|  |
| --- |
|  |
| Modélisation UML |
| Patrons MVC pour PowerDesigner |
|  |
| **Thomas Auguey** |
| **20/08/2015** |

|  |
| --- |
| Guide pratique des patrons MVC de modélisation UML applicables aux applications en développement. |

SOMMAIRE

[Définition 3](#_Toc427833717)

[Applications cibles 3](#_Toc427833718)

[Stéréotypes 3](#_Toc427833719)

[Options de génération 3](#_Toc427833720)

[Model C# 5](#_Toc427833721)

[Introduction 5](#_Toc427833722)

[Options de génération 5](#_Toc427833723)

[Stéréotypes 5](#_Toc427833724)

[Persistance 6](#_Toc427833725)

[Implémentation 7](#_Toc427833726)

[Génération dans PowerDesigner 7](#_Toc427833727)

[Model Visionner 8](#_Toc427833728)

[Introduction 8](#_Toc427833729)

[Génération dans PowerDesigner 8](#_Toc427833730)

# Définition

Ce document est un support utilisateur aux Templates de langages PowerDesigner. Ces templates permettent de générer la partie MVC des model de données pour diverses applications.

Le but de ces modèles et de générer une partie du code correspondant au model de données et l’architecture de l’application. Le gain de temps est conséquent et le produit final n’impose aucune contrainte au programmeur. En effet, tout le code généré étant totalement éditable manuellement dans le cas ou PowerDesigner ne serait plus utilisé dans les versions futures du programme.

## Applications cibles

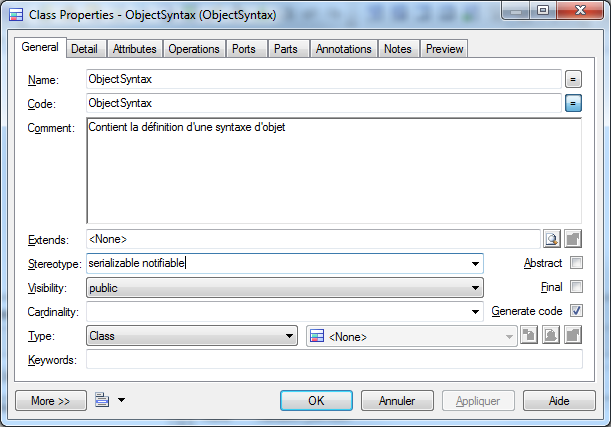
Ce tableau résume les modèles cibles d’applications :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Langage | Version | Framework | Fichier | Modèles cible |
| Visionner – Basic | 3.x | - |  | OOM et PDM |
| C# | - | - | Tpl\_CSharpApp.xol | OOM |
| C# | - | .Net | Tpl\_CSharpApp.xol | OOM |
| C++ | - | - | Tpl\_CPlusPlus.xol | OOM |
| C++ | - | wxWidget | Tpl\_CPlusPlus.xol | OOM |
| SQL Sever | 2012+ | - | Tpl\_SQLServer2012.xdb | PDM |
| Postgres SQL | 9 | - | Tpl\_pgsql9.xdb | PDM |

## Stéréotypes

Les stéréotypes sont utilisés dans les objets pour définir l’usage final dans le code. Par exemple, si les données d’une classes sont persistantes, serializables, validables, etc…

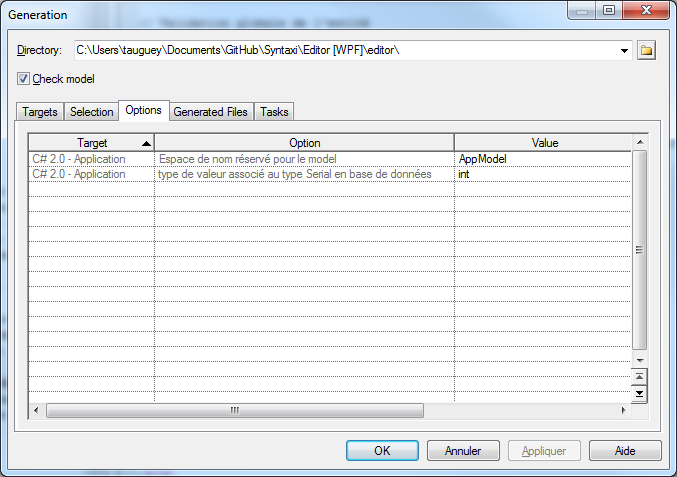
Les stéréotypes sont définissables dans les propriétés d’objets :



***Propriété d’une classe contenant les stéréotypes « serializable » et « validable »***

## Options de génération

Les options de générations sont spécifiques au langage cible, elles permettent d’ajuster la génération du code. Chaque option est définit plus loin dans son model.



***Options de génération dans le dialogue « Générer »***

# Model C#

Le model C# orienté objet permet la génération du model dans une application MVC type. Tout le code généré est indépendant de toutes librairies externes. Le code est seulement soumit aux contraintes du langage. Certaines implémentations additionnelles telles que WPF peut venir compléter l’architecture de l’application cible.

## Introduction

Le model exporte les classes de bases extraites du model UML.

L’intégration implémente :

* Les classes d’entités
* La validation des propriétés d’entités
* Les domaines de valeurs
* Les vues d’éditions
* Le squelette d’interfaces

Le schéma ci-dessous montre les répertoires cibles de l’arborescence générée.

|  |  |
| --- | --- |
| + RootDir  + %model namespace%  + Domain  + Format  + Entity  + Interface  + View  + View  + Lib | Dossier racine  Dossier correspondant à l’espace de nom du model cible  Enumération des domaines de valeurs  Classes de validation des formats de données  Classes des entités de données  Interfaces additionnelles du model  Vues du model (XAML)  Vues globales (XAML)  Interfaces et classes génériques du model de données |

## Options de génération

|  |  |  |
| --- | --- | --- |
| Nom | Option | Description |
| namespace | Espace de nom réservé pour le model | Indique le « **namespace** » de base pour toutes les classes générées |
| serial\_datatype | Type de valeur associée au type Serial en base de données | Type de donnée C# a faire correspondre avec le type Serial du model de données conceptuel. Il s’agit des index auto incrémenté en base de données (généralement du type **int** ou **long** suivant l’architecture de la BDD). |
| useDotNetValidation | Implémenter l’interface de validation .Net | Implémente également l’interface standard **IDataErrorInfo** pour les entités validable. |

## Stéréotypes

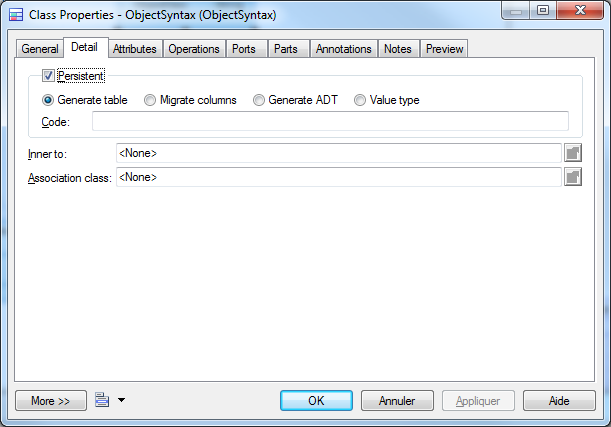
|  |  |
| --- | --- |
| Mot clé | Description |
| editable | Génère les vues **XAML** génériques pour l’édition et l’affichage des entités. |
| persistant | Implémente l’interface **IEntity** avec les méthodes d’accès à la base de données |
| notifiable | Implémente l’interface **IEntitySerializable** avec les méthodes de conversions binaire et XML |
| serializable | Implémente l’interface **INotifyPropertyChanged** avec les méthodes de notification des champs |
| validable | Implémente l’interface **IEntityValidable** avec les méthodes de validation des champs.  Utilisable avec l’option de génération **« useDotNetValidation»**. |

## Persistance

Il est possible de faire spécifier explicitement les noms des tables et champs en correspondance avec la classe généré. Par défaut le nom des tables est hérité des classes.

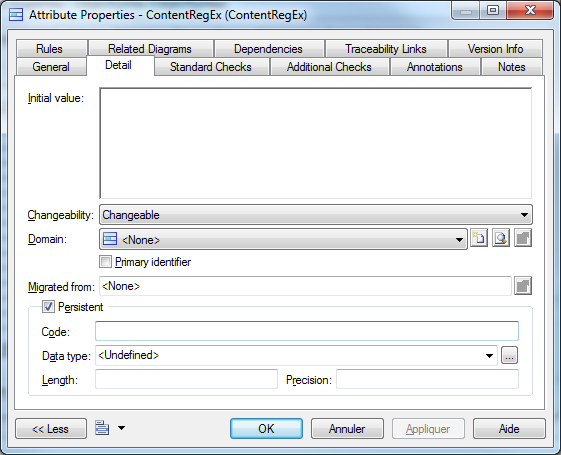
Spécifier le nom d’une table :

Rendez-vous dans le dialogue de propriété de classe sous l’onglet « **Détail** » et éditer le champ « **Code** » sous le groupe « **Persistent** ».



Spécifier le nom d’un champ :

Rendez-vous dans le dialogue de propriété de l’attribut sous l’onglet « Détail » et éditer le champ « Code » sous le groupe « Persistant ».



## Implémentation

### Base de données

Le programmeur peut implémenter sa propre interface **SGBD** basée sur l’interface **IEntityFactory**.

Les implémentations fournies sont :

|  |  |
| --- | --- |
| Fichier | Fournisseur |
| Lib/SqlODBCFactory.cs | Passerelle ODBC |
| Lib/SqlPostgresFactory.cs | Postgre SQL |
| Lib/SqlServerFactory.cs | SQL Server |

### Interface utilisateur

Les librairies d’exemples implémentent des contrôles utilisateurs dérivées de WPF.

Les implémentations fournies sont :

|  |  |  |
| --- | --- | --- |
| Classe | Classe de base | Implémentation |
| EditableDatagrid | DataGrid | Fournit les commandes de bases pour les entités : **Copier, Coller, Changements d’état** |

**Note : La génération des vues éditables requière une référence à la librairie « Xceed.WPF.Toolkit »**

## Génération dans PowerDesigner

Ce model est peut être générés depuis un « model conceptuel de données » ou directement édité.

# Model Visionner

## Introduction

Le model exporte la structure des tables et les fonctions d’accès génériques aux données SQL.

Le schéma ci-dessous montre les répertoires cibles de l’arborescence générée.

|  |  |
| --- | --- |
| + RootDir  + Model.txt  + Model.bas | Dossier racine  Model des structures de données  Fonctions « Basic » d’accés aux données |

## Génération dans PowerDesigner

Ce model est divisé en 2 module de langage « le model orienté objet » et « le model physique de données ». Ces 2 modèles peuvent être générés depuis un « model conceptuel de données » ou directement édité.