

PowerDesigner to Troux Export Guide

Revision date: October 27, 2010

Summary

This document provides information about configuring and using the PowerDesigner TUX export extension files to enable the export of Sybase PowerDesigner models to Troux Semantics.

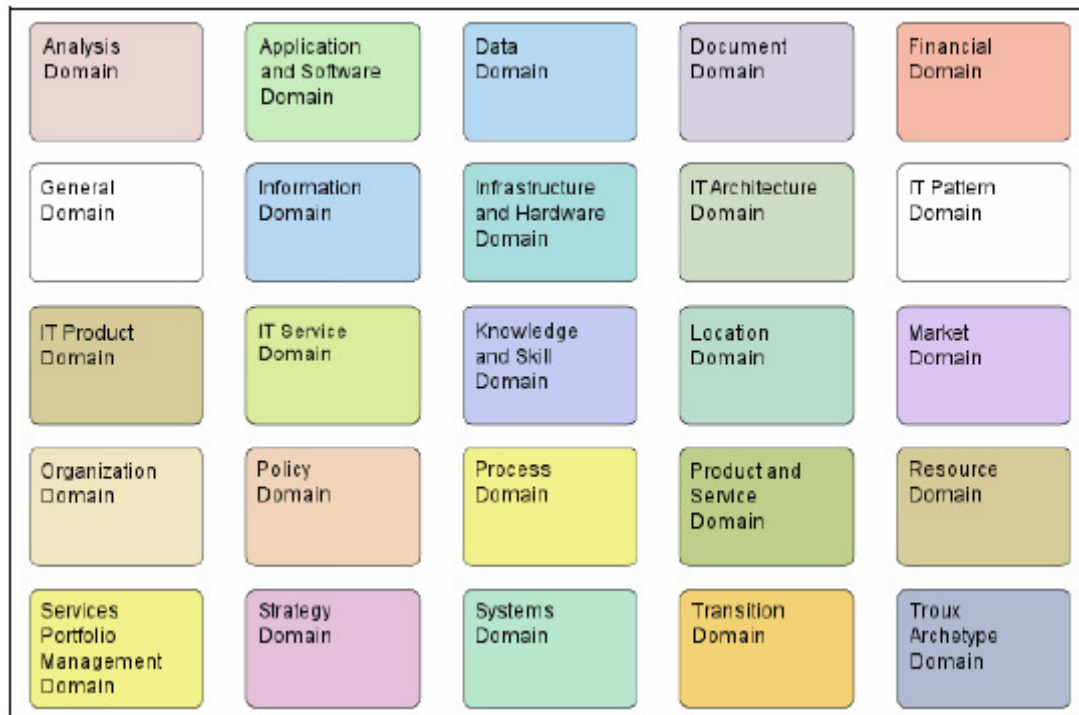
Table of Contents

Summary.....	1
Table of Contents.....	1
Introducing the PowerDesigner TUX Export.....	2
Configuring the PowerDesigner TUX Export.....	4
Adding a Metaclass to Export to the Extension File	5
Specifying a Mapping from a PowerDesigner Object to a Troux Component.....	5
Specifying a Mapping from a PowerDesigner Link Object to a Troux Relationship	5
Specifying Mappings from PowerDesigner Attributes to Troux Properties	7
Exporting PowerDesigner Objects to Troux.....	9
Attaching the Export Extension to a PowerDesigner Model	9
Launching the Export.....	10
Installing the PowerDesigner TUX Upload Configuration.....	10
Importing the PowerDesigner Data into Troux Metaverse	11
Understanding the Troux Export Engine	12
Generation of PowerDesigner Objects to Troux Components	13
Generation of PowerDesigner Link Objects to Troux Relationships.....	14
Generation of PowerDesigner Attributes to Troux Properties.....	15
Basics of the PowerDesigner GTL Language	16

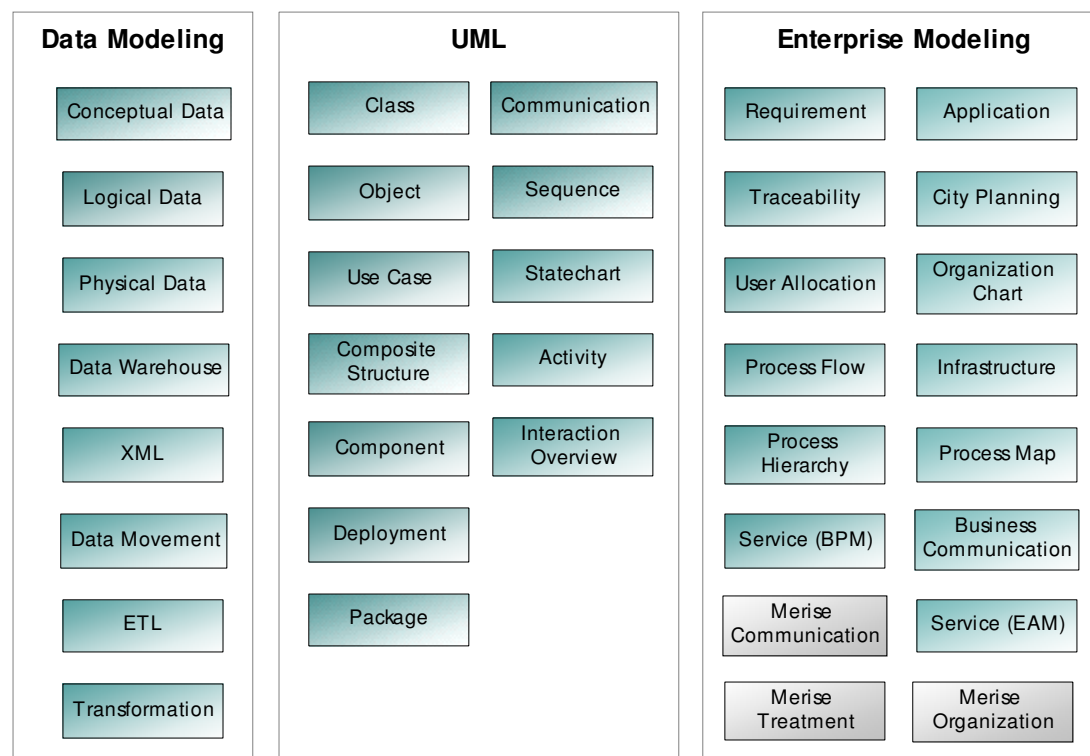
Introducing the PowerDesigner TUX Export

PowerDesigner provides a method to export metadata from PowerDesigner models to a file based on the Trous Upload XML format (TUX), which you can import using the Trous Collection feature.

Trous Semantics divides its metamodel into 25 domains:



The PowerDesigner metamodel can be divided into the following diagrams:



The following Troux domains can be mapped to their corresponding PowerDesigner concepts:

- Application and Software ⇔ EAM Application
- Data ⇔ Physical Data Model
- Document ⇔ EAM Data & Document
- General Domain ⇔ Free Model?
- Information domain ⇔ Conceptual Data Model
- Infrastructure and Hardware ⇔ EAM Infrastructure
- IT Service ⇔ EAM Service, BPM Service
- Location Domain ⇔ EAM Infrastructure & Site
- Organization Domain ⇔ EAM Organization chart
- Process Domain ⇔ BPM Process Flow, UML Activity
- Systems Domains ⇔ EAM Application & System

Other concepts could also be represented in PowerDesigner by using metamodel extensions and extending and/or customizing the TUX extension file.

We provide one Troux Upload extension file (XEM) for each type of PowerDesigner model, which must be configured with your preferred mappings (see Configuring the PowerDesigner TUX Export) before they can be used (see Exporting PowerDesigner Objects to Troux):

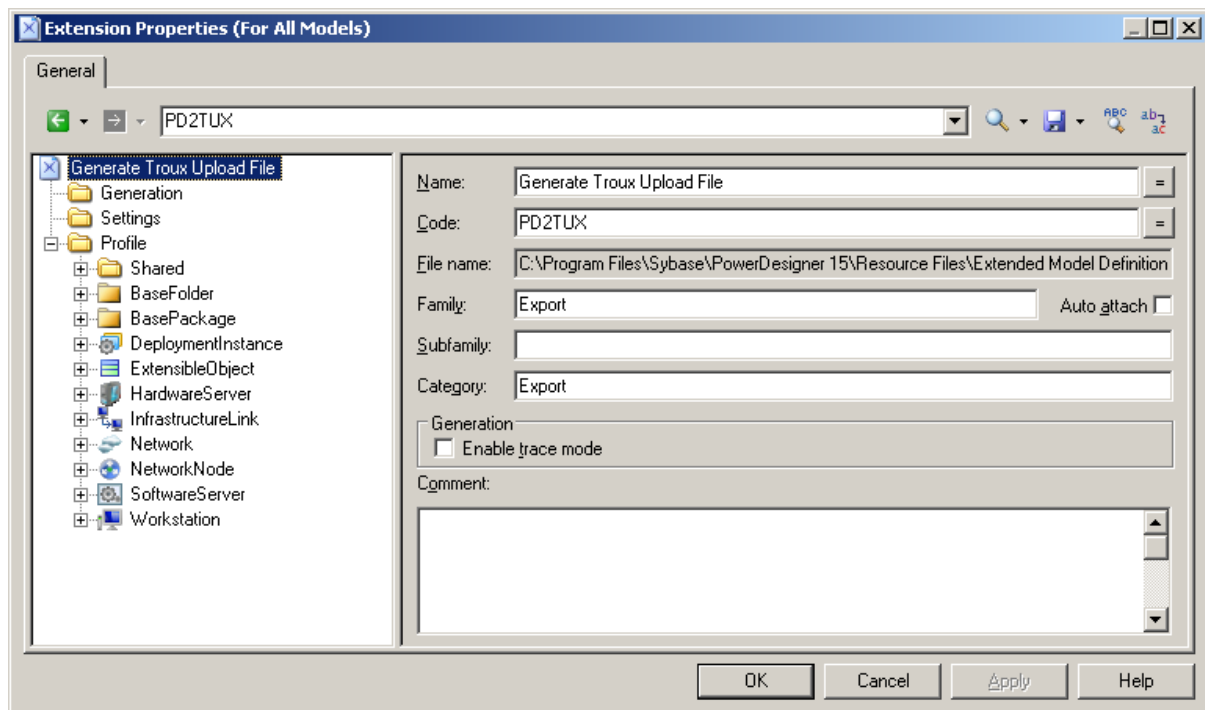
- TrouxUpload.EAM.xem ⇔ PowerDesigner Enterprise Architecture Model - example mappings for the Application and Software domains to be used as a guide for developing mappings for other domains.
- TrouxUpload.RQM.xem ⇔ PowerDesigner Requirements Model – framework only
- TrouxUpload.BPM.xem ⇔ PowerDesigner Business Process Model – framework only
- TrouxUpload.CDM.xem ⇔ PowerDesigner Conceptual Data Model – framework only
- TrouxUpload.PDM.xem ⇔ PowerDesigner Logical Data Model – framework only
- TrouxUpload.OOM.xem ⇔ PowerDesigner Object Oriented Model – framework only

Configuring the PowerDesigner TUX Export

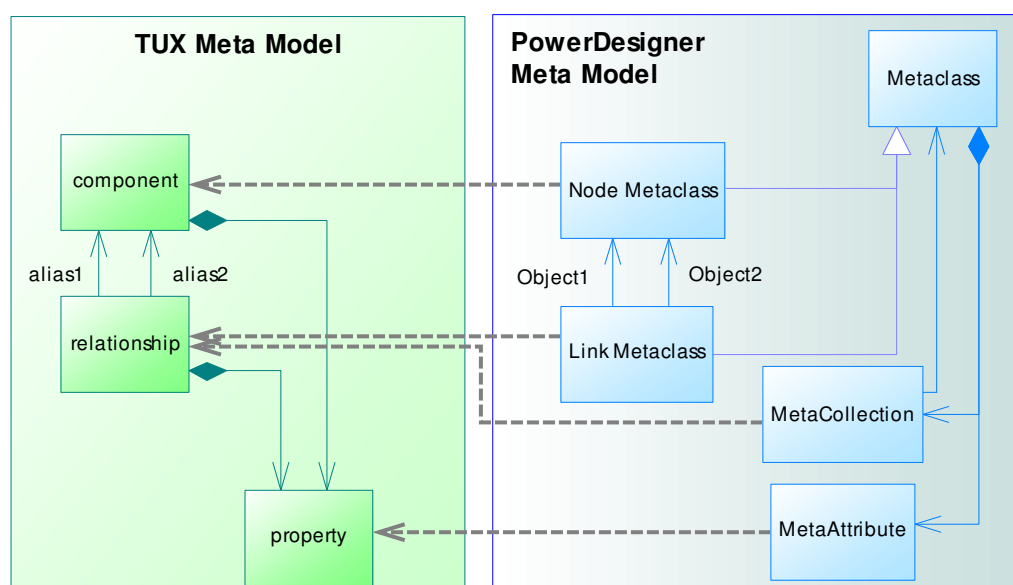
To prepare to export objects from a model, you must copy the appropriate extension file for editing from: \\<PowerDesigner home>\ Resource Files\Extended Model Definitions\Troux

to: \\<PowerDesigner home>\ Resource Files\Extended Model Definitions

To open the extension file for editing, select **Tools > Resources > Extended Model Definitions > Model Type** to open the list of extensions, select the Generate Troux Upload File in the list, and then click the Properties tool. The extension file will open in the PowerDesigner Resource Editor:



To enable the export, you must specify mappings from PowerDesigner object metaclasses to TUX components, and from link metaclasses and collections to TUX relationships. PowerDesigner metaclass attributes are mapped to Troux properties.

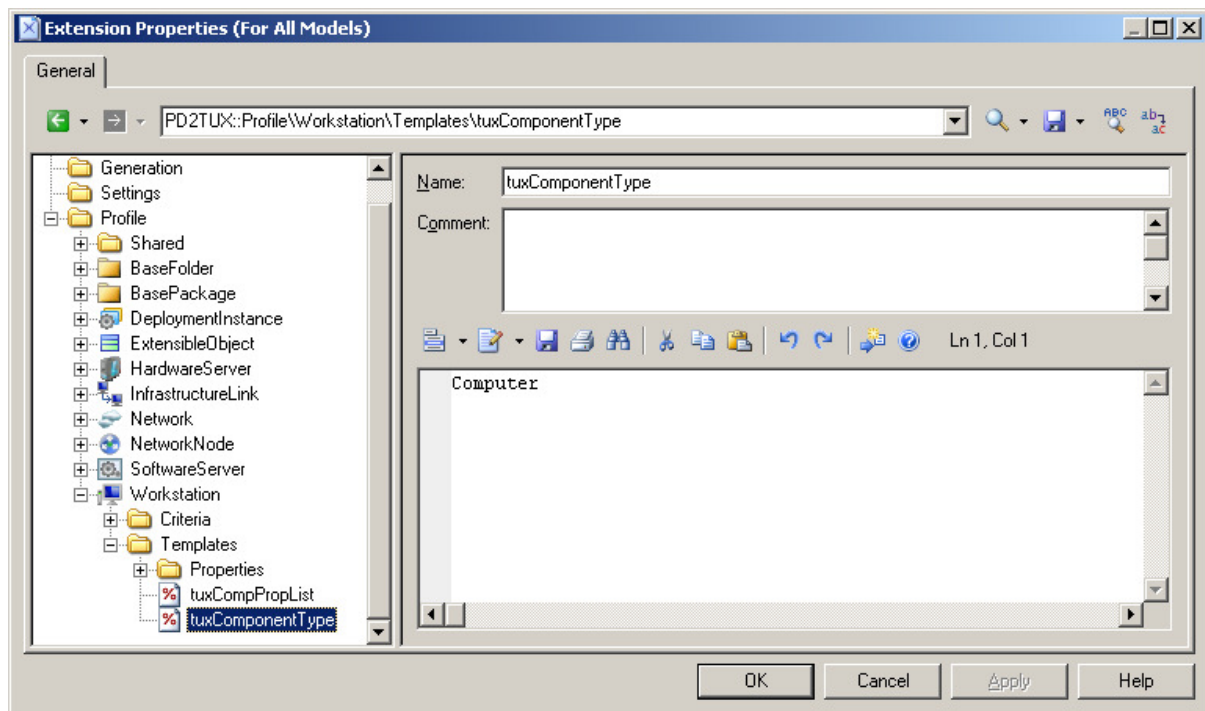


Adding a Metaclass to Export to the Extension File

To enable the export of a PowerDesigner metaclass to a Troux component type, right-click the Profile category and select Add Metaclasses. Select a metaclass from the list and click OK.

Specifying a Mapping from a PowerDesigner Object to a Troux Component

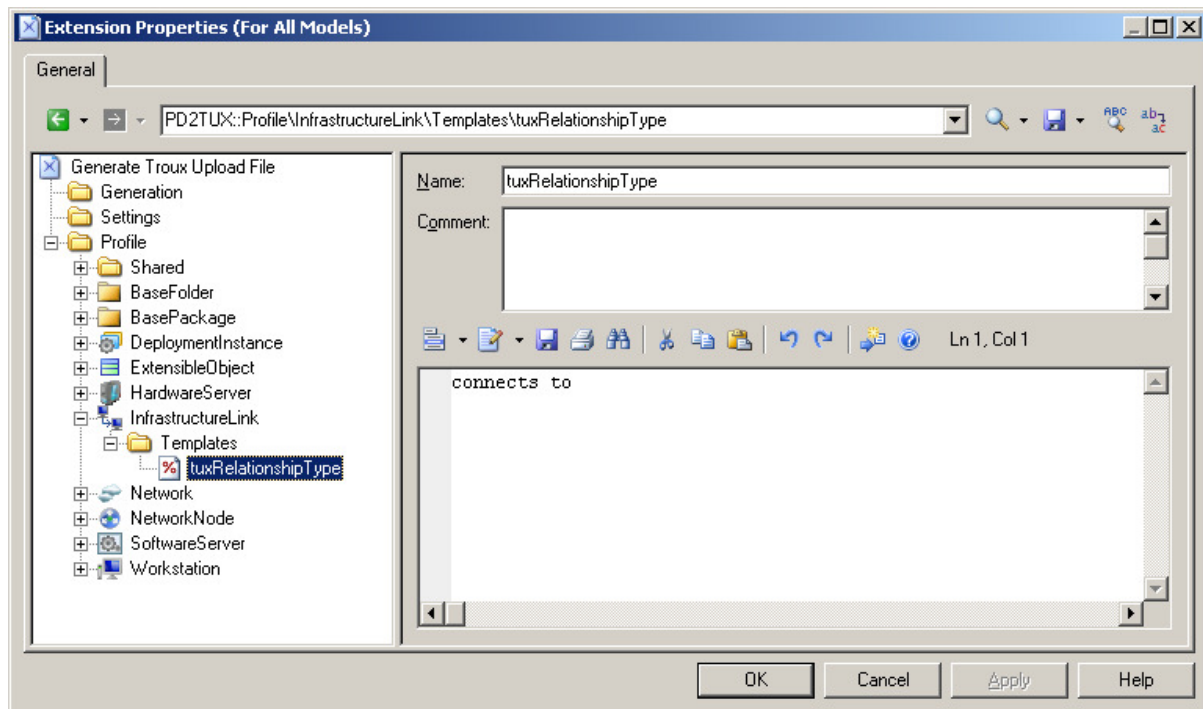
To define a mapping between a PowerDesigner metaclass and a Troux component type, right-click the metaclass and select New > Template. Set the name of the template to 'tuxComponentType' and enter the name of the component type as the template value:



This template must return the name of a valid Troux object type. Providing an empty value for this template skips the export of corresponding PowerDesigner objects.

Specifying a Mapping from a PowerDesigner Link Object to a Troux Relationship

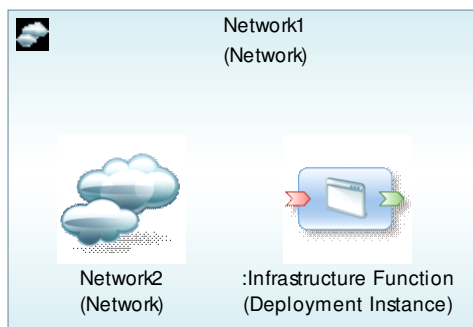
To define a mapping between a PowerDesigner link metaclass and a Troux relationship type, right-click the link metaclass and select New > Template. Set the name of the template to 'tuxRelationshipType' and enter the name of the relationship type as the template value:



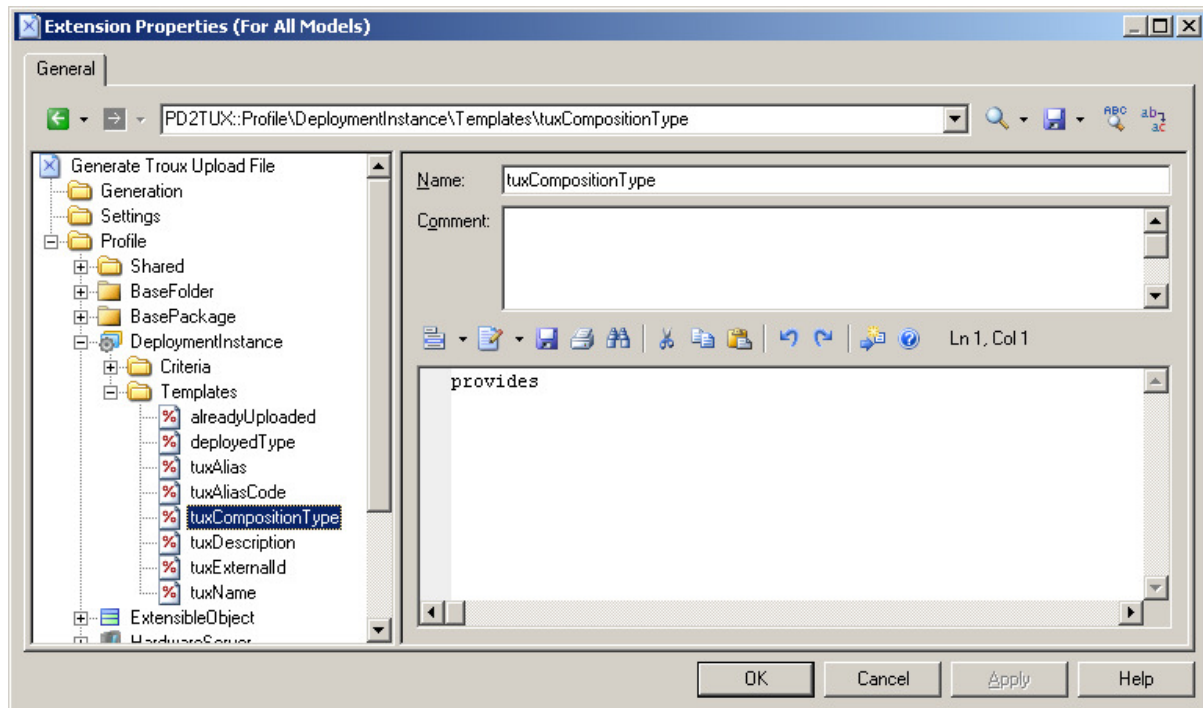
This template must return the name of a valid Troux relationship type. Providing an empty value for this template skips the export of corresponding PowerDesigner link objects.

PowerDesigner Composition Link

Some Troux relationships are represented in PowerDesigner not by link objects, but by a box- in- box (or composition) relationship. For instance in the following Infrastructure Diagram, “Network1 consists of Network2” and “Network1 provides Infrastructure Function”:



To export such a relationship, right-click the child metaclass and select New > Template. Set the name of the template to 'tuxCompositionType' and enter the name of the relationship type as the template value:



This template must return the name of a valid Troux relationship type. In the scope of this template, an additional %_parent% variable is available that contains the PowerDesigner object owning the current object. You can use this variable to generate distinct relationship types depending on the parent object type.

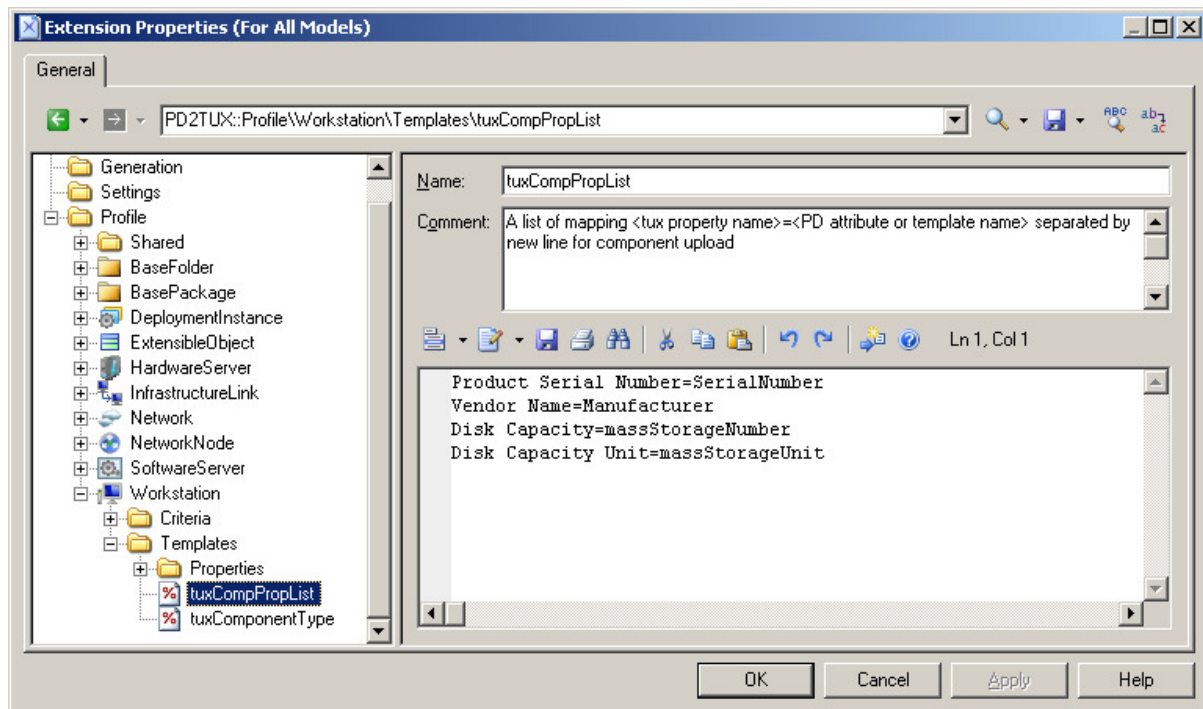
For instance, A PowerDesigner HardwareServer can contain other hardware servers, and this composition corresponds to the "stores" or "consists of" relationship depending on the parent component type:

```
.if %_parent.tuxComponentType% == "Cabinet"
    stores
.else
    consists of
.endif
```

Specifying Mappings from PowerDesigner Attributes to Troux Properties

To define mappings between PowerDesigner metaclass attributes and Troux object properties, right-click the metaclass and select New > Template. Set the name of the template to 'tuxCompPropList' (or 'tuxRlshPropList' for relationships) and enter a list of value pairs separated by new lines:

```
<Troux property name>=<PowerDesigner attribute public name>
```



The PowerDesigner attribute can be an extended attribute or even the name of a GTL template.

In case of extended attributes that are defined in another extension than the current one, prefix the attribute name by the code of the extension.

```
Current Business Value=AppExtension.CurrentBusinessValue
```



Exporting PowerDesigner Objects to Troux

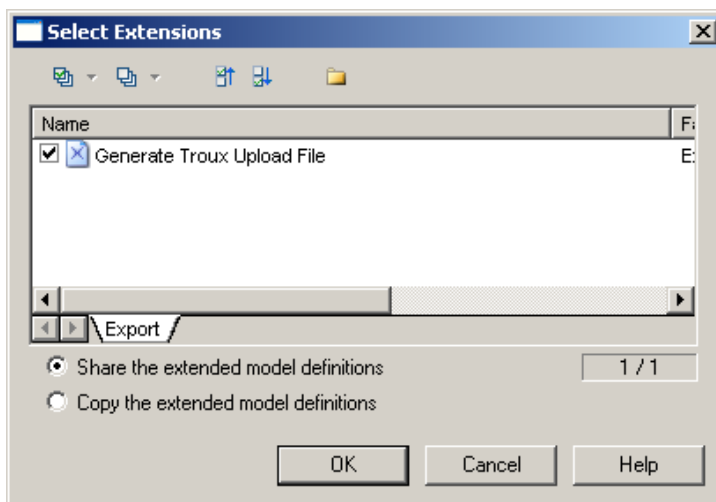
Before you can export objects to Troux, you must define mappings between your PowerDesigner model objects and Troux components in a Troux Export extension file (see Configuring the PowerDesigner TUX Export).


Once the extension file is ready to use, copy it to your Extended Model Definition directory, which by default is located at: \\<PowerDesigner home>\Resource Files\Extended Model Definitions

Attaching the Export Extension to a PowerDesigner Model

To invoke the export function, you must first attach the extension file to your model.

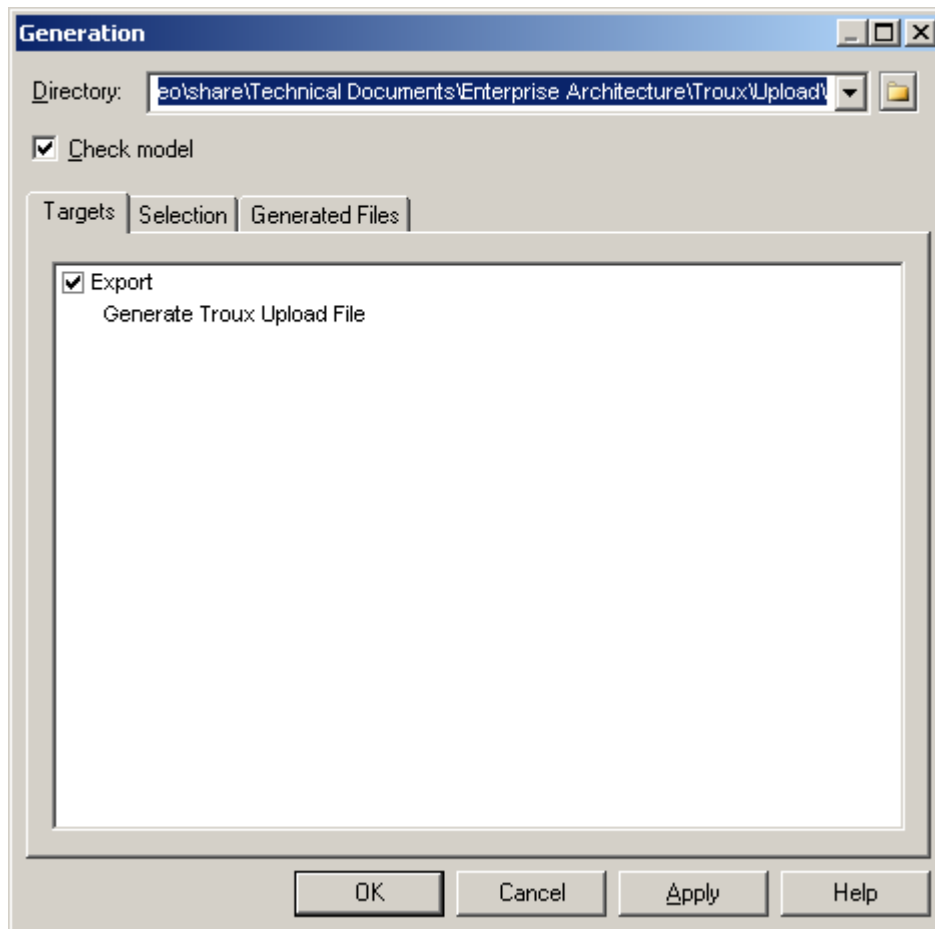
- 1) From your model, select **Model > Extensions ...** to open the list of Extensions.
- 2) Click on the Import an Extension tool () to open the Select Extensions dialog, click the Export tab, select the Generate Troux Upload File extension and click OK to attach it to your model:



If the dialog does not contain the Generate Troux Upload extension file as an XEM to select, use the browse ( button) to find the folder containing a file named TrouxUpload.xxx.xem, where xxx corresponds to the your model acronym (OOM, EAM...).

Launching the Export

Once the extension is attached to your model, select **Tools > Extended Generation...** to open the Generation dialog:

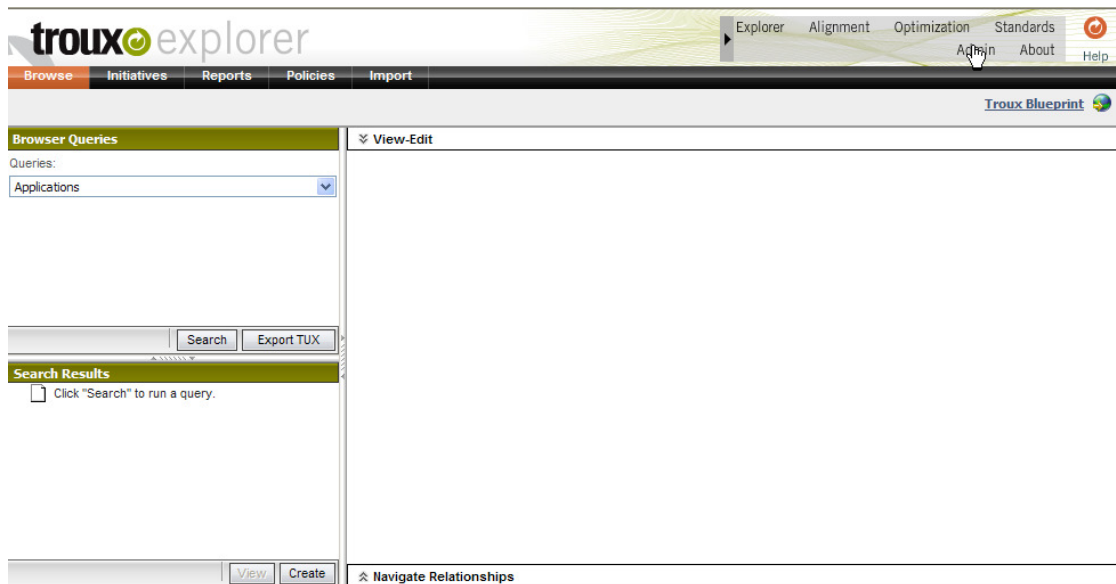


Enter the directory where you want to generate the TUX files click OK. A TUX file will be created for each model/package containing metadata to export.

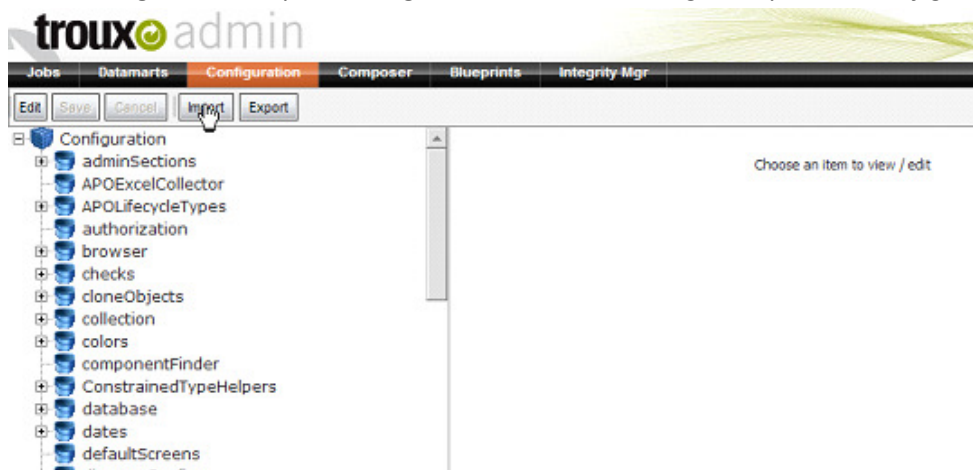
Installing the PowerDesigner TUX Upload Configuration

If you have not already done so, you must install the Trouw Metaverse PowerDesigner TUX Upload Configuration on your Trouw Metaverse Server. This configuration only needs to be installed once.

Login to the Trouw Metaverse as an Administrator and navigate to the Admin portal.



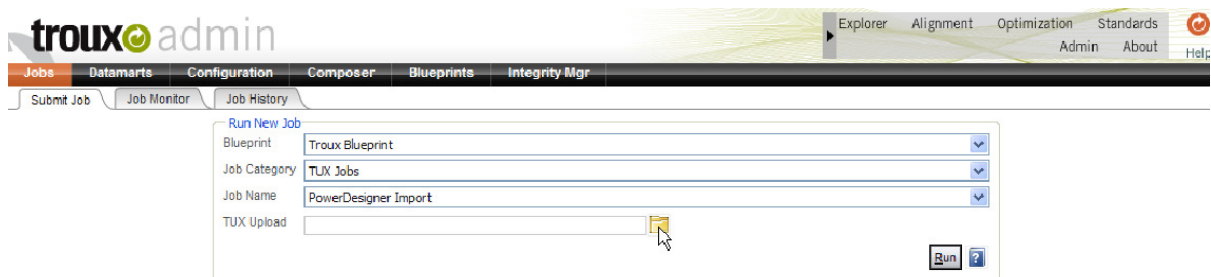
Once in the Admin Portal, select the Configuration tab, click the **Import** Button and upload the PowerDesigner TUX Import Configuration file, *PowerDesignerImportTUXConfiguration.xml*.



Once the Import is completed, you will now find a special import job for PowerDesigner listed in the TUX Jobs category of the Job Manager.

Importing the PowerDesigner Data into Troux Metaverse

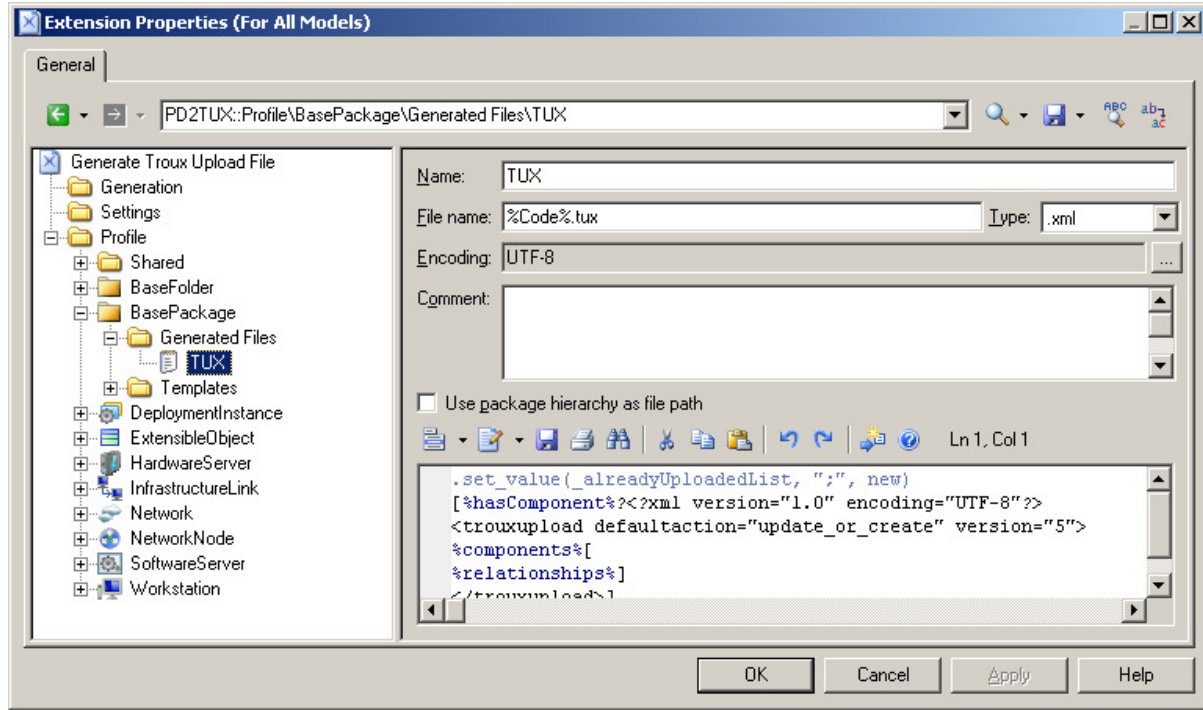
Log in to the Troux Metaverse Server and navigate to the Admin Portal. Select the Jobs tab. On the Jobs tab, select the Job Category, **TUX Jobs**, then select the Job Name, **PowerDesigner Import**. Click the folder button to browser for the tux file you exported from PowerDesigner and then click the **Run** button to import your data.




Understanding the PowerDesigner Troux Export Engine

The Troux export engine is implemented using generic GTL templates that you should not change.

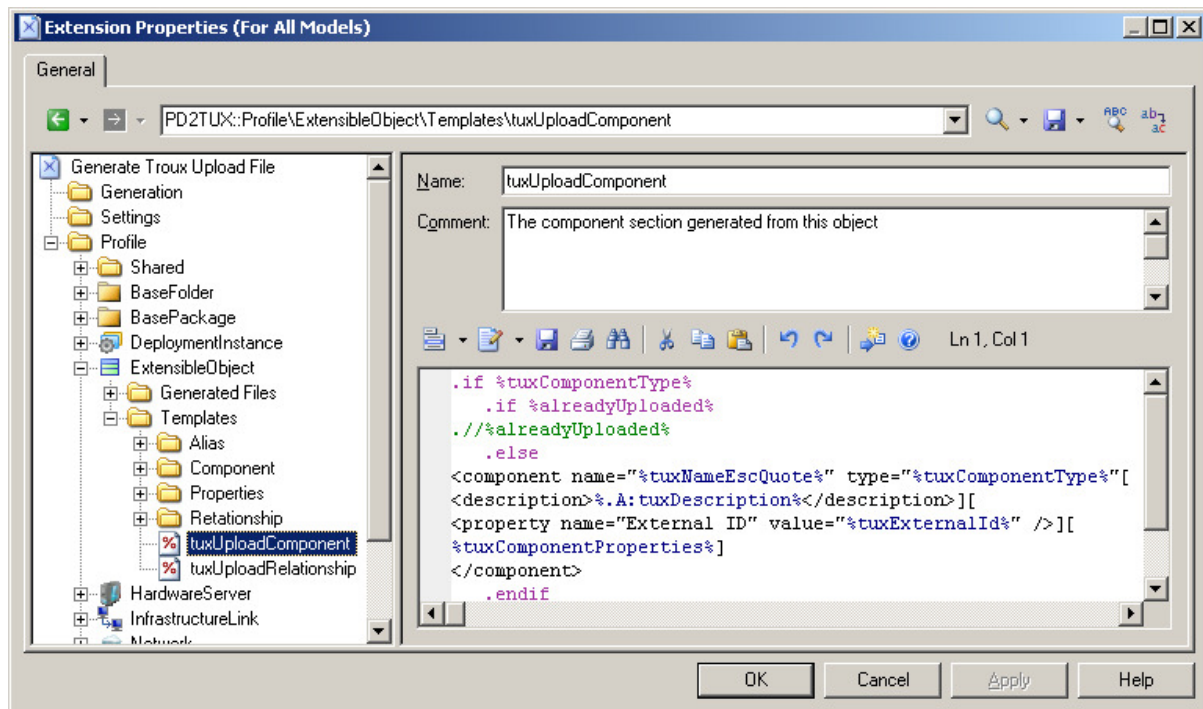
The TUX file to be generated is defined under the BasePackage metaclass:



Moving the text cursor on the template names (like %components% or %relationships%) and pressing F12 keys moves to the definition of the templates. The navigation bar on the top right corner () allows to go back to previous template definitions.

Generation of PowerDesigner Objects to Troux Components

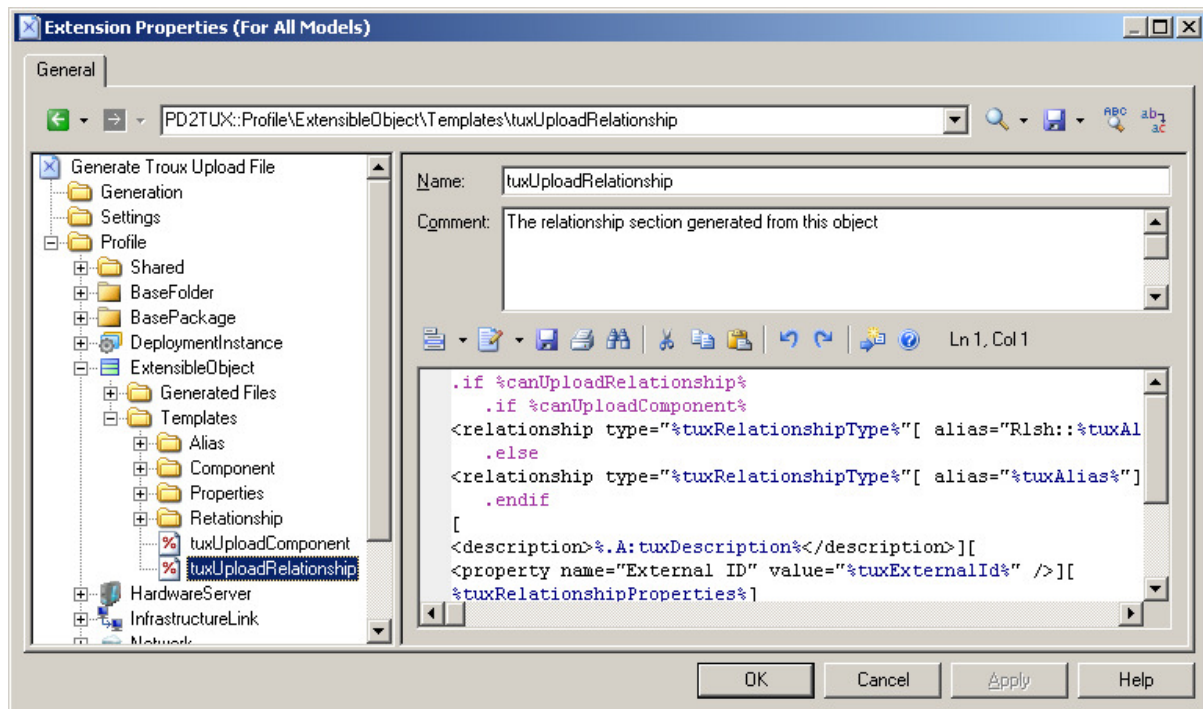
The template that generates a component from an object is defined under the ExtensibleObject metaclass that is a base class of all PowerDesigner objects:



A TUX component is generated for each PowerDesigner object if the “tuxComponentType” template is defined under its metaclass in the extension file (see Specifying a Mapping from a PowerDesigner Object to a Troux Component).

Generation of PowerDesigner Link Objects to Troux Relationships

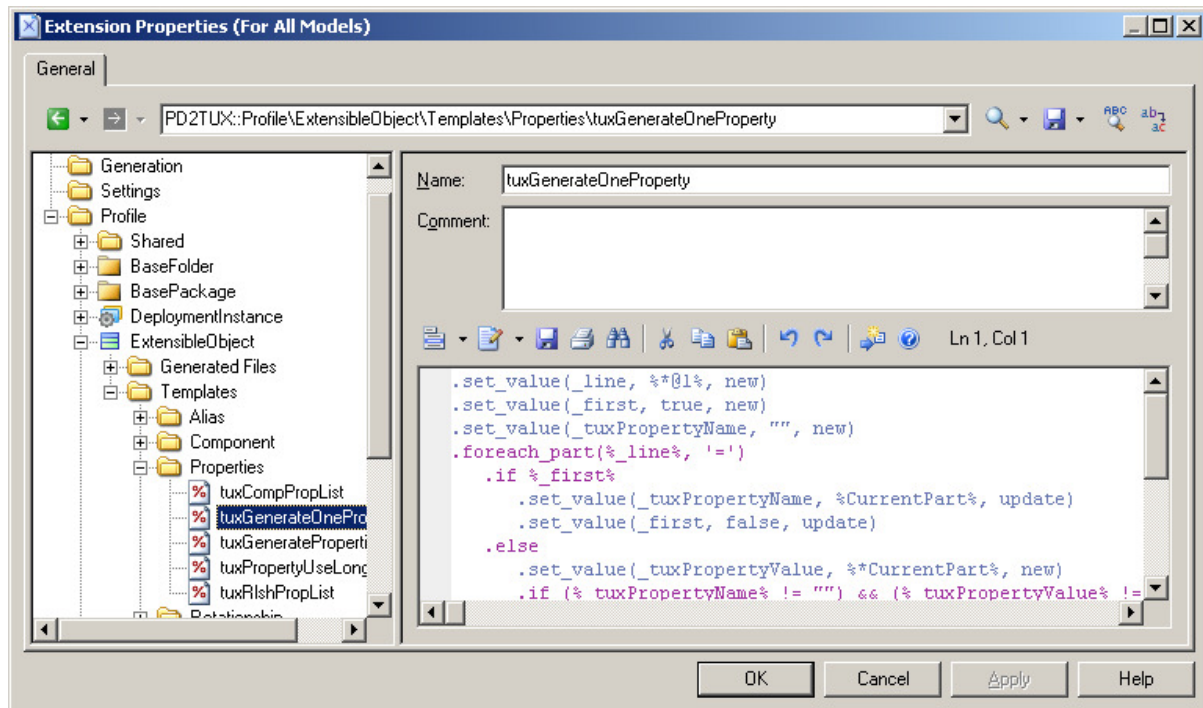
The template that generates a relationship from a link object is defined under the ExtensibleObject metaclass:



A TUX relationship is generated for each PowerDesigner link object if the “tuxRelationshipType” template is correctly defined under its metaclass in the extension file (see Specifying a Mapping from a PowerDesigner Link Object to a Troux Relationship) and if the objects on both its extremities are generated.

Generation of PowerDesigner Attributes to Troux Properties

The base template that generates attributes is defined under the ExtensibleObject metaclass:



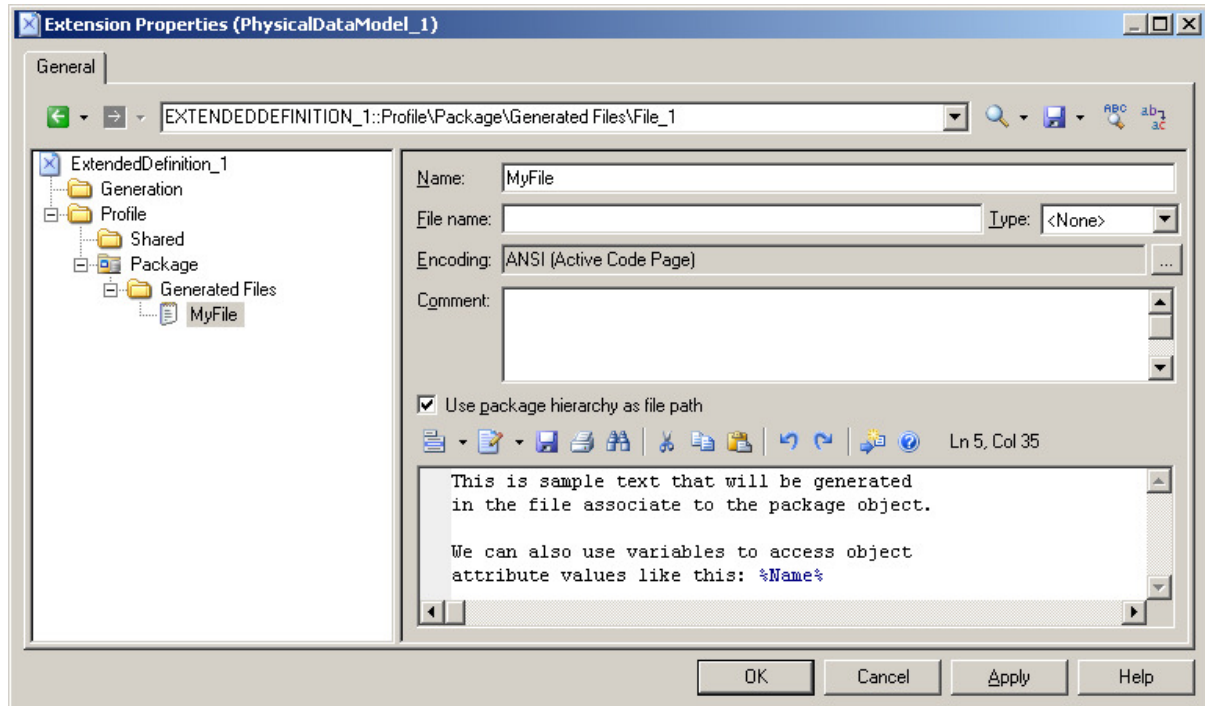
TUX properties are generated for each PowerDesigner attribute that has a mapping correctly defined in a “tuxCompPropList” (for components) or “tuxRlshPropList” (for relationships) under the appropriate metaclass in the extension file (see Specifying Mappings from PowerDesigner Attributes to Troux Properties).

Basics of the PowerDesigner GTL Language

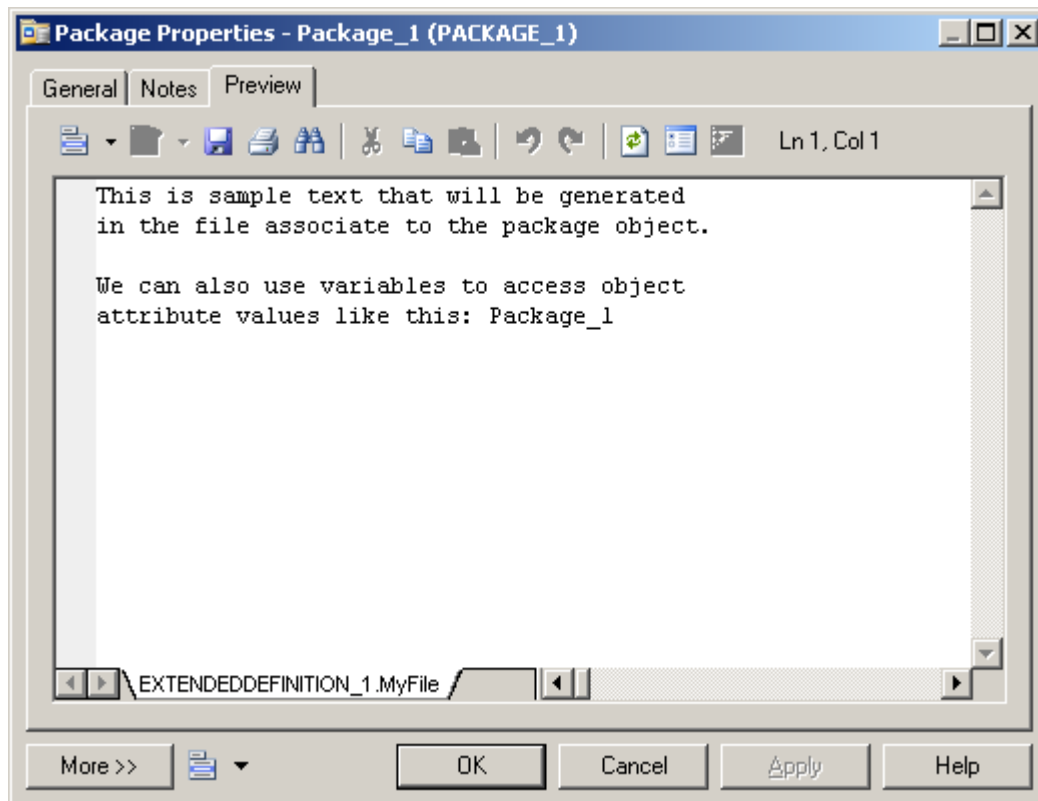
PowerDesigner generates text files from model objects using a simple language and framework called GTL (Generation Template Language).

This section presents a brief overview of the GTL language. For the complete reference guide, see **Help > Contents > Customizing and Extending PowerDesigner > Customizing Generation with GTL**.

To define a new generation, we need first to create a new extension file in a model that will contain the generation code and add a Generated file item on a metaclass:



Once a generated file has been defined in a model extension, you can generate it at **Tools > Extended Generation**, and the Preview tab on object property sheets can be used to preview what will be generated the result of the generation:



Generated Text

The text defined under the Generated file item in the extended model definition is generated into the file. As demonstrated in the screen shots above the text can access attributes of the current object. In the following example, the %Name% attribute is replaced by the value of the Name attribute on the object:

```
We can also use variables to access object
attribute values like this: %Name%
```

Generates the following code:

```
We can also use variables to access object
attribute values like this: Package_1
```

Conditions

GTL supports condition statements in the form:

```
.if %Name% == "Package_1"
I am the first package!
.else
I am not the first package...
.endif
```

The conditions are expressed using same logical operator than in C language: == and !=

We can also use brackets to condition the generation of a block of text to the first non empty variable:

```

This text is always generated.
[This text is generated only if comment value is not empty
%Comment%]

```

Loops

GTL supports looping on objects in a collection. In the following example, the text inside the `.foreach_item` statement changes the scope so that the `%Name%` and `%Priority%` variables are evaluated not on the `Package` object, but on the `Requirement` objects under the package:

```

.foreach_item(Requirements)
Requirement %Name% with priority %Priority%
.next(\n)

```

The previous code generates the following for a package containing 3 requirements:

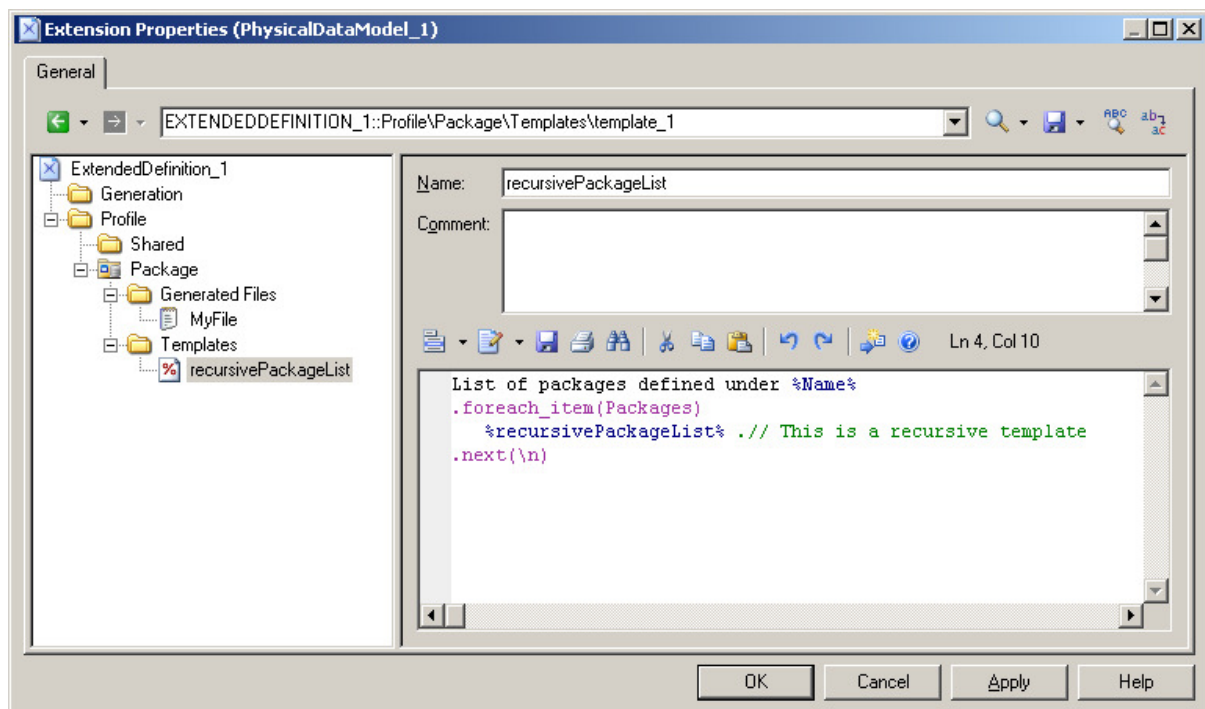
```

Requirement Requirement_1 with priority 1.500000
Requirement Requirement_2 with priority 2.000000
Requirement Requirement_3 with priority 1.000000

```

Templates

When the generation code becomes more complex, it can be useful to split it into several sub-definitions. We can create templates that store a part of the code generation (like sub-functions in a C program), and call them from a generated file or from another template as demonstrated below:



By convention, a variable starting with a lower case character is a template and a variable starting with an upper case is an attribute.