schemas

From the **Advanced** tab, click the **New SmartPlant Enumerated List with Correlated Entries** button.

Properties | Editable UML | Viewable UML | Edit Form | Overview | Advanced | Publish | Retrieve |

Map enumerated list definition
EngSys
ConstrainedMapEnumList
MapEnum

Overview Properties
UID: SP_12001
Tool UID: SPEN_12001
Short description: EngSys
Long description: EngSys
Number:
Select criteria:
Process criteria:
Contains: AA%BA%CA%DC%EA
Cognitrons:
Contained by:
Publish to:
Believe from:
Scoped map properties: EngSystems%2EngSystem%2EngSystems%2EngSyst...

EngSys
New Child
Import Clipboard Entries
New SmartPlant Enumerated List with Correlated Entries... (Red arrow)
New Sibling
Delete
Publish Map
Remove Map
Publish and Remove Maps...
Change UID

© 2008, Intergraph Corp.
All Rights Reserved.

Click the **New SmartPlant Enumerated List with Correlated Entries** button to create a new list in the SmartPlant Schema that matches the select list from SPPID.

Add New Enum List to Schemas

Verify the list and entries to be created and correlated, then click OK.

Created Enumerated List and Entries

Enumerations (in black) will be created and correlated to tool enumerations:

- EngSys
 - AA
 - BA
 - CA
 - DC
 - EA

OK Cancel (Red arrow)

© 2008, Intergraph Corp.
All Rights Reserved.

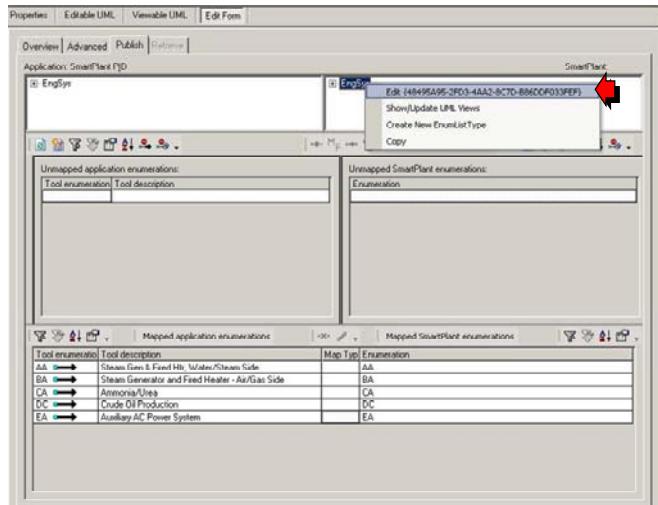
The new list has the same name and entries as the new select list from SPPID, and both the list and the values are automatically mapped by the software. However, if you intend to integrate this property with SmartPlant 3D, you will need to modify the number assigned to the enum enums.

From the **Publish** tab, which is now available for the map enumerated list, right-click the enum list type on the SmartPlant schema side. On the shortcut menu, choose the option to edit the list definition.

Note: If you are not going to publish this list to or from SmartPlant 3D, this step is not necessary.

Add New Enum List to Schemas

- From the **Publish** tab, right-click on the new enum list, **EngSys**, in the SmartPlant Schema area.
- Click **Edit** on the shortcut menu.

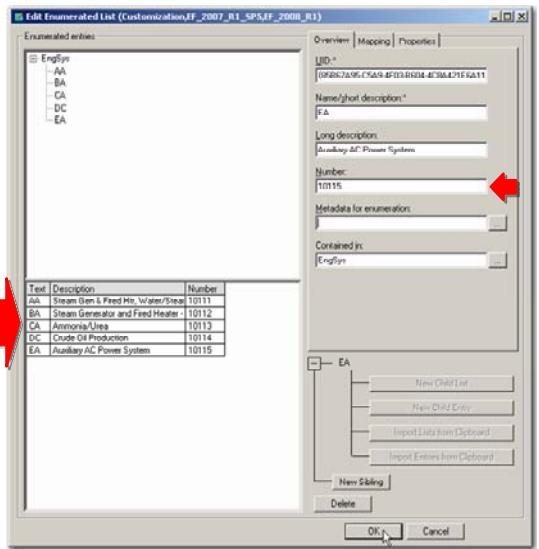


© 2008, Intergraph Corp.
All Rights Reserved.

For each of the enum enums, change the number to a value greater than 10,000.

Add New Enum List to Schemas

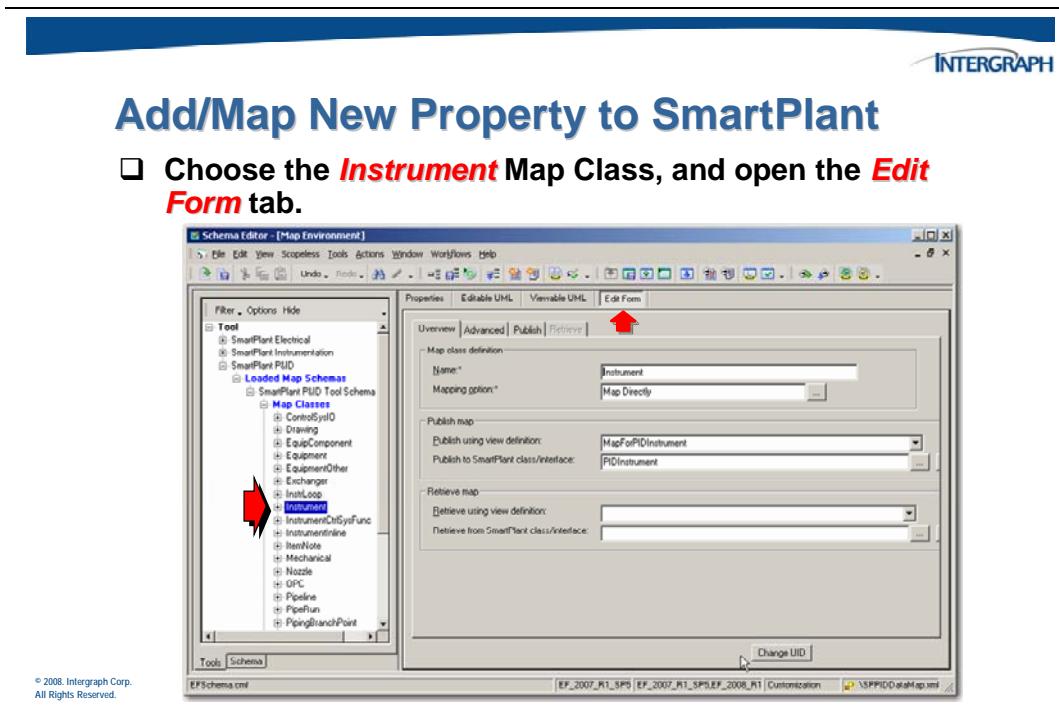
- Edit each new entry and set the *Number* value to match the value needed for loading into SmartPlant 3D later.**
- Click OK when the numbers have been changed.**



© 2008, Intergraph Corp.
All Rights Reserved.

8.10 Map the New Property

Next, choose the map class for which you want to publish the new property, and open the **Edit Form** view.



The new property can be found in the tool map schema, but not the SmartPlant schema.

Add/Map New Property to SmartPlant

- ❑ On the **Publish** tab, the **EngSystem** property is available in the tool map schema, but not the SmartPlant schema.

The screenshot shows the 'Add/Map New Property to SmartPlant' dialog. It has tabs for Overview, Advanced, Publish, and Retrieve. The Publish tab is selected. The left pane lists 'Unmapped application properties' including 'EngSystem'. The right pane lists 'Unmapped SmartPlant properties' which do not include 'EngSystem'. A red arrow points to the 'EngSystem' entry in the application properties list.

The **Auto-create** button creates a new property in the SmartPlant schema and maps it to the property selected in the tool map schema.

Add/Map New Property to SmartPlant

- ❑ This time, we will select the **Auto-create** button from middle/right side of the window, to let the software create the property for us.

The screenshot shows the 'Add/Map New Property to SmartPlant' dialog with the 'Auto-create' button highlighted by a red arrow in the toolbar. The interface is similar to the previous one, with tabs for Overview, Advanced, Publish, and Retrieve, and a list of unmapped application properties on the left and unmapped SmartPlant properties on the right.

The system creates a new property with the following default values:

Interface Definition – the primary interface of the class def to which the active map class is mapped. In our example, the default value for this field would be *IInstrumentOcc*, but we will change that value to the interface we created to expose our new properties, *ICustomInterface*.

Property Name – the name of the property from the tool map file with a prefix that is the active map class. In this case, the default value for this field is *Instrument_EngSystem*, but since this property will be used for other types of objects besides instruments, we will change the value to *EngineeringSys*. The name of the property in the tool map file and the name of the property in the SmartPlant schema do *not* have to match for a mapping relationship to be created.

Property Type – the name of the select list that was used for the original property from the tool. In this case, since we have an enumerated list type in the SmartPlant schema with the same name as the select list used for the property in the tool, the system provides that list as a default property type for our new property in the SmartPlant schema.

Add/Map New Property to SmartPlant

- In the **Create and Map Properties** dialog box, enter the information to create a new SmartPlant Schema **PropertyDef**.
- Be careful to choose the **ICustomInterface** interface def.
- Click **OK** when you are done.

The screenshot shows the 'Create and Map Properties' dialog box. The 'Object' tab is selected. In the 'Property Type' dropdown, 'EngSyst' is selected. Three red arrows point to the 'Interface Definition' dropdown, the 'Property Type' dropdown, and the 'Property Description' input field, highlighting these fields for modification.

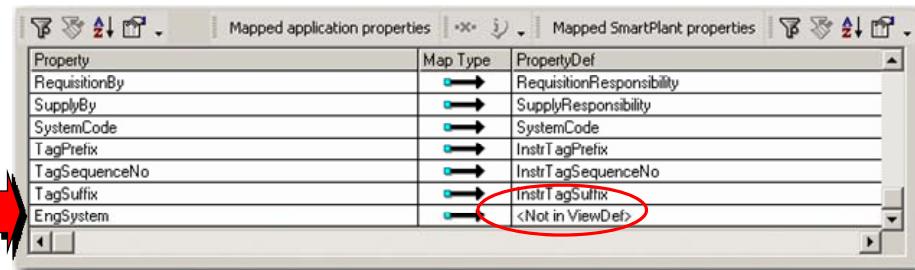
Once you have modified the applicable information and clicked **OK**, the software creates the new property and automatically maps it to the property in the tool map file.

Note:

- The name of the new property in the SmartPlant schema is not displayed in the Mapped Properties control initially. However, the name will appear during the next Schema Editor session. For the remainder of the current session, the name displays as <>Not in ViewDef>> for this map class, indicating that the property has not yet been added to the on-the-fly view def displaying available properties in the SmartPlant schema.

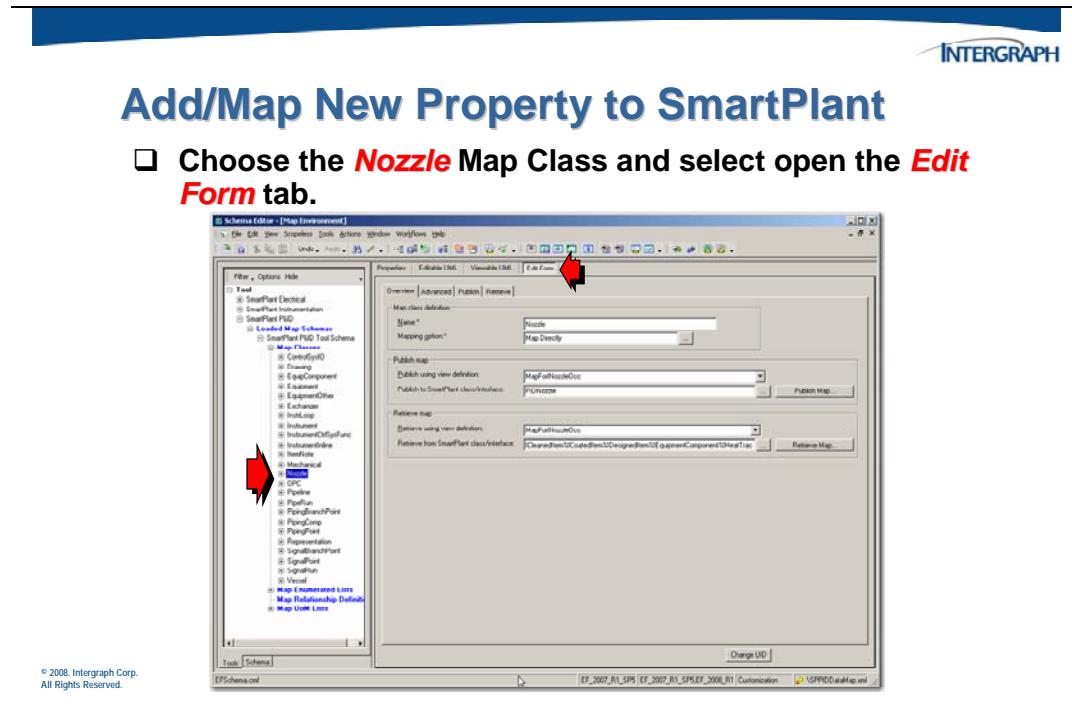
Add/Map New Property to SmartPlant

- Verify that the property from the two schemas is mapped at the bottom of the window.
- Note that the name you gave the property in the SmartPlant Schema does not appear initially.

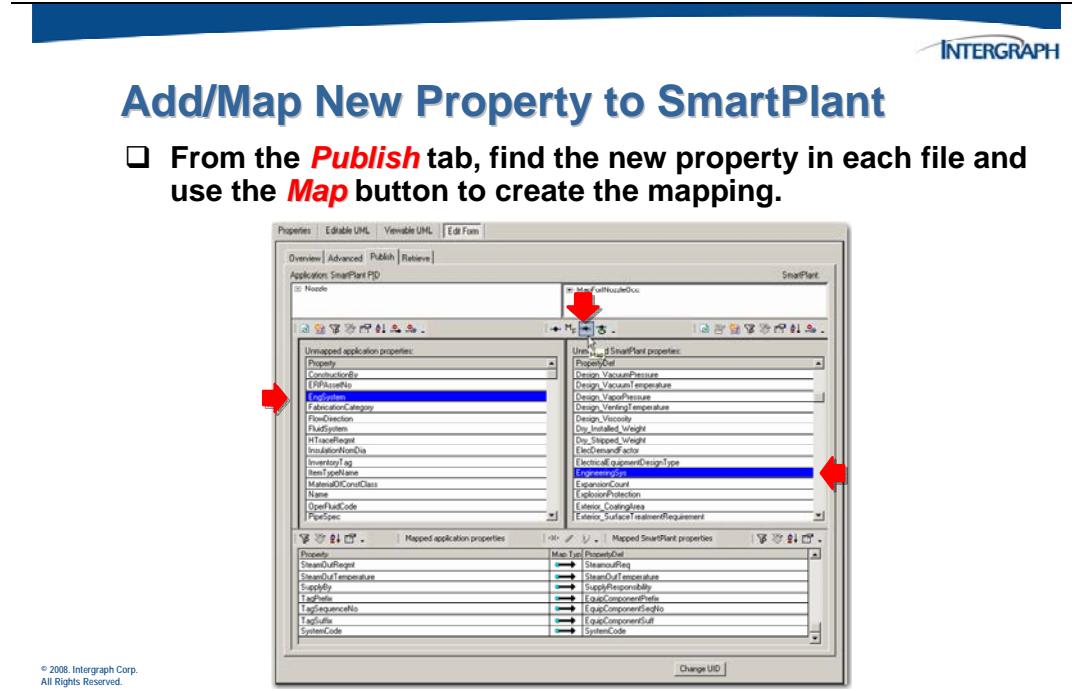


Property	Map Type	PropertyDef
RequisitionBy	→	RequisitionResponsibility
SupplyBy	→	SupplyResponsibility
SystemCode	→	SystemCode
TagPrefix	→	InstrTagPrefix
TagSequenceNo	→	InstrTagSequenceNo
TagSuffix	→	InstrTagSuffix
EngSystem	→	<Not in ViewDef>

Find the next map class for which you want to map the property, and open the **Publish** tab of the **Edit Form** view.



Find the new property in both the tool map file and the SmartPlant schema. Use the map button to create the mapping relationship.



Confirm the mapping relationship, and repeat the process for all applicable map classes.

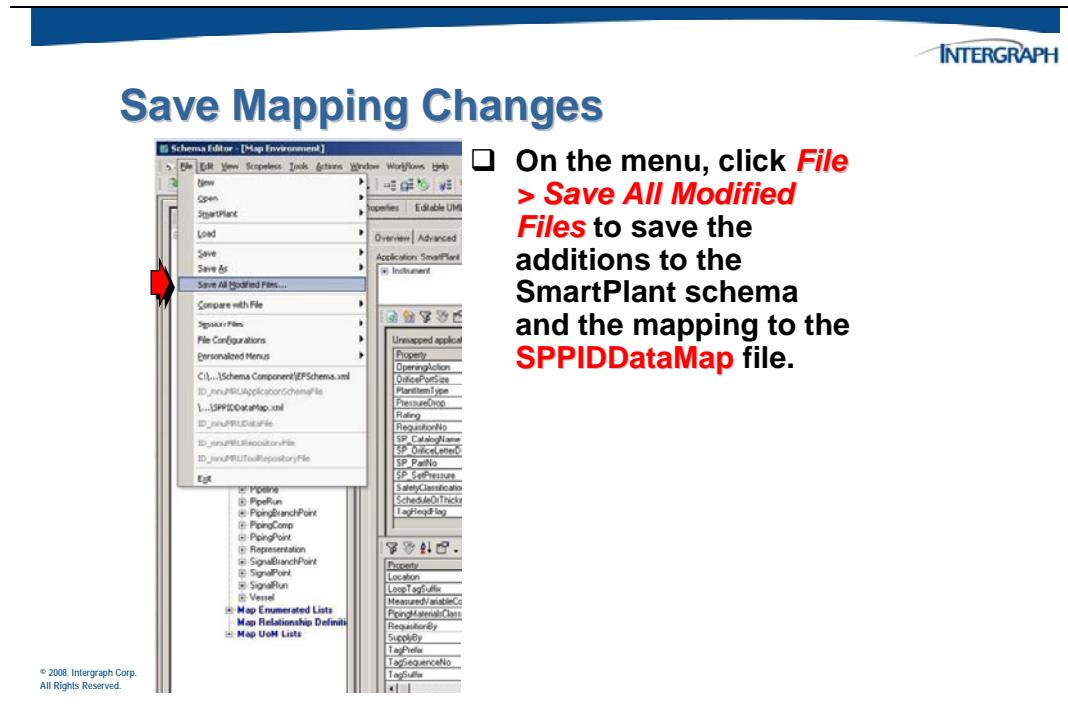
Add/Map New Property to SmartPlant

- Confirm the mapping in the bottom section the window.

Property	Map Typ	PropertyDef
SteamOutTemperature	→	SteamOutTemperature
SupplyBy	→	SupplyResponsibility
TagPrefix	→	EquipComponentPrefix
TagSequenceNo	→	EquipComponentSeqNo
TagSuffix	→	EquipComponentSuff
SystemCode	→	SystemCode
EngSystem	→	EngineeringSys

8.11 Save the Mapping Changes

Once you have mapped the appropriate tool map file properties, save your changes using the *File > Save All Modified Files* command.



When prompted, save the changes to the SmartPlant P&ID tool map file.

Save Mapping Changes

- Verify the tool map schema name for the mapping changes and choose Yes.



© 2008, Intergraph Corp.
All Rights Reserved.

Save the changes to the CMF file.

Save Mapping Changes

- Verify the CMF file for the property addition and choose Yes.

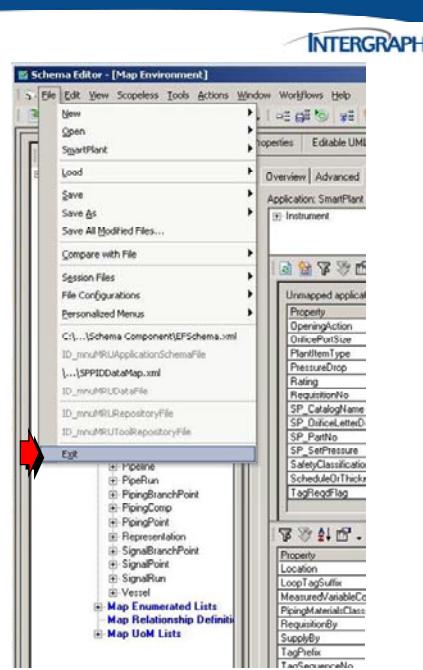


© 2008, Intergraph Corp.
All Rights Reserved.

Close the Schema Editor.

Save Mapping Changes

- On the menu, click **File > Exit** to close out of the schema editor.



© 2008, Intergraph Corp.
All Rights Reserved.

8.12 Activity 2 – Adding and Mapping a Complex Property

Complete the Chapter 8 – Activity 1 in the SmartPlant Foundation 2008 (4.2) Modeling and Mapping Activity workbook.

9

CHAPTER

Loading and Testing the Schema Changes

9. Extending the SmartPlant Database with Schema Changes

In the previous chapters, we have extended the SmartPlant schema to store information to be published from SmartPlant P&ID. However, until those changes are pushed to the SmartPlant Foundation database, the publish and load processes will not be able to actually save the property to the SmartPlant Foundation database.

The following chapter illustrates how we will export the schema changes out of the CMF file and import them into the database, using the *Schema Import Wizard*.



Schema Load Process

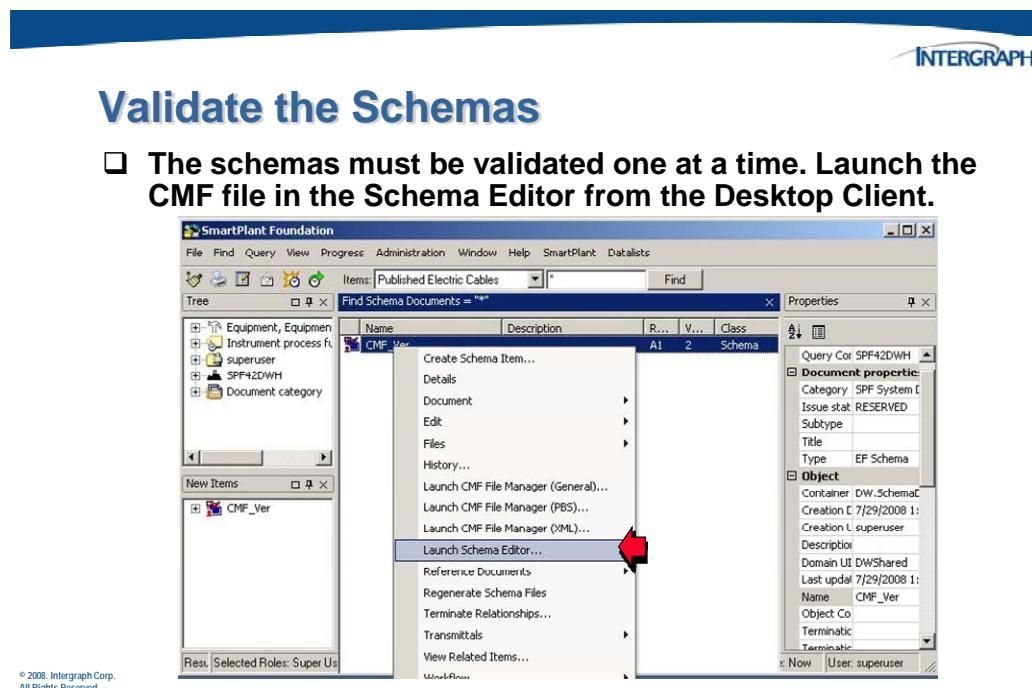
The **Schema Import Wizard**, available from within the Desktop Client, is used by system administrators to compare an XML file to the SmartPlant Foundation database.

Once the comparison is complete, the administrator can review the differences and choose which changes to make to the database.

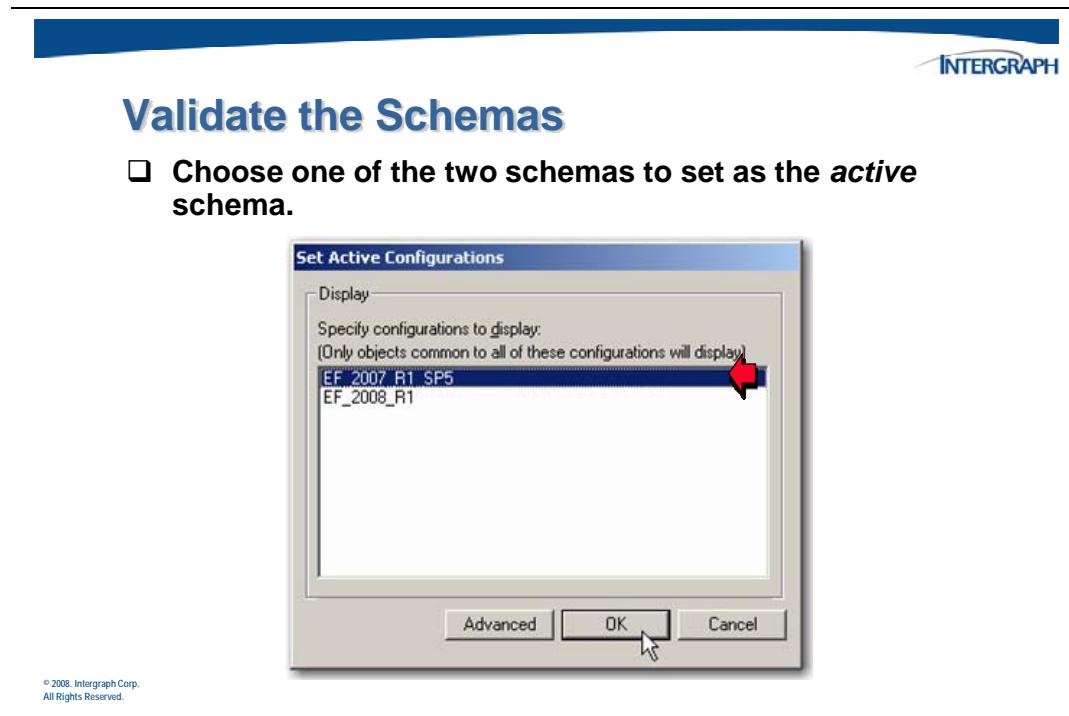
9.1 Validate the Schema Changes

Even if you are using the Auto Validate functionality that is now part of the Scheme Editor, you should still validate any changes that you have made to the schema before loading those changes into the database. The following example illustrates how to validate both schemas in the CMF file before continuing the load process.

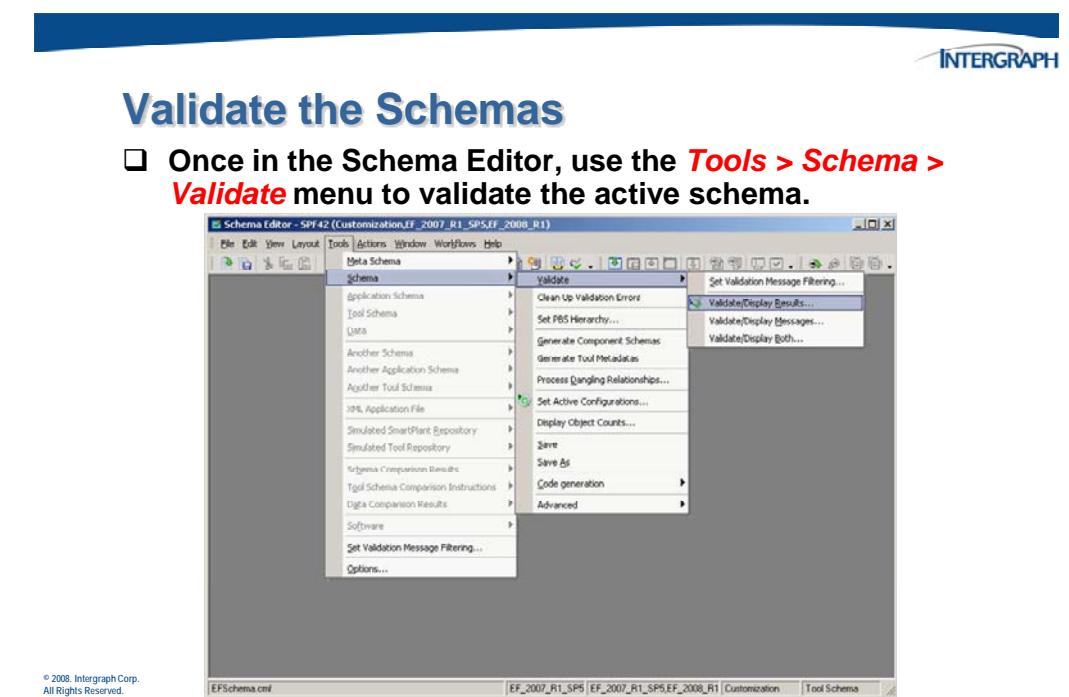
From the Desktop Client, find the CMF file, which is still checked out, and from it, launch the Schema Editor.



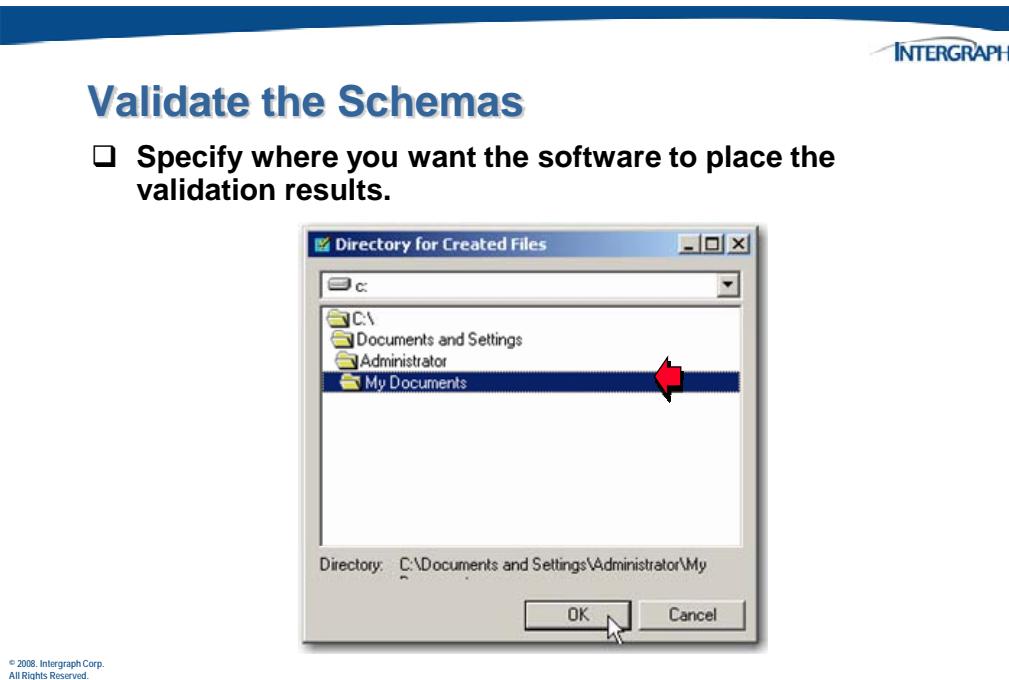
When prompted, select the schema that will be the active schema.



The Schema Editor will open. Use the **Tools > Schema > Validate > Validate/Display Results** command to validate just the active schema and check for errors that may not have been found during the process of extending the schema.



Specify where you want to write the results file.



The results will open automatically. Check the results for errors. Warnings should be checked but do not necessarily have to be resolved before loading the changes into the database.

Validate the Schemas

- View the validation results file for errors. Resolve errors before loading the schema changes.

```

EFS247.txt - Notepad
File Edit Format View Help
validation results for C:\Documents and settings\Administrator\My Documents\EFSchema.cmt (EF_A
=====
===== Warning: DeletePropShouldBeSet =====
if the minimum and maximum cardinalities for an end of a relationship definition
are the same, then all objects involved in relationships of that type (i.e., for
that relationship definition) should have the specific number of relationships (or
a validation message will be generated). That is, if an object at the end of the
relationship of this type where the cardinalities are the same is deleted, the
expectation would be that the related object would be deleted since failure to do so
would result in that object violating the cardinality rule.

if a relationship definition is defined for which the minimum and maximum cardinalities
are defined to be the same and that relationship definition is not defined to propagate
deletion from that end (for the reason specified above), then a DeletePropShouldBeSet
warning will be detected during validation.

Although this is not an error, failure to set the corresponding delete property on the
relationship definition can result in cardinality violations. Therefore, this warning
should normally be corrected by either adjusting the cardinalities or setting the delete
propagation property to be consistent.

"Min1 and Max1 have same value ('#1') for relationship definition '#2'. Therefore,
deletion of object at end 1 should be set to delete object at end 2."
#1 = Value for Min1 and Max1 values for relationship definition
#2 = Relationship definition

===== Instances:
#1 = '1'      #2 = 'PlannedMatlProcurement'
===== Warning: DuplicateEntryInList =====

```

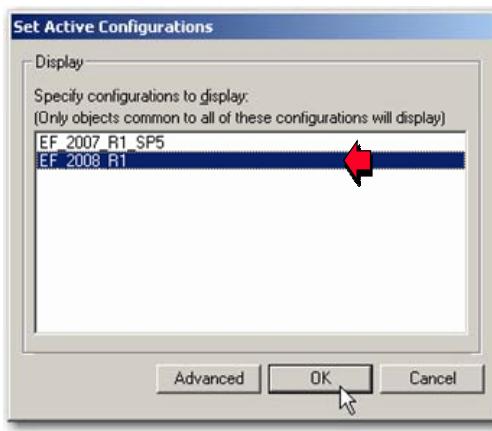
© 2008. Intergraph Corp.
All Rights Reserved.

When you have finished reviewing the results, close the Schema Editor. Then, from the Desktop Client, launch the CMF file and the Schema Editor again, but this time, choose the other schema. The validation process validates only the active schema, so if you have been writing your changes to both schemas, you should run the validation process twice, once for each schema, before loading the changes.



Validate the Schemas

- Close the Schema Editor and launch it again, selecting the other schema as the active schema.**



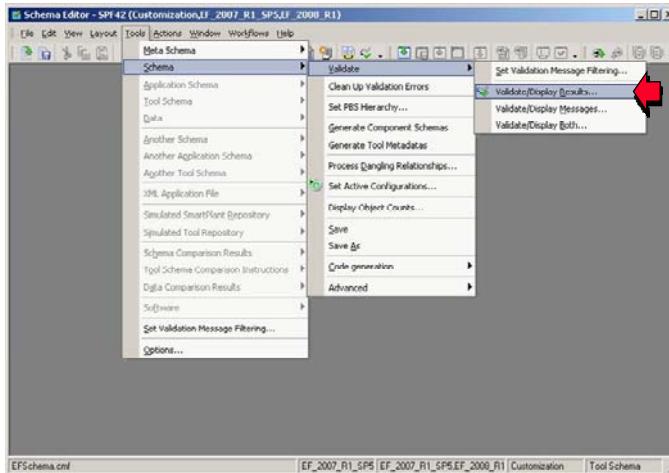
© 2008, Intergraph Corp.
All Rights Reserved.

Using the same process, run the validation process again.



Validate the Schemas

- Again, use the Validation feature to validate the active schema.

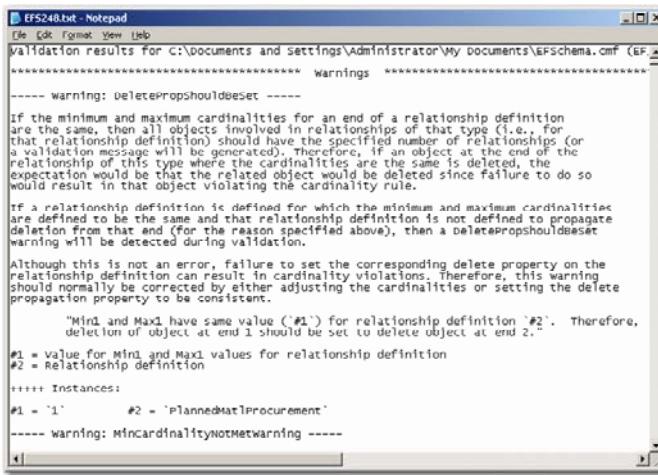


Again, review the results.



Validate the Schemas

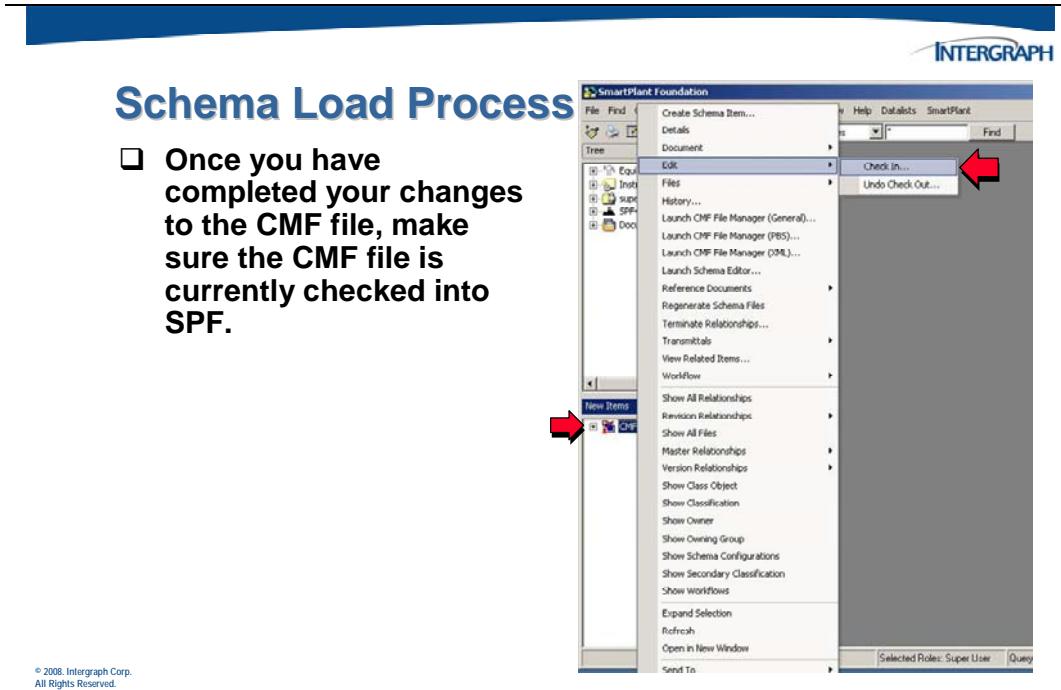
- Again, view the results to check for errors that should be resolved before the schema changes are loaded.



9.2 Extracting Schema Changes from the CMF File

Before the changes can be imported in the database, they must be exported out of the CMF file and into an xml that can be imported with the Schema Import Wizard. This process is managed from within the SmartPlant Desktop Client.

The first step is to make sure that the CMF file for the plant is checked into the SmartPlant Desktop Client.

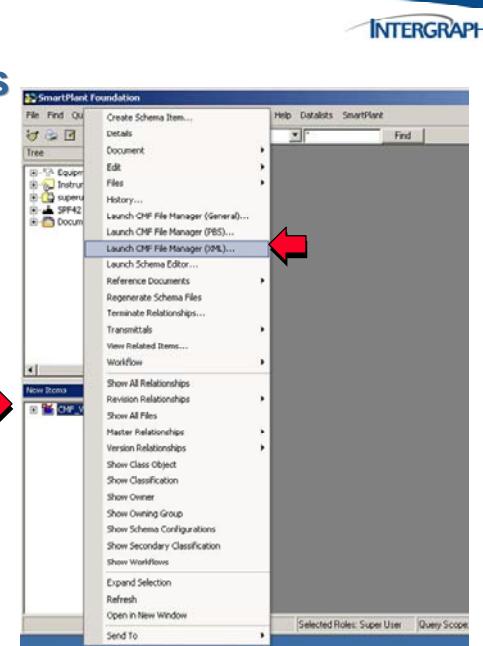


Once the CMF file is under the control of SmartPlant Foundation, use the CMF File Manager to create an XML file of the changes.

Schema Load Process

- Next, from the CMF document's shortcut menu, click **Launch CMF File Manager (XML)**.

© 2008. Intergraph Corp.
All Rights Reserved.

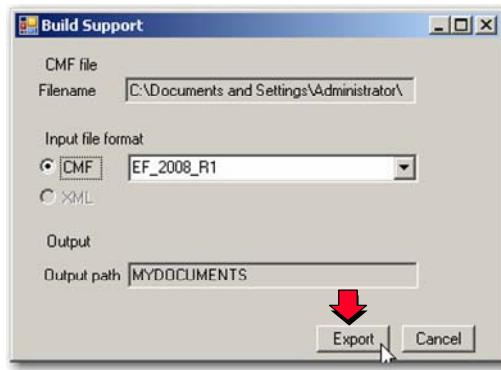


This command launches the ***Build Support*** window. Because this is SPF version 2008, we will choose that version of the CMF file. Note the location where the XML file will be placed. Click ***Export*** to begin creating the file.



Schema Load Process

- ❑ The ***Build Support*** dialog box that appears allows you to create a loadable XML file of your CMF file changes.
- ❑ Be sure to note the location of the file you are creating, and then click ***Export***.



© 2008, Intergraph Corp.
All Rights Reserved.

When the process is complete, a new file, *Schema_EFSchema-Gen.xml*, will be placed in the specified location. This XML file is a copy of the information in the 2008 schema in the CMF file.

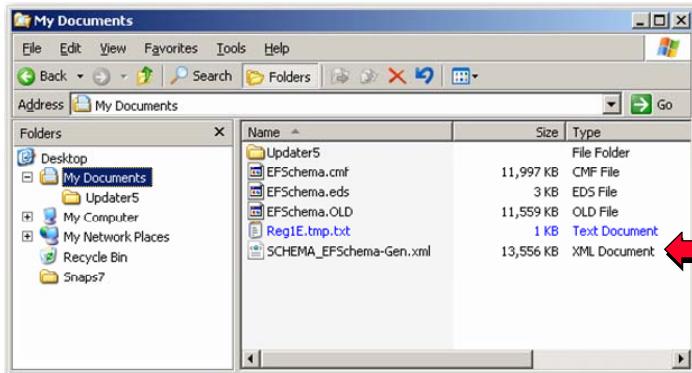
Note:

- Be sure to check in the CMF file before running this process. The **Launch CMF File Manager** command pulls a local copy of the stored CMF file, which is used to create the XML file. If you have another copy of the CMF file in the location default location, it will be replaced with the copy pulled from SPF.

Schema Load Process

- You will find the new file, *Schema_EFSchema-Gen.xml*, in the *My Documents* folder.

© 2008 Intergraph Corp.
All Rights Reserved.

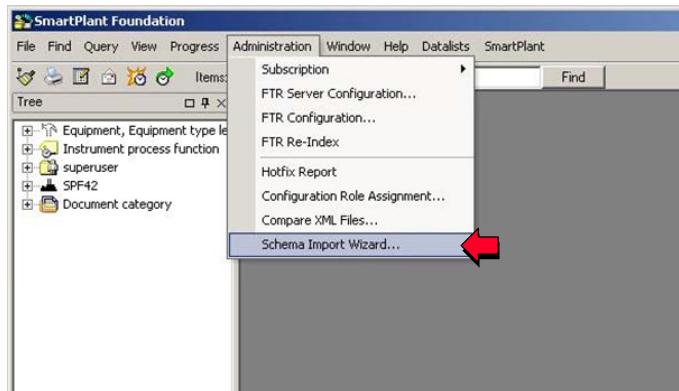


9.3 Load the Changes into the SPF Database

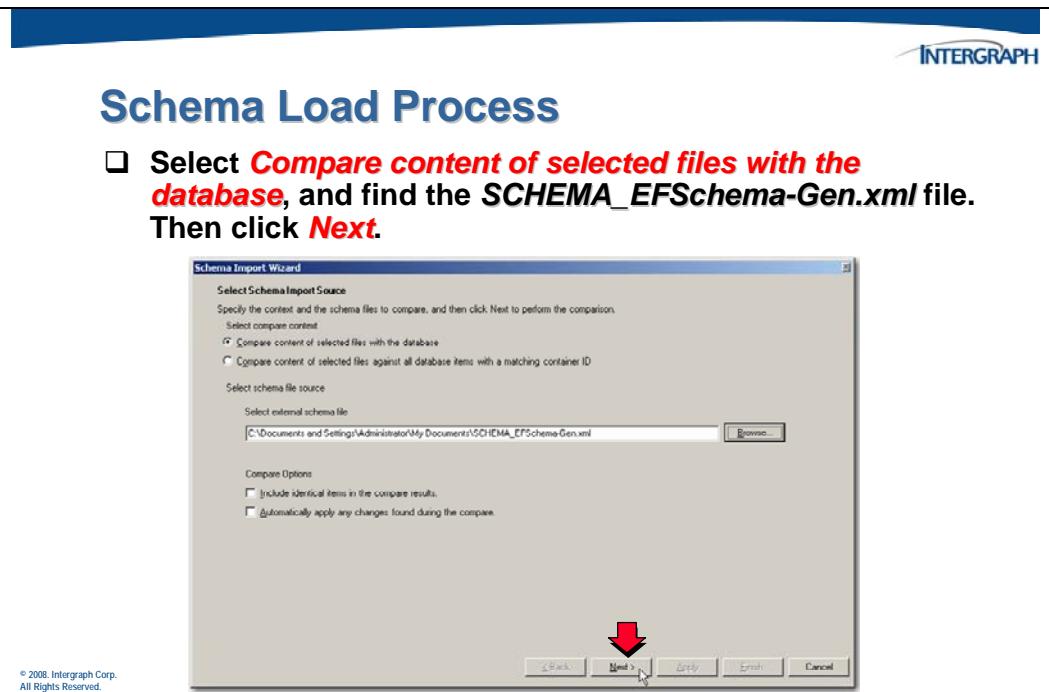
From within the Desktop Client, launch the **Administration > Schema Import Wizard** command.

Schema Load Process

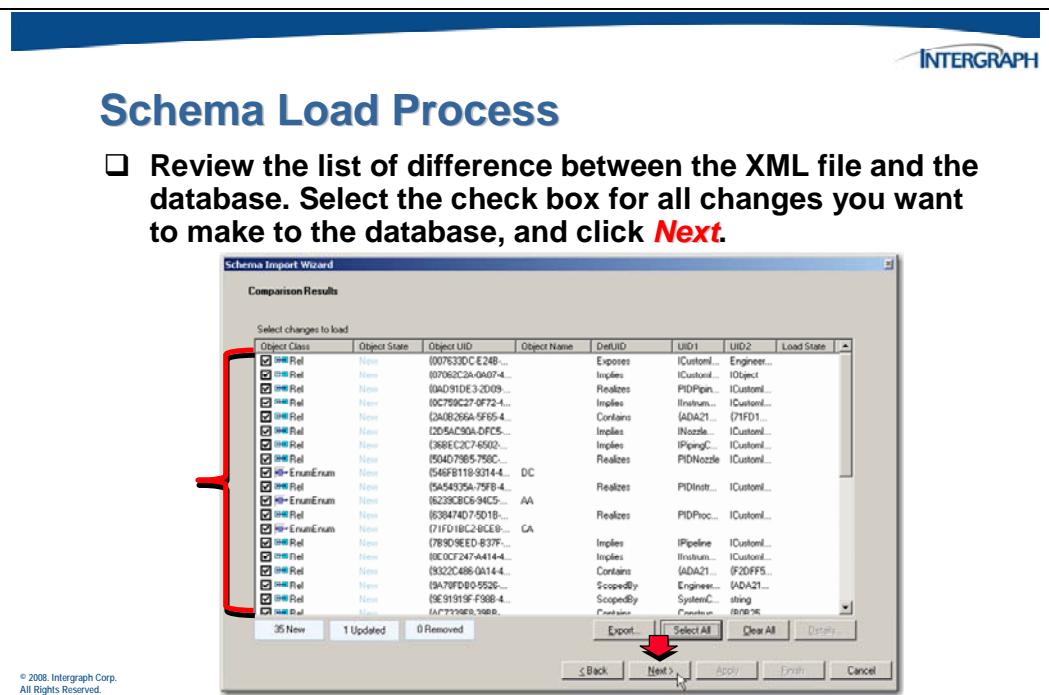
- In the Desktop Client, click **Administration > Schema Import Wizard**.



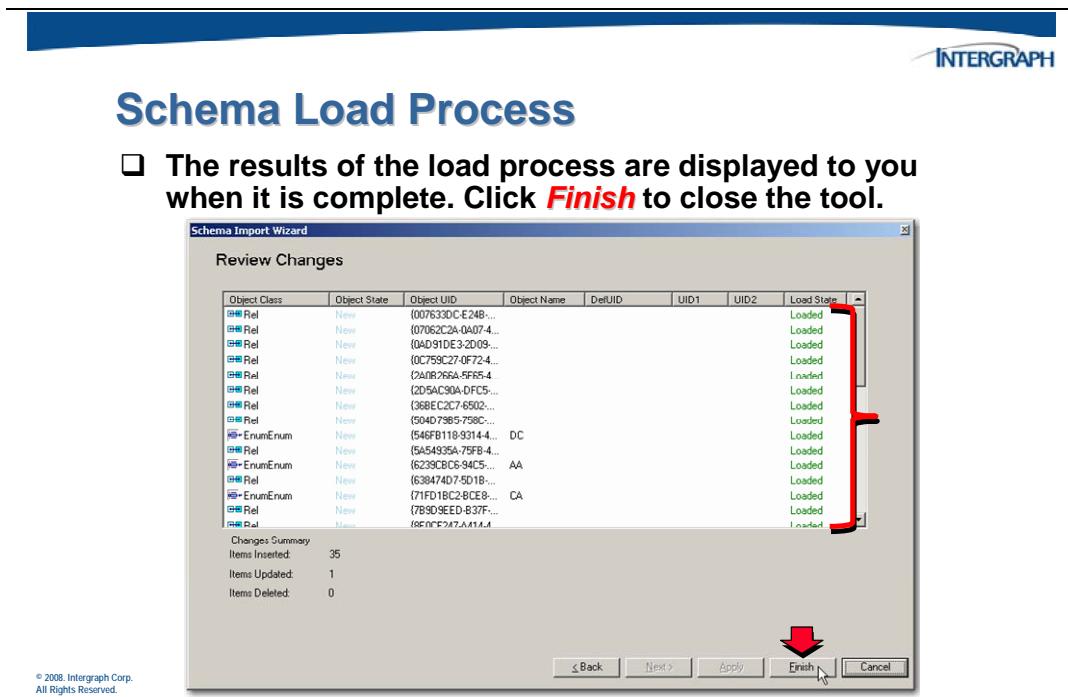
Specify that you want to compare the contents of the selected file against the database, and find the file you just created to use for the comparison.



The **Comparison** window shows the differences found between the XML file and the current SPF database. Select the changes you want to make, and click **Next**.



The **Review Changes** window shows the changes that have been loaded into the database.



© 2008, Intergraph Corp.
All Rights Reserved.

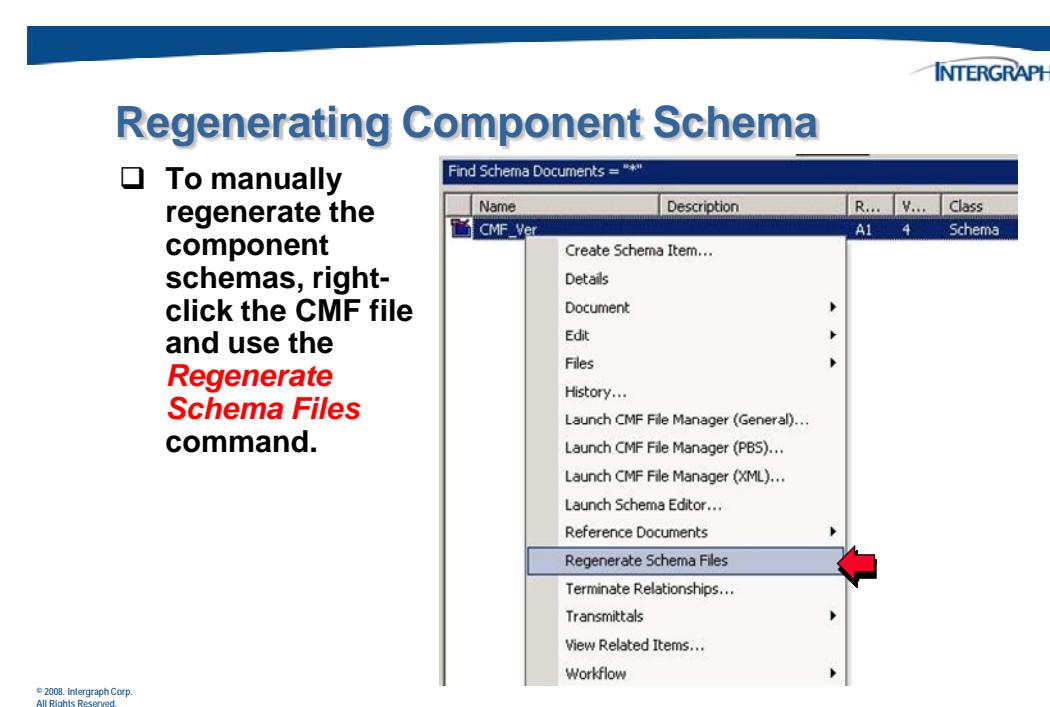
9.4 Regenerate the Component Schemas

During the process of loading the changes into the database, a couple of scheduler tasks are created to regenerate the component schemas. These tasks will regenerate all the components schemas for both the 2007 and 2008 schemas starting with the 2008 schema.

This process can take several minutes, and in some cases the component schemas may not be created before you are ready to publish a test document.

If you initiate the publish command from an authoring tool before the system has completed re-generating the component schema, the system will recognize that the existing components schemas are out-of-date in comparison with the CMF file, and the system will regenerated only the needed component schemas before the publish is performed.

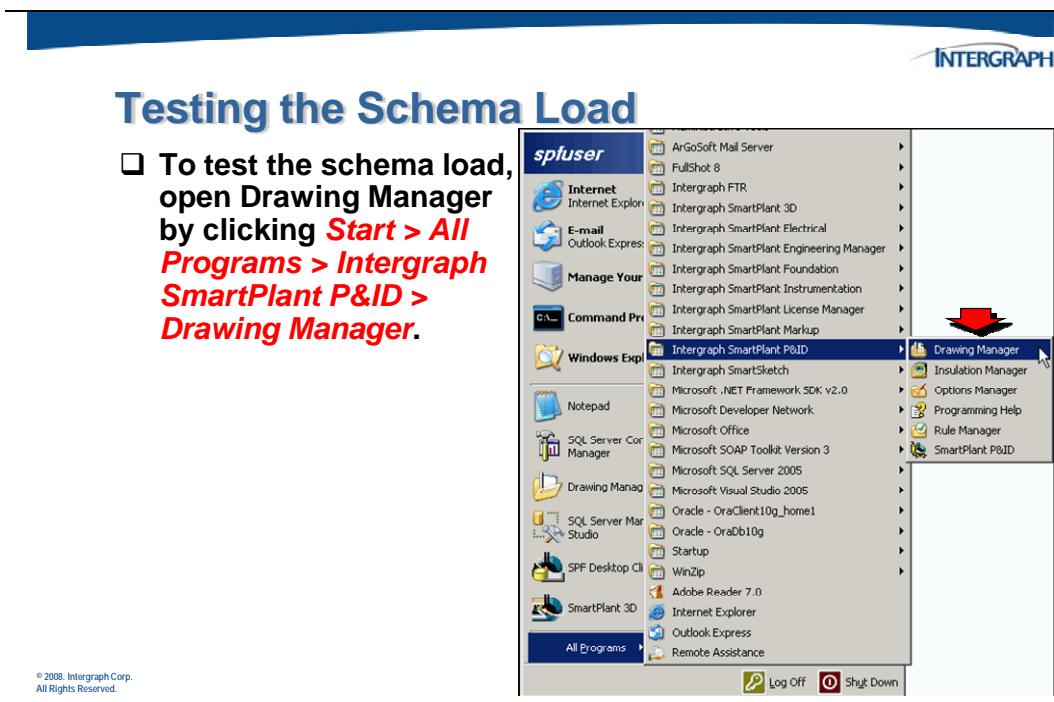
Should you wish to regenerate the component schemas yourself to ensure they are recreated before your test, you can perform the following steps.



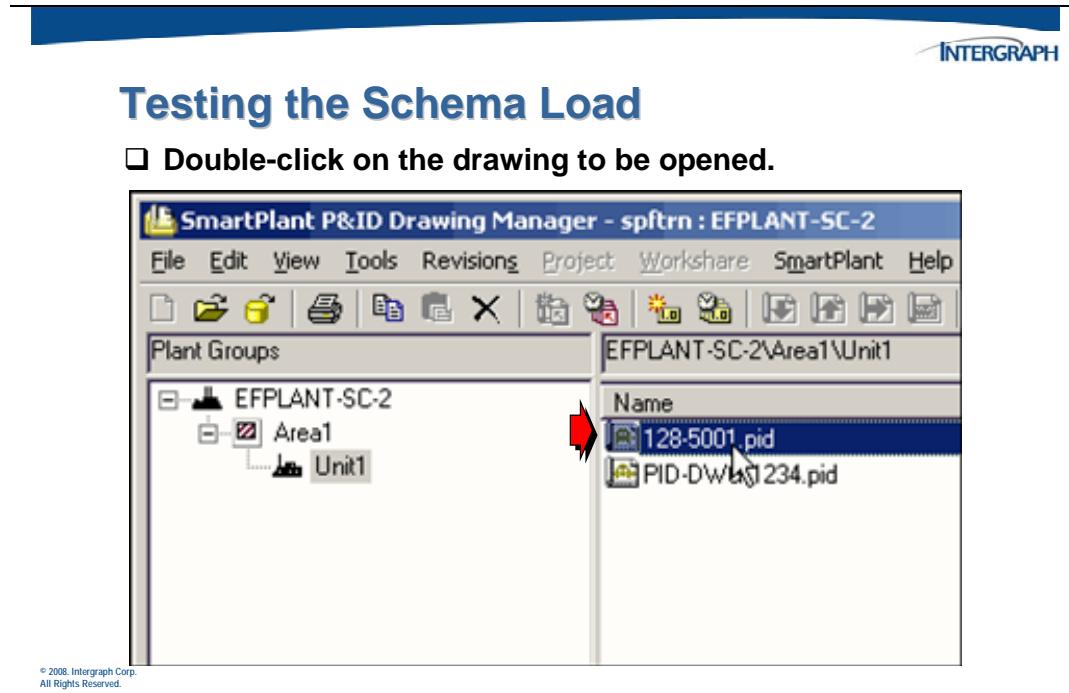
9.5 Testing the Schema Changes

To test whether the new mapping works, we will publish information in our new properties out of SmartPlant P&ID.

To access an existing P&ID to modify, launch the SmartPlant Drawing Manager.



The *SmartPlant P&ID Drawing Manager* window will display. The following illustration is an example of what the Drawing Manager looks like. The registered plant to be used for the publish operation is **SPF42**. Select the area and unit beneath that plant, and find the **128-5001.pid** drawing in the right-hand pane.



Testing the Schema Load

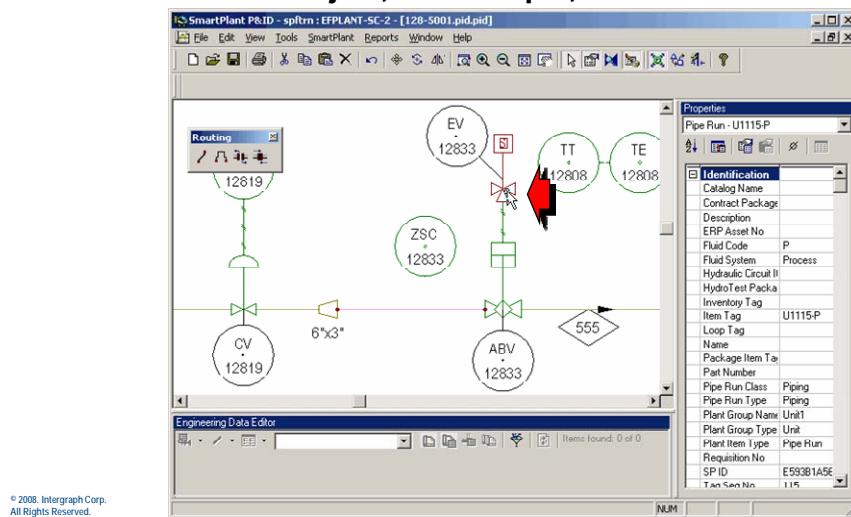
- Double-click on the drawing to be opened.

9.5.1 Modify the P&ID Drawing Data

Locate some of the offline instruments and nozzles that have been placed in this drawing.

Testing the Schema Load

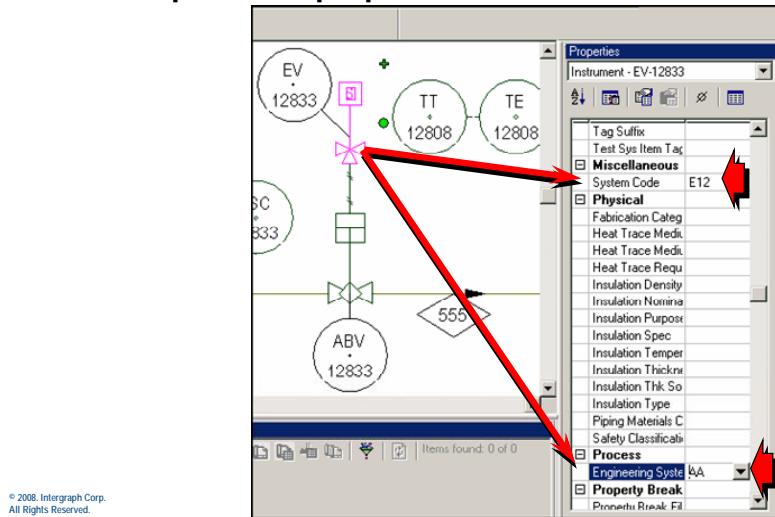
- Select an object, for example, an instrument.



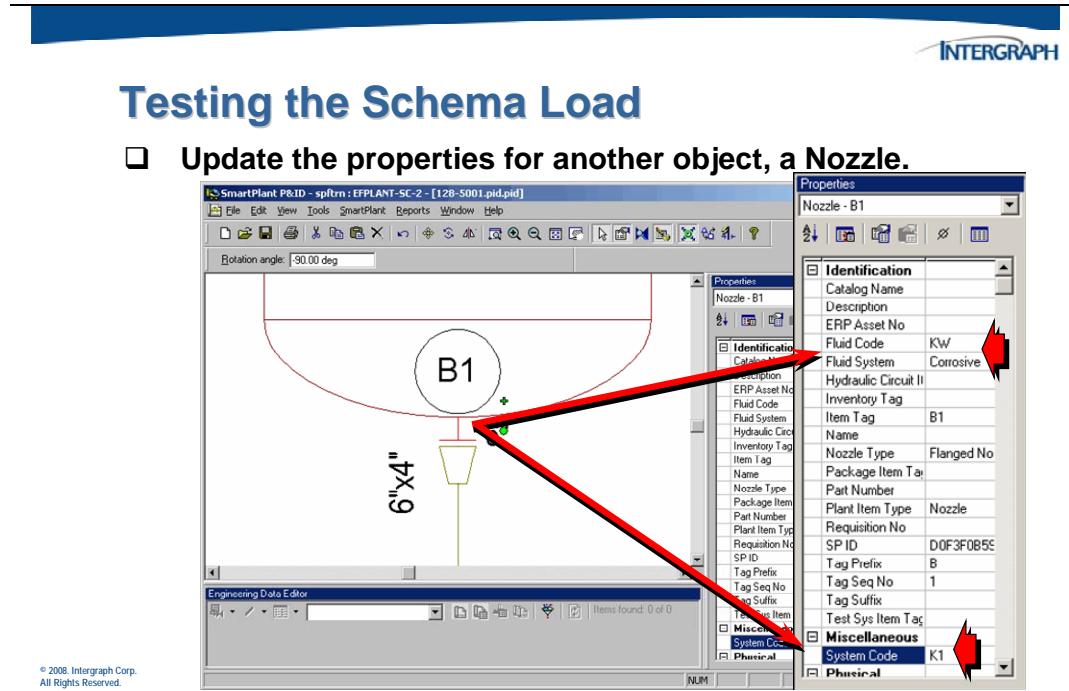
Provide values for the new properties.

Testing the Schema Load

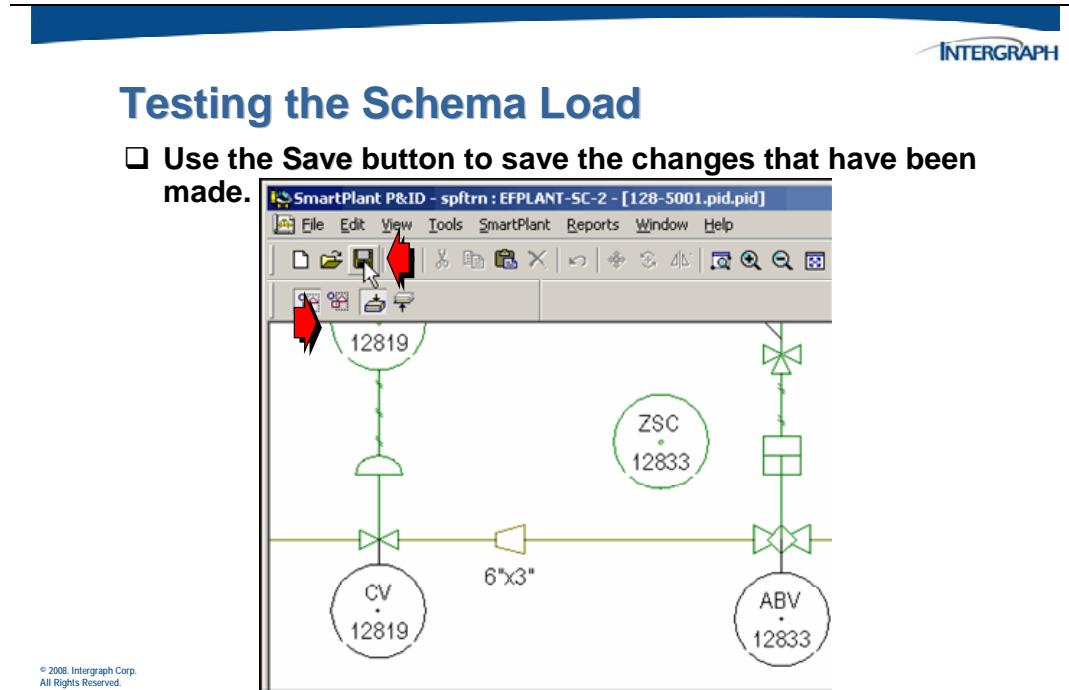
- Update the properties for the selected instrument.



Repeat the process for other objects in the drawing, such as nozzles.



Once you have modified the values for several objects, save your changes to the drawing.

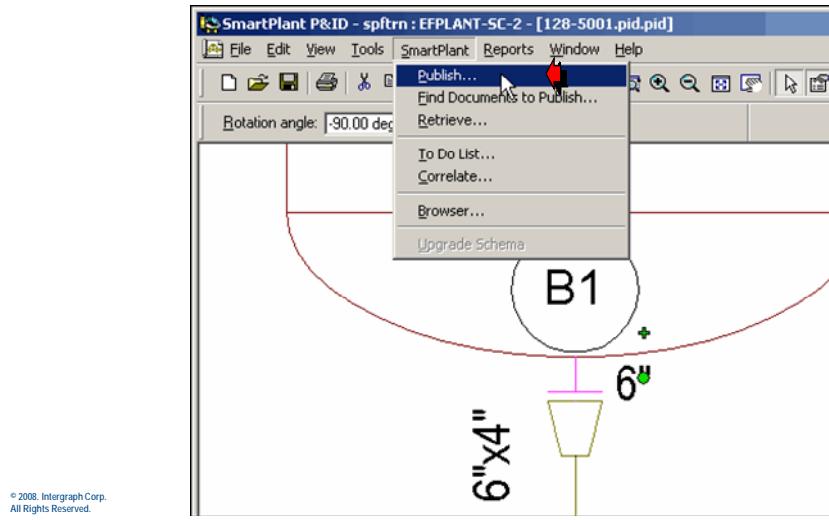


9.5.2 Publish the P&ID Drawing

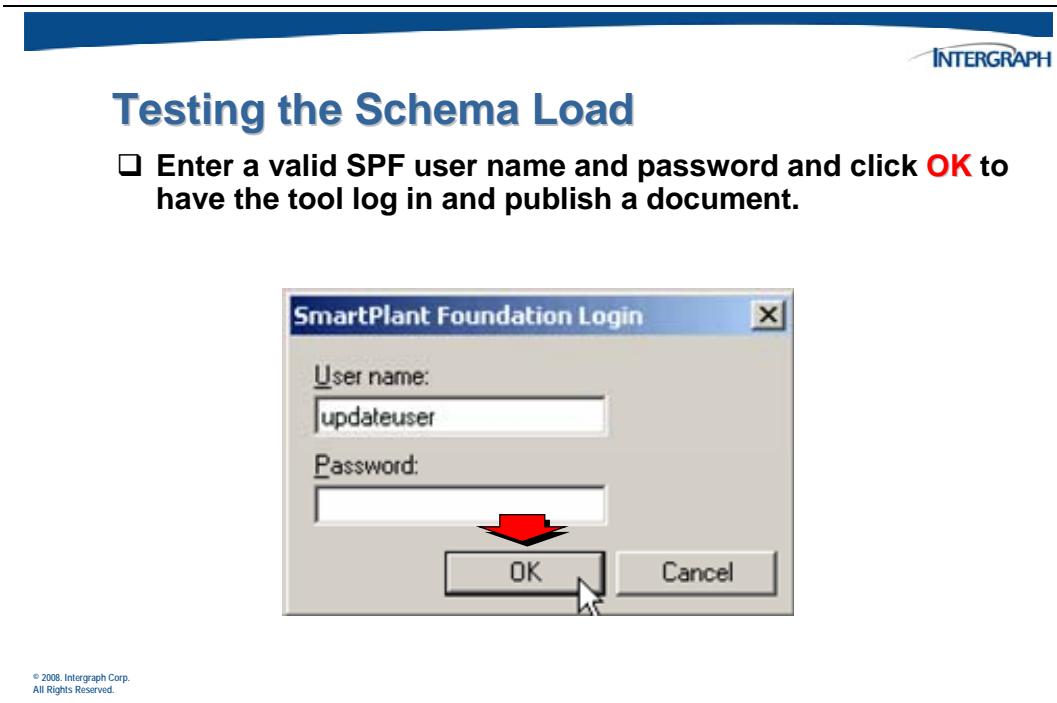
Use the *SmartPlant > Publish* command to instigate the publish process.

Testing the Schema Load

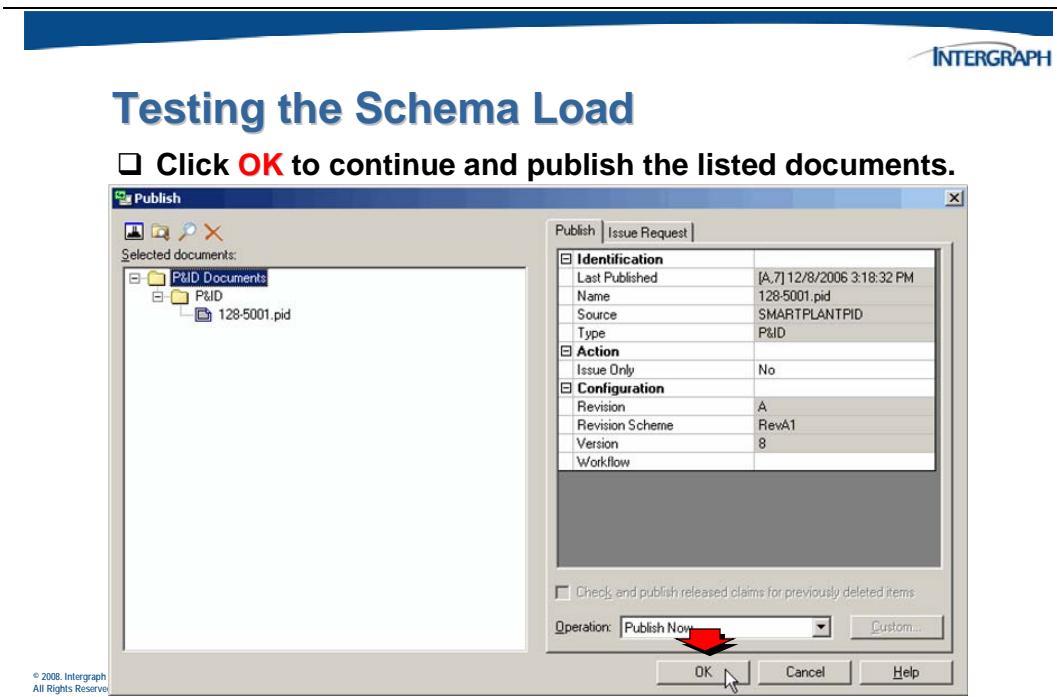
- Select the *SmartPlant > Publish* menu command.



In some cases, this command will require that you log into SmartPlant Foundation. In our environment, this is not necessary, as our system user has an SPF user account.



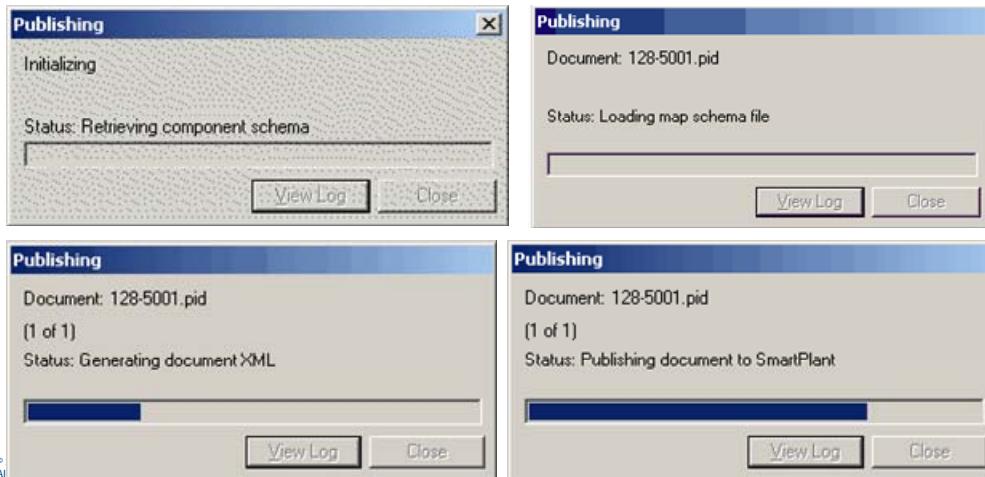
The standard **Publish** dialog command appears. Click **OK** to begin the publish process.



The following message will appear during the publish process.

Testing the Schema Load

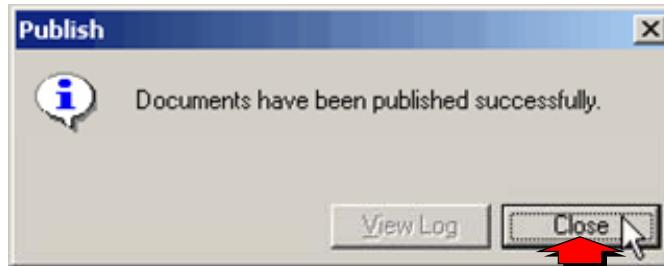
The **Publishing** dialog boxes will display the status of the publish operation.



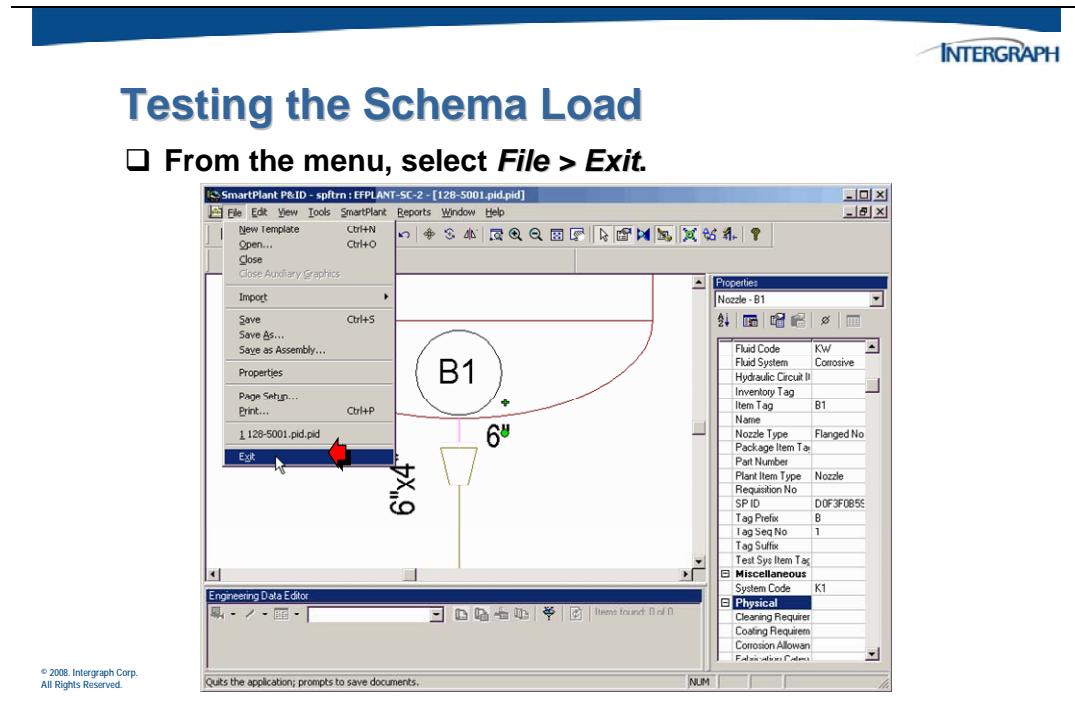
When the publish process is complete, the following message will appear.

Testing the Schema Load

- When the results dialog box displays, click **Close**.



Once the publish process is complete, close the P&ID and Drawing Manager applications.



9.5.3 Checking the Published File for New Properties

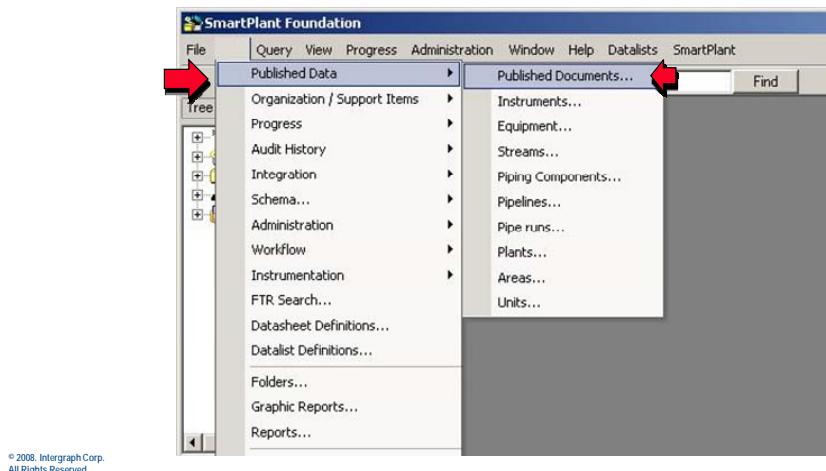
Because the published files that are placed in the vault have been encrypted, they must be decrypted before they can be viewed. The easiest way to do this is to extract a copy of the file out of SmartPlant Foundation for viewing is to use the Save Target As command from within the Desktop Client.

Start by finding the published file in the Desktop Client.

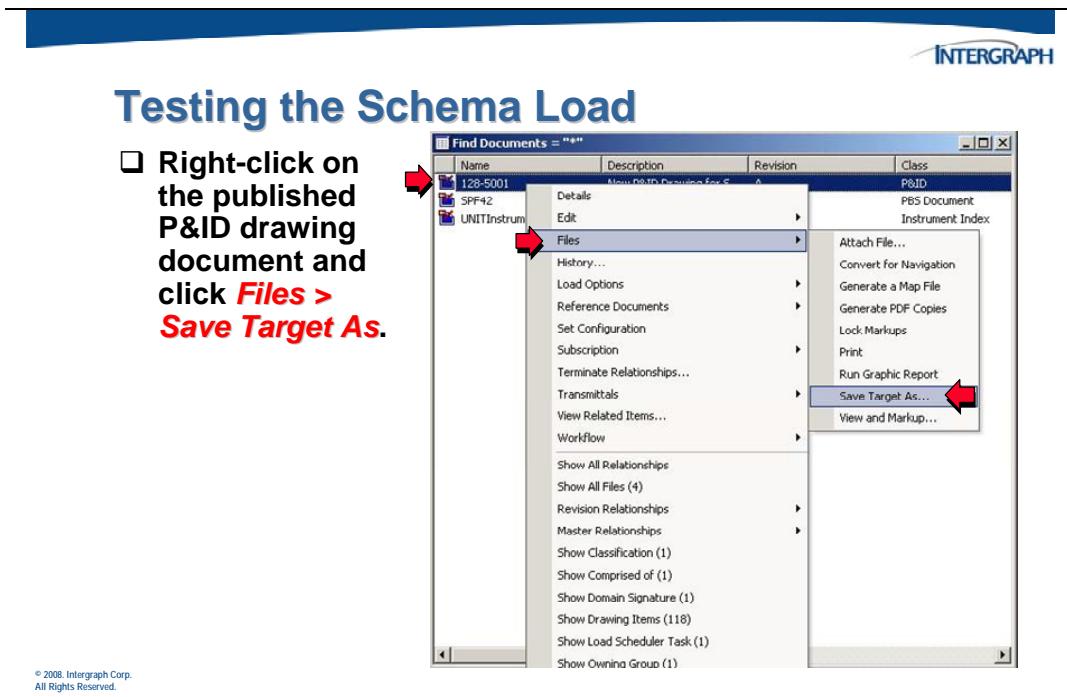


Testing the Schema Load

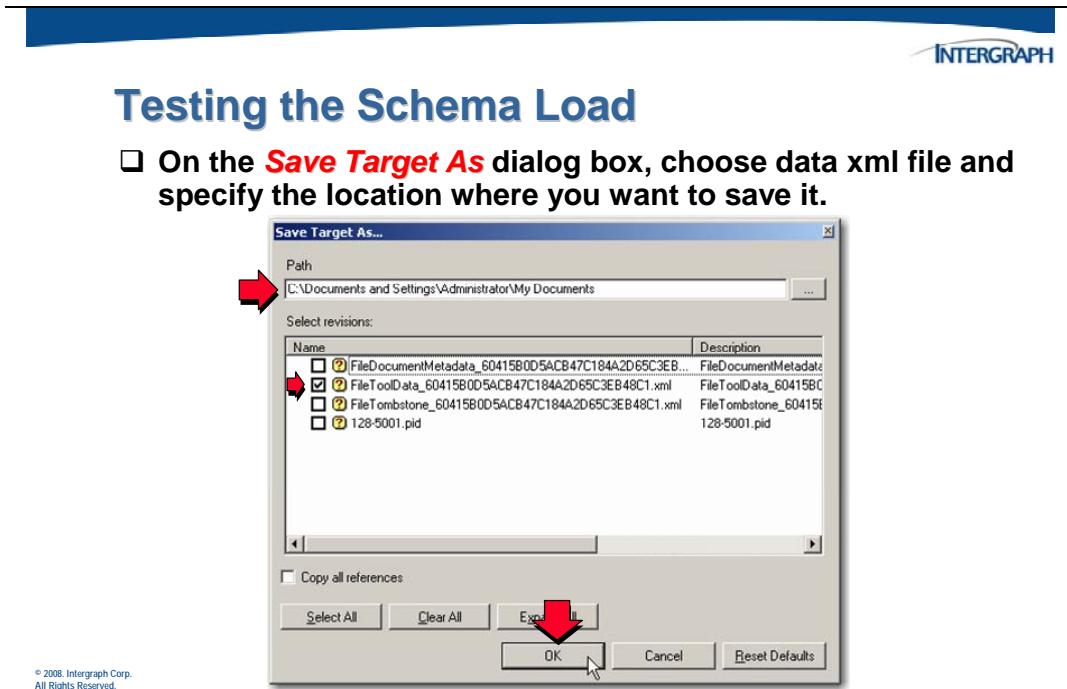
- From the Desktop Client, use the **Find > Published Data > Published Documents** command to find the published P&ID.



Once you have found the published P&ID drawing, right-click on it and use the **Files > Save Target As** command.



Choose which of the four attached files you want to copy out. You need only copy the FileToolData XML file. Be sure to note where the file will be placed.



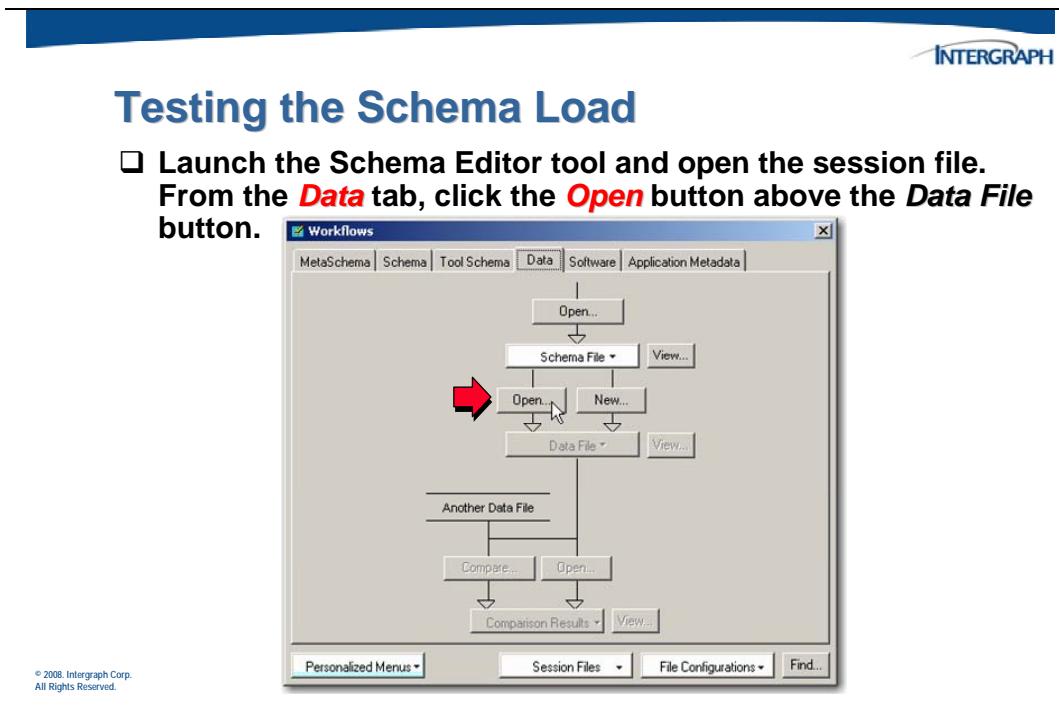
As the Schema Editor was created to view and manage XML files, it can also be used to view published XML files.

Launch the Schema Editor and open the session file.

Note:

- You will need to open the CMF file to compare the XML file again. So you will need to either copy out the CMF file and open it in the schema editor or check the CMF file out of the SPF Desktop Client and launch the CMF file from that application, as before.

From the *Overall Workflow* of the Schema Editor, go to the **Data** tab. Click the **Open** button above the **Data File** button.

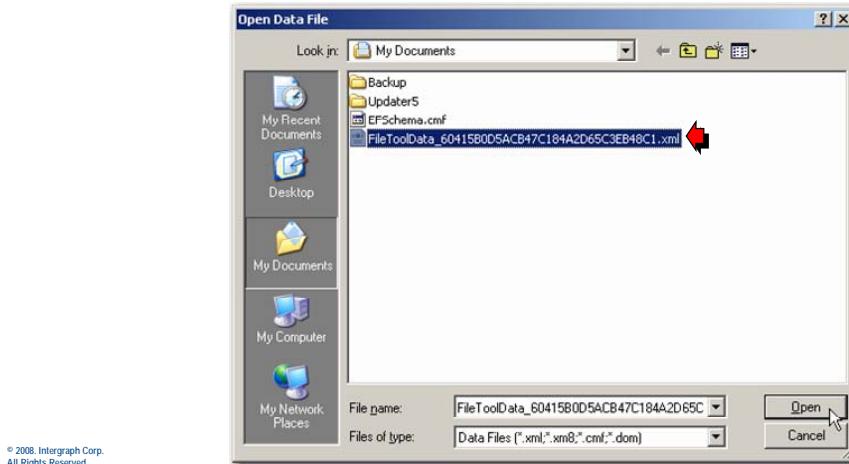


From the ***Open Data File*** dialog box, find the FileToolData XML file that you copied out of the Desktop Client and open it.



Testing the Schema Load

- Find the data xml file that you save out of the Desktop Client, and click ***Open***.

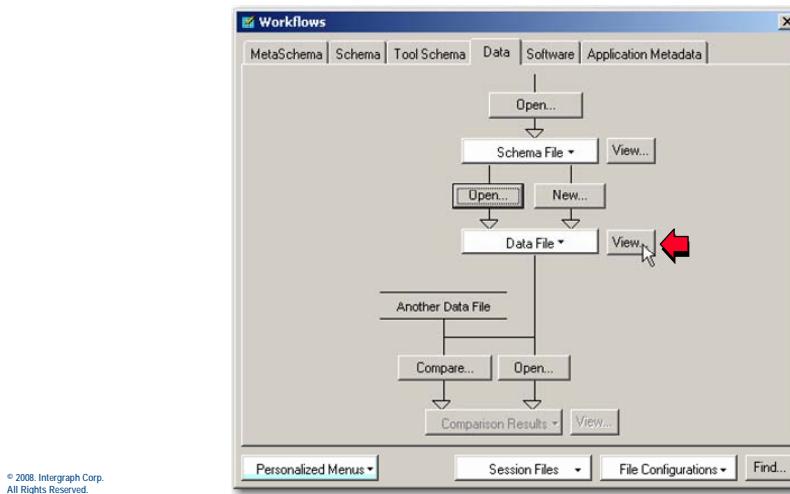


Once the data file is opened, click the ***View*** button beside the ***Data File*** button to view the published data.



Testing the Schema Load

- Click the ***View*** button beside the ***Data File*** button .



Select the viewer to use.

Testing the Schema Load

- Choose the view you want to use to access the data, and click **OK**.



The tree on the left displays a list of the class defs of the objects published in the drawing. Drill down beneath a class def to see the objects. The window on the right will provide views of the selected objects. Find the objects for which you provided values for the new fields and review them here.

Testing the Schema Load

Find the objects for which you changed and published data using the class definition list in the tree, and view the details in the Properties view.

The screenshot shows the INTERGRAPH Schema Editor interface. On the left, a tree view displays a list of class definitions under the 'PIDIInstrument' category. A red arrow points to the 'Properties' tab in the top navigation bar. Another red arrow points to the 'T1-12800' item in the tree view. To the right, a properties view window is open, also titled 'Properties'. It shows various tabs like 'Part', 'Power Consumers', 'Power Suppliers', 'Notes', 'Cables', etc. A large red oval highlights the 'Cables' section, which contains fields for 'Custom Interface', 'System Code', and 'EngSystem', with values 'Test' and 'AA' respectively. The bottom status bar shows 'EFSchema.mif' and several tabs including 'EF_2007_R1_SP5'.

9.6 Activity – Loading and Testing the Schema Changes

Complete the Chapter 9 – Activity 1 in the SmartPlant Foundation 2008 (4.2) Modeling and Mapping Activity workbook.

10

C H A P T E R

Mapping with SmartPlant Instrumentation

10. Mapping with SmartPlant Instrumentation

This chapter will demonstrate how to further extend the SmartPlant schema for SmartPlant Foundation along with the authoring tools (SmartPlant P&ID and SmartPlant Instrumentation) that integrate with it to create an integrated engineering system.



Extending the SmartPlant Schema

This example will add the “**preconstruction**” status to the existing **Construction Status** enumerated list.

The example includes **SmartPlant Foundation** and the following authoring tools: **SmartPlant P&ID** and **SmartPlant Instrumentation**

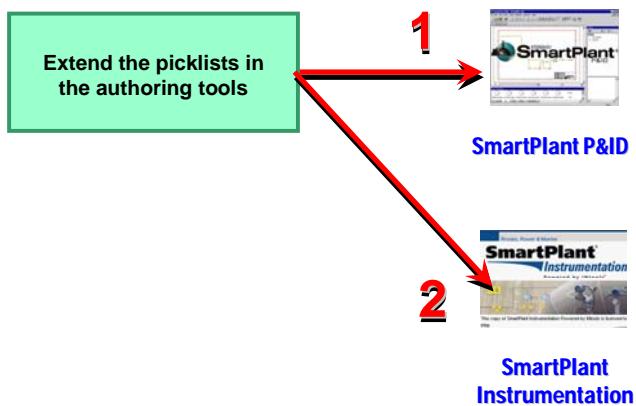


10.1 Modifying the Authoring Tools and Mapping the New List Values

There is no specific order to follow when extending the SmartPlant system. It may be easier and more logical to add the new picklist values to the authoring tool first. Once that has been accomplished, when extending the schema files, you will know exactly what values must be added there. It is critical that the value added to the tool schema match what has been added to the tool meta data.



Extend the Authoring Tools



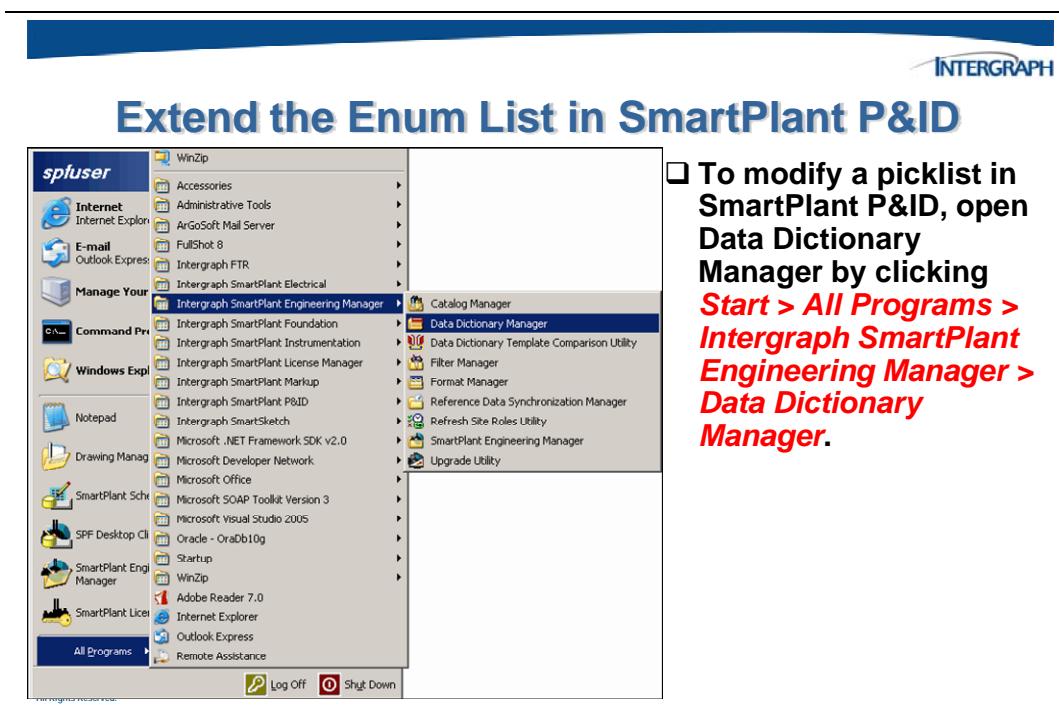
© 2005, Intergraph Corp.
All Rights Reserved.

In this example, two authoring tools are being extended so both will be modified in order to complete the extensions.

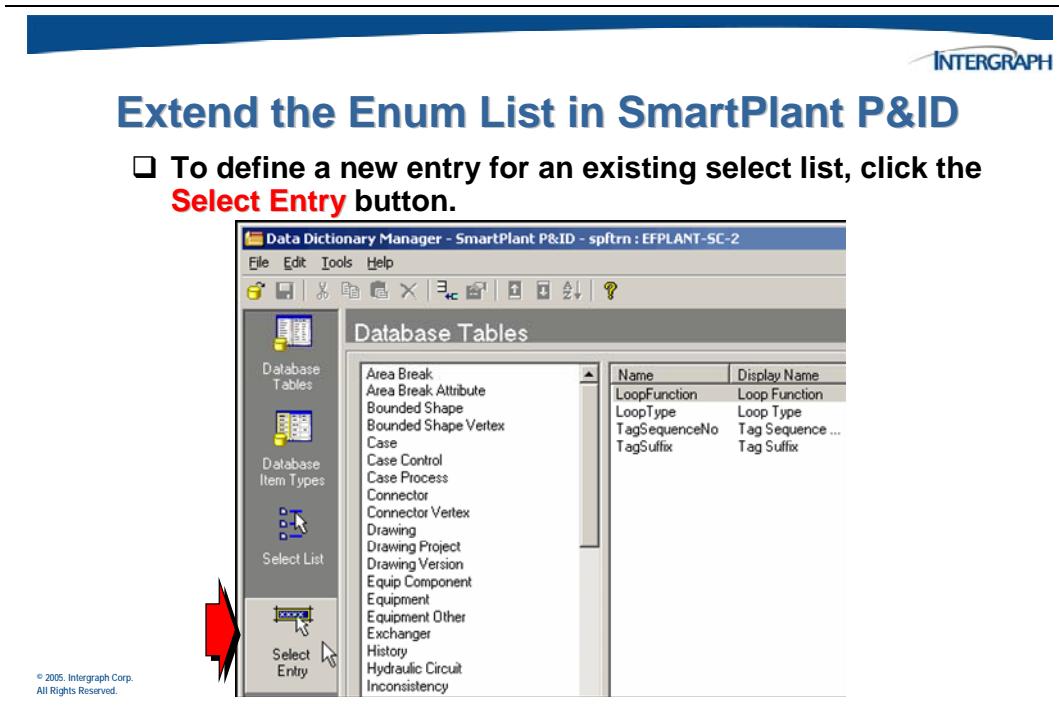
10.1.1 Extending an Enumerated List in SmartPlant P&ID

The first step to extending the SmartPlant system is to add the new information in one of the authoring tools. In the example in this chapter, the SmartPlant P&ID and SmartPlant Instrumentation authoring tools are demonstrated.

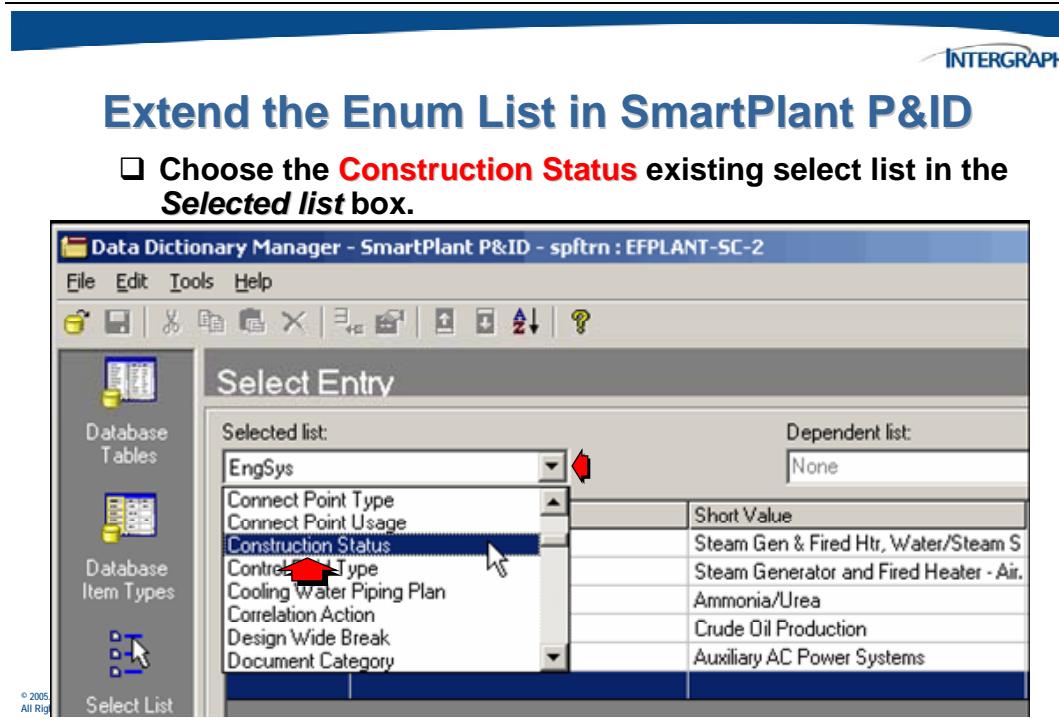
For SmartPlant P&ID, start the Data Dictionary Manager for the plant where you want to extend the construction status.



The *Data Dictionary Manager* application window will appear. Choose the **Select Entry** list.



Find **Construction Status** from the **Select List**.



Select the last entry in the list view and type “Preconstruction” as the **Value** and **Short Value**.

INTERGRAPH

Extend the Enum List in SmartPlant P&ID

Enter a new **Value** and **Short Value** for the new **Construction Status** entry.

Select Entry	<input type="checkbox"/>	Recalibrate	Recalibrate
	<input type="checkbox"/>	Existing, to be reused in place	Existing, to be reused in place
	<input type="checkbox"/>	Existing, to be modified	Existing, to be modified
	<input type="checkbox"/>	Existing, to be modified and relocated	Existing, to be modified and relocated
	<input type="checkbox"/>	Proposed	Proposed
	<input checked="" type="checkbox"/>	User Defined	
	<input type="checkbox"/>	Existing, not to be revamped	Existing, not to be revamped
	<input type="checkbox"/>	Preconstruction	Preconstruction

© 2005, Intergraph Corp.
All Rights Reserved.

Save the change to the select list.

INTERGRAPH

Extend the Enum List in SmartPlant P&ID

From the menu, select **File > Save**.

Data Dictionary Manager - SmartPlant P&ID - spftrn : EFPLANT-SC-2

File Edit Tools Help

Open Database... Ctrl+O

Save Ctrl+S

1 E:\SmartPlant\spftrn\smartplantv4.ini|EFPLANT-SC-2|SPPID

2 E:\SmartPlant\spftrn\smartplantv4.ini|EFPLANT-SC-2|SPEL

Exit

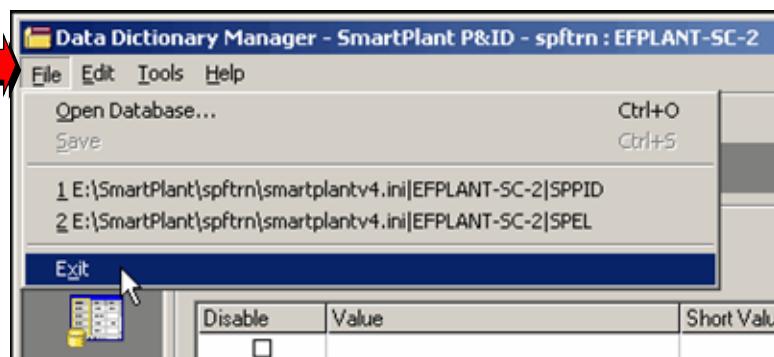
© 2005, Intergraph Corp.
All Rights Reserved.

Close the Data Dictionary Manager.



Extend the Enum List in SmartPlant P&ID

- ❑ Select **File > Exit** from the menu.

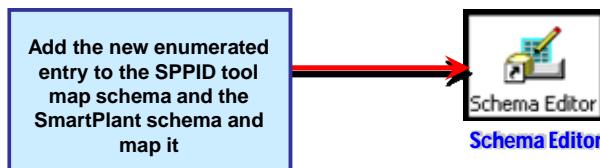


10.1.2 Extending the SPPID Tool Map Schema

Before you can define mapping for these new entries in the tool map schema, the change must be pushed to the tool map file. This is done automatically when the tool map schema and the tool schema are synchronized, using the metadata adapter. Once the change exists in the tool map file, the new value can be created in the SmartPlant schema, again using the Schema Editor to add new entries to the existing enumerated list. After the entry has been added, the schema must be saved and before publishing, the changes must be loaded into the database.



Extend the SmartPlant Schema

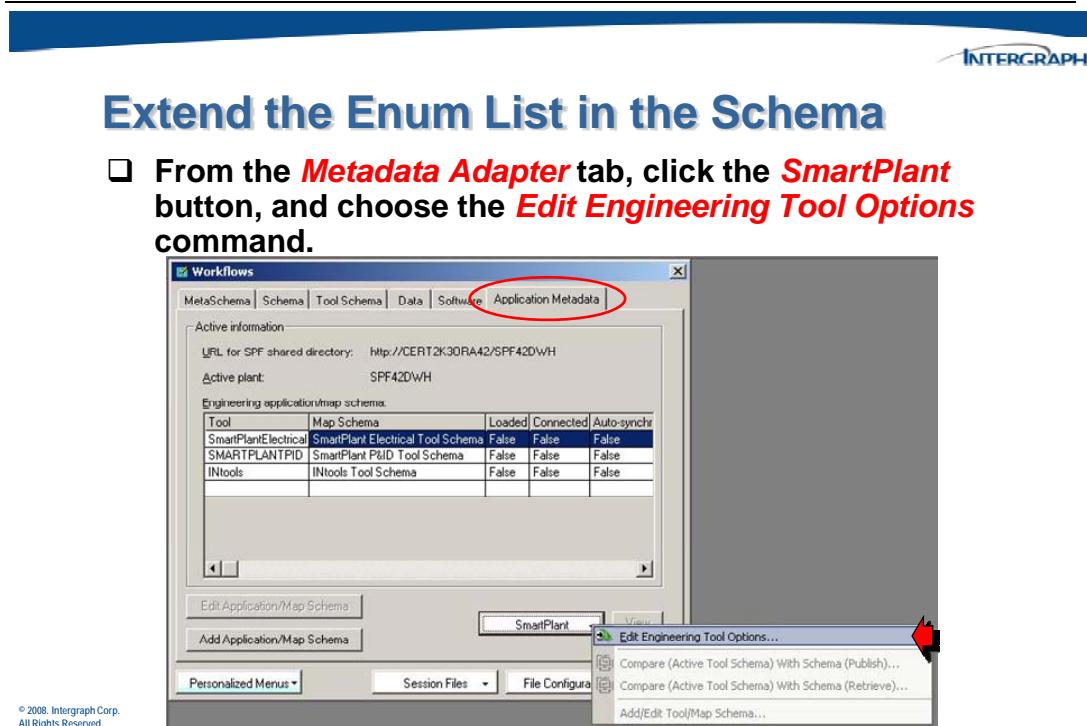


© 2008, Intergraph Corp.
All Rights Reserved.

These steps will all be accomplished in the Schema Editor. If you currently have the CMF file checked out, launch the Schema Editor and open the existing session file.

If you do not currently have the CMF file checked out, launch the SmartPlant Desktop Client, and check out the CMF file. Then launch the Schema Editor from the shortcut menu of the checked out CMF file.

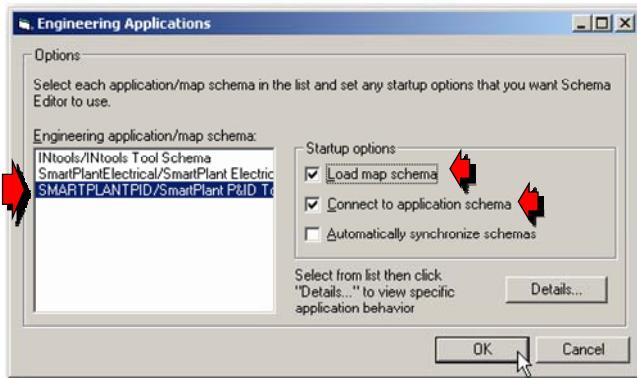
Once in the Schema Editor and viewing the applicable session file, launch the **Overall Workflows** and go to the **Application Metadata** tab. Click the **SmartPlant** button and choose the **Edit Engineering Tool Options** command.



Choose the SmartPlant P&ID map file and specify that you want to open the map file and connect to the tool application schema. Click **OK** to begin the synchronization process.

Extend the Enum List in the Schema

- ❑ Choose the metadata adapter for **SmartPlant P&ID**, and indicate that you want to launch the tool map schema and connect to the tool meta schema.

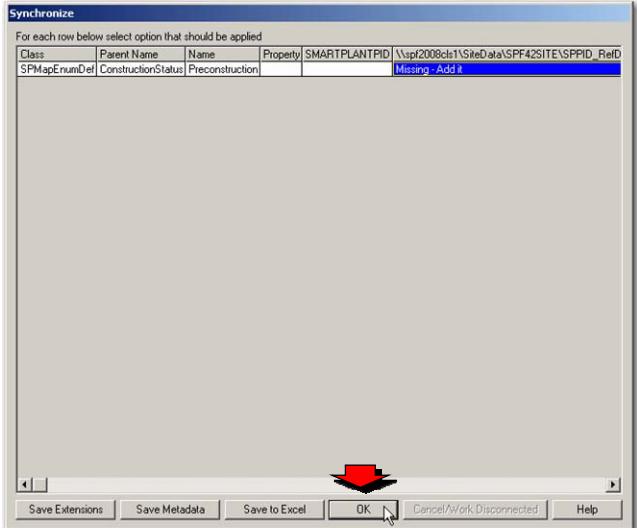


© 2008, Intergraph Corp.
All Rights Reserved.

The differences between the two schemas are displayed in the **Synchronize** form. For each difference the user must select (each one has a system-determined pre-selection) which synchronization action to perform.

Extend the Enum List in the Schema

Click **OK** from the **Synchronize** dialog to have the schema editor read the tool meta-schema and automatically add the new enum extension to the tool map schema.



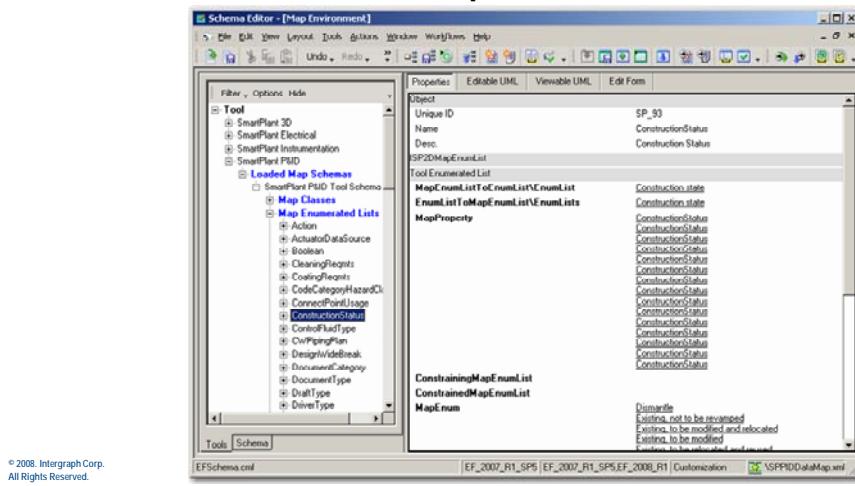
© 2008, Intergraph Corp.
All Rights Reserved.

10.1.3 Extending the SmartPlant Schema and Mapping the New Value

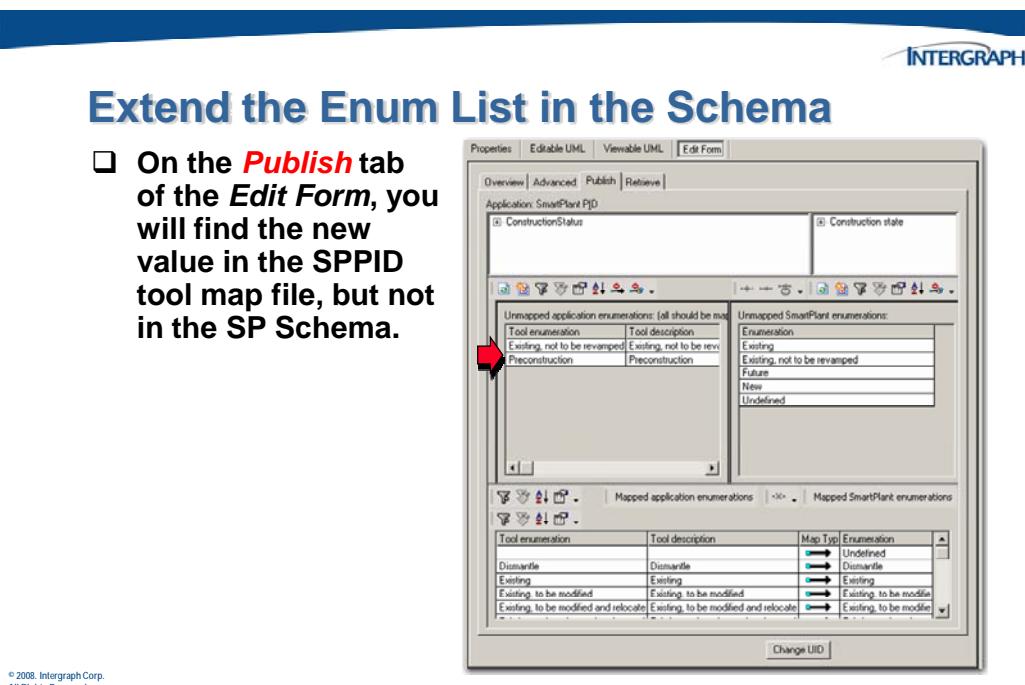
In the *Map Environment*, drill down to the *Map Enumerated Lists* and find the *Construction Status* list.

Extend the Enum List in the Schema

- In the SmartPlant P&ID tool map file, choose the **ConstructionStatus** Map Enumerated List.



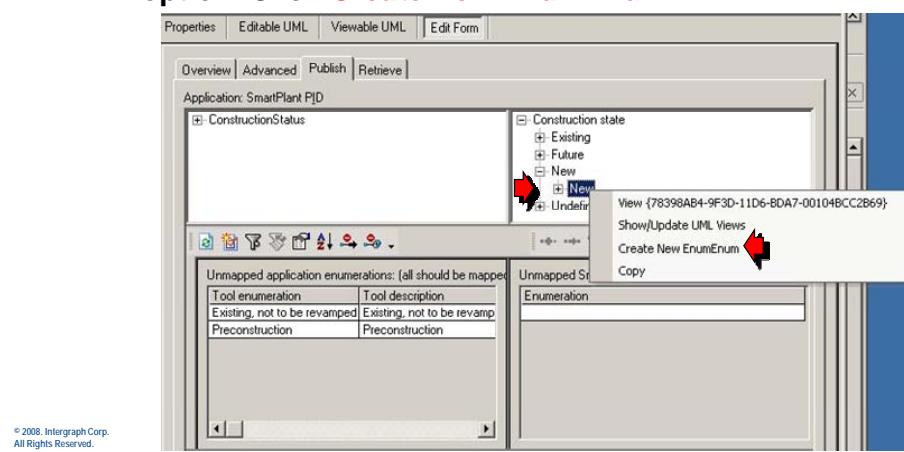
In the **Edit Form** view, go to the **Publish** tab.



Find an existing enum enum, right-click on it, and use the **Create New EnumEnum** command.

Extend the Enum List in the Schema

- In the SmartPlant Schema view def, drill down under **Construction state > New**, and right-click on the New option. Click **Create New EnumEnum**.

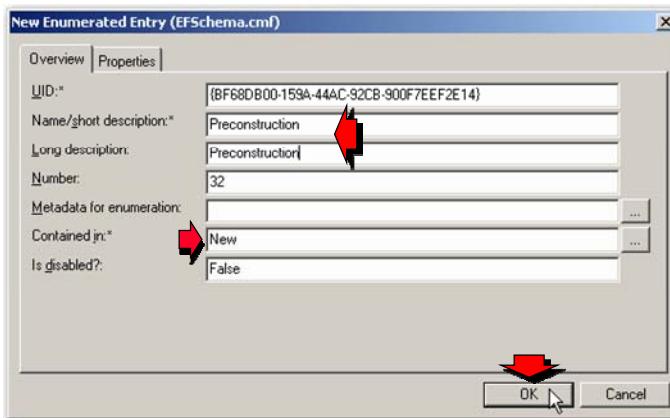


Provide information about the new enum enum that will be contained in the *New* EnumListType.



Extend the Enum List in the Schema

- Provide a name and description for the new enum enum, and confirm that it is contained in the *New* list. Click **OK**.



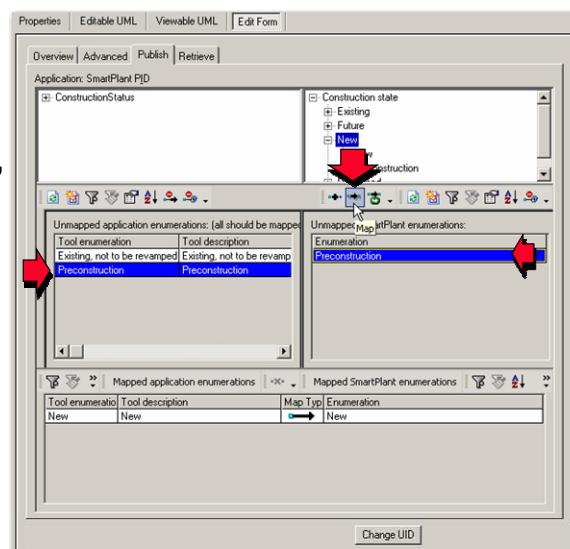
© 2008, Intergraph Corp.
All Rights Reserved.

Click **OK** to create the new value.



Extend the Enum List in the Schema

- Find the *Preconstruction* values in the tool map file and the SmartPlant schema, and use the Map button to create the publish mapping.

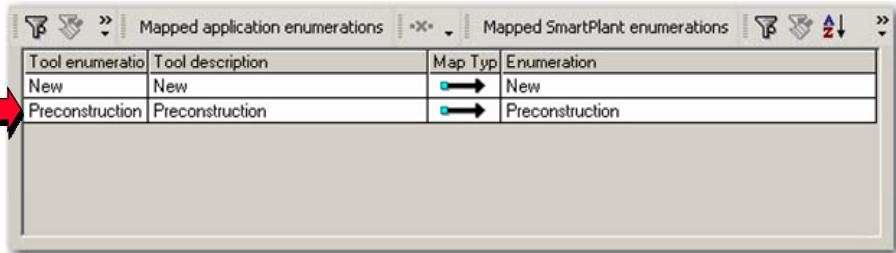


© 2008, Intergraph Corp.
All Rights Reserved.

At the bottom of the window, confirm the mapping.

Extend the Enum List in the Schema

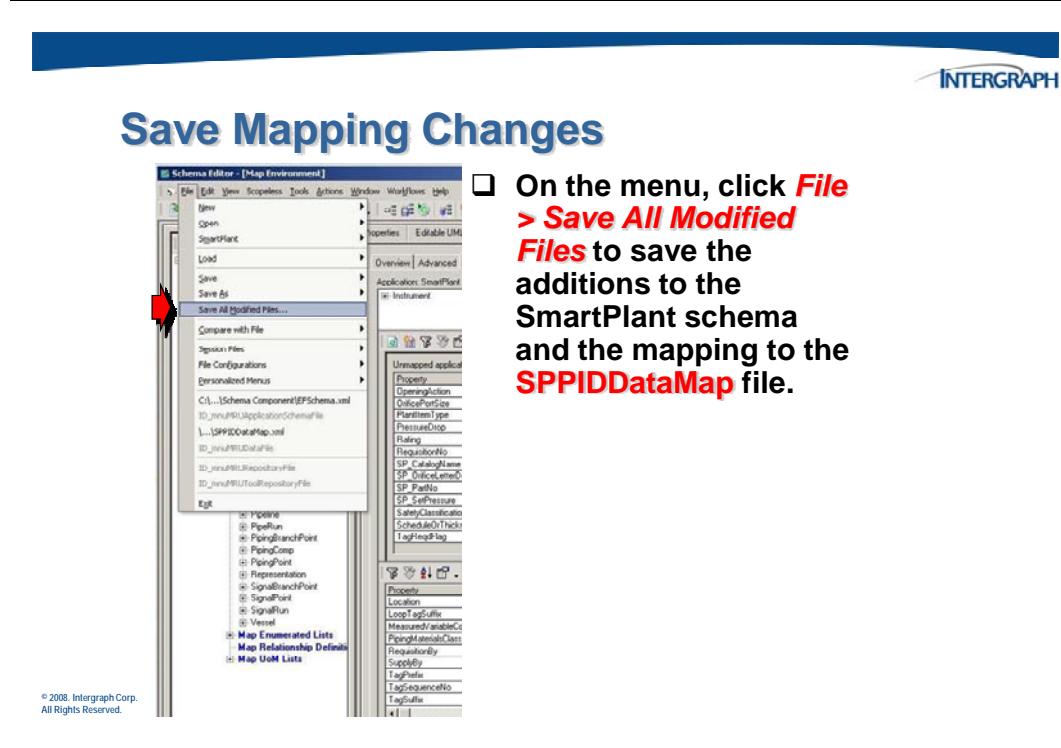
- Confirm the value was mapped properly.



Tool enumeration	Tool description	Map Typ	Enumeration
New	New	→	New
Preconstruction	Preconstruction	→	Preconstruction

10.1.4 Saving the Map File and Schema File Changes

Save all the modified files.





Save Mapping Changes

- Verify the tool map schema name for the mapping changes and choose **Yes**.



© 2008, Intergraph Corp.
All Rights Reserved.



Save Mapping Changes

- Verify the CMF file for the property addition and choose **Yes**.

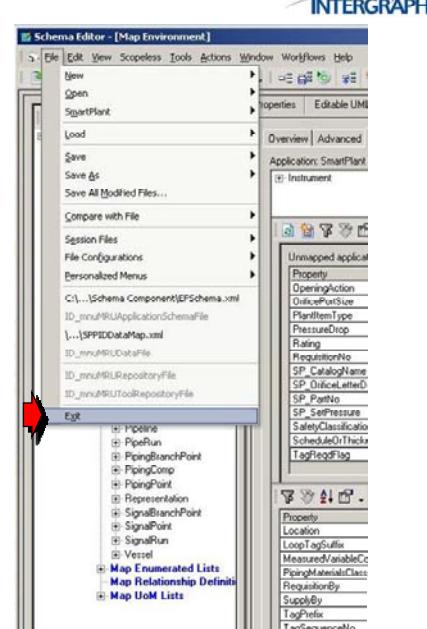


© 2008, Intergraph Corp.
All Rights Reserved.

Once all the modified files have been saved, close the Schema Editor.

Save Mapping Changes

- On the menu, click **File > Exit** to close out of the schema editor.



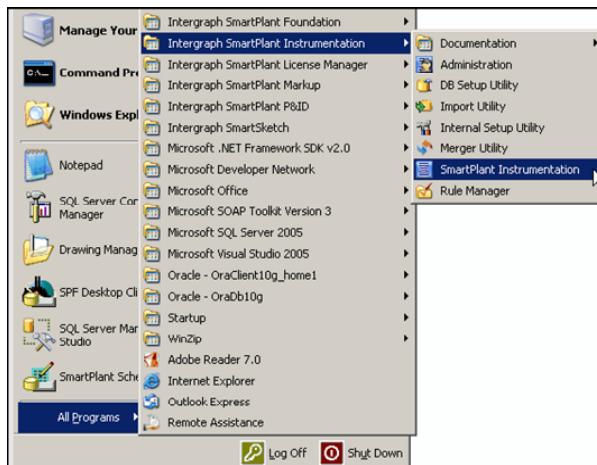
10.1.5 Extending an Enumerated List in SmartPlant Instrumentation

The next step to extending the system is to add the same information in the next authoring tool, such as SmartPlant Instrumentation.

Start SmartPlant Instrumentation in order to add an extension to an existing picklist.

Extend the Enum List in SPI

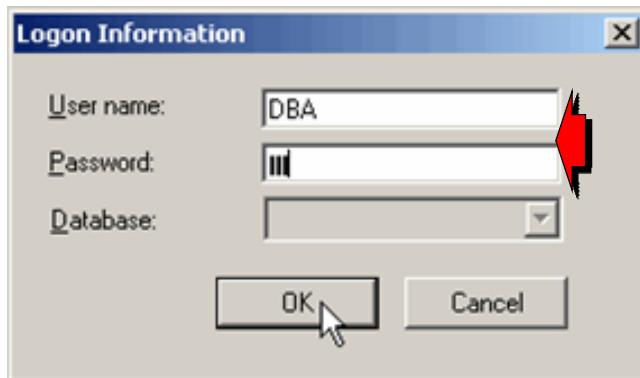
- ❑ To modify a picklist in SmartPlant Instrumentation, open SmartPlant Instrumentation by clicking **Start > All Programs > Intergraph SmartPlant Instrumentation > SmartPlant Instrumentation**.



The logon dialog box will appear. Log in as **DBA** with a password of **DBA**.

Extend the Enum List in SPI

- Enter a **User name** and **Password** to logon to SmartPlant Instrumentation.

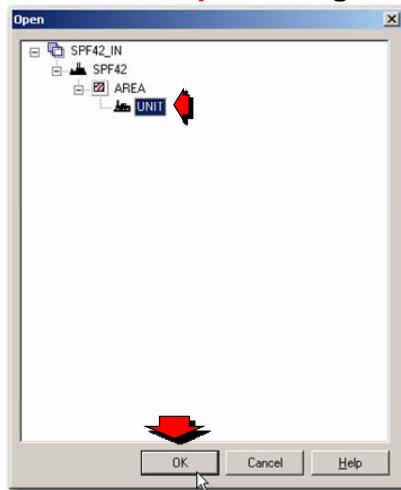


© 2008 Intergraph Corp.
All Rights Reserved.

When the **Open** window appears, choose a plant/unit from the displayed tree.

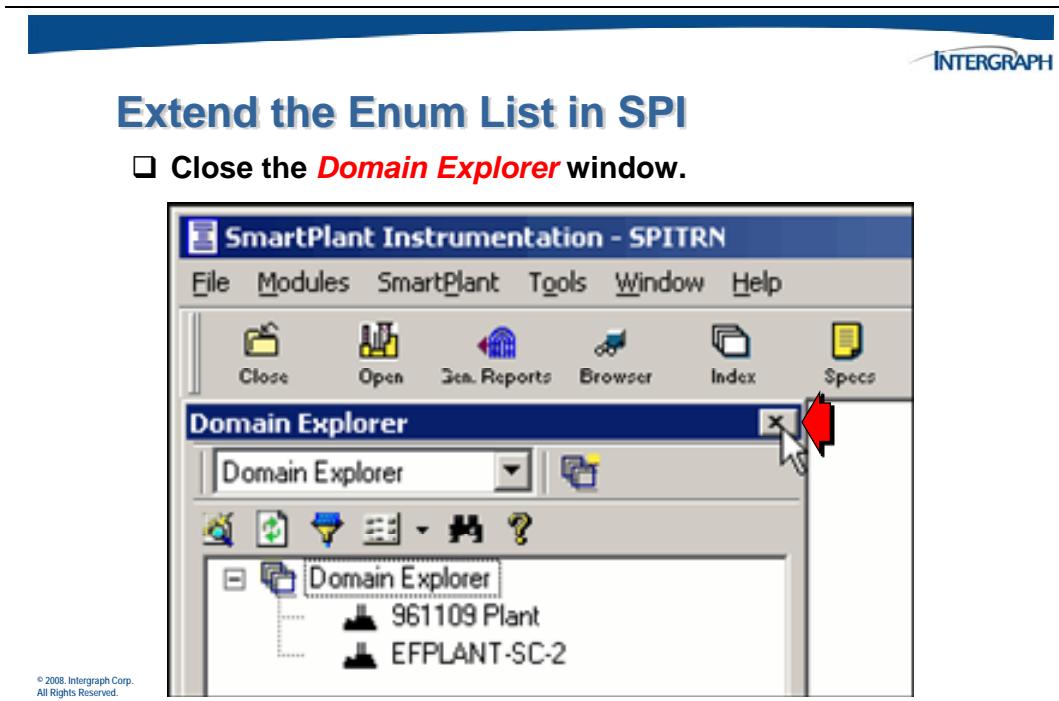
Extend the Enum List in SPI

- Choose a **Unit** from the **Open** dialog, and click **OK**.

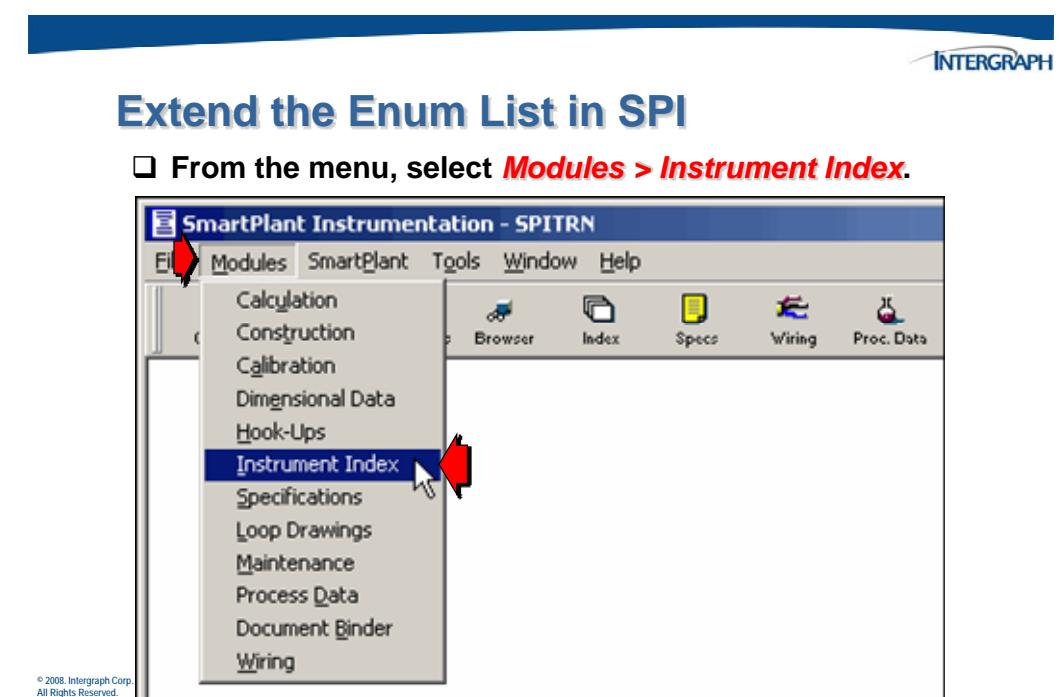


© 2008 Intergraph Corp.
All Rights Reserved.

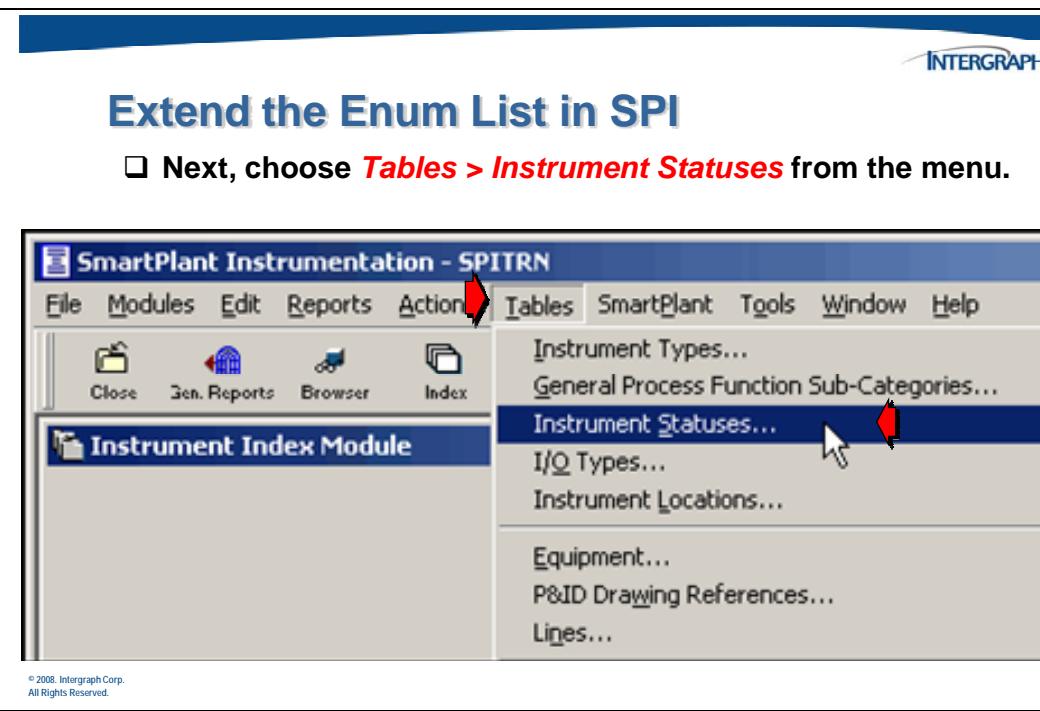
Close the Domain Explorer window that appears.



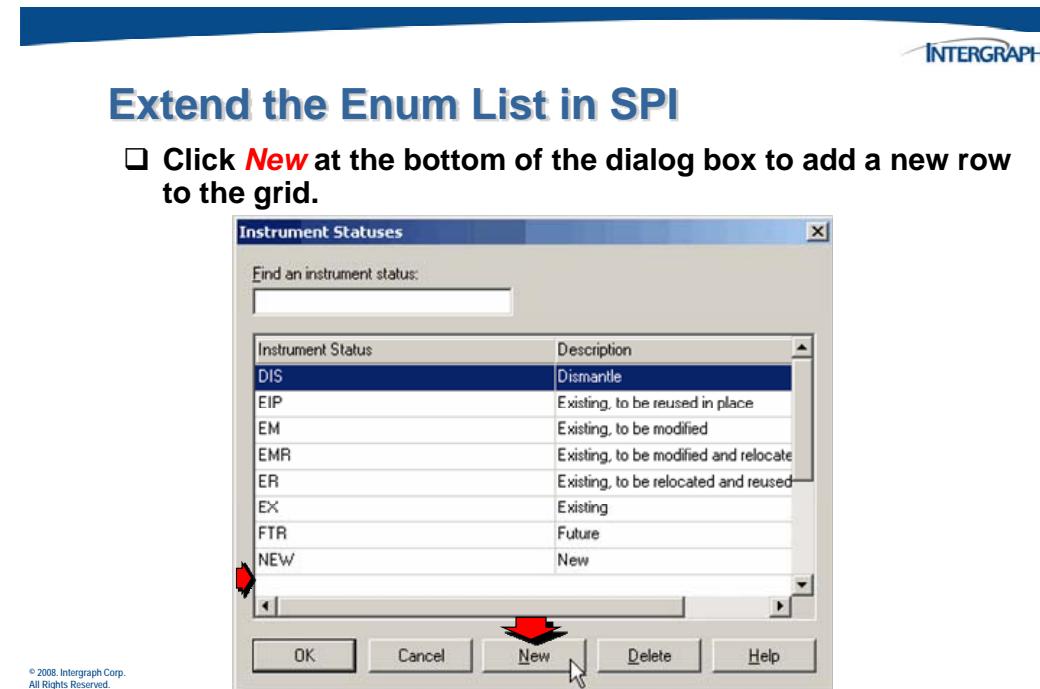
From the SmartPlant Instrumentation application window, open the *Instrument Index* module.



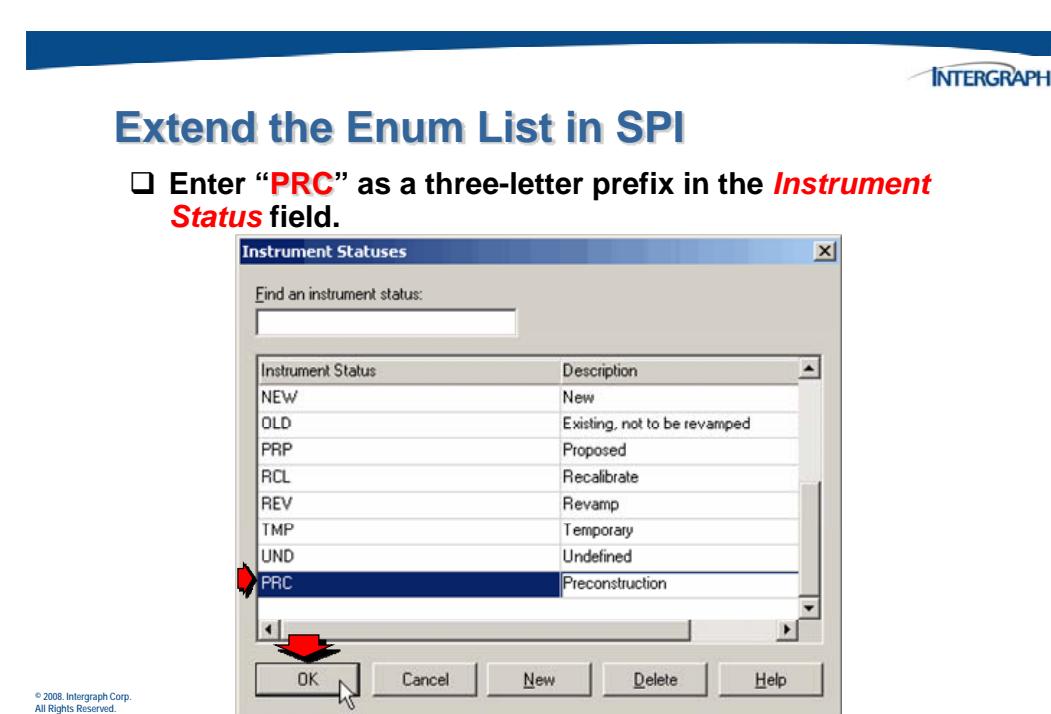
The *Instrument Index Module* window will appear. Click **Tables > Instrument Statuses**.



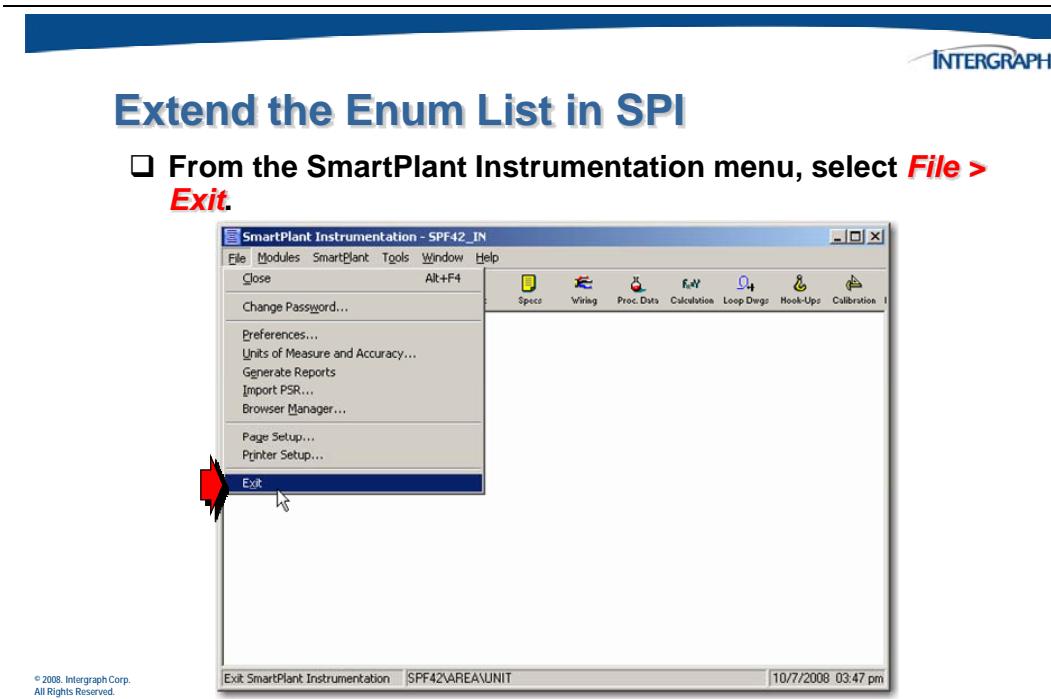
The *Instrument Statuses* dialog will appear. Click the **New** button to create a new row.



Enter the new picklist information in the new row at the bottom of the dialog, and click **OK**.



You can now exit the SmartPlant Instrumentation application.



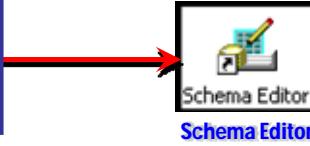
10.1.6 Extending the SPI Tool Map Schema

Again, before you can define mapping for these new entries in the tool map schema, the enumerated list must first be extended in the tool map file and the SmartPlant schema.



Extend the SPI Tool Map Schema

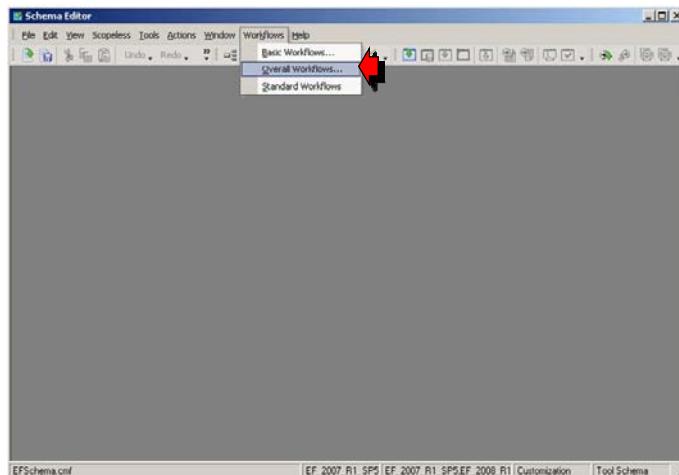
Add the new enumerated entry to the SmartPlant Instrumentation tool map schema and map it



Launch the Schema Editor application. Again, this is assuming that you currently have the CMF file checked out of the Desktop Client and an existing session file available.

Extend the SPI Tool Map Schema

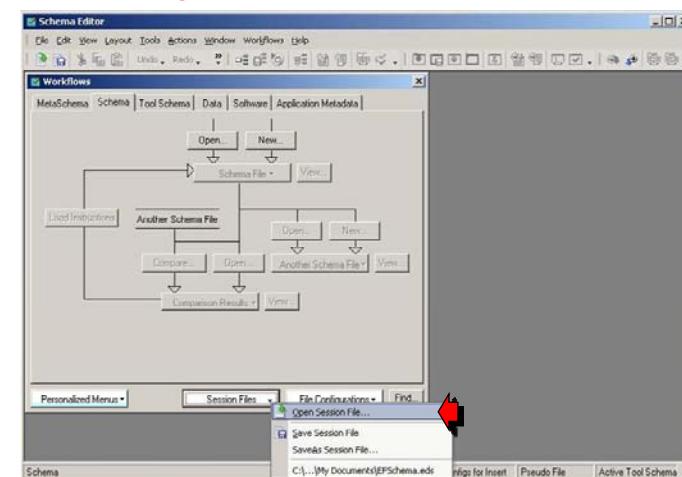
- Launch the Schema Editor and from the **Windows** menu, choose the **Overall Workflow**.



Open the **Overall Workflows**, and use the **Session Files** button to open the existing session file for the plant.

Extend the SPI Tool Map Schema

- On the **Schema** tab, click the **Session Files** button and choose the **Open Session** file command.

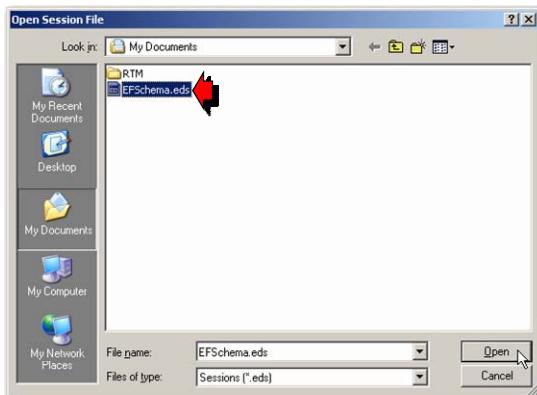


Find the session file, and choose it to open.



Extend the SPI Tool Map Schema

- Find the **EFSchema.eds** file in your *My Documents* folder, and open it.



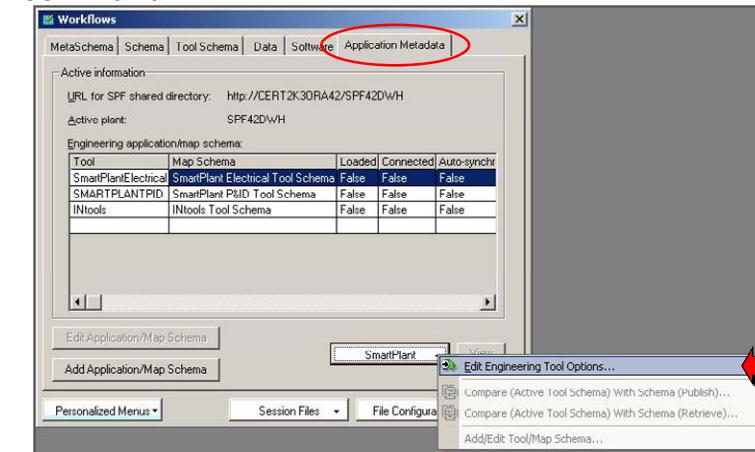
© 2008, Intergraph Corp.
All Rights Reserved.

Go to the *Application Metadata* tab, and click the *SmartPlant* button. From the shortcut menu, click the *Edit Engineering Tool Options* command.



Extend the SPI Tool Map Schema

- From the *Metadata Adapter* tab, click the *SmartPlant* button, and choose the *Edit Engineering Tool Options* command.

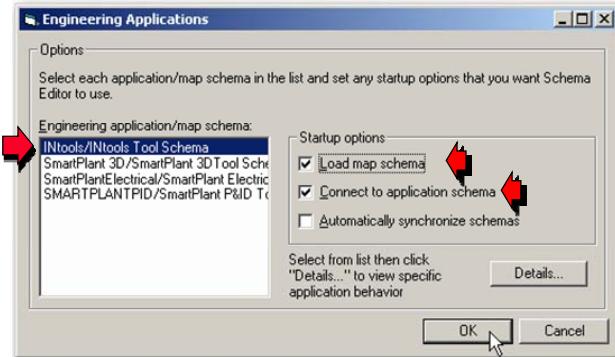


© 2008, Intergraph Corp.
All Rights Reserved.

From the list of available tool map files, choose the file for SmartPlant Instrumentation (INtools). Use the check boxes to open both the tool map file and the tool application schema.

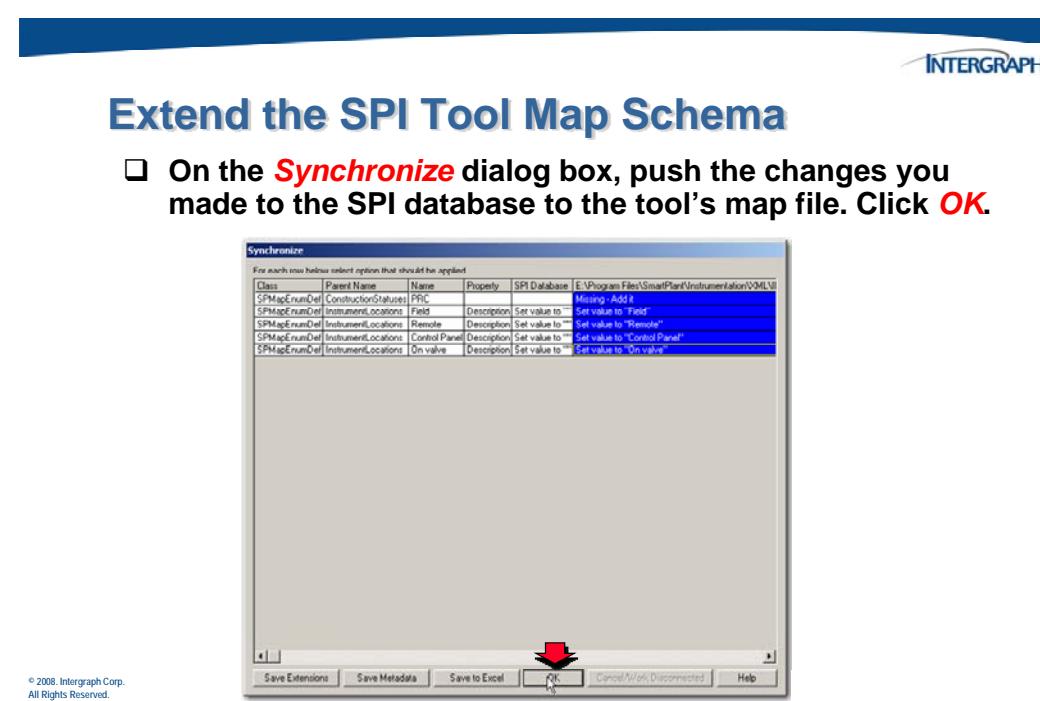
Extend the SPI Tool Map Schema

- ❑ Choose the *INTools tool schema*, and select **Load map schema** and **Connect to application schema**. Then click **OK**.



© 2008, Intergraph Corp.
All Rights Reserved.

The **Synchronize** dialog box shows the differences between the tool map file and the tool meta schema. Click **OK** to update the tool map file.



10.1.7 Mapping New Enum Entries for SPI

The enum entry “PRC” is automatically added to the tool map schema. Because the value for Preconstruction has already been added to the SmartPlant schema (CMF file) when we set up the mapping from SmartPlant P&ID, all that remains to be done is the mapping.

In the **Map Environment** window, drill down under SmartPlant Instrumentation to the **Map Enumerated Lists** section, and find **Construction Status**.

From the **Publish** tab on the **Edit Form** view, find the new Preconstruction value in the tool map file and the SmartPlant Schema, and click the **Map** button.

Map the Enum List in the Tool Schema

Choose the **Construction Statuses Map Enumerated List**, and from the **Publish** tab of the **Edit Form**, find the new value in the tool map file and the SP Schema.

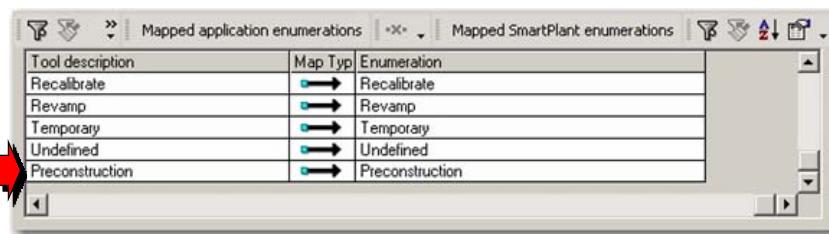
Click the **Map** button to create the mapping.

© 2008, Intergraph Corp.
All Rights Reserved.

Confirm that the status was mapped properly in the bottom of the window.

Map the Enum List in the Tool Schema

- Confirm that the values were mapped.

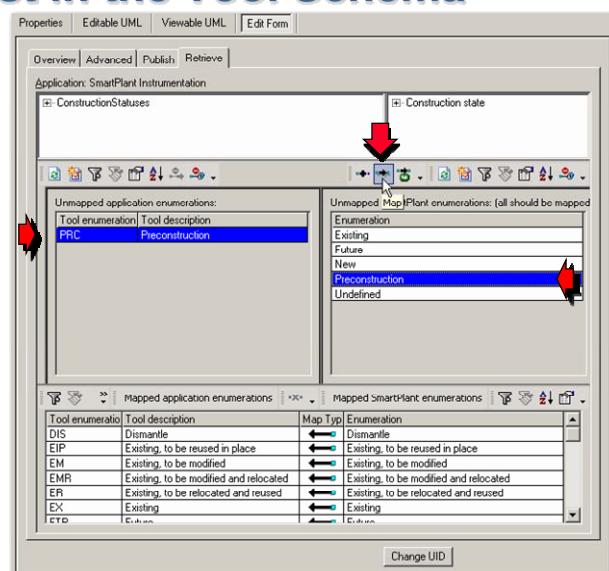


© 2008, Intergraph Corp.
All Rights Reserved.

On the **Retrieve** tab, repeat the same steps.

Map the Enum List in the Tool Schema

- From the **Retrieve** tab of the **Edit Form**, repeat the process to map the values.

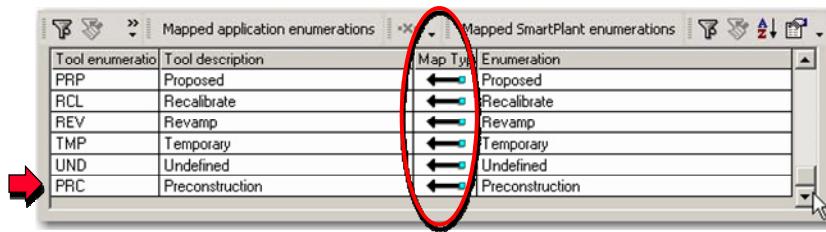


© 2008, Intergraph Corp.
All Rights Reserved.

Verify the mapping relationship was created properly for the retrieve.

Define Schema Mapping

- Again, verify that the mapping relationship was created.



Tool enumeration	Tool description	Map Type	Enumeration
PRP	Proposed	↳	Proposed
RCL	Recalibrate	↳	Recalibrate
REV	Revamp	↳	Revamp
TMP	Temporary	↳	Temporary
UND	Undefined	↳	Undefined
PRC	Preconstruction	↳	Preconstruction

10.2 Modifying the View Def

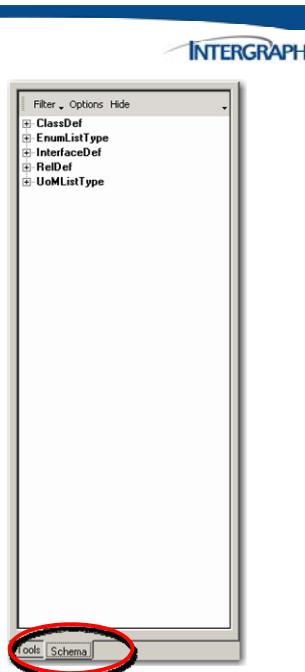
Depending on the property that you want to add to the SmartPlant Schema, you may have to extend the *view definitions*, *graph definitions* and *edge definitions*. When an enumerated list is being extended to add an entry, nothing additional has to be done to the Edge Definition if the property is already in the View Definition. In the case for construction status enumeration, a new entry, “Preconstruction”, was added to **ConstructionStatus2** or the second level of the construction status enumerations. The delivered View Definitions only have the first level of construction status enumeration in the definition. If you want to see **Preconstruction** when you select a piece of equipment in SPF, you will have to add *Construction Status2* to the view definition which is the second level of the Construction Status Hierachal Enumeration.

To extend the view def, we need to make modifications to the SmartPlant schema. In the window on the far left, go to the **Schema** tab.

Modifying View Defs

- If you want to be able to see the Preconstruction status in the Desktop Client, you will need to modify the delivered view def to display it.

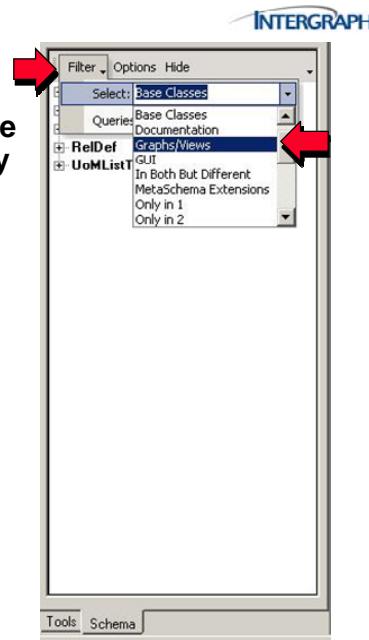
- Open the **Schema** tab of the left-hand pane in the *Map Environment* window.



Open the **Filter** menu, and use the **Select** option to have the tree display Graph defs and View Defs.

Modifying View Defs

- By default, this window displays the base classes. Open the **Filter** list by clicking on **Filter** at the top of the window.
- From the **Select** list, choose the **Graphs/Views** option.

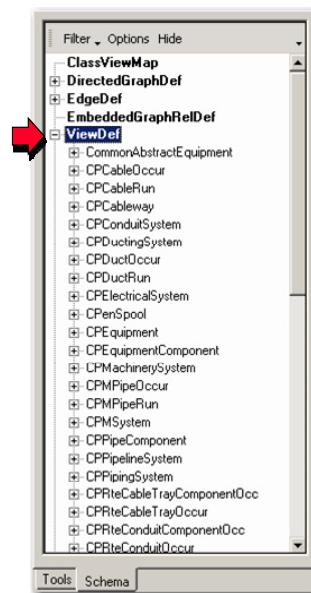


© 2008, Intergraph Corp.
All Rights Reserved.

Drill down under **ViewDefs**, and find the view def called **PIDInstrument**.

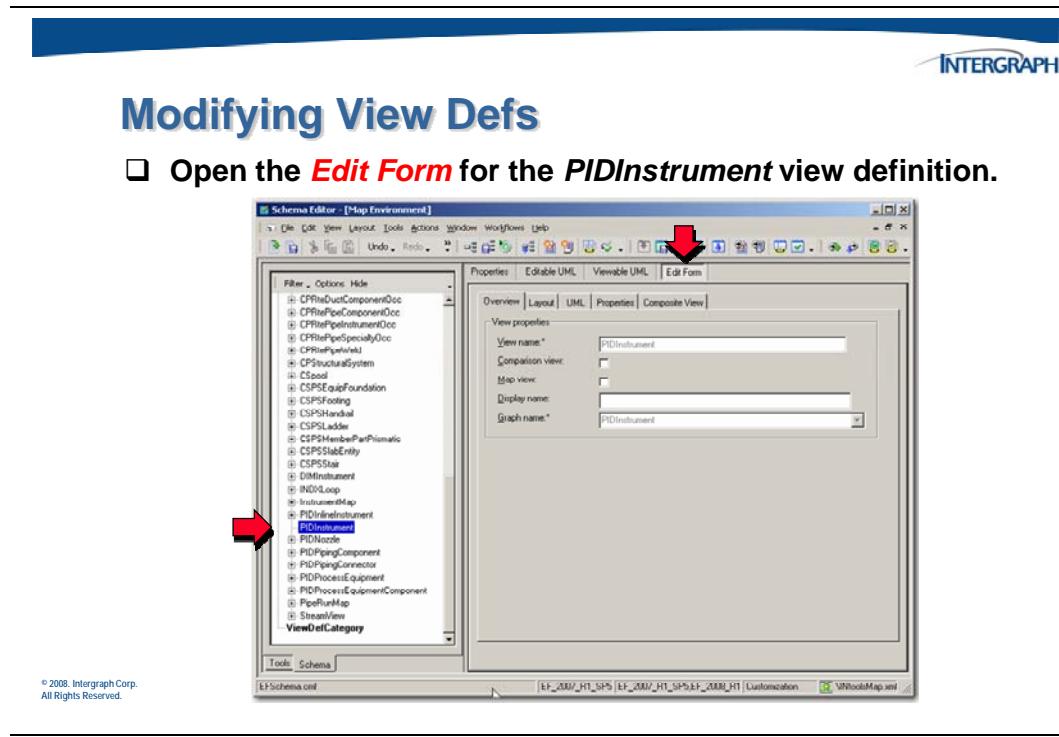
Modifying View Defs

- Expand the **ViewDef** list, and find the **PIDInstrument** view definition.

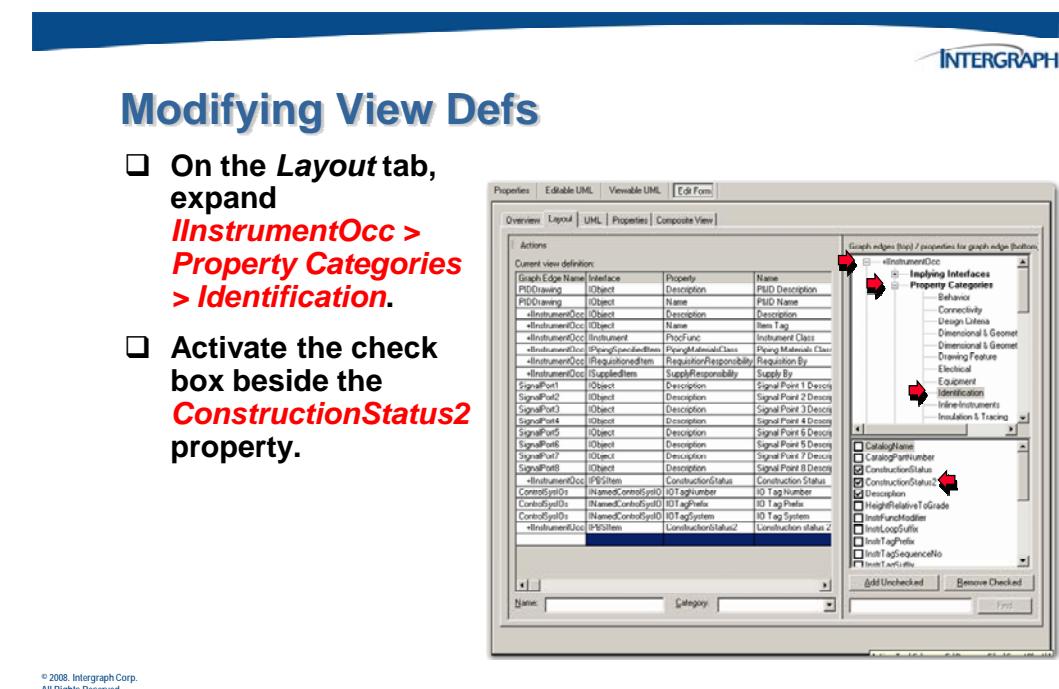


© 2008, Intergraph Corp.
All Rights Reserved.

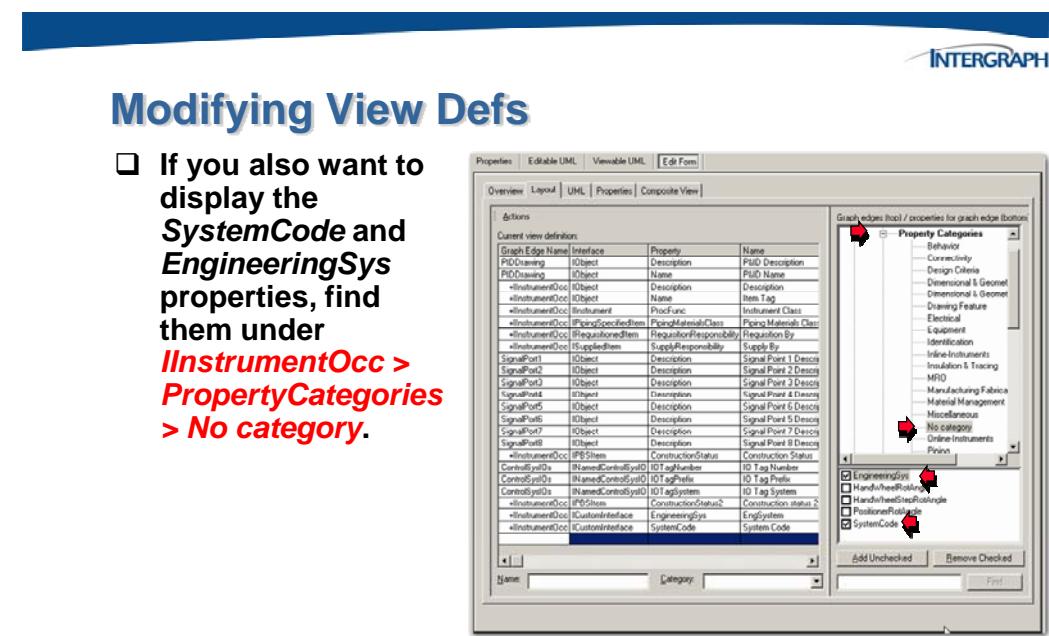
Open the **Edit Form** view of the view def.



Go to the **Layout** tab. Find the **ConstructionStatus2** property from the **IInstrumentOcc** interface, and activate the check box by it to add it to the view def.

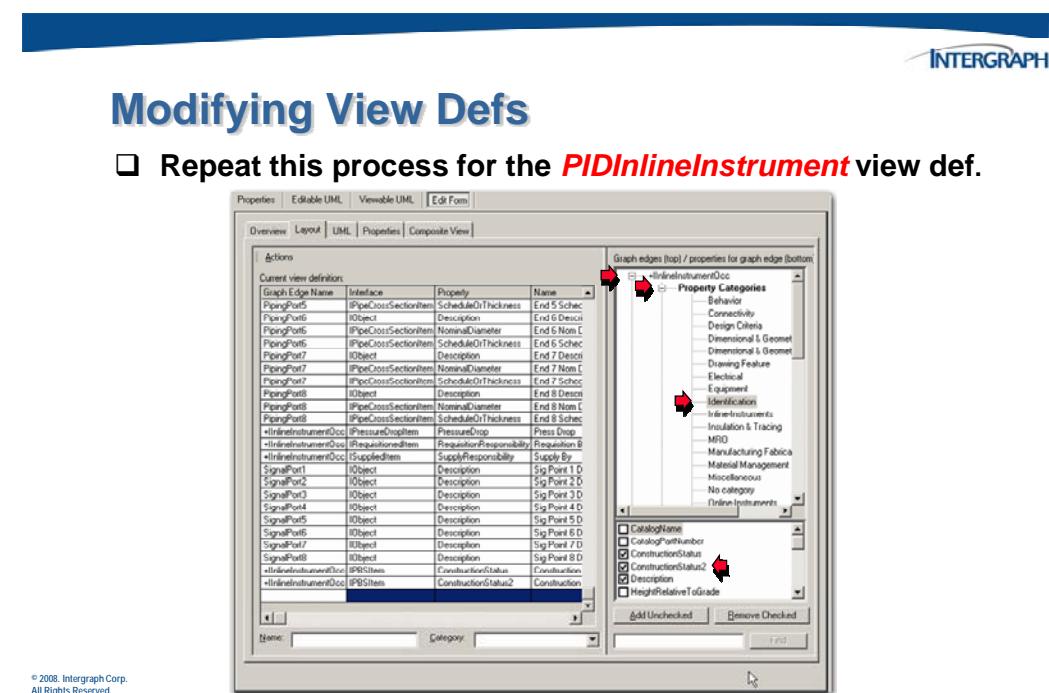


If you want to also add the new properties – *SystemCode* and *EngineeringSys* – select those properties as well.



© 2008, Intergraph Corp.
All Rights Reserved.

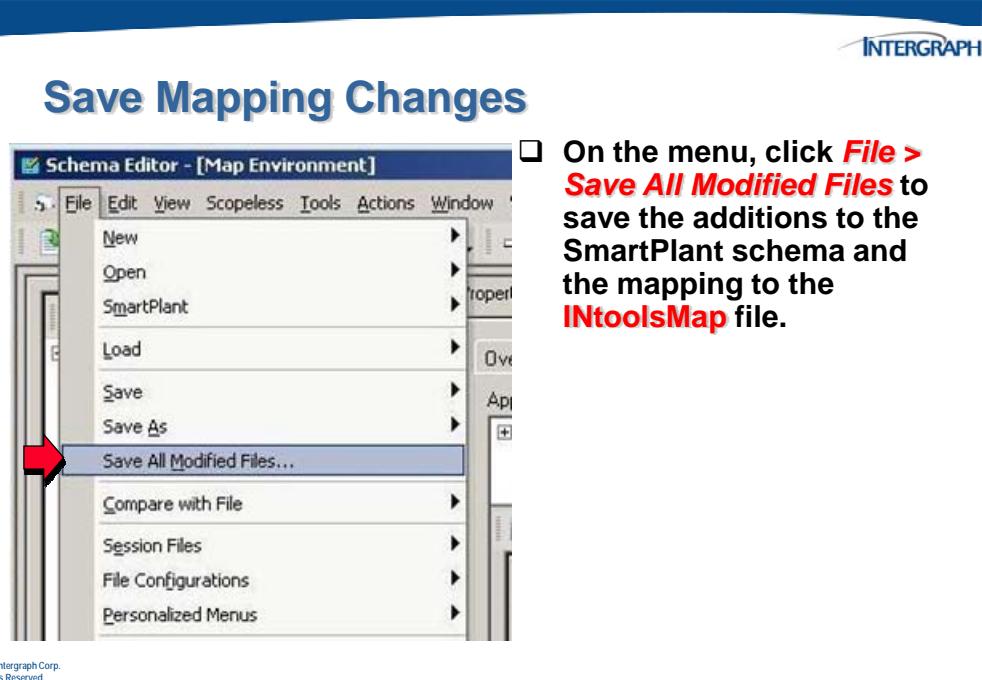
Repeat the same steps for other view defs, as well.



© 2008, Intergraph Corp.
All Rights Reserved.

10.3 Save the Schema Changes

Once you have created the mapping relationships and extended the view defs, you need to save the changes you have made to various files before closing the Schema Editor.



- On the menu, click **File > Save All Modified Files** to save the additions to the SmartPlant schema and the mapping to the INtoolsMap file.

© 2008, Intergraph Corp.
All Rights Reserved.

When prompted, save your changes to the SPI tool map file. This file is where mapping relationship information is stored.

Save Mapping Changes

- To confirm that you want to save the changes you have made to the *INtoolsMap.xml* file, click **Yes**.



© 2008, Intergraph Corp.
All Rights Reserved.

When prompted, save the changes to the CMF file. If you did not extend the view defs, you will not be prompted to save this file.

Save Mapping Changes

- Verify the CMF file for the property addition and choose **Yes**.

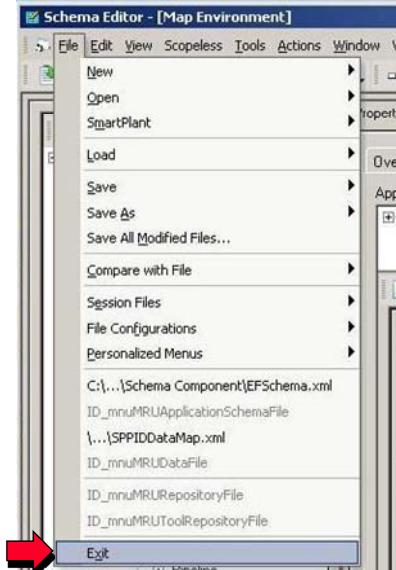


© 2008, Intergraph Corp.
All Rights Reserved.

Once the modified files have been saved, you can close the Schema Editor.

Save Mapping Changes

- Confirm that all modified files have been saved.**
- Then, click *File > Exit* to close out of the schema editor.**



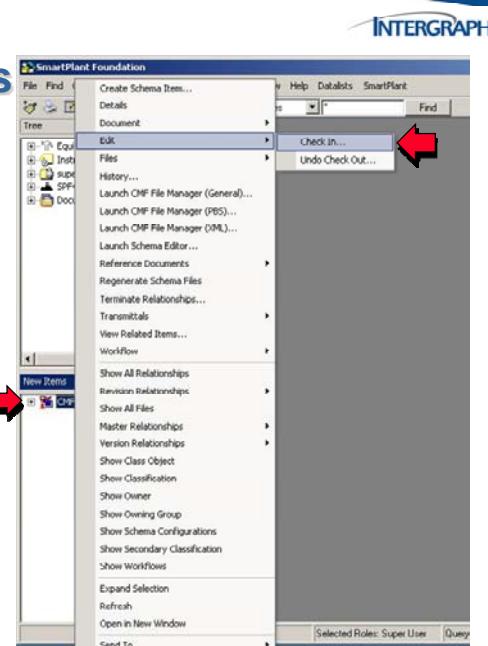
10.4 Loading the Schema Changes

Remember from chapter 9 that the Schema Import Wizard is used to load the SmartPlant schema extensions into the SPF database.

First, we need to make sure to check in the CMF file to copy up our changes we made to that file. From the Desktop Client, find the checked out CMF file, and use the **Edit > Check In** command.

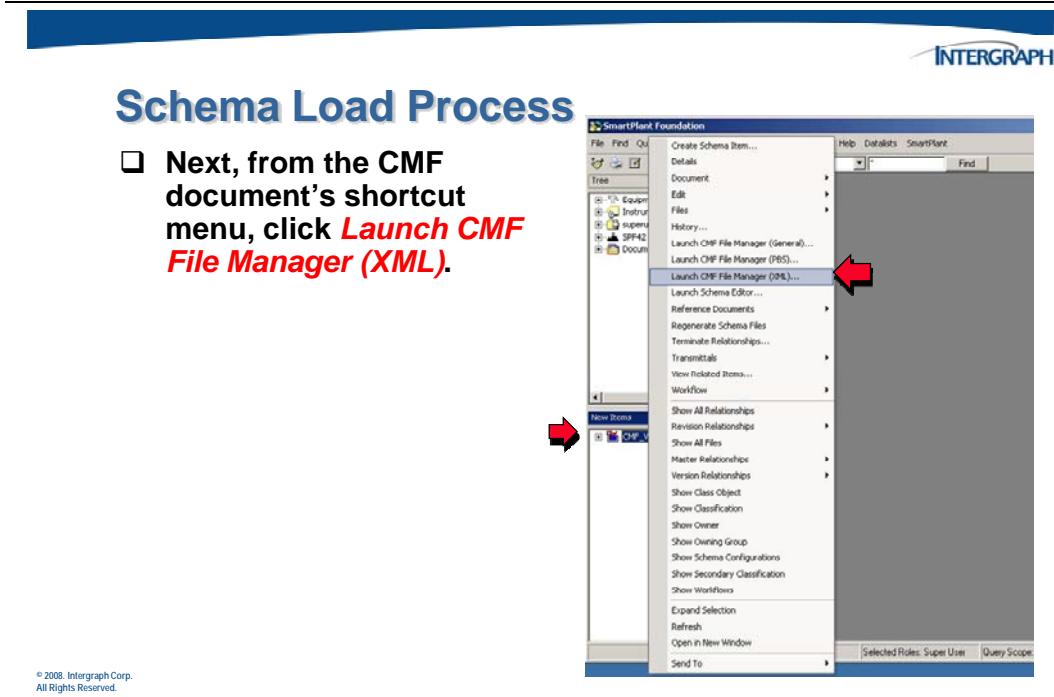
Schema Load Process

- Once you have completed your changes to the CMF file, return to the Desktop Client.
- Make sure the CMF file is currently checked into SPF.

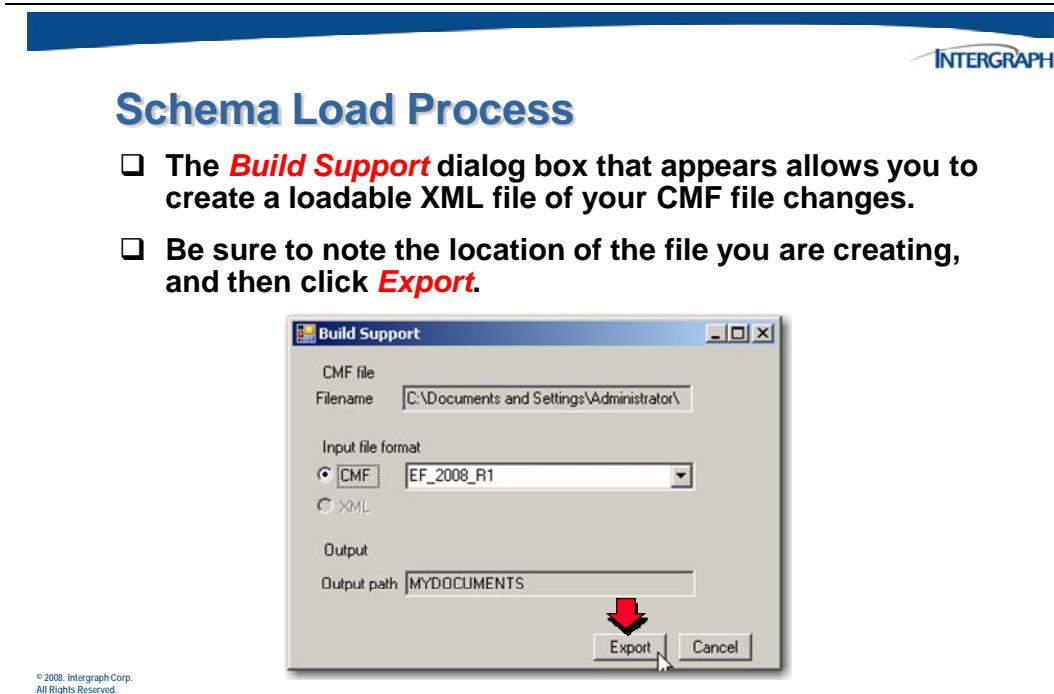


© 2008, Intergraph Corp.
All Rights Reserved.

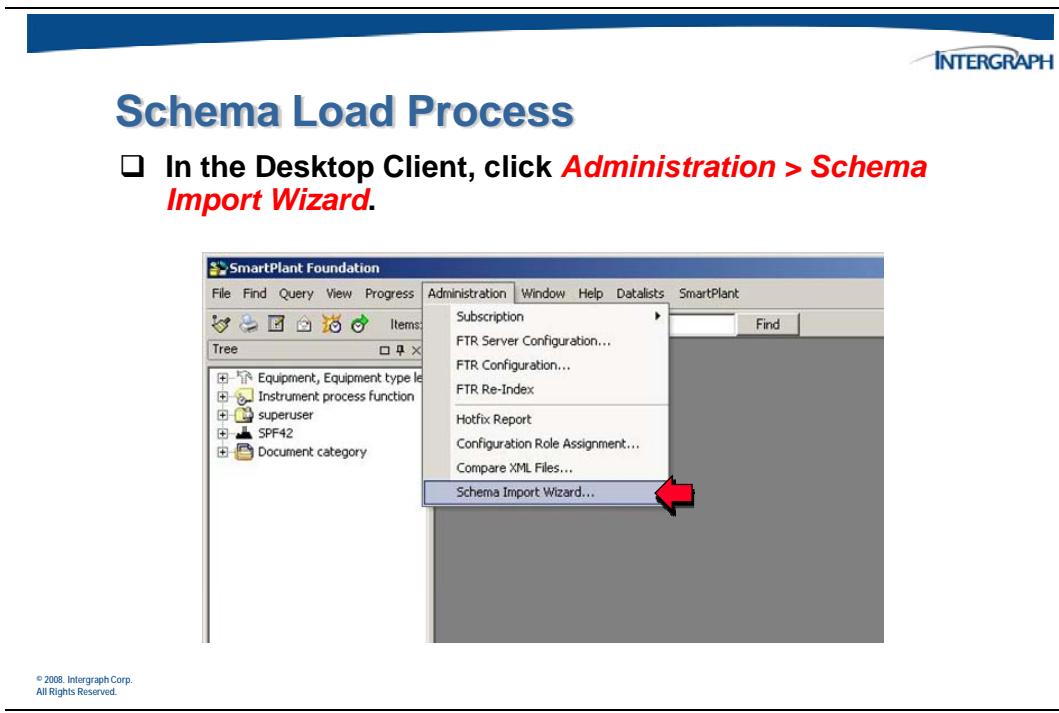
Once the file is checked in, right-click the object and choose the ***Launch CMF File Manager (XML)*** command to create an XML of the CMF.



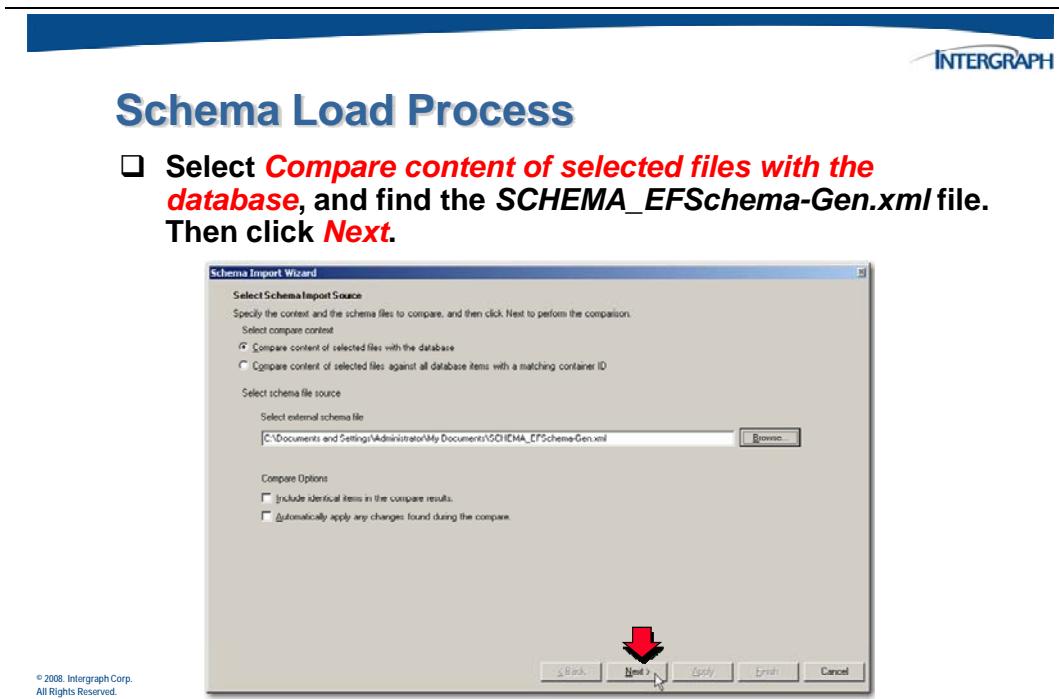
On the ***Build Support*** dialog box, choose the SPF 2008 CMF file, and be sure to note where the system will place the XML file it will create.



Return to the Desktop Client and start the Schema Import Wizard by clicking **Administration > Schema Import Wizard**.



Specify that you want to compare a selected file to the database, and then find the XML just created.



The Comparison screen shows you a list of the changes between the XML file and the SPF database. Choose which changes you want to incorporate, and click *Next*.

Schema Load Process

Review the list of differences between the XML file and the database. Select the check box for all changes you want to make to the database, and click **Next**.

Object Class	Object State	Object UID	Object Name	DefUID	UID1	UID2	Load State
Rel	New	(007633DC-E24B-...			Exposes	ICustom...	Engineer...
Rel	New	(07062C2A-A007-...			Implies	ICustom...	IDObject...
Rel	New	(0AD910E3-2D09-...			Realizes	PIDPin...	ICustom...
Rel	New	(0C759C27-0F72-...			Implies	IRnum...	ICustom...
Rel	New	(2A0266A-SF654...			Contains	(AD421...	(71FD1...
Rel	New	(2D5AC904-0FC5...			Implies	INovate...	ICustom...
Rel	New	(368EC2C7-6502...			Implies	IPpingC...	ICustom...
Rel	New	(504D7B65-709C...			Realizes	PIDNod...	ICustom...
EnumEnum	New	(546FB118-9314...	DC				
Rel	New	(5A54939A-79F8-4...			Realizes	PIDIntr...	ICustom...
EnumEnum	New	(6239CB69-94C5...	AA				
Rel	New	(639474D7-5D1B...			Realizes	PIDProc...	ICustom...
EnumEnum	New	(71FD1BC2-8CE8...	CA				
Rel	New	(7B9D9EED-837F...			Implies	IPipeline...	ICustom...
Rel	New	(8E0CF247-A414...			Implies	IRnum...	ICustom...
Rel	New	(9320C486-0A14...			Contains	(AD421...	(F20FF5...
Rel	New	(9A78FB09-5526...			ScopesBy	Engineer...	(AD421...
Rel	New	(9E91919F-F98B-4...			ScopesBy	SystemC...	String
Rel	New	(A7771105-3400...			Properties	Custom...	ISRT-SC
Rel	New	(8800FC2A7-A1A4...					

35 New 1 Updated 0 Removed Export Select All Clear All Details

< Back Next > Apply Finish Cancel

© 2008 Intergraph Corp.
All Rights Reserved.

The results screen shows you the changes that have been loaded into the database.

Schema Load Process

The results of the load process are displayed to you when it is complete. Click **Finish** to close the tool.

Object Class	Object State	Object UID	Object Name	DefUID	UID1	UID2	Load State
Rel	New	(007633DC-E24B-...					Loaded
Rel	New	(07062C2A-A007-...					Loaded
Rel	New	(0AD910E3-2D09-...					Loaded
Rel	New	(0C759C27-0F72-...					Loaded
Rel	New	(2A0266A-SF654...					Loaded
Rel	New	(2D5AC904-0FC5...					Loaded
Rel	New	(368EC2C7-6502...					Loaded
Rel	New	(504D7B65-709C...					Loaded
EnumEnum	New	(546FB118-9314...	DC				Loaded
Rel	New	(5A54939A-79F8-4...					Loaded
EnumEnum	New	(6239CB69-94C5...	AA				Loaded
Rel	New	(639474D7-5D1B...					Loaded
EnumEnum	New	(71FD1BC2-8CE8...	CA				Loaded
Rel	New	(7B9D9EED-837F...					Loaded
Rel	New	(8E0CF247-A414...					Loaded

Changes Summary
Items Inserted: 35
Items Updated: 1
Items Deleted: 0

< Back Next > Apply **Finish** Cancel

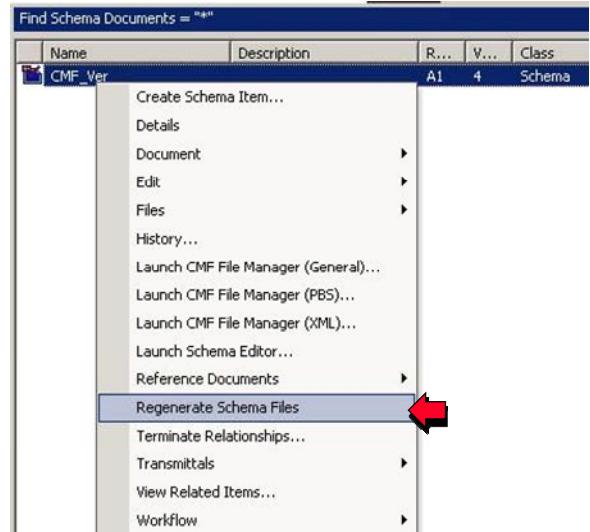
© 2008 Intergraph Corp.
All Rights Reserved.

Reminder

- After this import process, the system will regenerate the component schemas. If you want to test the mapping immediately, you can regenerate the component schemas manually.

Regenerating Component Schema

- To manually regenerate the component schemas, right-click the CMF file and use the **Regenerate Schema Files** command.



© 2008, Intergraph Corp.
All Rights Reserved.

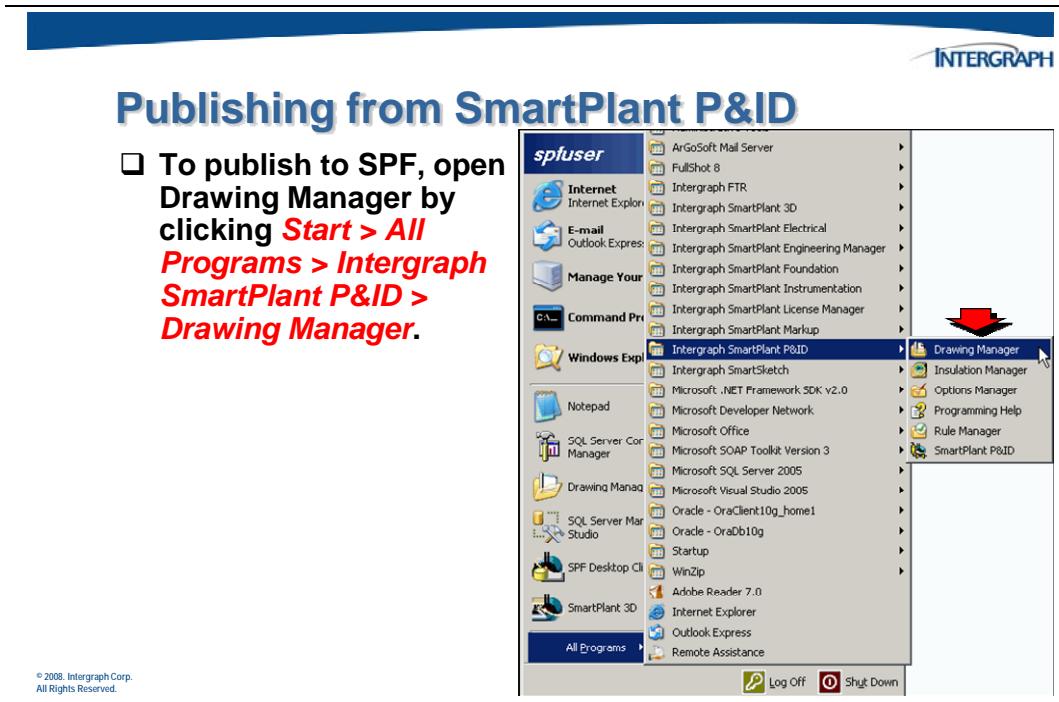
10.5 Testing the SPI Mapping

The following sections step you through the process of testing the SPI mapping. Since we want to test whether or not we can properly retrieve a value that has been mapped into SPI, we need to make sure that the value has been published from another authoring tool. That is the reason we set up the publish mapping from SPP&ID.

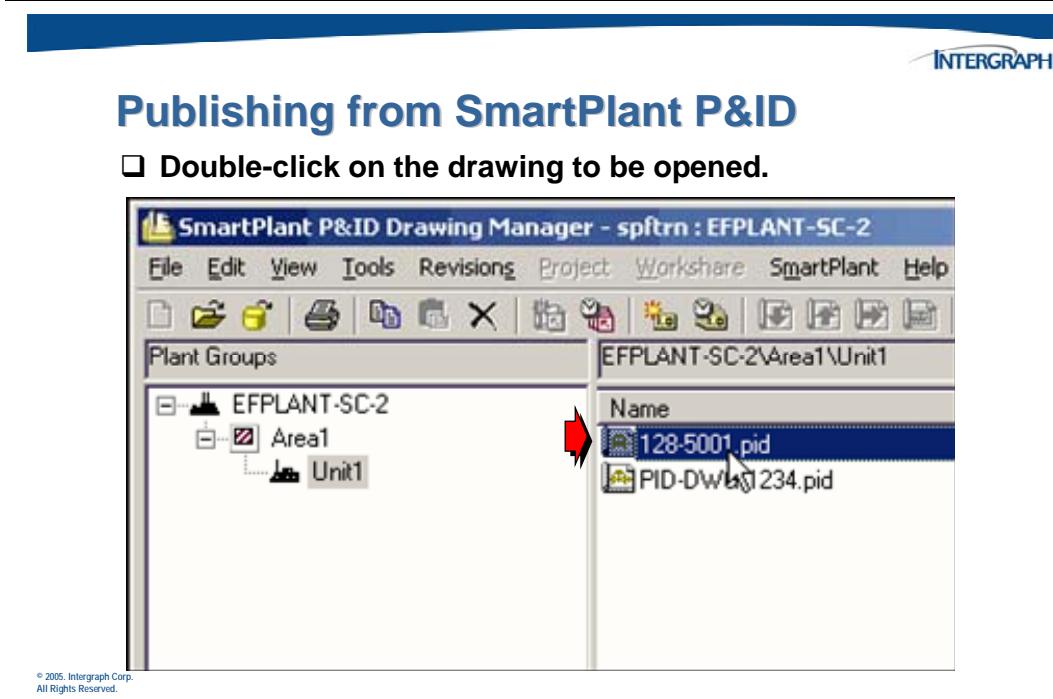
The first step of our test will be to publish the new value out of the SmartPlant P&ID tool on an instrument. Once the data has been published, we will also test the SmartPlant Instrumentation retrieve mapping by retrieving the P&ID drawing with the instrument and running the To Do List tasks to update the construction status of the instrument.

10.5.1 Publishing a Change from SmartPlant P&ID

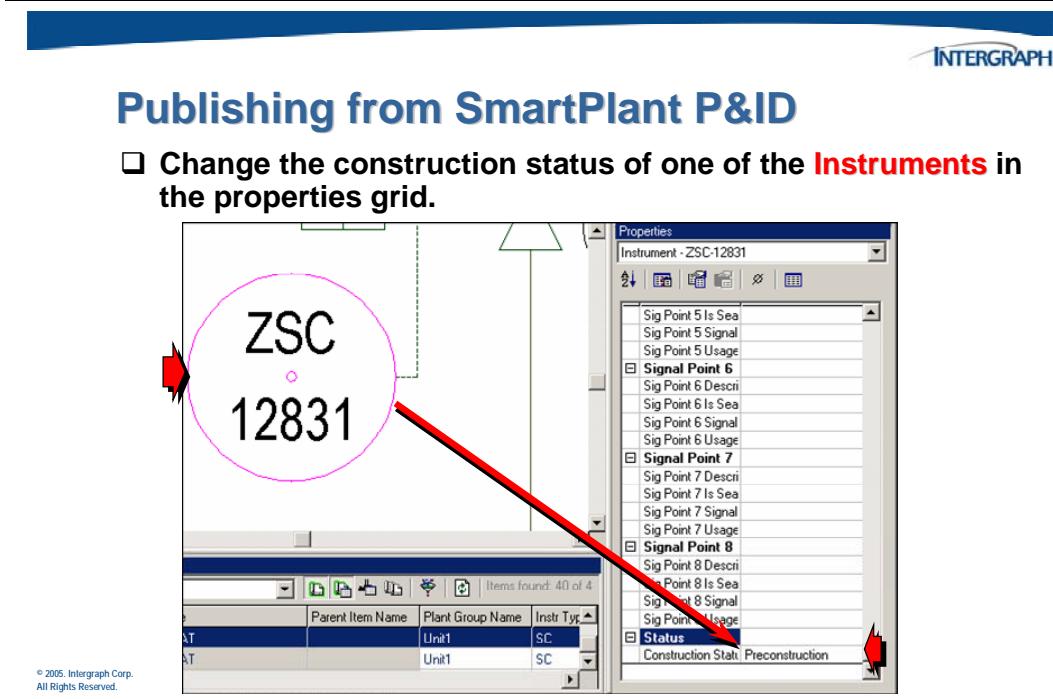
In this section, you will see an example of publishing a change to a P&ID instrument using the new **Construction Status** picklist entry of *Preconstruction*. Then, to test the SmartPlant capabilities, the changed instrument will be retrieved into SmartPlant Instrumentation and the Construction Status value reviewed.



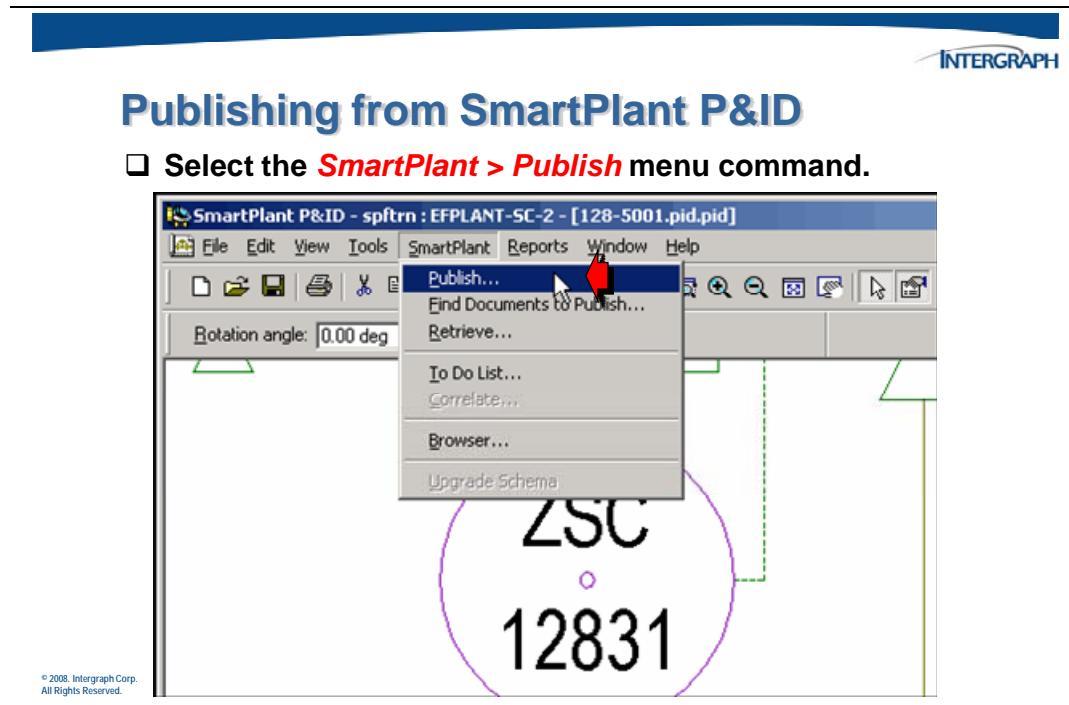
The *SmartPlant P&ID Drawing Manager* window will appear. The registered plant to be used for the publish operation is **SPF42**. Drill down beneath that plant\area\unit, and find the 128-5001.pid drawing. Double-click the drawing to open it.



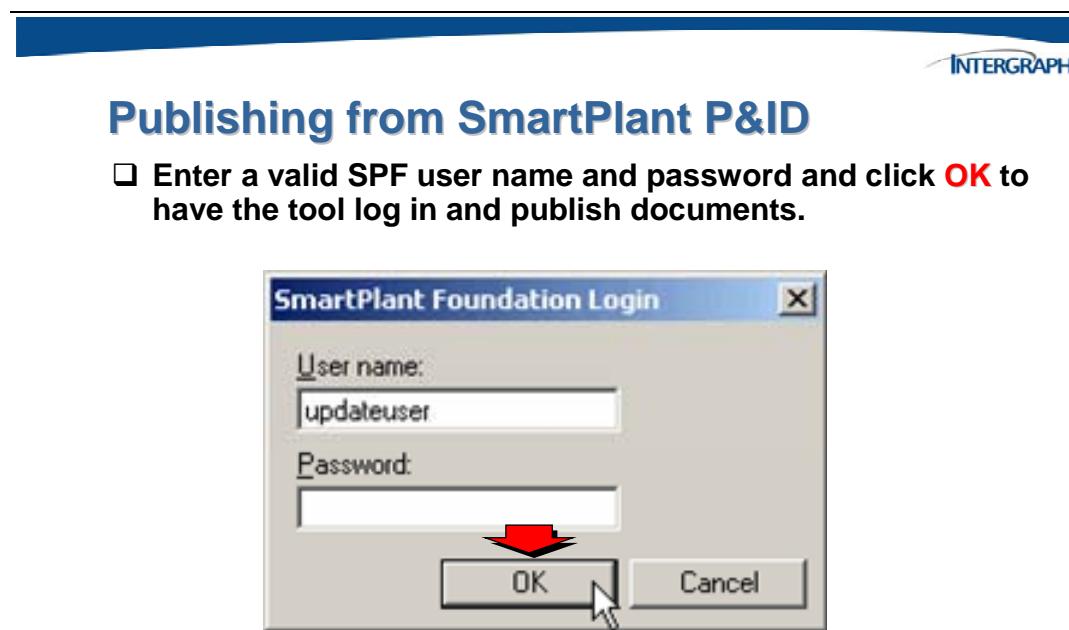
Locate some of the instruments that have been placed in this drawing. Update this existing instrument by using the new value for the construction status.



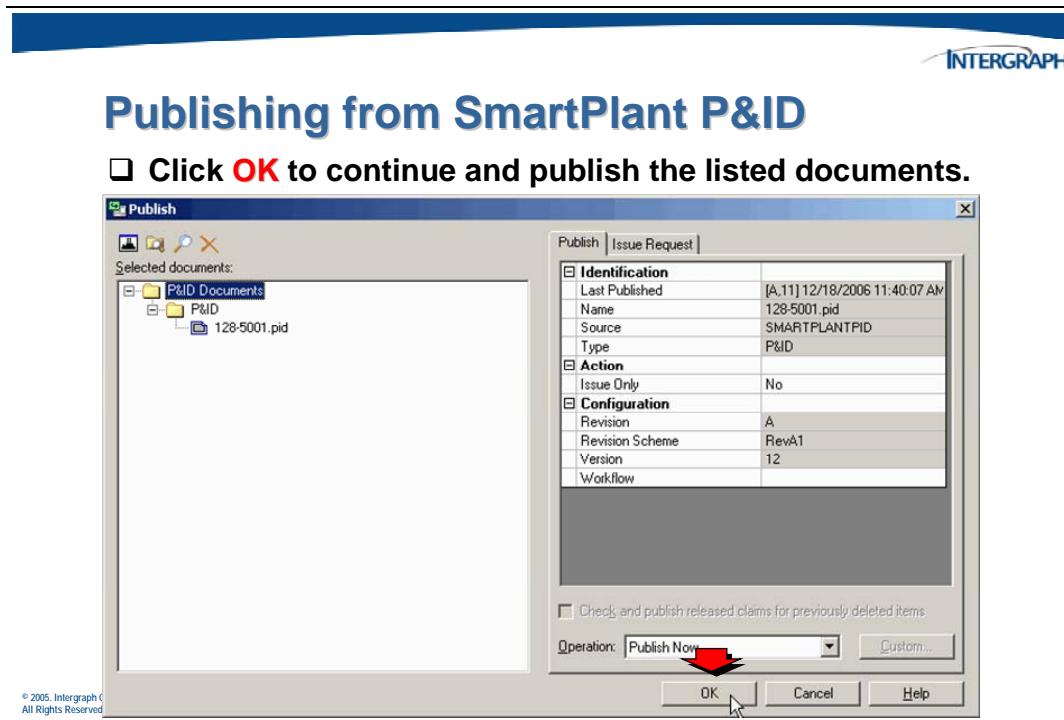
Save the change that has been made before publishing this document to SPF. To perform a publish operation, use the **Publish** command from SmartPlant P&ID.



In certain situations, you may be prompted to log on to SPF. In the class environment, you will not have to log on because your system user has an SPF account.



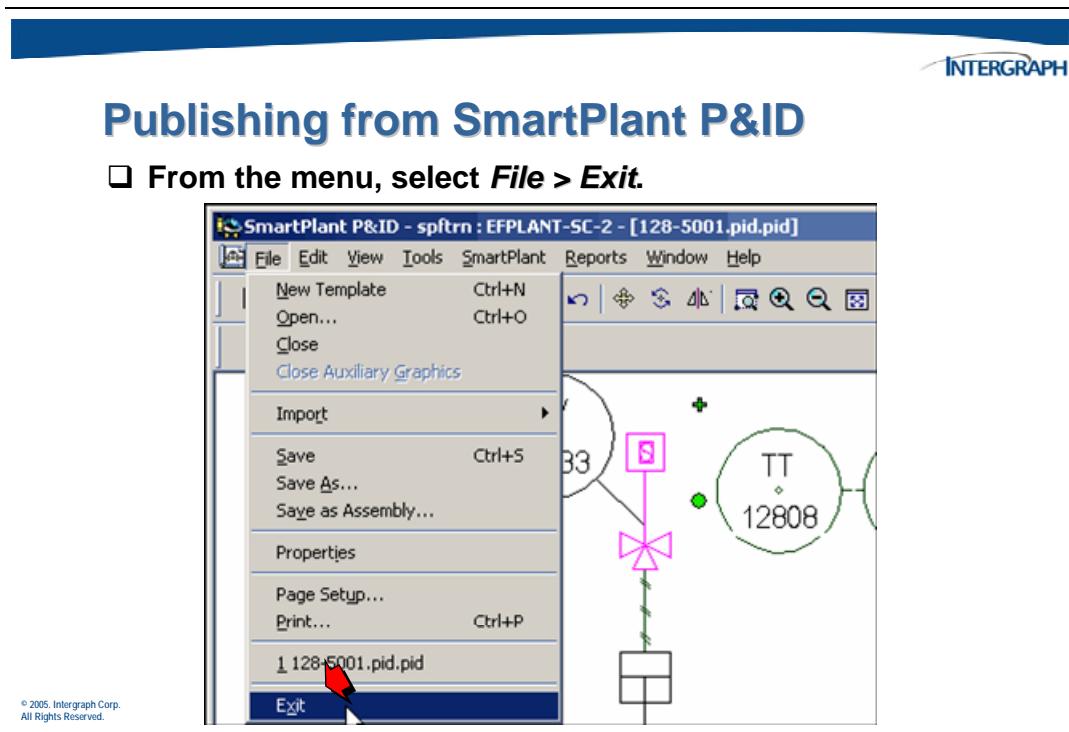
The *Publish* dialog box will appear.



A dialog box appears after some time to report success or failure of the publish operation.

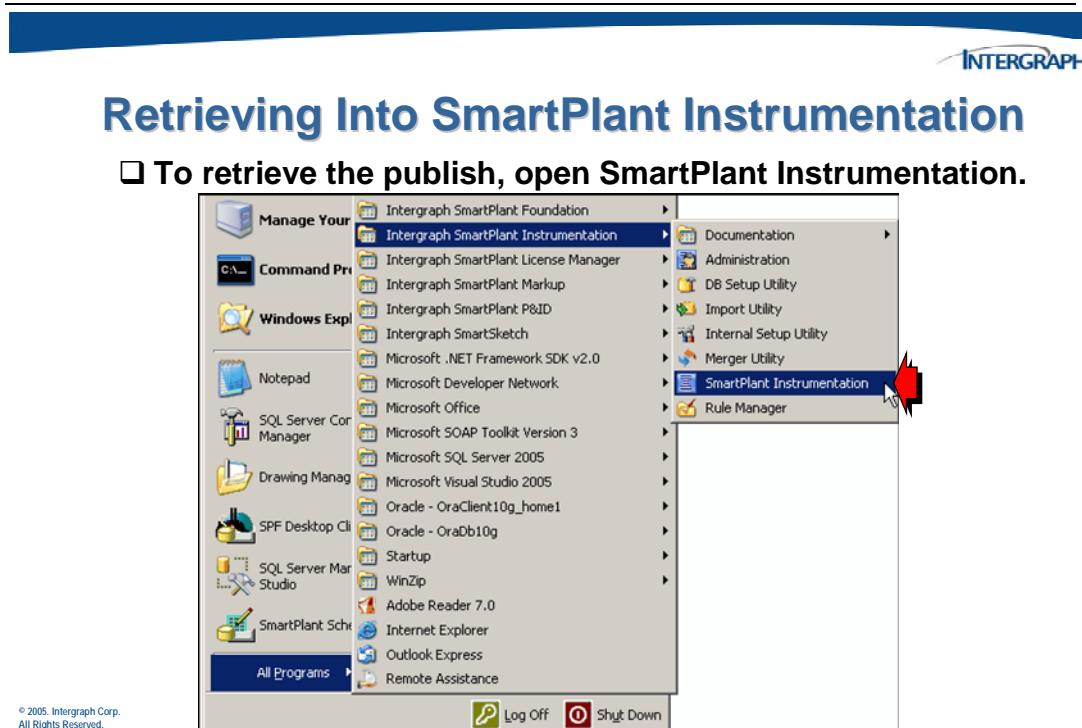


Now that the document has been successfully published, exit from SmartPlant P&ID. In the next section, this document will be retrieved into SmartPlant Instrumentation.

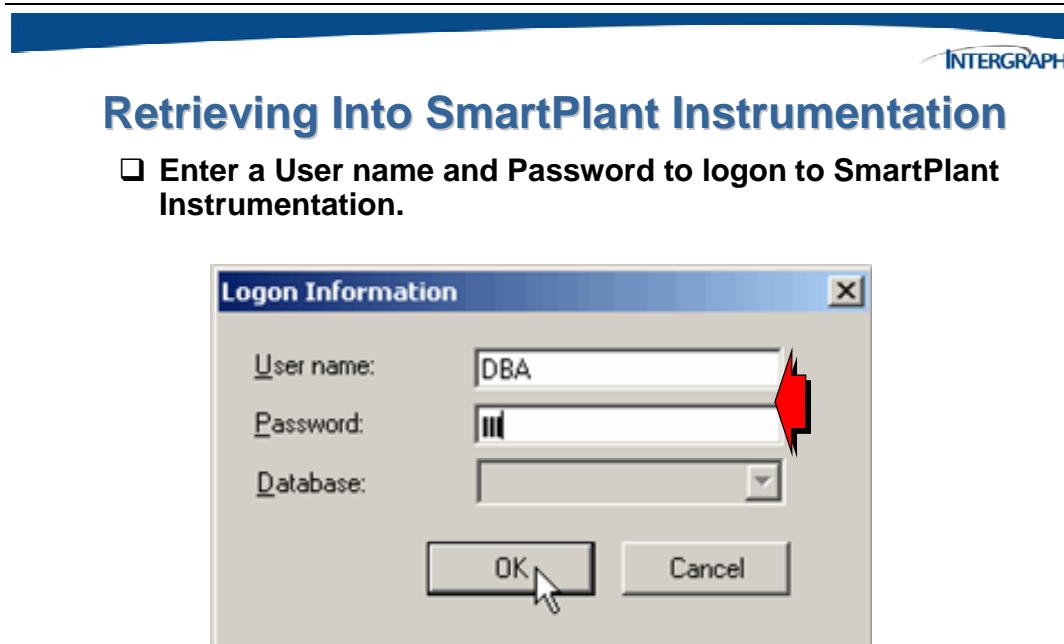


10.5.2 Retrieving a Change into SmartPlant Instrumentation

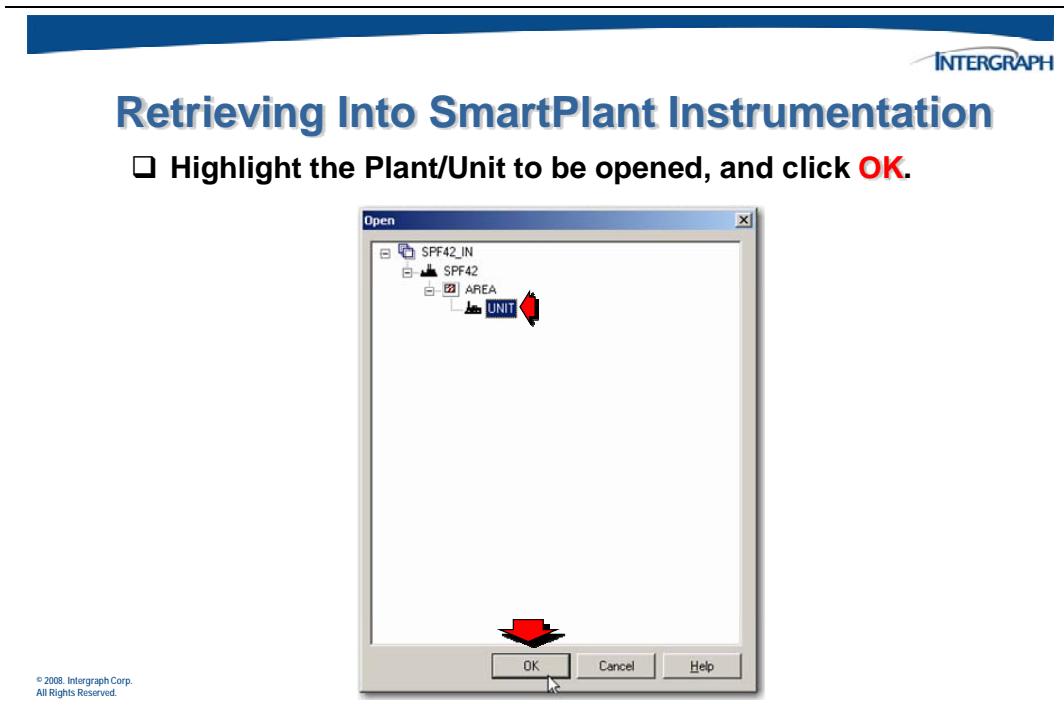
Now that the instrument change has been published into SmartPlant, the document can be retrieved by SmartPlant Instrumentation.



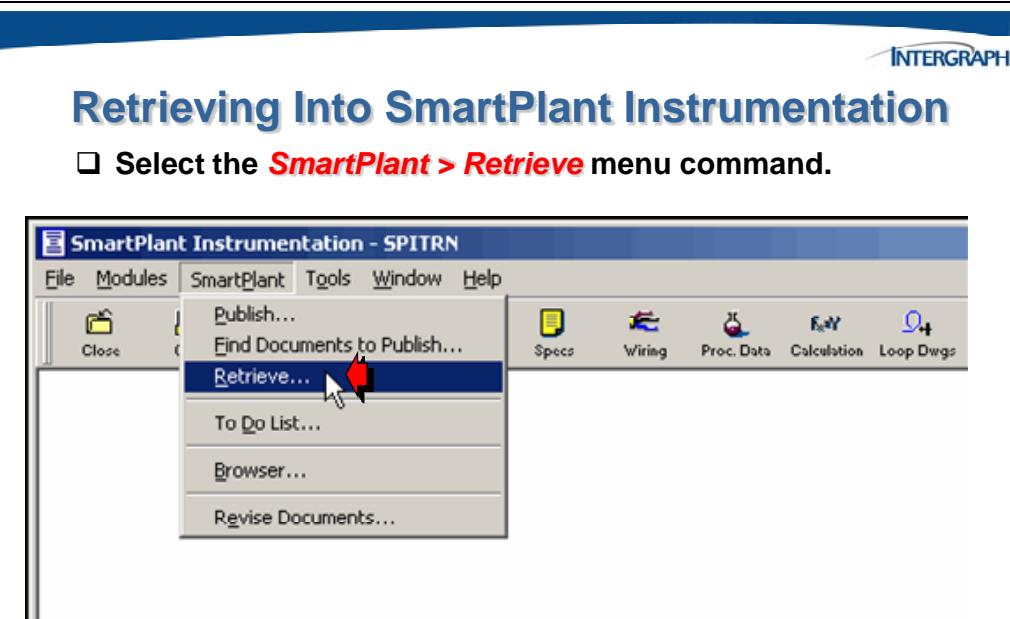
The login dialog box will appear.



When the *Open* window appears, choose a plant/unit from the displayed tree.



To perform a retrieve operation, use the **Retrieve** command from SmartPlant Instrumentation.

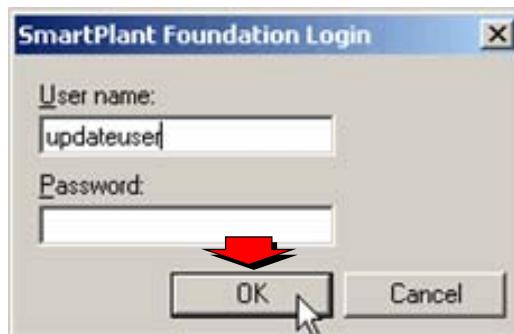


© 2008, Intergraph Corp.
All Rights Reserved.

The SmartPlant Foundation login dialog will appear in some cases, if the system user does not have an account in SmartPlant Foundation.

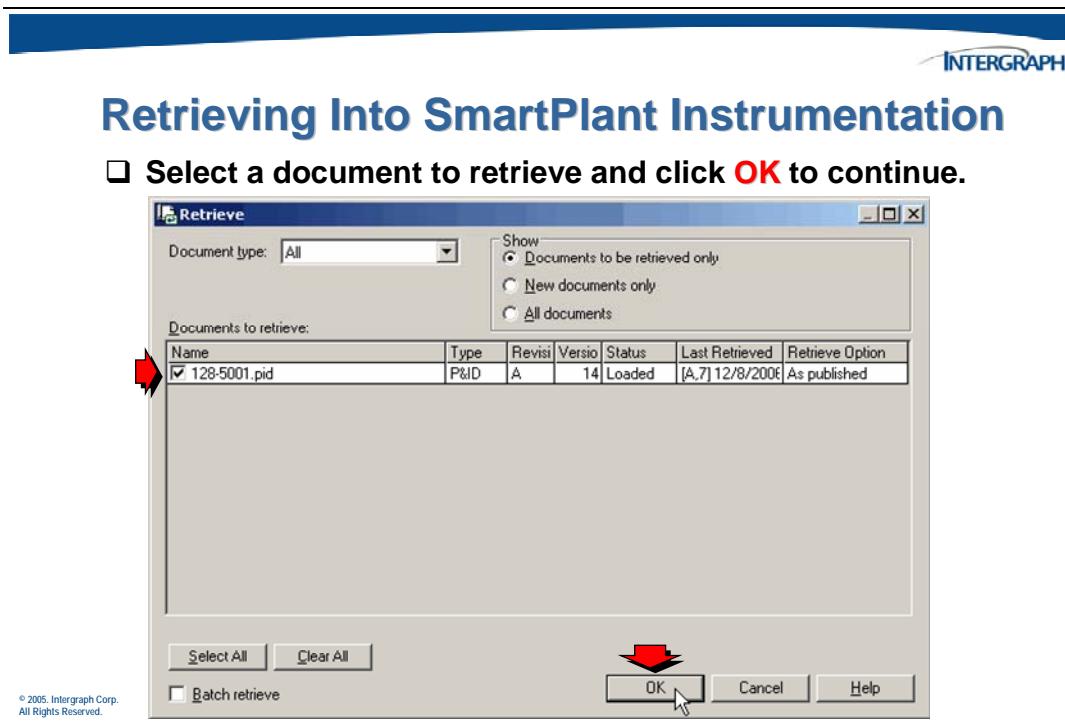
Retrieving Into SmartPlant Instrumentation

- ❑ Enter a valid SPF user name and password and click **OK** to have the tool log in and retrieve documents.

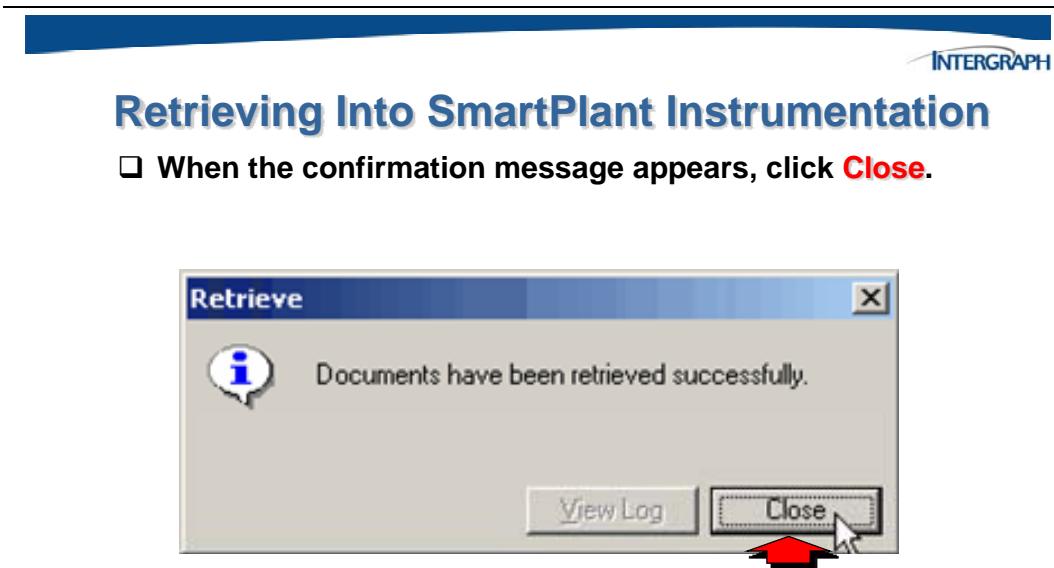


© 2005, Intergraph Corp.
All Rights Reserved.

When the retrieve dialog appears, enable the toggle box next to the previously published documents (from SPPID) to be retrieved by SmartPlant Instrumentation.



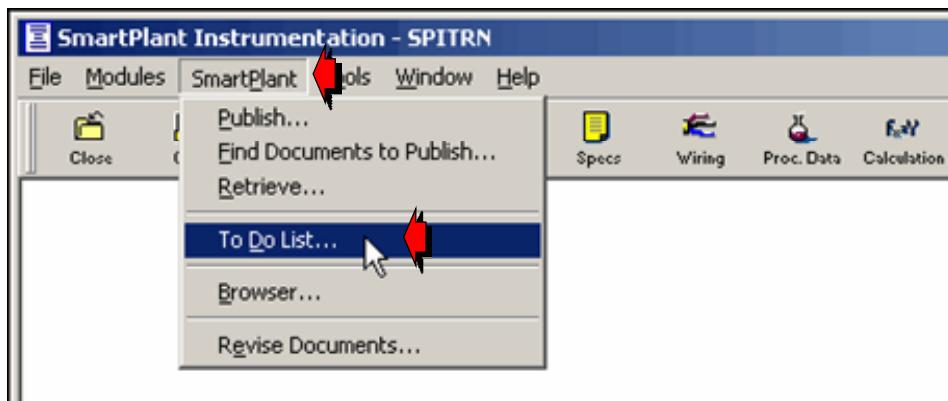
A dialog box will appear after some time to report success or failure of the publish operation.



The instrument changes are not automatically applied by SmartPlant Instrumentation. You must display the **To Do List**, which contains a list of all the changes needed to incorporate the information from the publish operation into the SPI database.

Retrieving Into SmartPlant Instrumentation

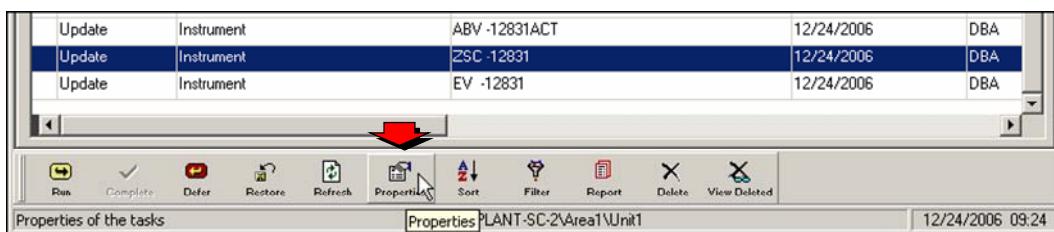
- From the menu, select the **SmartPlant > To Do List**.



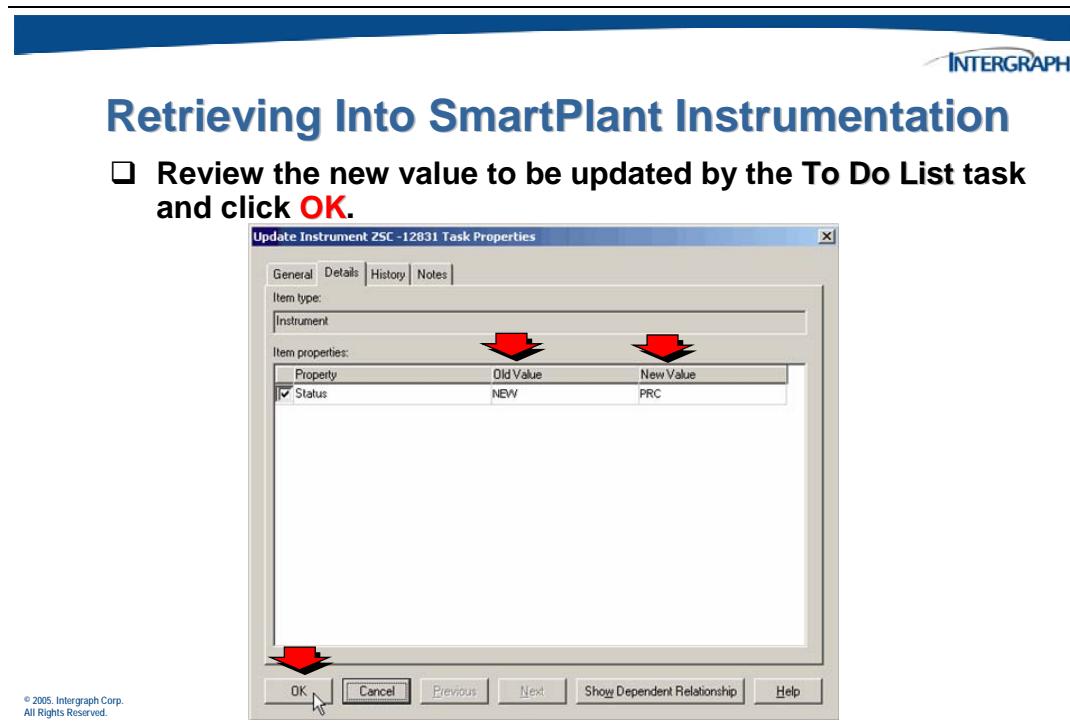
To see specifics about an item in the **To Do List**, select it, and click the **Properties** button on the bottom toolbar.

Retrieving Into SmartPlant Instrumentation

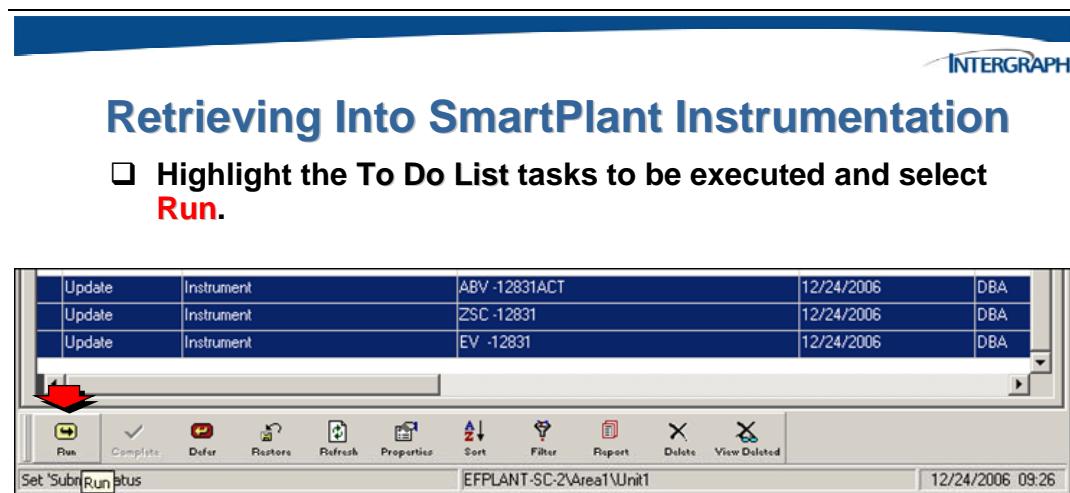
- Highlight one of the tasks to be executed and select **Properties**.



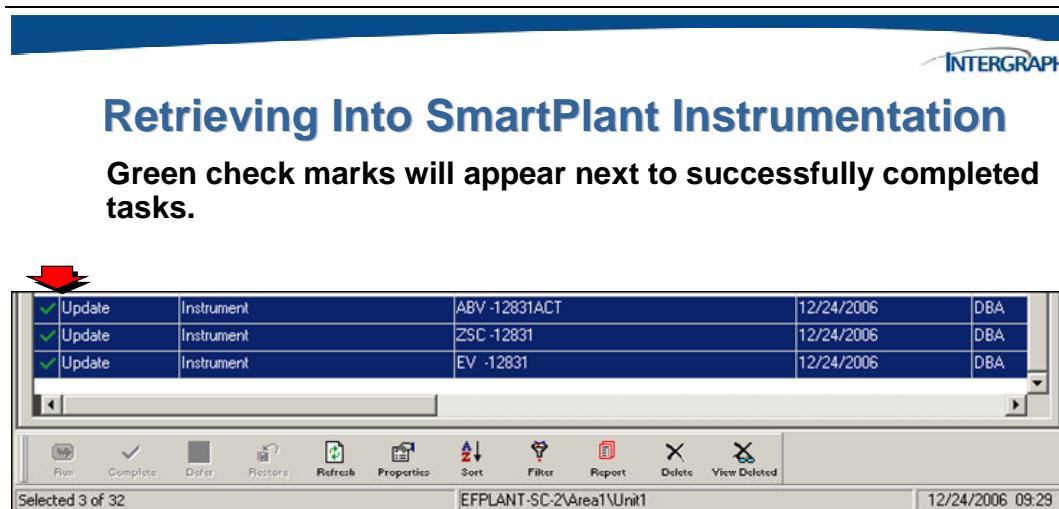
This will allow you to review the new value before it is applied to the existing data.



To apply changes to the database, select the changes to be made, and click the **Run** command.



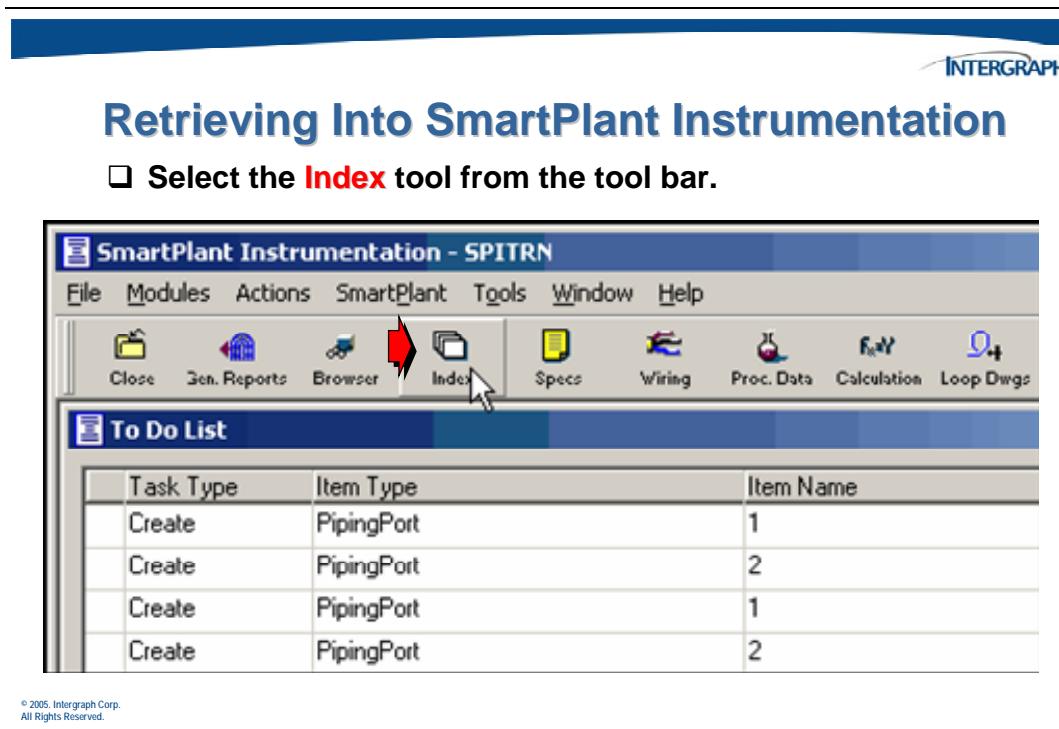
A status icon will be displayed in the left most column as tasks are completed.



The screenshot shows a software window titled "Retrieving Into SmartPlant Instrumentation". Below the title, a message states "Green check marks will appear next to successfully completed tasks." A red arrow points to the first row of a table. The table has columns for Task Type (Update), Item Type (Instrument), Item Name (ABV-12831ACT, ZSC-12831, EV-12831), Date (12/24/2006), and User (DBA). The first three rows have green checkmarks in the Task Type column. At the bottom of the table, it says "Selected 3 of 32". The toolbar below the table includes icons for Print, Complete, Defer, Restore, Refresh, Properties, Sort, Filter, Report, Delete, and View Deleted. The status bar at the bottom right shows the date and time: 12/24/2006 09:29.

© 2005. Intergraph Corp.
All Rights Reserved.

Once the tasks have been executed, use one of the browser windows to view the results.

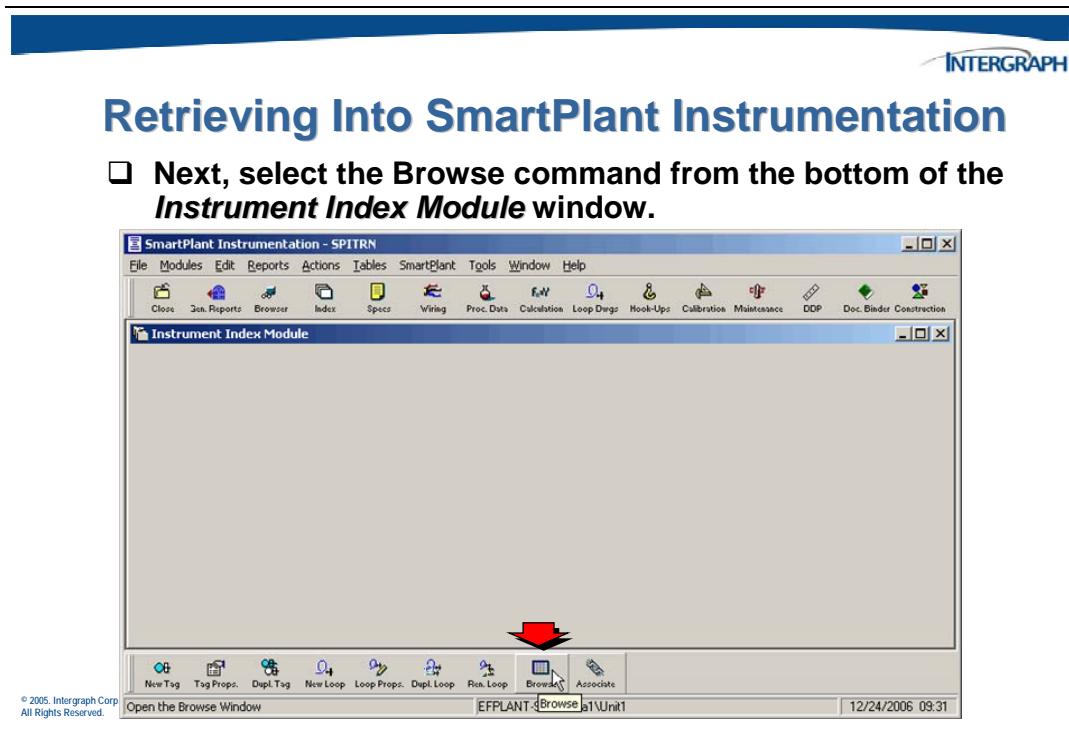


The screenshot shows the "SmartPlant Instrumentation - SPITRN" application window. The menu bar includes File, Modules, Actions, SmartPlant, Tools, Window, and Help. The toolbar below the menu bar has icons for Close, Gen. Reports, Browser, and Index (which is highlighted with a red arrow). The main window is titled "To Do List" and contains a table with four rows:

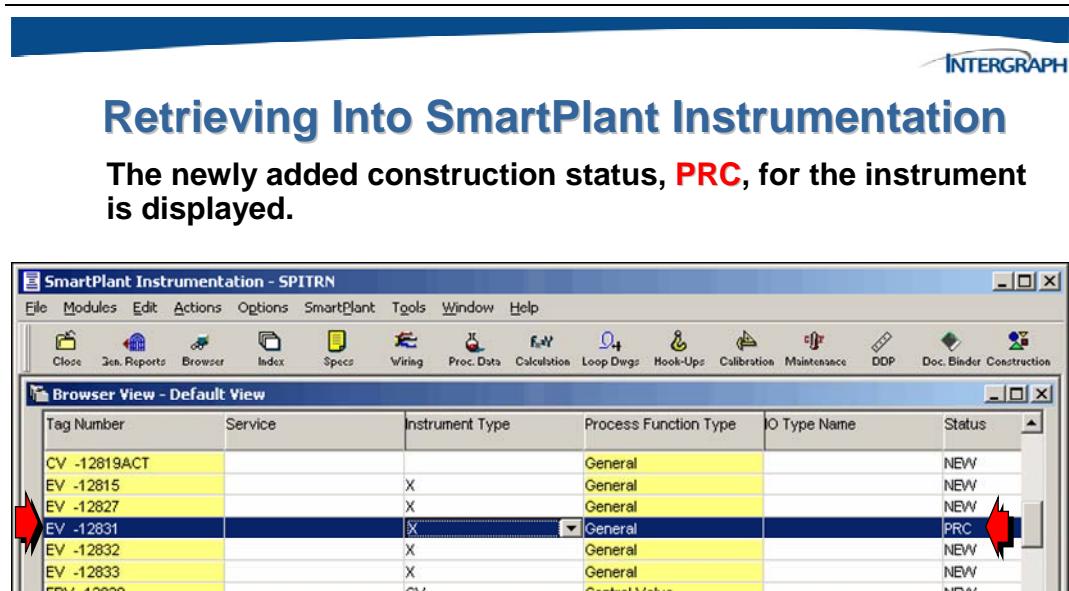
Task Type	Item Type	Item Name
Create	PipingPort	1
Create	PipingPort	2
Create	PipingPort	1
Create	PipingPort	2

At the bottom left of the main window, there is a copyright notice: "© 2005. Intergraph Corp.
All Rights Reserved."

The *Instrument Index Module* window will appear.



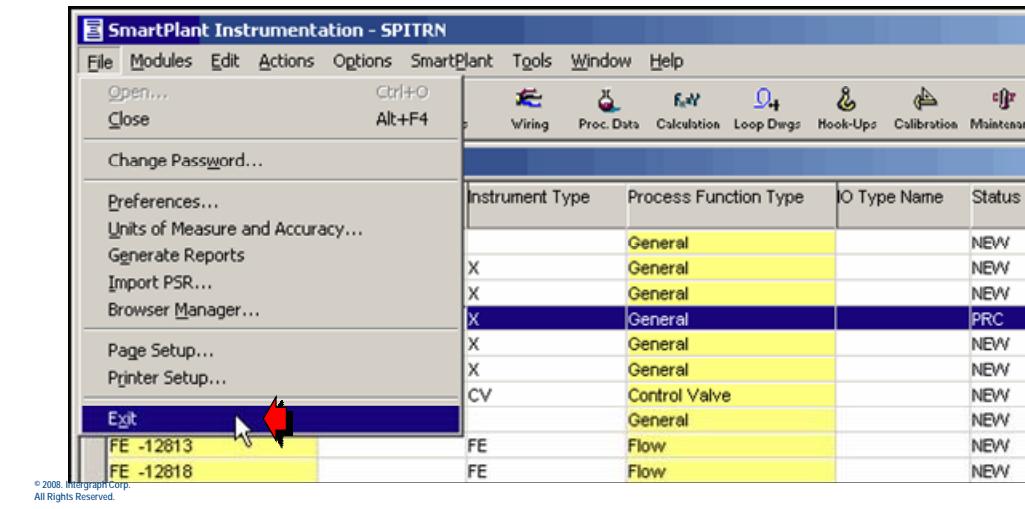
A *Browser View* window with instrument information will be displayed.



Now that the document has been successfully retrieved and the selected tasks executed, exit from SmartPlant Instrumentation.

Retrieving Into SmartPlant Instrumentation

- From the menu, select **File > Exit**.



10.6 Activity 1 – Extending an Existing Enumerated List

Complete the Chapter 10 – Activity 1 in the SmartPlant Foundation 2008 (4.2) Modeling and Mapping Activity workbook.

10.7 Creating and Mapping a Custom Property in SmartPlant Instrumentation

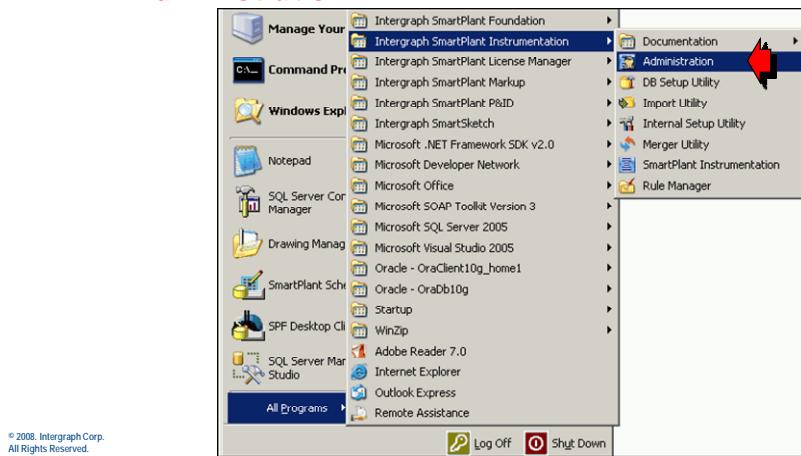
In this section, the process of adding a new custom property to SmartPlant Instrumentation (SPI) and then mapping that property to the SmartPlant schema will be discussed. The type of property that will be added is a simple or string property, as it is not possible to add a new custom enumerated list to SPI.

10.7.1 Adding a Custom Property to SPI

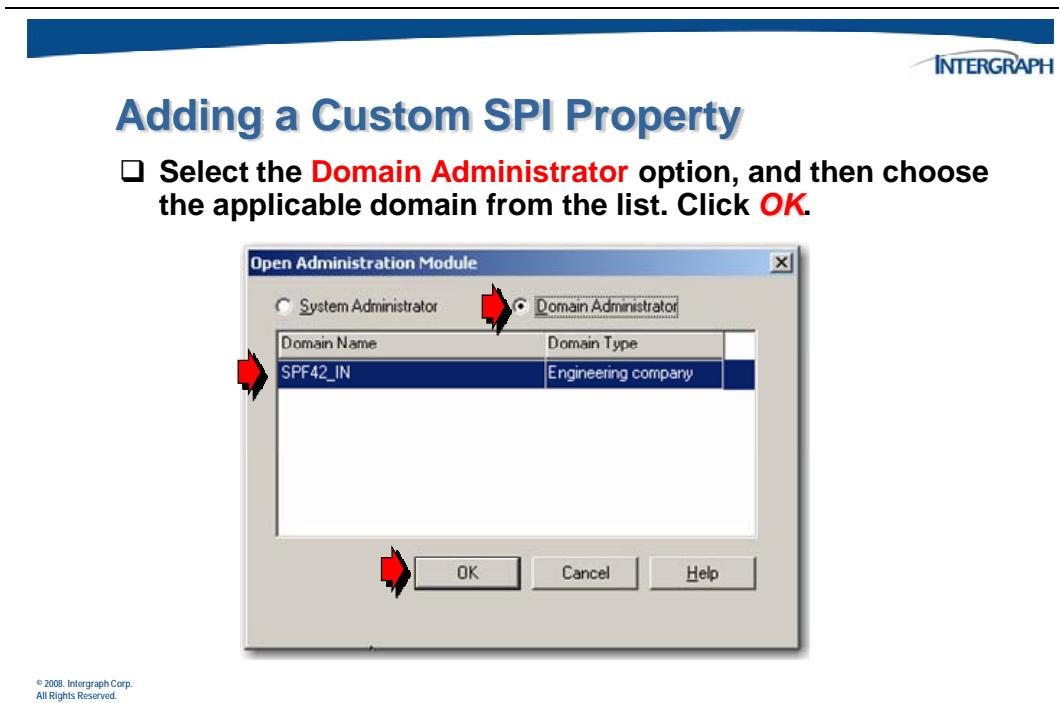
The first step is to use the SPI Administration module to add the new property, called a UDF (user defined field).

Adding a Custom SPI Property

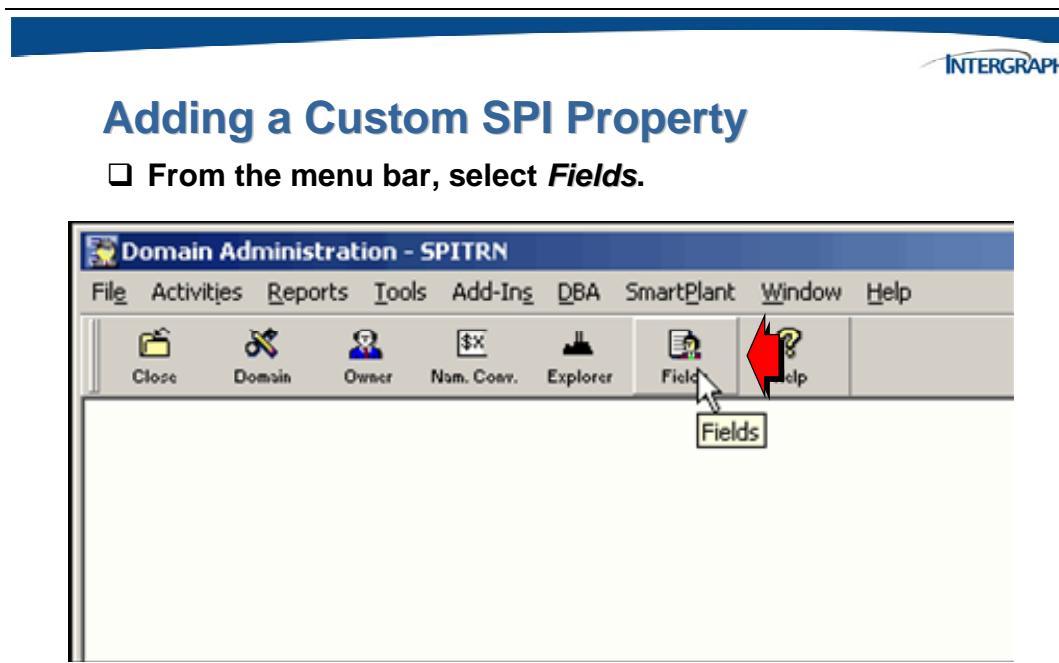
- From the menu, open SPI Administration by clicking **Start > All Programs > Intergraph SmartPlant Instrumentation > Administration**.



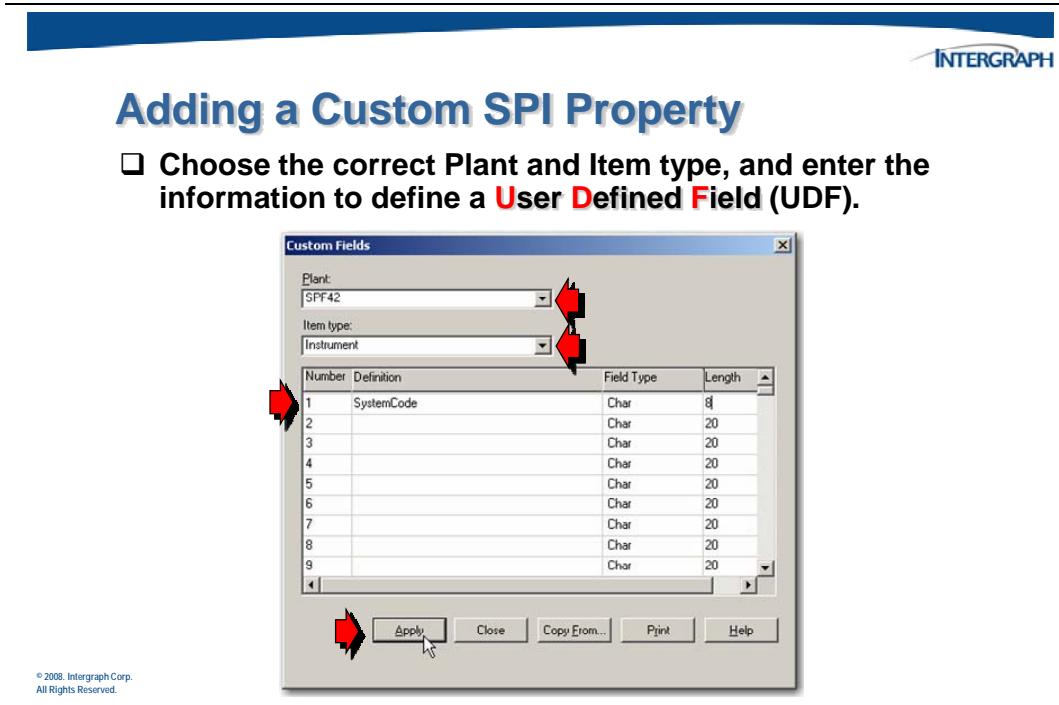
The *Open Administration Module* dialog will appear. Select the radio button to access the tool as a **Domain Administrator** and choose the access the **SPF42_IN** domain.



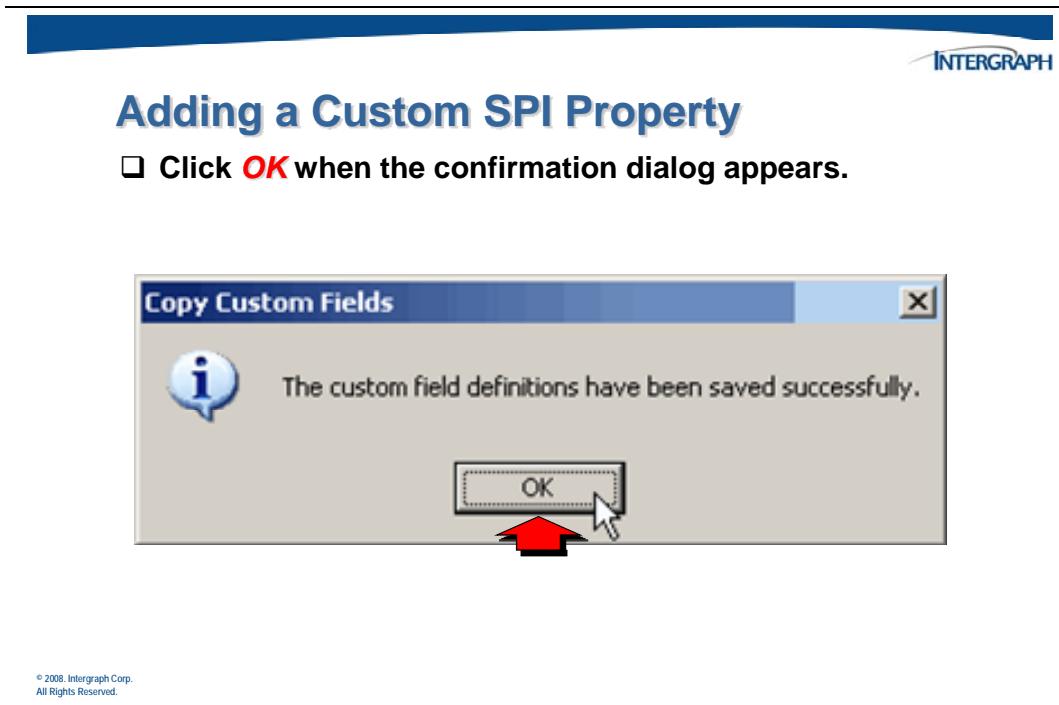
Use the **Fields** button to display the *Custom Fields* dialog where new UDF's can be added.



In the following example, the property *SystemCode* that was added to SPPID will be added as an SPI UDF for the **Instrument** class.



A confirmation dialog will appear.

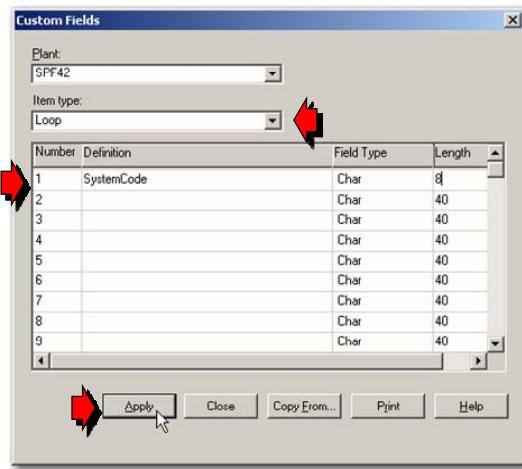


Repeat the process to add the *SystemCode* UDF to the Instrument **Loop** class.



Adding a Custom SPI Property

- ❑ Again, enter the information to define another **UDF** but for a different *Item type*, and click **Apply**.

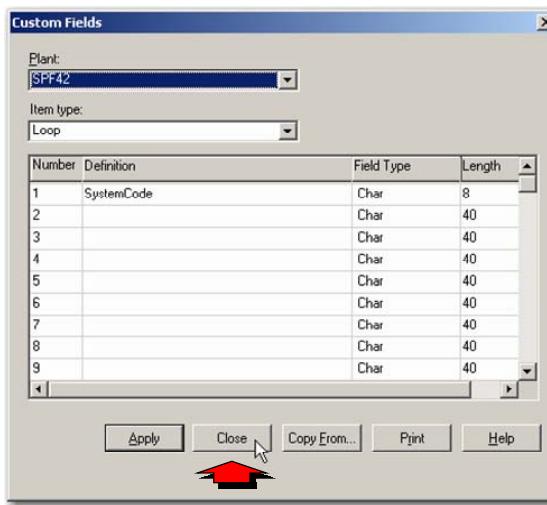


A confirmation dialog will once again appear.

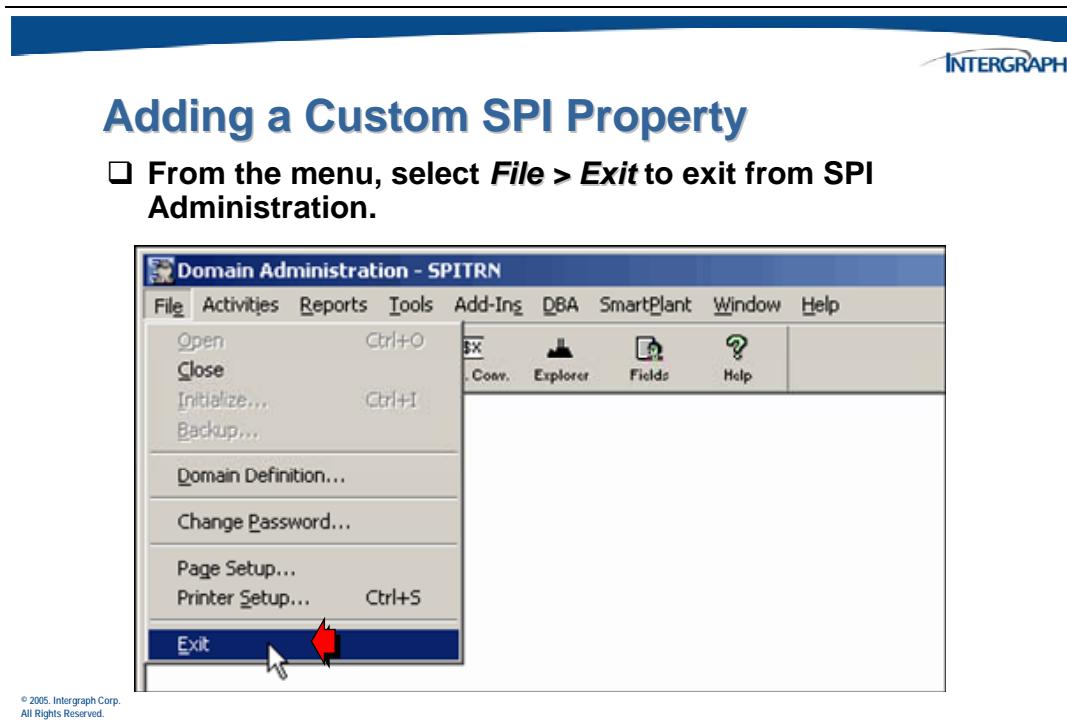


Adding a Custom SPI Property

- ❑ Once the UDFs have been added, click **Close**.

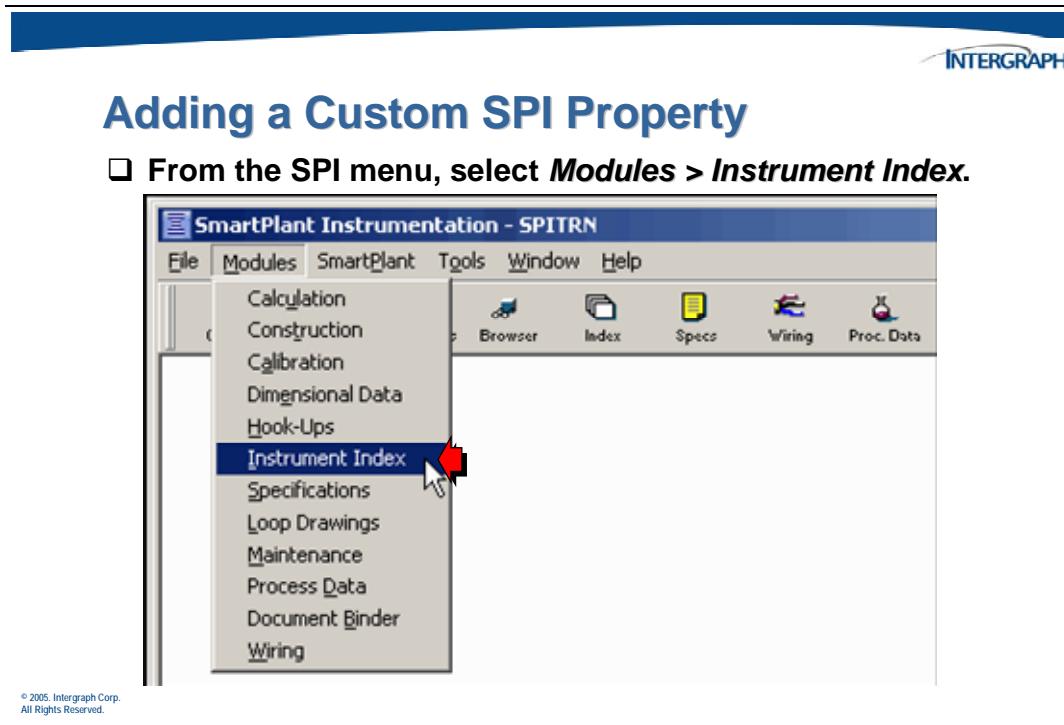


After the *SystemCode* UDF has been added, **exit** from the Administration module.

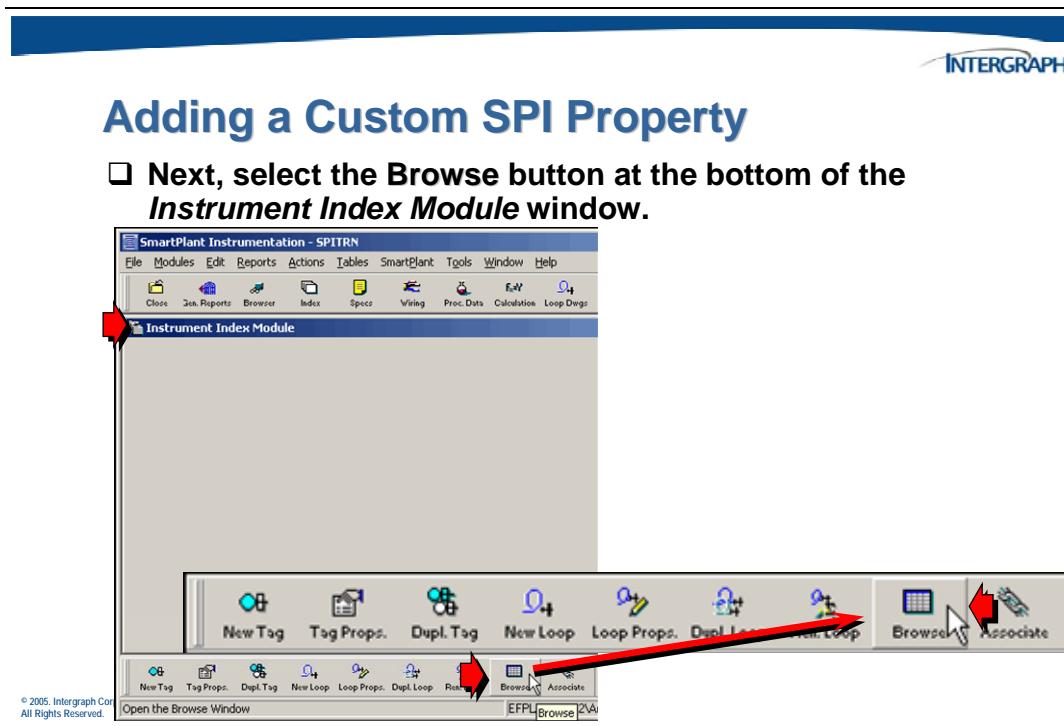


10.7.2 Adding the Custom Field to the Instrument Index Browser

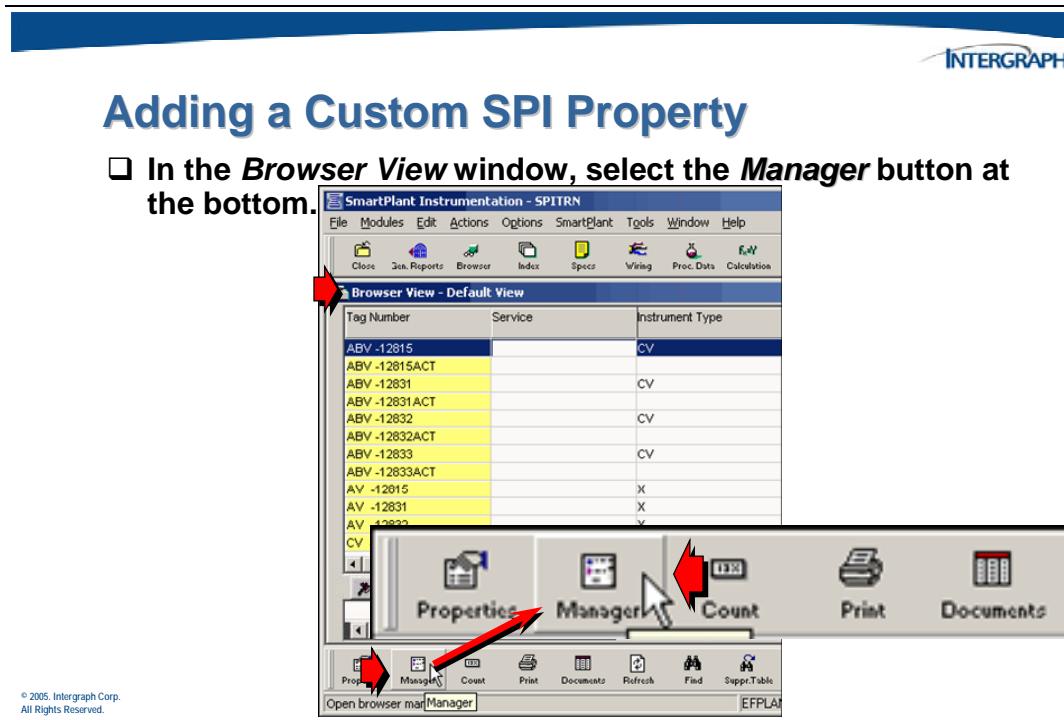
Next, start SmartPlant Instrumentation and configure the new field to be displayed in the *Instrument Index* browser.



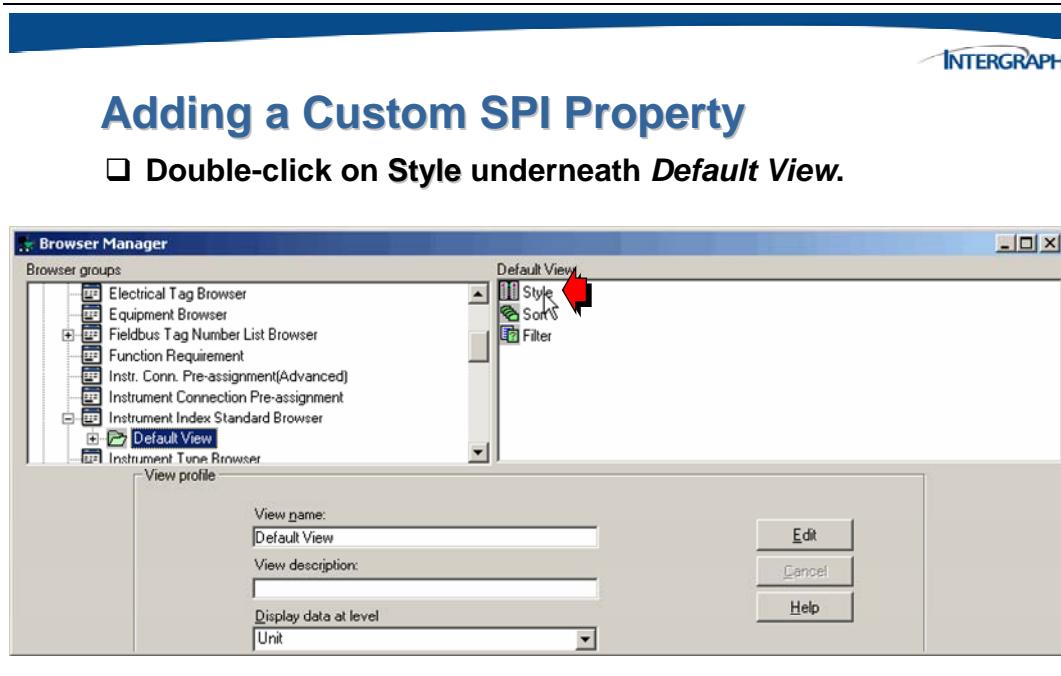
Display the *Instrument Index* browser in order to add the new UDF.



When the browser view appears, use the **Manager** button to change the default configuration.

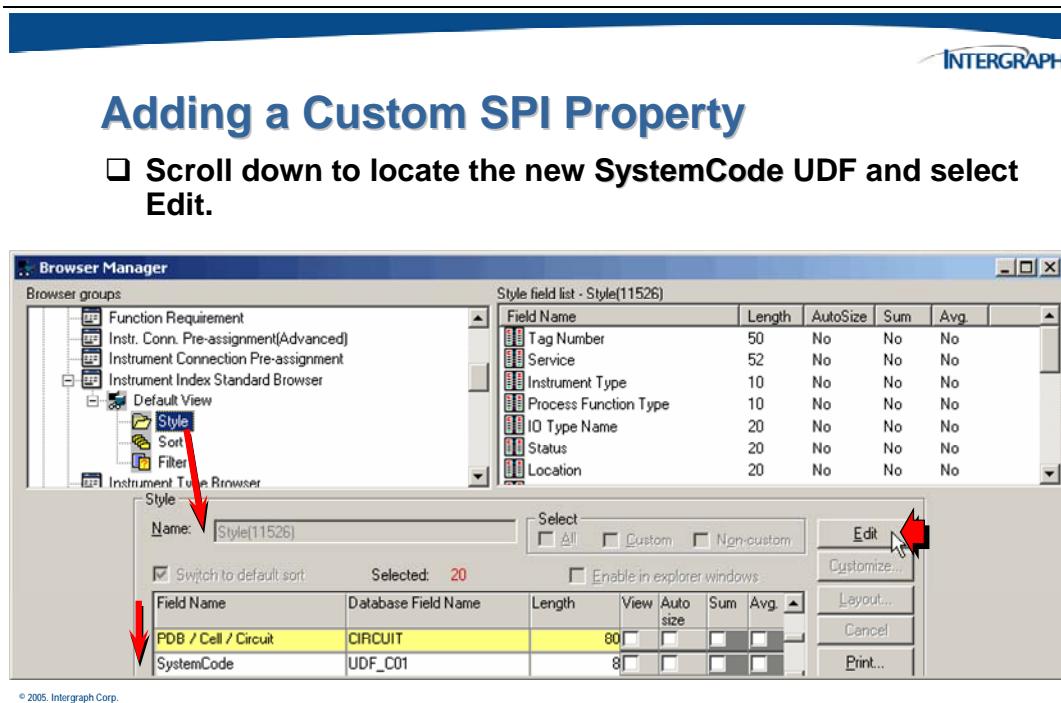


The **Browser Manager** window will appear. Expand **the Instrument Index Standard Browser**, and select **Default View**. From the **Default View** window, select **Style**.



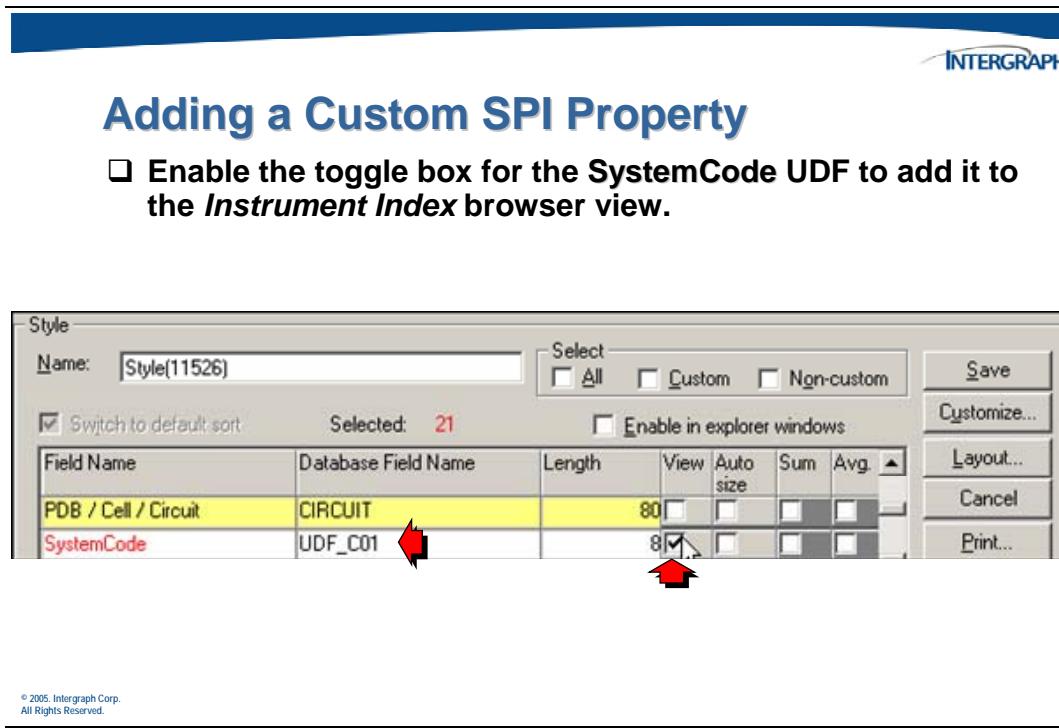
© 2005, Intergraph Corp.
All Rights Reserved.

The available fields for this view will appear in the bottom portion of the window. Scroll down to find the new **System Code** UDF, and click the **Edit** button.

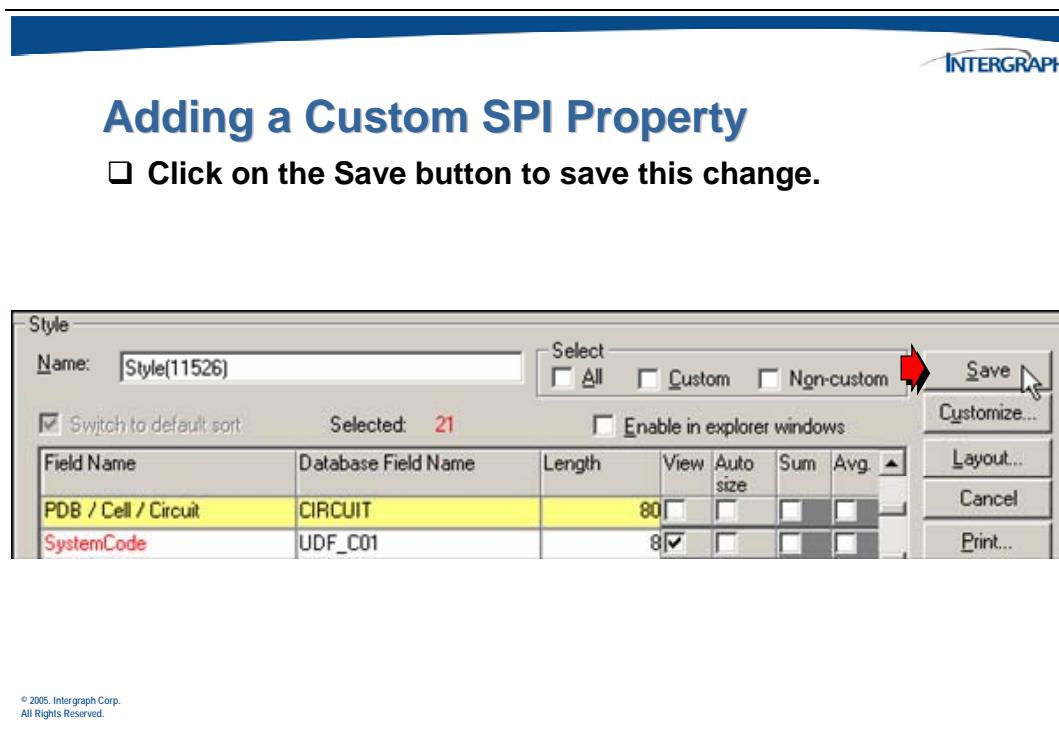


© 2005, Intergraph Corp.
All Rights Reserved.

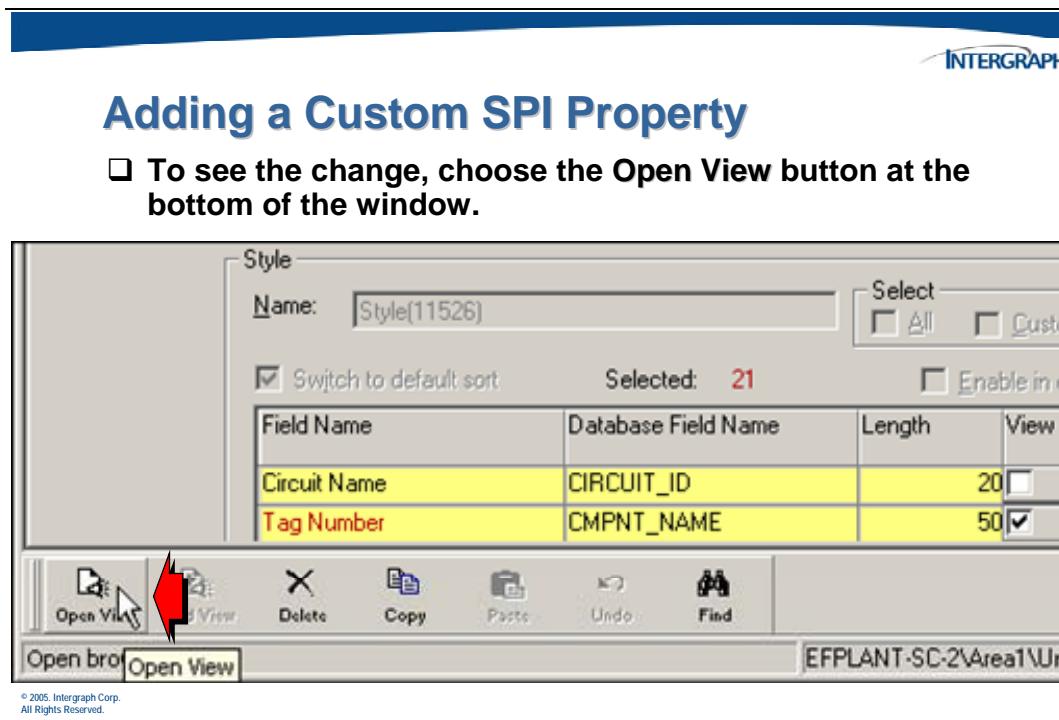
Activate the ***View*** option for the System Code property to indicate that it should be displayed in the Instrument Index Browser view.



Once the UDF has been added to the view, save the change.



Once the change has been made and saved, display the browser view and verify that the new UDF has been configured correctly.

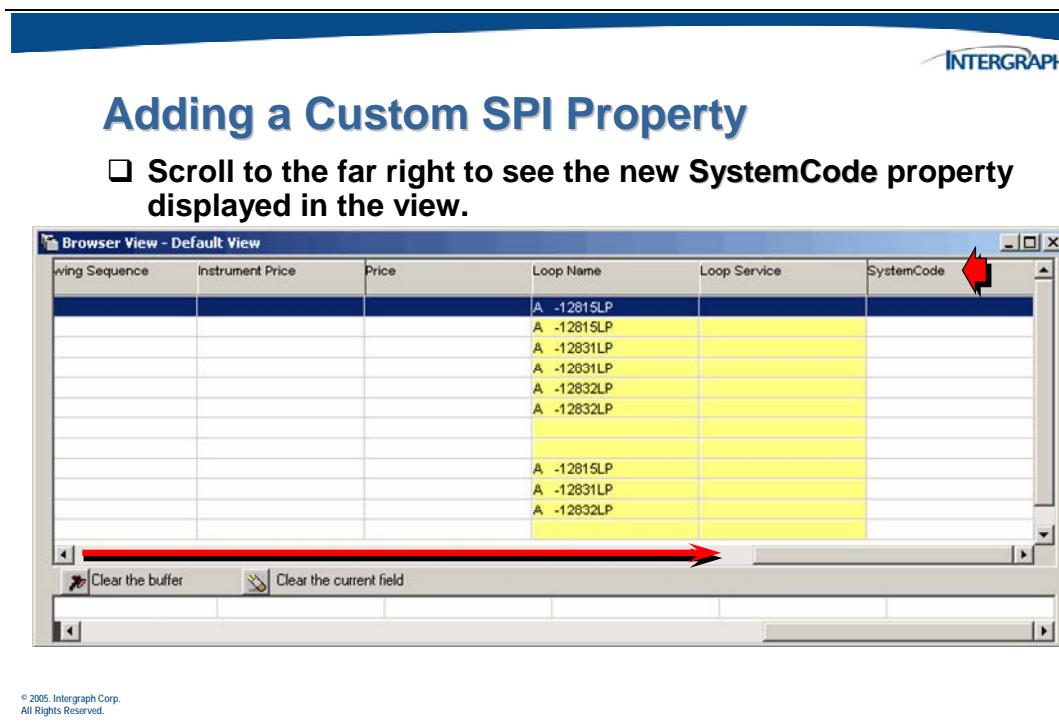


The screenshot shows the 'Style' configuration dialog box. In the 'Name' field, 'Style(11526)' is entered. Under the 'Selected' section, the value '21' is shown. A red arrow points to the 'Open View' button at the bottom left of the dialog. The 'View' tab is selected, displaying a table with two rows:

Field Name	Database Field Name	Length	View
Circuit Name	CIRCUIT_ID	20	<input type="checkbox"/>
Tag Number	CMPNT_NAME	50	<input checked="" type="checkbox"/>

The status bar at the bottom indicates 'EFPLANT-SC-2\Area1\Un'. A copyright notice at the bottom left reads: © 2005, Intergraph Corp. All Rights Reserved.

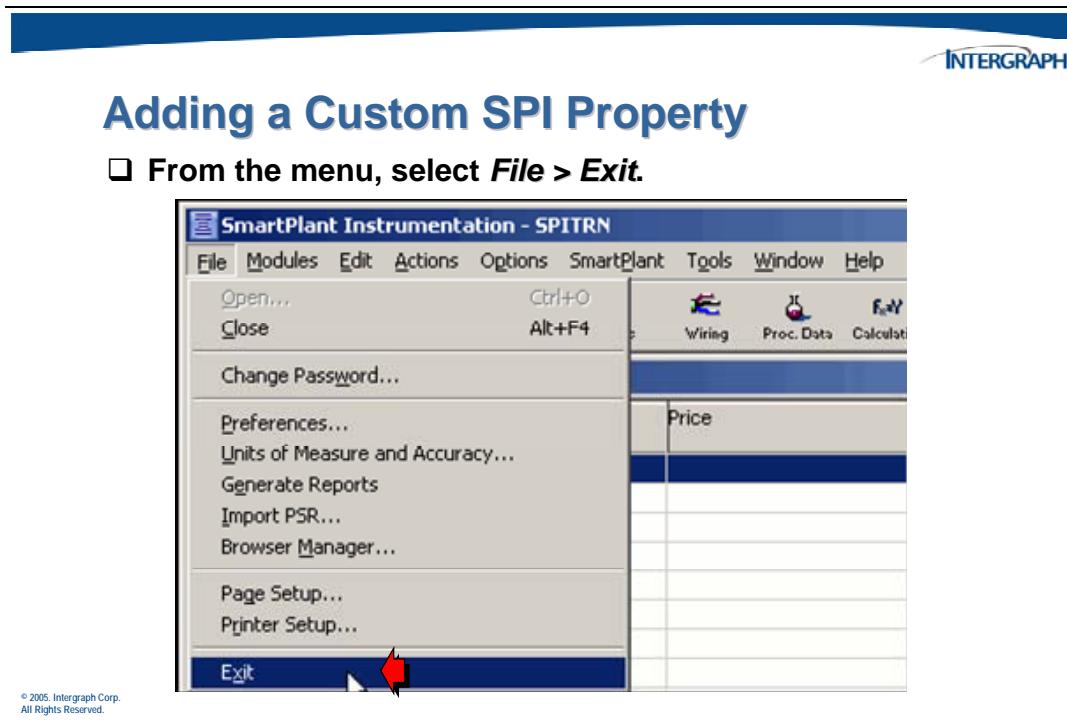
The *Instrument Index Browser View* will now include the new property.



The screenshot shows the 'Browser View - Default View' window. A red arrow points to the right edge of the view, indicating where the new 'SystemCode' column is located. The table has columns: Wing Sequence, Instrument Price, Price, Loop Name, Loop Service, and SystemCode. The 'SystemCode' column contains values like 'A -12815LP', 'A -12831LP', etc. At the bottom of the window, there are buttons for 'Clear the buffer' and 'Clear the current field'.

A copyright notice at the bottom left reads: © 2005, Intergraph Corp. All Rights Reserved.

You are now ready to exit SmartPlant Instrumentation and configure the mapping.



10.7.3 Mapping a Custom Field Value in SPI

Before defining mapping for the new UDF in the tool map schema, the new SystemCode property must be added to the tool map schema. Remember, this is done automatically when we synchronize the tool map file and the tool meta schema using the tool metadata adapter.

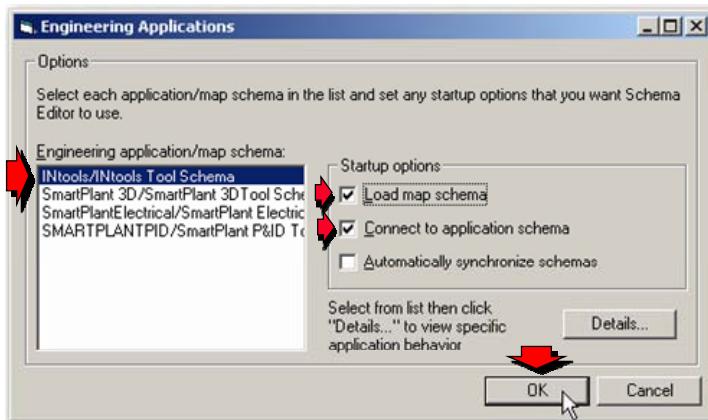
Note:

- Before starting the mapping process, you need to have access to the CMF file. Since we are not making changes to the CMF itself, that file does not have to be checked out of the Desktop Client, but you must have access to the SmartPlant Schema in order to create the mapping relationships.

In the following illustrations, we have already connected to the CMF file using an existing session file, and the next step is to connect to the SmartPlant Instrumentation tool map file and tool meta schema.

Map the Property in the Tool Schema

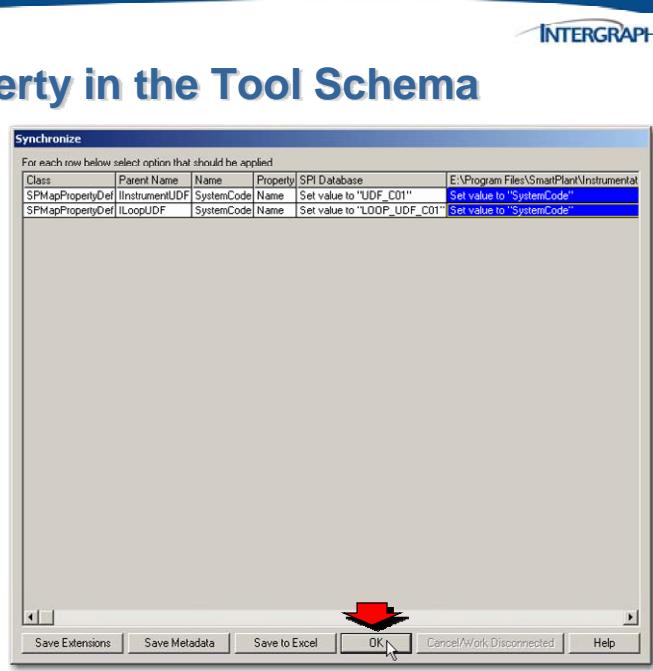
- Launch the session file in the schema editor and open the list of metadata adapters. Choose **INtools**, and click **OK**.



The **Synchronize** screen lists the differences between the two schemas.

Map the Property in the Tool Schema

- Click **OK** from the **Synchronize** dialog to have the schema editor read the tool meta-schema and automatically add the new property to the tool map schema.



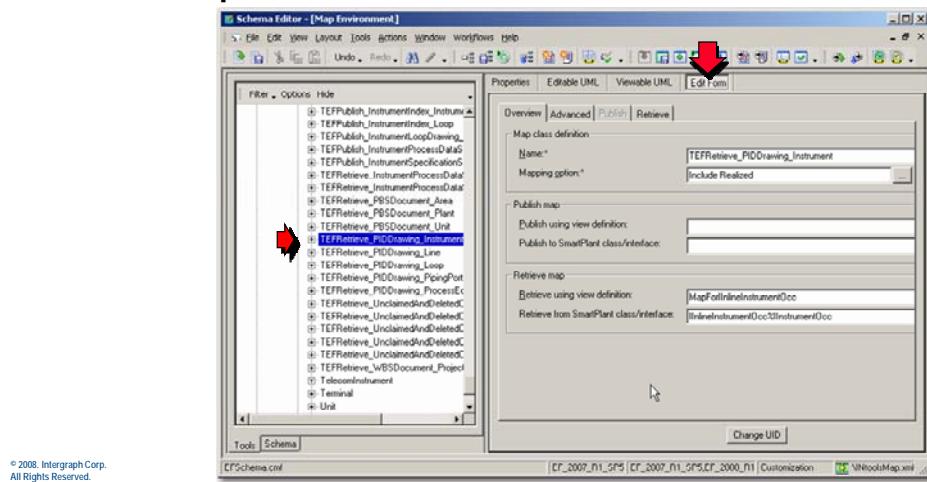
© 2008, Intergraph Corp.
All Rights Reserved.

In the **Map Environment**, find the Map Class **TEFRetrieve_PIDDrawing_Instrument**, which represents instruments published on a P&ID drawing and retrieved into SPI.

Notice that this map class is already mapped to the IIInlineInstrumentOcc and IIInstrumentOcc interfaces, and remember that our ICustomInterface interface is already implied by the IIInstrumentOcc interface. Because of this we can already create the mapping necessary for us to publish and retrieve the status code.

Map the Property in the Tool Schema

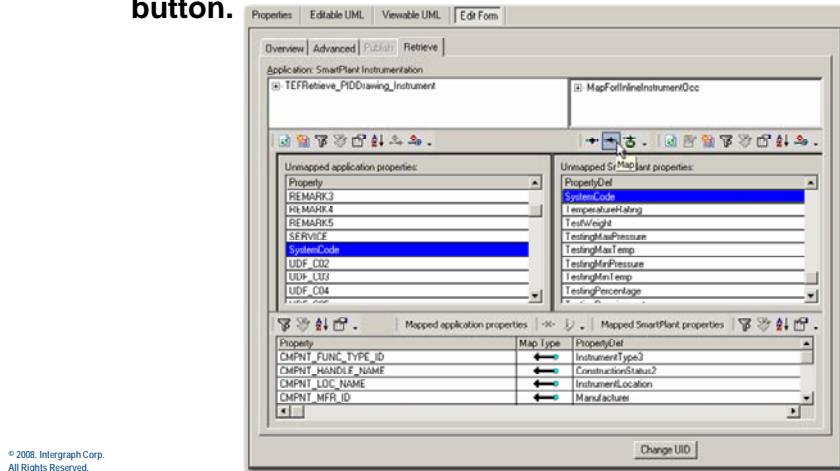
- Find the **TEFRetrieve_PIDDrawing_Instrument** map class, and open the *Edit Form*.



In the **Edit Form** view, open the **Retrieve** tab. Find the custom property in both the tool map file and the SmartPlant Schema, and click the **Map** button.

Map the Property in the Tool Schema

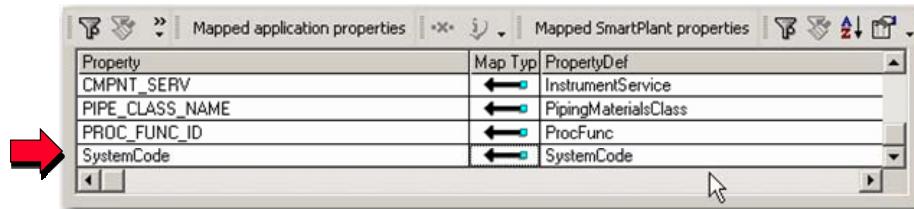
- Select the UDF property from the left pane and the SmartPlant property from the right pane, and click the **Map** button.



Confirm the retrieve mapping relationship in the bottom of the window.

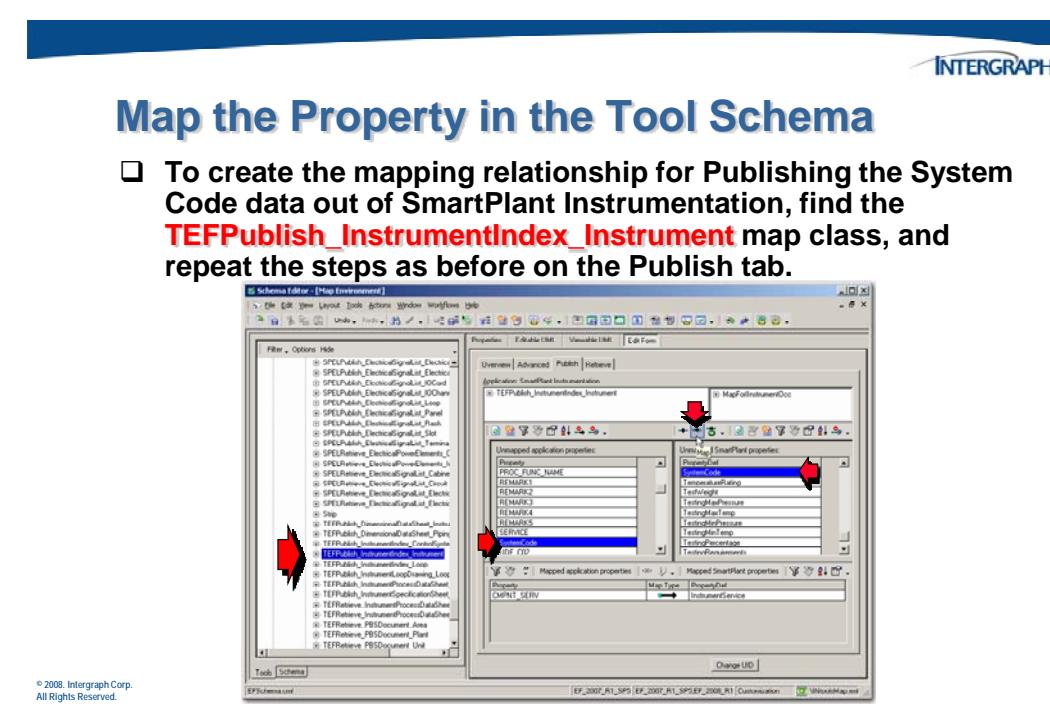
Map the Property in the Tool Schema

- Verify that the enum entries from the two schemas are mapped at the bottom of the dialog and click OK.



To create the mapping to publish the **SystemCode** property back out of SmartPlant Instrumentation, return to the list of map classes and find **TEFPublish_InstrumentIndex_Instrument**.

This time, from the **Edit Form** view, open the **Publish** tab. Find the **SystemCode** property in the tool map file and the SmartPlant schema, and use the **Map** button.



Confirm the publish mapping relationship in the bottom of the window.



Map the Property in the Tool Schema

- Verify that the enum entries from the two schemas are mapped at the bottom of the dialog and click OK.

A screenshot of a software dialog box titled "Map the Property in the Tool Schema". The dialog has two tabs at the top: "Mapped application properties" and "Mapped SmartPlant properties". The "Mapped application properties" tab is selected. Below the tabs is a table with three columns: "Property", "Map Type", and "PropertyDef". There are two rows in the table:

Property	Map Type	PropertyDef
CMPNT_SERV	→	InstrumentService
SystemCode	→	SystemCode

Even though we were able to create the mapping relationship because of the Implies relationships that already existed, we still have to make some modifications to the ICustomInterface and update the applicable component schemas if the SPI adapter will be able to process the integrated data.



Modifying the Custom Interface

- The custom interface needs to be realized by some additional class defs if we want to publish and retrieve the information in SPI.
- We want to publish/retrieve our **System Code** property for the following types of objects in SPI:
 - **Instruments**
 - **Instrument Loops**

Compare this list of objects to the SmartPlant Schema to determine which class defs they correspond with, and for each class def determining the primary interface. Actually, the Implies interface relationships already exist, but you must make sure that the involved SmartPlant schema class defs realize the ICustomInterface.



Modifying the Custom Interface

- For each of these types of objects, we must investigate to determine the corresponding SPF class def and the primary interface of each of these class defs.
- The following table indicates the results of this research:

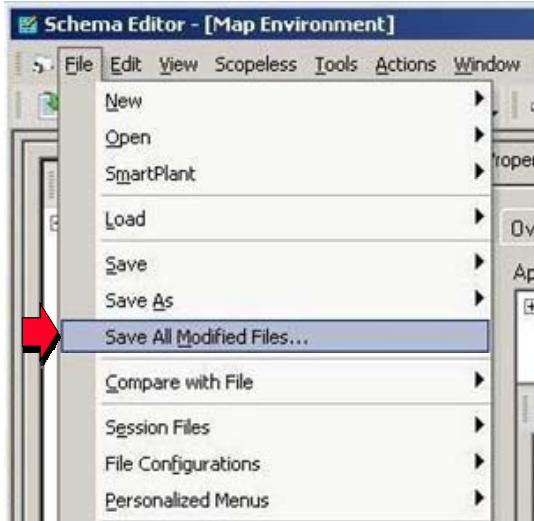
<u>Map Class</u>	<u>SPF Class Def</u>	<u>Primary Interfaces</u>
Instruments	INDXIInstruments	IInstrumentOcc
InstrumentLoop	INDXLoop	IInstrumentLoop

- Before saving the CMF file, make sure the ICustomInterface is realized by the above class defs and is implied by the above interfaces.

Once the custom interface is modified, save the changes to all modified files.

Saving Schema Changes

- ❑ On the menu, click **File > Save All Modified Files** to save the schema changes.



© 2008, Intergraph Corp.
All Rights Reserved.

Save the new property and the mapping relationships to the SPI map file.

Save Mapping Changes

- ❑ To confirm that you want to save the changes you have made to the **INtoolsMap.xml** file, click **Yes**.



© 2008, Intergraph Corp.
All Rights Reserved.

When prompted, save the changes to the CMF file.



Save Mapping Changes

- Verify the CMF file for the property addition and choose Yes.



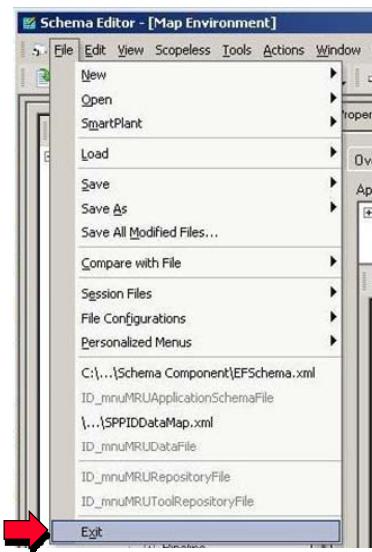
© 2008, Intergraph Corp.
All Rights Reserved.

Once your changes have been saved, close the Schema Editor.



Save Mapping Changes

- On the menu, click **File > Exit** to close out of the schema editor.



10.8 Test the Complex Property Mapping for SPI

Now, after we have added our custom property to SPI, we can test the mapping to get that information into and out of that tool. Since we have already published values for the System Code property for instruments in the P&ID drawing, we do not have to re-publish that document.

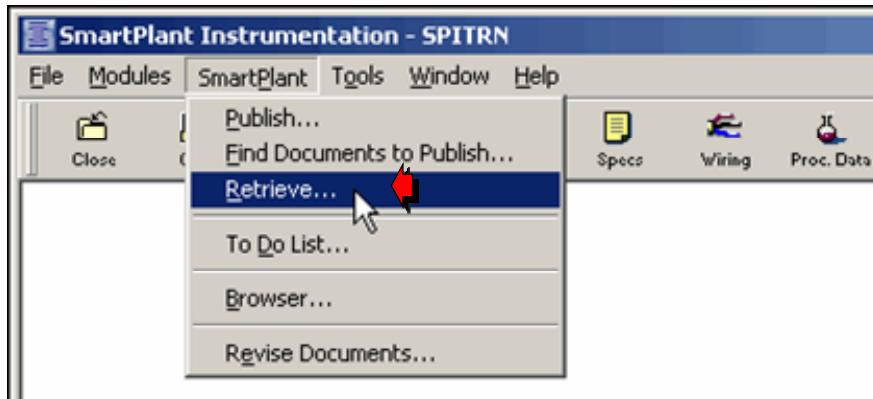
However, we do need to repeat the procedure to retrieve that document into SPI. When we retrieved the P&ID drawing in the previous example, SmartPlant Instrumentation was not configured to retrieve or process the System Code property. Since the mapping now exists for System Code, we need to retrieve the drawing again.

10.8.1 Retrieve the P&ID Drawing

From the SmartPlant Instrumentation application, use the **SmartPlant > Retrieve** command.

Retrieving Into SmartPlant Instrumentation

- From within SmartPlant Instrumentation, click the **SmartPlant > Retrieve** menu command.

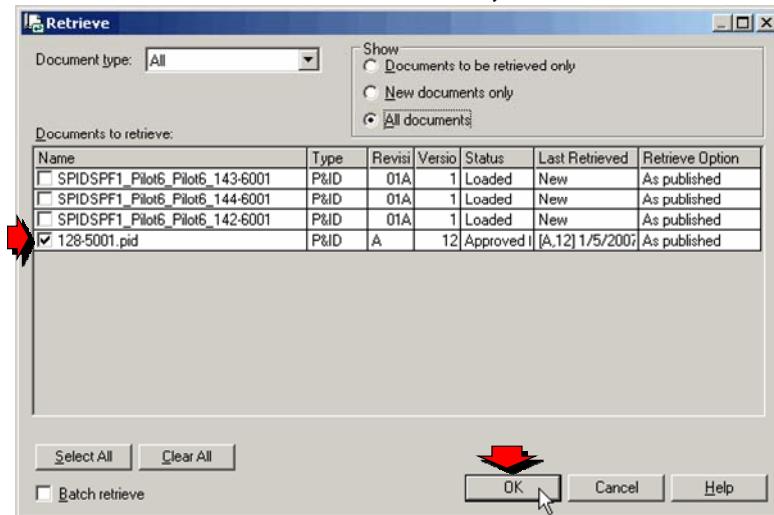


© 2008, Intergraph Corp.
All Rights Reserved.

On the **Retrieve** dialog box, find the P&ID drawing and activate its check box. You may have to select the **All documents** radio button in the **Show** section of this dialog to see the P&ID drawing in the list.

Retrieving Into SmartPlant Instrumentation

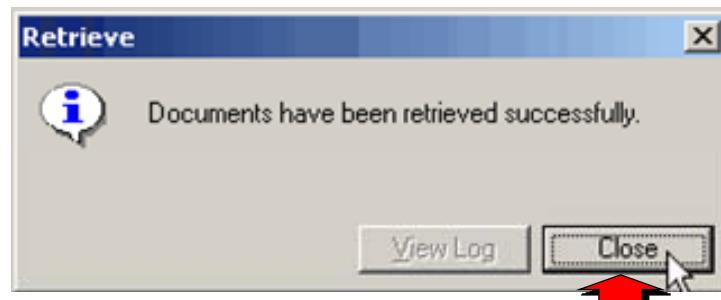
- Choose the document to retrieve, and click **OK** to continue.



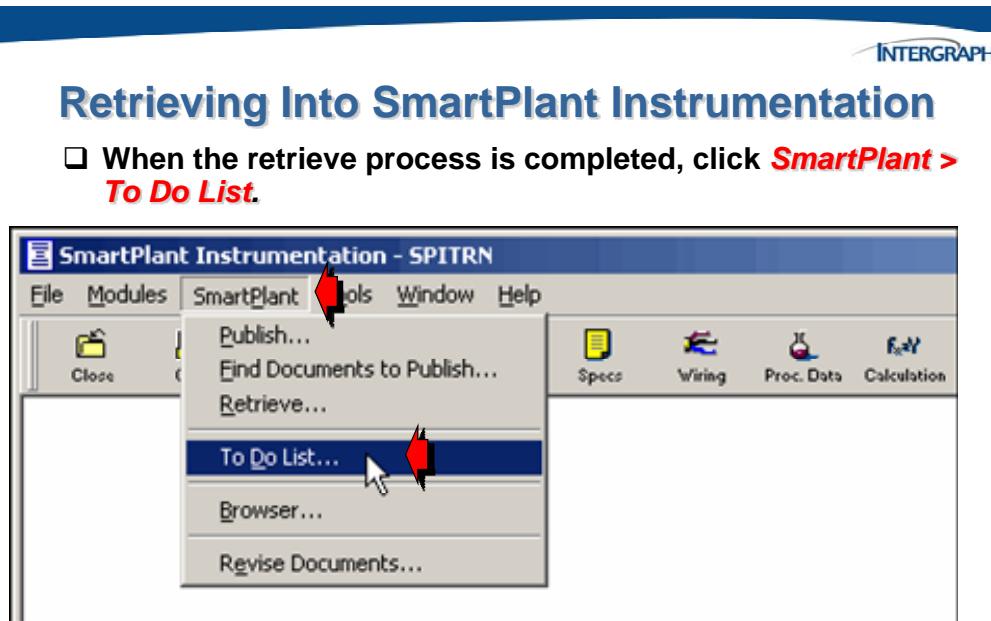
Dismiss the confirmation box when the retrieve operation is complete.

Retrieving Into SmartPlant Instrumentation

- When the confirmation dialog box appears, click **Close**.



Open the *To Do List* using the *SmartPlant > To Do List* command.



Review the list of updates and find the instrument for which you published a *SystemCode* value.

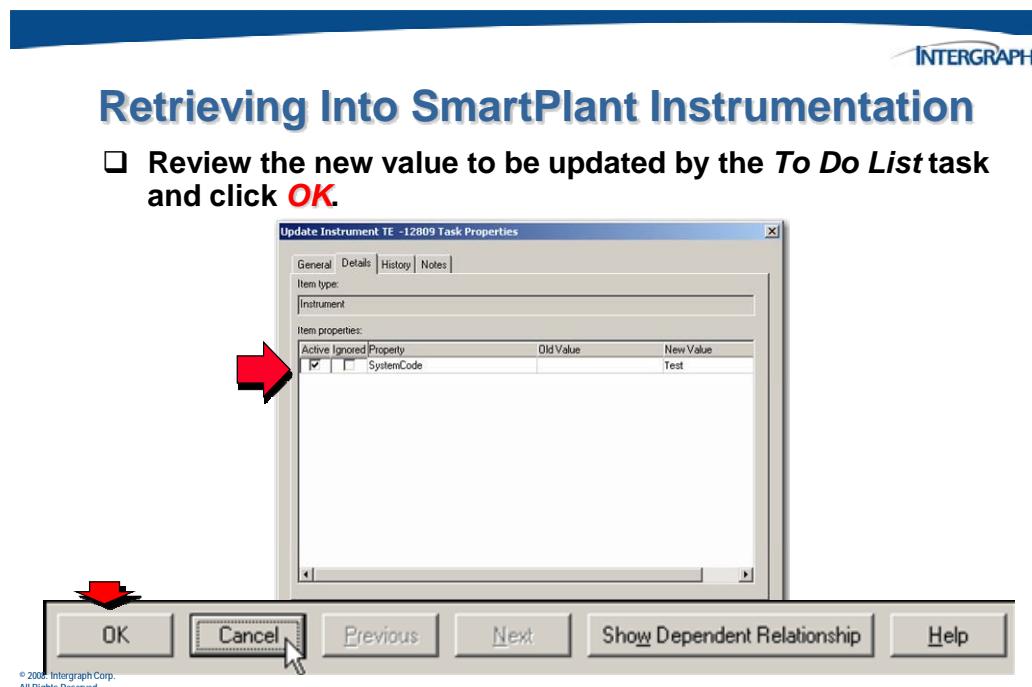
Retrieving Into SmartPlant Instrumentation

- Double-click on one of the tasks to be executed to see its properties.

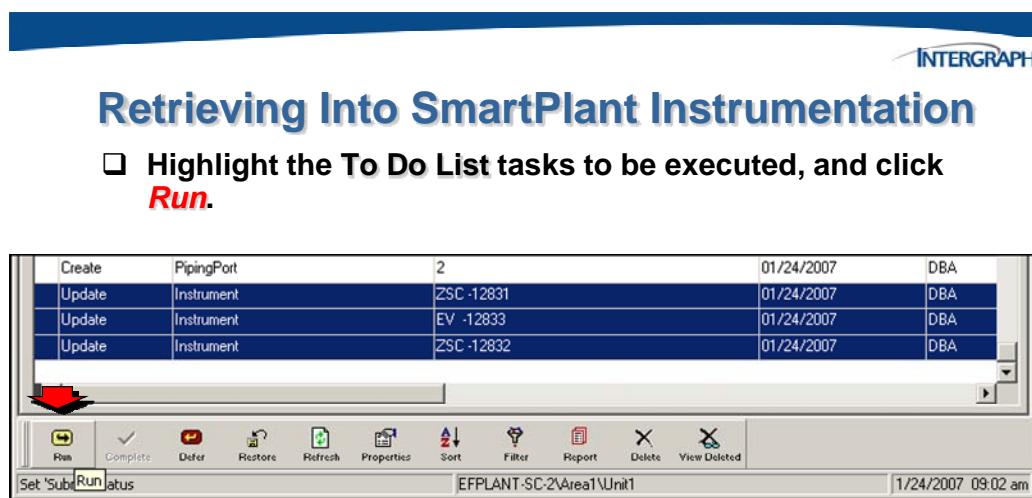
The screenshot shows a "To Do List" dialog box. The table has columns: Task Type, Item Type, Item Name, Created Date, and Created By. The data is as follows:

Task Type	Item Type	Item Name	Created Date	Created By
Create	PipingPort	1	01/24/2007	DBA
Create	PipingPort	2	01/24/2007	DBA
Create	PipingPort	1	01/24/2007	DBA
Create	PipingPort	2	01/24/2007	DBA
Create	PipingPort	3	01/24/2007	DBA
Create	PipingPort	4	01/24/2007	DBA
Create	PipingPort	1	01/24/2007	DBA
Create	PipingPort	2	01/24/2007	DBA
Update	Instrument	ZSC-12831	01/24/2007	DBA
	Instrument	EV-12833	01/24/2007	DBA
Update	Instrument	ZSC-12832	01/24/2007	DBA

View the properties for the instrument.



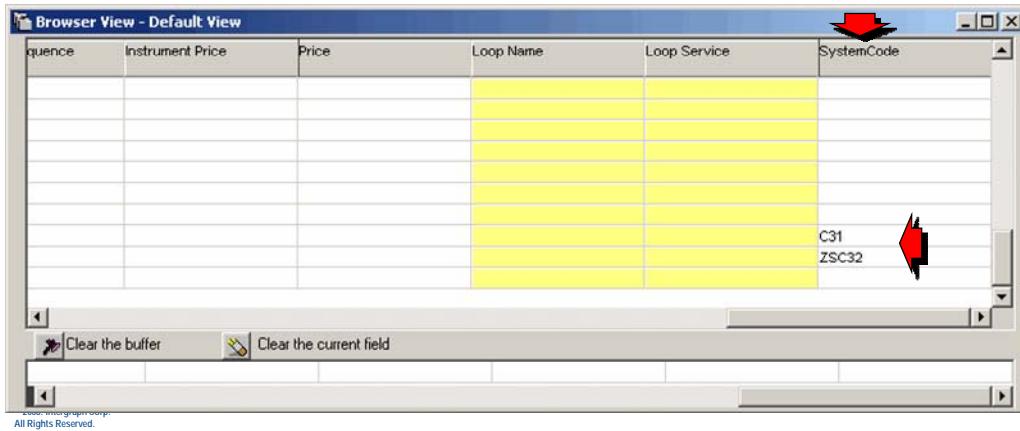
Cancel the *Properties* window, and run the tasks in the *To Do List*.



View the value for the **System Code** property.

Retrieving Into SmartPlant Instrumentation

- The newly added property for the instrument, **SystemCode**, is displayed.
- Make changes to the data to be published back out of SPI.

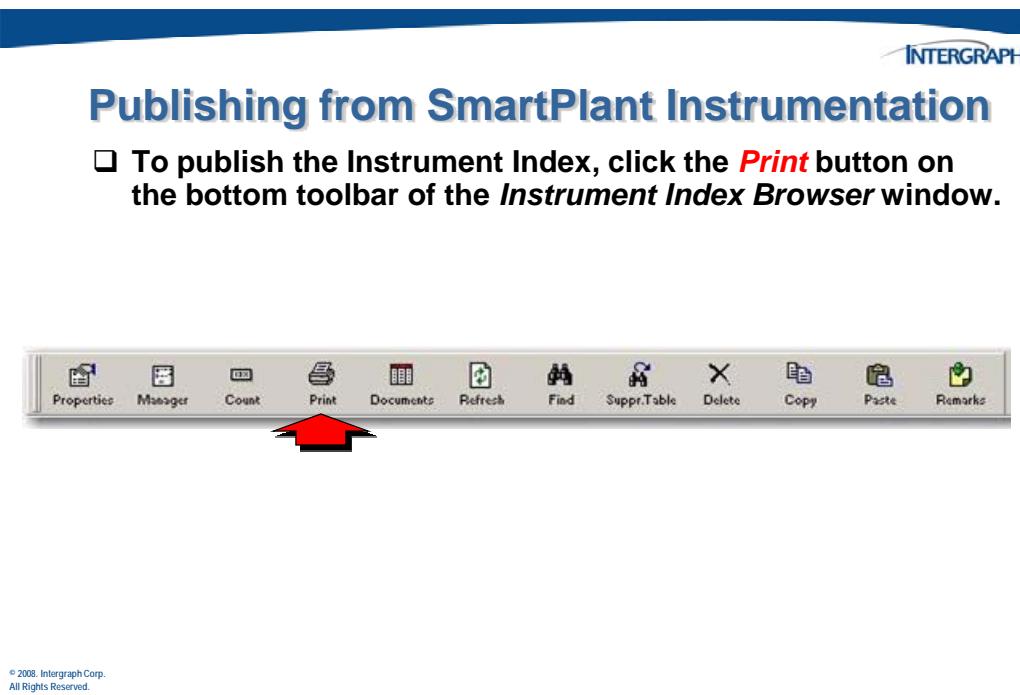


Provide new values for System Code for one or more instruments.

10.8.2 Publish the Instrument Index

Once you have added new data for instruments in the *System Code* property, you are ready to publish those changes back out of SPI.

From the **Instrument Index Browser** window, click the **Print** button on the bottom toolbar.



If prompted to view the report before printing, click **Yes**.



Publishing from SmartPlant Instrumentation

- When prompted, indicate that you do want to review the report before printing.



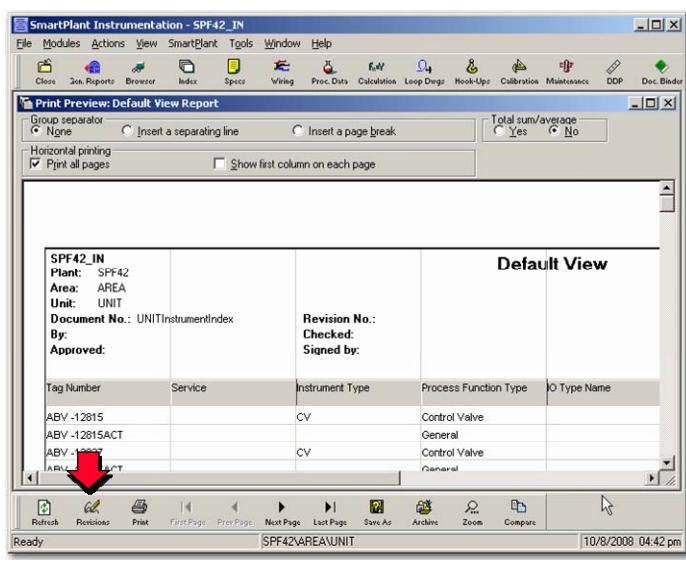
© 2008, Intergraph Corp.
All Rights Reserved.

From the bottom toolbar of the Report Preview window, click the **Revisions** button.



Publishing from SmartPlant Instrumentation

- When the report opens, click the **Revisions** button on the bottom toolbar.

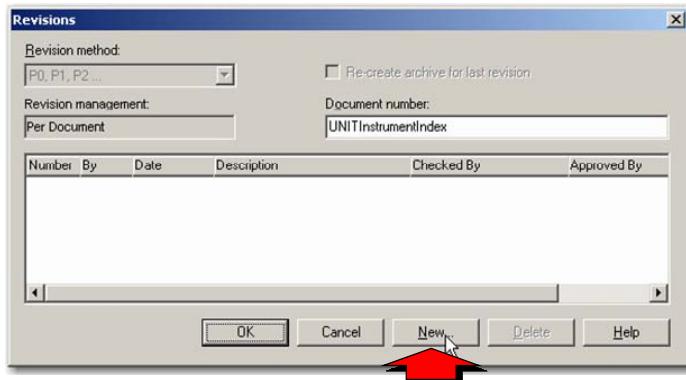


© 2008, Intergraph Corp.
All Rights Reserved.

On the **Revisions** dialog box, click **New** to create a new revision of Instrument Index Report.

Publishing from SmartPlant Instrumentation

- ❑ On the **Revisions** dialog box, click **New** to create a first revision of the document.

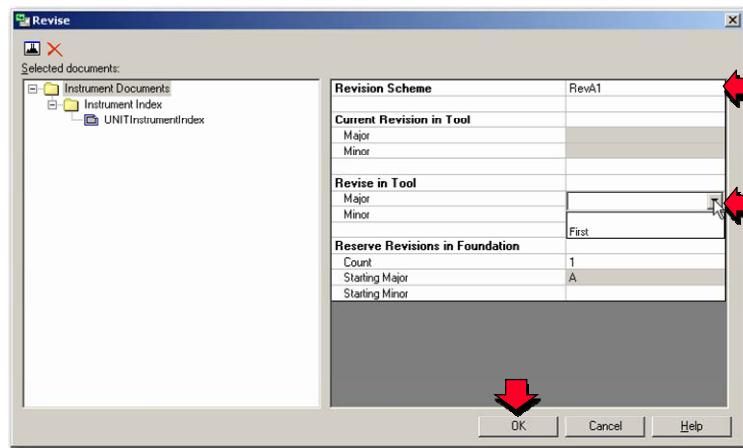


© 2008, Intergraph Corp.
All Rights Reserved.

On the **Revise** form, choose a revision scheme and a starting major revision. Click **OK** to reserve the revision in SmartPlant Foundation.

Publishing from SmartPlant Instrumentation

- ❑ On the **Revise** form, choose a revision scheme and specify the revision with which to start. Click **OK** to create the first revision.

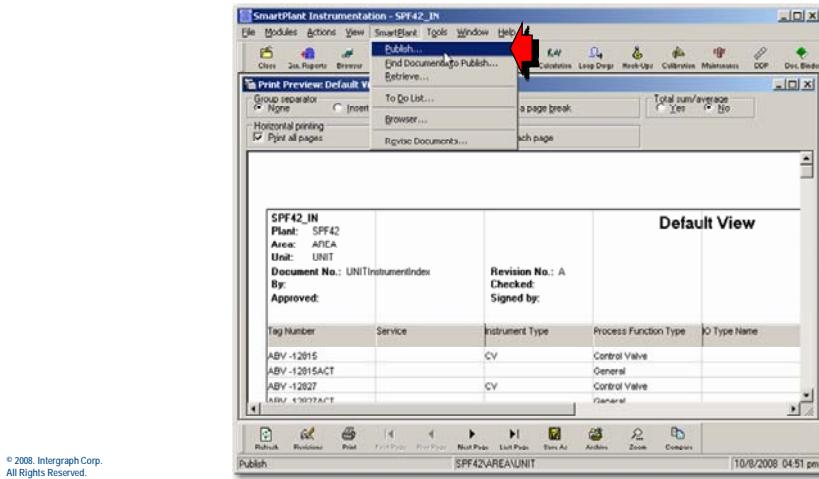


© 2008, Intergraph Corp.
All Rights Reserved.

Once the revision has been reserved, use the **SmartPlant > Publish** command to publish the report.

Publishing from SmartPlant Instrumentation

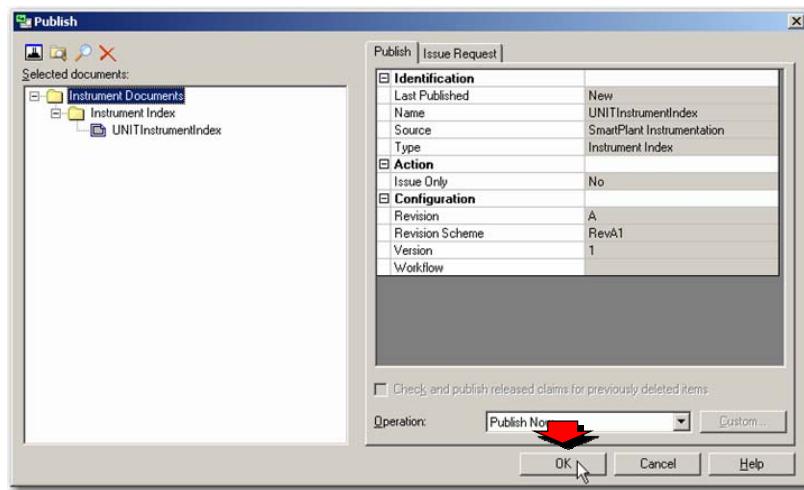
- ❑ Once the revision has been created, dismiss the **Revisions** dialog box, and then click **SmartPlant > Publish**.



Confirm the information as it appears on the **Publish** dialog box, and click **OK**.

Publishing from SmartPlant Instrumentation

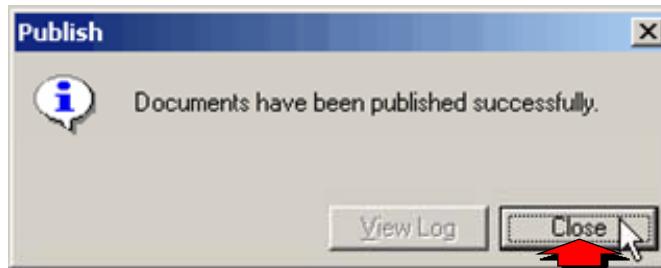
- ❑ Click **OK** to continue and publish the listed documents.



Close the confirmation box when the publish process is complete.

Publishing from SmartPlant Instrumentation

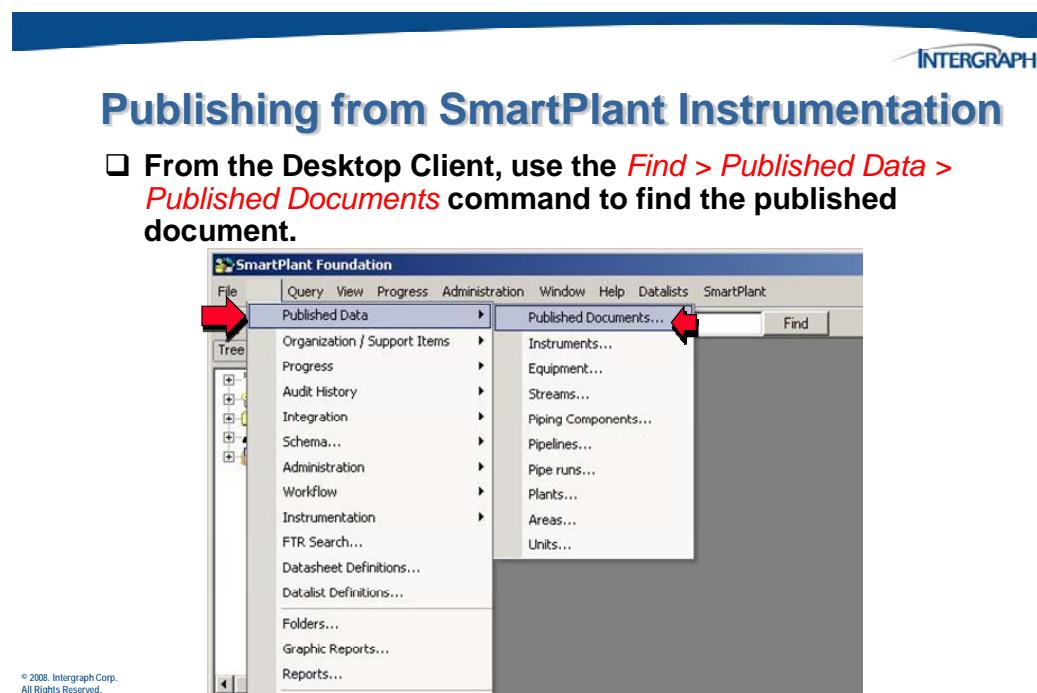
- ❑ When the confirmation dialog box appears, click **Close**.



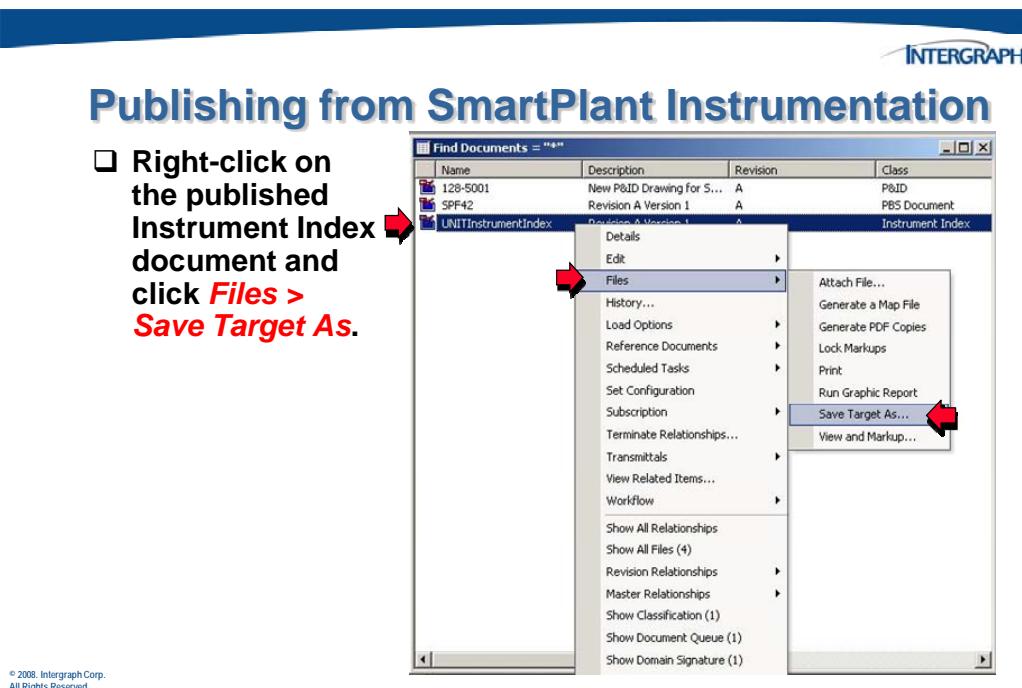
10.8.3 Confirm the Published Data

When the publish process is complete, you can check the published XML data file for the new values in the new property.

From SmartPlant Foundation Desktop Client, find the document you just published from SPI. Use the ***Find > Published Data > Published Documents*** command.



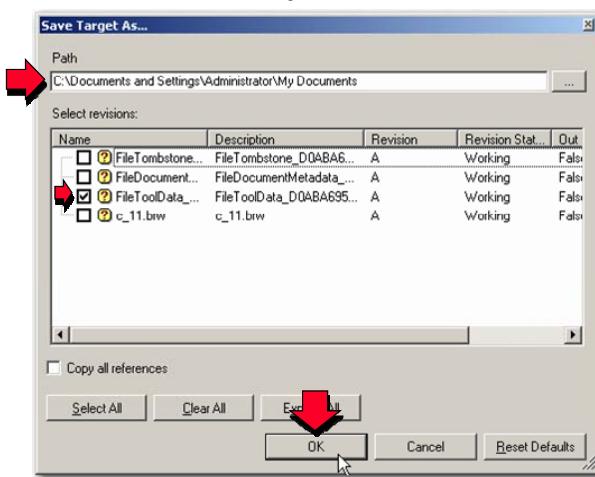
Find the Instrument Index document published from SPI. Use the ***File > Save Target As*** command from the shortcut menu to save out a decrypted copy of the file for viewing.



From the ***Save Target As*** dialog box, select only the check box of the FileToolData XML file. Be sure to note where the file will be placed when copied locally.

Testing the Schema Load

- On the *Save Target As* dialog box, choose data xml file and specify the location where you want to save it.**

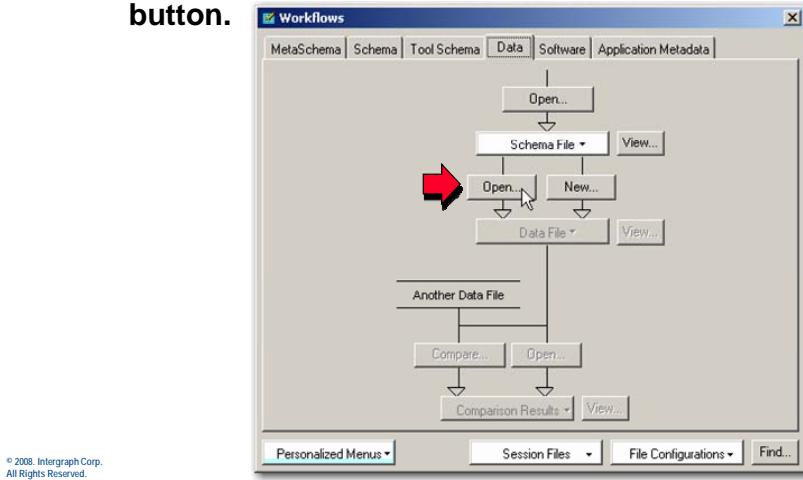


Use the Schema Editor to view the data file. Remember, to view the data file, you must first have a copy of the schema open for comparison. So, open the Schema Editor and connect to your session file before accessing the **Data** tab and using the **Open** button to open the published data file.



Publishing from SmartPlant Instrumentation

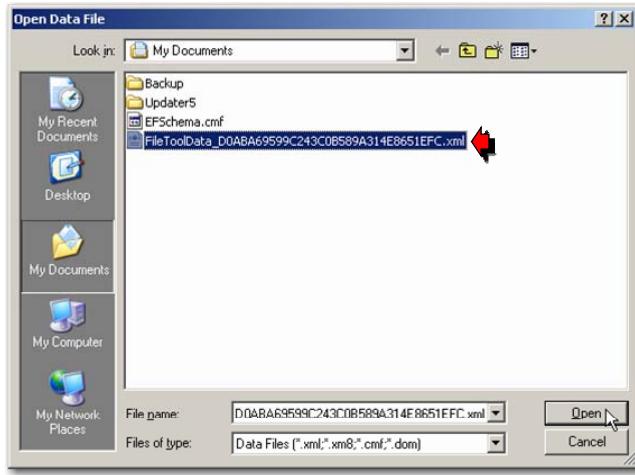
- Launch the Schema Editor tool and open the session file.
From the **Data** tab, click the **Open** button above the **Data File** button.



Find the data file that you copied out of SmartPlant Foundation.

Publishing from SmartPlant Instrumentation

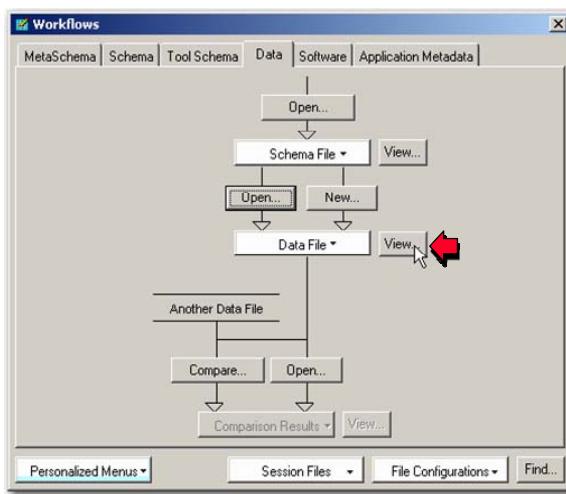
- Find the data xml file that you save out of the Desktop Client, and click **Open**.



Click the **View** button beside the **Data File** button to view the data file.

Publishing from SmartPlant Instrumentation

- Click the **View** button beside the **Data File** button .



Choose the viewer to use when viewing the data.

Publishing from SmartPlant Instrumentation

- Choose the view you want to use to access the data, and click **OK**.



10.9 Activity 2 – Adding a Custom Property to SmartPlant Instrumentation

Complete the Chapter 10 – Activity 1 in the SmartPlant Foundation 2008 (4.2) Modeling and Mapping Activity workbook.

11

C H A P T E R

Mapping with SmartPlant Electrical

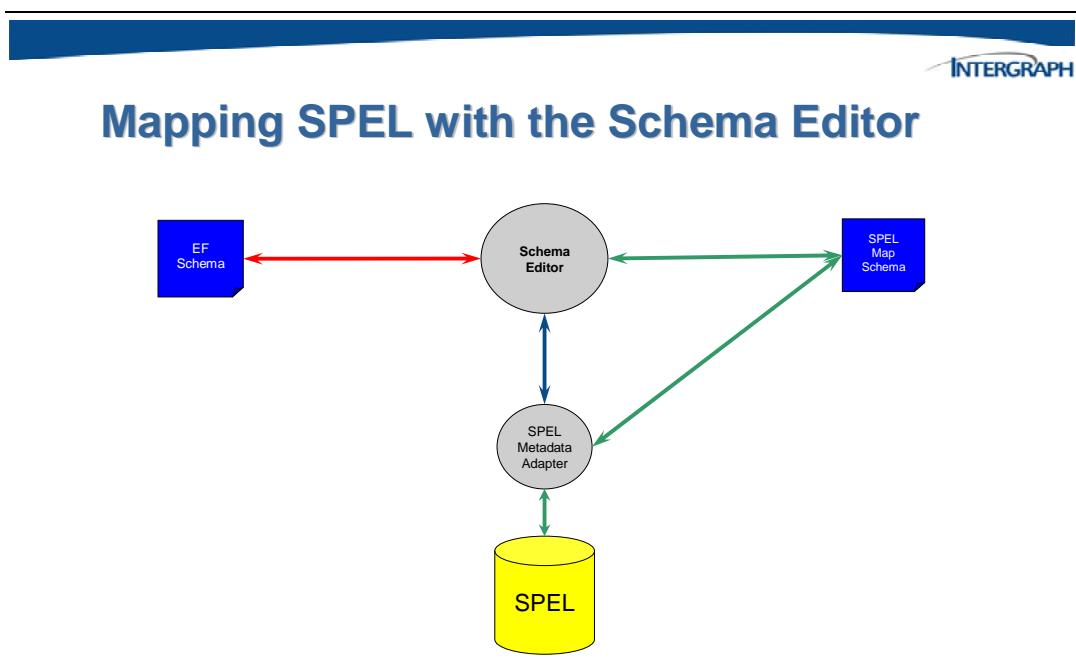
11. Mapping with SmartPlant SPEL

In the last chapters, the concept of extending the delivered schema was introduced along with using the SmartPlant Schema Editor to perform the necessary mapping. In the examples, new enumerated list items and enumerated lists were created, as well as properties to use those lists and string properties. The process involved using the a tool application to add the desired list entries to the tool's application meta schema. The next step was to use the Schema Editor and the tool's meta data adapter to read the new enumerated additions and automatically synchronize those additions with the tool map schema. The final steps were to add the new entries to the SmartPlant Schema and map the two schemas together using the schema editor Map Environment.

In this chapter, we will do the same kind of steps to make addition to the SmartPlant SPEL application and the SPEL tool map schema. The same properties that we added to SPPID will be propogated forward for use in SPEL and then will be mapped. The goal is to created the properties and mapping so that these values can be shared between SPPID and SPEL using SmartPlant Foundation as an integration tool.

Again, there are two different methods for making these changes. However, in this chapter we will focus on the “outside-in” approach, where changes are made on the tool side and then moved forward into SPF.

The figure below shows the function of the metadata adapter with the schema editor.



As in the previous examples, the desired change will first be made in the SmartPlant SPEL application using the Data Dictionary Manager tool. There is a specific reason for approaching the problem in this manner. The property to be added could be used by several different types of objects in SmartPlant SPEL; Electrical Motor, Cable, Wire, Instrument, etc. Rather than add the same property multiple times, if the new property will be used as a “global” Plant Item item type property, then it can be added to the Plant Item table in the data dictionary but be available to other classes through inheritance. The PlantItem item type is a classification that provides common relationships and attribution for all the types of objects that may exist independently of any other object. This item type is derived from the ModelItem class. The subclasses of ModelItem include all those types whose existence is not dependent on another.

Once the new property has been added with the Data Dictionary Manager, the schema editor will be used to read this new property from the application database, which is also called the tool meta schema. This is accomplished by a part of the software known as the metadata adapter. The task of the metadata adapter is to extract information from the meta schema and automatically update the contents of the tool map schema so that the meta schema and the tool map schema are synchronized.

11.1 Adding New Simple Properties

The procedure for adding properties to the authoring tools differs from tool to tool. The following example describes the procedure for adding a new property to the SmartPlant SPEL data model using SmartPlant Data Dictionary Manager (delivered with SmartPlant Engineering Manager). For more information about adding properties to SmartPlant SPEL, see the SmartPlant Data Dictionary Manager documentation.

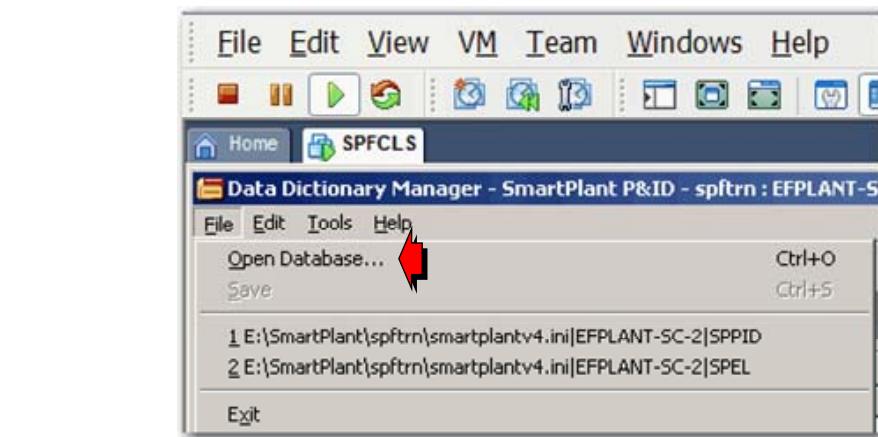
To add a property to SmartPlant Electrical, open Data Dictionary Manager by clicking **Start > All Programs > Intergraph SmartPlant Engineering Manager > Data Dictionary Manager**.

As we have been working in the SPPID database with the Data Dictionary Manager tool, you will need to use the **File > Open Database** command to open the SmartPlant Electrical database.



Add Properties to the Meta-Schema

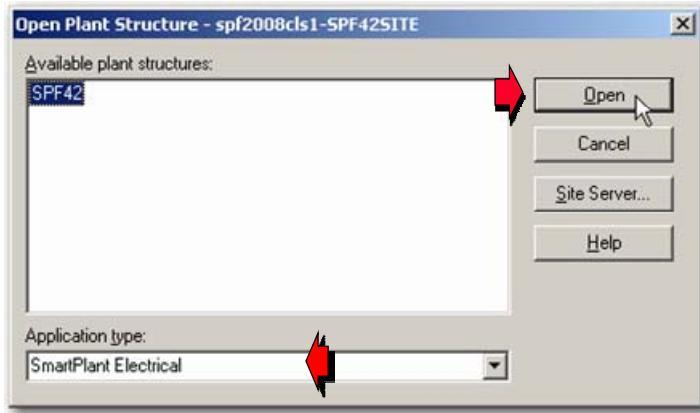
- Open the SmartPlant Electrical Database in SmartPlant Engineering Manager**



Choose the SmartPlant Electrical application.

Add Properties to the Meta-Schema

- Select **SmartPlant Electrical** under **Application type**, and click **Open**.

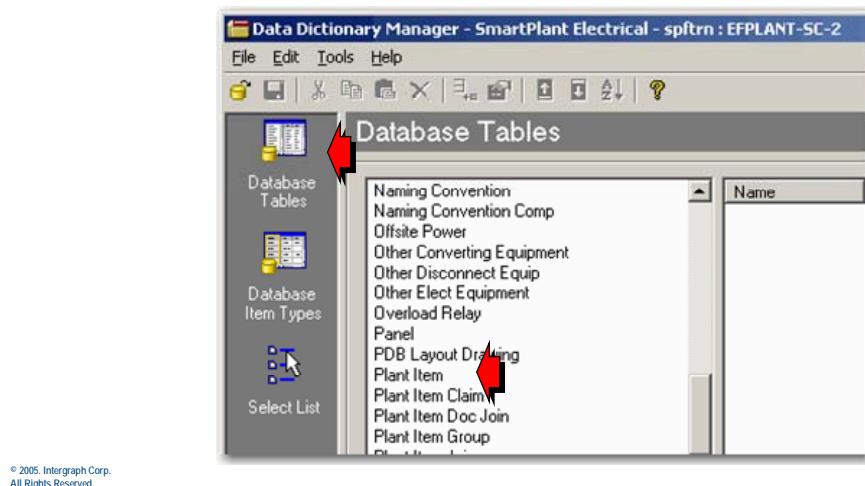


The following example describes how to add a new simple property to the SmartPlant SPEL data model.

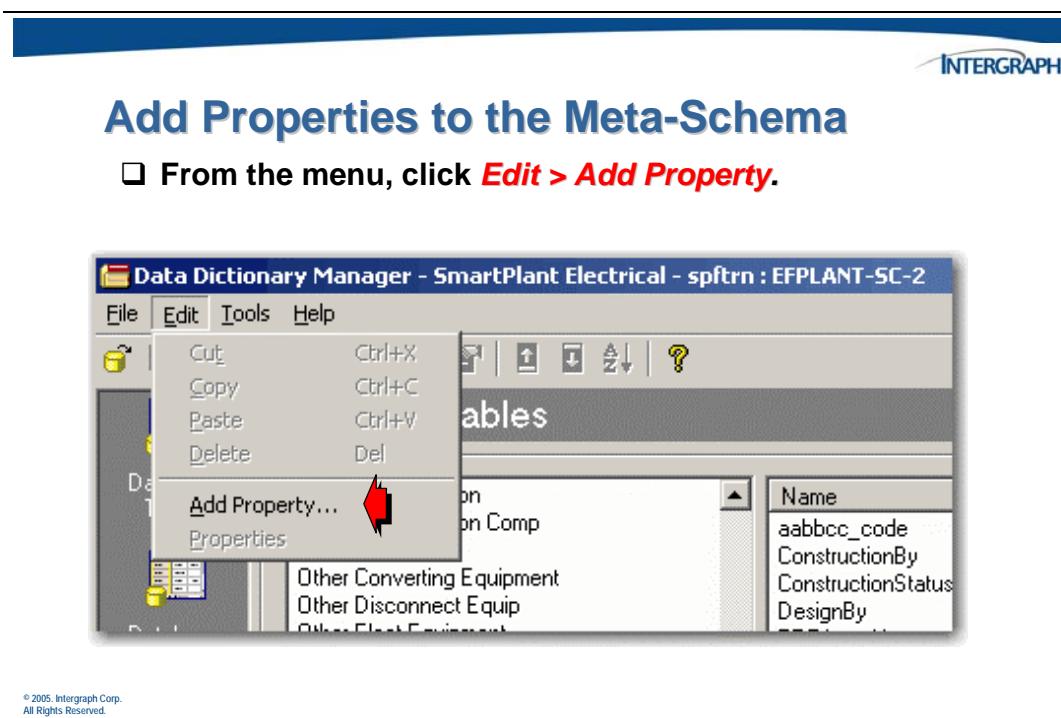
When you open Data Dictionary Manager, the **Database Tables** view is already selected. To select the database table to which you want to add a property, click the name of the table in the list.

Add Properties to the Meta-Schema

- To define a new property, click **Plant Item** in the **Database Tables** view to add the new property to the Plant Item table.



After you select the table that you want, click **Edit > Add Property** to create a property in that table.

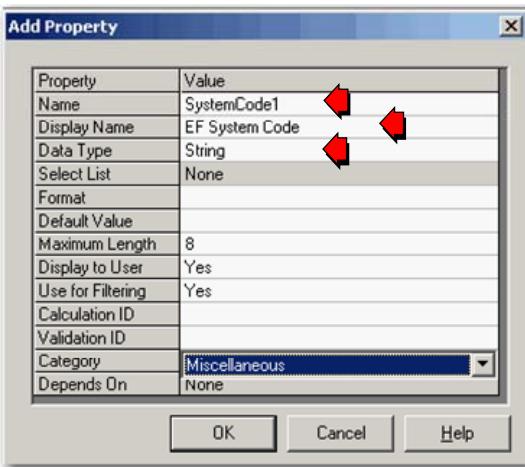


The *Add Property* dialog box appears. Define information for the new property you want to create.

In this example, we will be adding a new property called *SystemCode1*, because the SmartPlant Electrical database already contains a property called SystemCode.

Add Properties to the Meta-Schema

- In the *Add Property* dialog box, define attributes for the new property, including the **Name**, **Display Name**, and **Data Type**.



Provide a name, a description, data type, and maximum length. Additionally, you can specify a category to make it easier to find out property in the authoring tool.

Property	Value
Name	SystemCode1
Display Name	EF System Code
Data Type	String
Select List	None
Format	
Default Value	
Maximum Length	8
Display to User	Yes
Use for Filtering	Yes
Calculation ID	
Validation ID	
Category	Miscellaneous
Depends On	None

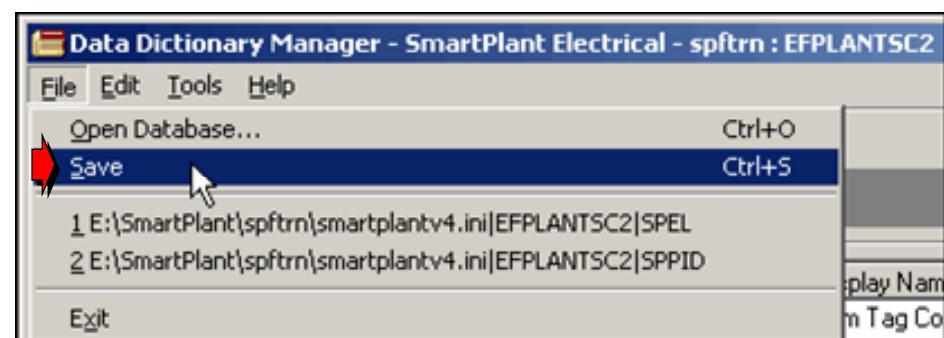
© 2005. Intergraph Corp.
All Rights Reserved.

The new property is now in the *Plant Item* table.

Name	Display Name	Data Type	Select List	Format
ItemTagName	Item Tag Compl...	String	None	a80
Name	Name	String	None	Variable Length
PlantItemCode	Plant Item Type	Select List	Plant Item Type	Variable Length
RequisitionData...	Reference Data...	Select List	Boolean Values	Variable Length
RequisitionBy	Requisition By	Select List	Requisition Res...	Variable Length
RequisitionNo	Requisition Nu...	String	None	Variable Length
SP_AllowPublish...	Allow Publish Fl...	Select List	Boolean Values	Variable Length
SP_KKSCompon...	KKS Compon...	String	None	Variable Length
SP_KKSEquipm...	KKS Compon...	String	None	Variable Length
SP_KKSEquipm...	KKS Equipment...	String	None	a3
SP_KKSEquipm...	KKS Equipment...	String	None	Variable Length
SP_KKSEquipm...	KKS Equipment...	String	None	Variable Length
SP_KKSSystem...	KKS System Key	String	None	Variable Length
SP_KKSSystem...	KKS System Se...	String	None	a2
SP_KKSTotalPlant	KKS Total Plant	String	None	Variable Length
SP_PartNo	Part Number	String	None	Variable Length
SP_PIDDrawing...	P&ID Drawing ...	String	None	Variable Length
SupplyBy	Supply By	Select List	Supply Respon...	Variable Length
SystemCode1	EF System Code	String	None	None
TagPrefix	Tag Prefix	String	None	a50
TagSequenceNo	Tag Sequence ...	String	None	Upper Case - V...
TagSuffix	Tag Suffix	String	None	a50

© 2005. Intergraph Corp.
All Rights Reserved.

Save your changes to the SmartPlant Electrical meta schema.

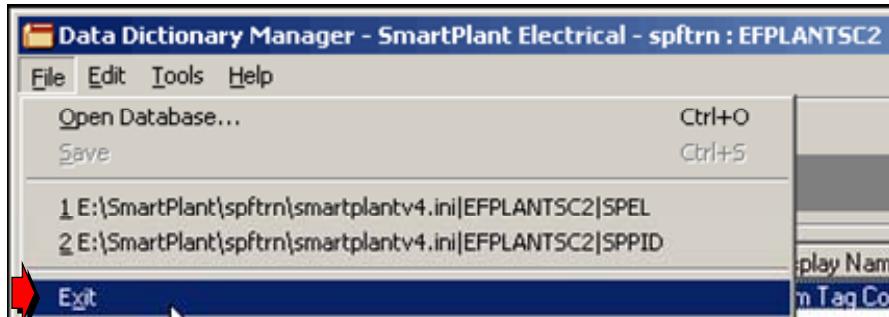


© 2005. Intergraph Corp.
All Rights Reserved.

Click the **File > Exit** command to close the Data Dictionary Manager.

Add Properties to the Meta-Schema

- Click **File > Exit** from the menu.



© 2005. Intergraph Corp.
All Rights Reserved.

11.2 Updating the Tool Map Schema

Once the changes are part of the tool's meta schema, you need to make sure the new properties are part of the tool map file. To update the tool map file, use the Metadata adapter, which is access through the Schema Editor.

Typically, you would then need to add the new property to the SmartPlant schema, also using the Schema Editor. However, in this case, we already created the property to the SmartPlant Schema when we mapped the property from SPPID. Once we have updated the tool map schema, we will need to make sure that our custom interface is available for use on the applicable class defs, and we will be ready to create our mapping.



Adding a New SmartPlant Property

Add a Property
to Tool Map
Schema

Add a Property
to SmartPlant
Schema

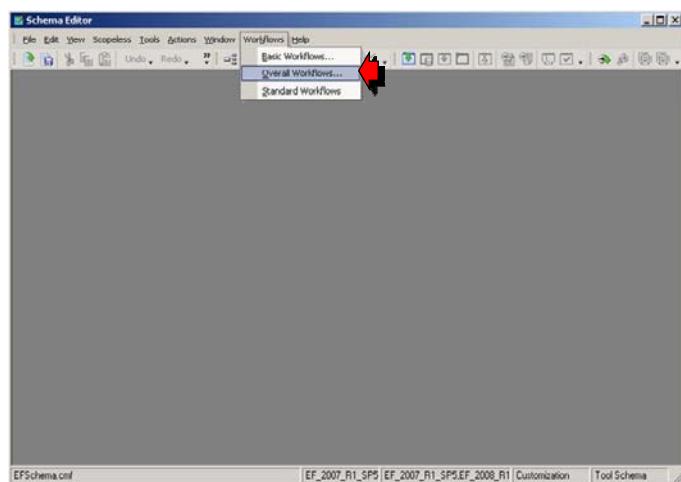
To synchronize our tool map file with our tool meta schema, we will use the metadata adapter and the schema editor.

Even though our System Code property already exists in the SmartPlant Schema, we will need to make changes to ICustomInterface. So we need to check out and launch the plant's CMF file from the SmartPlant Foundation Desktop Client.

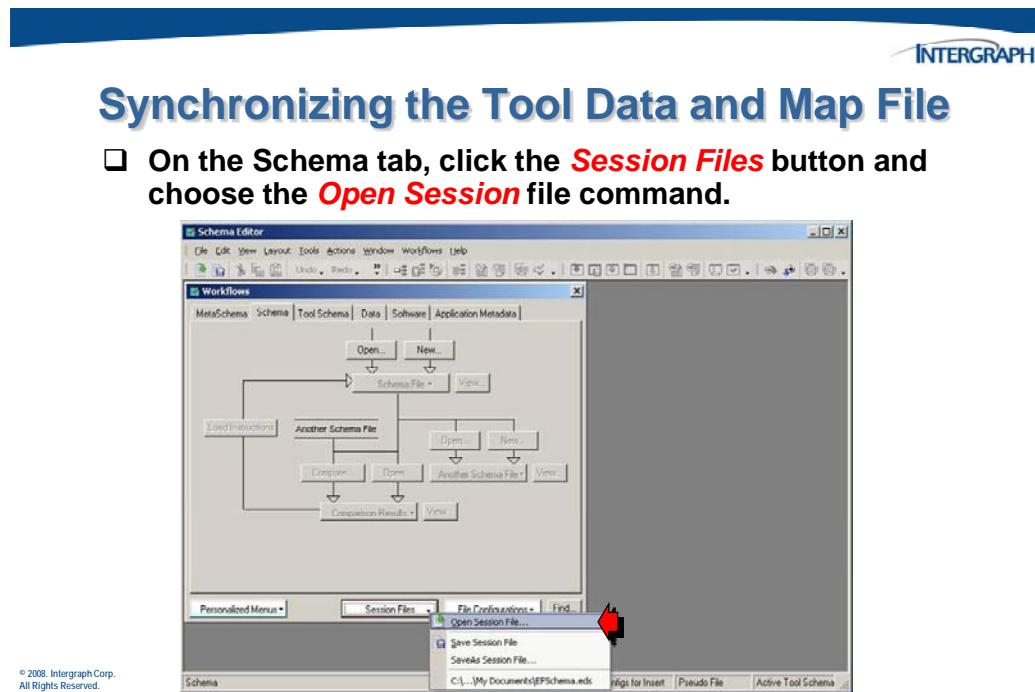
In the following example, we have already been working with the checked out CMF file and will open the Schema Editor and launch the saved session file.

Synchronizing the Tool Data and Map File

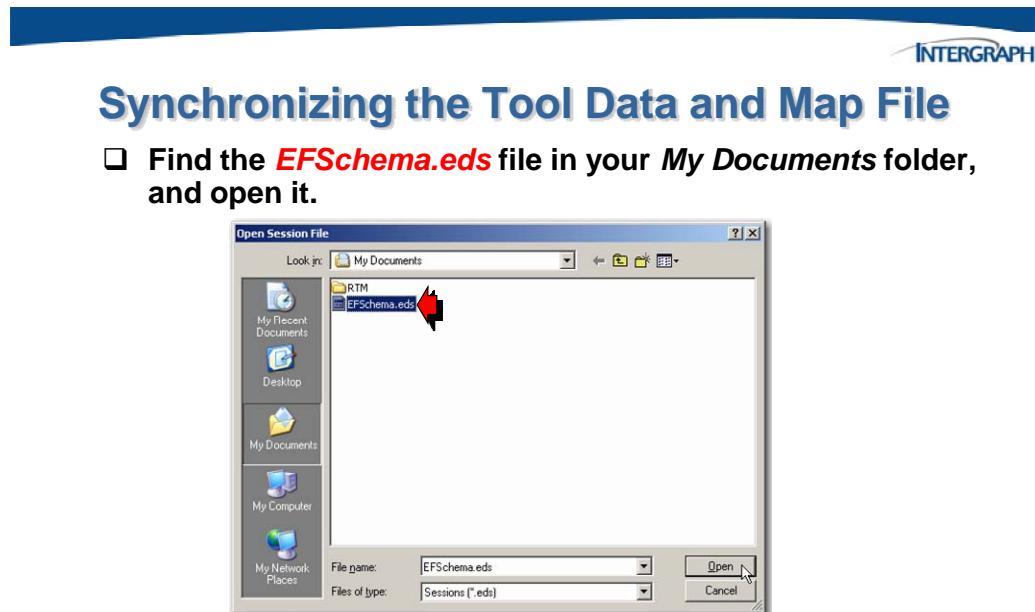
- Launch the Schema Editor and from the **Windows** menu, choose the **Overall Workflow**.



Open the ***Overall Workflow***, and use the ***Session Files*** button to access the ***Open Session File*** command.



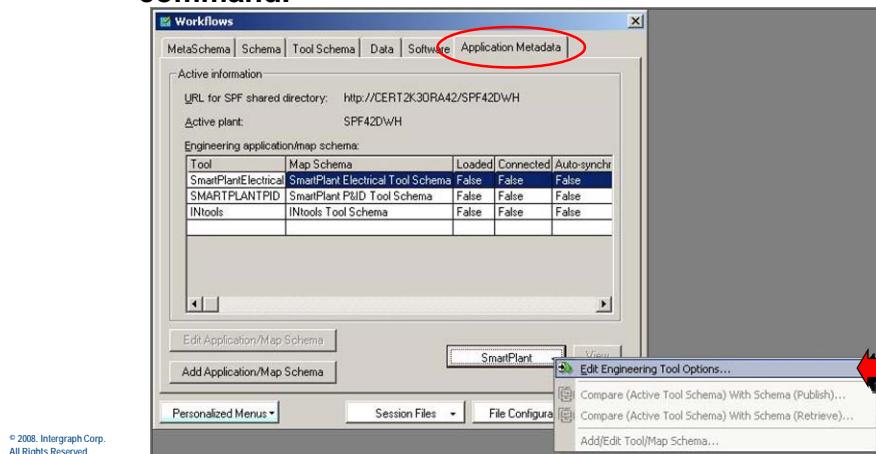
Launch the session file with the applicable connection information to the plant schema.



From the *Application Metadata* tab, click the *SmartPlant* button, and choose the *Edit Engineering Tool Options* command.

Synchronizing the Tool Data and Map File

- From the *Metadata Adapter* tab, click the *SmartPlant* button, and choose the *Edit Engineering Tool Options* command.



Select the SmartPlant Electrical tool map file, and indicate that you want to connect to both the tool map file and the tool application schema.

Synchronizing the Tool Data and Map File

- Choose the metadata adapter for *SmartPlant Electrical*, and indicate that you want to launch the tool map schema and connect to the tool meta schema.



The **Synchronize** screen shows the differences in the two schemas and allows you to choose what changes you want to make to correct these discrepancies.

Synchronizing the Tool Data and Map File

- On the **Synchronize** screen, click **OK** to push the changes you made in the Data Dictionary tool to the SPEL Tool Map Schema.

Synchronize					
For each row below select option that should be applied					
Class	Parent Name	Name	Property	SmartPlantElectrical	
SPMapPropertyDef	Junctionbox	SystemCode1		Missing	Add # ▾
SPMapPropertyDef	LocalPanel	SystemCode1		Missing	Add # ▾
SPMapPropertyDef	Motor	SystemCode1		Missing	Add # ▾
SPMapPropertyDef	Multimeter	SystemCode1		Missing	Add # ▾
SPMapPropertyDef	OffStepPower	SystemCode1		Missing	Add # ▾
SPMapPropertyDef	OtherConvertingEquipment	SystemCode1		Missing	Add # ▾
SPMapPropertyDef	OtherDisconnectedEquip	SystemCode1		Missing	Add # ▾
SPMapPropertyDef	OtherEquipment	SystemCode1		Missing	Add # ▾
SPMapPropertyDef	OverloadRelay	SystemCode1		Missing	Add # ▾
SPMapPropertyDef	PDBRow	SystemCode1		Missing	Add # ▾
SPMapPropertyDef	PDBSection	SystemCode1		Missing	Add # ▾
SPMapPropertyDef	PotentialTransformer	SystemCode1		Missing	Add # ▾
SPMapPropertyDef	PowerDistributionBoard	SystemCode1		Missing	Add # ▾
SPMapPropertyDef	PowerDistributionEquip	SystemCode1		Missing	Add # ▾
SPMapPropertyDef	ProcessEquipment	SystemCode1		Missing	Add # ▾
SPMapPropertyDef	ProtectorRelay	SystemCode1		Missing	Add # ▾
SPMapPropertyDef	RelayFunction	SystemCode1		Missing	Add # ▾
SPMapPropertyDef	Resistor	SystemCode1		Missing	Add # ▾
SPMapPropertyDef	SegmentPartition	SystemCode1		Missing	Add # ▾
SPMapPropertyDef	SignalRun	SystemCode1		Missing	Add # ▾
SPMapPropertyDef	Starter	SystemCode1		Missing	Add # ▾
SPMapPropertyDef	Strip	SystemCode1		Missing	Add # ▾
SPMapPropertyDef	Terminal	SystemCode1		Missing	Add # ▾
SPMapPropertyDef	Transformer	SystemCode1		Missing	Add # ▾
SPMapPropertyDef	TransformerComponent	SystemCode1		Missing	Add # ▾
SPMapPropertyDef	Tray	SystemCode1		Missing	Add # ▾
SPMapPropertyDef	UPS	SystemCode1		Missing	Add # ▾
SPMapPropertyDef	VariableFrequencyDrive	SystemCode1		Missing	Add # ▾
SPMapPropertyDef	Voltmeter	SystemCode1		Missing	Add # ▾

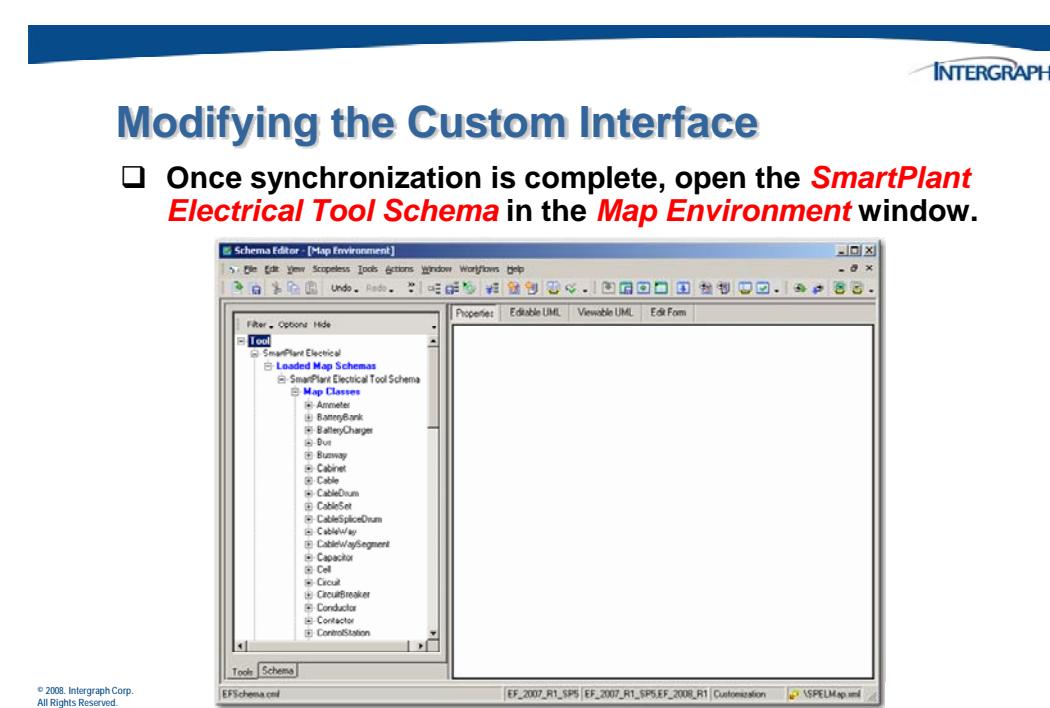
Save Extensions | Save Metadata | Save to Excel | OK | Cancel/Work Disconnected | Help

© 2008, Intergraph Corp.
All Rights Reserved.

11.3 Modifying the Custom Interface

Now the tool map schema is synchronized with the tool application schema and contains the new property that we will want to map. But before we can map the property to the System Code property that already exists in the SmartPlant schema, we need to make some modifications to our custom interface that exposes the System Code property, to make it available to our SmartPlant Electrical objects.

The **Map Environment** window is displayed by default when the synchronization process is complete.



We want to add the system code property to the following objects in SPEL.

Modifying the Custom Interface

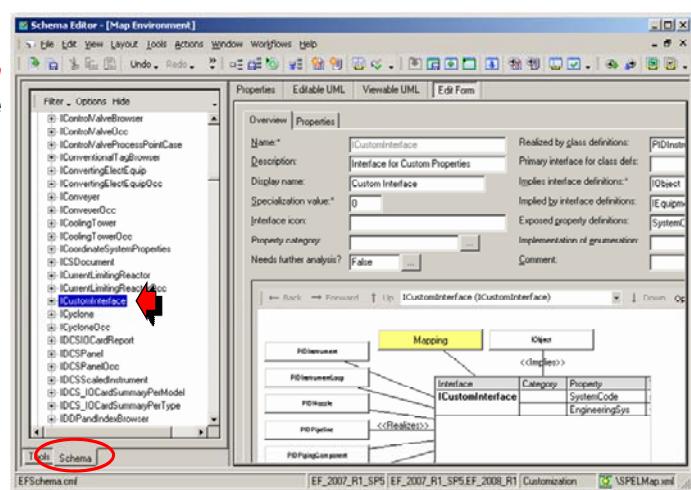
- Next, we should investigate where our custom interface will be needed.
- We want to publish our **System Code** property for the following types of objects from SPEL:
 - **Instruments**
 - **Wires**
 - **Cables**
 - **Motors**

© 2008, Intergraph Corp.
All Rights Reserved.

From the **Map Environment**, go to the **Schema** tab on the left window. Under **InterfaceDefs**, find the **ICustomInterface**. Open the **Edit Form** view.

Modifying the Custom Interface

- Again from the **Schema** tab, find the **ICustomInterface** from the **Interface Defs** list, and open the **Edit Form** tab on the top.



© 2008, Intergraph Corp.
All Rights Reserved.

Using the list below, make sure the interface is realized and implied by the class defs and interface defs for the SPEL objects.

Modifying the Custom Interface

- Edit the interface so that it is realized by the class defs and implied by the primary interfaces of those class defs in the list below:

<u>Class Defs to Realize the Interface</u>	<u>Interfaces to Imply our Interface</u>
ELEElectricMotor	IElectricMotorOcc
ELECable	ICableOcc
ELEWire	IWireOcc
ELEInstrument	IInstrumentOcc

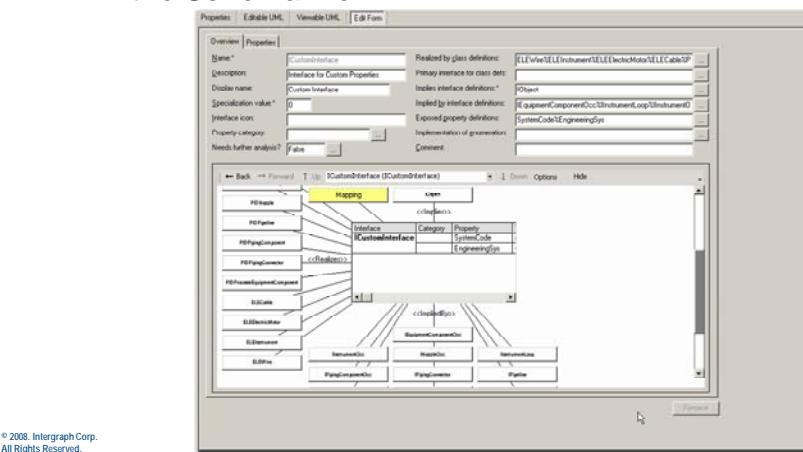
- The *IInstrumentOcc* interface should already exist in the list of Implied By Interfaces because it was also the Primary Interface for the *PIDInstrument* class def.**

© 2008. Intergraph Corp.
All Rights Reserved.

Save your changes to the SmartPlant Schema.

Modifying the Custom Interface

- Edit the ***ICustomInterface*** to be realized by the classdefs for which we want to create mapping. Save your changes to the Schema file.



11.4 Mapping the New Property

So far in this chapter, we have created a new property called **SystemCode** with a property type of *string* and added it to our tool database. Next, we used the metadata adapter to synchronize our tool meta data (the tool database) with the tool schema. Then we extended our custom interface, which exposes our property in the SmartPlant schema, to be available to SmartPlant Electrical objects.

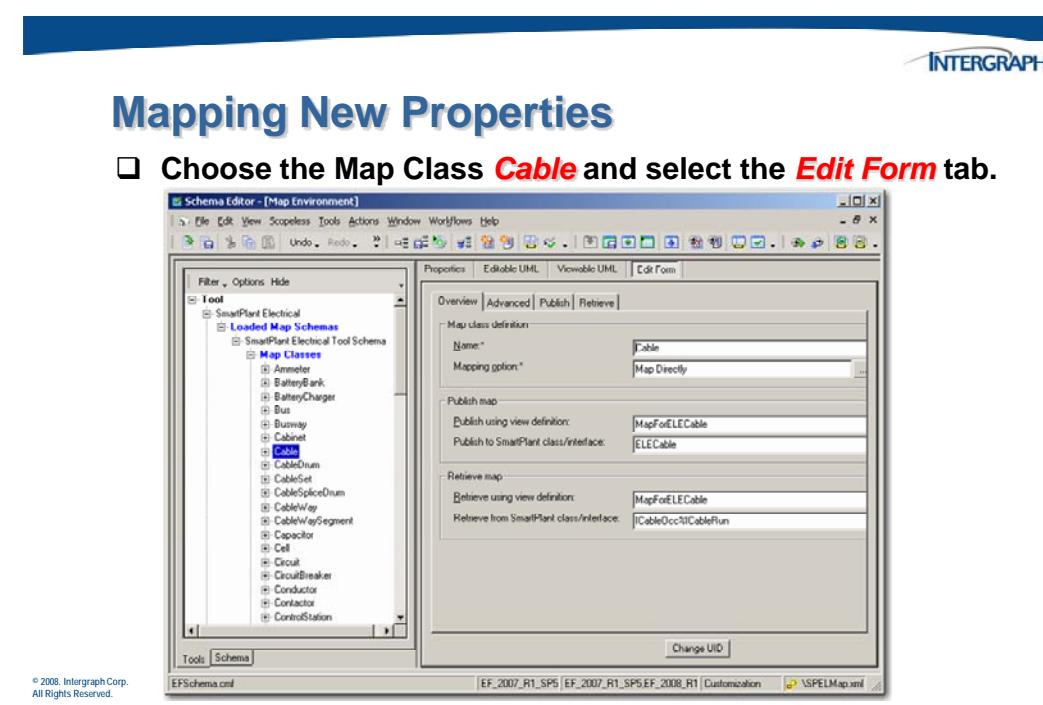
In some cases, the next step would be to add the new property to the SmartPlant schema so that the properties in the tool schema and the SmartPlant schema can be mapped. However, in other cases, like this one, we may have already created the new property in the SmartPlant schema when we added the property to another authoring tool and mapped the property from that tool's schema. In this case, the new properties we added to the SmartPlant Electrical database already exist in the SmartPlant schema, because we added them when we mapped the same properties from SmartPlant P&ID.



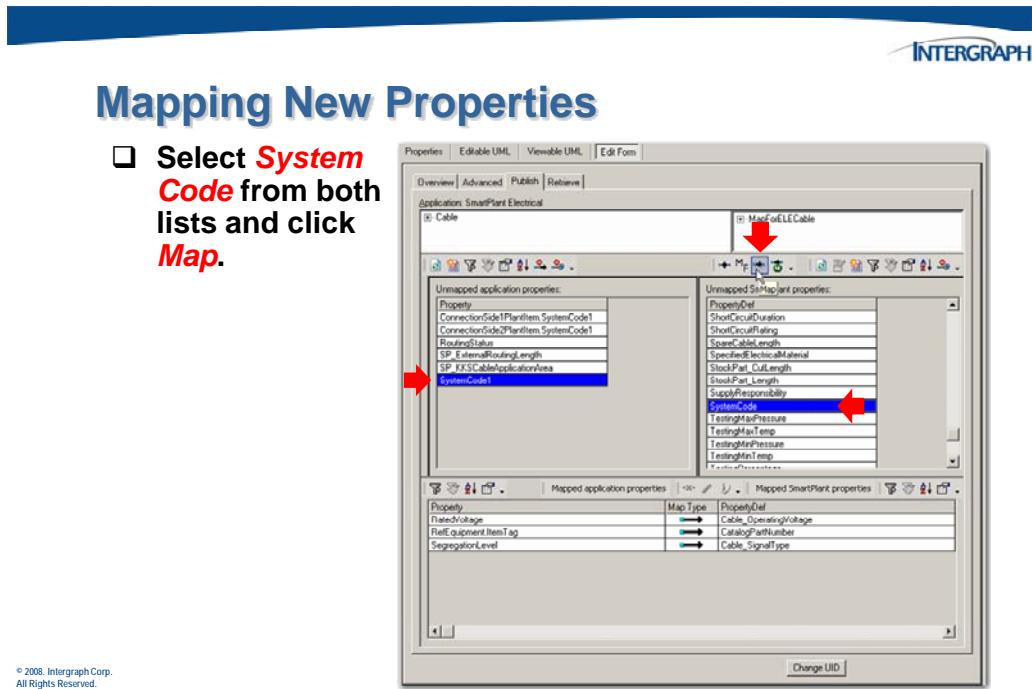
Mapping New Properties

Map Property in
Tool Map
Schema

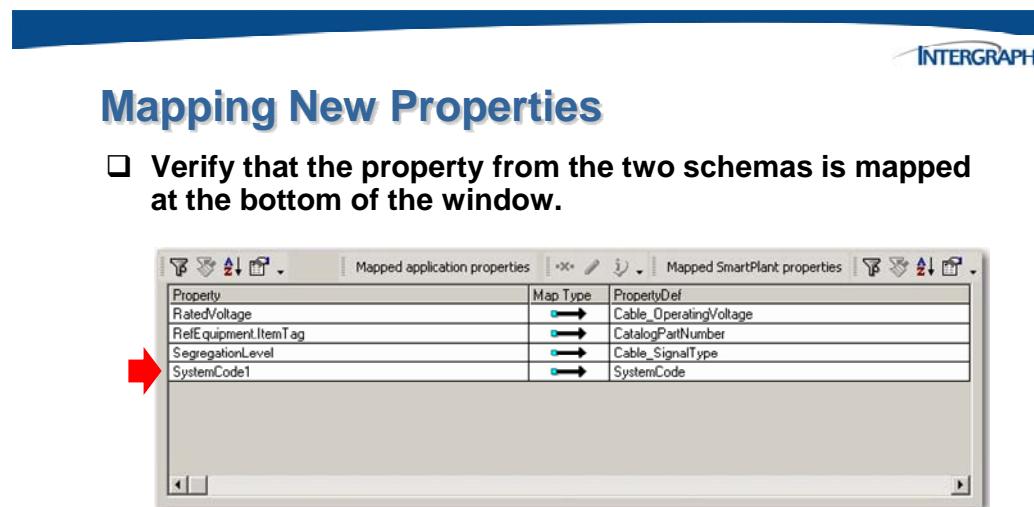
From the **Map Environment** window, expand the **SmartPlant Electrical** tool map file, and find the **Cable** map class. Open the **Edit Form**.



On the **Publish** tab, find the **SystemCode1** property in the tool map file and the **SystemCode** property in the SmartPlant schema, and use the **Map** button to create a mapping relationship.

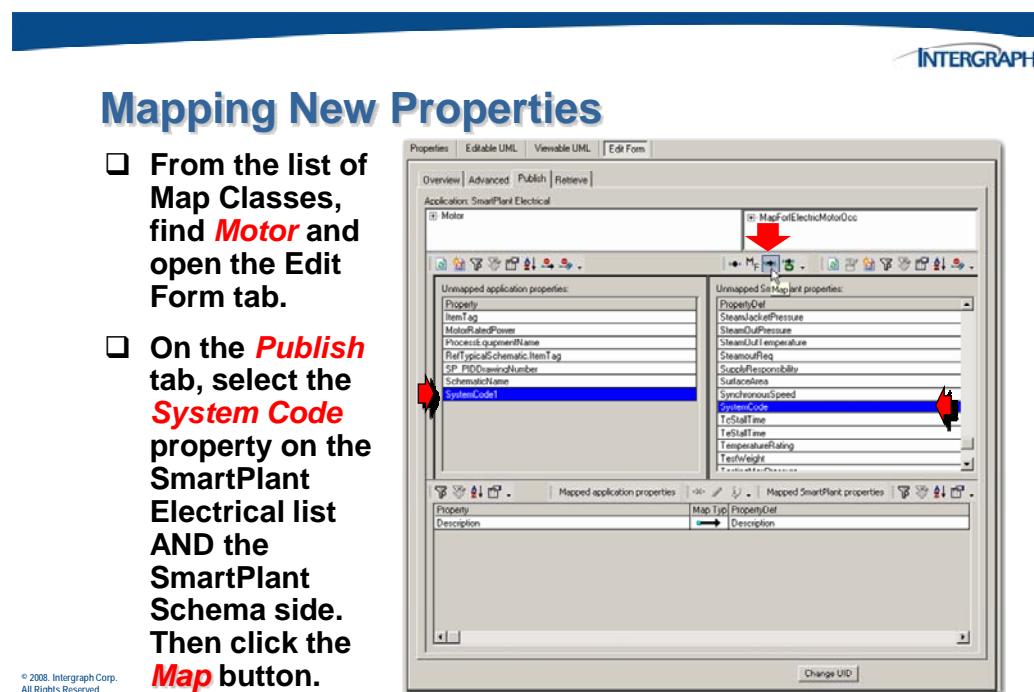


Confirm the mapping in the bottom of the window.



© 2008, Intergraph Corp.
All Rights Reserved.

Repeat the process for other map classes where the property will be used – like ***Motor***.

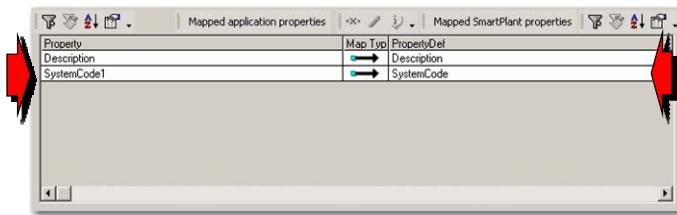


© 2008, Intergraph Corp.
All Rights Reserved.

Confirm the mapping.

Mapping New Properties

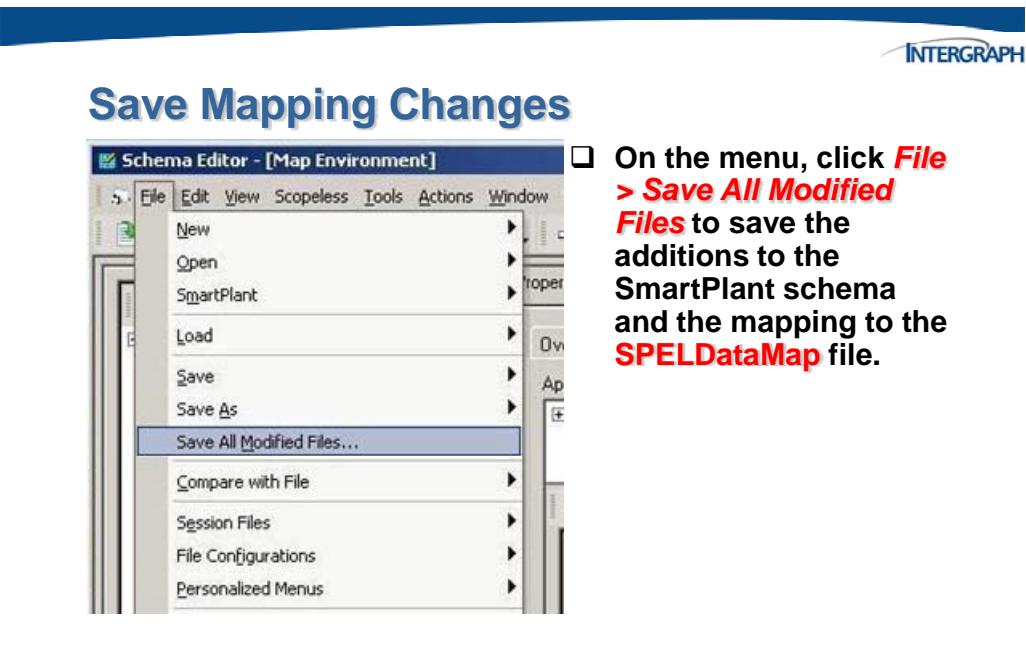
- Verify that the property from the two schemas is mapped at the bottom of the dialog.



© 2008, Intergraph Corp.
All Rights Reserved.

11.5 Saving Mapping Changes

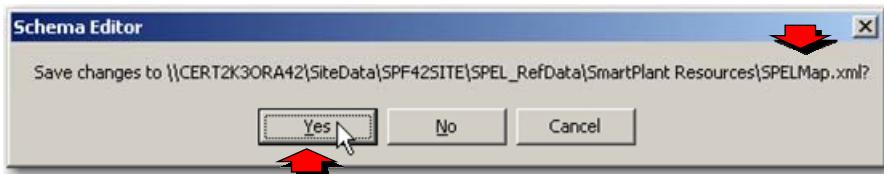
At this point, the additions to the tool schema and the mapping information is stored in memory and will need to be saved.



You will be prompted to save the changes to the tool map schema.

Save Mapping Changes

- Verify the tool map schema name for the mapping changes and choose Yes.



If you saved your changes to the CMF file when you finished extending the ICustomInterface, you will not be prompted to save the CMF file again, as the mapping relationships are stored in the tool map file.

However, if you did not save your changes earlier, the following prompt will appear.

Save Mapping Changes

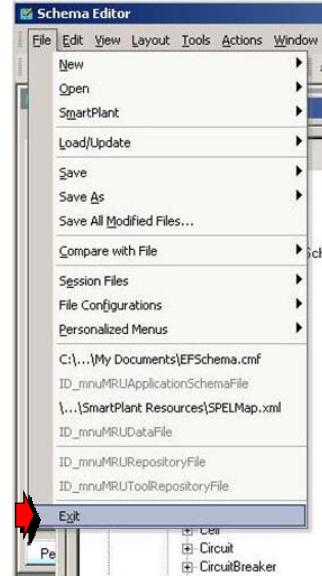
- Verify the CMF file for the property addition and choose Yes.



Once the schema file has been saved exit the Schema Editor.

Save Mapping Changes

- On the menu, click **File > Exit** to close out of the schema editor.



© 2008, Intergraph Corp.
All Rights Reserved.

11.6 Activity 1 – Adding and Mapping a Simple Property

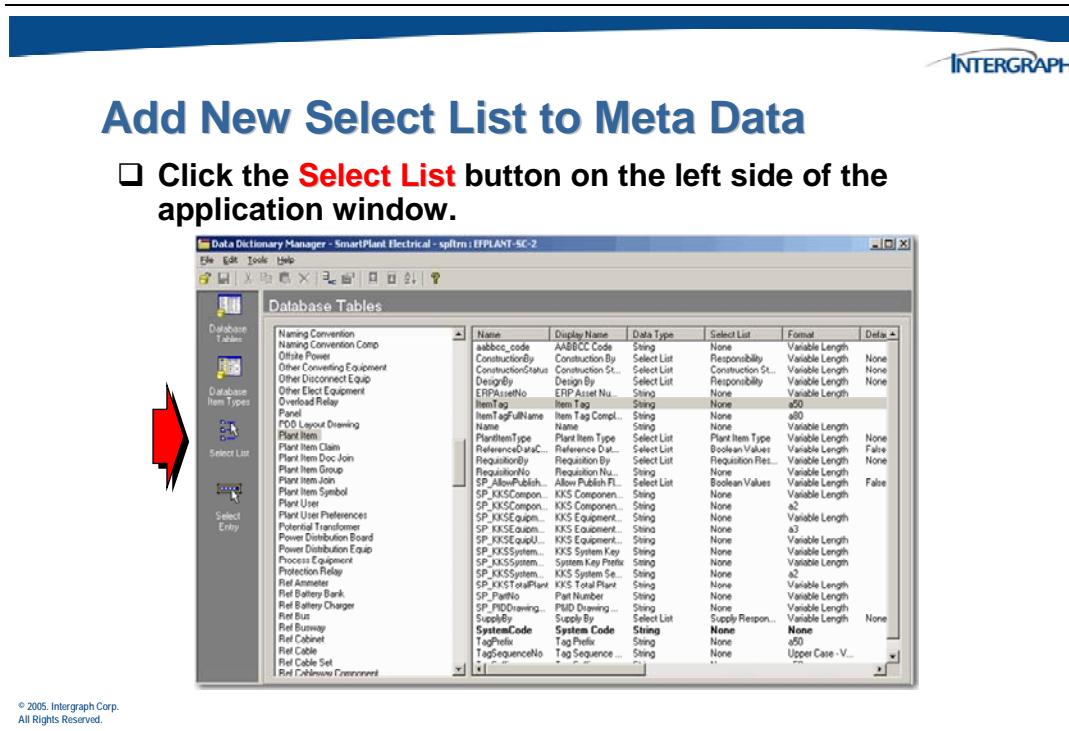
Complete the Chapter 10 – Activity 1 in the SmartPlant Foundation 2008 (4.2) Modeling and Mapping Activity workbook.

11.7 Adding a New Select List/Enum List

In the last section, we saw how to extend the SmartPlant Electrical data structure to add a new simple property and to how to map the new simple property into SmartPlant Foundation.

In this section, we will see how to do the same extension and mapping for our new complex property with its custom list of values.

To add a new property to SPEL that uses a select list, begin by launching the **Data Dictionary Manager**.



Before we can create our new property, we must first define the new pick list that it will use as its data type.

You can add a new select list by typing the name of the list in the blank row at the bottom of the **Select List** table.



Add New Select List to Meta Data

- Scroll to the bottom of the **Select List** table, and in the empty row at the bottom of the table, enter the name of the new select list.
- Select the **EngSys** row, and click the **Select Entry** button.



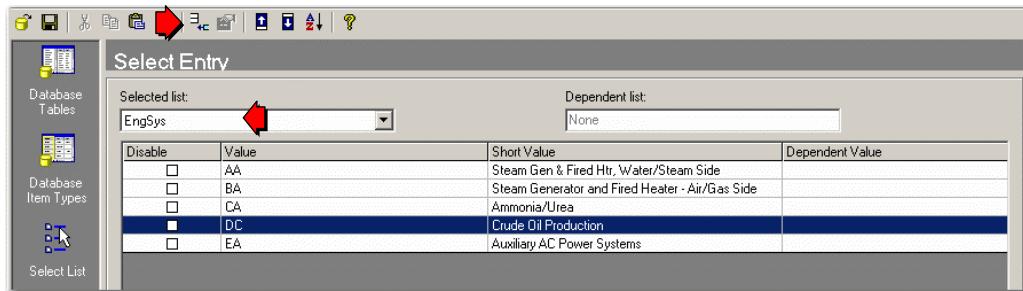
© 2005, Intergraph Corp.
All Rights Reserved.

All that is required to define a new select list is the name of the select list. To define entries for the new select list, click the **Select Entry** button. Add the first select entry to the list by typing it in the first row of the table. Click the **Add Row** button at the top of the window to add a new empty row and continue adding your new entries until you have defined them all.

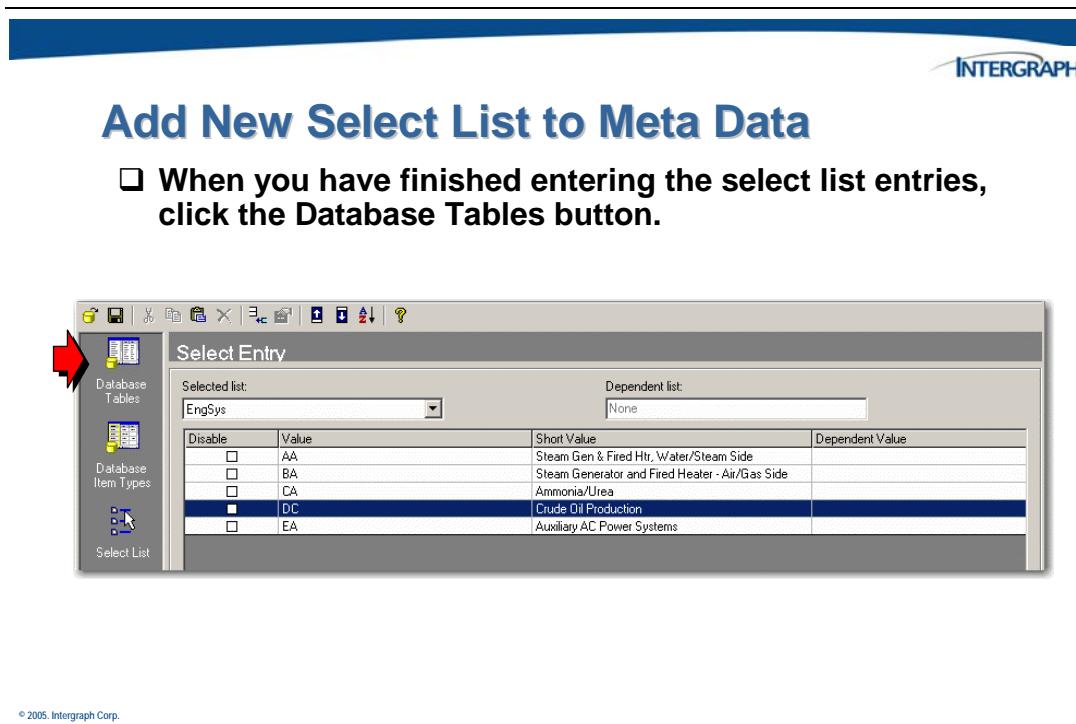


Add New Select List to Meta Data

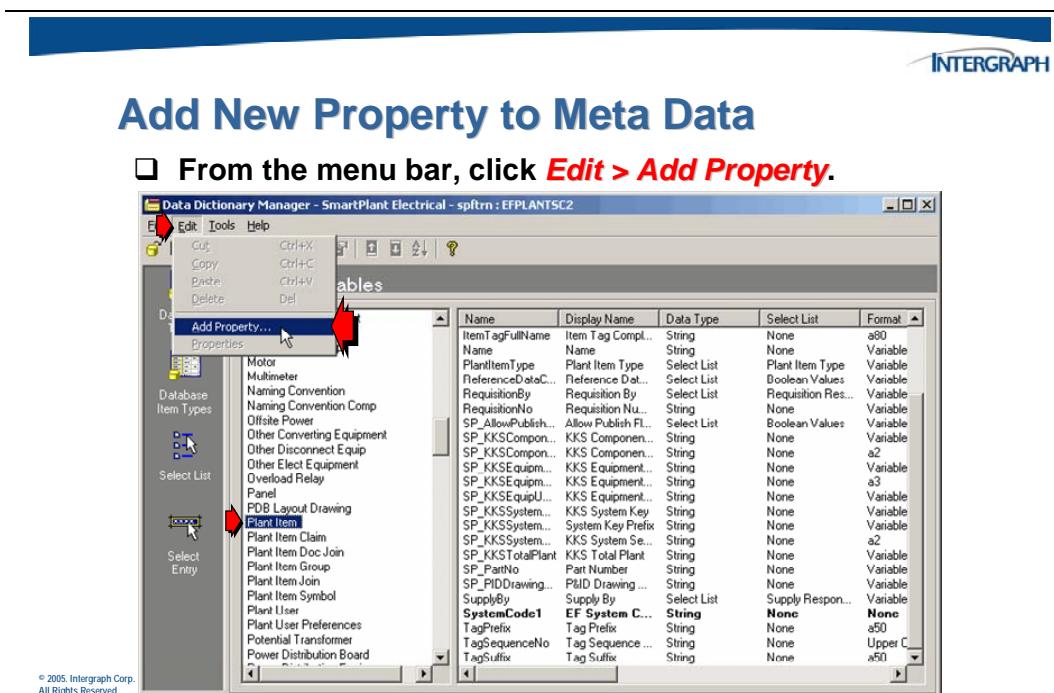
- The select list that was highlighted when you clicked the **Select Entry** button appears in the **Selected list** box.
- In this table, define the options that will be available in the enumerated list.



When you have defined all your entries, click the **Database Tables** button to return to the screen where you will add your new property that will use the select list you just created.



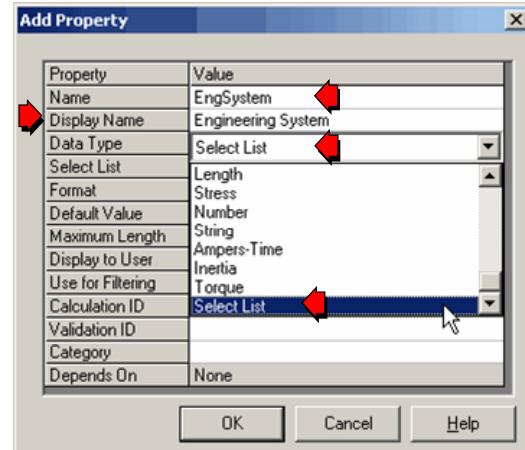
To create the property, select *Plant Item* in the list of database tables, and click **Edit > Add Property**.



Add the new **EngSystem** property to the *Plant Item* table. Choose **Select List** as the data type and then choose the appropriate list in the **Select List** box.

Add New Property to Meta Data

- In the **Add Property** dialog box, enter the **name** and **display name** for the property.
- In the **Data Type** list, choose **Select List**.

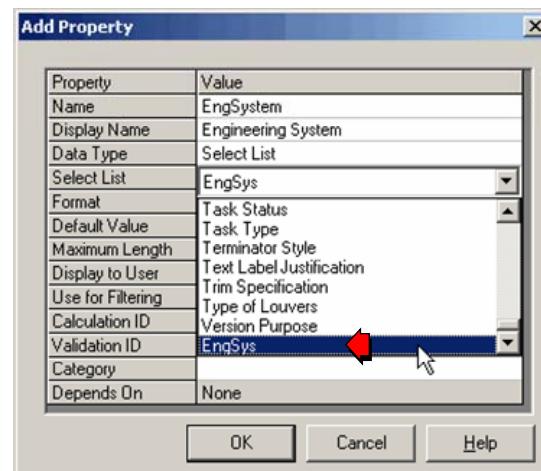


© 2005, Intergraph Corp.
All Rights Reserved.

The new property will use the custom select list, **EngSys**, defined earlier.

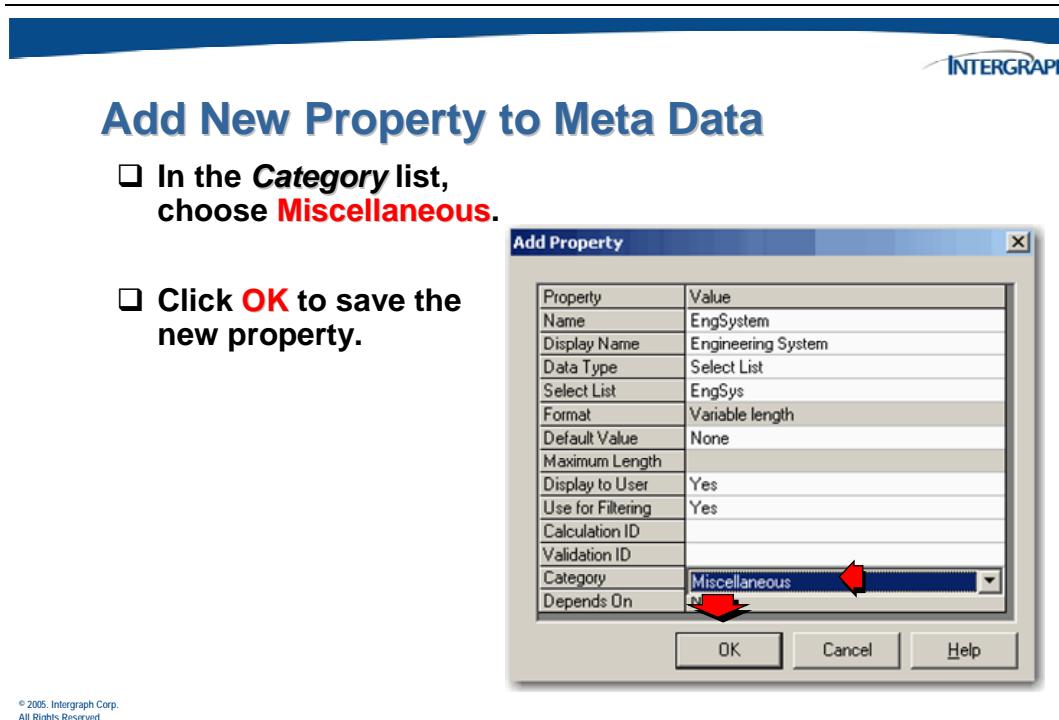
Add New Property to Meta Data

- In the **Select List** list, choose the name of the select list you want to associate with the property.



© 2005, Intergraph Corp.
All Rights Reserved.

Finish specifying the new property characteristics.



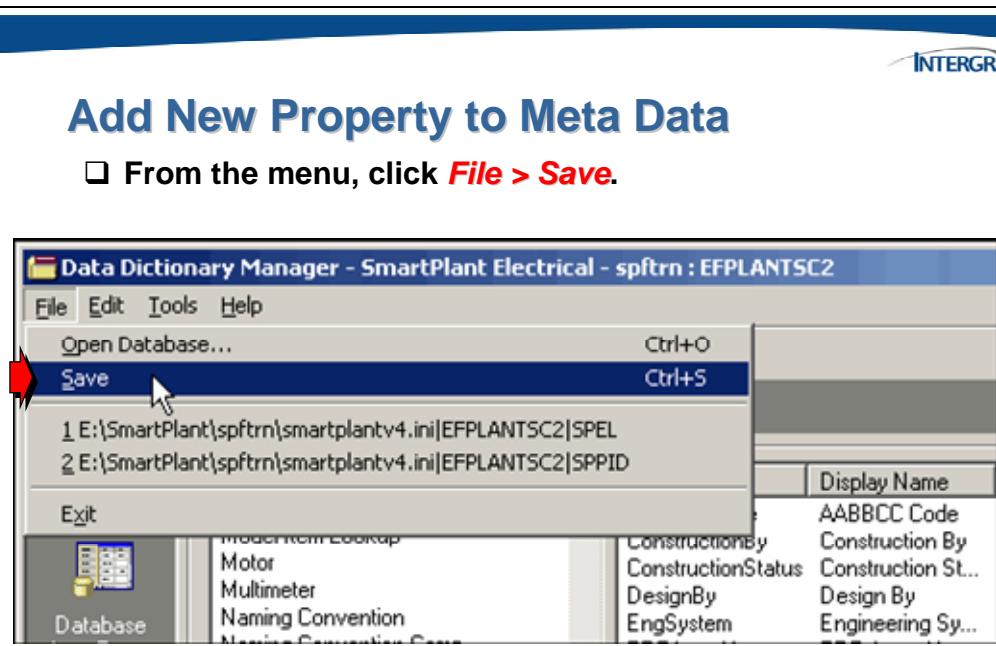
© 2005, Intergraph Corp.
All Rights Reserved.

When you click **OK**, the list of properties for the *Plant Item* table updates with the new property.

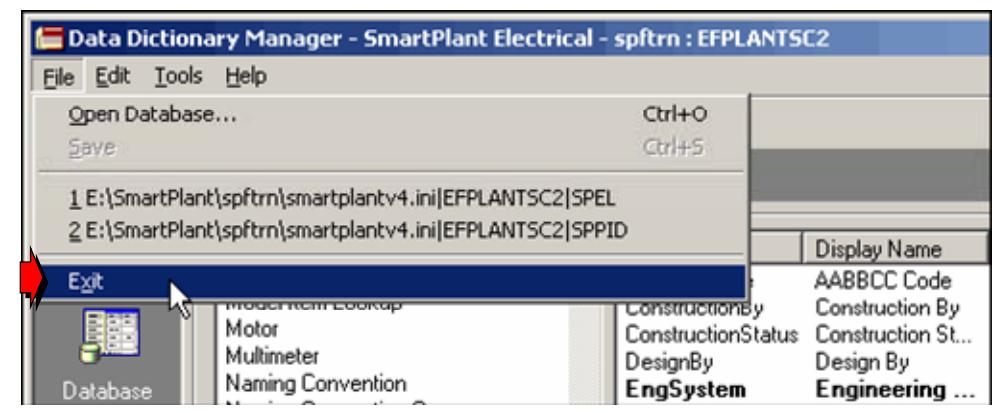
	Name	Display Name	Data Type	Select List	Format
Miscellaneous Element					
Model Item	EngSystem	Engineering Sy...	Select List	EngSys	Variable
Model Item Lookup	ERPAssetNo	ERP Asset Nu...	String	None	Variable
Motor	ItemTag	Item Tag	String	None	a50
Multimeter	ItemTagFullName	Item Tag Compl...	String	None	a80
Naming Convention	Name	Name	String	None	Variable
Naming Convention Comp	PlantItemType	Plant Item Type	Select List	Plant Item Type	Variable
Offsite Power	ReferenceData...	Reference Dat...	Select List	Boolean Values	Variable
Other Converting Equipment	RequisitionBy	Requisition By	Select List	Requisition Res...	Variable
Other Disconnect Equip	RequisitionNo	Requisition Nu...	String	None	Variable
Other Elect Equipment	SP_AllowPublish...	Allow Publish Fl...	Select List	Boolean Values	Variable
Overload Relay	SP_KKSCompon...	KKS Componen...	String	None	Variable
Panel	SP_KKSCompon...	KKS Componen...	String	None	a2
PDB Layout Drawing	SP_KKSEquipm...	KKS Equipment...	String	None	Variable
Plant Item	SP_KKSEquipm...	KKS Equipment...	String	None	a3
Plant Item Claim	SP_KKSEquipU...	KKS Equipment...	String	None	Variable
Plant Item Doc Join	SP_KKSSystem...	KKS System Key	String	None	Variable
Plant Item Group	SP_KKSSystem...	System Key Prefix	String	None	Variable
Plant Item Join	SP_KKSSystem...	KKS System Se...	String	None	a2
Plant Item Symbol	SP_KKSTotalPlant	KKS Total Plant	String	None	Variable
Plant User	SP_PartNo	Part Number	String	None	Variable
Plant User Preferences	SP_PIDDrawing...	P&ID Drawing ...	String	None	Variable
Potential Transformer	SupplyBy	Supply By	Select List	Supply Respon...	Variable
Power Distribution Board	SystemCode1	EF System C...	String	None	None

© 2005, Intergraph C...
All Rights Reserved

After adding the property, save the changes to the SmartPlant Electrical database.



Then click **File > Exit**, to close the Data Dictionary Manager.



11.8 Adding Enum Extensions to the Tool Schema

Once the new property and select list have been added to the SPEL database, use the Schema Editor to add the new property to the tool schema file, using SmartPlant Electrical metadata adapter.

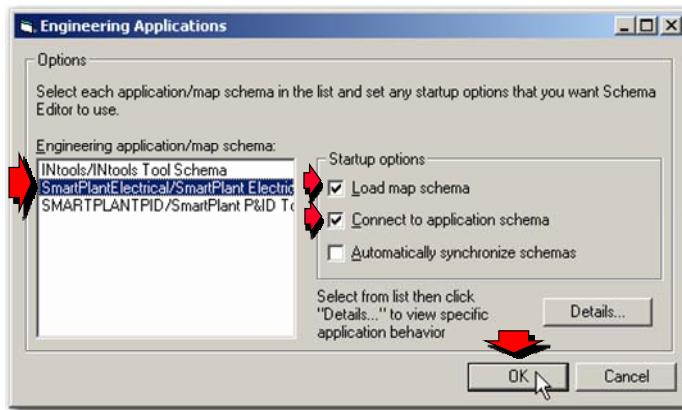
In the following example, we still have our CMF file checked out of SmartPlant Foundation Desktop Client, so we can launch the Schema Editor and connect to the saved session file.

However, if you have already configured the ICustomInterface to be available to the map classes that will use the EngineeringSys property that it exposes, you do not need to edit the CMF file. You can connect to any version of the CMF file, even if you do not have it checked out. But you must still have an up-to-date version of the CMF file open to create your mapping relationships.

From the Schema Editor, launch the SmartPlant Electrical tool map file and connect to the tool application schema.

Add New Property/Enum List to Schemas

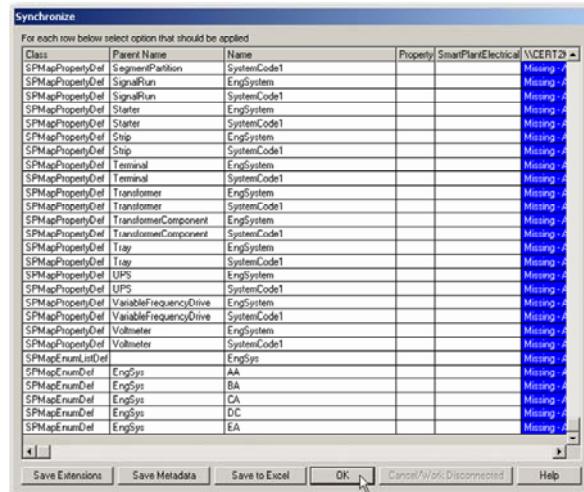
- Open Schema Editor and the session file, and launch the metadata adapter for SmartPlant Electrical.



The **Synchronize** dialog box shows the discrepancies between these two schemas and allows you to indicate what changes to make to synchronize the files.

Add New Property/Enum List to Schemas

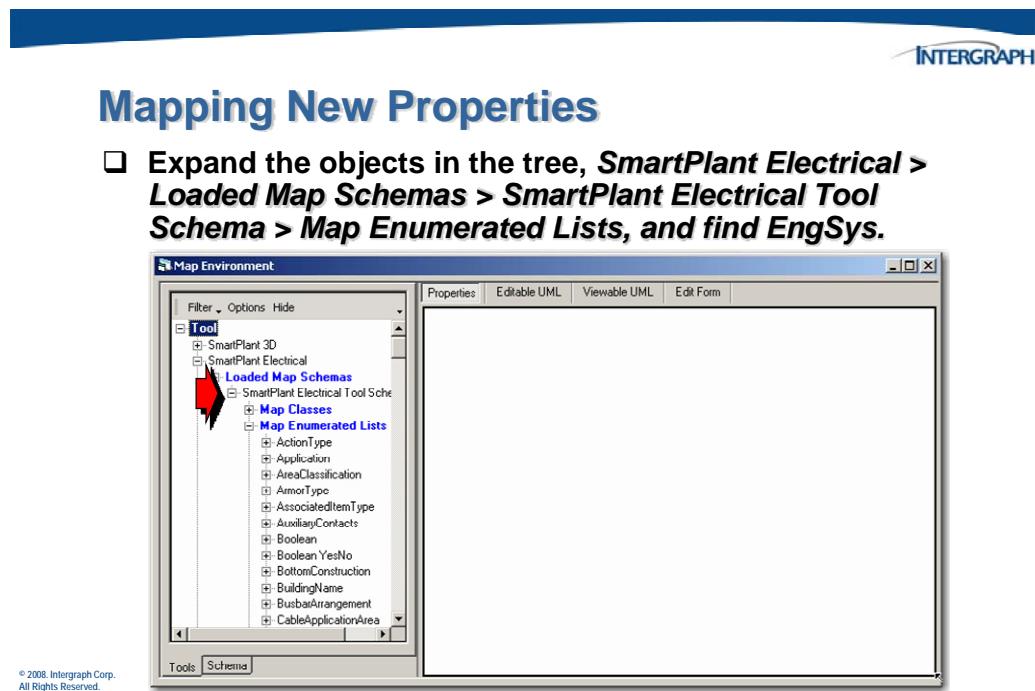
- Click **OK** on the **Synchronize** dialog to have the schema editor read the tool meta-schema and automatically add the new property/enum extensions to the tool map schema.



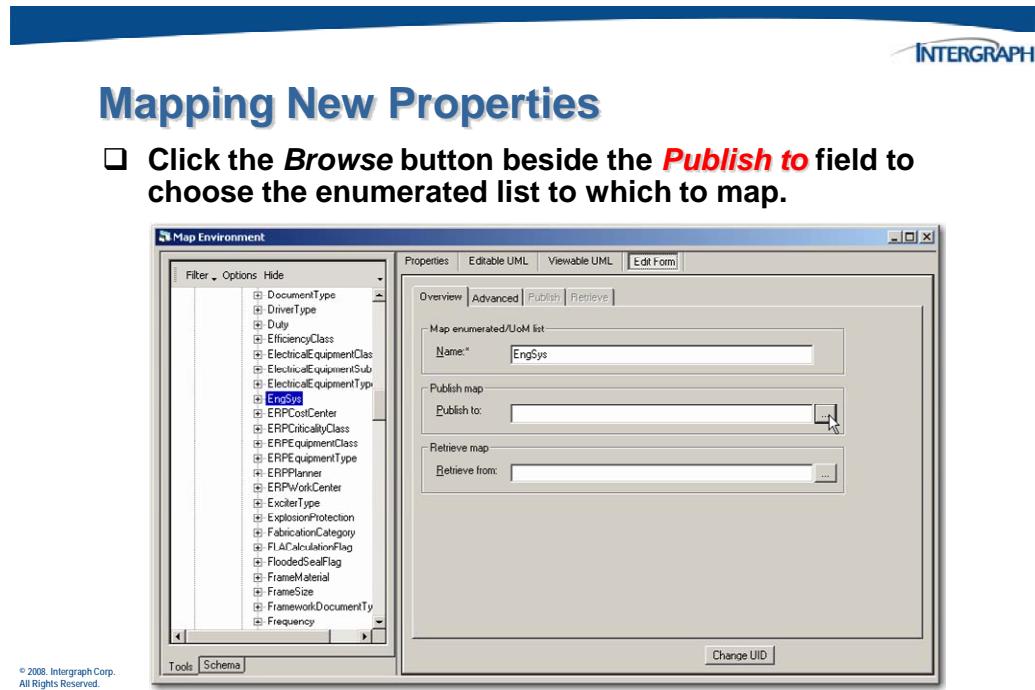
© 2008, Intergraph Corp.
All Rights Reserved.

11.9 Mapping Enumeration List Entries

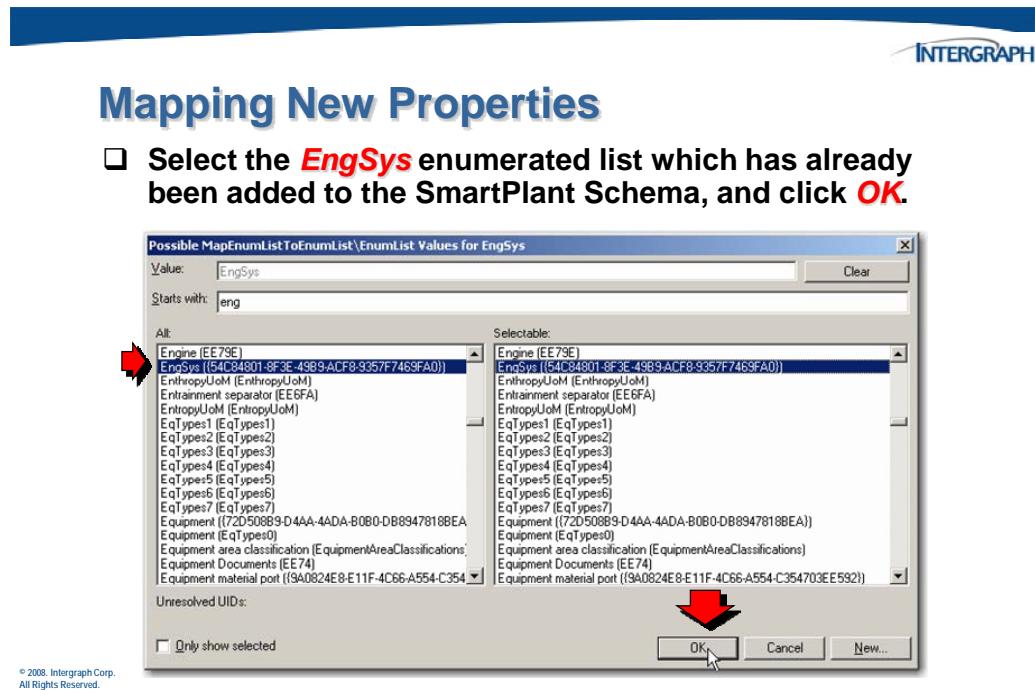
Next, we need to map the values available in the select list (enumerated list) to the enum enums as defined in the SmartPlant schema. Expand the tree in the **Map Environment** to show the **Map Enumerated Lists** under **SmartPlant Electrical**. Find the **EngSys** list and open the **Edit Form**.



On the **Overview** tab, click the *Browse* button beside the **Publish to** field to select the enumerated list in the SmartPlant schema to which you want to map the new list.



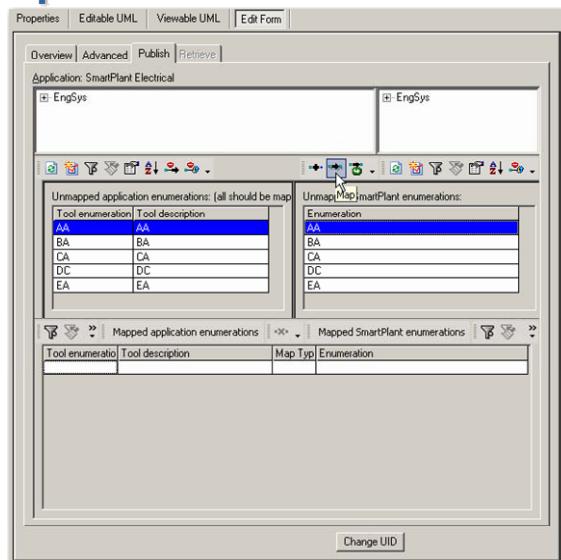
Find the **EngSys** enumerated list in the SmartPlant Schema, select it, and click **OK**.



The **Publish** tab is now available. From the **Publish** tab, select the list values on each side and map them one at a time.

Mapping New Properties

- Open the **Publish** tab.
- One at a time, map each of the select list entry values on the tool side to the applicable enum enum in the SmartPlant schema.



Verify that the mapping was completed successfully.

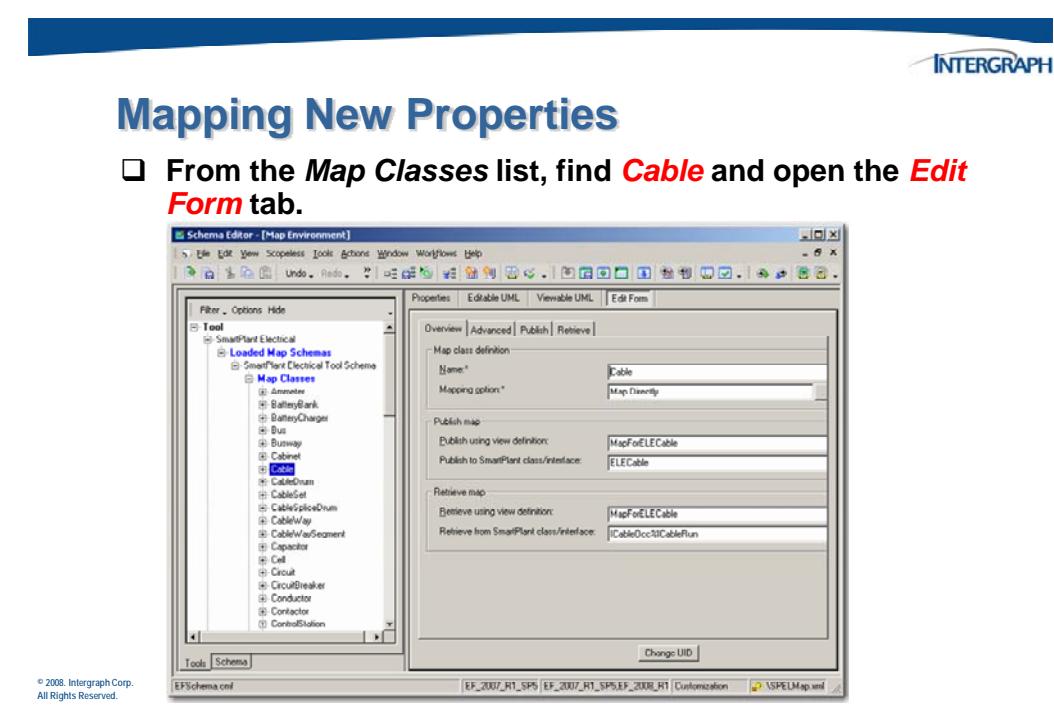
Mapping New Properties

- When you have finished mapping each enumerated list value, confirm the mapping in the bottom of the window.

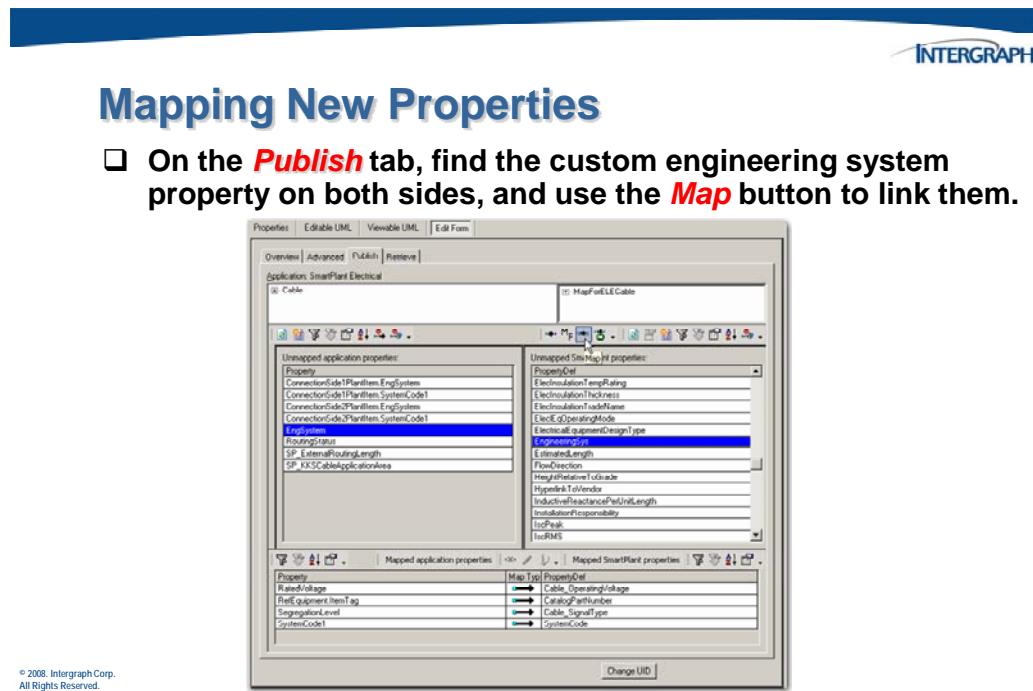
Mapped application enumerations		Mapped SmartPlant enumerations	
Tool enumeration	Tool description	Map Type	Enumeration
AA	AA	→	AA
BA	BA	→	BA
CA	CA	→	CA
DC	DC	→	DC
EA	EA	→	EA

11.10 Mapping the Complex Property

Once the list is mapped, you can then map the custom property, as well. From the **Map Environment** window, expand the SmartPlant Electrical map file to show the map classes. Find the **Cable** map class, and open the **Edit Form**.



On the **Publish** tab, find the **EngSystem** property in the tool map file and the **EngineeringSys** property in the SmartPlant schema, and use the **Map** button to create the mapping relationship.



Confirm the mapping relationship.

Mapping New Properties

- Verify that the property from the two schemas is mapped at the bottom of the form.

Property	Map Typ
RatedVoltage	PropertyDef
RefEquipment.ItemTag	Cable.OperatingVoltage
SegregationLevel	CatalogPartNumber
SystemCode1	Cable.SignalType
EngSystem	SystemCode
	EngineeringSys

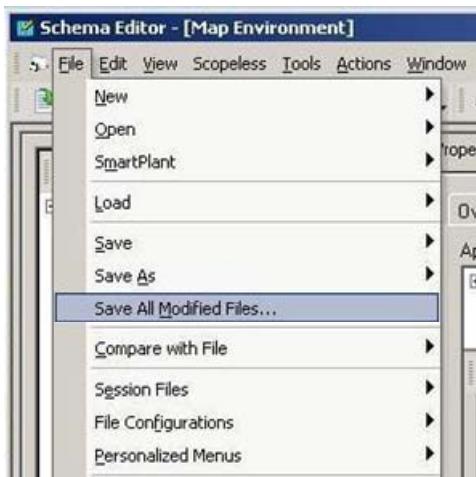
© 2008, Intergraph Corp.
All Rights Reserved.

Repeat these steps for all the map classes where you want to publish the new property.

11.11 Save the Mapping Changes

Again, the additions to the mapping information will need to be saved to the tool map file. You will be prompted to save the changes to the tool map schema.

Save Mapping Changes



- On the menu, click **File** > **Save All Modified Files** to save the additions to the SmartPlant schema and the mapping to the SPELDataMap file.

When prompted, save the changes to the SmartPlant Electrical Map file.

Save Mapping Changes

- Verify the tool map schema name for the mapping changes and choose Yes.



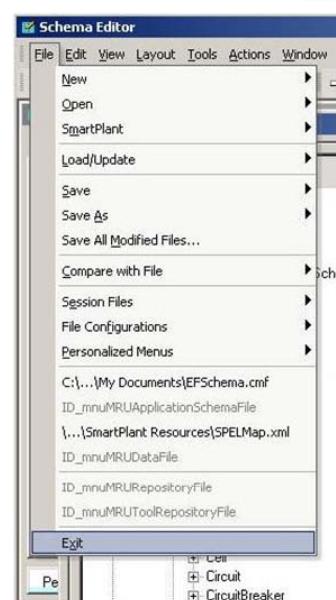
© 2008. Intergraph Corp.
All Rights Reserved.

Since you made changes to the SmartPlant schema, you are not prompted to save the CMF file.

You can now exit the Schema Editor.

Save Mapping Changes

- On the menu, click **File > Exit** to close out of the schema editor.



11.12 Test Mapped Properties

The next steps will be to test your changes. Retrieve a published P&ID document into SmartPlant Electrical, change information about a Motor in SPEL, and publish the updated data. Verify the published data is in the XML.



Test Mapped Properties

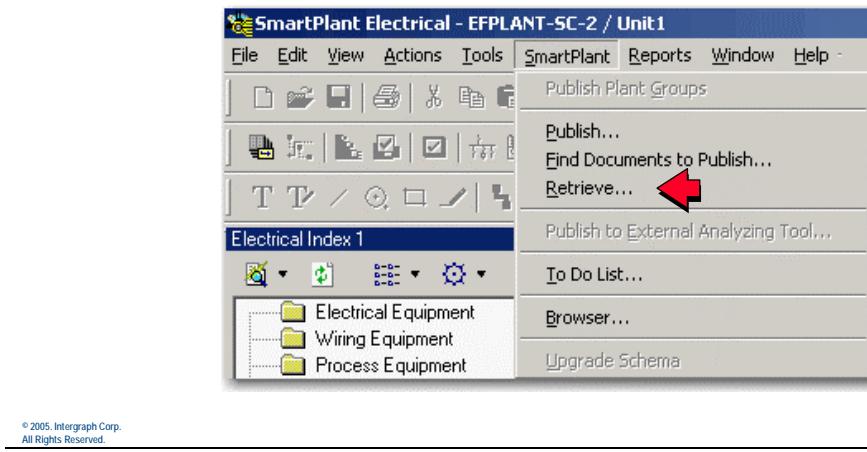
- Open SmartPlant Electrical and perform a Retrieve of a document published by P&ID.**
- Find a Motor, and change its system code.**
- Create a Document Report for publishing.**
- Revise the new Document.**
- Publish the new Document.**
- Verify that the data was published.**

Open SmartPlant Electrical, and retrieve a document that was published by SmartPlant P&ID.



Retrieve the Motor Properties

- Open SmartPlant Electrical, and retrieve the latest P&ID-published document.

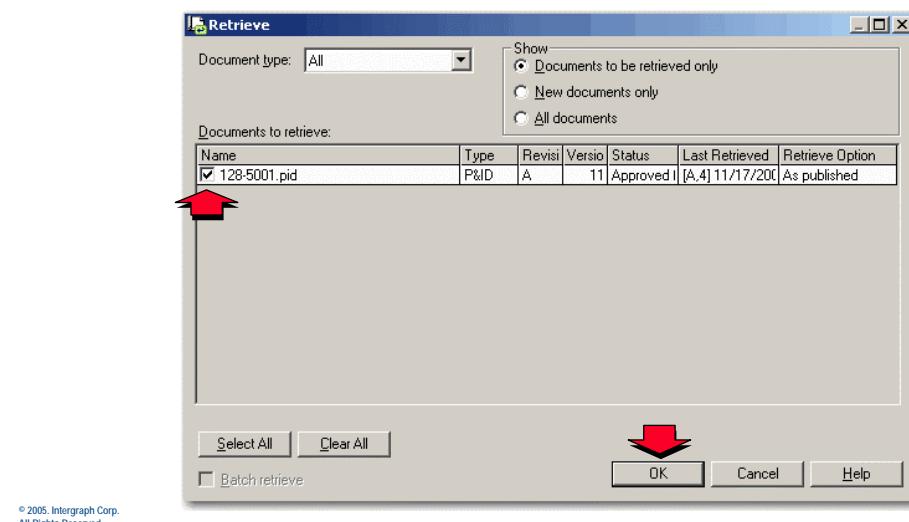


On the *Retrieve* dialog box, select a document to retrieve from SPPID.

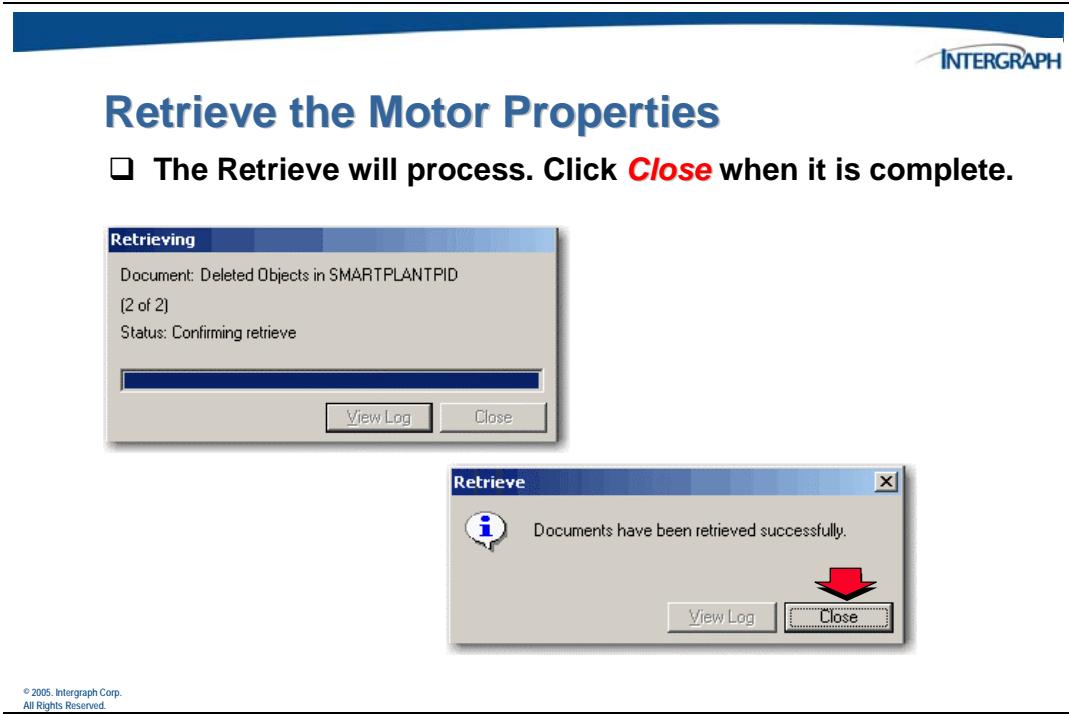


Retrieve the Motor Properties

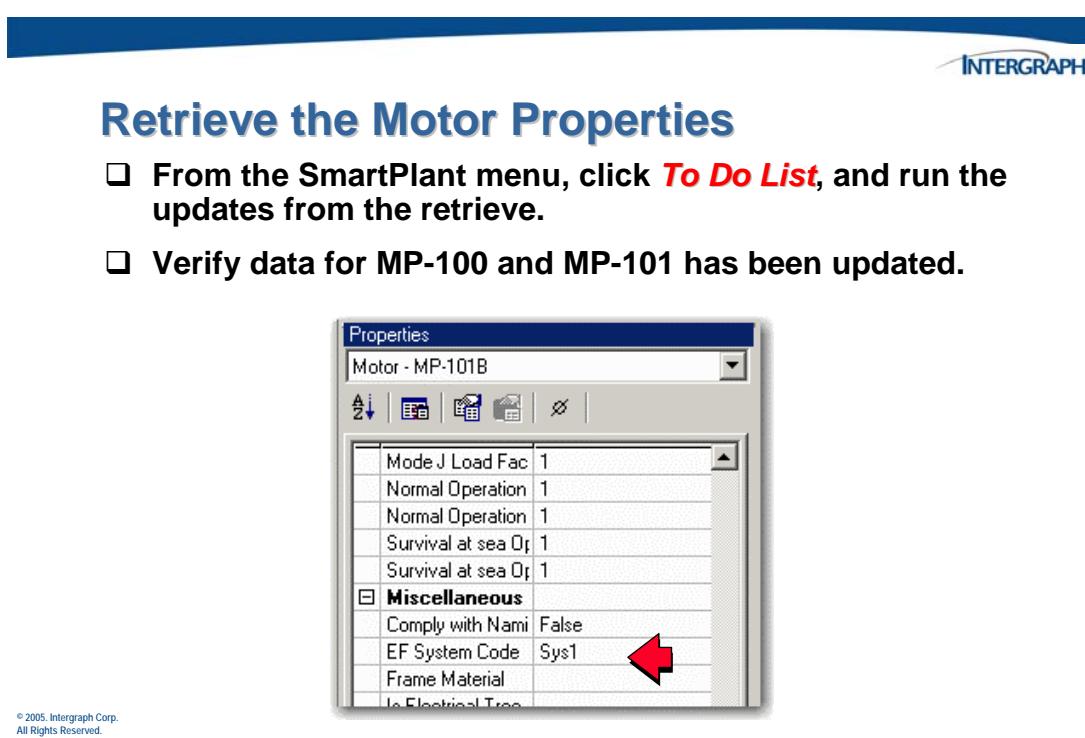
- Check the latest P&ID document, and click **OK**.



The progress message box will appear, followed by another information window when the retrieval process is complete.



Open the **To Do List** and run the updates required as a result of the retrieve.

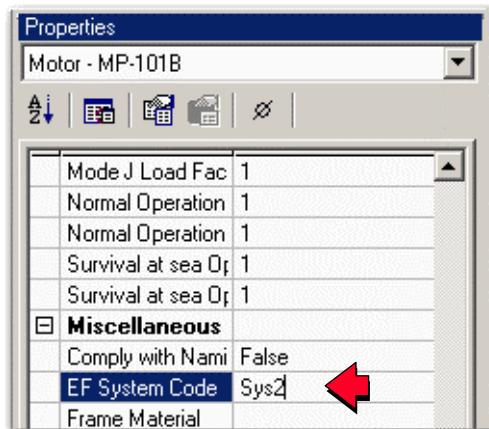


Change the value for *EF System Code*.



Retrieve the Motor Properties

- Change the ***EF System Code*** value to **Sys2**.

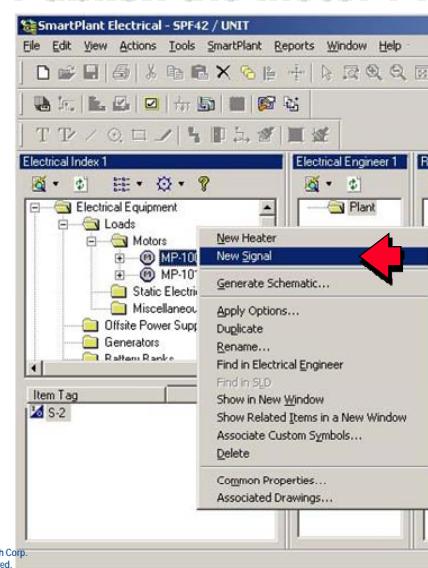


© 2005 Intergraph Corp.
All Rights Reserved.

Create new signals for each of the motors from SmartPlant P&ID.



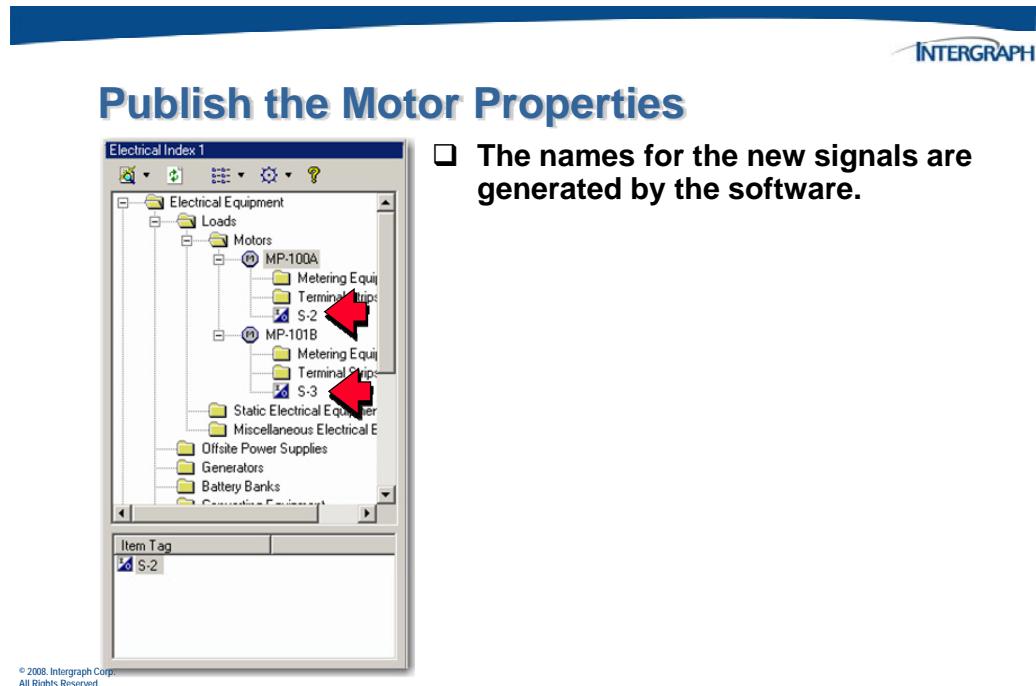
Publish the Motor Properties



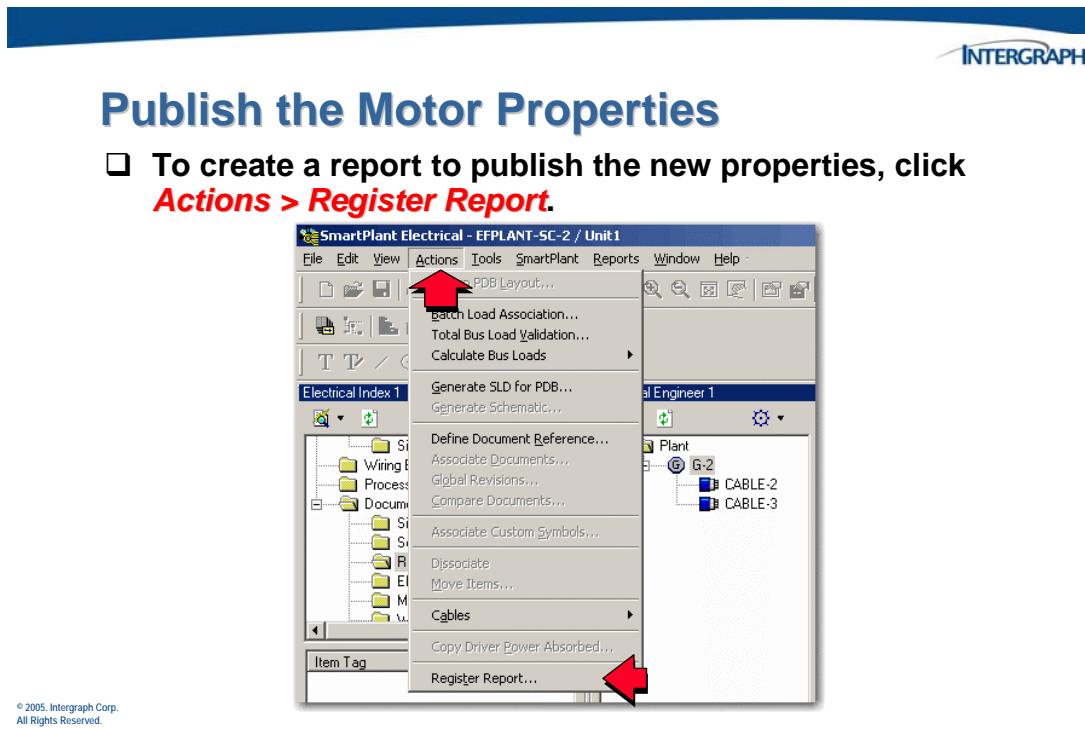
- Create a new signal for each motor retrieved from P&ID.
- Right-click on the motors in the ***Electric Index*** window under **Loads**.
- From the shortcut menu, click the ***New Signal*** command.

© 2008 Intergraph Corp.
All Rights Reserved.

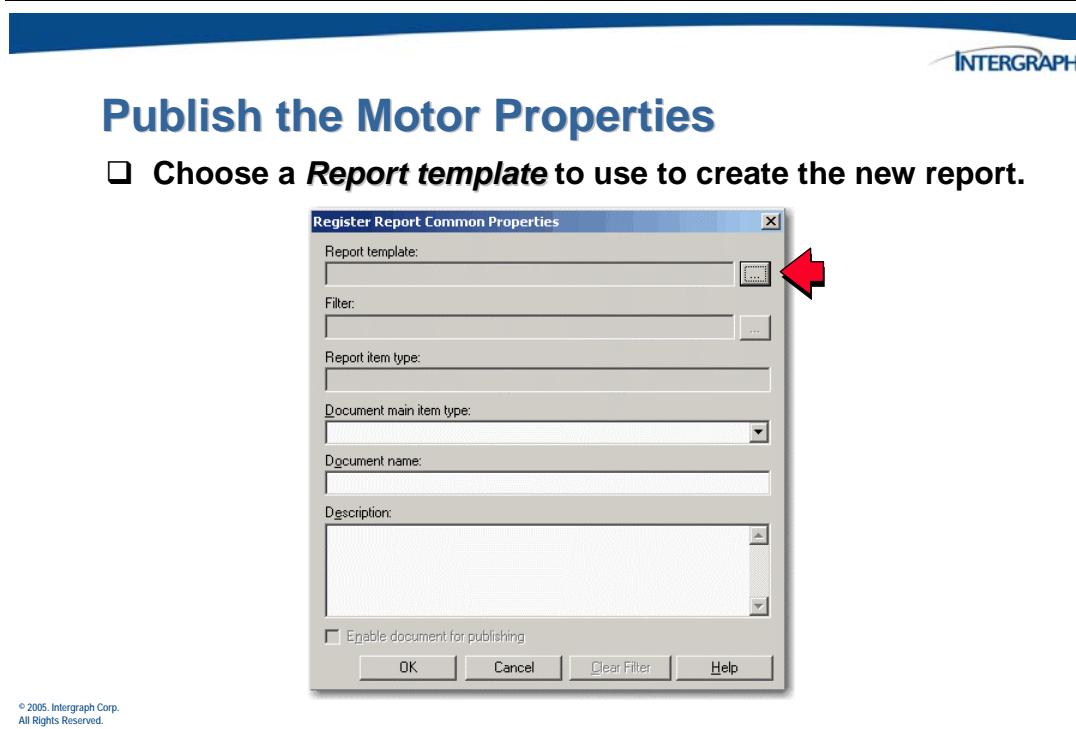
The new signals appear in the tree beneath the associated motor.



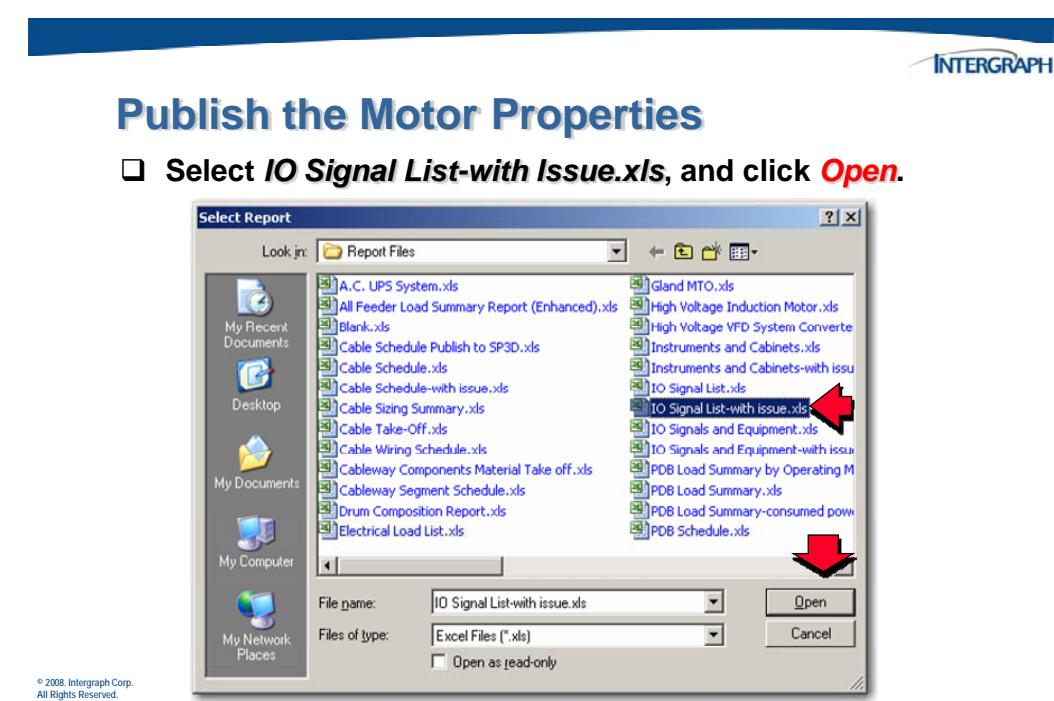
Create a report to publish from SmartPlant Electrical.



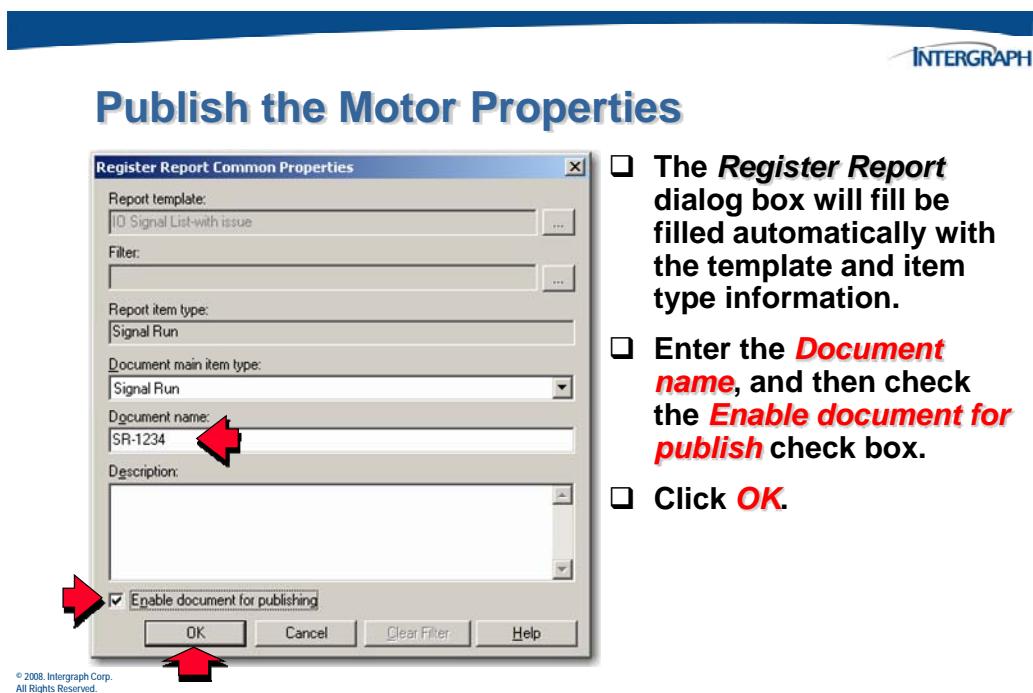
Click the browse button by the report template field to select a template to use.



Choose the *IO Signal List-with issue* template from the list.

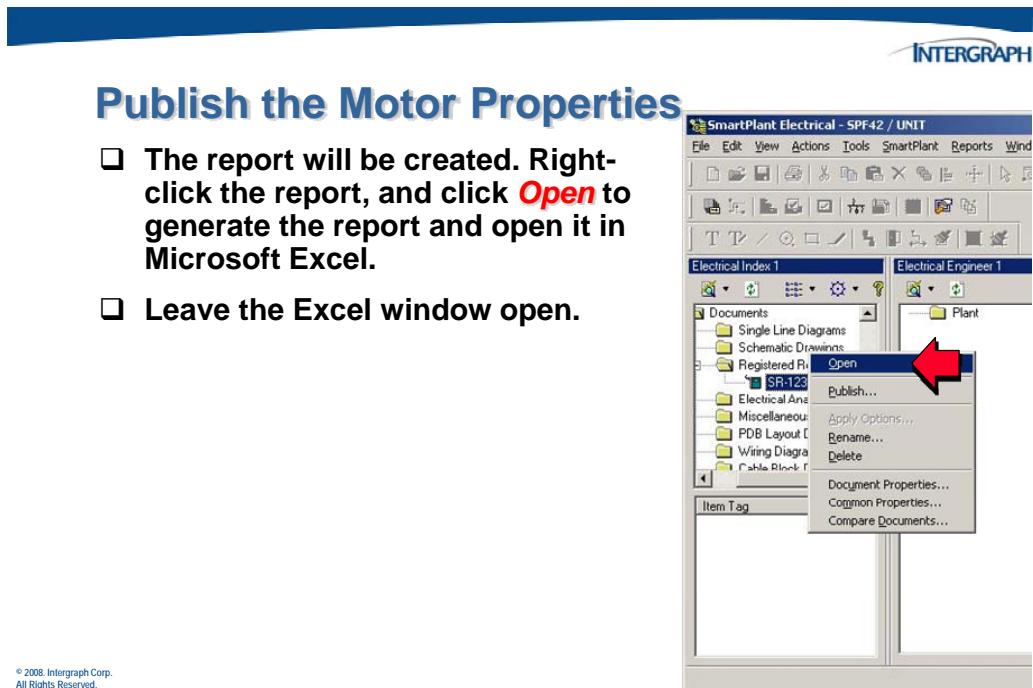


Provide a name for the report, and make sure the *Enable document for publishing* check box is checked.

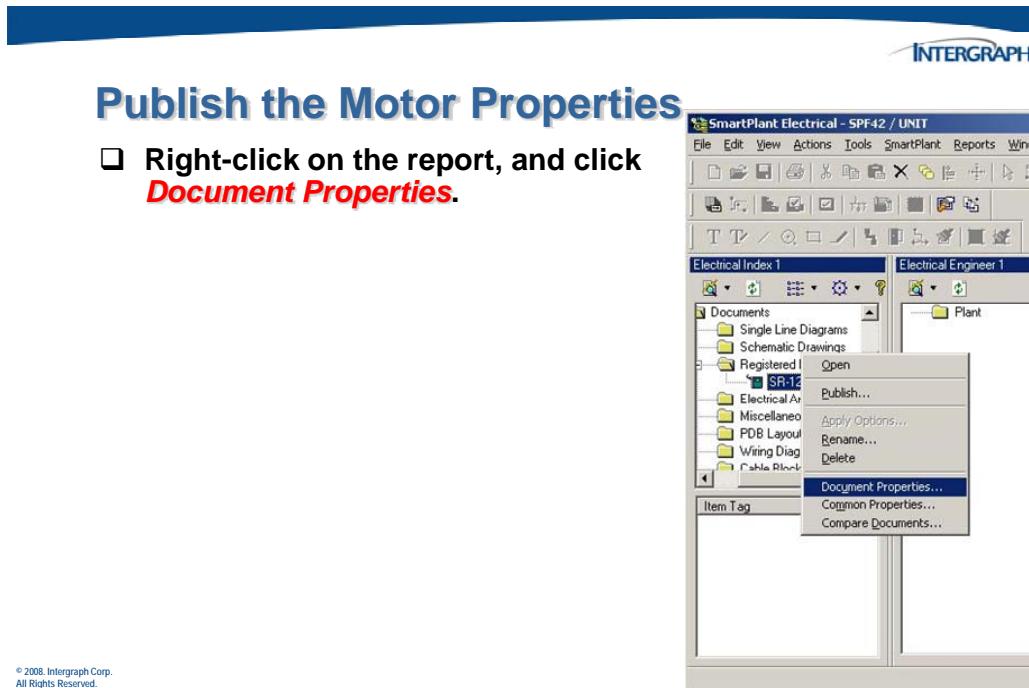


- ❑ The **Register Report** dialog box will be filled automatically with the template and item type information.
- ❑ Enter the **Document name**, and then check the **Enable document for publish** check box.
- ❑ Click **OK**.

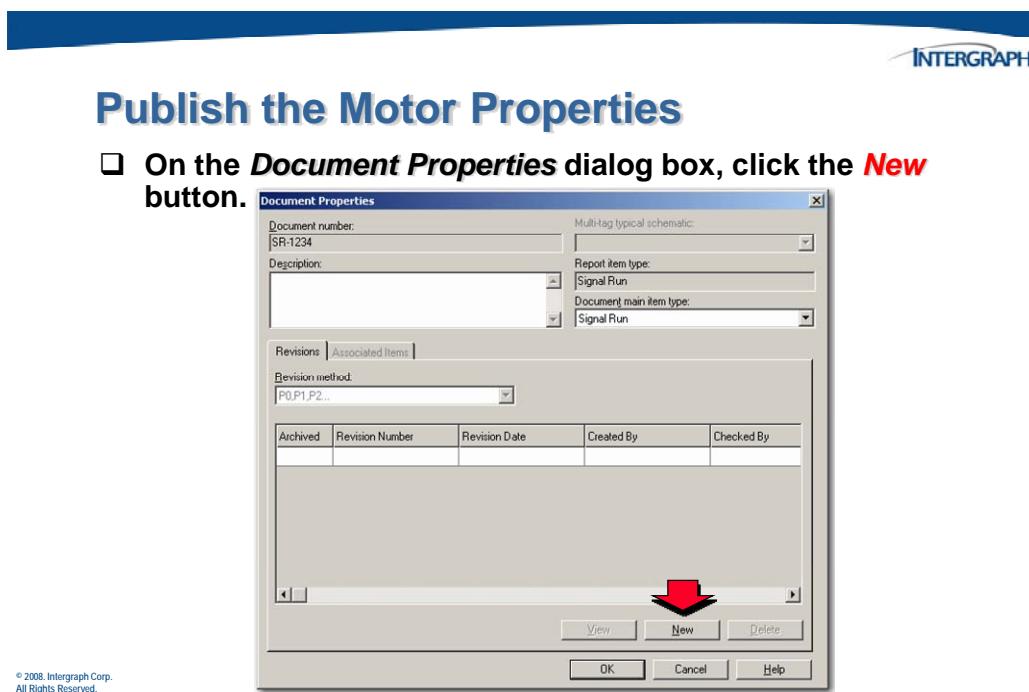
Open the report in Microsoft Excel.



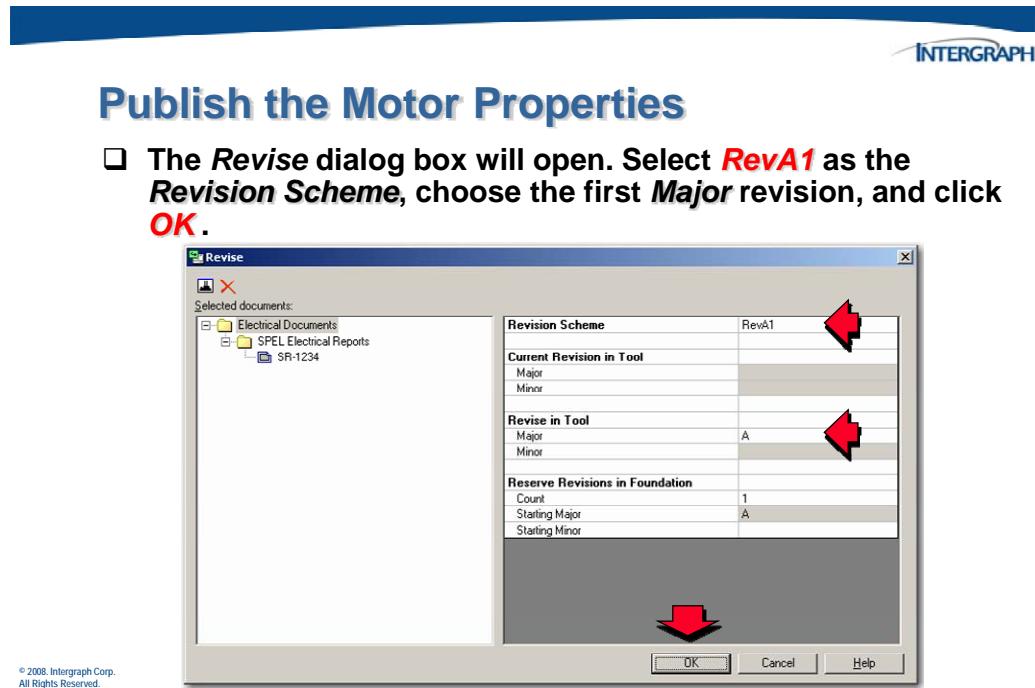
Open the **Document Properties** dialog box.



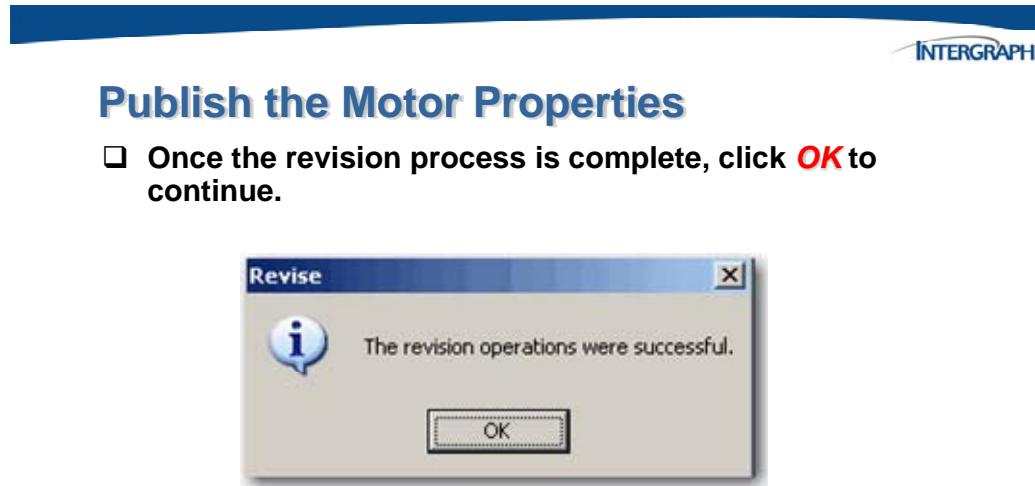
Click **New** on the **Document Properties** dialog box to open the **Revise** dialog box.



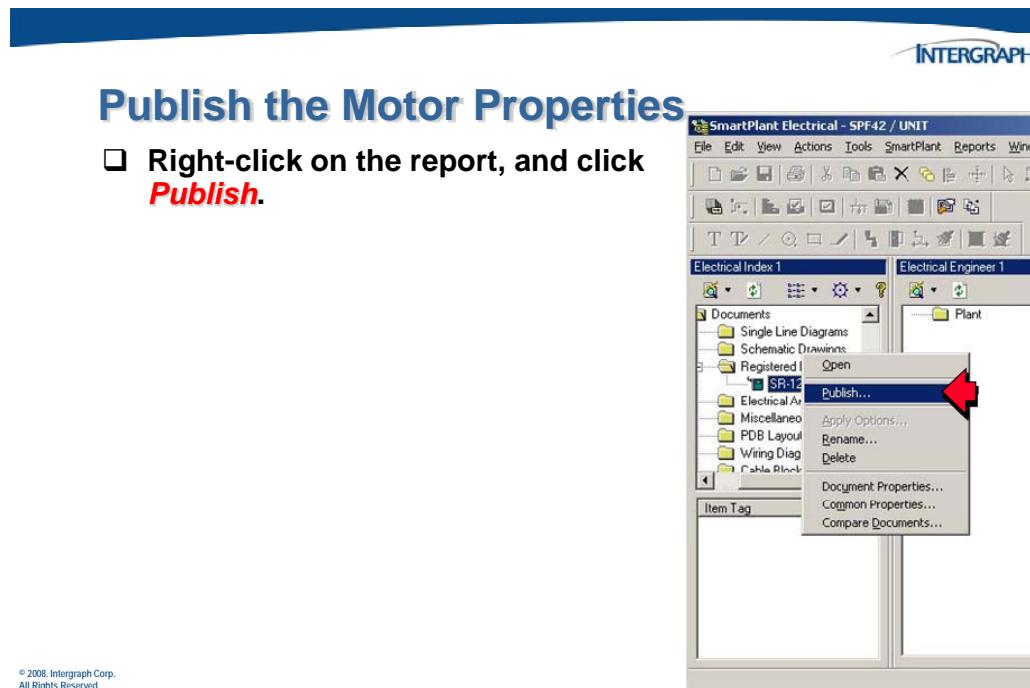
On the **Revise** dialog box, select a **Revision Scheme** and specify the first major revision.



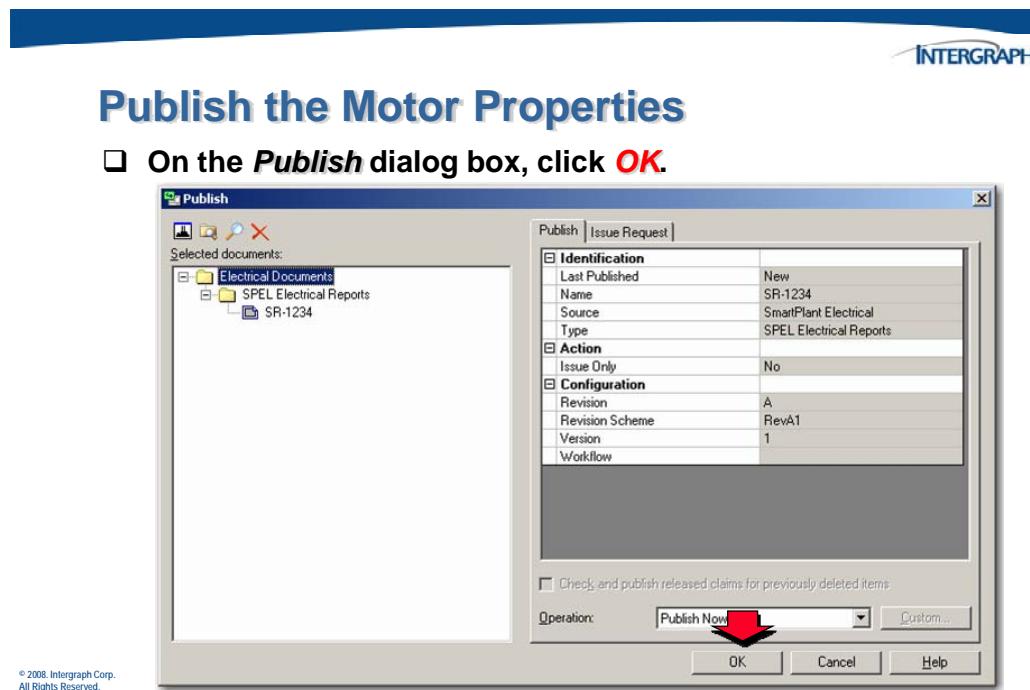
A message box will indicate the progress of the revision creation.



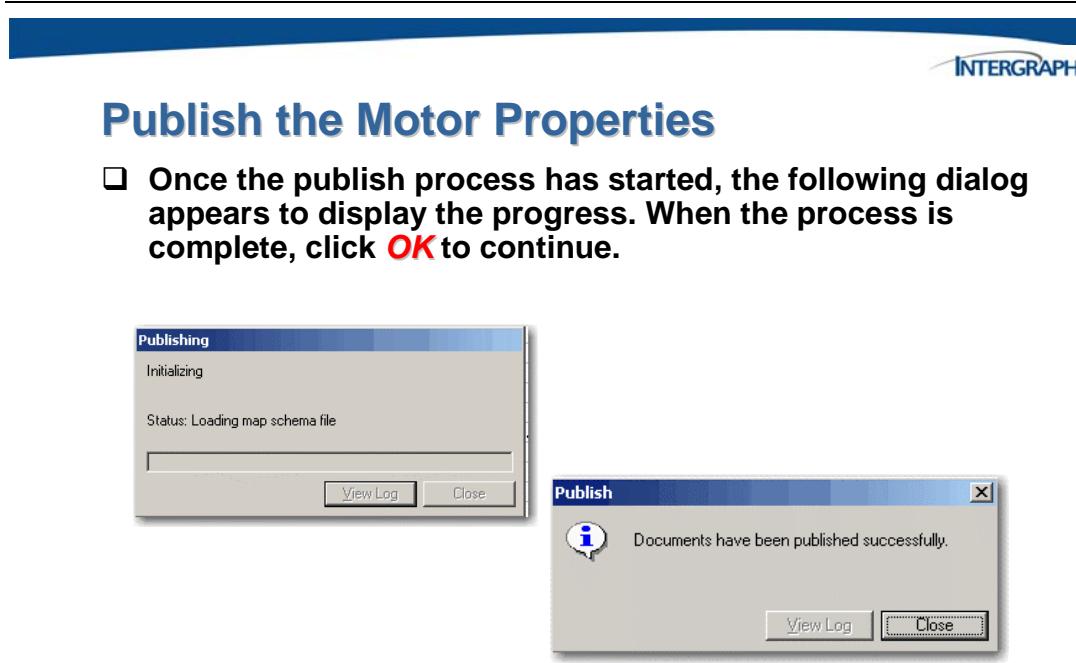
Once you have created a new revision, publish the report.



Click **OK** on the **Publish** dialog box.



A message dialog box will indicate the progress of the publish process.



© 2005, Intergraph Corp.
All Rights Reserved.

Confirm that the new information was published in the XML file.

Publish the Motor Properties

- In Windows Explorer, open the published XML file, and verify that the properties were published with the new value **Sys2**.

```

<IPart />
</ELEElectricGenerator>
- <ELEElectricMotor>
  <Object UID="600E4205E7E042C69F118025B59B6794" Name="MP-100A" />
  <IPBSItem SystemCode="Sys2" />
  <IPBSItemCollection />
  <IPlannedMatl />
  <INonDrawingItem />
  <IEquipment EqType0="@{47BF0267-DD41-4E1A-9B41-C4B714C8FF92}" EqType2="@EE79D" EqType1="@EE792" />
  <IEquipmentOcc />

```

© 2005, Intergraph Corp.
All Rights Reserved.

11.13 Activity 2 – Adding and Mapping a Complex Property

Complete the Chapter 11 – Activity 2 in the SmartPlant Foundation 2008 (4.2) Modeling and Mapping Activity workbook.

C H A P T E R

12

Mapping with SmartPlant 3D

12. Mapping for SmartPlant 3D Overview

When the schema in SmartPlant is extended, such as when new classes, properties, or select list entries are added, you or an administrator must react to the change. You must update the mapping data in SmartPlant 3D. Mapping data is necessary for the publish and retrieve operations, as well as for correlating design basis objects with model objects. You also must update the Catalog Schema database in SmartPlant 3D, if the new information does not already exist in the database.

12.1 Adapters

Each authoring tool has an adapter that processes information during the publish and retrieve operations. One of the adapter's functions is to map information between SmartPlant and the particular authoring tool. SmartPlant 3D currently has two adapters: one for retrieving and another for publishing. The retrieve adapter for SmartPlant 3D requires that each document type have a registry entry present. These registry entries define the map file used and define callback objects that perform additional processing. It is possible that each document type could have its own map file. The retrieve adapter uses a map file, which is defined by tool schema files. The publish adapter uses a tool schema file (generated from the SmartPlant 3D Catalog Schema and supplemented with mapping information) and a schema called the P3DComponent Schema, which is derived from the SmartPlant Schema.



SP3D Mapping Adapters

- SmartPlant 3D currently has two adapters. One for retrieving, and another for publishing.**
- The adapter used for retrieving requires that each document type that will be retrieve have a registry entry that defines what map file will be used to retrieve that type of document.**
- The publishing adapter uses a standard tool map schema file like the other SmartPlant Enterprise authoring tools.**
- Both adapters interact with the P3DComponent component schema created from the SmartPlant Schema.**

12.2 Map Files

For retrieve, the delivered map file is `DesignBasis_map.xml`. For publish, the map file is `SP3DPublishMap.xml`. The map file for correlating design basis objects with SmartPlant 3D is `SP3DToEFWClassMap.xml`.

Notes:

- The software does not currently provide a tool to manage change in the `SP3DToEFWClassMap.xml` file. You should save a copy of this file before you make changes to it. All the map files are located in the symbol share directory that is associated with the catalog for the site.
- To ensure piping properties are passed correctly from SPP&ID the value **SmartPlant 3D** must be set for the **Use Piping Specification in Options Manager**.



SP3D Mapping Map Files

- ❑ The Retrieve map file is `DesignBasis_map.xml`.
- ❑ For publish, the map file is `SP3DPublishMap.xml`.
- ❑ The map file for correlating design basis objects with SmartPlant 3D is `SP3DToEFWClassMap.xml`.

12.2.1 General Mapping Information

Mapping for retrieve is configured from a set of component schemas contained in the EFSchema.xml file on the SPF server. The information in these component schemas is processed into a SmartPlant 3D schema package and a map file. The schema package is loaded into the SmartPlant 3D Catalog Schema (metadata database), and the map file is made available to each SmartPlant 3D client. Modification to the map file is automatic and does not require manual editing. For retrieve, the Generate Design Basis tool provided by Core does the mapping. The **Generate Design Basis** tool updates the map file that is, in turn, used to generate the metadata package for the design basis objects. During a retrieve operation, the map file is used to translate the data received from SmartPlant into the form defined in the Catalog. For example, attributes in a SmartPlant document are typically passed as strings. The corresponding attribute in SmartPlant 3D could be a string, a long, a double, a codelist, or other type. The map file could translate a string, such as TRUE, to the Boolean value True in SmartPlant 3D.

During a retrieve operation, each object is identified by its UID. A one-to-one relationship exists between an object and its UID. Once the object is identified and brought into memory, the mapped properties are stored. You might also wonder if you can map attributes on the document object itself. This mapping is possible as long as the attributes are defined in both the SmartPlant Schema and the SmartPlant 3D Catalog Schema. Typically, a document object implements the IDocument interface in addition to other special purpose interfaces.

For the mapping from the design basis objects to the SmartPlant 3D objects, the SmartPlant Schema is not used. SmartPlant 3D uses a custom mapping definition. The filename for this definition is SP3DToEFWClassMap.xml.



SP3D Mapping Information

- Component Schemas from the other applications are loaded into the catalog database using the Generate Design Basis.
- The base tool schema is defined by the SmartPlant 3D Catalog Schema.
- During a retrieve operation, each object is identified by its UID, because of the one-to-one relationship between an object and its UID.
- If SmartPlant 3D will be retrieving data from SPPID, set the ***Use Piping Specifications*** entry in SPPID's Option Manager to SmartPlant 3D rather than PDS.

12.2.2 Publish Mapping

Mapping for publish is done by adding information to the tool schema. The base tool schema is defined by the SmartPlant 3D Catalog Schema. Then, this base tool schema is modified using utilities provided by the Core, including a merge utility and an extract utility. The Schema Editor from SmartPlant is also used in the modification. The net result of the modifications is that the tool schema includes applicable mapping information. During a publish operation, the tool schema (with added mapping information) and the SP3DComponent (derived from the SmartPlant Schema) are used to convert SmartPlant 3D objects to SmartPlant objects.

When you publish documents, the software:

- Publishes a visual representation of the document that you can view without SmartPlant 3D. For drawings, this is an Intergraph proprietary file, called a RAD file (.sha). For reports, the viewable file is a Microsoft Excel workbook. You can review and mark-up the visual representation of the document using SmartPlant Markup or SmartSketch.
- Places the published XML file and any viewable files in the appropriate SmartPlant Foundation vault. This XML file can be retrieved when users are in other authoring tools.

SmartPlant 3D receives notification when the publish operation is complete. The software stores the XML file in the appropriate location and loads the data into the database. The XML file can then be retrieved as published data by other tools, such as SmartPlant Electrical.



SP3D Mapping - Publish

- The Publish mapping for SP3D is managed like the other authoring tools.**
- Once changes have been made to the tool schema, the metadata adapter can synchronize the tool schema and the map file used for publish.**
- Once these files are synchronized, the SmartPlant 3D properties can then be mapped with SmartPlant Schema using the Schema Editor.**
- When you publish from 3D,**
 - The software publishes a visual representation of the model that can be viewed using either SmartPlant Markup, SmartSketch, or SmartPlant Review.**
 - Reports are published in Microsoft Excel workbook format.**

© 2008. Intergraph Corp.
All Rights Reserved.

When you work in an integrated environment with SmartPlant Enterprise, you must publish documents containing the drawing data and relationships before other authoring tools can share this information. You can publish your documents within the Drawings and Reports task.

The **Publish** command is available for the following document types:

- 3D Model Data (SmartPlant Review file type)
- 3D Cable Data (SmartPlant Review file type)
- Orthographic Drawings (viewable file with links to data)
- Piping Isometric Drawings (viewable file with links to data)
- Reports (viewable file with links to data)

The viewable files created when you publish drawings and reports provide relationship links to the 3D Model Data. You must also publish the 3D Model Data to provide the navigation between the viewable files and the 3D Model Data.



SP3D Mapping - Publish

- Publish is available for the following document types in SP3D:**
- 3D Model Data (SmartPlant Review file type)
 - 3D Cable Data (SmartPlant Review file type)
 - Orthographic Drawings (viewable file with links to data)
 - Piping Isometric Drawings (viewable file with links to data)
 - Reports (viewable file with links to data)

12.2.3 Retrieve Mapping

During a retrieve operation, the adapter assumes that the data it receives is synchronized with the metadata defined in the Catalog Schema. The map file cannot have a one-sided definition. That is, a class or property must exist both in the SmartPlant Schema and in the SmartPlant 3D Catalog Schema. The map file is responsible for making this connection between the two definitions. An important limitation of the retrieve process is that shared objects (objects that contain information from two or more tools) are completely deleted when a single tool deletes an object.



SP3D Mapping - Retrieve

- The retrieve map file must be managed manually. The metadata adapter will not synchronize the SP3D retrieve map file with the tool schema.**
- Additionally, the mapping must be managed manually. In the map file, you will need to map the tool property to the appropriate SmartPlant Schema property**
- An important limitation of the retrieve process is that shared objects (objects that contain information from two or more tools) are completely deleted when a single tool deletes an object.**



SP3D Mapping - Retrieve

- ❑ **Mapping for retrieve carries the following limitations:**
 - The mapping is done against the P3D Component only.
 - The Schema Editor does not currently show the properties for mapped edge definitions. Any mapping spreadsheet generated using the Schema Editor will not have all the mappings.
 - Some properties are set in the publish code and cannot be changed in this release.

© 2008, Intergraph Corp.
All Rights Reserved.



SP3D Mapping – SP Schema Limitations

- ❑ For enumerate lists that will be retrieved into SmartPlant 3D, values (codelist entries) must be greater than 10000 and cannot have duplicated indices within a single list.
- ❑ Duplicated values in another level within a hierarchical codelist is permissible.
- ❑ In rare cases, if an entry is removed from the SmartPlant Schema, you may need to edit the retrieve map file to remove the deleted entry.

© 2008, Intergraph Corp.
All Rights Reserved.

12.3 Extending the Application Schema

The first step of extending the SP3D schema is to create or modify the Excel spreadsheets that will include the new properties and code lists.



Extending the SP3D Schema

- In order to extend the SP3D schema to add custom properties and custom enumerated lists, you will load the changes through a bulkloader utility using Excel spreadsheets.
- Intergraph delivers a number of samples files that you may use with the SmartPlant 3D application.
- For example to add the new fluid code values, create a copy of the **AllCodeLists.xls** spreadsheet delivered with the SP3D software.
- Open the spreadsheet, locate the **Fluid Code** tab, locate the **Gas, Fluid System**, and then insert a row under this Fluid System.

© 2008. Intergraph Corp.
All Rights Reserved.

Make a copy of the original **AllCodesList.xls**, which is in the directory **E:\Program Files\SmartPlant\SP3D\CatalogData\BulkLoad\DataFiles** on this server. Edit the copy of **AllCodesList.xls** to add the new Fluid Code object to its worksheet.

Open the spreadsheet, locate the **Fluid Code** tab, and then locate the **Gas Fluid System**. Insert a row under this Fluid System, add the information that was added to the Schema files, and then add an “A” to add this value during bulkload. Save the file.

This adds the updated entries for **Corrosive** to the load file for loading.



Extending the SP3D Schema

- Add the fluid codes to the Schema files, and then add an A in the “A” column. This value tells the loader to add the value during the bulkload process. When you are finished, save the file.

	A	B	C	D	E	F	G	H	I
-	E4			CM	Make up gas	249			
-	65			GN	Natural gas	221			
-	66			GR	Hydrogen gas	251			
-	67			GOX	Oxygen gas	277			
-	59			GP	Purge gas	239			
-	60			GR	Reformed gas	233			
-	61			GS	Sulphuric acid gas	231			
-	62			GSO	Sour gas	239			
-	63			GRW	Swirl gas	242			
-	64			GRY	Waste gas	243			
-	65			GF	Water gas	239			
-	66			Solvent		30			
-	67			K	Acetone	441			
-	68			KG	Cyclohex	446			
-	69			RF	Formic al	391			
-	70			RH	Brine solvent	356			
-	71	A	Corrosive	RH	Other solvent	370			
-	72	A		RA	(KA) Ammonia, Anhydrous	19160			
-	73	A		RC	(CA) Ammonium carbide	19161			
-	74	A		KP	(OK) Process Chemical	19163			
-	75	A		KW	(OW) Ammonia, Aqueous	19164			
-	76			M	Chemical	461			
-	77			MCI	Chemical injection	467			
-	78			MAA	Anhydrous ammonia	468			
-	79			MAC	Ammonium carbide	469			
-	80			MAW	Aqueous ammonia	471			
-	81			MCS	9.5% caustic solution	417			
-	82			MCL	Chlorine	421			
-	83			MEO	Ethylen oxide	425			
-	84			MPC	Hydrogen chloride	429			
-	85			MRI	Insulation	431			
-	86			MRR	Insulator R	434			
-	87			MR_A	Lube oil additives	440			
-	88			MRB	Mineral oil	444			
-	89			MS	Oil	445			
-	90			MSL	Liquid sulfur	463			
-	91			MZ	Other chemical	469			
-	92								

© 2005, Intergraph Corp.
All Rights Reserved.

Next, create a Custom Interface XLS file to load the new custom properties into SP3D.

The file D:\SPF_Training\SP3D\EFCustomProps.xls file will contain the SystemCode and EngineeringSystem properties. Open this file to verify the properties.



Extending the SP3D Schema

- ❑ The image below illustrates an Excel file that will create the custom properties **System Code** and **EngSystem** in the SmartPlant 3D schema.
- ❑ You can review this file on your class machine here: D:\SPF_Training\SP3D\EFCustomProps.xls.
- ❑ Here, we see the names of the new properties and the name of the SP3D interface on which we want to place them (**IUPBSItem**).

A	B	C	D	E	F	G	H
1	Back to Index						
2							
3	Head	InterfaceName	CategoryName	AttributeName	AttributeUserName	Type	UnitsType
4							
5	Start						
6	IUPBSItem	Standard	SystemCode	System Code	Char	0	C
7	IUPBSItem	Standard	EngineeringSystem	Engineering System	Long	0	C
8							
9	End						
10							
11							

© 2008, Intergraph Corp.
All Rights Reserved.

Specify the code list used.



Extending the SP3D Schema

- ❑ Scrolling to the right, we see the name of the codelist that will be used by the *EngSystem* property (*EngineeringSystem*).

	G	H	I	J	K	L	M
1							
2							
3	UnitsType	PrimaryUnits	CodeList	codelisttablename	OnPropertyPage	ReadOnly	SymbolParameter
4							
5							
6	0	0			TRUE	FALSE	
7	0	0	EngineeringSystem	UDP	TRUE	FALSE	
8							
9							
10							
11							
12							

© 2008. Intergraph Corp.
All Rights Reserved.

Specify the class defs that use will use the new custom interface.



Extending the SP3D Schema

- ❑ You will also need to specify, in that file, what classes will use the interface on which you created the new customer properties. Here, the *IUPBSItem* is added to pipeline systems and piperuns.

A	B	C	D	E	F	G	H	I
1								
2	HEAD	ClassName	InterfaceName					
3								
4	Start							
5								
6	!	Example of adding interfaces to virtual classes						
7								
8								
9	!	Adding interfaces to non-virtual classes						
10	a	CPPipelineSystem	IUPBSItem					
11	a	CPMPipeRun	IUPBSItem					
12								
13								
14	End							
15								

© 2008. Intergraph Corp.
All Rights Reserved.

Also in the same directory is the file **EFCustomCodeList.xls**

Extending the SP3D Schema

- You must also create the new codelist that will be used by the EngSystem property.
- The file **EFCustomCodeList.xls** in the same directory provides and example and is illustrated below.

A	B	C	D	E
1	! Back to Index			
2				
3				
4				
5	EngineeringSystem	EngineeringSystem	Codelist	Sort
HEAD	ShortDescription	LongDescription	Number	Order
6				
START				
7	a AA	Steam Gen & Fired Htr, Water/Steam Side	10111	
8	a BA	Steam Generator and Fired Heater, Air/Gas Side	10112	
9	a CA	Ammonia Area	10113	
10	a DC	Crude Oil Production	10114	
11	a EA	Auxiliary AC Power Systems	10115	
12	END			
13				

© 2008, Intergraph Corp.
All Rights Reserved.

Once verified close and save the changes.

Next, update the SP3DToEFWClassMap.xml file. This will tell SP3D which object to pass into 3D for the other applications.

Extending the SP3D Schema

- Next, update the SP3D retrieve map file (**SP3DToEFWClassMap.xml**).
- Add the XML lines displayed here:

```

</ClassMap>
- <ClassMap>
  <ClassName>CPMPipeRun</ClassName>
  <MappedClassName>PidPipingConnector</MappedClassName>
  - <InterfaceMap>
    <InterfaceName>IUPBSItem</InterfaceName>
    - <PropertyMap>
      <PropertyName>SystemCode</PropertyName>
      <MappedInterfaceName>ICustomCode</MappedInterfaceName>
      <MappedPropertyName>SystemCode</MappedPropertyName>
      <Updatable>True</Updatable>
    </PropertyMap>
    - <PropertyMap>
      <PropertyName>EngineeringSystem</PropertyName>
      <MappedInterfaceName>ICustomCode</MappedInterfaceName>
      <MappedPropertyName>EngineeringSys</MappedPropertyName>
      <Updatable>True</Updatable>
    </PropertyMap>
  </InterfaceMap>
- <InterfaceMap>

```



© 2008, Intergraph Corp.
All Rights Reserved.

12.4 Loading objects into SP3D

In this section, we will see the steps for loading the custom modified properties into SP3D. This process includes four basic steps:

1. Bulkload the data into the Catalog Database.
2. Create the Design Basis.
3. Regenerate the views in the Model Database.
4. Regenerate the Reports Database.

Note:

- These steps have been performed for you on the Class VM. This was done because of the time necessary to perform these steps.
-



Extending the SP3D Schema

- The SP3D loading process includes 4 steps:**
 - Bulkload the data into the Catalog Database.
 - Generate the Design Basis.
 - Regenerate the views in the Model Database.
 - Regenerate the Reports Database.
- Due to time constraints and the amount of time required to complete these tasks, these steps have been performed for you on the Class VM. However, the following slides illustrate the process.**

The first step is to load the data using the Bulkload utility from SP3D. To do this select **All Programs > Intergraph SmartPlant 3D > Database Tools > Bulkload Reference Data**.

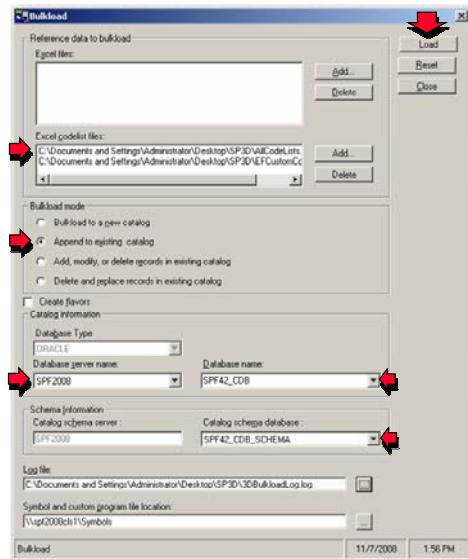


Once the dialog is opened, you will need to select the codelists you wish to input into SP3D. Click the **Add** button next to the codelist section. Browse to the directory where you codelist xls files are located. The file names are **AllCodeLists.xls** (which contains your new fluid system and fluid code values) and **EFCustomCodeLists.xls** (which contains the new Engineering System enumerated list and values). Provide the necessary connection information, and the name and path of the log file.

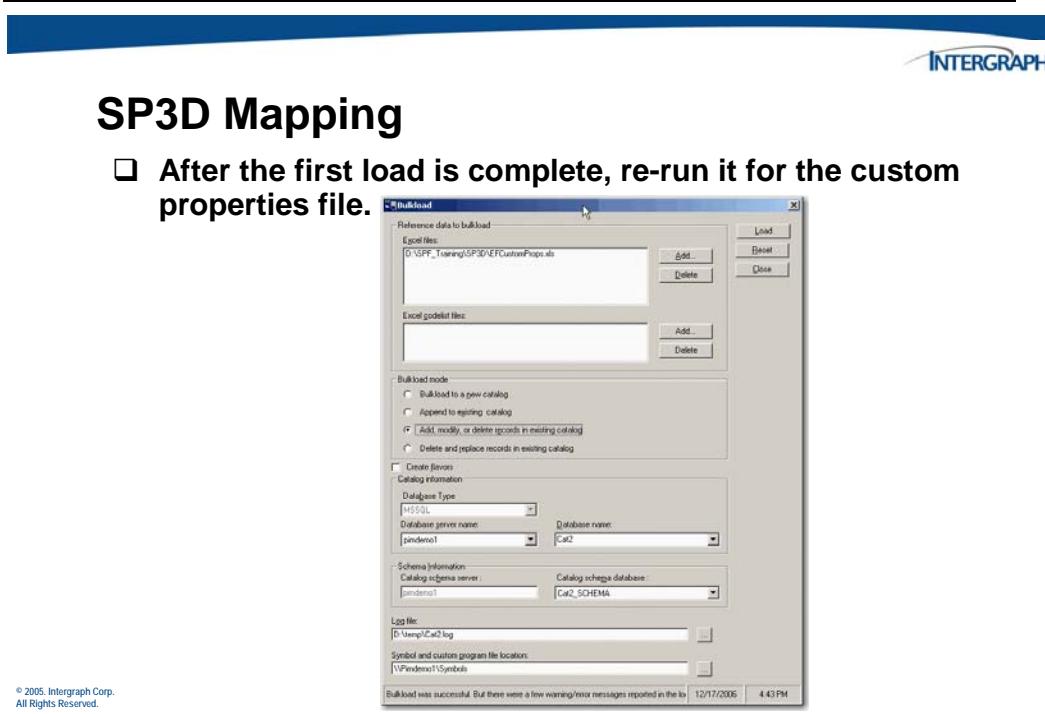
Click the **Load** button to start the bulkload process.

Extending the SP3D Schema

- Find the codelist files you want to add to SP3D, and provide connection information for the specified databases.
- Once you have provided the requested information, click **Load** to load the changes.



After the first load is complete, re-run the bulkload utility to load the custom properties file. Remove the two codelist files from the list and add the property file to the top section in the dialog. Change the **Append to catalog** to **Add, Modify, delete records in existing catalog**. The other settings will be the same as the first load.

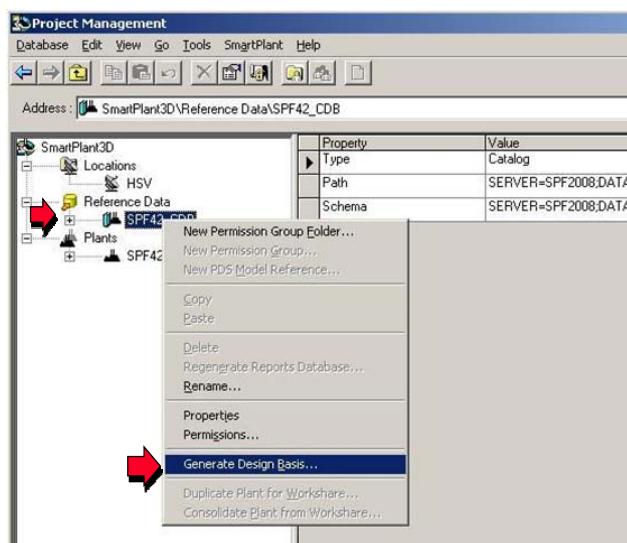


Close the Bulkload utility.

Once you have used the bulkload utility to add the new properties and lists to the SmartPlant 3D schema, recreate the **Design Basis**. To do this, open the Project Management utility for SP3D. It is located under **All Programs > Intergraph SmartPlant 3D > Project Management**. Under the 3DPlant structure, open the Reference Data section, and right-click on the node there. Click the **Generate Design Basis** command. It will start the process and display a completed dialog when completed.

Extending the SP3D Schema

- From the **Project Management** utility, use the **Generate Design Basis** command.

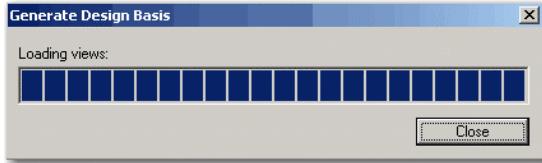
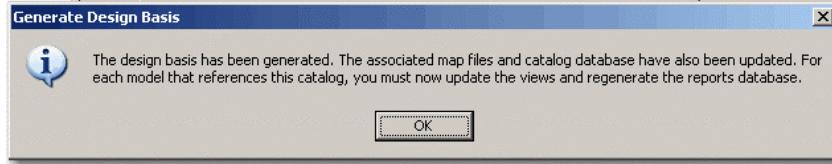


© 2008, Intergraph Corp.
All Rights Reserved.

The following progress bar will indicate the progress of the process, which can take several minutes.

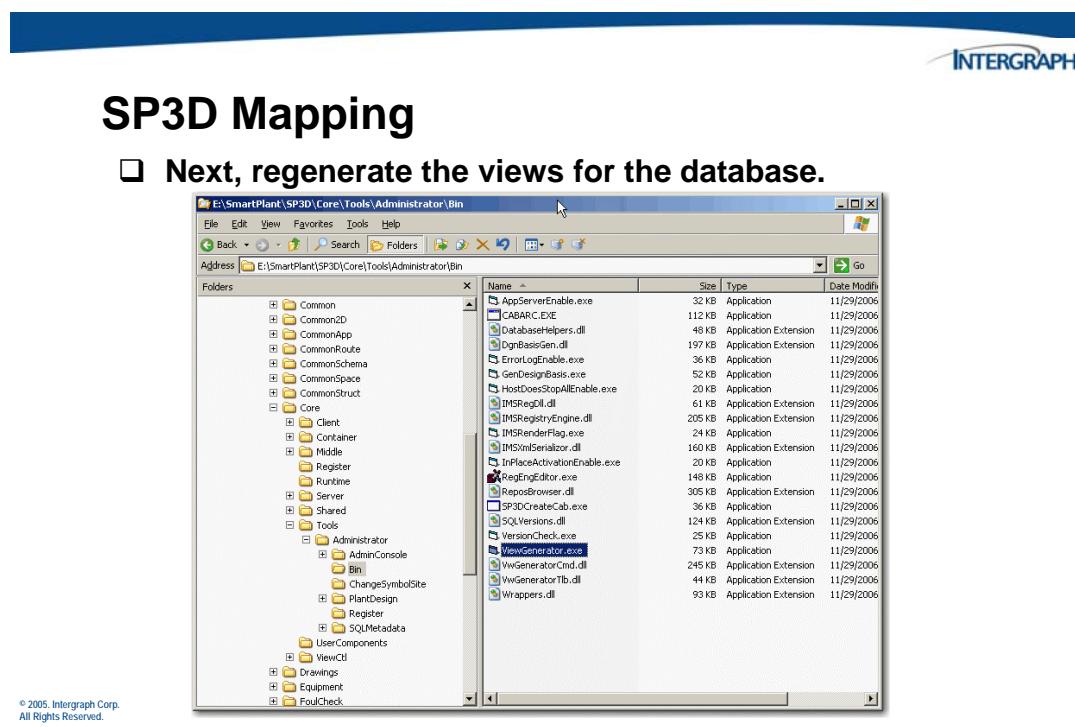
SP3D Mapping

- The following warning box and progress dialog will appear.



The next step is to regenerate the Views for the Database. To do this you will need to open Windows Explorer and go to the following directory: **E:\Program Files\SmartPlant\SP3D\Core\Tools\Administrator\Bin** (the file location will be different on your server depending on your load directory)

Run the file **ViewGenerator.exe**.

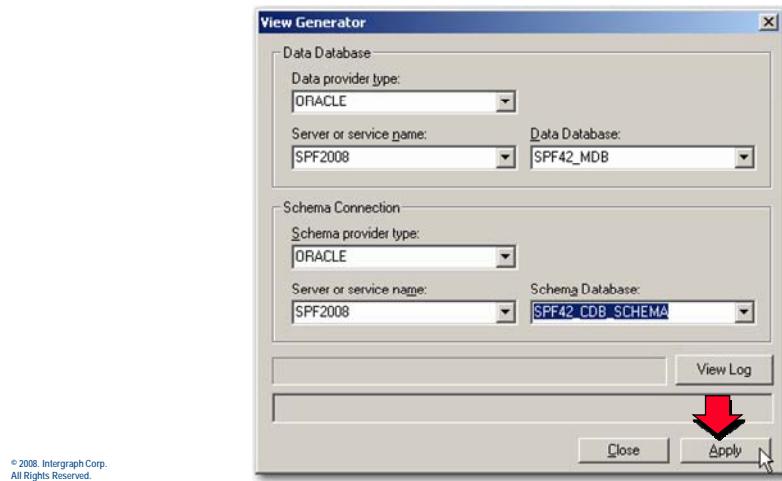


Provide connection information to the applicable SmartPlant 3D databases.



Extending the SP3D Schema

- Confirm the database information, and then click **Apply**.

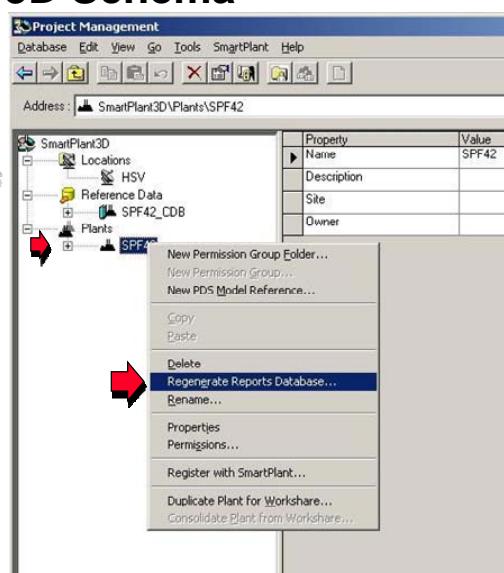


Finally, return to the Project Management utility, right-click on the plant, and click **Regenerate Reports Database**.



Extending the SP3D Schema

- Finally, reopen the Project Utility.
Right-click on the plant, and click **Regenerate Reports Database**.

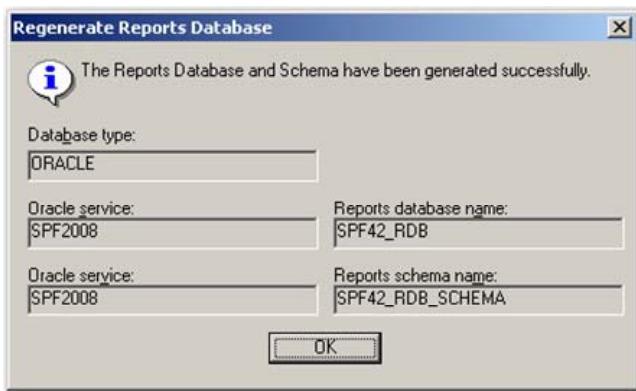


The applicable database information will appear by default in the form illustrated below. Click **OK**.

The following dialog will appear at the end of the report load.

Extending the SP3D Schema

- ❑ Confirm the database information, and then click **OK**. Once the Reports Database and Schema have been successfully generated, close the dialog box below.



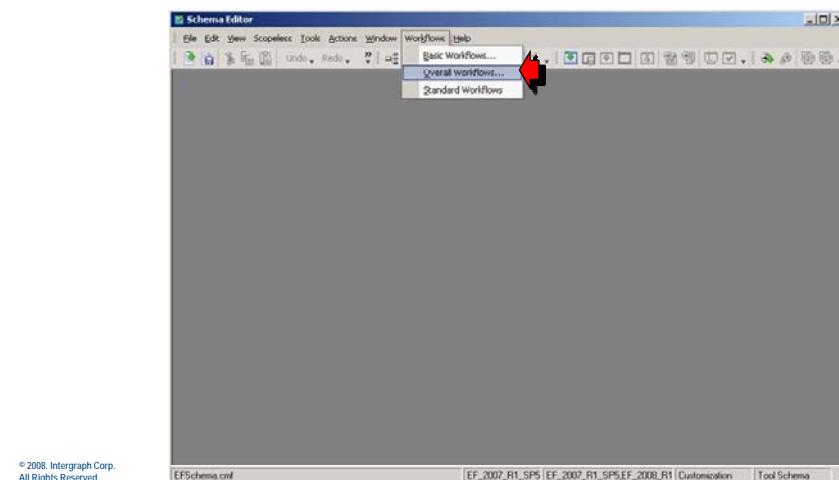
12.5 Mapping Configuration for Publish in SP3D

Launch the Schema Editor.

In our environment, the CMF file is already checked out and a session file exists with the appropriate connection information.

Synchronizing the Tool Data and Map File

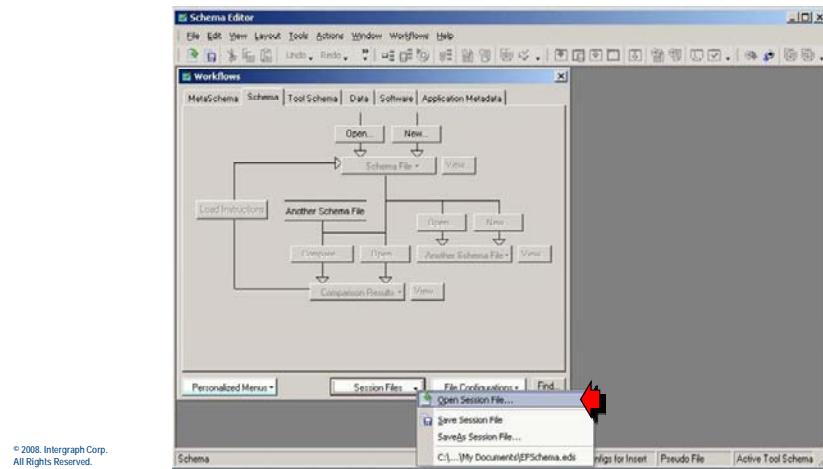
- Launch the Schema Editor and from the **Windows** menu, choose the **Overall Workflow**.



From the *Overall Workflows* window, click the *Sessions File* button.

Synchronizing the Tool Data and Map File

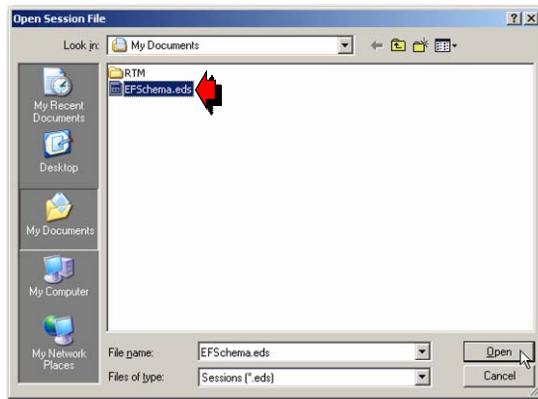
- ❑ On the Schema tab, click the *Session Files* button and choose the *Open Session* file command.



Find the existing session file.

Synchronizing the Tool Data and Map File

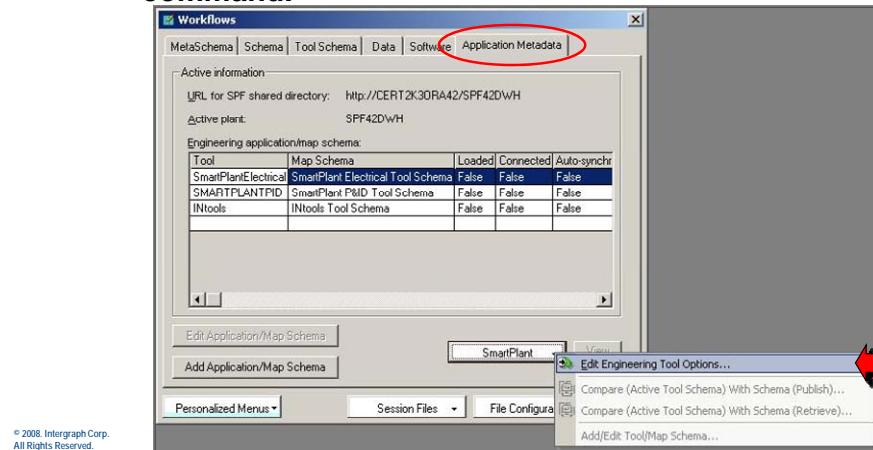
- ❑ Find the *EFSchema.eds* file in your *My Documents* folder, and open it.



From the *Application Metadata* tab, use the *SmartPlant* button to access the *Edit Engineering Tool Options*.

Synchronizing the Tool Data and Map File

- ❑ From the *Application Metadata* tab, click the *SmartPlant* button, and choose the *Edit Engineering Tool Options* command.



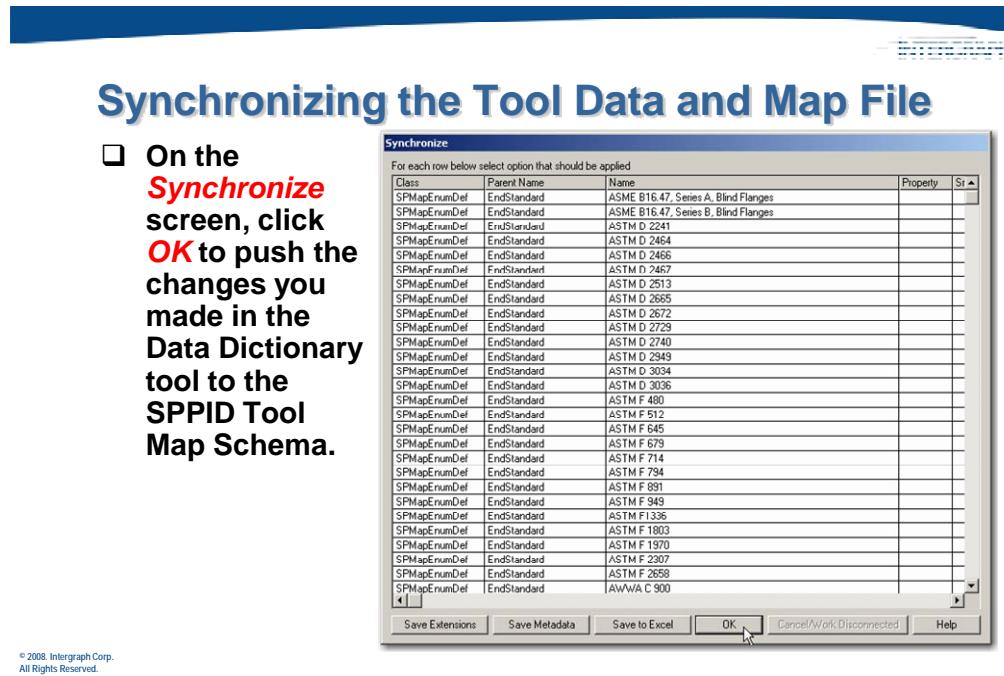
Launch the 3D metadata adapter and connect to the tool schema.

Synchronizing the Tool Data and Map File

- ❑ Choose the metadata adapter for *SmartPlant 3D*, and indicate that you want to launch the tool map schema and connect to the tool meta schema.

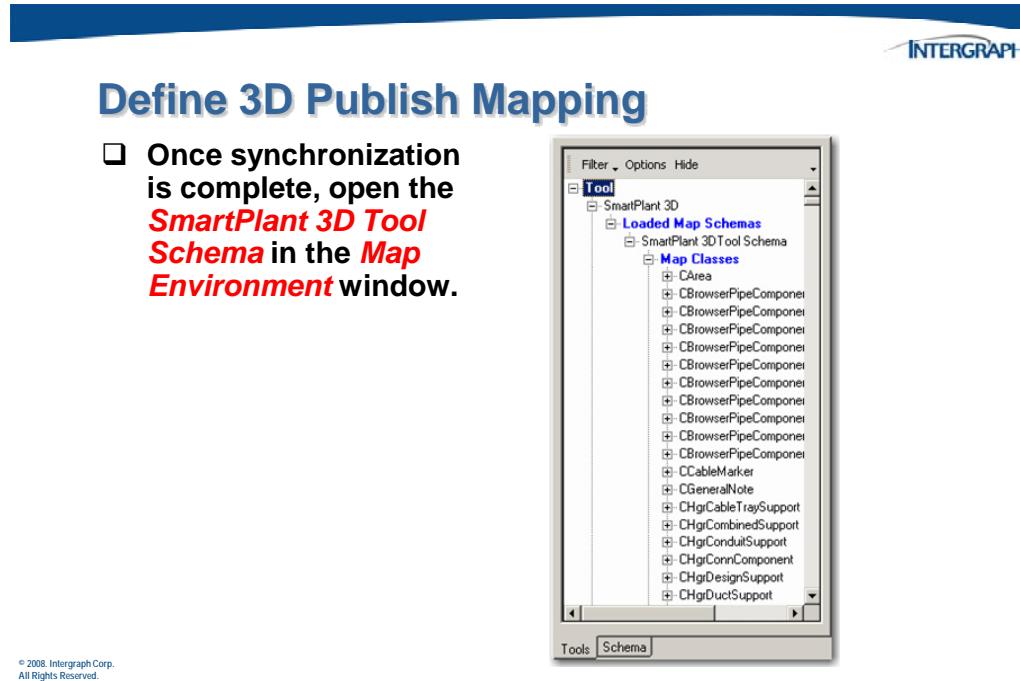


Use the *Synchronize* dialog to push the changes to the tool map file.



© 2008, Intergraph Corp.
All Rights Reserved.

Drill down under SmartPlant 3D in the Map Environment window and find the Map Classes.

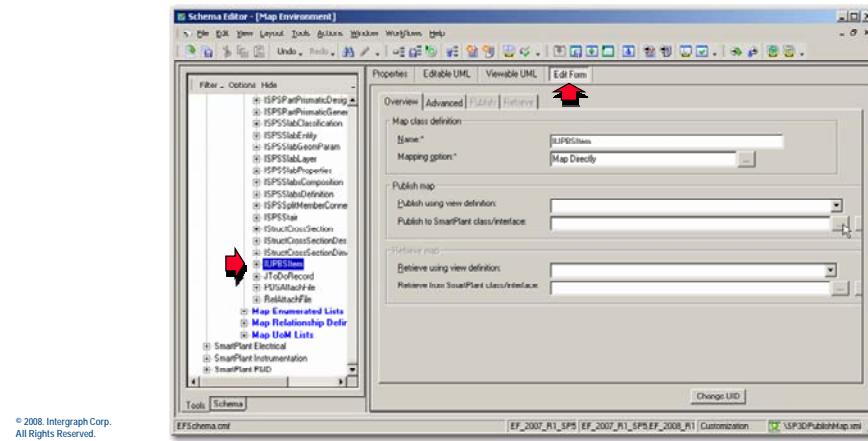


© 2008, Intergraph Corp.
All Rights Reserved.

Under **Map Classes**, find the **IUPBSItem** interface on which you placed the custom properties during the bulkload process. On the **Edit Form** window, click the **Browse** button beside the **Publish to SmartPlant class/interface** field.

Define 3D Publish Mapping

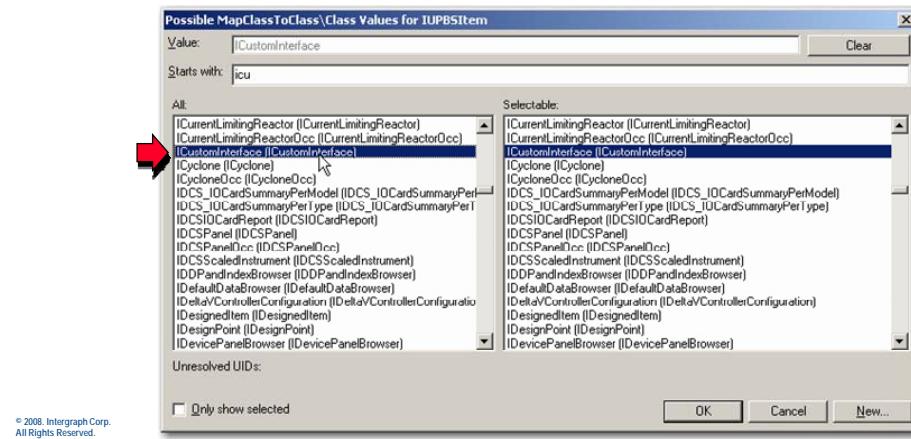
- ❑ Under **Map Classes**, find the **IUPBSItem** interface, and open the **Edit Form** view. Click the **Browse** button beside the **Publish to SmartPlant class/interface** field.



Map the IUPBSItem interface to the ICustomInterface in the SmartPlant schema.

Define 3D Publish Mapping

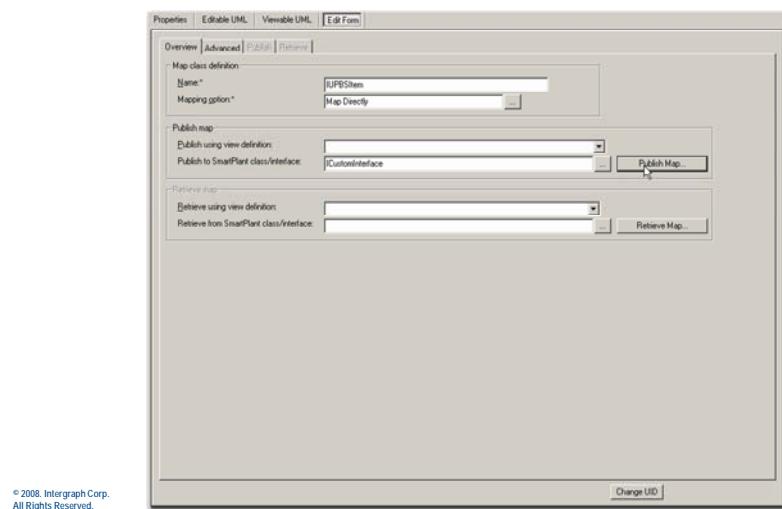
- ❑ From the list of class and interface definitions, find **ICustomInterface**, where the **SystemCode** and **EngSystem** properties were exposed.



Click the **Publish Map** button.

Define 3D Publish Mapping

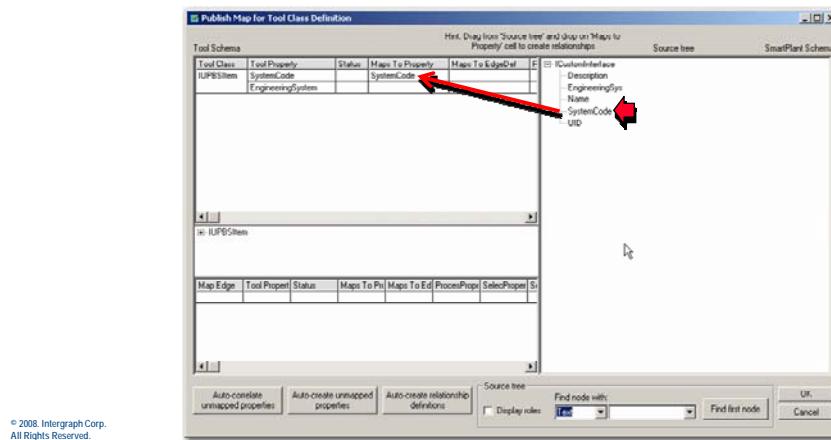
- Next, click the **Publish Map** button.



When you expand the **ICustomInterface** on the right side of the window, the new properties will appear. Drag the property from the right and drop it onto the corresponding property on the left side of the window

Define 3D Publish Mapping

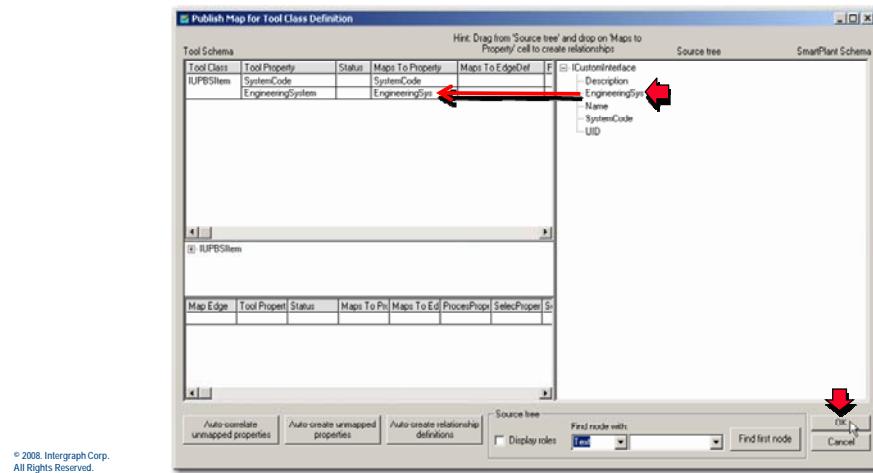
- Expand **ICustomInterface** in the window on the left, and drag and drop **SystemCode** onto tool property in the top left window.



Map both properties, and click ***OK*** when you are done.

Define 3D Publish Mapping

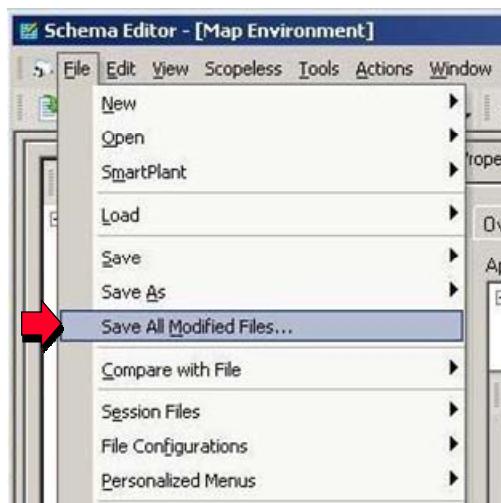
- Repeat with the ***EngineeringSys*** property, and then click ***OK***.



Save all the modified files.

Save Mapping Changes

- On the menu, click ***File > Save All Modified Files*** to save the additions to the map file.



When prompted save the changes to the SP3D tool map file.

Save Mapping Changes

- To confirm that you want to save the changes you have made to the *SP3DPublishMap.xml* file, click **Yes**.

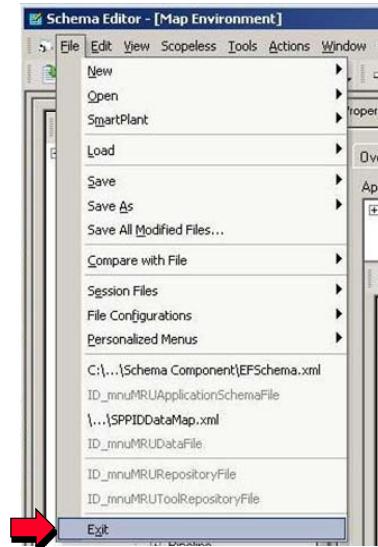


© 2008, Intergraph Corp.
All Rights Reserved.

Then exit the schema editor.

Save Mapping Changes

- Confirm that all modified files have been saved.
- Then, click **File > Exit** to close out of the schema editor.



© 2008, Intergraph Corp.
All Rights Reserved.

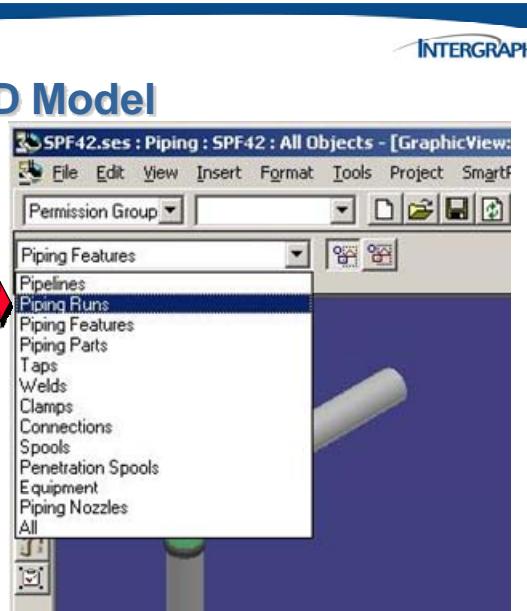
12.6 Verify the SP3D Publish Mapping

Finally, we are ready to test the mapping from SmartPlant 3D. First, launch SmartPlant 3D using the saved session file on your desktop. The name of the session file is SPF42.ses. If SmartPlant 3D does not open to the Piping task, as illustrated below, use the Tasks > Piping command to go to that portion of the tool.

From the list box near the top, choose Piping Runs as the type of selectable object.

Publishing the SP3D Model

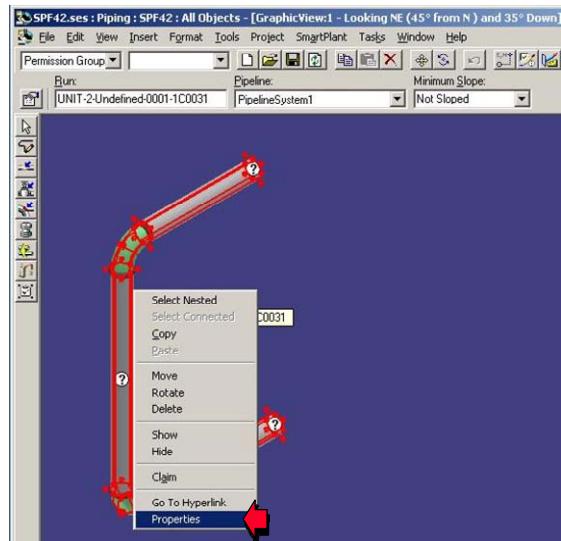
- To check that the properties were loaded into the SP3D database correctly and to publish from that tool, open SP3D by double-clicking the **SPF42.ses** file icon on your desktop.
- From the list box in the top, left corner, change the selection from Piping Features to Piping Runs.



Next, move your cursor over the pipe run in the window, and right-click on it. From the shortcut menu that appears, choose the Properties command.

Publishing the SP3D Model

- Find a pipe run in the model, right-click on it, and click **Properties** on the shortcut menu.

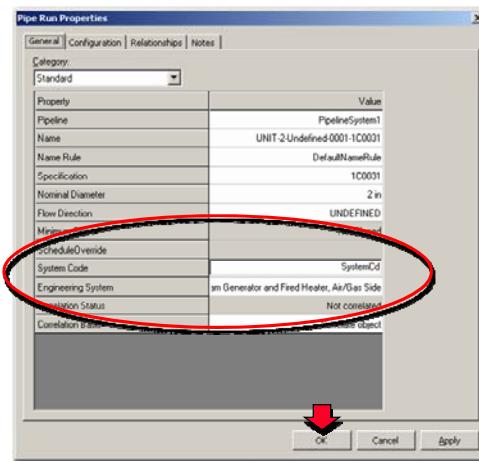


© 2008. Intergraph Corp.
All Rights Reserved.

The **System Code** and **Engineering System** fields should appear in the Standard properties. Provide values for these two new fields.

Publishing the SP3D Model

- In the **Properties** window, provide values for the two custom properties.

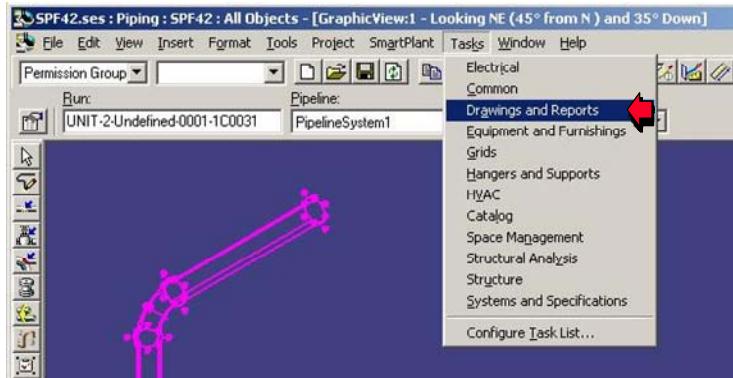


© 2008. Intergraph Corp.
All Rights Reserved.

Then use the **Tasks > Drawings and Reports** command to change to the next task.

Publishing the SP3D Model

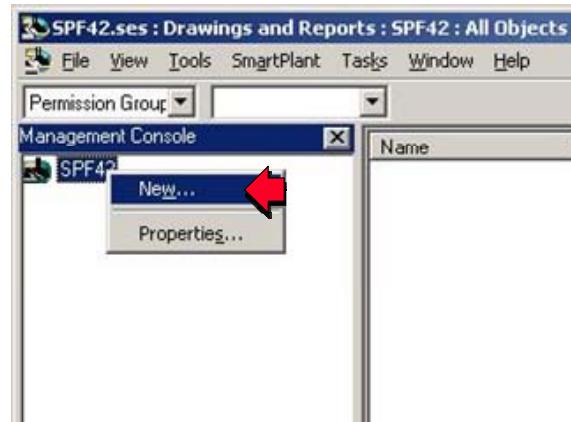
- From the menu bar, select **Tasks > Drawings and Reports**.


© 2008, Intergraph Corp.
All Rights Reserved.

In the **Management Console** frame, right-click on the plant and click **New**.

Publishing the SP3D Model

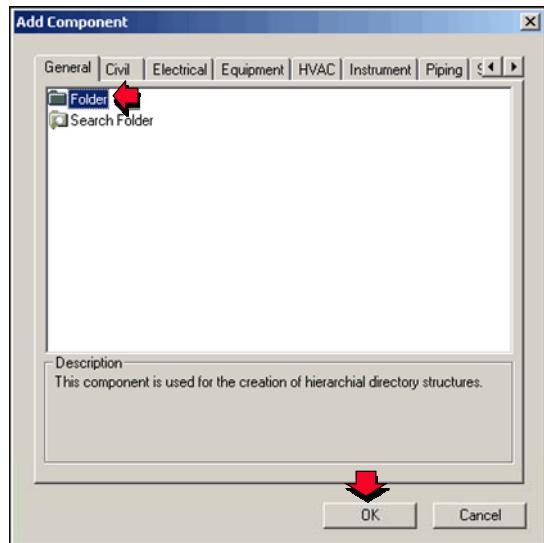
- In the **Management Console**, right-click on the plant, and click **New**. On the short-cut menu.


© 2008, Intergraph Corp.
All Rights Reserved.

From the list of new components, select **Folder** and click **OK**.

Publishing the SP3D Model

- On the **Add Component** dialog, select the **Folder** option on the **General** tab, and click **OK**.

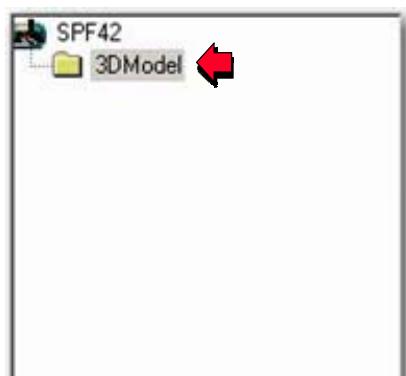


© 2008, Intergraph Corp.
All Rights Reserved.

The new folder appears under the plant. Rename it.

Publishing the SP3D Model

- Right-click on the new folder, and click **Rename** to change its name to something reflective of its purpose. In our example, we will use the name **3DModel**.

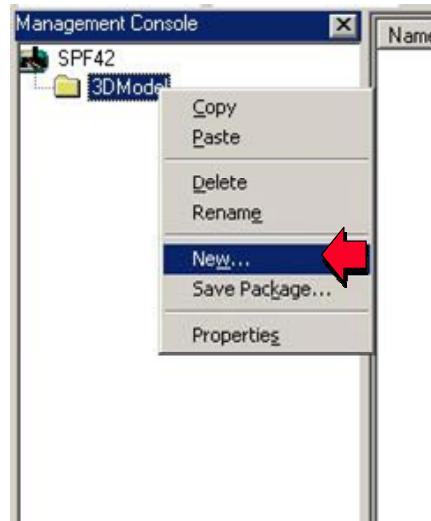


© 2008, Intergraph Corp.
All Rights Reserved.

Right-click on the new folder, and click *New* again.

Publishing the SP3D Model

- Select the **3DModel** folder, right click, and click **New** from the short-cut menu.

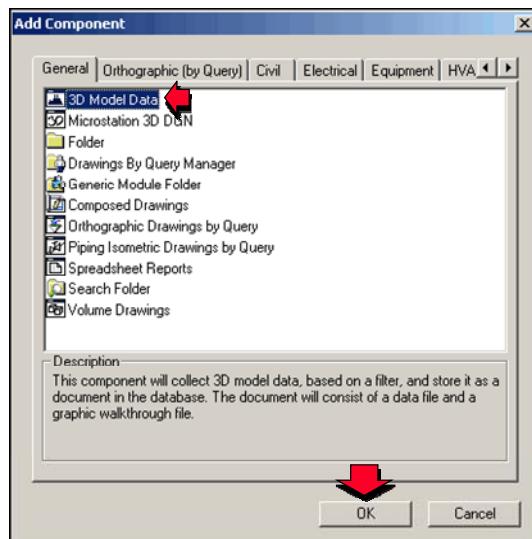


© 2008, Intergraph Corp.
All Rights Reserved.

This time, from the *New Component* form, select the **3D Model Data** option.

Publishing the SP3D Model

- On the **Add Component** dialog, highlight **3D Model Data**. This is the object type you want to publish. Click **OK**.

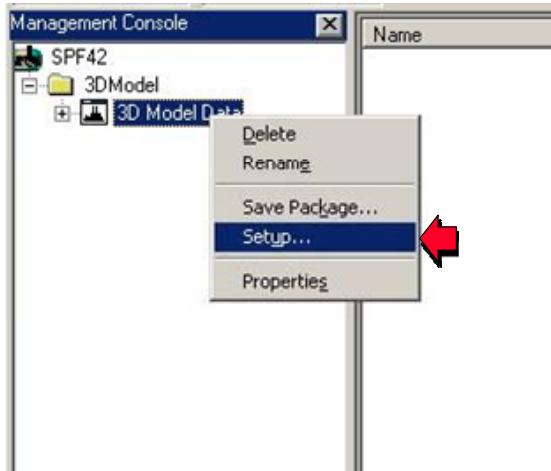


© 2008, Intergraph Corp.
All Rights Reserved.

Right-click on the new **3D Model Data** node, and click **Setup**.

Publishing the SP3D Model

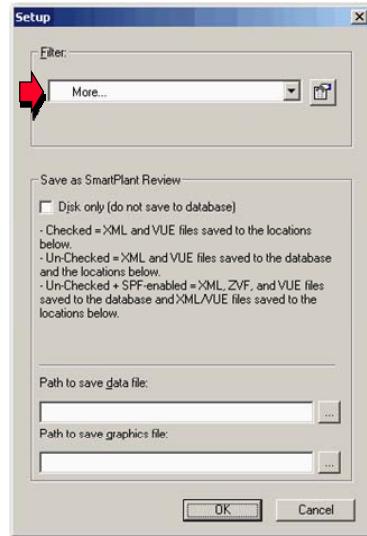
- Right-click the category **3D Model Data**, and click **Setup**.



From the **Filter** field, select and click on the **More** option to open the **Select Filter** dialog box.

Publishing the SP3D Model

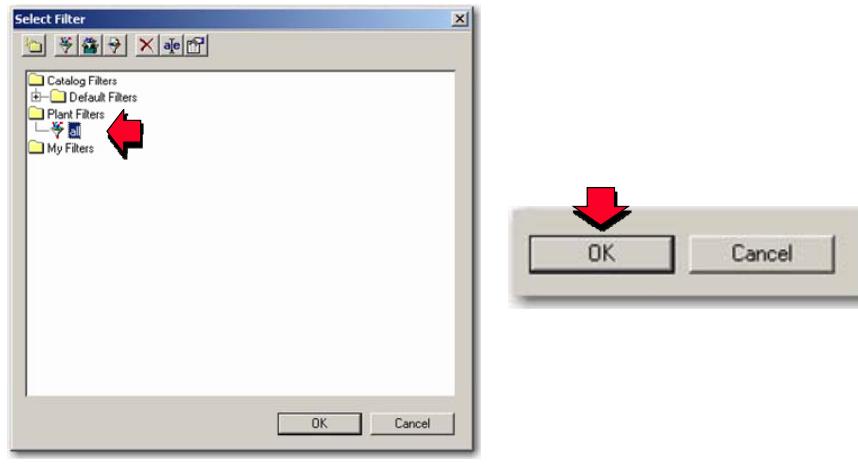
- Under **Filter**, select **More**.



Under **My Filters**, select the option labeled **MyFilter**, and then click **OK**.

Publishing the SP3D Model

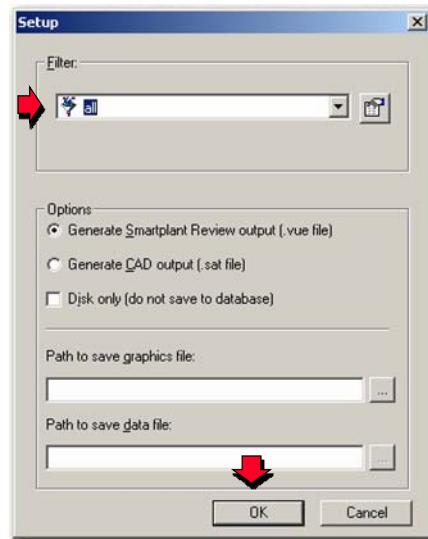
- On the **Select Filter** dialog box, select the **all** option, and then click **OK**.



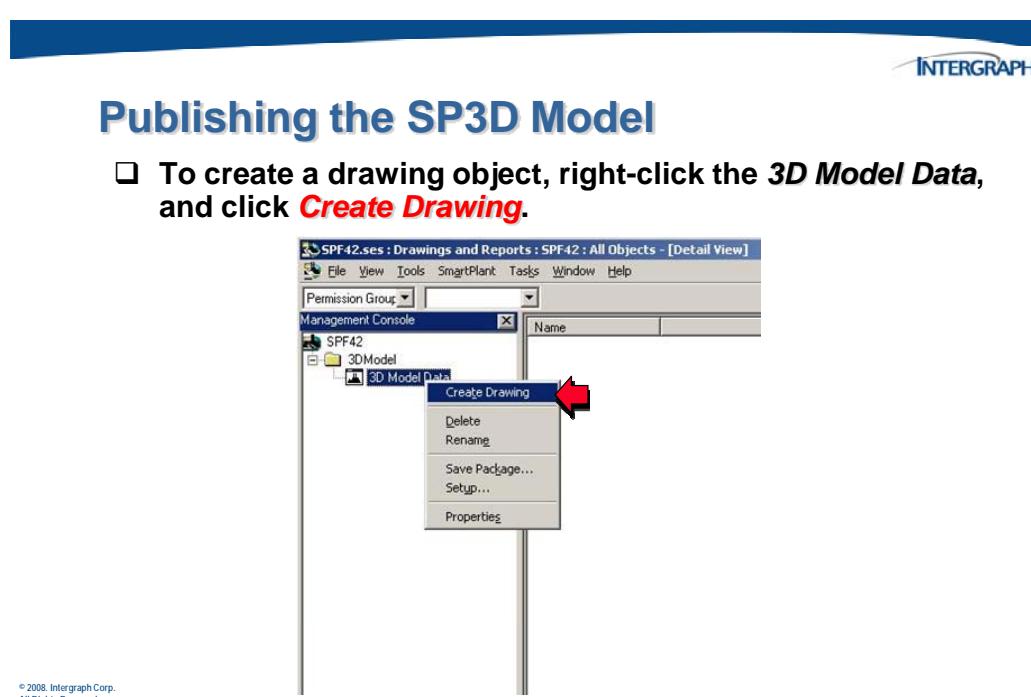
Click **OK** to dismiss the **Setup** dialog box.

Publishing the SP3D Model

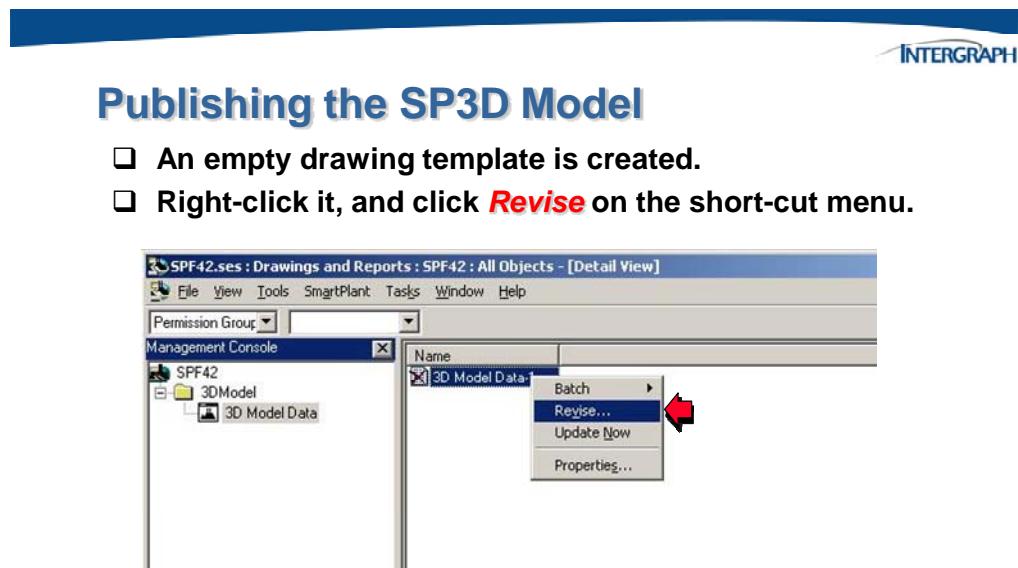
- Verify the **all** filter is selected on the **Setup** dialog box, and then click **OK** to continue.



Right-click on the **3D Model Data** node. This time, click the **Create Drawing** option.



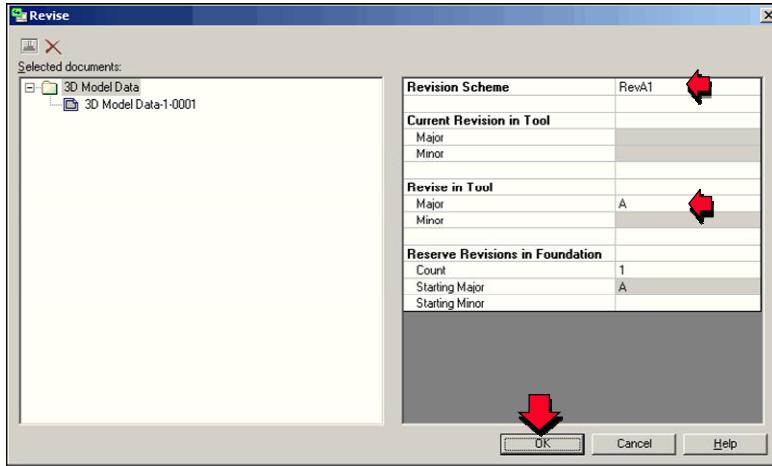
The empty drawing template will appear in the right-hand frame. Right-click on the drawing template, and click **Revise** to start the process of creating the first revision.



On the **Revise** dialog box, choose a revision scheme and a first major revision, and then click **OK**.

Publishing the SP3D Model

- Choose a **Revision Scheme** to use. Then select the **Major Revision** for this drawing, and click **OK**.



© 2008, Intergraph Corp.
All Rights Reserved.

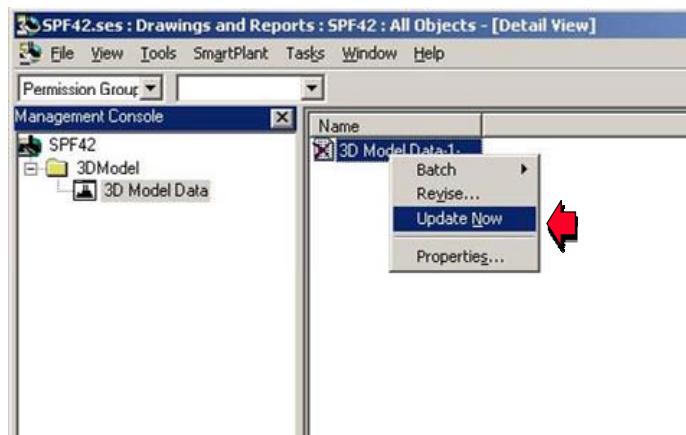
When the revision process is complete, right-click again on the drawing template. Click the **Update Now** command to update the information in the drawing to be published.

Note:

- This process may take a few minutes, and the software may not provide a wait cursor to indicate that the update is in process. Please be patient. When the update is complete, the red x in the drawing icon will be replaced with a green check mark.

Publishing the SP3D Model

- Right-click the drawing object, and click **Update Now**.

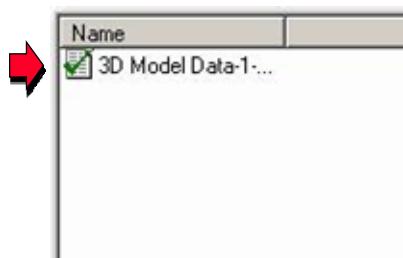


The green check indicates that the update was completed successfully.



Publishing the SP3D Model

- ❑ When the update is complete, a green check will appear on the icon for the drawing.



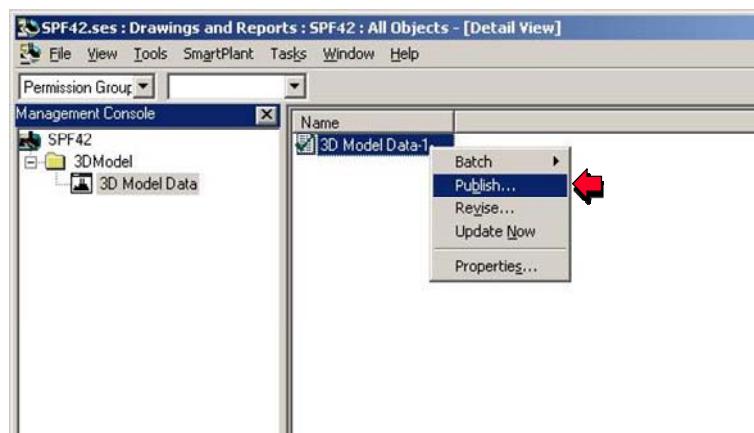
© 2008, Intergraph Corp.
All Rights Reserved.

Finally, right-click on the drawing template one last time, and click the ***Publish*** command.



Publishing the SP3D Model

- ❑ Right-click on the drawing, and click ***Publish***.

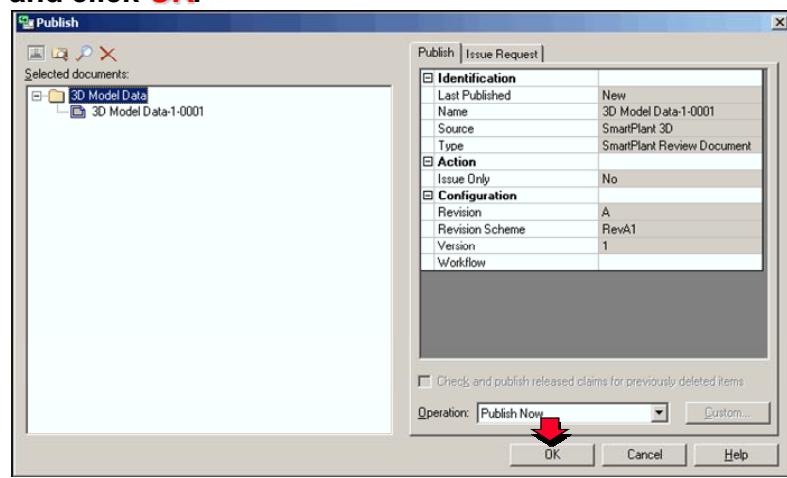


© 2008, Intergraph Corp.
All Rights Reserved.

Click **OK** on the **Publish** dialog box.

Publishing the SP3D Model

- The *SmartPlant Publish* dialog appears. Verify the data, and click **OK**.



© 2008, Intergraph Corp.
All Rights Reserved.

The following progress dialog will appear.

Publishing the SP3D Model

- A process dialog will show you the progress of the publish process, and when the process is complete, the following success dialog box will appear.



© 2008, Intergraph Corp.
All Rights Reserved.

To verify the new properties published properly, find the published document in the Desktop Client, and save the attached FileToolData xml file out of the SPF vault with the **Save Target As** command. Open that file with the Notepad or an Internet Explorer window, and find the new properties and their values in the file.

12.7 Activity – Mapping SP3D Properties for Publish

Complete the Chapter 12 – Activity 1 in the SmartPlant Foundation 2008 (4.2) Modeling and Mapping Activity workbook.

DSPF1-TP-100008A

SmartPlant Modeling and Mapping

Course Guide