SYBASE®
An SAP® Company

Personnalisation et extension de PowerAMC

PowerAMC™ 16.0

Windows

ID DU DOCUMENT : DC20013-01-1600-01
DERNIERE REVISION : Juillet 2011

Sybase, Inc., One Sybase Drive, Dublin, CA 94568

Table des matières

Chapitre 1 Utilisation des fichiers de ressources PowerAMC

L'environnement de modélisation PowerAMC™ est alimenté par les fichiers de ressources au format XML, qui définissent les objets disponibles dans chaque modèle, avec les méthodes permettant leur génération et leur reverse engineering. Vous pouvez afficher, copier et éditer les fichiers de ressources fournis et créer les votre afin de personnaliser et d'étendre le comportement de l'environnement.

Les types de fichiers de ressources suivants sont fournis :

- *Fichier de définition* - définissent les objets standard disponibles dans un modèle :
  - *Fichiers de définition de SGBD* (.xdb) - définissent un SGBD particulier dans le MPD (voir *Chapitre 4, Fichiers de définition de SGBD* à la page 139).
  - *Fichiers de définition de langage de processus* (.xpl) – définissent un langage de processus métiers particulier dans le MPM (voir *Chapitre 3, Fichiers de définition pour les langage objet, de processus et XML* à la page 127).
  - *Fichiers de définition de langage objet* (.xol) - définissent un langage objet particulier dans le MOO (voir *Chapitre 3, Fichiers de définition pour les langage objet, de processus et XML* à la page 127).
  - *Fichiers de définition de langage XML* (.xsl) - définissent un langage XML particulier dans le MSX (voir *Chapitre 3, Fichiers de définition pour les langage objet, de processus et XML* à la page 127).
- *Fichiers d'extension* (.xem) – étendent la définitions standard des langages cible afin, par exemple, de spécifier un environnement de persistance ou un serveur dans un MOO. Vous pouvez créer ou attacher un ou plusieurs fichiers XEM à un modèle (voir *Chapitre 2, Fichiers d'extension* à la page 23).
- *Modèles de rapport* (.rtp) - spécifient la structure d'un rapport. Modifiables à l'aide de l'Editeur de modèle de rapport (voir *Guide des fonctionnalités générales > L'interface de PowerAMC > Rapports*).
- *Fichiers de langue de rapport* (.xrl) – traduisent les en-têtes et autres textes standard dans un rapport (voir *Chapitre 6, Traduction de rapports à l'aide des fichiers de ressource de langue de rapport* à la page 329).
- *Jeux de règles d'analyse d'impact et de lignage* (.rul) - spécifient les règles définies pour la génération d'analyses d'impact et de lignage (voir *Guide des fonctionnalités générales > Liaison et synchronisation de modèles > Analyse d'impact et de lignage*).
- *Profils de permissions sur les objets* (.ppf) - personnalisent l'interface de PowerAMC afin de masquer des modèles, des objets et des propriétés (voir *Guide des fonctionnalités générales > Administration de PowerAMC > Personnalisation de l'interface de PowerAMC*).

- *Profils utilisateur* (.upf) - stockent les préférences relatives aux options de modèle, options générales, préférences d'affichage, etc (voir *Guide des fonctionnalités générales > L'interface de PowerAMC > Personnalisation de votre environnement de modélisation > Profils utilisateur*).
- *Jeux de catégories de modèle* (.mcc) - personnalisent la boîte de dialogue Nouveau modèle afin de guider la création de modèle (voir *Guide des fonctionnalités générales > Administration de PowerAMC > Personnalisation de l'interface de PowerAMC > Personnalisation de la boîte de dialogue Nouveau modèle*).
- *Tables de conversion* (.csv) - définissent des conversions entre le nom et le code d'un objet (voir *Guide des fonctionnalités générales > L'interface de PowerAMC > Objets > Propriétés d'un objet > Conventions de dénomination*).

Ces fichiers de ressources sont basés sur le métamodèle public PowerAMC (voir *Métamodèle public PowerAMC* à la page 9).

Vous pouvez passer en revue tous les fichiers de ressources disponibles à partir de la liste des fichiers de ressources, disponible en sélectionnant **Outils > Ressources > *Type***.

Les outils suivants sont disponibles dans chaque type de fichier de ressources :

Outil	Description
	Propriétés - Ouvre le fichier de ressources dans l'éditeur de ressources
	Nouveau - Crée un nouveau fichier de ressources en utilisant un fichier original comme modèle (voir *Création et copie de fichiers de ressources* à la page 7).
	Enregistrer - Enregistre le fichier de ressources sélectionné.
	Enregistrer tout - Enregistre tous les fichiers de ressources de la liste.
	Chemin - Permet de sélectionner le répertoire qui contient le fichier de ressources.
	Comparer - Permet de sélectionner deux fichiers de ressources à comparer (voir *Comparaison des fichiers de ressources* à la page 7).
	Fusionner - Permet de sélectionner deux fichiers de ressources à fusionner (voir *Fusion de fichiers de ressources* à la page 8).
	Consolider - [si le référentiel est installé] Consolide le fichier de ressource sélectionné dans le référentiel. Pour plus d'informations sur le stockage de fichiers de ressources dans le référentiel, voir *Guide des fonctionnalités générales > Administration de PowerAMC > Déploiement de fichiers de ressources partagés*.
	Mettre à jour à partir du référentiel - [si le référentiel est installé] Extrait une version du fichier sélectionné depuis le référentiel sur votre machine locale.
	Comparer avec la version du référentiel - [si le référentiel est installé] Compare le fichier sélectionné avec un fichier de ressource stocké dans le référentiel.

Ouverture de fichiers de ressources dans l'Editeur de ressources

Lorsque vous travaillez avec un MPM, MPD, MOO ou MSX, vous pouvez ouvrir dans l'Editeur de ressources le fichier de définition qui contrôle les objets disponibles dans votre modèle afin d'en visualiser ou modifier le contenu. Vous pouvez également ouvrir et éditer les fichiers d'extension attachés ou incorporés à votre modèle ou accéder à la liste de ressources appropriées et ouvrir n'importe quel fichier de ressources PowerAMC.

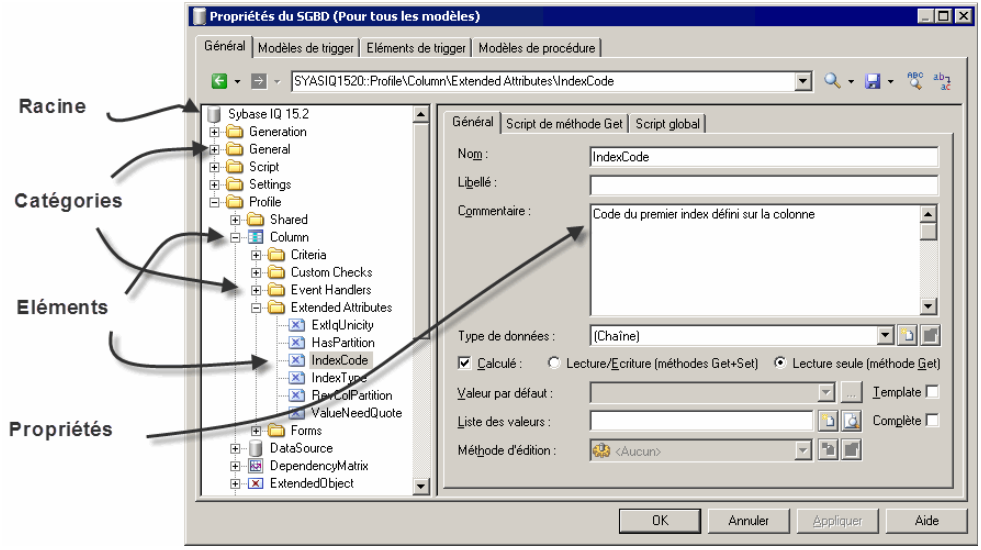
Pour afficher un fichier de définition utilisé par votre modèle :

- Dans un MPD, sélectionnez **SGBD > Editer le SGBD courant**.
- Dans un MPM, sélectionnez **Langage > Editer le langage de processus courant**.
- Dans un MOO, sélectionnez **Langage > Editer le langage objet courant**.
- Dans un MSX, sélectionnez **Langage > Editer le langage courant**.

Pour ouvrir un fichier d'extension attaché à votre modèle, double-cliquez sur l'entrée correspondante dans la catégorie **Extensions** dans l'Explorateur d'objets.

Pour ouvrir un autre fichier de ressources, sélectionnez **Outils > Ressources > *Type*** pour afficher la liste de ressources appropriée, sélectionnez un fichier, puis cliquez sur l'outil **Propriétés**.

Dans chaque cas, le fichier s'affiche dans l'Editeur de ressources, dans lequel vous pouvez passer en revue et éditer la structure de la ressource. Le volet de gauche montre une arborescence d'entrées contenues dans le fichier de ressources, tandis que le volet de droite affiche les propriétés de l'élément sélectionné :



Remarque : Ne modifiez pas les fichiers de ressources fournis avec PowerAMC. Si vous souhaitez modifier un fichier, créez une copie en utilisant l'outil **Nouveau** (voir *Création et copie de fichiers de ressources* à la page 7).

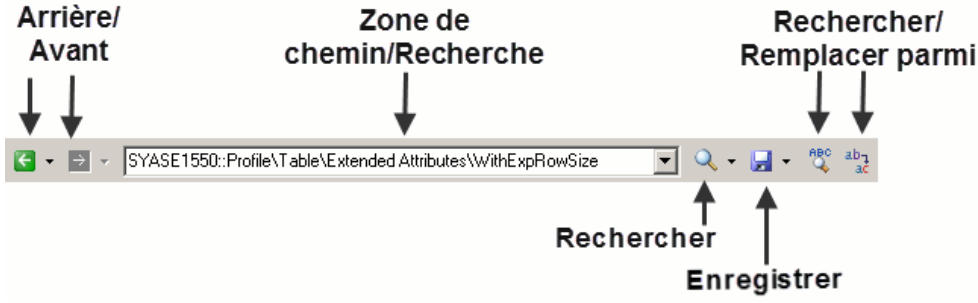
Chaque entrée est une partie de la définition d'un fichier de ressources, et les entrées sont organisées en catégories logiques. Par exemple, la catégorie Script dans un fichier de définition de SGBD collecte toutes les entrées liées à la génération et au reverse engineering de base de données.

Vous pouvez faire glisser des catégories ou des entrées dans l'arborescence de l'éditeur de ressources, ainsi qu'entre des éditeurs de ressources du même type (par exemple, entre deux éditeurs de fichier XOL).

Remarque : Certains fichiers de ressources sont fournis avec la mention "Not certified" dans leur nom. Sybase® s'efforce de procéder à tous les contrôles de validation possibles, toutefois, Sybase n'assure pas la maintenance d'environnements spécifiques permettant la certification complète de ces fichiers de ressources. Sybase assure le support de la définition en acceptant les rapports de bogues et fournit les correctifs nécessaires dans le cadre d'une politique standard, mais ne peut être tenu de fournir une validation finale de ces correctifs dans l'environnement concerné. Les utilisateurs sont donc invités à tester ces correctifs fournis par Sybase afin de signaler d'éventuelles incohérences qui pourraient subsister.

Navigation et recherche dans les fichiers de ressources

Les outils situés en haut de l'Editeur de ressources permettent de naviguer dans les fichiers de ressources et d'y effectuer des recherches.



Outil	Description
	Arrière (**Alt+Gauche**) - Affiche l'entrée ou la catégorie visitée précédente. Cliquez sur la flèche vers le bas pour sélectionner directement dans votre historique.
	Avant (**Alt+Droite**) - Affiche l'entrée ou la catégorie visitée suivante. Cliquez sur la flèche vers le bas pour sélectionner directement dans votre historique.

Outil	Description
	Recherche (**Entrée**) - Affiche l'élément nommé dans la zone de texte. Si plusieurs éléments sont trouvés, ils sont affichés sous forme de liste dans une boîte de résultats, et vous pouvez alors double-cliquer sur l'élément souhaité ou le sélectionner et cliquer sur **OK** pour l'afficher. Cliquez sur la flèche vers le bas pour définir les options de recherche : • [type d'extension] - sélectionnez le type d'extension à rechercher, par exemple vous pouvez faire porter la rechercher uniquement sur les stéréotypes • Permettre l'utilisation des caractères génériques - Permet d'utiliser des métacaractères `*` pour renvoyer n'importe quelle chaîne et `?` pour renvoyer n'importe quel caractère unique. Par exemple saisissez `is*` pour retrouver toutes les extensions appelées `is...`. • Respect de la casse - Prend en compte la casse lors de la recherche.
	Enregistrer (**Ctrl+Maj+S**) – Enregistre le fichier de ressources courant. Cliquez sur la flèche vers le bas pour enregistrer le fichier sous un nouveau nom.
	Rechercher parmi les éléments (**Ctrl+Maj+F**) - Recherche du texte dans les entrées.
	Remplacer parmi les éléments (**Ctrl+Maj+H**) - Recherche et remplace du texte dans les entrées.

Remarque : Pour passer directement à la définition d'un template à partir d'une référence située dans un champ d'un autre template, placez votre curseur entre les signes pourcent et appuyez sur **F12** (voir *Templates and Generated Files (Profile)* à la page 114).

Afficher la super-définition

Si une extension redéfinit un autre élément, vous pouvez utiliser la commande Afficher la super-définition dans le menu contextuel de l'objet correspondant pour accéder à l'élément redéfini.

Edition de fichiers de ressources

Lorsque vous pointez sur une catégorie ou une entrée puis cliquez le bouton droit de la souris dans l'arborescence du fichier de ressources, les options d'édition suivantes s'affichent :

Option d'édition	Description
Nouveau	Permet d'ajouter une entrée utilisateur

Option d'édition	Description
Ajouter des éléments...	Affiche une boîte de dialogue de sélection permettant de sélectionner des catégories ou entrées de métamodèle prédéfinies afin de les ajouter dans le noeud courant. Vous ne pouvez pas modifier le nom de ces éléments, mais vous pouvez modifier leurs commentaires et valeurs en sélectionnant le noeud correspondant.
Effacer	Supprime la catégorie et/ou l'entrée sélectionnée.
Restaurer le commentaire	Restaure le commentaire par défaut de la catégorie ou de l'entrée sélectionnée
Restaurer la valeur	Restaure la valeur de l'entrée sélectionnée.

Remarque : Vous pouvez renommer une catégorie ou une entrée définie par l'utilisateur directement dans l'arborescence du fichier de ressources en sélectionnant l'élément approprié, puis en appuyant sur la touche **F2**.

Enregistrement des modifications

Si vous modifiez un fichier de ressources, puis cliquez sur OK pour fermer l'éditeur de ressources sans cliquer sur l'outil Enregistrer, les changements sont enregistrés en mémoire, l'éditeur se ferme et vous revenez à la liste des fichiers de ressources. Ensuite, lorsque vous cliquez sur Fermer dans la liste des fichiers de ressources, une boîte de confirmation vous demande si vous souhaitez enregistrer le fichier de ressources modifié. Si vous cliquez sur Oui, les modifications sont enregistrées dans le fichier de ressources lui-même. Si vous cliquez sur Non, les modifications sont conservées en mémoire jusqu'à la fermeture de la session de PowerAMC.

La prochaine fois que vous ouvrirez un modèle qui utilise le fichier de ressources personnalisé, les modifications seront prises en compte par le modèle. Toutefois, si vous avez au préalable modifié les mêmes options directement dans le modèle, les valeurs contenues dans le fichier de ressources ne modifient pas ces options.

Partage et incorporation de fichiers de ressources

Les fichiers de ressources peuvent être partagés et référencés par plusieurs modèles ou bien copiés dans un fichier de modèle pour y être incorporés. Les modifications apportées à un fichier de ressources partagé sont disponibles pour tous les modèles qui utilisent cette ressource, tandis que celles que vous effectuez dans une ressource incorporé ne sont disponibles que pour le modèle dans lequel elle est incorporée. Les fichiers de ressources incorporés sont enregistrés comme faisant partie du modèle qui les contient, et non sous la forme d'un fichier distinct.

Remarque : Ne modifiez pas les extensions d'origine fournies avec PowerAMC. Pour créer une copie d'un fichier à modifier, affichez la boîte de dialogue Liste des extensions, cliquez sur l'outil **Nouveau**, spécifiez un nom pour le nouveau fichier, puis sélectionnez le fichier .xem que vous souhaitez modifier dans la zone **Copier depuis**.

La zone **Nom de fichier** affiche l'emplacement du fichier de ressource que vous modifiez. Cette zone est vide si le fichier de ressource est incorporé.

Création et copie de fichiers de ressources

Vous pouvez créer un nouveau fichier de de ressources dans la liste de fichiers de ressources appropriée. Pour créer une copie d'un fichier de ressources existant, sélectionnez-le dans la zone **Copier depuis** de la boîte de dialogue **Nouveau...**.

Avertissement ! Chaque fichier de ressources est doté d'un ID unique, vous devez donc copier les fichiers de ressources uniquement depuis PowerAMC, pas dans l'Explorateur Windows.

1. Sélectionnez **Outils** > **Ressources** > ***Type*** pour afficher la liste des ressources appropriée.
2. Cliquez sur l'outil **Nouveau**, saisissez un nom pour le nouveau fichier et sélectionnez un fichier existant à copier. Sélectionnez `<Template par défaut>` pour créer un fichier de ressource avec le contenu minimal.
3. Cliquez sur **OK** afin de créer le nouveau fichier de ressource, puis spécifiez un nom de fichier et cliquez sur **Enregistrer** afin de l'ouvrir dans l'Editeur de ressources.

   Remarque : Vous pouvez créer un fichier d'extension directement dans votre modèle à partir de la boîte de dialogue Liste des extensions. Pour plus d'informations, voir *Création d'un fichier d'extension* à la page 26.

Comparaison des fichiers de ressources

Vous pouvez sélectionner deux fichiers de ressources et les comparer afin d'identifier les différences entre ces deux fichiers.

1. Sélectionnez **Outils** > **Ressources** > ***Type*** pour afficher la liste de fichiers de ressources appropriée.
2. Sélectionnez le premier fichier de ressource à comparer dans la liste, puis cliquez sur l'outil **Comparer** afin d'ouvrir une boîte de dialogue de sélection.

   Le fichier sélectionné est affiché dans la second zone de comparaison.
3. Sélectionnez l'autre fichier de ressources à comparer dans la première zone de comparaison.

   Si le fichier de ressources que vous souhaitez comparer ne se trouve pas dans la liste, cliquez sur l'outil **Sélectionner un chemin** et sélectionnez le répertoire qui contient le fichier de ressources désiré.

4. Cliquez sur **OK** pour ouvrir la boîte de dialogue **Comparer...**, qui permet de passer en revue les différences entre les fichiers.

   Pour plus d'informations sur cette fenêtre, voir *Guide des fonctionnalités générale > L'interface de PowerAMC > Comparaison et fusion de modèles*.

5. Examinez les différences, puis cliquez sur **Fermer** pour fermer la fenêtre de comparaison et revenir à la liste.

Fusion de fichiers de ressources

Vous pouvez sélectionner deux fichiers de ressources de même type et les fusionner. La fusion s'effectue de gauche à droite : le fichier de ressources situé dans le volet droit est comparé à celui situé dans le volet gauche, les différences sont mises en évidence et des actions de fusion sont proposées dans le fichier de ressources de droite.

1. Sélectionnez **Outils > Ressources > *Type*** pour afficher la liste de ressources appropriée.
2. Sélectionnez un fichier de ressources que vous souhaitez modifier dans la liste, puis cliquez sur l'outil **Fusionner** pour afficher une boîte de dialogue de sélection.

   Le fichier sélectionné est affiché dans la zone **Vers**.
3. Sélectionnez le fichier de ressources à partir duquel vous souhaitez effectuer la fusion dans la zone **Depuis**.

   Si le fichier que vous souhaiter fusionner ne se trouve pas dans la liste, cliquez sur l'outil **Sélectionner un chemin** et sélectionnez le répertoire qui contient le fichier de ressources désiré.

4. Cliquez sur **OK** pour ouvrir la boîte de dialogue **Fusionner...**, qui permet de passer en revue les actions de fusion avant de les valider.

Pour obtenir des informations détaillées sur cette fenêtre, voir *Guide des fonctionnalités générale > L'interface de PowerAMC > Comparaison et fusion de modèles*.

5. Sélectionnez ou rejetez les actions de fusion proposées, puis cliquez sur **OK** pour effectuer la fusion.

Métamodèle public PowerAMC

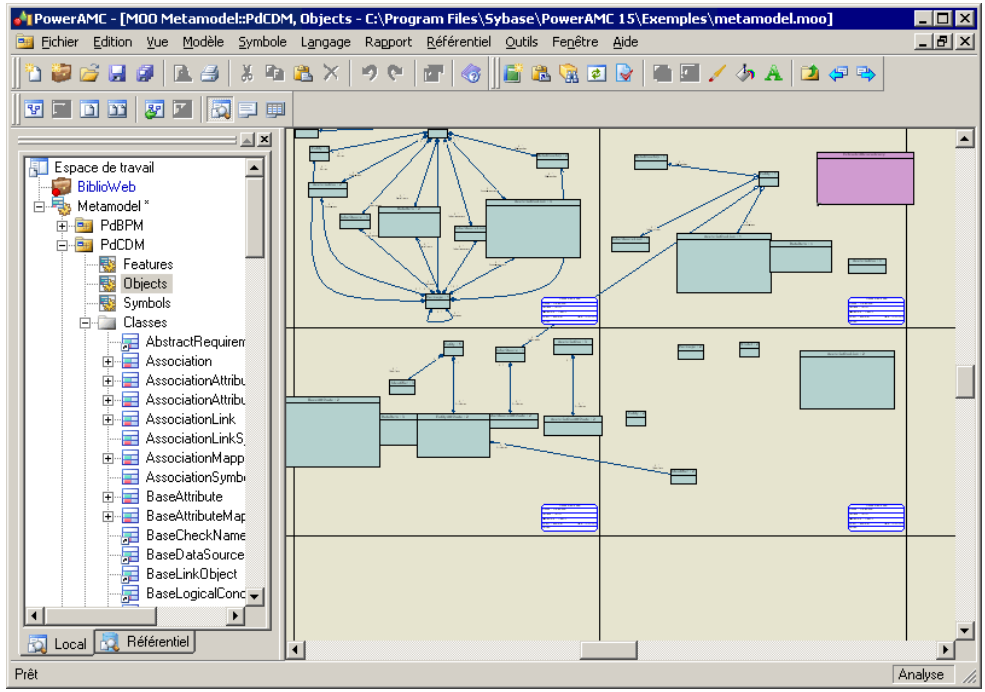
Un métamodèle décrit les éléments d'un modèle, ainsi que la syntaxe et la sémantique de la notation qui permet leur manipulation. Un modèle est une abstraction des données, et peut être décrit à l'aide de métadonnées. Un métamodèle est une abstraction des métadonnées.

Le métamodèle public PowerAMC est une abstraction des métadonnées pour tous les modèles PowerAMC qui est représentée dans un modèle orienté objet. Ce métamodèle est destiné à vous aider à comprendre la structure générale des métadonnées de modélisation PowerAMC lorsque vous utilisez :

- des scripts VB scripts
- des templates de langage de génération par template (Generation Template Language, GTL)
- des fichiers de modèle XML PowerAMC (voir *Fichiers de modèle et métamodèle PowerAMC* à la page 16)

Le MOO du métamodèle public est situé à l'emplacement suivant :

```
[Répertoire d'installation de PowerAMC]\Exemples\MetaModel.moo
```



Pour obtenir une documentation, sélectionnez **Aide > Aide sur les objets du métamodèle**.

Le métamodèle est réparti dans les principaux packages suivants :

- PdBPM - Modèle de Processus Métiers (MPM)
- PdCDM - Modèle Conceptuel de Données (MCD)
- PdCommon - contient tous les objets communs à au moins deux modèles, ainsi que les classes abstraites du modèle. Par exemple, les règles de gestion, qui sont disponibles dans tous les modèles, et la classe BaseObject, à partir de laquelle tous les objets sont dérivés, sont définies dans ce package. Les autres packages de modèle sont liés à PdCommon via des liens de généralisation indiquant que chaque modèle hérite des objets communs du package PdCommon.
- PdEAM - Modèle d'Architecture d'Entreprise (MAE)
- PdFRM - Modèle libre (MLB
- PdILM - Modèle de Fluidité de l'Information (MFI)
- PdLDM - Modèle Logique de Données (MLD)
- PdMTM - Modèle des Traitements Merise
- PdOOM - Modèle Orienté Objet (MOO)
- PdPDM - Modèle Physique de Données (MPD)
- PdPRJ - Projet
- PdRMG - Référentiel

- PdRQM - Modèle de gestion des exigences (MGX)
- PdXSM - Modèle XML
- PdWSP - Espace de travail

Chacun de ces packages racine contient les types de sous-objets suivants, organisés par diagramme ou, dans le cas de PdCommon, en sous-packages :

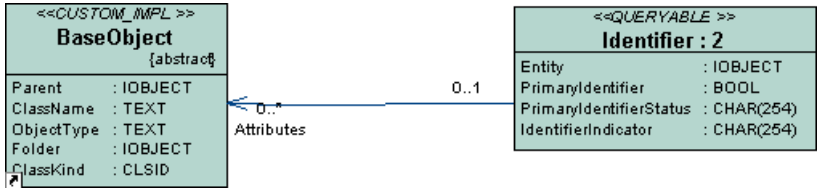
- Features - Toutes les fonctionnalités mises en oeuvre par des classes dans le modèle. Par exemple, Report (disponible dans tous les modèles) appartient à PdCommon, et AbstractDataType appartient à PdPDM.
- Objects - Objets de conception dans le modèle
- Symbols - Représentation graphique des objets de conception

Concepts relatifs au métamodèle

Le métamodèle public de PowerAMC utilise les concepts UML standard :

- *Noms publics* - Chaque objet du métamodèle PowerAMC a un nom et un code qui correspondent au nom public de l'objet. Le nom public d'un objet est identificateur unique de cet objet dans la bibliothèque du modèle ou dans un package (par exemple, par exemple PdCommon) visible dans le diagramme Modules du métamodèle. Les noms publics sont également utilisés dans les fichiers XML (voir *Fichiers de modèle et métamodèle PowerAMC* à la page 16) ainsi que dans le langage de génération par template (voir *Chapitre 5, Personnalisation de la génération à l'aide du langage de génération par template* à la page 285). Le nom public ne correspond pas toujours au nom de l'objet dans l'interface de PowerAMC.
- *Classes* - Les classes sont utilisées pour représenter les métadonnées de la façon suivante
  - *Classes abstraites* : elles sont utilisées pour partager des attributs et comportements. Elles ne sont pas visibles dans l'interface PowerAMC. Les classes pouvant être instanciées héritent des classes abstraites via des liens de généralisation. Par exemple, NamedObject est une classe abstraite, elle stocke les attributs standard tels que name, code, comment, annotation et description hérités par la plupart des objets de conception PowerAMC
  - Classes instanciables*/concrètes* : elles correspondent aux objets affichés dans l'interface, elles sont dotées de leurs propres attributs tels que type ou persistance, et héritent des attributs et comportements des classes abstraites via les liens de généralisation
- *Attributs de classes* - Les attributs sont des propriétés de classe qui peuvent être *dérivées* ou non. Les classes liées à d'autres classes via des liens de généralisation contiennent le plus souvent des attributs dérivés qui sont calculés à partir des attributs ou des collections de la classe parent. Les attributs non dérivés sont les attributs propres de la classe. Ces attributs sont stockés dans le modèle et enregistrés dans le fichier du modèle.
- *Associations* - Les associations sont utilisées pour exprimer les connexions sémantiques entre des classes appelées *collections*. Dans la feuille de propriétés d'une association, les rôles transportent l'information relative à l'objet d'extrémité de l'association. Dans le

métamodèle PowerAMC, ce rôle a le même nom qu'une collection pour l'objet courant. Les objets PowerAMC sont liés à d'autres objets via des collections.
En règle générale, les associations n'ont qu'un seul rôle, ce rôle se trouve à l'opposé de la classe qui représente une collection. Dans l'exemple suivant, Identifier a une collection appelée Attributes :



Lorsque les associations ont deux rôles, les deux collections ne peuvent pas être enregistrées dans le fichier XML, seule la collection ayant le rôle *navigable* sera enregistrée (voir *Fichiers de modèle et métamodèle PowerAMC* à la page 16).

- *Composition* - Expriment une association dans laquelle les enfant vivent et meurent avec les parents. Si le parent est copié, l'enfant l'est également. Par exemple, dans le package PdCommon, diagramme Option Lists, la classe NamingConvention est associée avec la classe BaseModelOptions via trois associations de composition : NameNamingConventions, CodeNamingConventions et NamingConventionsTemplate. Ces associations de composition expriment le fait que la classe NamingConvention n'existerait pas sans la classe BaseModelOptions.
- *Généralisations* - Montrent les liens d'*héritage* entre une classe plus générale (le plus souvent une classe abstraite) et une classe plus spécifique (le plus souvent une classe instanciable). La classe la plus spécifique hérite des attributs de la classe plus générique, ces attributs étant appelés attributs dérivés.
- *Commentaires et notes sur les objets* - Expliquent le rôle de l'objet dans le métamodèle. Certains détails de mise en oeuvre interne sont également disponibles dans la page **Notes** > **Annotation** de la feuille de propriétés des classes.

Navigation dans le métamodèle

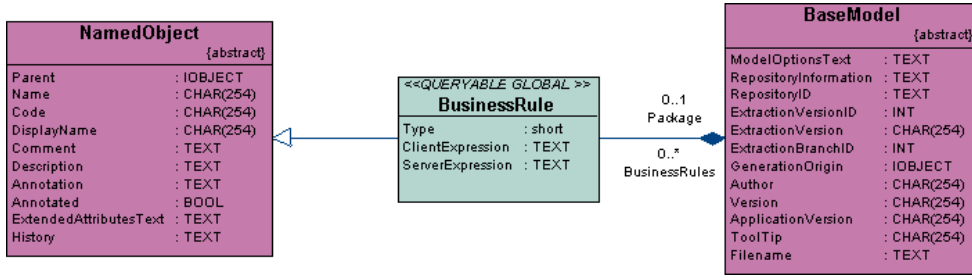
Vous pouvez utiliser l'interface graphique du MOO pour étendre et réduire les packages afin d'explorer leur contenu. Pour afficher un diagramme dans la zone de travail, il vous suffit de double-cliquer sur son icône.

Chaque diagramme montre des classes reliées entre elles par le biais d'associations et de généralisations. Chaque classe a un nom (le nom public) et peut être décrite par des attributs. Elle peut assumer plusieurs rôles dans les associations avec d'autres classes. De nombreuses associations affichent leurs rôles afin de permettre l'identification des collections d'objets (voir *Concepts relatifs au métamodèle* à la page 11).

Les classes en *vert* sont des classes dont le comportement est expliqué dans le diagramme courant, tandis que les classes en *mauve* sont le plus souvent des raccourcis d'une classe existant dans un autre package. Le raccourci facilite la lecture du diagramme et la compréhension des liens de généralisation entre les classes. Si vous souhaitez obtenir une explication relative à une classe mauve dans un diagramme, pointez sur cette classe, cliquez le

bouton droit de la souris et sélectionnez *Ouvrir un diagramme associé* pour afficher le diagramme dans lequel la classe est définie.

Dans l'exemple courant issu de Common Instantiable Objects dans le package Objects de PdCommon, BusinessRule (couleur verte) est développé tandis que NamedObject et BaseModel sont utilisés pour exprimer les liens d'héritage et de composition à l'aide de classes abstraites.



Vous pouvez double-cliquer sur une classe pour afficher sa feuille de propriétés. L'onglet *Dépendances* contient notamment les sous-onglets suivants :

- *Associations* : vous pouvez personnaliser le filtre afin d'afficher le rôle des associations, vous obtenez ainsi la liste des collections de l'objet courant
- *Généralisations* : affiche une liste des liens de généralisation dans lesquelles l'objet courant est le parent. Cette liste permet d'afficher les enfants de la classe courante. Les classes enfant héritent des attributs de la classe parent et n'affichent pas d'attribut dérivé
- *Spécialisations* : affiche le parent de l'objet courant. La classe courante hérite des attributs de ce parent
- *Raccourcis* : affiche la liste des raccourcis créés pour l'objet courant

L'onglet *Associations* répertorie les associations migrées pour la classe courante.

Utilisation du métamodèle avec VB Script

Vous pouvez accéder aux objets internes de PowerAMC et les manipuler à l'aide de VB Script. Le métamodèle PowerAMC (et son aide en ligne, accessible en sélectionnant**Aide > Aide sur les objets du métamodèle**) fournit des informations utiles relatives à ces objets :

Information	Description
Nom public	Le nom et le code des objets du métamodèle sont les noms publics des objets internes de PowerAMC. Exemples : AssociationLinkSymbol, ClassMapping, CubeDimensionAssociation

Information	Description
Collections d'objets	Vous pouvez identifier les collections d'une classe en observant les associations liées à cette classe dans le diagramme. Le rôle de chaque association est le nom de la collection. Exemples : Dans PdBPM, l'association Format relie les classes MessageFormat et MessageFlow. Le rôle de cette association est Usedby, qui correspond à la collection de messages de la classe MessageFormat
Attributs d'objet	Vous pouvez afficher les attributs d'une classe avec les attributs que cette classe hérite d'une autre classe via des liens de généralisation Exemples : Dans PdCommon/Common Instantiable Objects, vous pouvez afficher les attributs de BusinessRule, FileObject et ExtendedDependency, ainsi que ceux dont ils héritent des classes abstraites via des liens de généralisation
Opérations d'objet	Les opérations dans des classes d'un métamodèle correspondent aux méthodes objet utilisées dans VBS. Exemples : BaseModel contient l'opération Compare qui peut être utilisée dans VBS
Stéréotype <<notScriptable>>	Objets qui ne prennent pas en charge les scripts VB qui ont le stéréotype <<notScriptable>>. Exemples : RepositoryGroup

Pour plus d'informations sur les noms publics et autres concepts liés au métamodèle, voir *Concepts relatifs au métamodèle* à la page 11.

Pour plus d'informations sur l'utilisation de VB Script avec PowerAMC, voir *Chapitre 7, Pilotage de PowerAMC à l'aide de scripts* à la page 349.

Accès au métamodèle à l'aide du langage de génération par template

Le langage de génération par template (*Generation Template Language*, GTL) utilise des *templates* afin de générer des fichiers. Un template est un fragment de code défini sur une métaclasse PowerAMC et les métaclasses qui héritent de cette classe. Il peut être utilisé dans différents contextes pour la génération de texte et de code.

Ces templates peuvent être considérés comme des extensions de métamodèle car ils sont des types particuliers d'attributs de classes de métamodèle. Vous pouvez définir autant de templates que nécessaire pour n'importe quelle métaclasse donnée en utilisant la syntaxe suivante :

```
<métamodèle-nomclasse> / <nom-template>
```

Les templates sont hérités par tous les descendants de la métaclasse pour laquelle ils sont définis, ce qui permet de les utiliser afin de partager du code de template entre différentes métaclasses ayant un ancêtre commun. Par exemple, si vous définissez un template pour la

classe abstraite BaseObjects, toutes les classes liées via des liens de généralisation à cette classe héritent de ce template.

Le langage de génération par template utilise des macros telles que foreach_item, afin de procéder à l'itération dans les collections d'objets. Le template spécifié dans le bloc est converti sur tous les objets contenus dans la collection spécifiée. Le métamodèle fournit des informations très utiles concernant les collections de métaclasses sur lesquelles vous définissez un template contenant une macro d'itération.

Les attributs calculés suivants sont des extensions de métamodèle spécifiques au langage de génération par template :

Métaclasse	Attributs
PdCommon.BaseObject	• isSelected (boolean) - True si l'objet correspondant fait partie de la sélection dans la boîte de dialogue de génération • isShorctut (boolean) - True si l'objet était accessible via un raccourci
PdCommon.BaseModel	• GenOptions (struct) - Permet d'accéder aux options de génération définies par l'utilisateur
PdOOM.*	• ActualComment (string) - Commentaire supprimé (avec /**, /*, */ et // supprimés)
PdOOM.Association	• RoleAMinMultiplicity (string) • RoleAMaxMultiplicity (string) • RoleBMinMultiplicity (string) • RoleBMaxMultiplicity (string)
PdOOM.Attribute	• MinMultiplicity (string) • MaxMultiplicity (string) • Overridden (boolean) • DataTypeModifierPrefix (string) • DataTypeModifierSuffix (string) • @<tag> [spécifique Java] (string) - Attribut étendu Javadoc@<tag> avec formatage supplémentaire
PdOOM.Class	• MinCardinality (string) • MaxCardinality (string) • SimpleTypeAttribute [XML-specific] • @<tag> [spécifique Java] (string) - Attribut étendu Javadoc@<tag> avec formatage supplémentaire

Métaclasse	Attributs
PdOOM.Interface	• @<tag> [spécifique Java] (string) - Attribut étendu Java-doc@<tag> avec formatage supplémentaire
PdOOM.Operation	• DeclaringInterface (object) • GetSetAttribute (object) • Overridden (boolean) • ReturnTypeModifierPrefix (string) • ReturnTypeModifierSuffix (string) • @<tag> [spécifique Java] (string) - Attribut étendu Java-doc@<tag> avec formatage supplémentaire (particulièrement pour @throws, @exception, @params)
PdOOM.Parameter	• DataTypeModifierPrefix (string) • DataTypeModifierSuffix (string)

Les collections calculées suivantes sont des extensions de métamodèle spécifiques au langage de génération par template :

Nom de métaclasse	Nom de collection
PdCommon.BaseModel	Generated <metaclass-name>List - Collection de tous les objets du type <metaclass-name> qui font partie de la sélection dans la boîte de dialogue de génération
PdCommon. BaseClassifier-Mapping	SourceLinks
PdCommon. BaseAssociation-Mapping	SourceLinks

Fichiers de modèle et métamodèle PowerAMC

Les modèles PowerAMC sont composés d'objets dont les propriétés et interactions sont expliquées dans le métamodèle public. Les modèles peuvent être enregistrés soit au format binaire, soit au format XML. Les fichiers binaires sont plus petits, et leur ouverture et leur enregistrement est significativement plus rapide, mais les fichiers de modèle XML peuvent être édités à la main ou par voie logicielle (et des DTD sont fournis pour chaque type de modèle dans le dossier `DTD` du répertoire d'installation).

Avertissement ! Vous pouvez modifier un fichier de modèle XML en utilisant un éditeur de texte ou un éditeur XML, mais vous devez procéder avec précautions, car la plus petite erreur de syntaxe risque de rendre le fichier inutilisable. Si vous créez un objet dans un fichier XML en faisant appel au copier-coller, prenez garde de bien supprimer l'OID dupliqué. PowerAMC va automatiquement affecter un OID au nouvel objet dès que vous allez ouvrir ce modèle.

Les éléments suivants sont utilisés dans le fichiers XML PowerAMC :

- <o:*object*> - Objet de modèle PowerAMC. La première fois que l'objet est mentionné dans une collection, PowerAMC lui affecte un ID à l'aide de la syntaxe <o:*object* Id="*XYZ*"> (dans laquelle *XYZ* est un identificateur unique affecté à un objet la première fois qu'il est rencontré) ou le référence à l'aide de la syntaxe <o:object Ref="*XYZ*"/>. Une définition d'objet n'est utilisée que dans des collections de type composition, dans lesquelles l'objet parent possède les enfants dans l'association.
- <c:*collection*> - Collection d'objets liés à un autre objet. Vous pouvez utiliser le métamodèle PowerAMC afin de visualiser les collections d'un objet. Par exemple, <c:Children>.
- <a:*attribute*> - Un objet est composé de plusieurs attributs que vous pouvez modifier de façon indépendante. Par exemple, <a:ObjectID>.

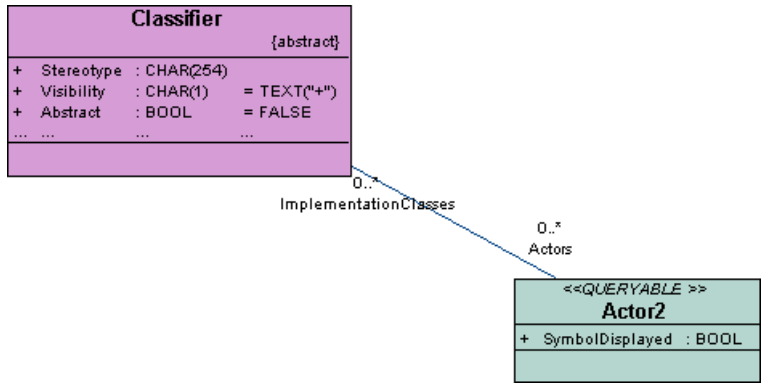
Les fichiers de modèle XML de PowerAMC ont un élément <o:model> à leur racine, qui contient les collections définies dans le métamodèle de PowerAMC. L'objet modèle et tous les autres éléments d'objet qui le contiennent définissent leurs attributs et collections dans des sous-éléments. La définition d'un objet implique la définition de ses attributs et collections. PowerAMC vérifie chaque objet et analyse en profondeur les collections de cet objet pour définir chaque nouvel objet et collection dans ces collections, et ainsi de suite, jusqu'à ce que le processus trouve les objets terminaisons qui ne nécessitent pas d'analyse supplémentaire.

Vous pouvez chercher un objet dans le métamodèle en utilisant son nom d'objet dans le fichier XML afin de mieux comprendre sa définition. Une fois vous que vous avez trouvé l'objet dans le métamodèle, vous pouvez lire les informations suivantes :

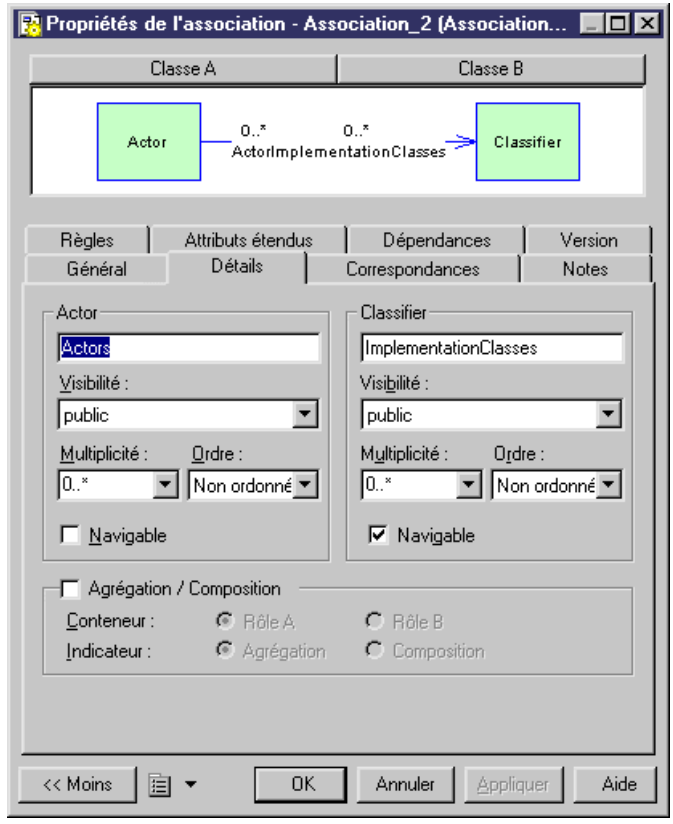
- Chaque objet PowerAMC peut comporter plusieurs collections correspondant aux autres objets avec lesquels il doit interagir. Ces collections sont représentées par les associations existant entre objets. Les *rôles* des associations (agrégations et compositions incluses) correspondent aux collections d'un objet. Par exemple, chaque modèle PowerAMC contient une collection de domaines appelée Domains.

  En règle générale, les associations n'ont qu'un seul rôle, le rôle s'affiche à l'opposé de la classe pour laquelle il représente une collection. Toutefois, le métamodèle contient également des associations ayant deux rôles, auquel cas, les deux collections ne peuvent pas être enregistrées dans le fichier XML. Vous pouvez identifier la collection qui sera enregistrée à partir de la feuille de propriétés de l'association : il s'agit du rôle pour lequel la case *Navigable* est cochée.

  Dans l'exemple suivant, les association ont deux rôles qui signifient que Classifier a une collection Actors, et que Actor2 a une collection ImplementationClasses :



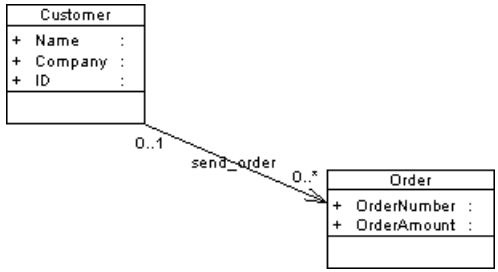
Si vous affichez la feuille de propriétés de l'association, vous pouvez voir que la case Navigable est cochée pour le rôle ImplementationClass, ce qui signifie que seule la collection ImplementationClass sera enregistrée dans le fichier.



- Les attributs ayant le type de données *IOBJECT* sont des attributs dans le métamodèle alors qu'ils apparaissent sous forme de collections contenant un seul objet dans le fichier XML. Ce n'est pas le cas pour Parent et Folder qui ne contiennent pas de collection.

Exemple : Fichier XML correspondance à un MOO simple

Le modèle suivant contient deux classes et une association. Nous allons explorer le fichier XML correspondant à ce modèle.



Le fichier commence par plusieurs lignes qui spécifient des détails relatifs à XML et au modèle.

Le premier objet qui apparaît est la racine du modèle <o:RootObject Id="01">. RootObject est un conteneur de modèle qui est défini par défaut lorsque vous créez et enregistrez un modèle. RootObject contient une collection appelée Children qui est composée de modèles.

Dans notre exemple, Children ne contient qu'un objet de modèle qui est défini comme suit :

```
<o:Model Id="o2">
 <a:ObjectID>3CEC45F3-A77D-11D5-BB88-0008C7EA916D</a:ObjectID>
 <a:Name>ObjectOrientedModel_1</a:Name>
 <a:Code>OBJECTORIENTEDMODEL_1</a:Code>
 <a:CreationDate>1000309357</a:CreationDate>
 <a:Creator>arthur</a:Creator>
 <a:ModificationDate>1000312265</a:ModificationDate>
 <a:Modifier>arthur</a:Modifier>
 <a:ModelOptionsText>
[ModelOptions]
...
```

Sous la définition de l'objet modèle, vous pouvez voir la série d'attributs ModelOptions. Remarquez que ModelOptions n'est pas limité aux options définies dans la boîte de dialogue Options du modèle d'un modèle, mais rassemble toutes les propriétés enregistrées dans un modèle, notamment les options relatives à la génération intermodèle.

Après ModelOptions, vous pouvez identifier la collection <c:ObjectLanguage>. Il s'agit du langage objet lié au modèle. La seconde collection du modèle est <c:ClassDiagrams>. Il s'agit de la collection des diagrammes liés au modèle. Dans notre exemple, un seul diagramme est défini dans le paragraphe suivant :

```
<o:ClassDiagram Id="o4">
   <a:ObjectID>3CEC45F6-A77D-11D5-BB88-0008C7EA916D</a:ObjectID>
   <a:Name>ClassDiagram_1</a:Name>
   <a:Code>CLASSDIAGRAM_1</a:Code>
   <a:CreationDate>1000309357</a:CreationDate>
   <a:Creator>arthur</a:Creator>
   <a:ModificationDate>1000312265</a:ModificationDate>
```

```
   <a:Modifier>arthur</a:Modifier>
   <a:DisplayPreferences>
...
```

Tout comme dans le cas des options de modèle, la définition ClassDiagram est suivie d'une série d'attributs de préférences d'affichage.

Dans la collection ClassDiagram se trouve une nouvelle collection appelée <c:Symbols>. Cette collection rassemble tous les symboles contenus dans le diagramme du modèle. Le premier objet à être défini dans la collection Symbols est AssociationSymbol :

```
<o:AssociationSymbol Id="o5">
   <a:CenterTextOffset>(1, 1)</a:CenterTextOffset>
   <a:SourceTextOffset>(-1615, 244)</a:SourceTextOffset>
   <a:DestinationTextOffset>(974, -2)</a:DestinationTextOffset>
   <a:Rect>((-6637,-4350), (7988,1950))</a:Rect>
   <a:ListOfPoints>((-6637,1950),(7988,-4350))</a:ListOfPoints>
   <a:ArrowStyle>8</a:ArrowStyle>
   <a:ShadowColor>13158600</a:ShadowColor>
   <a:FontList>DISPNAME 0 Arial,8,N
```

AssociationSymbol contient les collections <c:SourceSymbol> et <c:DestinationSymbol>. Dans ces deux collections, les symboles font l'objet de références mais ne sont pas définis, car ClassSymbol n'appartient pas aux collections SourceSymbol et DestinationSymbol.

```
<c:SourceSymbol>
    <o:ClassSymbol Ref="o6"/>
   </c:SourceSymbol>
   <c:DestinationSymbol>
    <o:ClassSymbol Ref="o7"/>
   </c:DestinationSymbol>
```

La collection des symboles d'association est suivie par la collection <c:Symbols>. Cette collection contient la définition des deux symboles de classe.

```
<o:ClassSymbol Id="o6">
   <a:CreationDate>1012204025</a:CreationDate>
   <a:ModificationDate>1012204025</a:ModificationDate>
   <a:Rect>((-18621,6601), (-11229,12675))</a:Rect>
   <a:FillColor>16777215</a:FillColor>
   <a:ShadowColor>12632256</a:ShadowColor>
   <a:FontList>ClassStereotype 0 Arial,8,N
```

La collection <c:Classes> suit la collection <c:Symbols>. Dans cette collection, les deux classes sont définies avec leurs collections d'attributs.

```
<o:Class Id="o10">
   <a:ObjectID>10929C96-8204-4CEE-911#-E6F7190D823C</a:ObjectID>
   <a:Name>Order</a:Name>
   <a:Code>Order</a:Code>
   <a:CreationDate>1012204026</a:CreationDate>
   <a:Creator>arthur</a:Creator>
   <a:ModificationDate>1012204064</a:ModificationDate>
   <a:Modifier>arthur</a:Modifier>
```

```
   <c:Attributes>
  <o:Attribute Id="o14">
```

L'attribut est un objet terminal : aucune ramification supplémentaire n'est nécessaire pour en détailler la définition.

Chaque collection appartenant à un objet analysé est développée et analysée, y compris les collections contenues dans d'autres collections.

Une fois tous les objets et toutes les collections parcourus, les balises suivantes s'affichent :

```
</o:RootObject>
</Model>
```

Chapitre 2 Fichiers d'extension

Les fichiers d'extension (`*.xem`) permettent de personnaliser et d'étendre les métaclasses, paramètres et génération de PowerAMC. Vous pouvez utiliser des extensions afin de définir des propriétés supplémentaires pour des types de données existants ou de nouveaux types d'objets, pour modifier l'interface de PowerAMC (en réorganisant et ajoutant des onglets de feuilles de propriétés, des outils de palette et des commandes de menus), mais aussi afin de définir des cibles et options de génération supplémentaires.

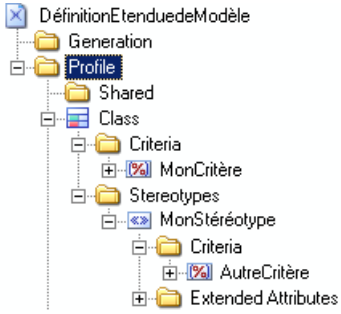
PowerAMC fournit des fichiers d'extension prédéfinis et vous pouvez créer vos propres fichiers d'extension. Chaque fichier d'extension contient deux catégories de premier niveau :

- *Generation* - permet de développer et de compléter la génération d'objets par défaut de PowerAMC (pour les MPM, MOO et MSX) ou de créer une génération séparée. Pour plus d'informations, voir *Catégorie Generation* à la page 132.
- *Profile* - un mécanisme d'extension UML, qui est utilisé afin d'étendre un métamodèle pour une cible particulière. Les profils sont utilisés dans PowerAMC afin d'ajouter des métadonnées supplémentaires aux objets et pour créer de nouveaux types de liens entre eux, pour sous-diviser les types d'objet (via les stéréotypes et critères), personnaliser les symboles, menus et formulaires, et pour modifier les résultats de génération. Par exemple :
  - Fichier de ressources du langage objet Java 5.0 - étend la métaclasse Component avec plusieurs niveaux de critères afin de modéliser différentes formes d'EJB.
  - Fichier de ressources du langage de processus BPEL4WS 1.1 - étend la métaclasse Event via des stéréotypes pour modéliser des événements Compensation, Fault et Timer
  - Fichier de ressources SGBD MSSQLSRV2005 - utilise des objets étendus stéréotypés afin de modéliser les agregates, les assemblies et autres objets spécifiques à SQL Server

  Vous pouvez étendre le métamodèle des façons suivantes :
  - Ajoutez ou sous-classifiez de nouveaux types d'objets :
    - Métaclasses – tirées du métamodèle comme base pour l'extension.
    - Stéréotypes [pour les métaclasses et les stéréotypes uniquement] – pour sous-classifier les objets.
    - Critères – Pour évaluer des conditions afin de sous-classifier des objets.
    - Objets, sous-objets et liens étendus – pour créer de nouveaux types d'objet.
  - Fournissez de nouveaux moyens de visualiser les connexions entre objets :
    - Matrices de dépendances – pour montrer les connexions entre deux types d'objets.
    - Collections et compositions étendues – pour permettre de lier manuellement des objets.

    - Collections calculées – pour lier automatiquement des objets.
- Ajoutez de nouvelles propriétés aux objets et affichez-les :
    - Attributs étendus – pour fournir des métadonnées supplémentaires.
    - Formulaires – pour afficher des onglets ou des feuilles de propriétés personnalisés.
    - Symboles personnalisés – pour vous aider à distinguer visuellement des objets.
- Ajoutez des contraintes et des règles de validation sur les objets :
    - Vérifications personnalisées – pour tester les données.
    - Gestionnaires d'événement – pour appeler des méthodes lorsqu'un événement se produit.
- Exécutez des commande sur les objets :
    - Méthodes – pour être appelées par d'autres extensions de profil telles que des menus et boutons de formulaire (écrites en VBScript).
    - Menus [pour les métaclasses et stéréotypes uniquement] – pour personnaliser les menus PowerAMC.
- Générez des objets de nouvelles manières :
    - Templates et fichiers générés – pour personnaliser la génération.
    - Transformations et profils de transformation – pour automatiser les changements sur les objets lors de la génération ou à la demande.

Vous pouvez visualiser et éditer le profil dans un fichier de ressource en ouvrant ce dernier dans l'Editeur de ressources et en développant la catégorie Profile racine. Vous pouvez ajouter des extensions à une métaclasse (un type d'objet, telle que Class dans un MOO ou Table dans un MPD), ou à un stéréotype ou critère, qui a été précédemment défini sur une métaclasse :

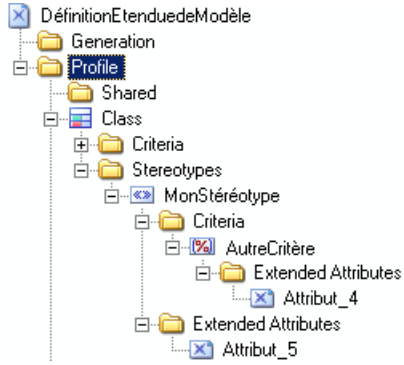


Dans l'exemple ci-dessus :

- Class est une métaclasse. Les métaclasses proviennent du métamodèle PowerAMC, et apparaissent toujours à la racine, immédiatement sous la catégorie Profile.
- MonCritère est un critère qui affine la métaclasse Class. Les classes qui remplissent le critère peuvent être présentées et traitées différemment des autres classes.
- MonStéréotype est un stéréotype qui affine la métaclasse Class. Les classes ayant le stéréotype MonStéréotype peuvent être présentées et traitées différemment des autres classes.

- AutreCritère est un critère qui affine encore plus les classes portant le stéréotype MonStéréotype. Les classes ayant le stéréotype ET qui remplissent le critère peuvent être présentées et traitées différemment des classes qui on seulement le stéréotype.

Les extensions sont héritées, ainsi les extensions d'une métaclasses sont disponibles pour ses enfants stéréotypés, et par celles auxquelles s'applique le critère.



Ainsi, dans l'exemple ci-avant, les classes ayant le stéréotype MonStéréotype ont l'attribut étendu Attribut_5, tandis que ceux qui ont ce stéréotype ET remplissent le critère AutreCritère ont l'attribut Attribut_4 et l'attribut Attribut_5

Remarque : Etant donné que vous pouvez attacher plusieurs fichiers de ressource à un modèle (par exemple, un langage cible et un ou plusieurs fichiers d'extension) vous pouvez créer des conflits, dans lesquels plusieurs extensions portant un nom identique (par exemple, deux définitions de stéréotype différentes) sont définies sur la même métaclasse dans des fichiers de ressources distincts. Si un tel conflit se produit, le fichier d'extension prévaut le plus souvent. Lorsque deux extensions sont en conflit, la priorité va à celui qui apparaît le premier dans la liste.

Création, attachement et incorporation de fichiers d'extension

Les extensions peuvent être des fichiers `*.xem` indépendants attachés aux modèles ou peuvent être incorporées dans des fichiers de modèle. Les fichiers d'extension indépendants peuvent être référencés par plusieurs modèles, et les changements que vous y apportez sont partagés par tous les modèles auxquels vous les attachez. Les modifications que vous apportez aux extensions incorporées dans un fichier de modèle n'affectent que ce seul modèle.

Création d'un fichier d'extension

Vous pouvez créer un fichier d'extension à partir de la liste des fichiers d'extension ou directement incorporé dans votre modèle.

Remarque : Pour plus d'informations sur la création d'un fichier d'à partir de la liste des fichiers d'extension, voir *Création et copie de fichiers de ressources* à la page 7.

1. Ouvrez votre modèle, puis sélectionnez **Modèle > Extensions** pour afficher la boîte de dialogue Liste des extensions.
2. Cliquez sur l'outil **Ajouter une ligne** et saisissez un nom pour le nouveau fichier d'extension.
3. Cliquez sur l'outil **Propriétés** pour ouvrir le nouveau fichier d'extension dans l'Editeur de ressources, et créez les extensions appropriées.
4. Lorsque vous avez terminé, cliquez sur **OK** pour enregistrer vos modifications et revenir à la boîte de dialogue Liste des extensions.

   Le nouveau fichier XEM est initialement incorporé dans votre modèle, et ne peut pas être partagé avec un autre modèle. Pour plus d'informations sur l'exportation de vos extensions et leur mise à disposition à des fins de partage, voir *Exportation d'un fichier d'extension incorporé à partager* à la page 27.

Attachement d'extensions à un modèle

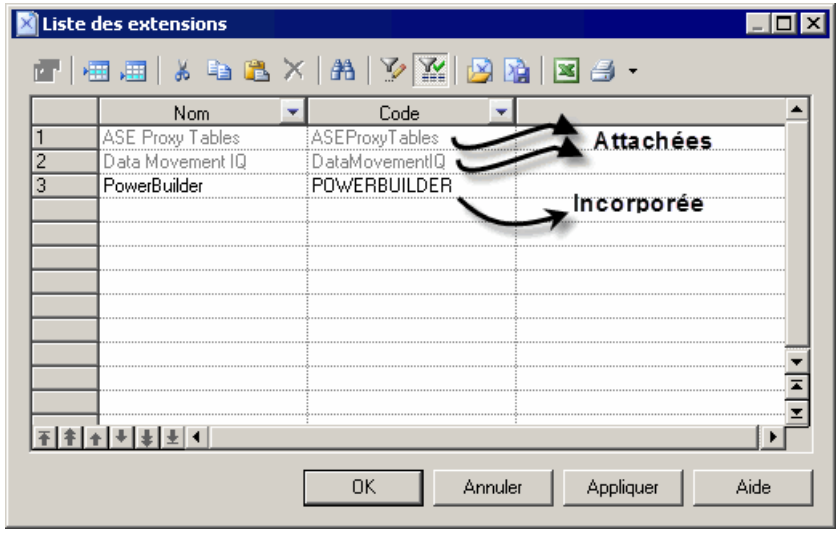
Vous pouvez attacher un fichier d'extension (fichier .xem) à votre modèle lorsque vous le créez en cliquant sur le bouton **Sélectionner des extensions** dans la boîte de dialogue Nouveau modèle. Vous pouvez ensuite attacher un fichier d'extension à votre modèle à tout moment à partir de la boîte de dialogue Liste des extensions.

Remarque : Vous ne devez jamais modifier les extensions d'origine fournies avec PowerAMC. Pour créer une copie du fichier à modifier, affichez la boîte de dialogue Liste des extensions, cliquez sur l'outil **Nouveau**, spécifiez un nom pour le nouveau fichier et sélectionnez le fichier .xem que vous souhaitez modifier dans la zone **Copier depuis**.

1. Sélectionnez **Modèle > Extensions** pour afficher la boîte de dialogue Liste des extensions.
2. Cliquez sur l'outil **Importer** pour afficher la boîte de dialogue Sélection des extensions.
3. Passez en revue les différentes sortes d'extensions disponibles en cliquant sur les sous-onglets, puis sélectionnez-en une ou plusieurs à attacher à votre modèle.

   Par défaut, PowerAMC crée un lien dans le modèle vers le fichier spécifié. Pour copier le contenu du fichier d'extension et le coller dans votre fichier de modèle, cliquez sur le bouton **Incorporer la ressource dans le modèle** dans la barre d'outils. Le fait d'incorporer un fichier de cette façon permet de faire en sorte que toute modification apportée à la ressource est spécifique à ce modèle et n'a aucune incidence sur les autres modèles qui font référence à la ressource partagée

4. Cliquez sur **OK** pour revenir à la boîte de dialogue Liste des extensions.



Les extensions répertoriées en gris sont attachées au modèle, celles qui sont répertoriées en noir sont incorporées dans le modèle.

Remarque : Si vous importez un fichier d'extension et l'incorporez dans le modèle, le nom et le code de l'extension peuvent être modifiés afin de respecter les conventions de dénomination de la catégorie Autres objets figurant dans la boîte de dialogue Options du modèles.

Exportation d'un fichier d'extension incorporé à partager

Si vous exportez une extension créée dans un modèle, elle devient disponible dans la boîte de dialogue Liste des extensions, et peut être partagée avec d'autres modèles. Lorsque vous exportez une extension, l'original reste incorporé dans le modèle.

1. Sélectionnez **Modèle > Extensions** pour ouvrir la boîte de dialogue Liste des extensions.
2. Sélectionnez une extension dans la liste.
3. Cliquez sur l'outil **Exporter une extension**.
4. Saisissez un nom et sélectionnez un répertoire pour le fichier d'extension.
5. Cliquez sur Enregistrer.

   L'extension est enregistrée dans un répertoire bibliothèque dans lequel elle peut être partagée avec d'autres modèles.

Propriétés d'une extension

Tous les fichiers d'extension ont la même structure de catégories de base.

Le noeud racine de chaque fichier contient les propriétés suivantes :

Propriété	Description
Nom	Spécifie le nom de l'extension. Ce nom doit être unique dans un modèle pour des extensions génériques ou spécifiques.
Code	Spécifie le code de l'extension. Ce code doit être unique dans un modèle pour les extensions génériques ou spécifiques
Nom de fichier	[lecture seule] Spécifie le chemin et nom du fichier de l'extension. Si l'extension a été copiée dans votre modèle, cette zone est vide.
Famille	Restreint la disponibilité de l'extension à une famille cible particulière. Par exemple, lorsqu'une extension a comme famille JAVA, elle n'est disponible que pour les cibles de la famille de langage objet JAVA
Sous-famille	Affine la famille. Par exemple, EJB 2.0 est une sous-famille de Java
Rattachement automatique	Spécifie que l'extension correspondante sera automatiquement attachée aux modèles créés avec une cible appartenant à la famille spécifiée
Catégorie	Regroupe les extensions par type pour la génération ainsi que dans la boîte de dialogue Sélection d'extensions. Les extensions de même catégorie ne peuvent pas être générées simultanément. Si vous ne spécifiez aucune catégorie, l'extension est affichée dans la catégorie Général et traitée comme une cible de génération.
Activer le suivi	Permet d'afficher un aperçu des templates utilisés lors de la génération. Avant de commencer la génération, cliquez sur la page Aperçu de la feuille de propriétés de l'objet approprié, puis cliquez sur le bouton Réactualiser pour afficher ces templates Lorsque vous double-cliquez sur une ligne de suivi dans la page Aperçu, l'éditeur de ressources ouvre la définition correspondante dans la catégorie Profile\Object \Templates
Compléter la génération de langage	Spécifie que l'extension est utilisée pour compléter la génération d'un langage cible. Les éléments de génération des langages objets sont fusionnés avec ceux de l'extension avant la génération. Tous les fichiers générés spécifiés dans le fichier de ressource cible et les éventuelles extensions attachées sont générées. Dans le cas de fichiers générés ayant des noms identiques, le fichier de l'extension remplace celui défini dans le langage objet. Notez que PowerBuilder ne prend pas en charge les extensions pour compléter la génération
Commentaire	Fournit des commentaires relatifs à l'extension

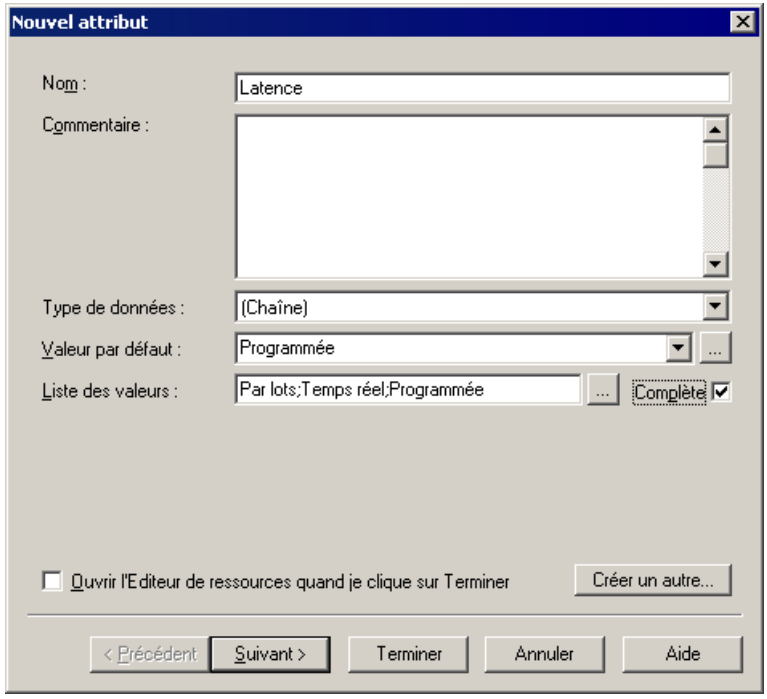
Les catégories suivantes sont également disponibles :

- Generation - contient des commandes, options et tâches de génération permettant de définir et d'activer un processus de génération (voir *Catégorie Generation* à la page 132).
- Transformation Profile - un profil de transformation est un groupe de transformations utilisé lors de la génération de modèle, lorsque vous devez appliquer des changements aux objets dans les modèles source ou cible. Pour plus d'informations sur la création de transformations et de profils de transformation, voir *Transformations et profils de transformation (Profile)* à la page 121. Pour plus d'informations sur l'appel de transformations, voir *Guide des fonctionnalités générales > Liaison et synchronisation de modèles > Génération de modèles et d'objets de modèle > Génération de modèles et d'objets de modèle > Fenêtre d'options de génération > Application de transformations*

Exemple : Ajout d'un nouvel attribut à partir d'une feuille de propriétés

Dans cet exemple, nous allons rapidement ajouter un attribut directement dans la feuille de propriétés d'un objet. PowerAMC va gérer la création du fichier d'extension ainsi que la création de toutes les extensions nécessaires.

1. Cliquez sur le bouton **Menu de la feuille de propriétés** dans l'angle inférieur gauche de la feuille de propriétés, juste à droite du bouton **Plus/Moins**, puis sélectionnez **Nouvel attribut**.
2. Dans la boîte de dialogue Nouvel attribut, saisissez `Latence` dans la zone **Nom**, sélectionnez `Chaîne` pour le type de données.
3. Cliquez sur le bouton Points de suspension à droite de la zone **Liste des valeurs**, saisissez la liste de valeurs prédéfinies suivantes, puis cliquez sur **OK** :
   - Par lots
   - Temps réel
   - Programmée
4. [facultatif] Sélectionnez `Programmée` dans la zone **Valeur par défaut**.
5. [facultatif] Cliquez sur **Suivant** pour spécifier la page de feuille de propriétés sur laquelle vous souhaitez que le nouvelle attribut s'affiche. Ici, nous allons laisser la valeur par défaut, de sorte qu'il sera inséré sur l'onglet **Général**.

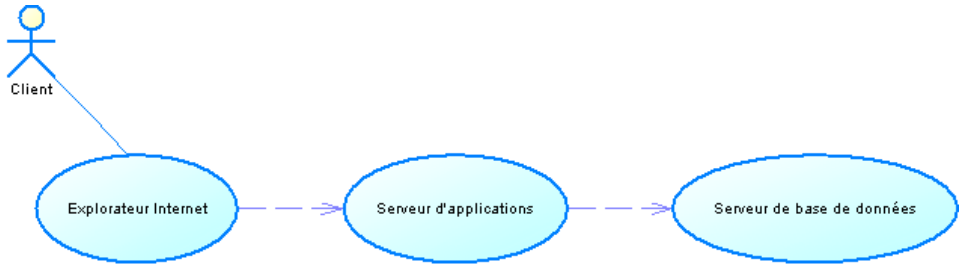


Exemple : Création d'extensions de diagramme de robustesse

Dans cet exemple, nous allons recréer le fichier d'extension Robustness Analysis fourni avec PowerAMC afin d'étendre le diagramme de communication du MOO. Les diagrammes de robustesse se trouvent entre les diagrammes de cas d'utilisation et le diagramme de séquence, et permettent de combler le vide existant entre ce que le système doit faire et comment il va s'y prendre pour accomplir sa tâche.

Pour pouvoir prendre en charge le diagramme de robustesse, nous allons devoir définir de nouveaux objets en appliquant des stéréotype à une métaclasse, spécifier des outils personnalisés et des symboles pour ces nouveaux objets, mais aussi définir des vérifications personnalisées pour les liens entre objets et produire un fichier qui va contenir une description des messages échangés entre objets.

La création des extensions Robustness Analysis va nous permettre de vérifier des cas d'utilisation tels que le cas suivant, qui représente une transaction Web de base :

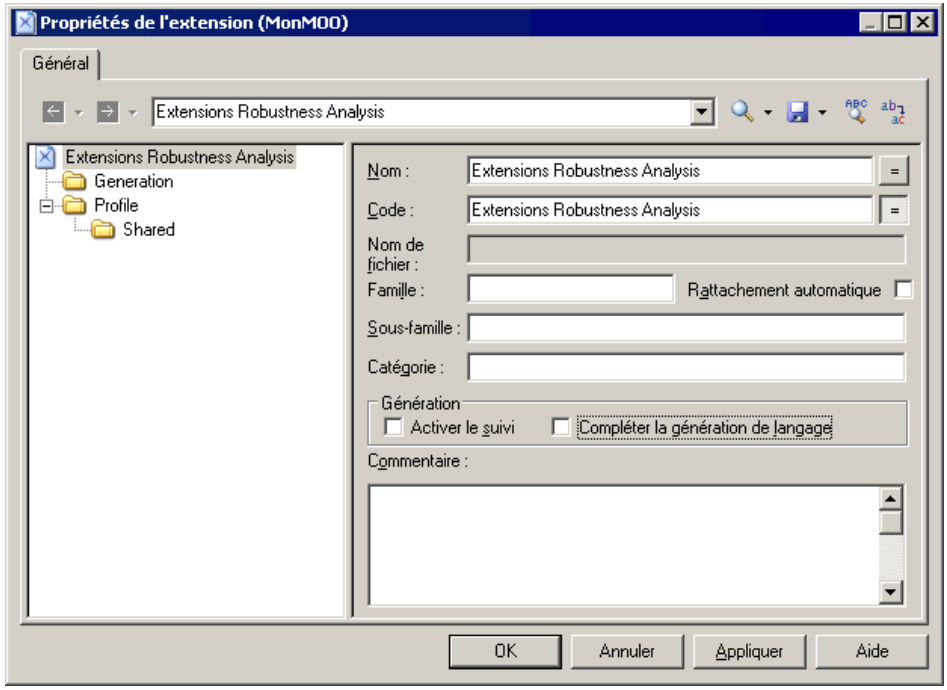


Un client souhaite connaître la valeur de ses actions afin de décider s'il va ou non les vendre. Il envoie une requête sur explorateur Internet pour obtenir la valeur de l'action, la requête est transférée depuis l'explorateur vers le serveur de bases de données via le serveur d'applications.

Création d'un nouveau fichier d'extension dans votre modèle

La première étape de la définition d'extensions consiste à créer un fichier d'extension (.xem) pour les stocker. Pour commencer cette procédure, vous devez ouvrir ou créer un MOO.

1. Sélectionnez **Modèle** > **Extensions** pour afficher la liste des extensions attachées au modèle.
2. Cliquez sur l'outil **Ajouter une ligne** afin de créer un nouveau fichier d'extension, puis sur l'outil **Propriétés** pour l'afficher dans l'Editeur de ressources.
3. Saisissez `Extensions Robustness Analysis` dans la zone **Nom**, puis décochez la case **Compléter la génération de langage**, car ces extensions n'appartiennent pas à une famille de langage objet et ne seront pas utilisées pour compléter une génération de langage objet.
4. Développez la catégorie Profile, dans laquelle nous allons créer les extensions :



Création de nouveaux objets à l'aide de stéréotypes

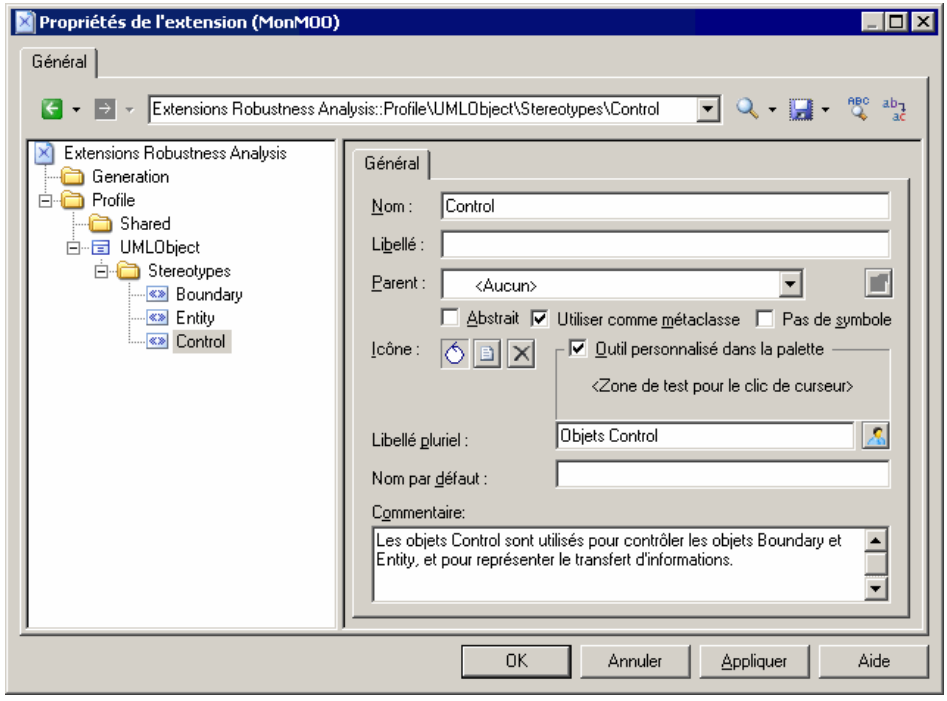
Pour mettre en oeuvre l'analyse de robustesse dans PowerAMC, nous devons créer trois nouveaux types d'objet (Bondary, Entité et Control), que nous allons définir dans la catégorie Profile en étendant la métaclasse `UMLObject` à l'aide de stéréotypes.

1. Pointez sur la catégorie Profile, cliquez le bouton droit de la souris, puis sélectionnez **Ajouter des métaclasses** pour afficher la boîte de dialogue Sélection de métaclasses.
2. Sélectionnez `UMLObject` sur l'onglet PdOOM, puis cliquez sur **OK** pour ajouter cette métaclasse au fichier d'extension.

   Remarque : Cliquez sur l'outil **Rechercher dans l'aide sur les objets du métamodèle** à droite de la zone **Nom** (ou cliquez sur **Ctrl+F1**) pour afficher des informations relatives à cette métaclasse et voir où elle est située dans le métamodèle PowerAMC.
3. Pointez sur la catégorie `UMLObject`, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau** > **Stéréotype** afin de créer une stéréotype pour étendre cette métaclasse.
4. Saisissez `Boundary` dans la zone **Nom**, puis `Les objets Boundary sont utilisés par les objets Actors lorsqu'ils communiquent avec le système. Il peut s'agir de fenêtres, d'écrans, de boîtes de dialogue ou de menus.` dans la zone **Commentaire**.

5. Cochez la case **Utiliser comme métaclasse** afin de promouvoir le type d'objet dans l'interface de sorte qu'il ait sa propre liste d'objets et sa propre catégorie dans l'Explorateur d'objets.
6. Cliquez sur l'outil **Sélectionner une icône** afin d'afficher la boîte de dialogue de bibliothèque d'images PowerAMC, cliquez sur l'onglet **Recherche d'images**, saisissez `boundary` dans la zone **Rechercher**, puis cliquez sur le bouton **Rechercher**.
7. Sélectionnez l'image `Boundary.cur` dans les résultats, puis cliquez sur OK pour l'affecter afin de représenter les objets Boundary dans l'Explorateur et les autres éléments d'interface. Cochez la case **Outil personnalisé dans la Boite à outils** afin de créer un outil avec cette même icône dans la Boîte à outils pour créer ce nouvel objet.
8. Répétez ces étapes pour créer les stéréotypes et icônes suivants :

Stéréotype	Commentaire	Fichier d'image
Entity	Les objets Entité représentent des données stockées telles qu'une base de données, des tables de base de données ou tout type d'objet temporaire tel que des résultats de recherche.	entity.cur
Control	Les objets Control sont utilisés pour contrôler les objets Boundary et Entity, et représenter le transfert d'informations.	control.cur

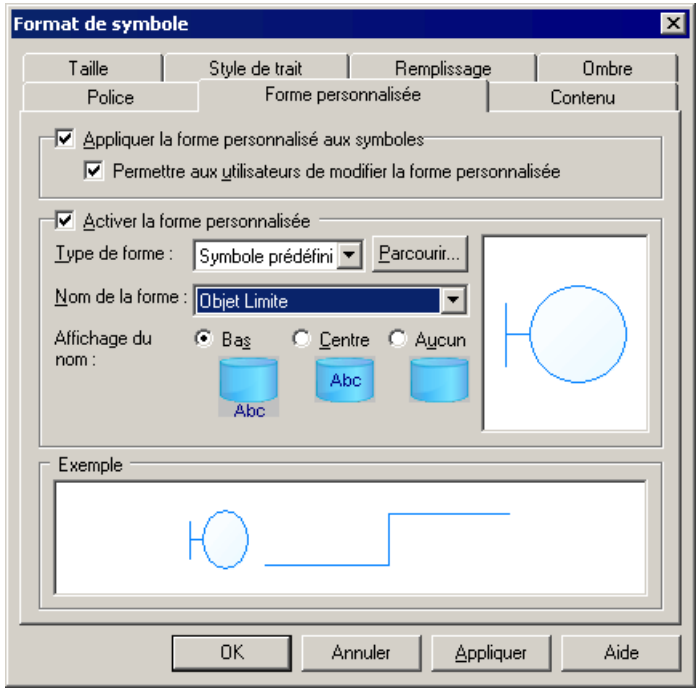


9. Cliquez sur **Appliquer** afin d'enregistrer vos modifications avant de poursuivre.

Spécification de symboles personnalisés pour les objets Robustness Analysis

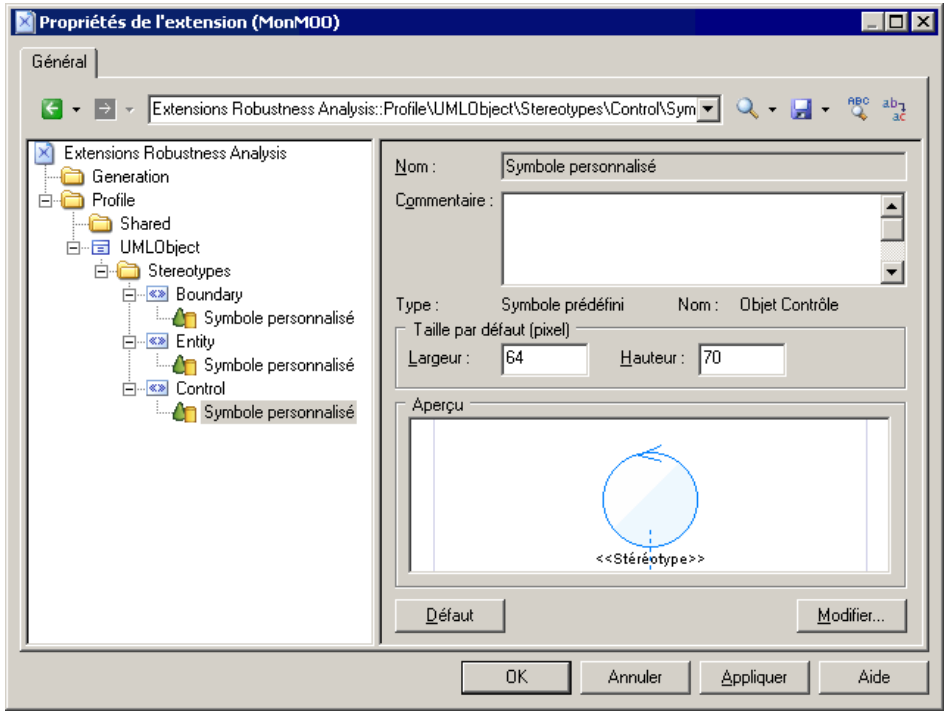
Nous allons spécifier des symboles de diagramme pour chacun de nos nouveaux objets Robustness Analysis en ajoutant des symboles personnalisés à nos nouveaux stéréotypes.

1. Pointez sur le stéréotype `Boundary`, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau** > **Symbole personnalisé** afin de créer un symbole personnalisé sous le stéréotype.
2. Cliquez sur le bouton Modifier afin d'afficher la boîte de dialogue Format de symbole, puis cliquez sur l'onglet **Forme personnalisée**.
3. Cochez la case **Activer la forme personnalisée**, puis sélectionnez Objet Limite dans la liste **Forme personnalisée**.



4. Cliquez sur **OK** pour terminer la définition du symbole personnalisé et revenir dans l'Editeur de ressources.
5. Répétez ces étapes pour les autres stéréotypes :

Stéréotype	Nom de la forme
Entity	Objet Entité
Control	Objet Contrôle



6. Cliquez sur **Appliquer** pour enregistrer vos modifications avant de poursuivre.

Création de vérifications personnalisées sur les liens entre objets

Nous allons maintenant créer trois vérifications personnalisées sur les liens entre objets qui vont connecter les différents objets Robustness Analysis. Ces vérifications, qui sont écrites en VB, n'empêchent pas les utilisateurs de créer des diagrammes non-pris en charge par la méthodologie Robustness Analysis, mais définissent des règles dont l'application sera contrôlée à l'aide de la fonctionnalité de vérification de modèles.

Pour plus d'informations sur la syntaxe VBS, voir *Chapitre 7, Pilotage de PowerAMC à l'aide de scripts* à la page 349.

1. Pointez sur la catégorie Profile, cliquez le bouton droit de la souris, puis sélectionnez **Ajouter des métaclasses** pour afficher la boîte de dialogue Sélection de métaclasses, sélectionnez `InstanceLink` sur l'onglet PdOOM et cliquez sur **OK** pour l'ajouter dans le fichier d'extension.
2. Pointez sur la catégorie `InstanceLink`, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau > Vérification personnalisée** pour créer une vérification sous la métaclasse.
3. Saisissez les valeurs suivantes pour les propriétés sur l'onglet **Général** :

Champ	Valeur
Nom	Collaboration d'acteur incorrecte
Commentaire	Cette vérification contrôle si des acteurs sont liés aux objets Boundary. Robustness Analysis ne permet pas de lier des acteurs aux objets Control ou Entity.
Message d'aide	Cette vérification s'assure que les acteurs ne communiquent qu'avec les objets Boundary.
Message de résultats	Les liens entre objets suivants sont incorrects :
Sévérité par défaut	Erreur
Exécuter la vérification par défaut	[sélectionnée]

4. Cliquez sur l'onglet **Script de vérification** et saisissez le script suivant dans la zone de texte :

```
Function %Check%(link)
   ' La valeur par défaut est True
   %Check% = True

   ' La valeur par défaut est True
   If link is Nothing then
      Exit Function
   End if
   If not link.IsKindOf(PdOOM.cls_InstanceLink) then
      Exit Function
   End If

   ' Extrait les extrémités du lien
   Dim src, dst
   Set src = link.ObjectA
   Set dst = link.ObjectB

   ' La source est un acteur
   ' Appelle la fonction globale CompareObjectKind() définie dans
le volet Script global
   If CompareObjectKind(src, PdOOM.Cls_Actor) Then
      ' Vérifie si la destination est un objet UML avec le
stéréotype "Boundary"
      If not CompareStereotype(dst, PdOOM.Cls_UMLObject,
"Boundary") Then
         %Check% = False
      End If
   ElseIf CompareObjectKind(dst, PdOOM.Cls_Actor) Then
      ' Vérifie si la source est un objet UML avec le stéréotype
"Boundary
      If not CompareStereotype(src, PdOOM.Cls_UMLObject,
```

```
"Boundary") Then
         %Check% = False
      End If
   End If
End Function
```

5. Cliquez sur l'onglet **Script global** (dans lequel vous stockez les fonctions et les attributs statiques qui peuvent être réutilisés dans les différentes fonctions) et saisissez le script suivant dans la zone de texte :

```
' Cette fonction globale vérifie si un objet a un type particulier
' ou s'il est un raccourci d'un type particulier
Function CompareObjectKind(Obj, Kind)
   ' La valeur par défaut est false
   CompareObjectKind = False

   ' Vérifie l'objet
   If Obj is Nothing Then
      Exit Function
   End If
   ' Cas particulier du raccourci, recherche de son objet cible
   If Obj.IsShortcut() Then
      CompareObjectKind = CompareObjectKind(Obj.TargetObject,
Kind)
      Exit Function
   End If
   If Obj.IsKindOf(Kind) Then
      ' Cas particulier du raccourci, recherche de son objet cible
      CompareObjectKind = True
   End If
End Function

' Cette fonction globale vérifie si un objet a un type particulier
' et compare sa valeur de stéréotype
Function CompareStereotype(Obj, Kind, Value)
   ' La valeur par défaut est false
   CompareStereotype = False

   ' La valeur par défaut est false
   If Obj is Nothing then
      Exit Function
   End If
   if (not Obj.IsShortcut() and not
Obj.HasAttribute("Stereotype")) Then
      Exit Function
   End If
   ' Cas particulier du raccourci, recherche de son objet cible
   If Obj.IsShortcut() Then
      CompareStereotype = CompareStereotype(Obj.TargetObject,
Kind, Value)
      Exit Function
   End If
   If Obj.IsKindOf(Kind) Then
      ' Cas particulier du raccourci, recherche de son objet cible
      If Obj.Stereotype = Value Then
         ' Cas particulier du raccourci, recherche de son objet
```

```
cible
         CompareStereotype = True
      End If
   End If
End Function

' Cette fonction globale copie l'attribut standard
' de la source vers la cible
Function Copy (src, trgt)
   trgt.name = src.name
   trgt.code  = src.code
   trgt.comment = src.comment
   trgt.description = src.description
   trgt.annotation = src.annotation
   Dim b, d
   for each b in src.AttachedRules
      trgt.AttachedRules.insert -1,b
   next
   for each d in src.RelatedDiagrams
      trgt.RelatedDiagrams.insert -1,d
   next
   output " "
   output trgt.Classname & " " & trgt.name & " a été créé."
   output " "
End Function
```

6. Répétez ces étapes pour créer une seconde vérification en saisissant les valeurs suivantes :

Champ	Valeur
Nom	`Lien entre objets Boundary incorrect`
Message d'aide	`Cette vérification s'assure qu'un lien entre objets n'est pas défini entre deux objets Boundary.`
Message de résultats	`Les liens entre objets Boundary suivants sont incorrects :`
Sévérité par défaut	`Erreur`
Exécuter la vérification par défaut	[sélectionnée]

Champ	Valeur

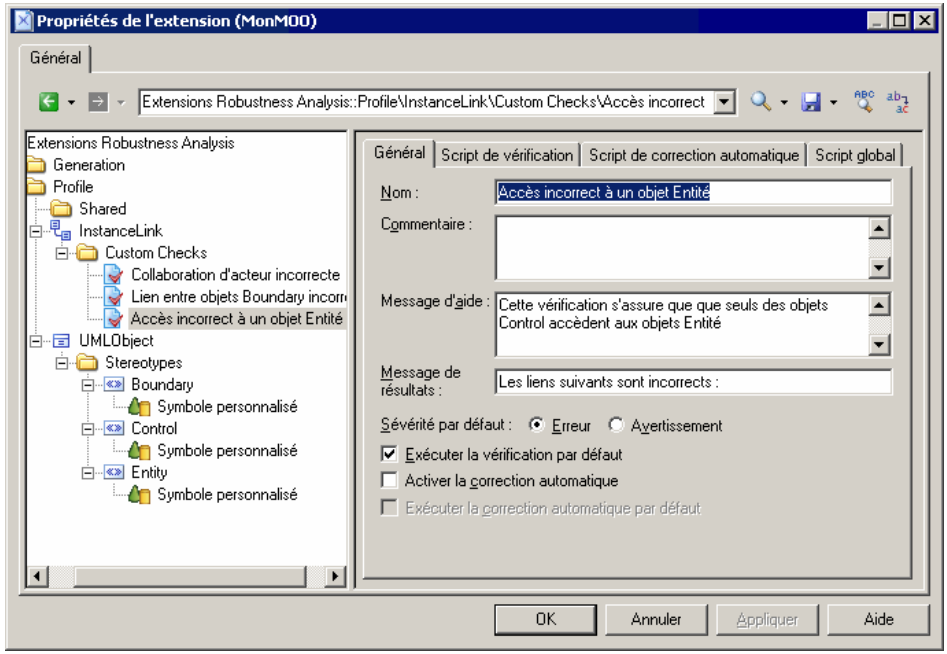
Script de vérification	`Function %Check%(link)`
	`  ' La valeur par défaut est True`
	`  %Check% = True`
	
	`  ' L'objet doit être un lien entre objets`
	`  If link is Nothing then`
	`  Exit Function`
	`  End if`
	`  If not link.IsKindOf(PdOOM.cls_InstanceLink) then`
	`  Exit Function`
	`  End If`
	
	`  ' Extrait les extrémités du lien`
	`  Dim src, dst`
	`  Set src = link.ObjectA`
	`  Set dst = link.ObjectB`
	
	`  ' Erreur si les deux objets sont de type 'Boundary'`
	`  If CompareStereotype(src, PdOOM.Cls_UMLObject, "Boundary") Then`
	`  If CompareStereotype(dst, PdOOM.Cls_UMLObject, "Boundary") Then`
	`  %Check% = False`
	`  End If`
	`  End If`
	`End Function`

7. Répétez ces étapes pour créer une troisième vérification en saisissant les valeurs suivantes :

Champ	Value
Nom	`Accès incorrect à un objet Entité`
Message d'aide	`Cette vérification s'assure que que seuls des objets Control accèdent aux objets Entité.`
Message de résultats	`Les liens suivants sont incorrects :`
Sévérité par défaut	`Erreur`
Exécuter la vérification par défaut	[sélectionnée]

Champ	Value

Script de vérification	`Function %Check%(link)`
	`  ' La valeur par défaut est True`
	`  %Check% = True`
	
	`  ' L'objet doit être un lien entre objets`
	`  If link is Nothing then`
	`  Exit Function`
	`  End if`
	`  If not link.IsKindOf(PdOOM.cls_InstanceLink) then`
	`  Exit Function`
	`  End If`
	
	`  ' Extrait les extrémités du lien`
	`  Dim src, dst`
	`  Set src = link.ObjectA`
	`  Set dst = link.ObjectB`
	
	`  ' La source est un objet UML avec un stéréotype "Entity" ?`
	`  ' Appelle la fonction globale CompareStereotype() définie dans le volet Script global`
	`  If CompareStereotype(src, PdOOM.Cls_UMLObject, "Entity") Then`
	`  ' Vérifie si la destination est un objet UML avec un stéréotype "Control"`
	`  If not CompareStereotype(dst, PdOOM.Cls_UMLObject, "Control") Then`
	`  %Check% = False`
	`  End If`
	`  ElseIf CompareStereotype(dst, PdOOM.Cls_UMLObject, "Entity") Then`
	`  ' Vérifie si la source est un objet UML avec un stéréotype "Control"`
	`  If not CompareStereotype(src, PdOOM.Cls_UMLObject, "Control") Then`
	`  %Check% = False`
	`  End If`
	`  End If`
	`End Function`



8. Cliquez sur **Appliquer** pour sauvegarder vos modifications avant de poursuivre.

Définition de templates pour extraire les descriptions de message

Nous allons générer des descriptions sous forme de texte pour les messages dans le diagramme, en fournissant pour chaque message le nom de l'émetteur, du message et du destinataire. Pour ce faire, nous allons devoir définir un template en utilisant le GTL (*Generation Template Language*, langage de génération par template) de PowerAMC afin d'extraire les informations et un fichier généré pour contenir et afficher les informations extraites.

Pour pouvoir générer ce texte de description, nous allons devoir extraire des informations des métaclasses suivantes :

- *Message* - pour extraire le numéro d'ordre du message, son nom, l'émetteur et le récepteur
- *CommunicationDiagram* - pour rassembler tous les messages de chaque diagramme et les trier

1. Pointez sur la catégorie Profile, cliquez le bouton droit de la souris, puis sélectionnez **Ajouter des métaclasses** pour afficher la boîte de dialogue Sélection de métaclasses, sélectionnez `CommunicationDiagram` et `Message` sur l'onglet PdOOM et cliquez sur **OK** pour les ajouter au fichier d'extension.
2. Pointez sur la catégorie `Message`, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau** > **Template** pour créer un template sous la métaclasse.

3. Saisissez `description` dans la zone **Nom**, puis saisissez le code de langage de génération par template suivant dans la zone de texte :

```
.set_value(_tabs, "", new)
.foreach_part(%SequenceNumber%, '.')
   .set_value(_tabs, "   %_tabs%")
.next
%_tabs%%SequenceNumber%) %Sender.ShortDescription% envoie le
message "%Name%" à %Receiver.ShortDescription%
```

   La première ligne du template initialise la variable `_tabs`, et la macro `foreach_part` calcule le montant d'indentation approprié en bouclant sur chaque numéro d'ordre, et en ajoutant 3 espaces chaque fois qu'il trouve un point. La dernière ligne utilise cette variable afin de réaliser l'indentation, de mettre en forme et d'afficher des informations extraites pour chaque message.

   Pour obtenir des informations détaillées sur le GTL (*Generation Template Language*, langage de génération par template) de PowerAMC, voir *Chapitre 5, Personnalisation de la génération à l'aide du langage de génération par template* à la page 285.

4. Pointez sur la catégorie `CommunicationDiagram`, cliquez le bouton droit de la souris puis sélectionnez **Nouveau** > **Template** pour créer un template sous la métaclasse.
5. Saisissez `compareCbMsgSymbols` dans la zone **Nom**, puis saisissez le code de langage de génération par template suivant dans la zone de texte :

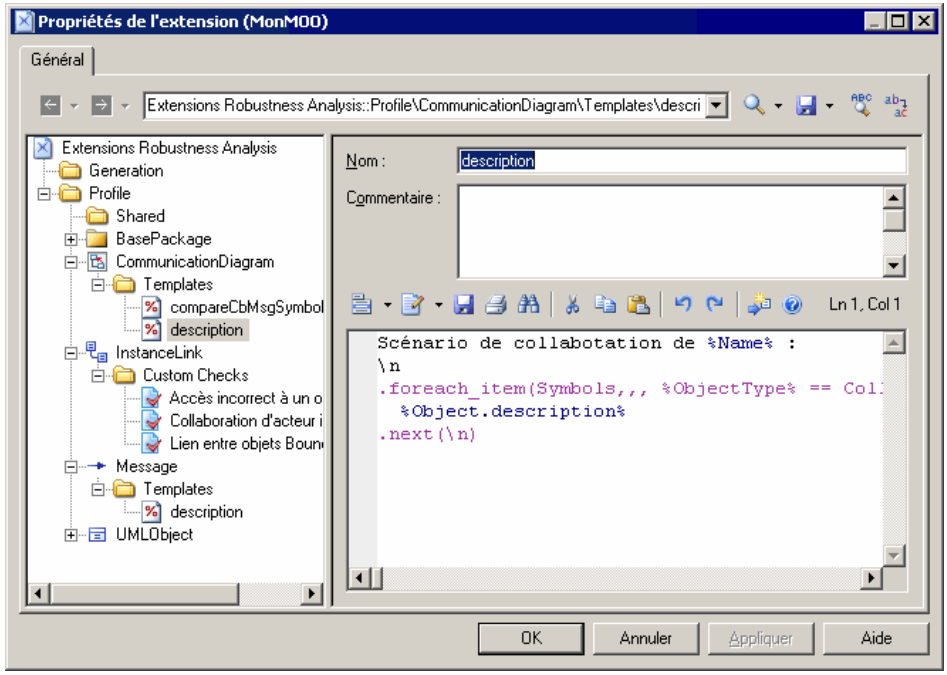
```
.bool (%Item1.Object.SequenceNumber% >=
%Item2.Object.SequenceNumber%)
```

   Ce template renvoie une valeur booléenne pour déterminer si un numéro de message est supérieur à un autre, et le résultat est utilisé dans un second template.

6. Pointez sur la catégorie `CommunicationDiagram`, cliquez le bouton droit de la souris puis sélectionnez **Nouveau** > **Template** pour créer un second template, et saisissez `description` dans la zone **Nom**, puis saisissez le code de langage de génération par template suivant dans la zone de texte :

```
Scénario de collabotation de %Name% :
\n
.foreach_item(Symbols,,, %ObjectType% ==
CollaborationMessageSymbol, %compareCbMsgSymbols%)
  %Object.description%
.next(\n)
```

   La première ligne de ce template est utilisée pour générer le titre du scénario à l'aide du nom du diagramme de communication. Puis la macro `.foreach_item` boucle sur chaque symbole de message, et appelle les autres templates afin de mettre en forme et générer les informations relatives au message.



7. Cliquez sur **Appliquer** pour enregistrer vos modifications avant de continuer.

Création d'un fichier généré pour les informations relatives aux messages

Après avoir créé des templates afin d'extraire des informations relatives aux messages dans le modèle, nous devons maintenant créer un fichier généré afin de les contenir et de les afficher dans l'onglet **Aperçu** de la feuille de propriétés de diagramme. Nous allons définir le fichier sur la métaclasse `BasePackage`, qui est une classe commune pour tous les packages et modèles, et nous allons le faire boucler sur les diagrammes de communication du modèle afin d'évaluer le template `description` défini sur la métaclasse `CommunicationDiagram`.

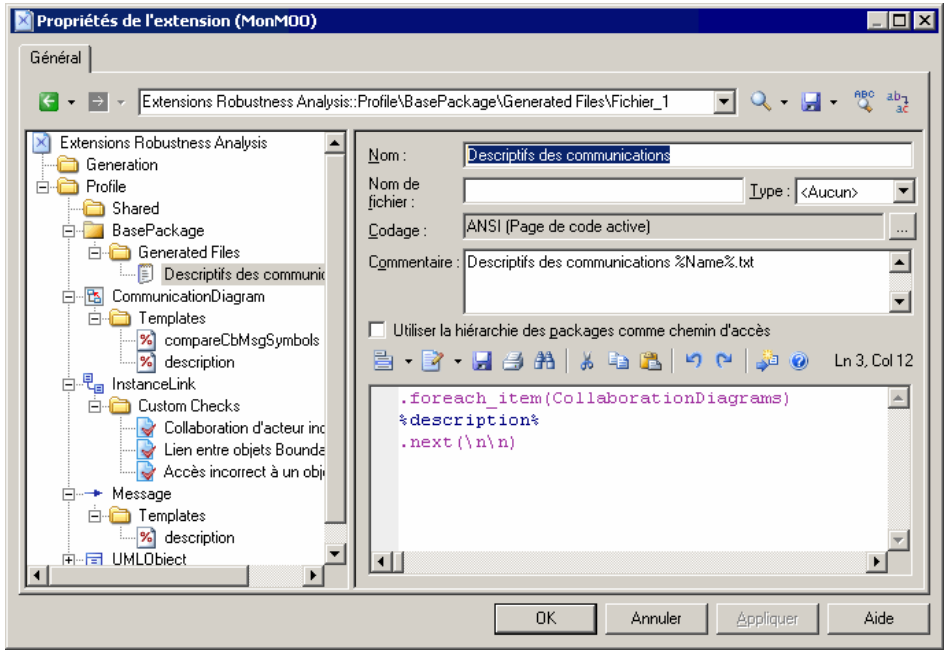
1. Pointez sur la catégorie Profile, cliquez le bouton droit de la souris, puis sélectionnez **Ajouter des métaclasses** pour afficher la boîte de dialogue Sélection de métaclasses, cliquez sur l'outil **Modifier le filtre des métaclasses**, sélectionnez `Afficher les métaclasses de modélisation abstraite`, puis cliquez sur l'onglet PdCommon.
2. Sélectionnez `BasePackage`, puis cliquez sur **OK** afin de l'ajouter dans le fichier d'extension.
3. Pointez sur la catégorie BasePackage, cliquez le bouton droit de la souris et sélectionnez **Nouveau** > **Fichier généré** pour créer un fichier sous la métaclasse.

4. Saisissez les valeurs suivantes dans les propriétés du fichier :

Zone	Value
Nom	`Descriptifs des communications`
Nom de fichier	`Descriptifs des communications %Name%.txt`
Codage	`ANSI`
Utiliser la hiérarchie des packages comme chemin d'a	[désélectionnée]

5. Saisissez le code suivant dans la zone de texte :

```
.foreach_item(CollaborationDiagrams)
 %description%
.next(\n\n)
```



6. Cliquez sur **Appliquer** pour enregistrer vos modifications, puis sur **OK** pour fermer l'éditeur de ressources.
7. Cliquez sur **OK** pour fermer la Liste des extensions.

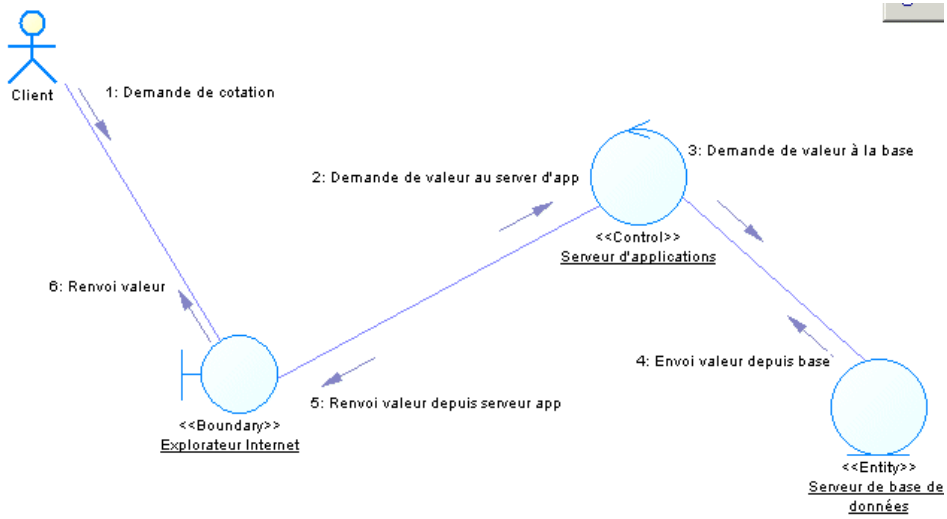
Test des extensions Robustness Analysis

Pour tester les extensions que nous avons créées, nous allons créer un petit diagramme de robustesse afin d'analyser notre cas d'utilisation.

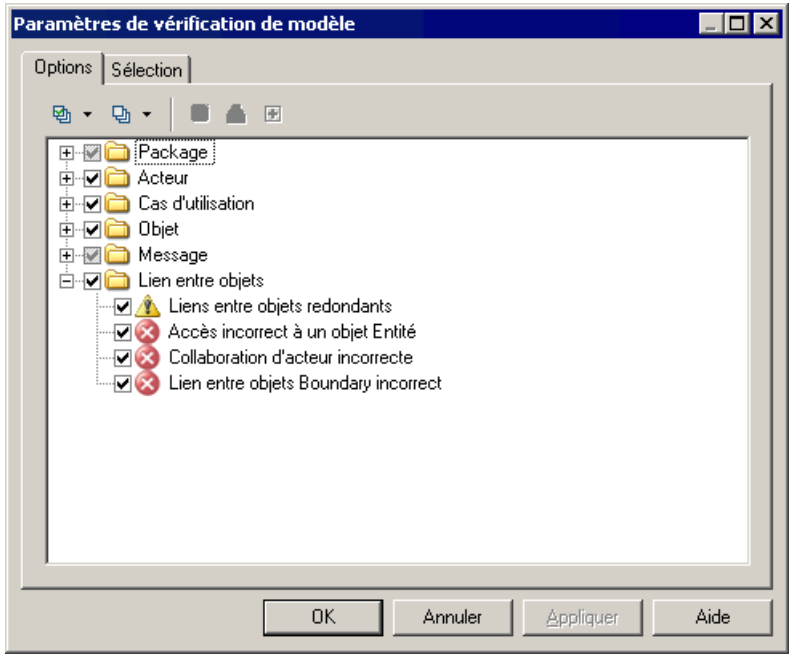
1. Pointez sur le noeud du diagramme dans l'Explorateur d'objets, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau** > **Diagramme de communication**.

   En plus de la Boîte à outils standard, une boîte à outils personnalisée est fournie avec les outils que vous avez définis pour créer des objets Boundary, Control et Entity.
2. Faites glisser l'acteur Client depuis la catégorie Acteurs de l'Explorateur d'objets dans le diagramme afin d'y créer un raccourci. Puis créez successivement un objet Boundary, Control et Entity avant de les nommer respectivement `Explorateur Internet`, `Serveur d'applications` et `Serveur de base de données`.
3. Utilisez l'outil **Lien entre objets** dans la Boîte à outils standard pour connecter Client, `Explorateur Internet`, `Serveur d'applications`, puis `Serveur de bases de données`.
4. Créez les messages suivants sur les onglets **Messages** des feuilles de propriétés de lien entre objets :

Direction	Nom du message	Numéro d'ordre
Client - Explorateur Internet	Demande de cotation	1
Explorateur Internet - Application Server	Demande de valeur au server d'app	2
Serveur d'applications - Serveur de base de données	Demande de valeur à la base	3
Serveur de base de données - Serveur d'applications	Envoi valeur depuis base	4
Serveur d'applications - Explorateur Internet	Renvoi valeur depuis serveur app	5
Explorateur Internet - Client	Renvoi valeur	6



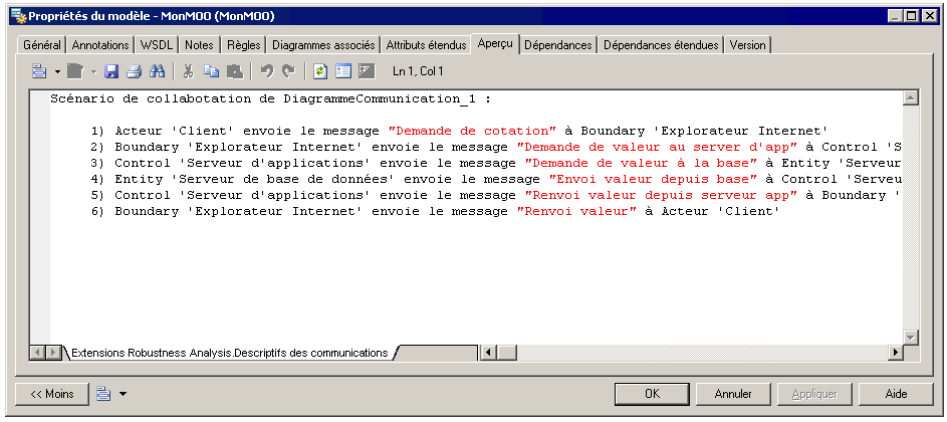
5. Sélectionnez **Outils > Vérifiez le modèle** pour afficher la boîte de dialogue Paramètres de vérification de modèle, dans laquelle les vérifications personnalisées que vous avez créées sont affichées dans la catégorie Lien entre objets :



Cliquez sur **OK** afin de tester la validité de liens entre objets que vous avez créés.

6. Pointez sur le noeud du modèle dans l'Explorateur d'objets, cliquez le bouton droit de la souris, puis sélectionnez **Propriétés** afin d'afficher la feuille de propriétés du modèle.

Cliquez sur l'onglet **Aperçu** afin de passer en revue les messages envoyés pour notre cas d'utilisation :



Extension de la génération et création de nouvelles cibles de génération

Les extensions peuvent être utilisées pour étendre la génération et créer de nouvelles cibles de génération.

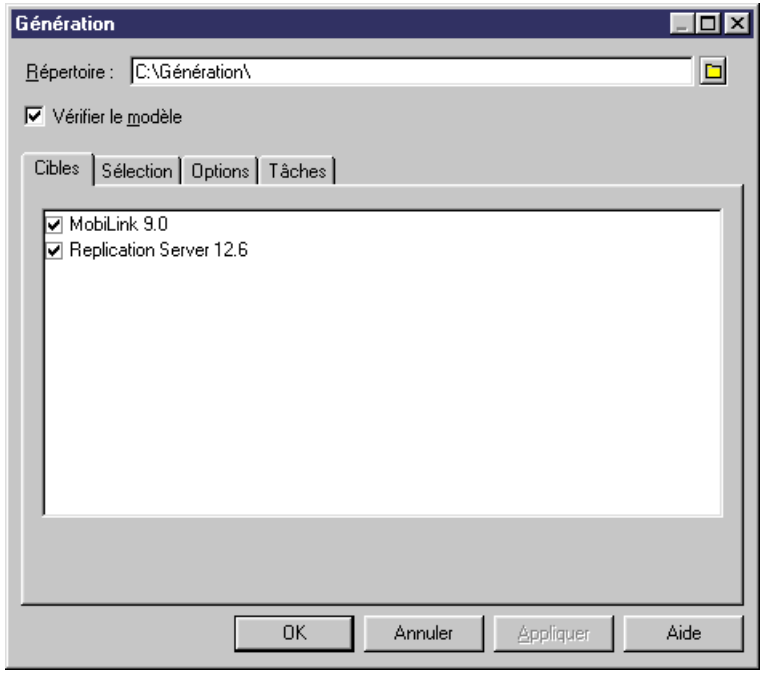
Le tableau suivant montre comment vous pouvez personnaliser la génération standard de MPM, MOO ou MSX à partir de l'Editeur de ressources :

Boîte de dialogue de génération	Extension
Page Cibles	La page Cibles est affichée si la propriété **Compléter la génération de langage** est sélectionnée (voir *Propriétés d'une extension* à la page 27) et si l'extension contient au moins une tâche ou un fichier généré.
Page Options	Définissez les options dans `Generation\Options` .
Page Tâches	Définissez les commandes dans `Generation\Commands` et référencez ces commande dans les tâches.

Si vous souhaitez créer des cibles de génération distinctes, (disponibles via la commande **Outils > Génération étendue**), vous devez remplir les conditions suivantes :

- La case **Compléter la génération de langage** ne doit pas être cochée dans la feuille de propriétés d'extension
- L'extension doit contenir des fichiers générés et des templates. Lors de la génération, l'évaluation d'un template génère du texte qui est écrit dans un fichier.

Ce type de génération est appelé *génération étendue*. Si vous avez plusieurs extensions conçues pour la génération étendue, celles-ci seront affichées dans la page Cibles de la boîte de dialogue de génération.



Vous pouvez créer des commandes dans le menu **Outils** afin d'accéder directement à la génération étendue pour une cible sélectionnée. Pour ce faire, vous devez procéder comme suit :

- Créer un menu (voir *Menus (Profile)* à la page 110) sous la métaclasse Model dans la catégorie Profile de l'extension, puis sélectionnez le `menu Outils` dans la liste **Emplacement**
- Créer une méthode (voir *Méthodes (Profile)* à la page 108) pour appeler la génération étendue comme suit :

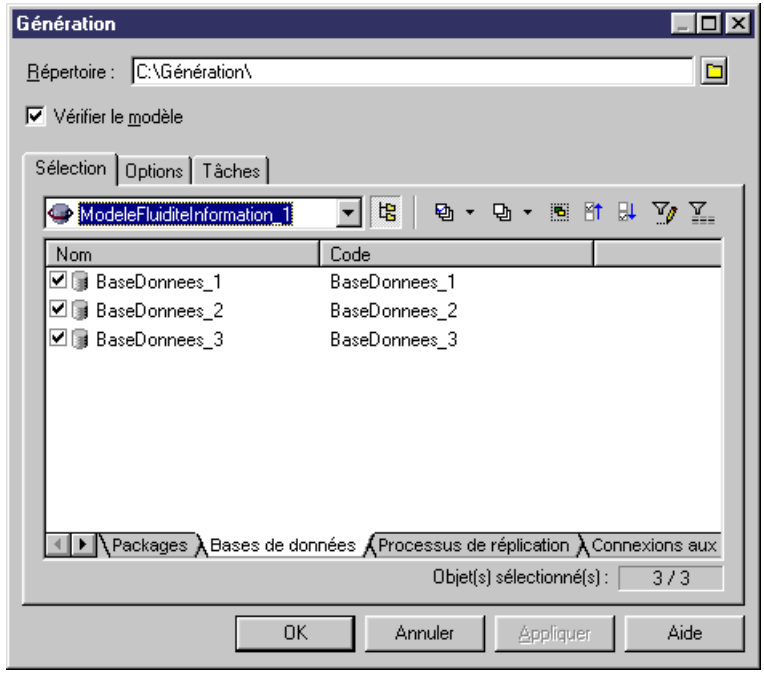
```
Sub %Method%(obj)

 Dim selection ' as ObjectSelection

 ' Crée une nouvelle sélection
 set selection = obj.CreateSelection

 ' Ajoute un objet de la sélection active dans la sélection créée
 selection.AddActiveSelectionObjects

 ' Génère des scripts pour la cible spécifique
 InteractiveMode = im_Dialog
 obj.GenerateFiles "", selection, "cible particulière"
```

```
End Sub
```

Où `cible particulière` représente le code de la cible de génération étendue.

- Ajoutez la méthode pour la génération étendue au menu afin de créer une commande particulière
- Enregistrez l'extension

La nouvelle commande s'affiche sous le menu **Outils**.



L'onglet **Cibles** ne s'affiche pas car la méthode sous-jacente spécifie une cible de génération.

Métaclasses (Profile)

Les métaclasses sont des classes tirées du métamodèle PowerAMC, et qui s'affichent à la racine de la catégorie Profile. Vous devez ajouter une métaclasse à un profil lorsque vous souhaitez étendre ce dernier.

Les métaclasses concrètes sont définies pour des types d'objet particuliers qui peuvent être créés dans un modèle, tandis que les métaclasses abstraites ne sont jamais instanciées, mais plutôt utilisées pour définir des extensions communes. Par exemple, BasePackage est ancêtre à la fois de model et de package

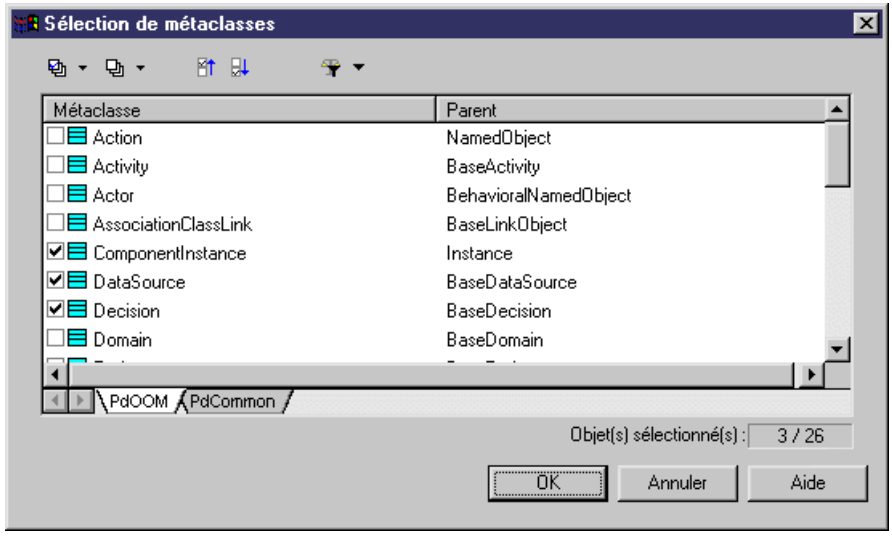
Pour plus d'informations sur la consultation et la navigation parmi les métaclasses dans le métamodèle, voir *Métamodèle public PowerAMC* à la page 9.

Si vous ne souhaitez pas étendre une métaclasse existante, mais préférez créer un type d'objet de modélisation entièrement nouveau, vous devez utiliser la métaclasse d'objet étendu (voir *Objets, sous-objets et liens étendus (Profile)* à la page 59).

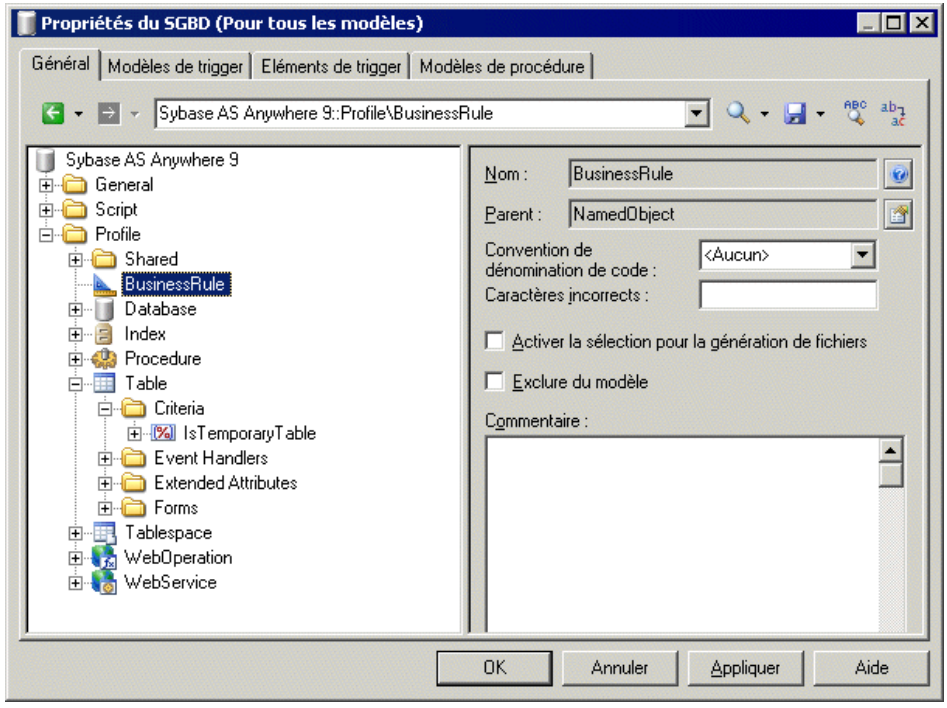
Ajout d'une métaclasse dans un profil

Vous ajoutez une métaclasse dans un profil afin de définir des extensions pour ce dernier.

1. Pointez sur la catégorie Profile, cliquez le bouton droit de la souris, puis sélectionnez Ajouter des métaclasses dans le menu contextuel afin d'afficher la boîte de dialogue Sélection de métaclasses.



2. Sélectionnez une ou plusieurs métaclasses à ajouter au profil. Vous pouvez utiliser les sous-onglets pour passer des métaclasses appartenant au module courant (par exemple, le MOO) aux métaclasses standard qui appartiennent au module PdCommon. Vous pouvez également utiliser l'outil Modifier le filtre des métaclasses afin d'afficher toutes les métaclasses, ou uniquement les métaclasses conceptuelles concrètes ou abstraites, dans la liste
3. Cliquez sur OK pour ajouter les métaclasses sélectionnées dans votre profil :



Propriétés d'une métaclasse

Vous spécifiez les propriétés pour une métaclasse en sélectionnant l'entrée correspondante dans l'Editeur de ressources.

Propriété	Description
Nom	[lecture seule] Spécifie le nom de la métaclasse. Cliquez sur le bouton à droite de cette zone pour afficher l'Aide sur les objets du métamodèle correspondant à la métaclasse.
Parent	[lecture seule] Spécifie le parent de la métaclasse. Cliquez sur le bouton à droite de cette zone pour afficher la feuille de propriétés de la métaclasse parent. Si la métaclasse parent n'est pas présente dans le profil, un message vous invite à l'ajouter automatiquement.

Propriété	Description
Conventions de dénomination de code	[métaclasses concrètes dans les fichiers de cible uniquement] Spécifie le format par défaut pour initialiser le script de conversion de nom en code pour les instances de la métaclasse. Les formats suivants sont disponibles : • `firstLowerWord` - Premier mot en minuscules, et la première lettre de chaque mot suivant en majuscule • `FirstUpperChar` - Premier caractère de chaque mot en majuscule • `lower_case` - Tous les mots en minuscules et séparés par un tiret bas • `UPPER_CASE` - Tous les mots en minuscules et séparés par un tiret bas Pour plus d'informations sur les scripts de conversion et les conventions de dénomination, voir *Guide des fonctionnalités générales > L'interface de PowerAMC > Objets > Propriétés d'un objet > Conventions de dénomination*.
Caractères illégaux	[métaclasses concrètes uniquement] Spécifie une liste de caractères illégaux qui ne peuvent pas être utilisés dans la génération de code pour la métaclasse. La liste des caractères doit être placée entre guillemets, par exemple : `"/!=<>""'()"` Lorsque vous travaillez sur un MOO, cette liste spécifique à l'objet prévaut sur les valeurs spécifiées dans le paramètre IllegalChar pour le langage objet (voir *Catégorie Settings : langage objet* à la page 131).
Activer la sélection pour la génération de fichiers	Spécifie que les instances de métaclasse correspondantes seront affichées dans l'onglet Sélection de la boîte de dialogue de génération étendue. Si une métaclasse parent est sélectionnée pour la génération de fichier, les métaclasses enfant sont également affichées dans l'onglet Sélection.
Exclure du modèle	[métaclasses concrètes uniquement] Empêche la création d'instances de la métaclasse dans le modèle et supprime toute référence à la métaclasse dans les menus, Boîte à outils, feuilles de propriétés etc, afin de simplifier l'interface. Par exemple, si vous n'utilisez pas les règles de gestion, vous pouvez cocher la case Exclure du modèle dans la feuille de propriétés de la métaclasse de règle de gestion afin de cacher les règles de gestion partout dans l'interface. Lorsque plusieurs fichiers de ressources sont attachés à un modèle, la métaclasse est exclue si l'un au moins des fichiers de ressources exclut cette métaclasse et qu'aucun autre fichier de ressources ne l'active de façon explicite. Si un modèle contient déjà des instances de cette métaclasse, les objets sont conservés mais il est impossible d'en créer de nouveaux.
Commentaire	Spécifie un commentaire descriptif pour la métaclasse.

Stéréotypes (Profile)

Les *stéréotypes* sont un mécanisme d'extension par instance. Lorsqu'un stéréotype est appliqué à une instance de métaclasse (en le sélectionnant dans la zone Stéréotype de la feuille

de propriétés de l'objet), les extensions que vous ajoutez au stéréotype sont ensuite appliquées à l'instance.

Les stéréotypes peuvent être promus au statut de *métaclasses* pour les rendre plus visibles dans l'interface, avec une liste, une catégorie dans l'Explorateurs d'objets et éventuellement un symbole personnalisé et un outil de boîte à outils. Pour plus d'informations, voir *Promotion d'un stéréotype au statut de métaclasse* à la page 56.

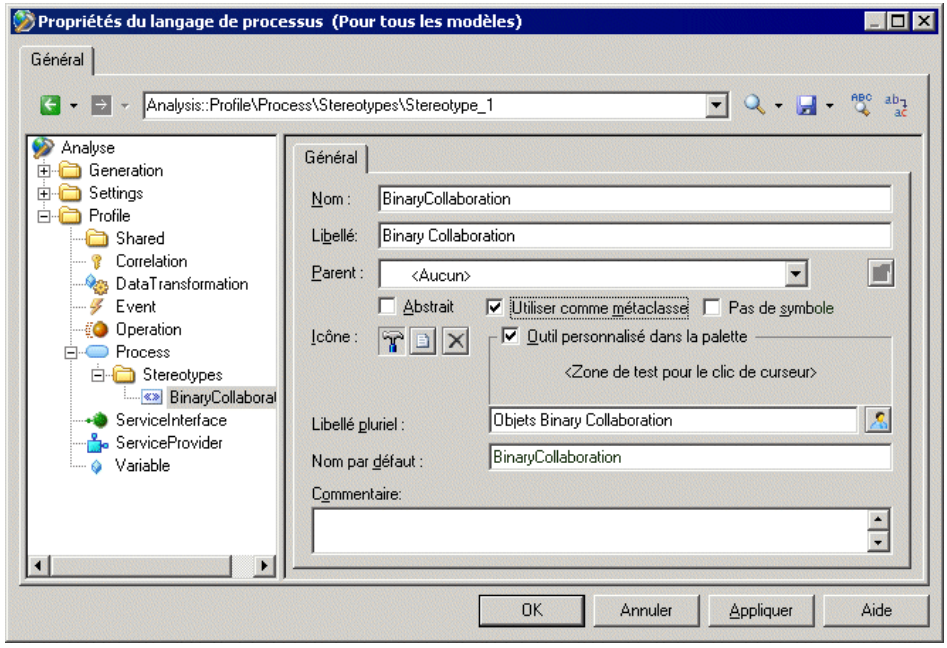
Vous pouvez définir plusieurs stéréotypes pour une métaclasse donnée, mais vous ne pouvez appliquer qu'un seul stéréotype à chaque instance. Les stéréotypes prennent en charge l'*héritage* : les extensions d'un stéréotype parent sont héritées par ses enfants.

Création d'un stéréotype

Vous pouvez créer un stéréotype dans une métaclasse, un critère ou un autre stéréotype.

1. Pointez sur une métaclasse, un critère ou un stéréotype, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau > Stéréotype**.

   Un nouveau stéréotype est créé avec un nom par défaut.
2. Saisissez un nom de stéréotype dans la zone Nom, et renseignez les propriétés appropriées.



Une fois que vous avez créé le stéréotype, vous pouvez définir des extensions telles qu'un outil personnalisé, ou bien des vérifications personnalisées pour le stéréotype. Ces extensions seront appliquées à toutes les instances de métaclasse ayant le stéréotype.

Propriétés d'un stéréotype

Vous spécifiez les propriétés pour une stéréotype en sélectionnant l'entrée correspondante dans l'Editeur de ressources.

Propriété	Description
Nom	Spécifie le nom interne du stéréotype, qui peut être utilisé pour le scripting.
Libellé	Spécifie le nom d'affichage du stéréotype, qui sera affiché dans l'interface PowerAMC.
Parent	Spécifie un stéréotype parent du stéréotype. Vous pouvez sélectionner un stéréotype défini dans la même métaclasse ou dans une métaclasse parent. Cliquez sur le bouton **Propriétés** pour aller au stéréotype parent dans l'arborescence et afficher ses propriétés.
Abstrait	Spécifie que le stéréotype ne peut pas être appliqué aux instances de métaclasse, ce stéréotype ne s'affiche pas dans la liste Stéréotype de la feuille de propriétés de l'objet, et ne peut être utilisé que comme parent d'autres stéréotypes enfant. Lorsque vous sélectionnez cette propriété, la case à cocher **Utiliser comme métaclasse** n'est pas disponible.
Utiliser comme métaclasse	Spécifie que le stéréotype est une sous-classification des instances de la métaclasse sélectionnée. Le stéréotype aura sa propre liste d'objets et sa catégorie dans l'Explorateur d'objets, et fera l'objet d'un onglet dans les boîtes de sélection multionglet telles que celles utilisées pour la génération. Pour plus d'informations, voir *Promotion d'un stéréotype au statut de métaclasse* à la page 56.
Pas de symbole	[disponible lorsque **Utiliser comme métaclasse** est cochée] Spécifie que les instances de la métaclasse stéréotypée sont créées, et qu'elles seront dépourvues de symbole de diagramme. Cela peut s'avérer utile lorsque vous souhaitez modéliser des sous-objets ou d'autres objets qui n'ont pas besoin d'être affichés dans le diagramme. L'option Outil personnalisé dans la Boîte à outils est désactivée quand cette option est sélectionnée.
Icône	Spécifie une icône pour des instances stéréotypées de la métaclasse. Cliquez sur les outils à droite de cette zone pour recherche un fichier .cur ou .ico.
Outil personnalisé dans la Boîte à outils	Associe un outil dans une boîte à outils au stéréotype courant. Cette option n'est disponible que pour les objets qui prennent en charge des symboles, et n'est pas disponible pour des objets comme les attributs, par exemple. Pour plus d'informations, voir *Spécification d'une icône et d'un outil personnalisé pour un stéréotype* à la page 57.
Libellé pluriel	[disponible lorsque **Utiliser comme métaclasse** est cochée] Spécifie la forme plurielle du nom d'affichage qui s'affichera dans l'interface PowerAMC.

Propriété	Description
Nom par défaut	[disponible lorsque **Utiliser comme métaclasse** ou **Outil personnalisé dans la Boîte à outils** est cochée] Spécifie un nom par défaut pour les objets créés. Un compteur est automatiquement ajouté au nom spécifié afin de générer des noms uniques. Le nom par défaut peut s'avérer très utile lorsque vous concevez pour un langage cible ou une application ayant des conventions de dénomination strictes. Toutefois, le nom par défaut ne prévaut pas sur les conventions de dénomination de modèle, de telle sorte que si un nom ne respecte pas ces dernières, il est automatiquement modifié.
Commentaire	Informations supplémentaires relatives au stéréotype.

Promotion d'un stéréotype au statut de métaclasse

Vous pouvez promouvoir un stéréotype au statut de métaclasse en cochant la case **Utiliser comme métaclasse** dans la page de propriétés du stéréotype. Chaque stéréotypes ainsi promu fait l'objet d'un dossier dans l'Explorateur d'objets, d'une entrée dans le menu Modèle et d'une commande dans le menu contextuel Nouveau.

Vous pouvez utiliser de tels stéréotypes afin de :

- Créer de nouveaux types d'objet qui ont en commun un grande partie du comportement du type d'objet existant, tels que les transactions métiers et les collaborations binaires dans un MPM pour ebXML.
- Pouvoir faire cohabiter des objets ayant le même nom, mais des stéréotypes différents, au sein d'un même espace de noms (une métaclasse stéréotype crée un sous-espace de noms dans la métaclasse courante).

Remarque : Les stéréotypes définis sur les sous-objets (par exemple, les colonnes de tables ou les attributs d'entité), peuvent être transformés en stéréotypes de métaclasse.

1. Dans la page de propriétés du stéréotype, cochez la case **Utiliser comme métaclasse**.

   Le stéréotype de métaclasse ainsi créé se comporte comme une métaclasse PowerAMC standard, et fait l'objet :
   - D'une liste distincte dans le menu **Modèle** - la liste de la métaclasse parent n'affichera pas les objets ayant le stéréotype de métaclasse. Ces objets seront affichés dans une liste distincte, sous la liste de la métaclasse parent. Les objets créés dans la nouvelle liste portent par défaut le nouveau stéréotype de métaclasse. Si vous changez le stéréotype d'un de ces objets, il disparaît de cette liste à sa prochaine ouverture.
   - Une catégorie dans l'Explorateur d'objets et une commande sous **Nouveau**, lorsque vous faites un clic droit sur un modèle ou un package.
   - Titre de feuille de propriétés basé sur le stéréotype de métaclasse.

2. [facultatif] Spécifiez une icône et un autre pour créer des instances du stéréotype de métaclasse (voir *Spécification d'une icône et d'un outil personnalisé pour un stéréotype* à la page 57).
3. Cliquez sur **Appliquer** pour enregistrer les modifications.

Spécification d'une icône et d'un outil personnalisé pour un stéréotype

Vous pouvez spécifier une icône pour un stéréotype afin de permettre aux utilisateurs d'identifier les instances de la métaclasse ayant ce stéréotype dans l'Explorateur, dans des feuilles de propriétés ou à d'autres endroits de l'interface. Vous pouvez également spécifier un outil permettant à un utilisateur de créer des instances à partir de la Boîte à outils.

1. Sur la page de propriétés du stéréotype, cliquez sur l'outil **Sélectionner une icône** pour afficher la boîte de dialogue Sélection d'une image. Sélectionnez l'image appropriée à utiliser comme icône dans l'interface puis cliquez sur **OK** pour l'associer au stéréotype.

   Remarque : L'icône est utilisée pour identifier les objets dans l'Explorateur d'objets et ailleurs dans l'interface, mais elle n'est pas utilisée comme symbole dans le diagramme. Pour spécifier un symbole de diagramme personnalisé, voir *Symboles personnalisés (Profile)* à la page 95.

2. [facultatif] Pour permettre à un outil de créer des instances de la métaclasse ayant le stéréotype à partir de la Boîte à outils, sélectionnez **Outil personnalisé dans la Boîte à outils**. Si vous ne sélectionnez pas cette option, les utilisateurs ne pourront créer des instances de la métaclasse avec ce stéréotype que depuis le menu Modèle ou par clic droit sur le noeud de modèle. Les outils personnalisés apparaissent dans un groupe séparé de la Boîte à outils, nommé d'après le fichiers de ressource dans lequel ils sont définis.

   Remarque : Si vous n'avez pas spécifié d'icône, c'est le marteau qui sera utilisé par défaut pour l'outil.

3. [facultatif] Cliquez dans la zone `<Zone de test pour le clic de curseur>` pour prévisualiser l'apparence du curseur de l'outil.
4. Cliquez sur **Appliquer** pour enregistrer les modifications.

Critères (Profile)

Vous pouvez contrôler le traitement des instances d'une métaclasse en fonction de leur respect d'un ou de plusieurs critères. Alors que vous ne pouvez appliquer qu'un stéréotype à une instance de métaclasse, vous pouvez tester l'instance en fonction de plusieurs formes de critères.

Vous pouvez définir un ou plusieurs critères pour une métaclasse sélectionnée. Les critères permettent de définir les mêmes extensions que les stéréotypes.

Lorsqu'une instance de métaclasse remplit la condition du critère, les extensions définies sur le crutère sont appliquées à l'instance. S'il y a un sous-critère, la condition du critère et celle du sous-critère doivent toutes deux être remplies pour que les extensions appropriées soient appliquées à l'instance.

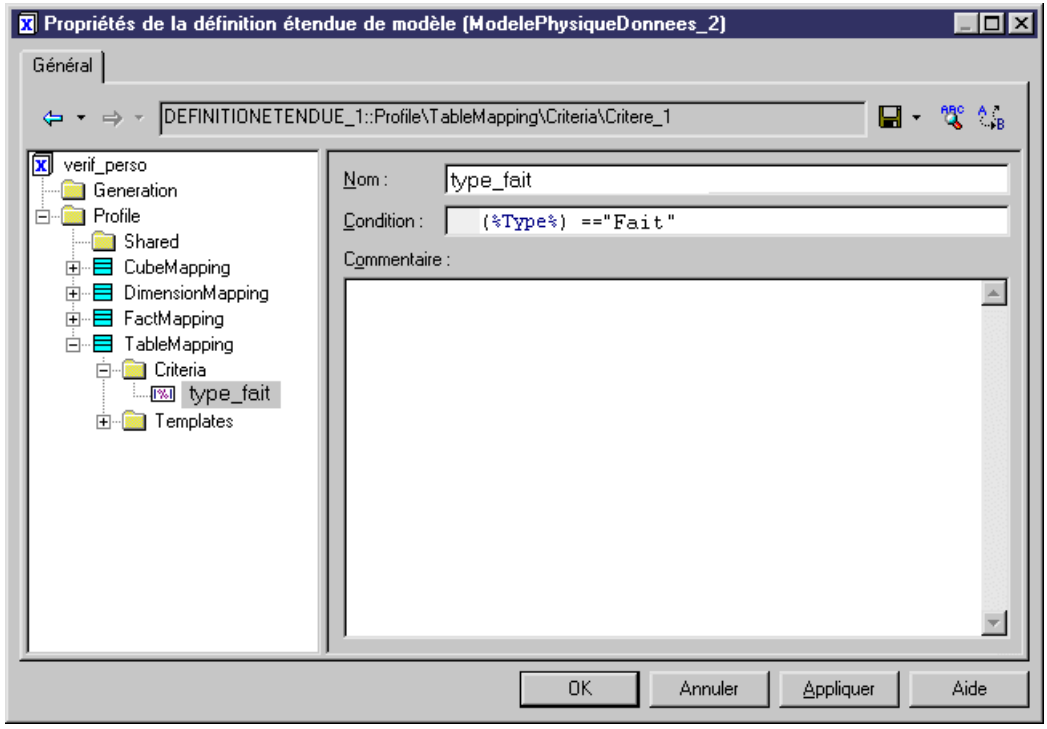
Création d'un critère

Vous pouvez créer un critère dans un profil.

1. Pointez sur une métaclasse, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau > Critère** dans le menu contextuel.

   Un nouveau critère est créé avec un nom par défaut.

2. Modifiez le nom par défaut dans la zone Nom, puis saisissez une condition dans la zone Condition. Vous pouvez utiliser n'importe quelle expression valide utilisée par la macro .if (voir *Macro .if* à la page 320).



3. Cliquez sur OK pour valider les modifications.

Propriétés d'un critère

Vous spécifiez les propriétés pour un critère en sélectionnant l'entrée correspondante dans l'Editeur de ressources.

Propriété	Description
Nom	Spécifie le nom du critère.
Condition	Spécifie la condition que les instances doivent remplir afin d'accéder aux extensions de critère. Vous pouvez utiliser n'importe quelle expression valide pour la macro .if du langage de génération par template de PowerAMC (voir *Chapitre 5, Personnalisation de la génération à l'aide du langage de génération par template* à la page 285. Vous pouvez faire référence aux attributs étendus définis au niveau de la métaclasse dans la condition, mais pas à ceux définis dans le critère lui-même. Par exemple, dans un MPD, vous pouvez personnaliser les symboles des tables de fait en créant un critère qui va tester le type de la table au moyen de la condition suivante : `(%DimensionalType% == "1")` `%DimensionalType%` est un attribut de l'objet `BaseTable`, qui comporte un jeu de valeurs définies, incluant `"1"`, qui correspond au fait `"fact"`. Pour plus d'informations, sélectionnez **Aide > Aide sur les objets du métamodèle**, puis allez à la section **Libraries > PdPDM > Abstract Classes > BaseTable**.
Parent	Spécifie un critère parent du critère. Vous pouvez sélectionner un critère défini dans la même métaclasse ou dans une métaclasse parent. Cliquez sur l'outil **Propriétés** pour revenir au parent dans l'arborescence et afficher ses propriétés.
Commentaire	Spécifie des informations supplémentaires relatives au critère.

Objets, sous-objets et liens étendus (Profile)

Les objets, sous-objets et liens étendus sont des métaclasses spéciales qui sont conçues pour vous permettre d'ajouter des types entièrement nouveaux d'objets dans vos modèles, plutôt que de les baser sur des objets PowerAMC existants

Pour plus d'informations sur les métaclasses, voir *Métaclasses (Profile)* à la page 50. Vous devez utiliser les objets, sous-objets et liens étendus comme suit :

- Objets étendus– peuvent être créés n'importe où
- Sous-objets étendus – ne peuvent être créés que dans la feuille de propriétés de leur objet parent, dans laquelle ils sont définis via une composition étendue (voir *Collections et compositions étendues (Profile)* à la page 72)
- Liens étendus – peuvent être définis pour lier des objets étendus

Ajout d'objets étendus, de sous-objets étendus et de liens étendus dans un profil

Les objets, sous-objets et liens étendus ne s'affichent pas, par défaut, dans les modèles autres que le modèle libre, sauf si vous les ajoutez sous la forme d'une extension ou d'un autre fichier de ressource.

1. Pointez sur la catégorie **Profile**, cliquez le bouton droit de la souris, puis sélectionnez **Ajouter des métaclasses**, cliquez sur le sous-onglet **PdCommon** en bas de la boîte de dialogue pour afficher la liste des objets communs à tous les modèles.
2. Cochez ou décochez une ou plusieurs cases `ExtendedLink`, `ExtendedSubObject` et `ExtendedObject` puis cliquez sur **OK** afin de les ajouter à votre profil.

   Remarque : Pour rendre disponibles les outils de création d'objets et de liens étendus dans la Boîte à outils des modèles autres que le modèle libre, vous devez les ajouter via la boîte de dialogue disponible en sélectionnant **Outils > Personnaliser les menus et les outils**.
3. [facultatif] Pour créer votre propre objet et ajouter des stéréotypes (voir *Stéréotypes (Profile)* à la page 53 et définir les extensions appropriées sous le stéréotype. Pour faire apparaître votre objet dans l'interface PowerAMC comme métaclasse standard, avec son propre outil, sa catégorie dans l'Explorateur d'objets et sa liste d'objets de modèle, sélectionnez **Utiliser comme métaclasse** dans la définition de stéréotype (voir *Promotion d'un stéréotype au statut de métaclasse* à la page 56).
4. Cliquez sur **Appliquer** pour enregistrer les modifications.

Matrices de dépendances (Profile)

La matrices de dépendance permet de passer en revue et de créer des liens entre des objets de n'importe quel type. Vous spécifiez une métaclasse pour la ligne de matrice, et la même ou une autre métaclasse pour les colonnes. Le contenu des cellules est ensuite calculé à partir d'une collection ou d'un objet lien.

Par exemple, vous pouvez créer des matrices de dépendance qui montrent les liens entre les types d'objets suivants :

- Entre des classes et des interfaces de MOO – connectées par des réalisations
- Entre des tables de MPD – connectés par des références

	DBA Materials	DBA Sales	DBA Staff
Customers		✓	
Divisions			✓
Employees			✓
Groups			✓
Order Lines		✓	

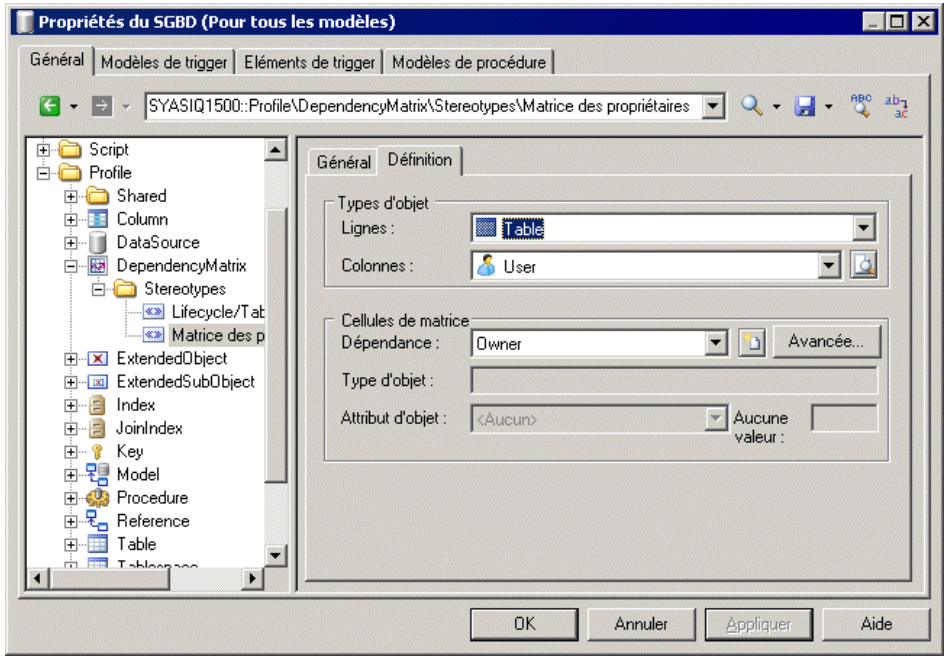
- Entre des tables de MPD et des classes de MOO – connectées par des dépendances étendues

Création d'une matrice de dépendances

Vous pouvez créer une matrice de dépendance dans un profil.

1. Pointez sur la catégorie **Profile** cliquez le bouton droit de la souris, puis sélectionnez **Ajouter une matrice de dépendance**. Vous ajoutez ainsi la métaclasse DependencyMatrix au profil et créez un stéréotype sous cette métaclasse, dans lequel vous allez définir les propriétés de la matrice.
2. Saisissez un nom pour la matrice (par exemple `Matrice des propriétaires de table`) avec un libellé et un libellé pluriel à utiliser dans l'interface PowerAMC, ainsi qu'un nom par défaut pour les matrices que les utilisateurs vont créer à partir de cette définition.
3. Cliquez sur l'onglet **Définition** pour spécifier les lignes et colonnes de votre matrice.
4. Sélectionnez un type d'objet à partir du modèle courant afin de remplir les lignes de votre matrice et un type d'objet dans le type de modèle courant ou dans un autre type de modèle afin de remplir les colonnes.
5. Spécifiez de quelle façon les lignes et les colonnes de votre matrice seront associées en sélectionnant une dépendance dans la liste.

   Seules les dépendances directes sont disponibles dans la liste. Pour spécifier une dépendance plus complexe, cliquez sur le bouton **Avancée** pour afficher la boîte de dialogue Définition du chemin de dépendance (voir *Spécification des dépendances avancées* à la page 62).



6. Dans le cas de certaines dépendances, le **Type d'objet** sur lequel la dépendance est basé sera affiché, et vous pouvez sélectionner un **Attribut d'objet** pour afficher les cellules de matrice avec le symbole **Aucune valeur**, qui s'affiche si l'attribut n'est pas défini dans une instance particulière.
7. Cliquez sur **OK** pour enregistrer votre matrice et fermer l'éditeur de ressources.

   Vous pouvez maintenant créer des instances de la matrice dans votre modèle comme suit :

   - Sélectionnez **Vue > Diagramme > Nouveau diagramme > Nom de matrice**.
   - Pointez sur le fond d'un diagramme, cliquez le bouton droit de la souris, puis sélectionnez **Diagramme > Nouveau diagramme > Nom de matrice**.
   - Pointez sur le modèle dans l'Explorateur d'objets, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau > Nom de matrice**.

Remarque : Pour plus d'informations sur l'utilisation des matrices de dépendance, voir *Guide des fonctionnalités générales > L'interface de PowerAMC > Diagrammes, matrices et symboles > Matrices de dépendance*.

Spécification des dépendances avancées

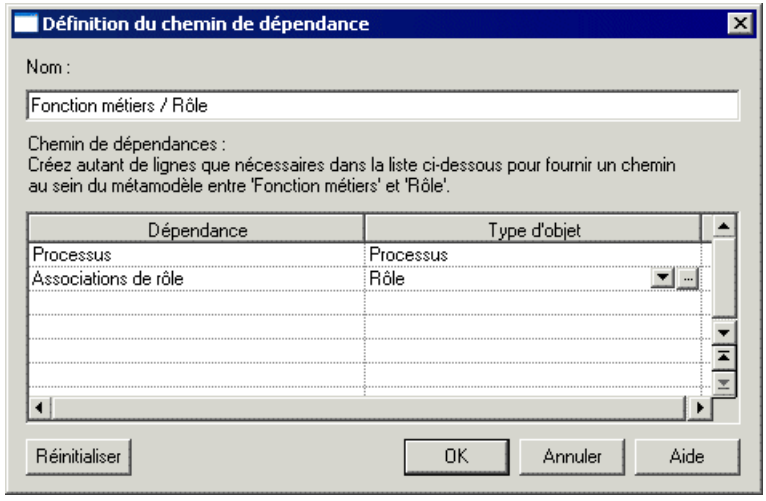
Vous pouvez examiner les dépendances entre deux types d'objet qui ne sont pas directement associés l'un à l'autre, en utilisant la boîte de dialogue Définition du chemin de dépendance,

qui est accessible en cliquant sur le bouton Avancé de l'onglet Définition, et qui permet de spécifier un chemin passant par autant d'objets intermédiaires que nécessaire.

Chaque ligne de cette boîte de dialogue constitue une étape sur un chemin de dépendance :

Propriété	Description
Nom	Spécifie un nom pour le chemin de dépendance. Par défaut, cette zone est renseignée à l'aide des objets d'origine et de destination.
Dépendance	Spécifie la dépendance pour cette étape sur le chemin. La liste est renseignée avec toutes les dépendances possibles compte tenu du type d'objet précédent.
Type d'objet	Spécifie le type d'objet lié au type d'objet précédent par le biais de la dépendance sélectionnée. Cette zone est automatiquement renseignée si un seul type d'objet est disponible via la dépendance sélectionnée.

Dans l'exemple suivant, un chemin est identifié entre les fonctions métiers et les rôles, en passant de la fonction métiers via les processus qu'elles contient jusqu'au rôle lié par le biais d'une association de rôle :



Propriétés d'une matrice de dépendances

Vous spécifiez les propriétés pour une matrice de dépendance en sélectionnant l'entrée correspondante dans l'Editeur de ressources.

Les matrices de dépendances sont basées sur les stéréotypes. Pour plus d'informations sur les propriétés de l'onglet **Général**, voir *Propriétés d'un stéréotype* à la page 55. Les propriétés suivantes sont disponibles sur l'onglet **Définition de matrice** :

Propriété	Description
Lignes	Spécifie le type d'objet avec lequel remplir vos lignes de matrice.
Colonnes	Spécifie le type d'objet avec lequel remplir vos colonnes de matrice. Cliquez sur le bouton **Sélectionner une métaclasse** à droite de la liste pour sélectionner une métaclasse dans un autre type de modèle.
Cellules de matrice	Spécifie la façon dont les lignes et les colonnes de votre matrice seront associées. Vous devez spécifier une **Dépendance** dans la liste, qui inclut toutes les collection et tous les liens disponibles pour l'objet. Dans le cas de certaines dépendances, le **Type d'objet** sur lequel est basé la dépendance sera affiché, et vous pouvez sélectionner un **Attribut d'objet** à afficher dans les cellules de matrice avec le symbole **Aucune valeur**, qui est affiché si cet attribut n'est pas défini dans une instance particulière. Cliquez sur le bouton **Créer** à droite de la liste pour créer une nouvelle collection étendue (voir *Collections et compositions étendues (Profile)* à la page 72) reliant vos objets, ou sur le bouton **Avancée** pour spécifier un chemin de dépendance complexe (voir *Spécification des dépendances avancées* à la page 62).

Attributs étendus (Profile)

Les attributs étendus permettent de définir des métadonnées supplémentaires pour vos objets.

Elles peuvent être définies pour des métaclasses, des stéréotypes et des critères, ce afin de :

- *Contrôler la génération* pour une cible de génération particulière. Dans ce cas, les attributs étendus sont définis dans le langage ou SGBD cible du modèle. Par exemple, dans le langage objet Java, plusieurs métaclasses sont dotées d'attributs étendus utilisés pour la génération de commentaires Javadoc.
- *Compléter la définition des objets du modèle* dans des extensions. Par exemple, dans l'extension Sybase ASA Proxy Tables, l'attribut étendu appelé GenerateAsProxyServer dans la métaclasse DataSource est utilisé pour définir la source de données comme un serveur proxy.

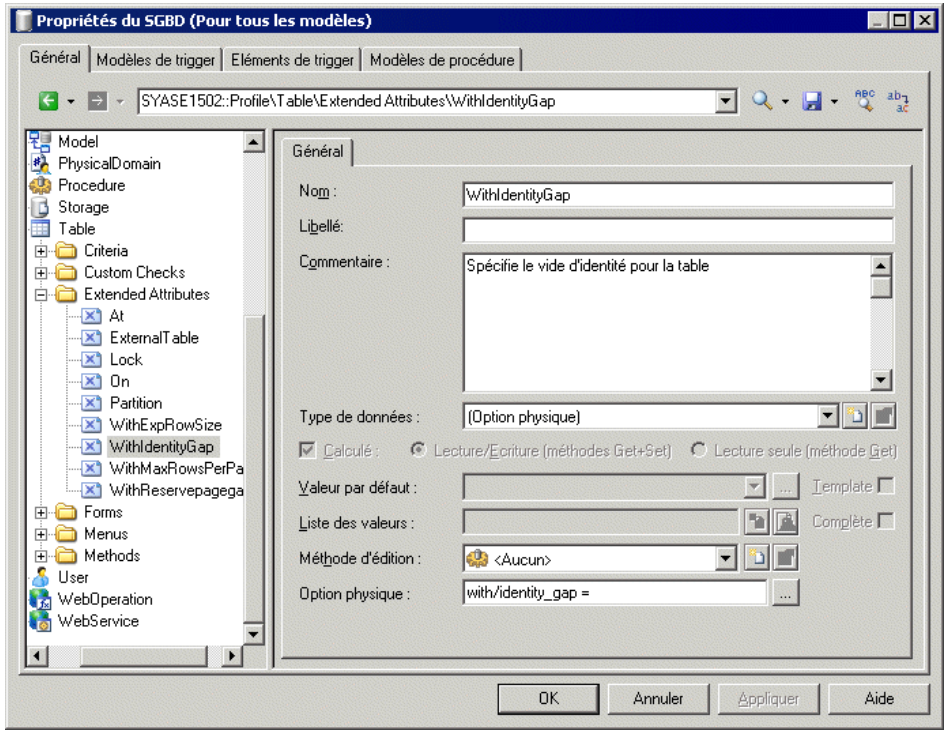
Remarque : Par défaut, les attributs étendus sont répertoriés sur un onglet Attributs étendus générique sur la feuille de propriétés d'objet. Vous pouvez personnaliser l'affichage des attributs en les insérant dans des formulaires (voir *Formulaires (Profile)* à la page 78). Si tous les attributs étendus sont alloués aux formulaires, la page générique n'est pas affichée.

Création d'un attribut étendu

Vous pouvez créer un attribut étendu pour une métaclasse, un stéréotype ou un critère.

1. Pointez sur une métaclasse, un stéréotype ou un critère dans la catégorie Profile, puis sélectionnez **Nouveau > Attribut étendu**.

2. Spécifiez les propriétés nécessaires.



3. Cliquez sur **Appliquer** pour enregistrer vos modifications.

Propriétés d'un attribut étendu

Vous spécifiez les propriétés d'un attribut étendu en sélectionnant l'entrée correspondante dans l'Editeur de ressources.

Propriété	Description
Nom	Spécifie le nom interne de l'attribut étendu, qui peut être utilisé pour le scripting.
Libellé	Spécifie le nom d'affichage de l'attribut, qui sera affiché dans l'interface PowerAMC.
Commentaire	Fournit des informations supplémentaires relatives à l'attribut étendu.

Propriété	Description
Type de données	Spécifie la forme des données qui doivent être conservées par l'attribut étendu. Vous pouvez choisir l'une des valeurs suivantes : • Booléen • Couleur • Date ou Heure • Fichier ou Répertoire • Entier, Réel ou Hexadécimal • Police, Nom de police ou Style de police • Objet - Spécifiez le **Type d'objet**, le **Stéréotype d'objet** (le cas échéant), ainsi que le **Nom de la collection inverse** dans les propriétés. Pour plus d'informations, voir *Liaison d'objets à l'aide d'attributs étendus* à la page 72. • Chaîne (monoligne) ou Texte (multiligne) Pour créer votre propre type de données, cliquez sur l'outil **Créer un type d'attribut étendu** à droite de cette zone (voir *Création d'un type d'attribut étendu* à la page 69).

Calculé	Spécifie que l'attribut étendu est calculé depuis d'autres valeurs à l'aide de VBScript sur les onglets **Script de méthode Get**, **Script de méthode Set** et **Script global**. Lorsque vous cochez cette case, vous devez choisir entre :
	• Lecture/Ecriture (méthodes Get+Set)
	• Lecture seule (méthode Get)
	Dans l'exemple de script suivant, l'attribut étendu `FileGroup` tire sa valeur de l'option physique `filegroup` de l'objet et la définit :
	`Function %Get%(obj)`
	`%Get% = obj.GetPhysicalOptionValue("on/<filegroup>")`
	`End Function`
	
	`Sub %Set%(obj, value)`
	`obj.SetPhysicalOptionValue "on/<filegroup>", value`
	`End Sub`

Valeur par défaut	[Sauf si `Calculé`] Spécifie une valeur par défaut pour l'attribut. Vous pouvez spécifier la valeur de l'une des façons suivantes : • Saisissez la valeur directement dans la liste. • [types de données prédéfinis] Cliquez sur le bouton Points de suspension pour afficher une boîte de dialogue qui répertorie les valeurs possibles. Par exemple, si le type de données est défini à Couleur, le bouton Points de suspension ouvre une fenêtre de palette. • [types de données utilisateur] Sélectionnez une valeur dans la liste.

Propriété	Description
Template	[Sauf si Calculé] Spécifie que la valeur de l'attribut doit être évaluée comme un template de langage de génération par template au moment de la génération. Par exemple, si la valeur de l'attribut est définie à %Code%, elle sera générée comme la valeur de l'attribut code de l'objet approprié. Par défaut (lorsque cette case n'est pas cochée, l'attribut est évalué de façon littérale, et une valeur de %Code% sera générée comme chaîne %Code%.

Liste des valeurs	Spécifiez une liste des valeurs possibles pour l'attribut étendu de l'une des façons suivantes :
	• Saisissez une liste statique de valeurs séparées par des points-virgules directement dans la zone.
	• Utilisez les outils à droite de la liste afin de créer ou de sélectionner un template de langage de génération par template afin de générer la liste de façon dynamique.
	Si l'attribut est de type Objet, et si vous ne souhaitez pas filtrer la liste des objets disponibles, vous pouvez laisser cette zone à blanc.
	Pour effectuer un filtrage simple de la liste des objets, utilisez la macro .collection (voir *Macro .collection* à la page 311). Dans l'exemple suivant, seules les tables ayant l'attribut Généré défini à true seront disponibles pour la sélection :
	`.collection(Model.Tables, %Generated%==true)`
	Pour un filtrage plus complexe, utilisez la macro foreach_item (voir *Macro .foreach_item* à la page 316) :
	`.foreach_item (Model.Tables)`
	`  .if %Generated%`
	`  .// (or more complex criteria)`
	`  %ObjectID%`
	`  .endif`
	`.next (\n)`
	Si l'attribut est basé sur un type d'attribut étendu (voir *Création d'un type d'attribut étendu* à la page 69), cette zone est indisponible car les valeurs du type d'attribut étendu seront utilisées.

Complète	Spécifie que les valeurs possibles pour l'attribut sont définies dans la **Liste des valeurs**, et que l'utilisateur ne peut pas saisir d'autre valeur.

Propriété	Description

Méthode d'édition	[Sauf si Calculé] Spécifie une méthode permettant de passer outre l'action par défaut associée à l'outil situé à droite de la zone.
	
	Cette méthode est souvent utilisée pour appliquer un filtre défini dans la zone **Liste des valeurs** à la boîte de sélection d'objet. Dans l'exemple suivant, seules les tables dont l'attribut Généré est défini à true sont disponibles pour la sélection :
	`Sub %Method%(obj)`
	
	`   Dim Mdl`
	`   Set Mdl = obj.Model`
	
	`   Dim Sel`
	`   Set Sel = Mdl.CreateSelection`
	
	`   If not (Sel is nothing) Then`
	`      Dim table`
	`      For Each table in Mdl.Tables`
	`          if table.generated then`
	`              Sel.Objects.Add table`
	`          end if`
	`      Next`
	
	`      ' Display the object picker on the selection`
	`      Dim selObj`
	`      set selObj = Sel.ShowObjectPicker`
	`      If Not (selObj is Nothing) Then`
	`         obj.SetExtendedAttribute "Storage-For-Each",`
	`selObj`
	`       End If`
	
	`      Sel.Delete`
	`   End If`
	
	`End Sub`

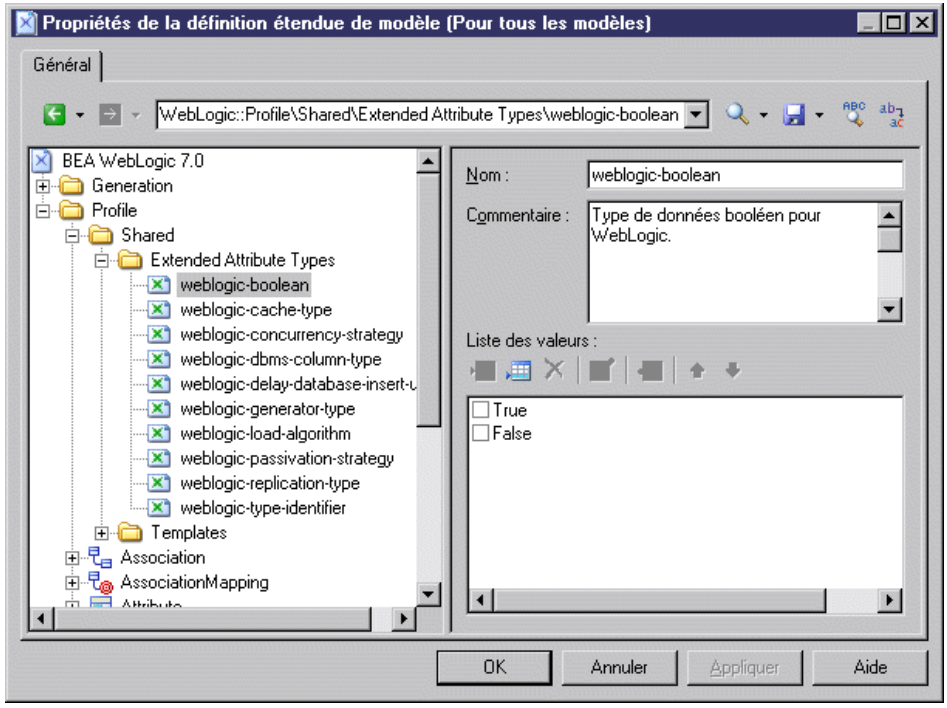
Format de texte	[Pour le type de données Texte uniquement] Spécifie le langage contenu dans l'attribut texte. Si vous sélectionnez une valeur autre que Texte, une barre d'outils d'éditeur et (le cas échéant) une coloration syntaxique sont fournis dans les champs de formulaire associés.
Type d'objet	[Pour les types de données Objet uniquement] Spécifie le type de l'objet que l'attribut contient (par exemple, Utilisateur, Table, Classe).
Stéréotype d'objet	[Pour les types de données Objet uniquement] Spécifie le stéréotype que doivent avoir les objets de ce type pour pouvoir être sélectionnés.

Propriété	Description
Nom de la collection inverse	[Pour les types de données `Objet` uniquement] Spécifie le nom sous lequel les liens vers l'objet seront répertoriés dans l'onglet **Dépendances** de la feuille de propriétés de l'objet cible. Une collection étendue portant le même nom que l'attribut étendu, qui gère ces liens, est automatiquement créée pour tous les attributs étendus non-calculés du type Objet, et est supprimée lorsque vous supprimez l'attribut étendu, changez son type ou cochez la case **Calculé**.
Option physique	[Pour les types de données [Option physique] uniquement] Spécifie l'option physique à laquelle l'attribut étendu est associé. Cliquez sur le bouton Points de suspension à droite de cette zone pour sélectionner une option physique. Pour plus d'informations, voir *Ajout d'options physiques de SGBD dans vos formulaires* à la page 85.

Création d'un type d'attribut étendu

Vous pouvez créer des types d'attribut étendu et des valeurs autorisées pour les attributs étendus. La création de types d'attribut étendu permet de réutiliser la même liste de valeurs pour plusieurs attributs étendus sans devoir rédiger le code correspondant.

1. Pointez sur la catégorie `Profile\Shared`, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau** > **Type d'attribut étendu**.
2. Spécifiez les propriétés appropriées, y compris la liste des valeurs et une valeur par défaut.



3. Cliquez sur **Appliquer** pour enregistrer vos modifications.

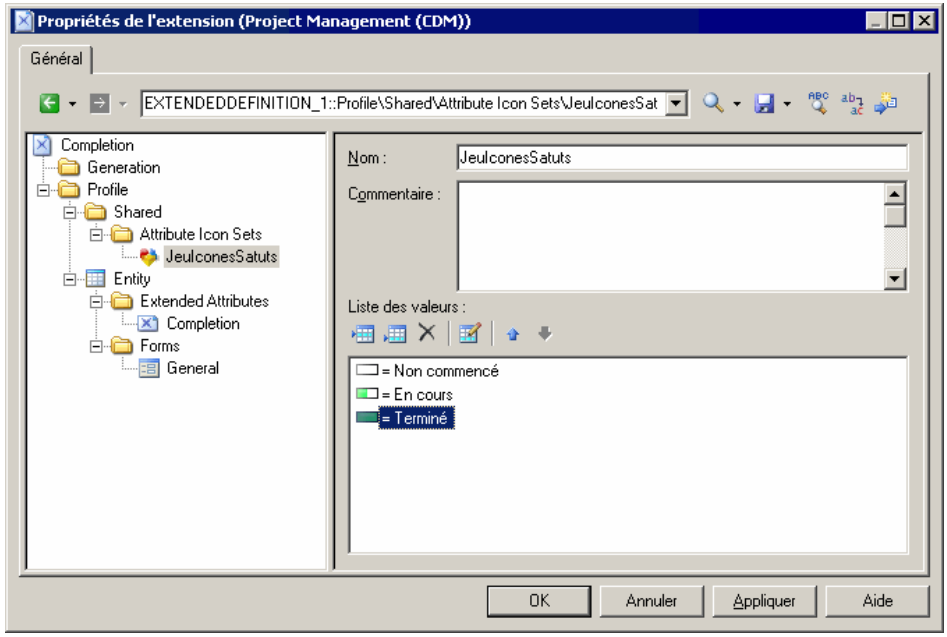
   Le nouveau type partagé est disponible pour tout attribut étendu dans la zone **Type de données**. Vous pouvez également définir une liste de valeurs pour un attribut étendu donné directement dans cette zone (voir *Propriétés d'un attribut étendu* à la page 65).

Spécification d'icônes pour les valeurs d'attributs

Vous pouvez spécifier l'affichage d'icônes sur les objets de symbole à la place de valeurs d'attributs étendus en créant un jeu d'icônes avec une icône différente pour chaque valeur d'attribut possible.

1. Créez un attribut étendu, puis sélectionnez un type de données standard ou un type d'attribut étendu (voir *Création d'un type d'attribut étendu* à la page 69).
2. Si nécessaire, spécifiez une liste de valeurs possibles et une valeur par défaut.
3. Cliquez sur l'outil **Créer un jeu d'icônes** à droite de la liste **Jeu d'icônes**.

   Un nouveau jeu d'icônes est créé sous **Profile > Shared > Attribute Icon Sets**, initialisé avec les valeurs possibles et une icône vide pour chaque valeur n'ayant pas encore d'icône définie (=*).
4. Pour chaque valeur dans la liste, double-cliquez sur la valeur, puis cliquez sur l'outil **Sélectionner une icône** afin de sélectionner une icône pour représenter cette valeur sur les symboles d'objet.

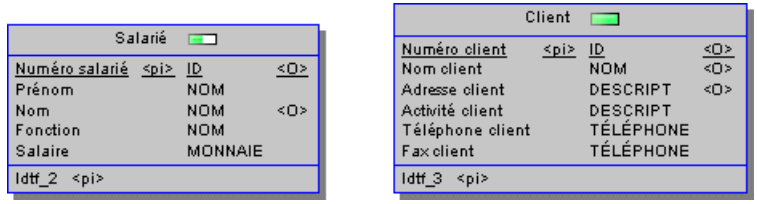


Remarque : Par défaut, la zone **Opérateur de filtre** est définie à `=`, et chaque icône correspond exactement à une valeur possible. Pour faire en sorte qu'une icône puisse correspondre à plusieurs valeurs, utilisez l'opérateur `Entre` ou un autre opérateur avec une **Valeur de filtre** appropriée. Par exemple, dans un jeu d'icônes défini pour un attribut `avancement` pour lequel l'utilisateur peut spécifier une valeur entre 0 et 100%, vous pouvez utiliser trois icônes :

- Non commencé - `= 0`
- En cours - `Entre 1,99`
- Terminé - `= 100`

5. Si nécessaire, ajoutez l'attribut dans un formulaire (voir *Formulaires (Profile)* à la page 78), pour permettre aux utilisateurs de modifier sa valeur.
6. Cliquez sur **OK** pour enregistrer vos modifications et revenir au modèle.
7. Pour activer l'affichage de l'icône sur votre symbole d'objets, sélectionnez **Outils > Préférences d'affichage**, sélectionnez votre type d'objet, puis cliquez sur le bouton **Avancé** pour ajouter votre attribut au symbole. Pour obtenir des informations détaillées sur l'utilisation des préférences d'affichage, voir *Guide des fonctionnalités générale > L'interface de PowerAMC > Diagrammes, matrices et symboles > Préférences d'affichage*.

   Votre attribut est maintenant affiché sur des symboles d'objets. Dans l'exemple suivant, l'entité `Salarié` est `En cours`, tandis que `Client` est `Terminé` :



Liaison d'objets à l'aide d'attributs étendus

Lorsque vous spécifiez le type de données [Objet], vous activez l'affichage des champs Type d'objet, Stéréotype d'objet et Collection inverse.

La zone Type d'objet spécifie la catégorie d'objet vers laquelle vous souhaitez établir un lien, et la zone de stéréotype permet de filtrer les objets disponibles pour cette sélection.

Par exemple, sous la métaclasse Table, vous créez un attribut étendu appelé Propriétaire, sélectionnez [Objet] dans la zone Type de données, et User dans la zone Type d'objet. Vous nommez la collection inverse "Tables ayant un propriétaire". Vous pouvez définir l'attribut Propriétaire dans la feuille de propriétés d'une table, et la table sera répertoriée sur l'onglet Dépendances de la feuille de propriétés de l'utilisateur, sous le nom "Tables ayant un propriétaire".

Collections et compositions étendues (Profile)

Une collection étendue permet d'associer plusieurs instances d'une métaclasse avec une instance d'une autre métaclasse.

Par exemple, pour attacher des documents contenant des spécifications de cas d'utilisation aux différents packages d'un modèle, vous pouvez créer une collection étendue dans la métaclasse Package et définir FileObject comme métaclasse cible. Vous pouvez créer une collection étendue sur la métaclasse Process du MOO pour montrer les composants utilisés comme ressources pour le processus, ce pour obtenir une vision plus précise de la mise en oeuvre physique du processus.

L'association entre les objets parent et enfant est relativement faible, de sorte que :

- Si vous copiez et collez un objet avec des collections étendues, les objets associés ne sont pas copiés.
- Si vous déplacez un objet avec des collections étendues, le lien avec les objets associés est préservé (le cas échéant, à l'aide de raccourcis).

Une composition étendue permet d'associer plusieurs instances de la métaclasse de sous-objet avec une métaclasse. L'association est plus forte que celle créée à l'aide d'une collection étendue – les sous-objets ne peuvent être créés qu'au sein de leur objet parent et sont déplacés, copiés et/ou supprimés avec ces derniers.

Lorsque vous créez une collection ou composition étendue dans une métaclasse, un nouvel onglet portant le nom de cette collection ou composition est ajouté dans la feuille de propriétés de la métaclasse.

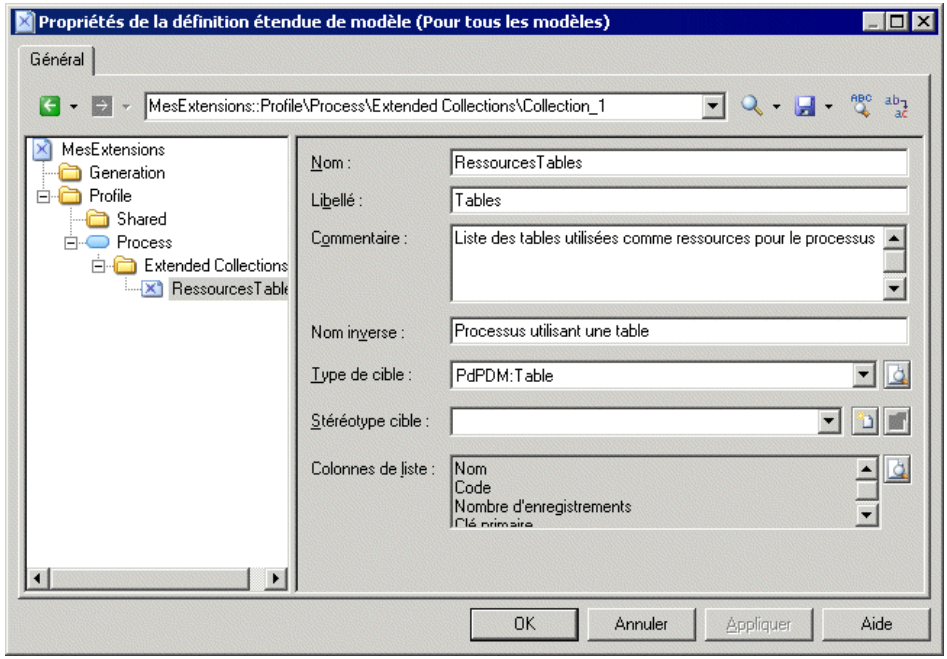
Remarque : Si vous créez une collection ou composition étendue sur un stéréotype ou critère, l'onglet correspondant s'affiche uniquement si l'instance de métaclasse porte le stéréotype ou satisfait au critère.

Pour les collections étendues, les feuilles de propriétés des objets contenus dans la collection répertorient leur objet parent dans l'onglet Dépendances.

Création de collections et de compositions étendues

Vous pouvez créer une collection étendue pour une métaclasse, un stéréotype ou un critère.

1. Pointez sur une métaclasse, un stéréotype ou un critère, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau > Collection étendue** ou **Composition étendue**.
2. Saisissez un **nom** de script et un **Libellé** d'affichage qui sera utilisé comme nom pour l'onglet associé à la collection dans la feuille de propriétés de l'objet parent.
3. [facultatif] Saisissez un **Commentaire** et un **Nom inverse**.
4. Sélectionnez une métaclasse dans la liste **Type de cible** afin de spécifier le type d'objet qui sera contenu dans la collection.
5. [facultatif] Sélectionnez ou spécifiez un **Stéréotype cible** pour affiner la définition des instances de la métaclasse cible qui peut apparaître dans la collection. Cliquez sur l'outil **Créer** à droite de cette zone pour créer un nouveau stéréotype.
6. [facultatif] Cliquez sur l'outil **Personnaliser les colonnes par défaut** afin de modifier les colonnes qui seront affichées par défaut lorsque l'utilisateur affichera l'onglet de propriétés associé à la collection.



7. Cliquez sur **Appliquer** pour enregistrer vos modifications.

   Vous pouvez voir l'onglet associé à la collection en affichant la feuille de propriétés d'une instance de métaclasse. L'onglet contient un outil **Ajouter des objets** (et, si la métaclasse appartient au même type de modèle, un outil **Créer un objet**), pour enrichir la collection.

Propriétés d'une collection/composition étendue

Vous spécifiez les propriétés pour une collection ou composition étendue en sélectionnant l'entrée correspondante dans l'Editeur de ressources.

Propriété	Description
Nom	Spécifie le nom de la collection étendue.
Label	Spécifie le nom d'affichage de la collection, qui sera utilisé dans l'interface de PowerAMC.
Commentaire	Décrit la collection étendue.
Nom inverse	[collection étendue uniquement] Spécifie le nom qui doit s'afficher dans l'onglet **Dépendances** de la métaclasse cible. Si vous ne saisissez aucune valeur, un nom inverse est automatiquement généré.

Propriété	Description
Type de cible	Spécifie la métaclasse dont les instances vont apparaître dans la collection. Dans le cas des collections étendues, la liste affiche uniquement les métaclasses qui peuvent être directement instanciées dans le modèle ou package courant, mais pas les sous-objets tels que les attributs de classe ou les colonnes de table. Cliquez sur l'outil **Sélectionner une métaclasse** à droite de cette zone pour choisir une métaclasse à partir d'un autre type de modèle. Dans le cas des compositions étendues, seul ExtendedSubObject est disponible, et vous devez spécifier un stéréotype pour cette cible.
Stéréotype cible	[requis pour les compositions étendues] Spécifie un stéréotype pour filtrer le type cible. Vous pouvez sélectionner un stéréotype existant dans la liste, ou en créer un nouveau.
Colonnes de liste	Spécifie les colonnes de propriétés qui seront affichées par défaut dans l'onglet de la feuille de propriétés de l'objet parent associé à la collection. Cliquez sur l'outil Personnaliser les colonnes par défaut à droite de cette liste ajouter ou supprimer des colonnes.

Lorsque vous ouvrez un modèle contenant des collections ou compositions étendues et l'associez à un fichier de ressources qui ne les prend pas en charge, les collections restent visibles dans les différentes feuilles de propriétés, ce qui vous permet de supprimer des objets dans les collections qui ne sont plus prises en charge.

Collections calculées (Profile)

Vous définissez une collection calculée sur une métaclasse, un stéréotype ou un critère lorsque vous devez afficher une liste d'objets associés avec une sémantique personnalisée.

Les collections calculées (contrairement aux collections étendues) ne peuvent pas être modifiées par l'utilisateur (voir *Collections et compositions étendues (Profile)* à la page 72.

Vous créez des collections calculées pour :

- Afficher des dépendances personnalisées pour un objet sélectionné, la collection calculée s'affiche dans l'onglet Dépendances de la feuille de propriétés de l'objet. Vous pouvez double-cliquer sur des éléments et naviguer parmi les dépendances personnalisées.
- Affiner l'analyse d'impact en créant vos propres collections calculées afin d'être en mesure de mieux évaluer l'impact d'un changement. Par exemple, dans un modèle dans lequel les colonnes et domaines peuvent diverger, vous pouvez créer une collection calculée sur la métaclasse domain qui répertorie toutes les colonnes qui utilisent le domaine et qui ont le même type de données.
- Améliorer vos rapports. Vous pouvez faire glisser n'importe quel livre sous un autre livre ou élément de liste et modifier sa collection par défaut afin de documenter un aspect particulier du modèle (voir *Guide des fonctionnalités générales > L'interface de*

PowerAMC > Rapports > L'Editeur de rapport > Ajout d'éléments dans un rapport > Modification de la collection d'un élément).

- Améliorer la génération à l'aide du langage de génération par template, puisque vous pouvez boucler sur des collections calculées personnalisées.

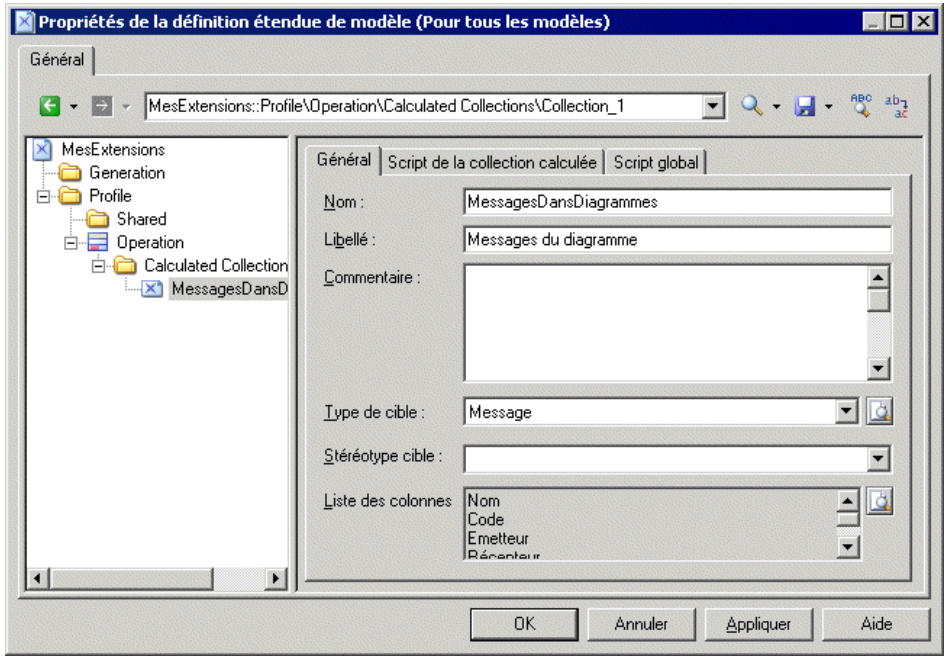
Par exemple, dans un MOO, vous pouvez être amené à créer une liste de diagrammes de séquence utilisant une opération, vous pouvez alors créer une collection calculée sur la métaclasse d'opération qui extrait cette information.

Dans un MPM, vous pouvez créer une collection calculée sur la métaclasse de processus qui répertorie les entités de MCD créée à partir des données associées au processus.

Création d'une collection calculée

Vous pouvez créer une collection calculée pour une métaclasse, pour un stéréotype ou pour un critère.

1. Pointez sur une métaclasse, un stéréotype ou un critère, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau > Collection calculée**.
2. Saisissez un **Nom** de script et un **Libellé** d'affichage destiné à être utilisé comme nom pour l'onglet associé à la collection dans la feuille de propriétés de l'objet parent.
3. [facultatif] Saisissez un **Commentaire** pour décrire la collection.
4. Sélectionnez une métaclasse dans la liste **Type de cible** afin de spécifier le type d'objet qui sera contenu dans la collection.
5. [facultatif] Sélectionnez ou spécifiez un **Stéréotype cible** pour affiner la définition des instances de la métaclasse cible qui peut apparaître dans la collection.
6. Cliquez sur l'onglet **Script de la collection calculée**, puis spécifiez un script qui va calculer quels objets vont former la collection. Si nécessaire, vous pouvez réutiliser les fonctions sur l'onglet **Script global**.



7. Cliquez sur **Appliquer** pour enregistrer vos modifications.

   Vous pouvez voir l'onglet associé à la collection en affichant la feuille de propriétés d'un instance de métaclasse.

Propriétés d'une collection calculée

Vous spécifiez les propriétés pour une collection étendue en sélectionnant l'entrée correspondante dans l'Editeur de ressources.

Propriété	Description
Nom	Spécifie le nom de la collection calculée.
Libellé	Spécifie le nom d'affichage de la collection, qui est utilisé dans l'interface de PowerAMC.
Commentaire	Décrit la collection calculée.
Type de cible	Définit la métaclasse dont les instances apparaîtront dans la collection. La liste affiche uniquement les métaclasses qui peuvent être instanciées directement dans le modèle ou package courant, comme des classes ou des tables, mais pas des sous-objets tels que les attributs de classe ou des colonnes de table. Cliquez sur l'outil **Sélectionner une métaclasse** en regard de la zone Type de cible pour sélectionner une métaclasse dans un modèle d'un autre type.

Propriété	Description
Stéréotype cible	Spécifie un stéréotype pour filtrer le type de cible. Vous pouvez sélectionner un stéréotype existant dans la liste, ou en saisir un nouveau.

L'onglet **Script de la collection calculée** contient la définition du corps de la fonctionnalité de collection calculée.

L'onglet **Script global** permet de partager des fonctions de bibliothèque et des attributs statiques dans le fichier de ressources. Vous pouvez déclarer des *variables globales* sur cet onglet, vous devez savoir que ces dernières ne sont pas réinitialisées chaque fois que la collection est calculée, et conservent leur valeur jusqu'à ce que vous modifiiez le fichier de ressources, ou jusqu'à la fermeture de PowerAMC. Cette caractéristique peut s'avérer une source d'erreurs, tout particulièrement lorsque les variables font référence à des objets qui peuvent être modifiés, voir supprimés. Assurez-vous de bien réinitialiser la variable globale si vous ne souhaitez pas conserver la valeur d'une exécution précédente.

Pour plus d'informations sur la définition d'un script et sur l'utilisation de l'onglet **Script global**, voir *Définition du script d'une vérification personnalisée* à la page 98 et *Utilisation du script global* à la page 101.

Formulaires (Profile)

Vous pouvez utiliser des formulaires afin de créer de nouveaux onglets de feuille de propriétés ou de remplacer des onglets existants, ou bien pour créer des boîtes de dialogue qui s'ouvrent grâce à des commandes de menus ou en cliquant sur des boutons sur vos onglets de feuilles de propriétés. La création d'un nouveau formulaire est une opération simple et facile en utilisant les outils de création de formulaire contenus dans l'éditeur de ressources.

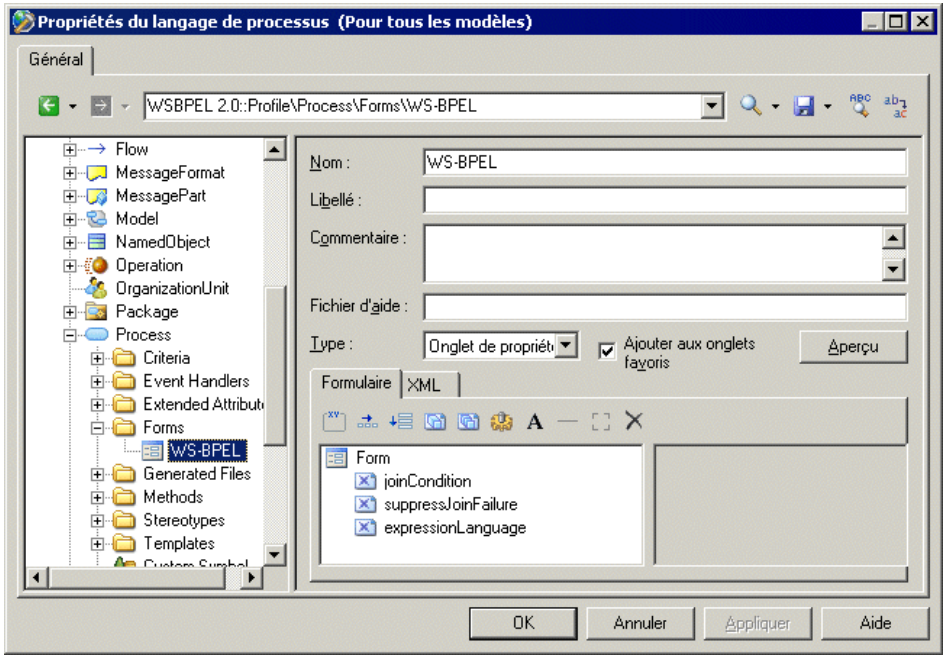
Par défaut, les attributs étendus s'affichent dans une liste sur l'onglet Attributs étendus d'une feuille de propriétés. En créant votre propre formulaire, vous pouvez rendre ces attributs plus visibles et faciles à utiliser, en les organisant de façon logique, en regroupant ceux qui sont liés et en mettant en évidence les plus importants. Les formulaires personnalisés sont utilisés dans les MPD pour mettre en exergue les options physiques les plus utilisées dans les onglets "Options physiques (Communes)".

Vous pouvez créer un formulaire sur n'importe quelle métaclasse dotée d'une feuille de propriétés, ou bien sur un stéréotype ou un critère. Dans le cas des onglets de propriétés, si l'onglet est lié à un stéréotype ou critère, il ne s'affiche que si l'instance de métaclasse porte ce stéréotype ou remplit le critère.

Création d'un formulaire

Vous pouvez créer un formulaire dans un profil afin de créer un nouvel onglet de propriétés ou une boîte de dialogue dans l'interface de PowerAMC, ou bien pour remplacer des onglets de feuilles de propriétés standard.

1. Pointez sur une métaclasse, un stéréotype ou un critère, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau** > **Formulaire** pour créer un formulaire vide.



2. Saisissez un **Nom** pour le script et un **Libellé** d'affichage pour le formulaire, sélectionnez un **Type** et saisissez toute autre propriété appropriée (voir *Propriétés d'un formulaire* à la page 80). Ce nom va s'afficher en haut de l'onglet de la feuille de propriétés ou dans la barre de titres de la boîte de dialogue. Vous pouvez également, le cas échéant, saisir une description du formulaire dans la zone **Commentaire**.
3. Insérez les contrôles nécessaires dans votre formulaire en utilisant les outils de la barre d'outils sur l'onglet **Formulaire** (voir *Ajout d'attributs étendus et d'autres contrôles dans votre formulaire* à la page 80).
4. Cliquez sur le bouton **Aperçu** pour contrôler la disposition de votre formulaire et, si vous le résultat vous convient, cliquez sur **Appliquer** pour enregistrer vos modifications.

Propriétés d'un formulaire

Vous spécifiez les propriétés pour un formulaire en sélectionnant l'entrée correspondante dans l'Editeur de ressources.

Propriété	Description
Nom	Spécifie le nom interne du formulaire, qui peut être utilisé dans des scripts.
Libellé	Spécifie le nom d'affichage du formulaire, qui sera affiché dans l'interface de PowerAMC.
Commentaire	Fournit des informations supplémentaires sur le formulaire.

Fichier d'aide	Permet d'activer l'affichage d'un bouton d'aide et spécifie une action qui sera effectuée lorsque le bouton ou la touche F1 sera enfoncé dans le contexte du formulaire.
	L'action peut être l'affichage d'un fichier d'aide (.hlp, .chm ou .html), et peut spécifier une rubrique particulière. Par exemple :
	`C:\PD1500\pddoc15.chm 26204`
	Si aucun suffixe de ficher d'aide n'est trouvé, la chaîne est traitée comme une commande d'interpréteur de commandes à exécuter. Par exemple, vous pouvez demander à PowerAMC d'ouvrir un simple fichier de texte :
	`notepad.exe C:\Temp\Readme.txt`

Type	Spécifie le type de formulaire. Vous pouvez choisir une des valeurs suivantes : • Boîte de dialogue – crée une boîte de dialogue qui peut être lancée à partir d'un menu ou via un bouton de formulaire • Onglet de propriétés – crée un nouvel onglet dans la feuille de propriétés de la métaclasse, du stéréotype ou du critère. • Remplacer l'onglet *<standard>* – remplace un onglet standard dans la feuille de propriétés de la métaclasse, du stéréotype ou du critère. Si votre formulaire est vide, il sera rempli avec les contrôles standard de l'onglet que vous remplacez.
Ajouter aux onglets favoris	[onglets de propriétés uniquement] Spécifie que l'onglet est affiché par défaut dans la feuille de propriétés de l'objet.

Ajout d'attributs étendus et d'autres contrôles dans votre formulaire

Vous insérez des contrôles dans votre formulaire en utilisant les outils de la barre d'outils située en haut de l'onglet Formulaire.

Vous pouvez réorganiser les contrôles dans l'arborescence des contrôles en les faisant glisser. Pour placer un contrôle dans un conteneur de contrôle (zone de groupe ou disposition

horizontale ou verticale), faites-le glisser sur le conteneur. Par exemple, si vous souhaitez afficher GUID, InputGUID, et OutputGUID dans une zone de groupe GUI, vous devez créer une zone de groupe, la nommer GUI et faire glisser ces trois attributs étendus sous la zone de groupe GUI.

Outil	Description
	**Ajouter une zone de groupe** - insère une zone de groupe, qui englobe d'autres contrôles dans un cadre nommé.
	**Ajouter un onglet** - insère une disposition en sous-onglets, dans lequel chaque contrôle enfant apparaît, par défaut, dans son propre sous-onglet. Pour placer plusieurs contrôles sur un seul sous-onglet, utilisez une disposition horizontale ou verticale.
	**Ajouter une disposition horizontale** - insère une disposition horizontale. Pour disposer les contrôles côte à côte, faites-les glisser sur la disposition horizontale dans la liste.
	**Ajouter une disposition verticale** - insère une disposition verticale. Pour disposer les contrôles les uns au-dessus des autres, faites-les glisser sur la disposition verticale dans la liste. Les dispositions verticales sont souvent utilisées avec une disposition horizontale, pour former des colonnes de contrôles.
	**Inclure un autre formulaire** - insère un formulaire défini sur cette métaclasse ou sur une autre métaclasse dans le formulaire courant (voir *Exemple : Inclusion d'un formulaire dans un autre formulaire* à la page 89).
	**Ajouter un attribut** – affiche une boîte de dialogue de sélection qui permet de sélectionner des attributs standard ou étendus appartenant à la métaclasse. Sélectionnez un ou plusieurs attributs, puis cliquez sur OK afin de les insérer dans le formulaire. Le type de contrôle associé à un attribut étendu dépend du type de l'attribut étendu : les attributs étendus booléens sont associés à des cases à cocher, les listes à des listes modifiables, le texte à des zones d'édition multiligne, et ainsi de suite. A moins que vous ne saisissiez un libellé, le nom de l'attribut est utilisé comme libellé pour le formulaire. Les commentaires saisis pour l'attribut sont affichés sous forme d'infobulle sur le formulaire.
	**Ajouter une collection** – affiche une boîte de dialogue de sélection qui permet de sélectionner des collections standard qui appartiennent à la métaclasse. Sélectionnez une ou plusieurs collections, puis cliquez sur OK pour les insérer dans le formulaire. Les collections sont affichées sous la forme de grilles standard avec tous les outils appropriés. A moins que vous ne saisissiez un libellé, le nom de la collection est utilisé comme libellé pour le formulaire. Les commentaires saisis pour la collection sont affichés sous forme d'infobulle sur le formulaire.

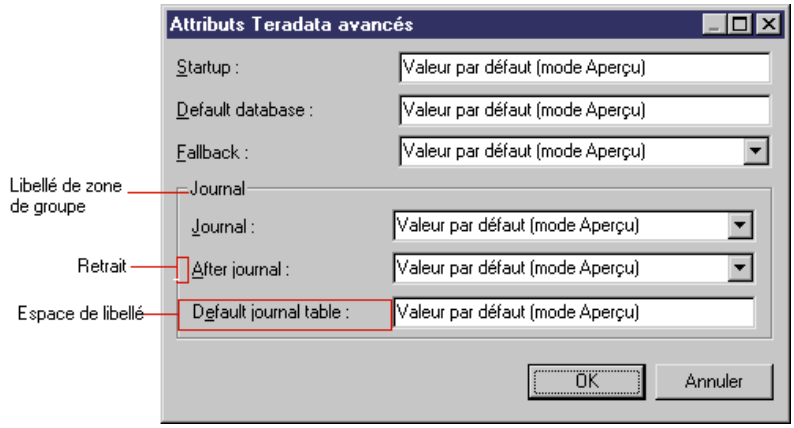
Outil	Description
	**Ajouter un bouton de méthode** - affiche une boîte de dialogue de sélection qui permet de sélectionner une ou plusieurs méthodes, qui seront associées au formulaire via des boutons. Cette liste est limité aux méthodes définies sous la même métaclasse dans le profil. Sélectionnez une ou plusieurs méthode, puis cliquez sur OK pour les insérer dans le formulaire. Chaque méthode est affichée sous la forme d'un bouton dans le formulaire. La méthode correspondante est appelée lorsque vous cliquez sur ce bouton. A moins que vous ne saisissiez un libellé, le nom de la méthode est utilisé comme libellé pour le bouton. Les commentaires saisis pour la méthode sont affichés sous forme d'infobulle sur le formulaire.
	**Ajouter une zone d'édition** [boîtes de dialogue uniquement] insère une zone d'édition.
	**Ajouter une zone d'édition multiligne** [boîtes de dialogue uniquement] - insère une zone d'édition multiligne sous l'élément sélectionné dans la liste.
	**Ajouter une liste modifiable** [boîtes de dialogue uniquement] - insère une liste modifiable.
	**Ajouter une zone de liste** [boîtes de dialogue uniquement] - insère une zone de liste.
	**Ajouter une case à cocher** [boîtes de dialogue uniquement] - insère une case à cocher.
	**Ajouter du texte** - insère un contrôle de texte.
	**Ajouter une ligne de séparation** – insère une ligne de séparation. La ligne est verticale si son contrôle parent est une disposition verticale.
	**Ajouter une zone d'espacement** – insère un espace vide.
	**Supprimer** – supprime le contrôle sélectionné.

Propriétés des contrôles d'un formulaire

Lorsque vous ajoutez des contrôles dans vos formulaires, vous pouvez spécifier des propriétés afin de contrôler leur format et leur contenu.

Propriété	Définition
Nom	Nom interne du contrôle. Ce nom doit être unique dans le formulaire. Le nom peut être utilisé dans des scripts pour obtenir et définir des valeurs de contrôle de boîte de dialogue (voir *Exemple : Ouverture d'une boîte de dialogue à partir d'un menu* à la page 112).

Propriété	Définition
Libellé	Spécifie un libellé pour le contrôle dans le formulaire. Si vous laissez cette zone vide, c'est le nom du contrôle qui est utilisé. Si vous saisissez un espace, aucun libellé n'est affiché. Vous pouvez insérer des retours à la ligne avec `\n`. Pour créer des raccourcis clavier permettant de naviguer dans les contrôles, faites précéder la lettre à utiliser comme raccourci d'une perluète. Si vous ne spécifiez aucun raccourci. PowerAMC en choisit un par défaut. Pour afficher une perluète dans un libellé, vous devez la faire précéder d'une autre perluète (par exemple : `&Johnson && Son` s'affiche sous la forme **Johnson & Son**.
Attribut	[formulaires inclus] Spécifie l'objet sur lequel le formulaire à inclure est défini. La liste est renseignée par les attributs de type `object` ainsi que par les objets suivants : • <aucune> - la métaclasse présente • Origine de génération - par exemple l'entité de MCD à partir de laquelle une table de MPD a été générée • Modèle - le modèle parent • Parent - l'objet parent immédiat pour les sous-objets (par exemple, la table contenant une colonne) • Dossier parent - l'objet parent immédiat pour les objets composites (par exemple, les processus de MPM qui contiennent d'autres processus) • Package parent - le package parent immédiat
Nom de formulaire	[formulaires inclus] Spécifie le nom du formulaire qui sera inclus. Vous pouvez : • Sélectionner un nom d'onglet de feuille de propriétés standard dans la liste. • Saisir le nom d'un formulaire personnalisé défini dans le fichier d'extension. • Saisir le nom d'un template de GTL (langage de génération par templates) afin de générer le code XML pour définir le formulaire.
Retrait	[contrôles de conteneur uniquement] Spécifie le nombre de pixels entre la marge gauche du conteneur (formulaire, zone de groupe, ou disposition horizontale ou verticale) et le début des libellés de ses contrôles

Propriété	Définition
Espace de libellé	[contrôles de conteneur uniquement] Spécifie l'espace en pixels réservé pour l'affichage des libellés des contrôles enfant entre le retrait du conteneur et les zones de contrôle. Pour aligner les contrôles sur les contrôles d'un conteneur précédent, saisissez une valeur négative. Par exemple, si vous avez deux zones de groupe, et que vous souhaitez que tous les contrôles des deux zones de groupes soient alignés, définissez le retrait approprié dans la première zone de groupe, puis définissez le retrait de la seconde zone de groupe à `-1`. Si un contrôle enfant est plus grand que la valeur spécifiée, la propriété d'espace du libellé est ignorée ; pour afficher le libellé, vous devez saisir un nombre de pixels supérieur à 50. 
Afficher le contrôle sous forme de libellé	[zones de groupe] Utilise le premier contrôle contenu au sein du groupe comme libellé.
Afficher l'attribut caché	[attributs étendus] Affiche en grisé les contrôles qui ne sont pas valides pour un formulaire particulier (car ils ne portent pas le stéréotype appropriés ou ne correspondent pas au critère). Si vous ne définissez pas cette option, les options non pertinentes sont cachées.
Valeur	[zones de saisie de boîte de dialogue] Spécifie une valeur par défaut pour le contrôle. Pour les attributs étendus, les valeurs par défaut doivent être spécifiées dans les propriétés de l'attribut (voir *Propriétés d'un attribut étendu* à la page 65).
Liste des valeurs	[zones de listes et listes modifiables] Spécifie une liste de valeurs possibles pour le contrôle. Pour les attributs étendus, les listes de valeurs doivent être spécifiées dans les propriétés de l'attribut (voir *Propriétés d'un attribut étendu* à la page 65).
Liste exclusive	[liste modifiables] Spécifie que seules les valeurs définies dans la **Liste des valeurs** peuvent être saisies dans la liste modifiable.

Propriété	Définition
Taille minimum (caractères)	Spécifie la largeur minimale (en caractères) à laquelle le contrôle peut être réduit si la fenêtre est redimensionnée.
Nombre minimum de lignes	Spécifie le nombre minimal de lignes auquel un contrôle multiligne peut être réduit si la fenêtre est redimensionnée.
Redimensionnement horizontal	Spécifie que le contrôle peut être redimensionné horizontalement si la fenêtre est redimensionnée.
Redimensionnement vertical	Spécifie que le contrôle multiligne être redimensionné verticalement si la fenêtre est redimensionnée.
Lecture seule	[formulaires inclus et zones de saisie de boîte de dialogue] Spécifie que le contrôle est en lecture seule, et qu'il sera grisé dans le formulaire.
Texte à gauche	[booléens] Place le texte du libellé à gauche de la case à cocher.
Afficher	[booléens et méthodes] Spécifie le formulaire dans lequel les options booléennes ou le bouton de méthode sont affichés. Pour les booléens, vous pouvez choisir une des options suivantes : • Case à cocher • Colonne de boutons radio • Ligne de boutons radio Pour les méthodes, vous pouvez choisir dans une sélection d'icônes standard ou **Texte**, qui imprime le texte spécifié dans la zone **Libellé** sur le bouton.
Largeur/ Hauteur	[zones d'espacement] Spécifie la largeur et la hauteur, en pixels, de la zone d'espacement.

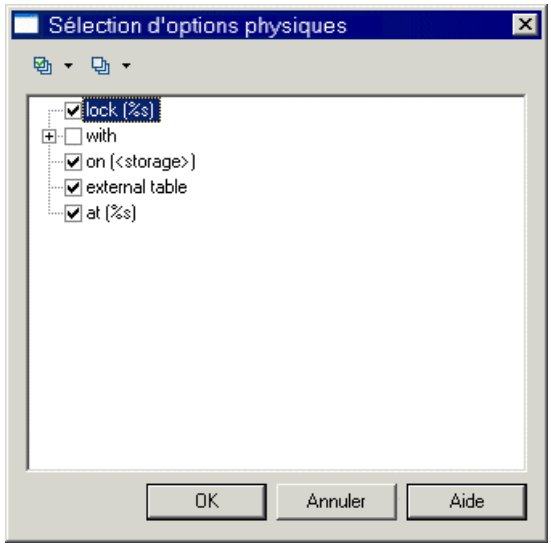
Ajout d'options physiques de SGBD dans vos formulaires

Nombre des SGBD utilisent des *options physiques* comme faisant partie de la définition de leurs objets. Les options physiques les plus couramment utilisées sont affichées sur un formulaire, **Physical Options (Common)**, défini sous la métaclasse appropriée. Vous pouvez éditer ce formulaire, ou bien ajouter des options physiques dans vos propres formulaires.

Remarque : PowerAMC affiche toutes les options disponibles pour un objet (définies dans la catégorie `Script/Objects/`*`objet`*`/Options`) sur l'onglet **Options physiques** (voir *Options physiques* à la page 240).

Pour qu'une option physique soit affichée sur un formulaire, elle doit être associée avec le type `option physique`.

1. Pointez sur la métaclasse, cliquez le bouton droit de la souris, puis sélectionnez **Nouvel attribut étendu à partir des options physiques** afin d'afficher la boîte de dialogue Sélection d'options physiques :



Remarque : Cette boîte de dialogue est vide si aucune option physique n'est définie dans `Script/Objects/objet/Options`.

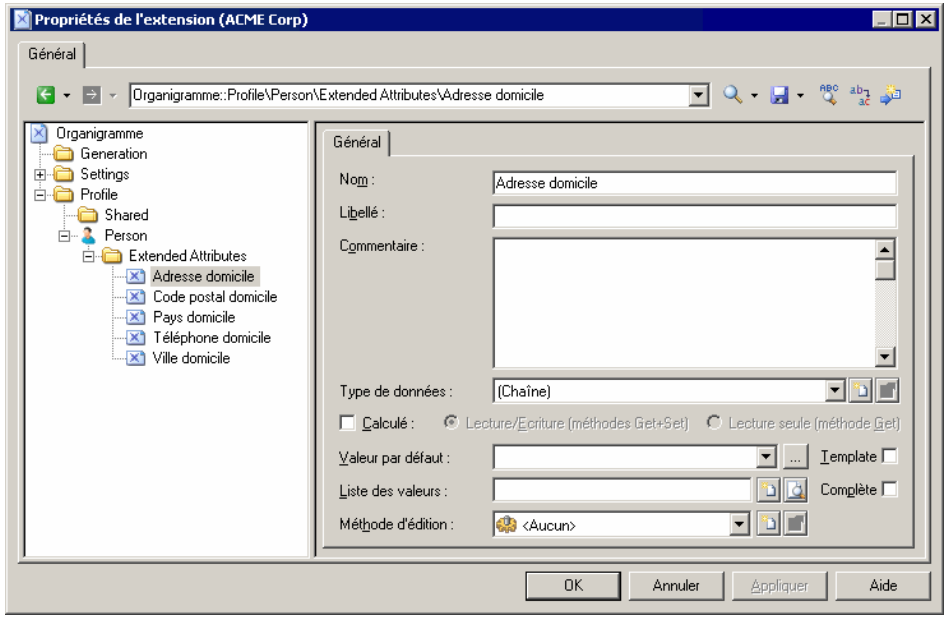
2. Sélectionnez l'option physique requise, puis cliquez sur **OK** pour créer un attribut étendu qui lui soit associé.
3. Spécifiez les éventuelles propriétés appropriées.
4. Sélectionnez le formulaire dans lequel vous souhaitez insérer l'option physique, puis cliquez sur l'outil Ajouter un attribut afin de l'ajouter comme contrôle (voir *Ajout d'attributs étendus et d'autres contrôles dans votre formulaire* à la page 80).

Remarque : Pour changer l'option physique associée avec un attribut étendu, cliquez sur le bouton Points de suspension à droite de la zone **Option physique** dans la feuille de propriétés de l'attribut étendu.

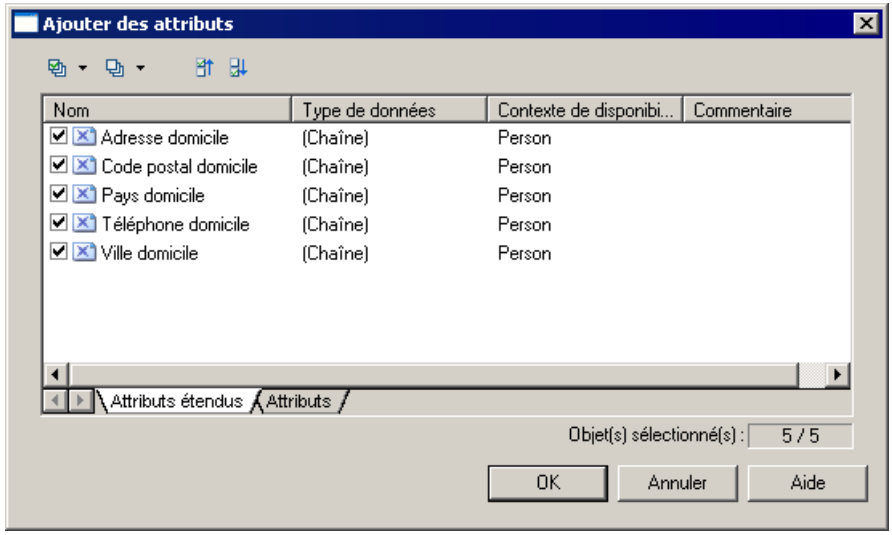
Exemple : Création d'un onglet de feuille de propriétés

Dans cet exemple, nous allons créer un nouvel onglet de propriétés pour la métaclasse de MAE Person, ce afin d'afficher des attributs étendus que nous définissons afin d'y stocker des informations personnelles.

1. Créez un nouveau fichier d'extension (voir *Création d'un fichier d'extension* à la page 26) dans un MAE, ajoutez la métaclasse `Person` (voir *Ajout d'une métaclasse dans un profil* à la page 51), et définissez cinq attributs étendus (voir *Création d'un attribut étendu* à la page 64) afin de contenir les coordonnées personnelles :

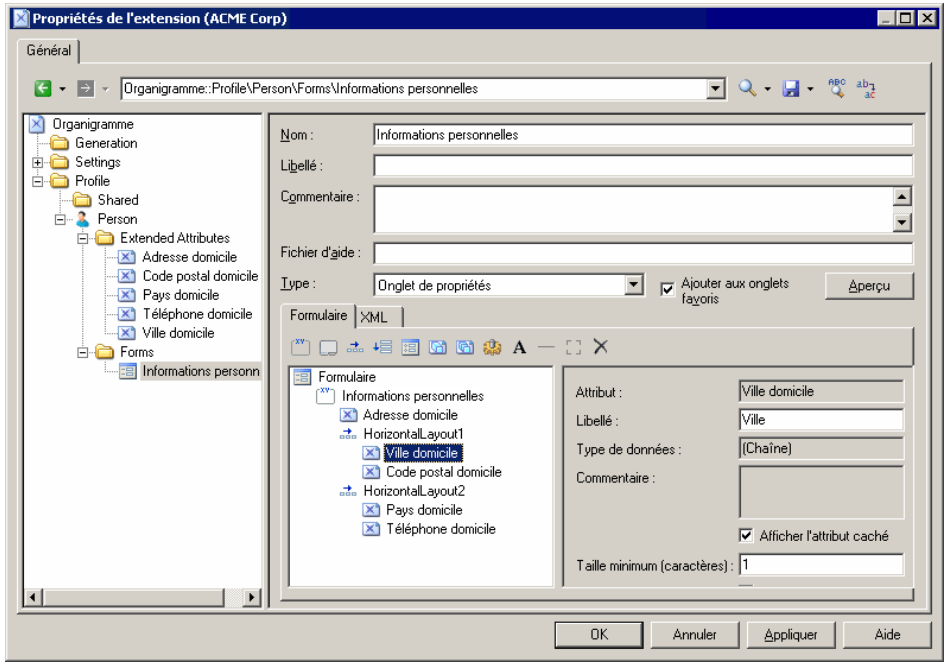


2. Pointez sur la métaclasse `Person`, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau** > **Formulaire**, saisissez `Informations personnelles` dans la zone **Nom**, sélectionnez `Onglet de propriétés` dans la liste **Type**, puis cliquez sur **Ajouter un attribut** afin de sélectionner tous les nouveaux attributs étendus à inclure dans le formulaire :

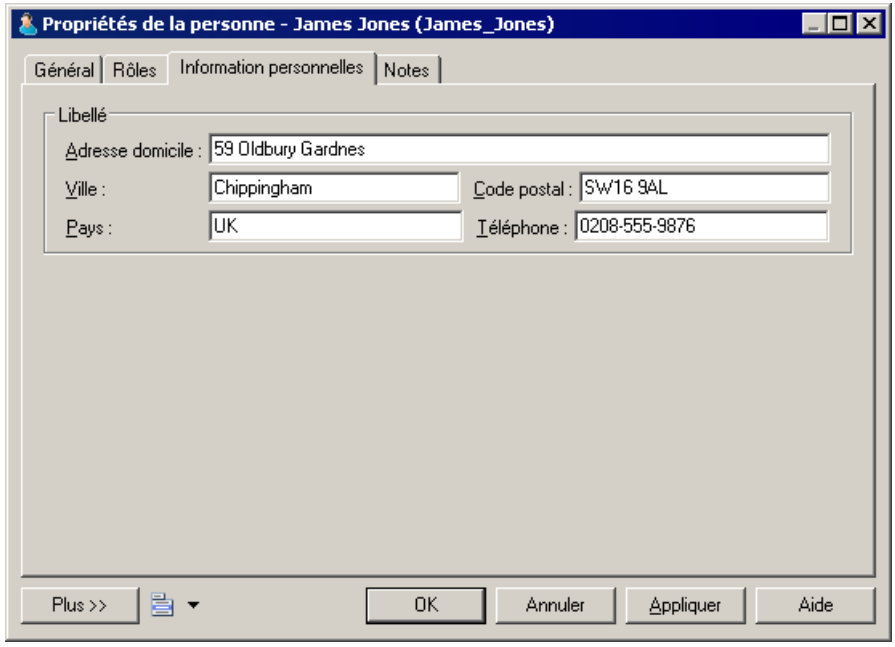


3. Cliquez sur **OK** afin d'ajouter les attributs dans le formulaire, puis réorganisez-les dans une zone de groupe, en utilisant des dispositions horizontales afin de les aligner. La zone

Libellé permet d'utiliser une formulation plus brève que celle du nom par défaut de l'attribut :



4. Cliquez sur **OK** pour enregistrer vos changements et revenir dans le modèle. Lorsque vous affichez ensuite la feuille de propriétés d'une personne, le nouvel onglet **Informations personnelles** est disponible et contient les attributs étendus :

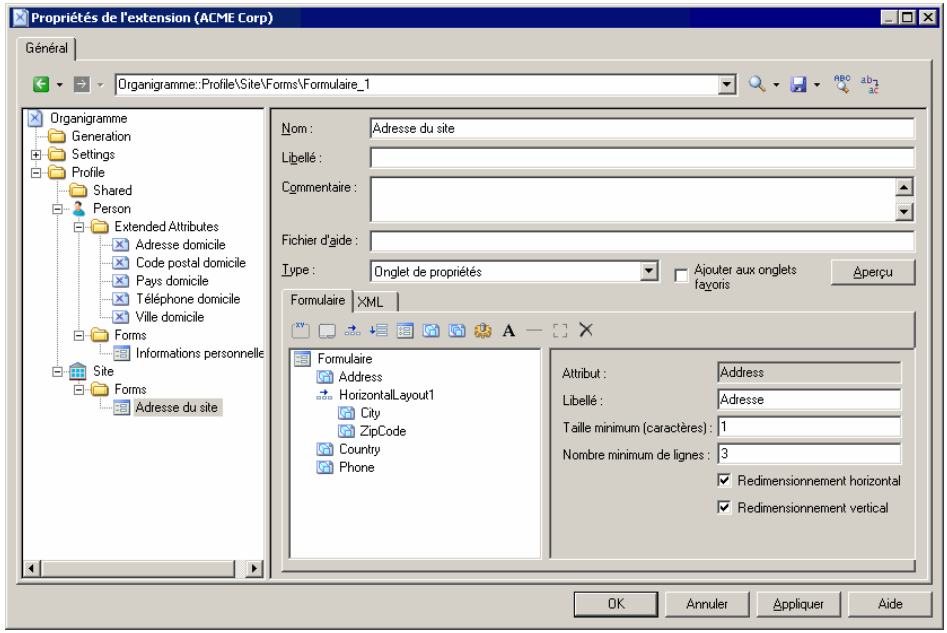


Exemple : Inclusion d'un formulaire dans un autre formulaire

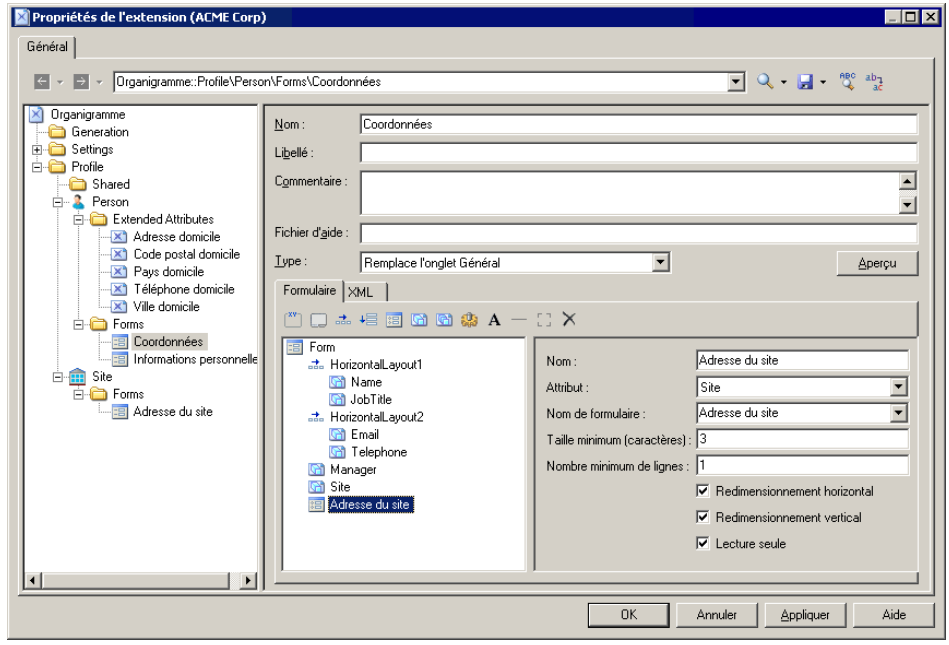
Dans cet exemple, nous allons remplacer l'onglet Général de la métaclasse Person d'un MAE par un formulaire qui inclut des propriétés provenant de la personne ainsi que du site auquel elle est affectée, ce afin d'inclure un formulaire défini sur la métaclasse Site comme contrôle en lecture seule défini sur la métaclasse Person.

Cet exemple utilise le fichier d'extension créé dans *Exemple : Création d'un onglet de feuille de propriétés* à la page 86.

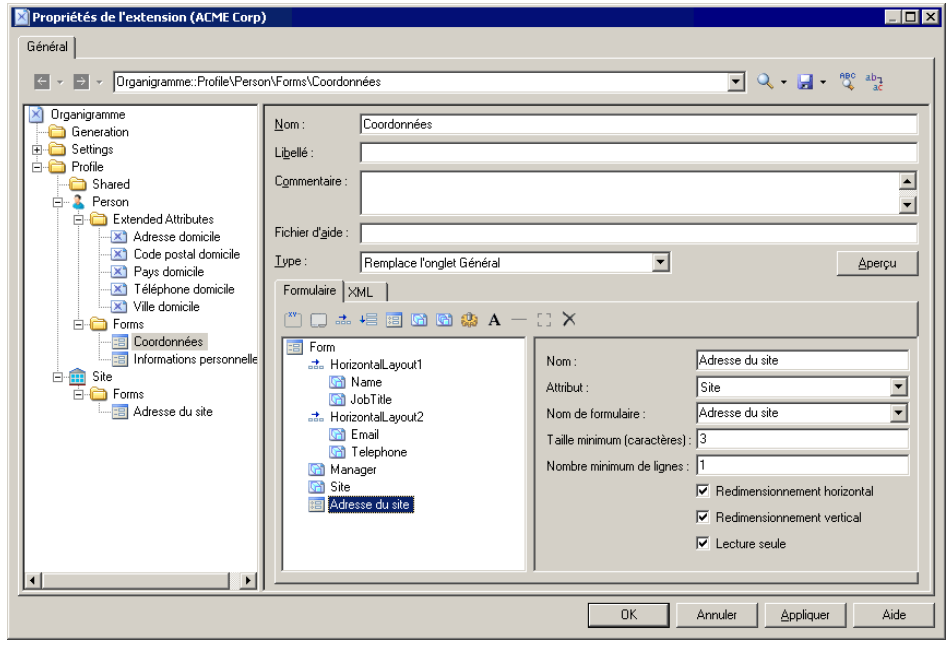
1. Ajoutez la métaclasse `Site` et créez un formulaire appelé `Adresse du site`. Sélectionnez `Onglet de propriétés` dans la liste **Type** puis décochez la case **Ajouter aux onglets favoris** (car nous ne souhaitons pas voir ce formulaire, qui duplique des propriétés de site standard, s'afficher dans les feuilles de propriétés de site).
2. Garnissez le formulaire à l'aide d'attributs standard pour afficher l'adresse complète du site :



3. Créez un formulaire sous la métaclasse `Person`, sélectionnez `Remplace l'onglet Général` dans la liste **Type**, puis changez le nom en `Coordonnées`.
4. Supprimez les attributs non souhaités de la liste, et réorganisez les attributs restants que vous souhaitez voir s'afficher, y compris l'attribut `Site` (qui est de type `Object`, et qui va permettre de récupérer les propriétés appropriées du formulaire de site associé) en utilisant des dispositions horizontales.
5. Cliquez sur l'outil **Inclure un autre formulaire**, sélectionnez `Site` dans la zone **Attribut**, puis saisissez `Adresse du site` dans les zones **Nom du formulaire** et **Nom**. Cochez la case **Lecture seule** afin d'empêcher l'édition du formulaire inclus à partir de la feuille de propriétés de la personne :



6. Cliquez sur **OK** pour enregistrer les extensions, puis revenez à votre modèle. La prochaine fois que vous affichez la feuille de propriétés d'une personne, l'onglet **Général** est remplacé par l'onglet **Coordonnées**, et si la personne est affectée à un site, les détails de l'adresse du site sont affichés en lecture seule dans la partie inférieure du formulaire :

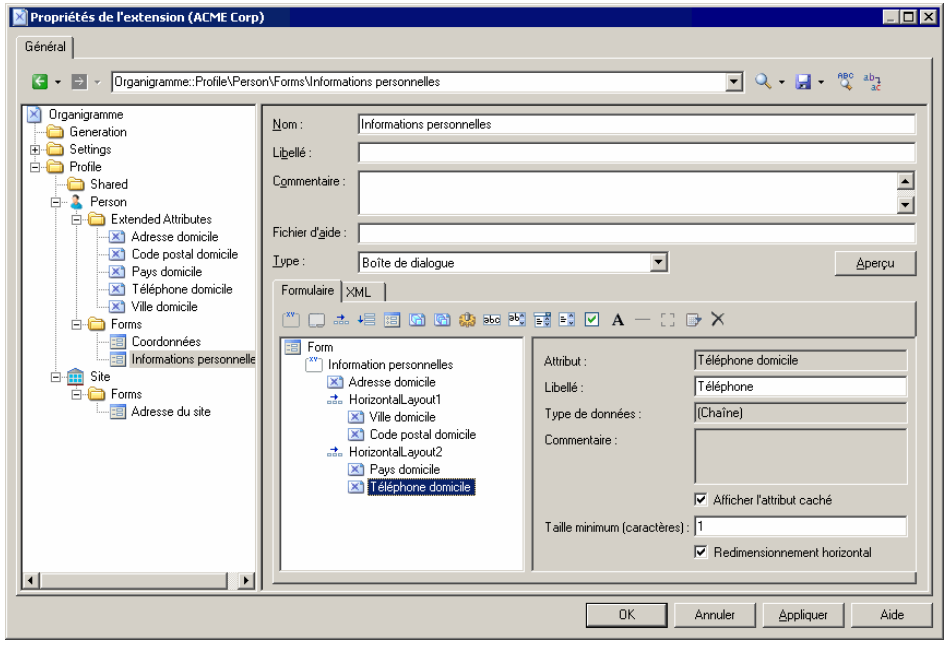


Exemple : Ouverture d'une boîte de dialogue à partir d'une feuille de propriétés

Dans cet exemple, nous allons ajouter un bouton à une feuille de propriétés, afin d'ouvrir une boîte de dialogue, vous permettant de saisir des informations personnelles supplémentaires pour une personne.

Cet exemple est basé sur le fichier d'extension développé dans *Exemple : Inclusion d'un formulaire dans un autre formulaire* à la page 89.

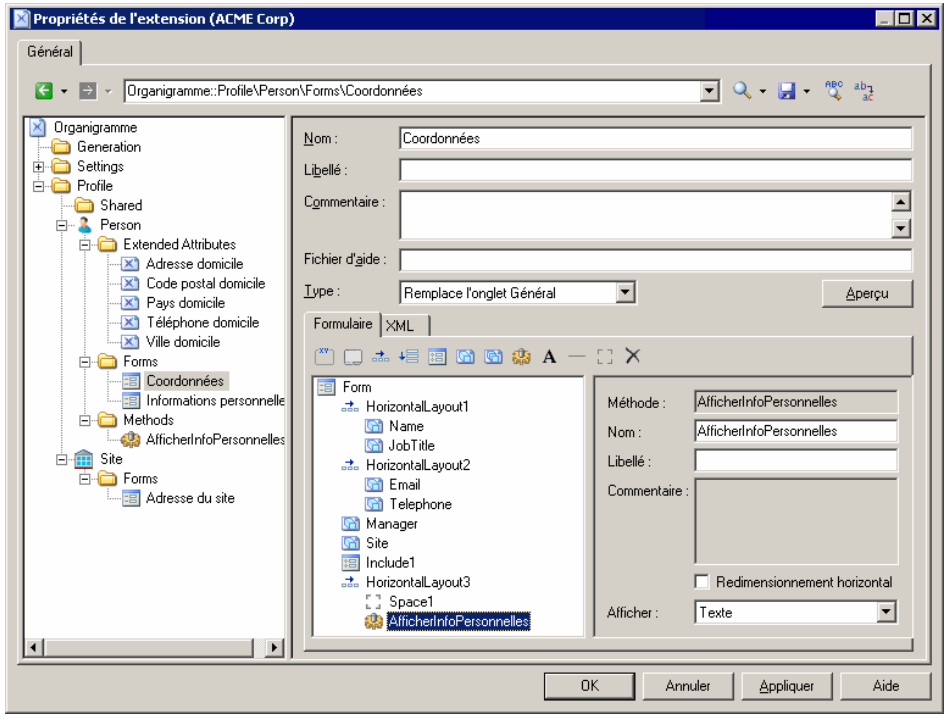
1. Affichez le formulaire `Informations personnelles` sous la métaclasse `Person`, puis sélectionnez `Boîte de dialogue` dans la zone **Type**, afin de la transformer d'onglet de feuille de propriétés en boîte de dialogue indépendante :



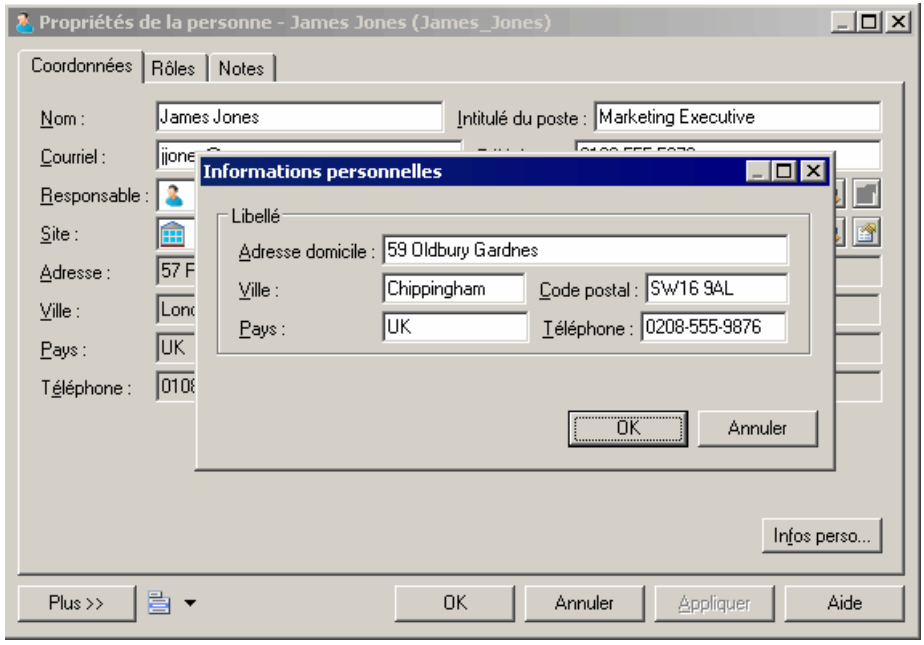
2. Pointez sur la métaclasse Person, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau** > **Méthode**. Saisissez le nom `AfficherInfoPersonnelles`, cliquez sur l'onglet **Script de méthode** puis saisissez le script suivant :

```
Sub %Method%(obj)
 ' Afficher une boite personnalisée pour les attributs étendus
avancés
 Dim dlg
 Set dlg = obj.CreateCustomDialog("%CurrentTargetCode
%.Informations personnelles")
 If not dlg is Nothing Then
  dlg.ShowDialog()
 End If
End Sub
```

3. Sélectionnez le formulaire `Coordonnées`, puis cliquez sur l'outil **Ajouter un bouton de méthode**, sélectionnez la méthode `AfficherInfoPersonnelles`, puis cliquez sur **OK** afin de l'ajouter dans le formulaire. J'utilise une disposition horizontale et une zone d'espacement afin d'aligner le bouton sur le bord droit du formulaire :



4. Saisissez `Infos perso...` dans la zone **Libellé**, puis cliquez sur **OK** pour enregistrer vos modifications et revenir au modèle. Dorénavant, lorsque vous affichez la feuille de propriétés d'une personne, l'onglet **Coordonnées** contient un bouton **Infos perso...** qui permet d'afficher la boîte de dialogue **Informations personnelles** :



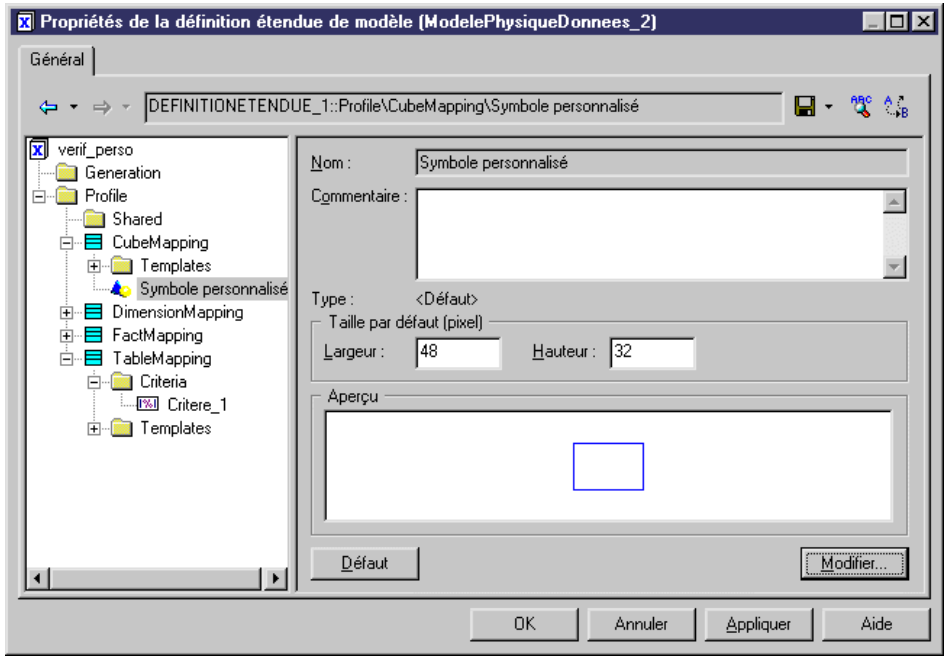
Symboles personnalisés (Profile)

Un symbole personnalisé permet de modifier l'apparence des instances de la métaclasse, du stéréotype ou du critère.

Lorsque vous personnalisez le style d'un symbole de lien, par exemple une référence de MPD, les paramètres que vous sélectionnez dans la liste Style et dans la zone de groupe Flèche sur l'onglet Style de ligne remplacent celui que vous avez sélectionné dans la boîte de dialogue Préférences d'affichage. Ceci peut provoquer un manque de cohérence dans votre modèle. Pour éviter toute confusion et préserver la définition de votre modèle, vous devez utilisez l'attribut Notation dans la liste Style ou dans la zone de groupe Flèche. Cet attribut n'est disponible que dans le profil.

1. Pointez sur une métaclasse, un stéréotype ou un critère dans la catégorie Profile, puis sélectionnez **Nouveau > Symbole personnalisé**.

   Un nouveau symbole est créé sous la catégorie sélectionnée.



2. Spécifiez une **Largeur** et une **Hauteur** par défaut pour le symbole, puis cliquez sur le bouton **Modifier** pour afficher la boîte de dialogue Format de symbole, et définissez les propriétés appropriées sur les différents onglets.

   Pour plus d'informations sur la boîte de dialogue Format de symbole (ainsi que sur les options de symboles personnalisés permettant de contrôle les options de format par défaut pour le symbole, et si les utilisateurs peuvent les éditer, onglet par onglet), voir *Guide des fonctionnalités générales > L'interface de PowerAMC > Diagrammes, matrices et symboles > Symboles > Propriétés d'un format de symbole*.

3. Cliquez sur **OK** pour revenir à l'Editeur de ressources, dans lequel vous pouvez visualiser vos changements dans la zone Aperçu.
4. Cliquez sur **Appliquer** pour enregistrer vos modifications.

Vérifications personnalisées (Profile)

Les vérifications personnalisées sont des vérifications de modèles, écrites en VBScript, qui permettent de vérifier que les objets de vos modèles sont correctement définis. Les vérifications personnalisées sont répertoriées avec les vérifications standard dans la boîte de dialogue Vérification de modèle.

Pour plus d'informations sur l'utilisation de VBScript, voir *Chapitre 7, Pilotage de PowerAMC à l'aide de scripts* à la page 349.

Propriétés d'une vérification personnalisée

Vous spécifiez les propriétés pour une vérification personnalisée en sélectionnant l'entrée correspondante dans l'Editeur de ressources.

Paramètre	Description
Nom	Nom de la vérification personnalisée. Ce nom s'affiche sous la catégorie d'objet sélectionnée dans la boîte de dialogue **Paramètres de vérification de modèle**. Ce nom est également utilisé (concaténé) dans le nom de la fonction de vérification, afin d'identifier cette dernière de façon unique.
Commentaire	Informations supplémentaires relatives à la vérification personnalisée.
Message d'aide	Texte affiché dans la zone de message qui s'affiche lorsque l'utilisateur sélectionne **Aide** dans le menu contextuel de la vérification personnalisée dans la boîte de dialogue **Paramètres de vérification de modèle**.
Message de résultats	Texte affiché dans la fenêtre **Liste de résultats** lors de l'exécution des vérifications.
Sévérité par défaut	Permet de définir si la vérification personnalisée correspond à une erreur (problème majeur qui interrompt la génération) ou un avertissement (problème mineur ou simple recommandation).
Exécuter la vérification par défaut	Permet de vous assurer que cette vérification personnalisée est sélectionnée par défaut dans la boîte de dialogue **Paramètres de vérification de modèle**.
Exécuter la correction automatique	Permet d'autoriser la correction automatique pour la vérification personnalisée.
Exécuter la correction automatique par défaut	Permet de vous assurer que la correction automatique pour cette vérification personnalisée est exécutée par défaut.
Script de vérification	Cet onglet contient le script de vérification. Voir *Définition du script d'une vérification personnalisée* à la page 98.
Script de correction automatique	Cet onglet contient le script de vérification. Voir *Définition du script d'une correction automatique* à la page 99.
Script global	Cet onglet permet de partager les fonctions de bibliothèque et les attributs statiques dans le fichier de ressources. Voir *Utilisation du script global* à la page 101.

Définition du script d'une vérification personnalisée

Cette section s'applique également à la définition du script pour une méthode personnalisée, une collection calculée, un gestionnaire d'événement ou une transformation.

Vous pouvez saisir le type d'une vérification personnalisée dans l'onglet Script de vérification des propriétés de vérification personnalisée. Par défaut, l'onglet Script de vérification affiche les éléments de script suivants :

- %Check% est le nom de la fonction, il est passé sur le paramètre obj. Il est affiché sous forme de variable, cette variable étant une concaténation de nom du fichier de ressource, du nom de la métaclasse courante, du nom du stéréotype ou critère ainsi que du nom de la vérification elle-même défini dans l'onglet Général. Si l'un de ces noms comporte un espace, ce dernier est remplacé par un trait de soulignement
- Commentaire expliquant le comportement attendu du script
- La ligne de valeur de résultat qui indique si la vérification a réussi (true) ou non (false)

Dans Sybase AS IQ, vous devez créer des vérifications supplémentaires sur les index afin de vérifier leurs colonnes. La vérification personnalisée que vous allez créer vérifie si les index de type HG, HNG, CMP ou LF sont liés aux colonnes ayant comme type de données VARCHAR et si la longueur est supérieure à 255.

1. Pointez sur une métaclasse, un stéréotype ou un critère sous la catégorie Profile, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau > Vérification personnalisée**.
2. Cliquez sur l'onglet Script de vérification dans la feuille de propriétés de la vérification personnalisée pour afficher l'éditeur de script.

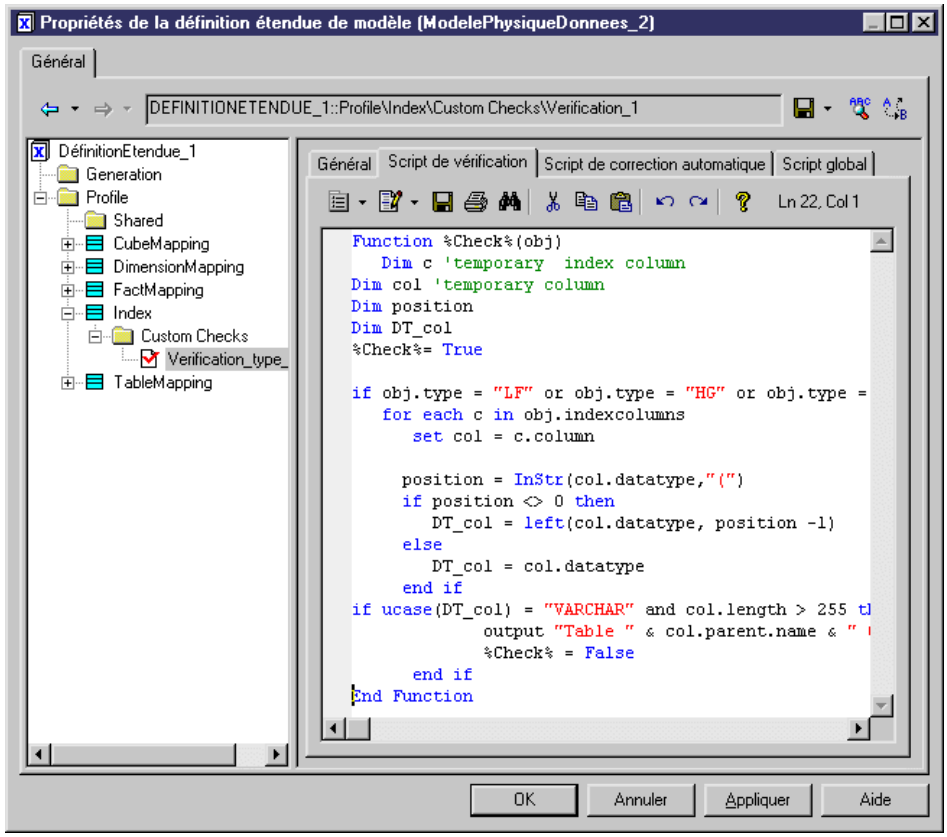
   Par défaut, la fonction est déclarée au début du script. Vous ne devez pas modifier cette ligne.
3. Saisissez un commentaire après la déclaration de la fonction afin de documenter la vérification personnalisée, et déclarez les différentes variables utilisées dans le script.

```
Dim c 'temporary  index column
Dim col 'temporary column
Dim position
Dim DT_col
```

4. Saisissez le corps de la fonction.

```
%Check%= True

if obj.type = "LF" or obj.type = "HG" or obj.type = "CMP" or
obj.type ="HNG" then
 for each c in obj.indexcolumns
  set col = c.column

   position = InStr(col.datatype,"(")
   if position <> 0 then
    DT_col = left(col.datatype, position -1)
   else
    DT_col = col.datatype
```

```
   end if
if ucase(DT_col) = "VARCHAR" and col.length > 255 then
    output "Table " & col.parent.name & " Column " & col.name & " :
Data type is not compatible with Index " & obj.name & " type " &
obj.type
    %Check% = False
  end if
```



5. Cliquez sur Appliquer pour enregistrer les modifications.

Définition du script d'une correction automatique

Si la vérification personnalisée que vous avez définie prend en charge la correction automatique, vous pouvez saisir le corps de cette fonction dans l'onglet Script de correction automatique de la feuille de propriétés de vérification personnalisée.

La correction automatique est visible dans la boîte de dialogue Paramètres de vérification de modèle, elle est sélectionnée par défaut si vous cochez la case Exécuter la correction automatique par défaut dans l'onglet Général de la feuille de propriétés de la vérification personnalisée.

Par défaut, l'onglet Script de correction automatique affiche les éléments de script suivants :

- %Fix% est le nom de la fonction, il est passé sur le paramètre obj. Il est affiché sous forme de variable, cette variable étant une concaténation de nom du fichier de ressource, du nom de la métaclasse courante, du nom du stéréotype ou critère ainsi que du nom de la correction. Si l'un de ces noms comporte un espace, ce dernier est remplacé par un trait de soulignement
- La variable *outmsg* est un paramètre de la fonction de correction. Vous devez spécifier le message de correction qui va s'afficher lors de l'exécution du script de correction
- La ligne de valeur de résultat qui indique si la correction a fonctionné

Nous allons reprendre l'exemple de la section Définition du script d'une vérification personnalisée afin de définir un script de correction automatique qui supprime de l'index les colonnes ayant un type de données incorrect.

1. Cliquez sur l'onglet Script de correction automatique dans la feuille de propriétés de vérification personnalisée.

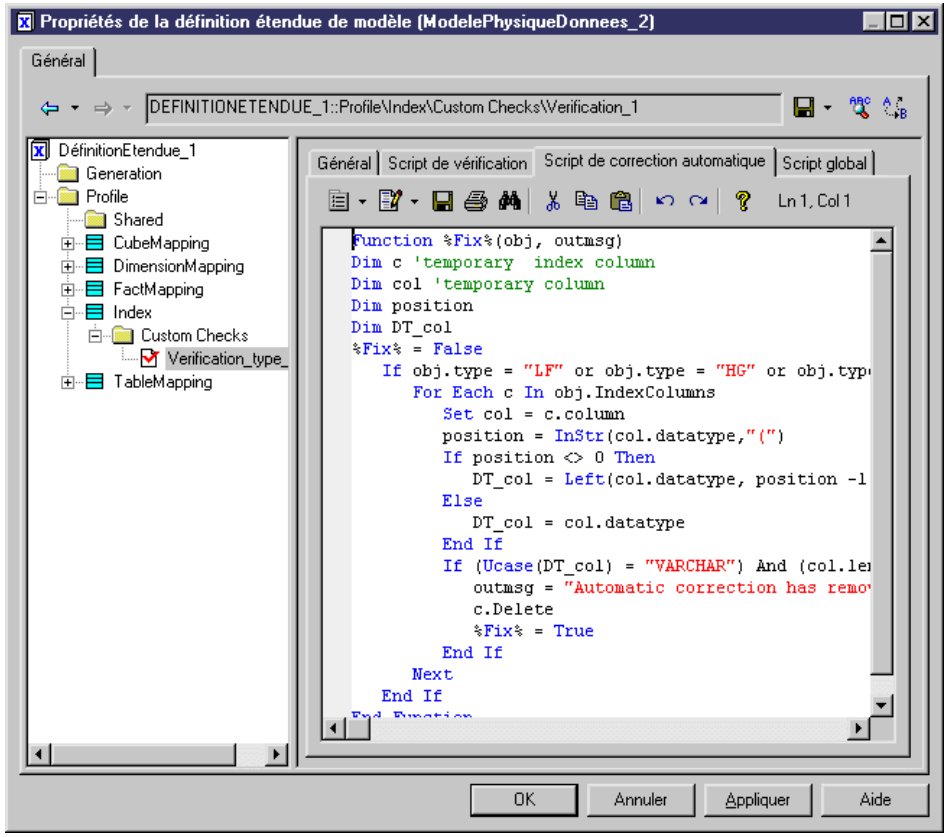
   Par défaut, la fonction est déclarée au début du script. Vous ne devez pas modifier cette ligne.

2. Saisissez un commentaire après la déclaration de la fonction afin de documenter la vérification personnalisée, puis déclarez les différentes variables utilisées dans le script.

```
Dim c 'temporary  index column
Dim col 'temporary column
Dim position
Dim DT_col
```

3. Saisissez le corps de la fonction.

```
%Fix% = False
 If obj.type = "LF" or obj.type = "HG" or obj.type = "CMP" or
obj.type ="HNG" Then
  For Each c In obj.IndexColumns
   Set col = c.column
   position = InStr(col.datatype,"(")
   If position <> 0 Then
    DT_col = Left(col.datatype, position -1)
   Else
    DT_col = col.datatype
   End If
   If (Ucase(DT_col) = "VARCHAR") And (col.length > 255) Then
    outmsg = "Automatic correction has removed column " & col.Name
& " from index."
    c.Delete
    %Fix% = True
   End If
  Next
 End If
```



4. Cliquez sur Appliquer pour enregistrer les modifications.

Utilisation du script global

Cette section s'applique également à la définition du script pour une méthode personnalisée, une collection calculée, un gestionnaire d'événement ou une transformation.

L'onglet Script global est utilisée pour stocker les fonctions et attributs statiques qui peuvent être réutilisés entre les différentes fonctions. Cette page affiche une bibliothèque de sous-fonctions disponibles.

Exemple

Dans l'exemple Sybase AS IQ, vous pouvez utiliser une fonction appelée DataTypeBase qui extrait le type de données d'un élément afin de mieux l'analyser.

Cette fonction est définie comme suit :

```
Function DataTypeBase(datatype)
   Dim position
   position = InStr(datatype, "(")
   If position <> 0 Then
```

```
      DataTypeBase = Ucase(Left(datatype, position -1))
   Else
      DataTypeBase = Ucase(datatype)
   End If
End Function
```

Dans ce cas, cette fonction a seulement besoin d'être référencée dans les scripts de vérification et de correction automatique :

```
Function %Check%(obj)
Dim c   'temporary  index column
   Dim col 'temporary column
   Dim position
   %Check%= True
   If obj.type = "LF" or obj.type = "HG" or obj.type = "CMP" or
obj.type ="HNG" then
      For Each c In obj.IndexColumns
         Set col = c.column
        If (DataTypeBase(col.datatype) = "VARCHAR") And (col.length >
255) Then
          Output "Table " & col.parent.name & " Column " & col.name &
" : Data type is not compatible with Index " & obj.name & " type " &
obj.type
            %Check% = False
         End If
      Next
   End If
End Function
```

Variables globales

Vous pouvez également déclarer des *variables globales* dans le script global. Ces variables sont réinitialisées chaque fois que vous exécutez la vérification personnalisée.

Exécution de vérifications personnalisées et dépannage d'erreurs VBScript

Toutes les vérifications personnalisées définies dans les fichiers de ressources attachés au modèle sont fusionnées et toutes les fonctions de toutes les vérifications personnalisées sont ajoutées dans un seul et même script. La boîte de dialogue Paramètres de vérification de modèle affiche toutes les vérifications personnalisées définies sur les métaclasses, les stéréotypes et les critères sous les catégories correspondantes.

Si des erreurs sont détectées lors de la vérification personnalisée, les actions suivantes sont proposées à l'utilisateur :

Bouton	Action
Ignorer	Permet de sauter le script problématique et de reprendre la vérification.
Ignorer tout	Permet de sauter tous les scripts problématiques et de reprendre le processus avec les vérifications standard.
Annuler	Arrête la vérification du modèle.

Bouton	Action
Déboguer	Arrête la vérification du modèle, ouvre l'éditeur de ressources et indique sur quelle ligne se trouve le problème. Vous pouvez corriger les erreurs et redémarrer la vérification du modèle.

Gestionnaires d'événement (Profile)

Un gestionnaire d'événement peut lancer automatiquement un script VBScript lorsqu'un événement se produit sur un objet. Vous pouvez associer un gestionnaire d'événement à une métaclasse ou à un stéréotype ; les critères ne prennent pas en charge les gestionnaires d'événement.

Les gestionnaires d'événement suivants peuvent être définis sur des éléments dans la catégorie Profile :

Gestionnaire d'événement	Description

CanCreate	[tous les objets] Met en oeuvre une règle de validation de création afin d'empêcher la création d'objets dans un contexte invalide. Par exemple, dans un MPM pour ebXML, un processus ayant le stéréotype BusinessTransaction ne peut être créé que sous un processus ayant le stéréotype BinaryCollaboration. Le script du gestionnaire d'événement CanCreate associé au processus ayant comme stéréotype BusinessTransaction se présente comme suit :
	`Function %CanCreate%(parent)`
	`  if parent is Nothing or`
	`  parent.IsKindOf(PdBpm.Cls_Process) then`
	`  %CanCreate% = False`
	`  else`
	`  %CanCreate% = True`
	`  end if`
	`End Function`
	Si ce gestionnaire d'événement est défini sur un *stéréotype* et que la valeur de retour de la fonction est True, vous pouvez utiliser l'outil personnalisé pour créer l'objet stéréotypé. Dans le cas contraire, l'outil personnalisé n'est pas disponible, et la liste Stéréotype n'affiche pas le stéréotype correspondant. S'il est défini sur une *métaclasse* et qu'il renvoie True, vous pouvez alors créer l'objet à partir de la Boîte à outils, à partir de l'Explorateur d'objets ou bien dans une liste.
	Remarquez que si vous importez un modèle ou procédez à son reverse engineering, les fonctions CanCreate sont ignorées car elles pourraient modifier le modèle et y créer des divergences par rapport au modèle d'origine.

Gestionnaire d'événement	Description

Initialize	[tous les objets] Utilisé pour instancier des objets avec des templates prédéfinis. Par exemple, dans un MPM, un processus BusinessTransaction doit être un processus composite avec un sous-graphe prédéfini. Le script du gestionnaire d'événement Initialize associé au stéréotype de processus BusinessTransaction contient toutes les fonctions nécessaires pour la création du sous-graphe. L'extrait de script suivant est un sous-ensemble du gestionnaire d'événement Initialize pour un processus BusinessTransaction.
	`...`
	`' Search for an existing requesting activity symbol`
	`Dim ReqSym`
	`Set ReqSym = Nothing`
	`If Not ReqBizAct is Nothing Then`
	`If ReqBizAct.Symbols.Count > 0 Then`
	`Set ReqSym = ReqBizAct.Symbols.Item(0)`
	`End If`
	`End If`
	`' Create a requesting activity if not found`
	`If ReqBizAct is Nothing Then`
	`Set ReqBizAct = BizTrans.Processes.CreateNew`
	`ReqBizAct.Stereotype = "RequestingBusinessActivity"`
	`ReqBizAct.Name = "Request"`
	`End If`
	`...`
	Si le gestionnaire d'événement Initialize est défini sur un *stéréotype* et que la valeur de retour de la fonction est True, le script d'initialisation sera lancé chaque fois que le stéréotype est affecté, soit à l'aide d'un outil personnalisé dans la Boîte à outils, soit à partir de la feuille de propriétés d'objet. S'il est défini sur une *métaclasse* et si la valeur de retour de la fonction est True, le script d'initialisation sera lancé lorsque vous créez un nouvel objet à partir de la Boîte à outils, à partir de l'Explorateur d'objets, dans une liste ou dans une feuille de propriétés.
	Si le gestionnaire d'événement Initialize est défini sur la métaclasse *model* et renvoie True, le script d'initialisation est lancé lorsque vous affectez une cible (SGBD, langage objet, langage de processus, langage de schéma) au modèle au moment de la création, lorsque vous changez la cible du modèle ou lorsque vous affectez une extension à ce modèle.

Gestionnaire d'événement	Description

Validate	[tous les objets] Exécuté lorsque vous changez d'onglet ou que vous cliquez sur **OK** ou **Appliquer** dans une feuille de propriétés d'objet. Utilisé pour valider les modifications des propriétés d'objet et mettre en oeuvre des modifications en cascade.
	
	Vous pouvez définir un message d'erreur qui apparaît si la condition n'est pas satisfaite. Pour ce faire, renseignez la variable message et définissez la variable %Validate% à False.
	
	Dans l'exemple suivant, le gestionnaire d'événement Validate vérifie qu'un commentaire est ajouté dans la définition d'un objet lorsque l'utilisateur valide dans la feuille de propriétés. Un message s'affiche pour expliquer le problème.
	
	`Function %Validate%(obj, ByRef message)`
	`  if obj.comment = "" then`
	`   %Validate% = False`
	`   message = "Comment cannot be empty"`
	`  else`
	`   %Validate% = True`
	`  end if`
	`End Function`

CanLinkKind	[objets de lien uniquement] Exécuté lorsque vous créez un lien à l'aide d'un outil de la Boîte à outils ou que vous modifier les extrémités d'un lien dans une feuille de propriétés. Utiliser pour limiter le type et le stéréotype des objets pouvant être liés.
	
	Ce gestionnaire d'événement a deux paramètres d'entrée : l'extrémité source et l'extrémité de destination du lien. Vous pouvez également utiliser les paramètres sourceStereotype et destinationStereotype qui sont facultatifs et qui permettent d'effectuer des contrôles supplémentaires sur les stéréotypes.
	
	Dans l'exemple suivant, la source du lien étendu doit être un début :
	
	`Function %CanLinkKind%(sourceKind,`
	`  sourceStereotype, destinationKind,`
	`  destinationStereotype)`
	` if sourceKind = cls_Start Then`
	` %CanLinkKind% = True`
	` end if`
	`End Function`

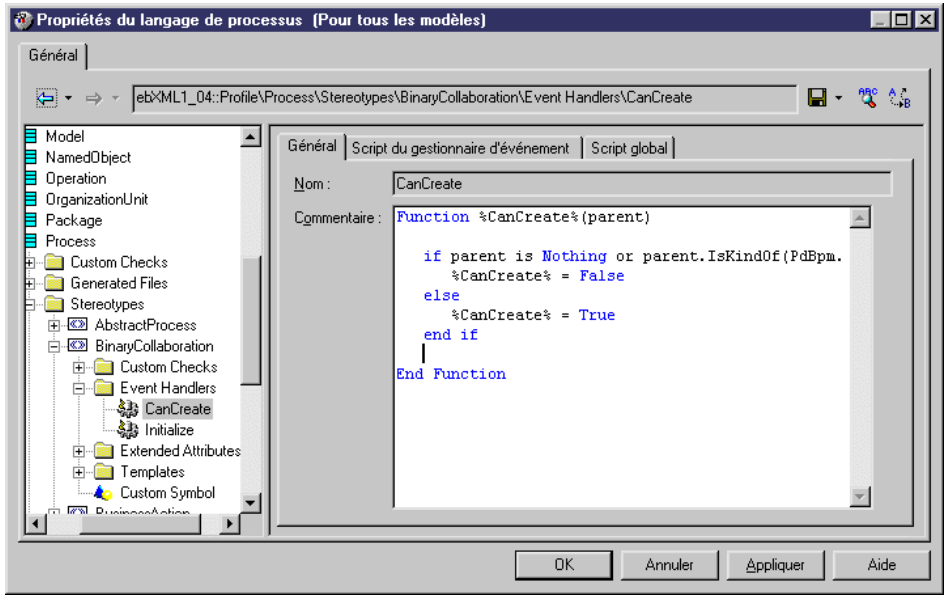
OnModelOpen	[modèles uniquement] Exécuté immédiatement après l'ouverture d'un modèle.
OnModelSave	[modèles uniquement] Exécuté immédiatement avant l'enregistrement d'un modèle.
OnModelClose	[modèles uniquement] Exécuté immédiatement avant la fermeture d'un modèle.

Gestionnaire d'événement	Description
OnLanguageChangeRequest	[modèles uniquement] Exécuté immédiatement avant le changement de SGBD ou de fichier de définition de langage du modèle. Si ce gestionnaire d'événement renvoie false, le changement de langage est annulée.
OnLanguageChanging	[modèles uniquement] Exécuté immédiatement après le changement de SGBD ou de fichier de définition de langage du modèle, mais avant que les transformations ne soient appliquées aux objets afin de les rendre conformes à la nouvelle définition de langage.
OnLanguageChanged	[modèles uniquement] Exécuté immédiatement après le changement de SGBD ou de fichier de définition de langage du modèle et la transformation des objets.
OnNewFromTemplate	[modèles uniquement] Exécuté immédiatement après la création d'un modèle ou d'un projet à partir d'un templatre de modèle ou de projet.
BeforeDatabaseGenerate	[MPD uniquement] Exécuté immédiatement avant la génération d'une base de données.
AfterDatabaseGenerate	[MPD uniquement] Exécuté immédiatement après la génération d'une base de données.
BeforeDatabaseReverseEngineer	[MPD uniquement] Exécuté immédiatement avant le reverse-engineering d'une base de données.
AfterDatabaseReverseEngineer	[MPD uniquement] Exécuté immédiatement après le reverse-engineering d'une base de données.
GetEstimatedSize	[MPD uniquement] Exécute le mécansime d'estimation de la taille de base de données. Pour obtenir des informations détaillées, voir *Modification du mécanisme d'estimation de taille de base de données* à la page 235.

Ajout d'un gestionnaire d'événement à une métaclasse ou à un stéréotype

Vous pouvez créer un gestionnaire d'événement dans un profil.

1. Pointez sur une métaclasse ou sur un stéréotype, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau > Gestionnaire d'événement** afin d'afficher une boîte de sélection qui répertorie les gestionnaires d'événement disponibles.
2. Sélectionnez un ou plusieurs gestionnaires d'événement, puis cliquez sur OK pour les ajouter à la métaclasse.
3. Cliquez sur le gestionnaire d'événement dans l'arborescence, puis spécifiez un nom et un commentaire.
4. Cliquez sur l'onglet Script du gestionnaire d'événement puis saisissez votre script :



5. Cliquez sur Appliquer pour enregistre vos modifications.

Propriétés d'un gestionnaire d'événement

Vous spécifiez les propriétés d'un gestionnaire d'événement en sélectionnant l'entrée correspondante dans l'Editeur de ressources.

Propriété	Description
Nom	Spécifie le nom du gestionnaire d'événement.
Commentaire	Fournit une description du gestionnaire d'événement.
Script du gestionnaire d'événement	Ce onglet spécifie le code VBScript qui est exécuté lorsque l'événement se produit. Notez que vous ne pouvez pas utiliser d'instructions telles que as msgbox ou input box pour afficher une boîte de dialogue dans la fonction de gestionnaire d'événement.
Script global	Ce onglet peut être utilisé pour partager des fonctions de bibliothèque et attributs statiques dans le fichier de ressources. Pour plus d'informations sur la définition d'un script et sur l'utilisation de l'onglet **Script global**, voir *Définition du script d'une vérification personnalisée* à la page 98 et *Utilisation du script global* à la page 101.

Méthodes (Profile)

Les méthodes permet d'effectuer des actions sur les objets.

Elles sont rédigées en VBScript, et sont appelées par d'autres composants du profil, tels que les commandes de menu (voir *Menus (Profile)* à la page 110) ou les boutons de formulaires (voir *Formulaires (Profile)* à la page 78).

La méthode exemple suivante, créée dans la métaclasse Class, convertit les classes en interfaces. Elle copie les propriétés et opérations de base des classes, supprime la classe (pour éviter tout problème d'espace de noms), et crée la nouvelle interface.

Notez que le script ne gère pas d'autres propriétés de classe, ni l'affichage d'interface, mais une méthode peut être utilisée pour lancer une boîte de dialogue personnalisée afin de demander à l'utilisateur final d'interagir avant d'effectuer une action (voir *Exemple : Création d'une boîte de dialogue affichée depuis une commande de menu* à la page 112).

```
Sub %Mthd%(obj)
 ' Convertit la classe en interface

 ' Copie les propriétés de base de la classe
 Dim Folder, Intf, ClassName, ClassCode
 Set Folder = obj.Parent
 Set Intf = Folder.Interfaces.CreateNew
 ClassName = obj.Name
 ClassCode = obj.Code
 Intf.Comment = obj.Comment

 ' Copie les opérations de la classe
 Dim Op
 For Each Op In obj.Operations
  ' ...
  Output Op.Name
 Next

 ' Détruit la classe
 obj.Delete

 ' Renomme l'interface avec le nom enregistré
 Intf.Name = ClassName
 Intf.Code = ClassCode
End Sub
```

Pour plus d'informations sur l'utilisation de VBScript dans PowerAMC, voir *Chapitre 7, Pilotage de PowerAMC à l'aide de scripts* à la page 349.

Création d'une méthode

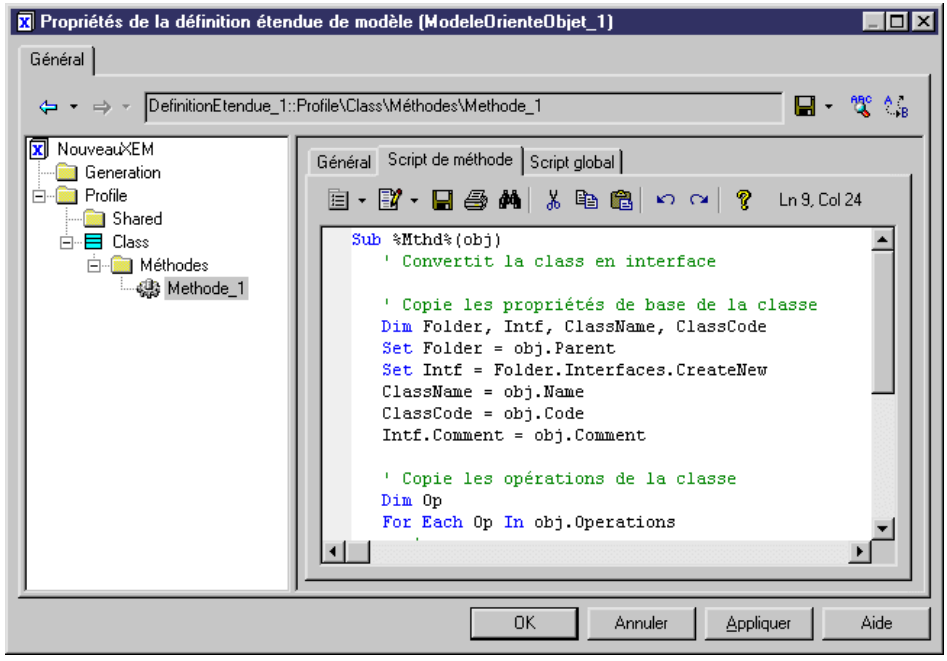
Vous pouvez créer une méthode dans un profil.

1. Pointez sur une métaclasse, un stéréotype ou un critère, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau** > **Méthode**.
2. Saisissez un nom et un commentaire pour la méthode.
3. Cliquez sur l'onglet Script de la méthode, puis saisissez le script. Le cas échéant, vous pouvez réutiliser les fonctions stockées dans l'onglet Script global.

   Par défaut, cet onglet contient le squelette de script suivant :

   ```
   Sub %Method%(obj)
     ' Implement your method on <obj> here
   End Sub
   ```

   %Method% est une concaténation du nom du fichier de ressource, du nom de la métaclasse courante, du nom du stéréotype ou critère, ainsi que du nom de la méthode elle-même dans l'onglet Général. Si l'un de ces noms contient un espace, ce dernier est remplacé par un tiret bas.



4. Cliquez sur Appliquer.

Propriétés d'une méthode

Vous spécifiez les propriétés pour une méthode en sélectionnant l'entrée correspondante dans l'Editeur de ressources.

Propriété	Description
Nom	Nom de la méthode qui identifie un script.
Commentaire	Informations supplémentaires relatives à la méthode.

L'onglet **Script de méthode** contient le corps de la fonction de méthode.

L'onglet **Script global** est utilisé pour partager les fonctions de bibliothèques et attributs statiques dans le fichier de ressource. Cet onglet est partagé avec les gestionnaires d'événement et les transformations.

Vous pouvez déclarer des *variables globales* sur cet onglet, vous devez savoir que ces dernières ne sont pas réinitialisées chaque fois que la collection est calculée, et conservent leur valeur jusqu'à ce que vous modifiiez le fichier de ressources, ou jusqu'à la fermeture de PowerAMC. Cette caractéristique peut s'avérer une source d'erreurs, tout particulièrement lorsque les variables font référence à des objets qui peuvent être modifiés, voir supprimés. Assurez-vous de bien réinitialiser la variable globale si vous ne souhaitez pas conserver la valeur d'une exécution précédente.

Pour plus d'informations sur la définition d'un script et sur l'utilisation de l'onglet **Script global**, voir *Définition du script d'une vérification personnalisée* à la page 98 et *Utilisation du script global* à la page 101.

Menus (Profile)

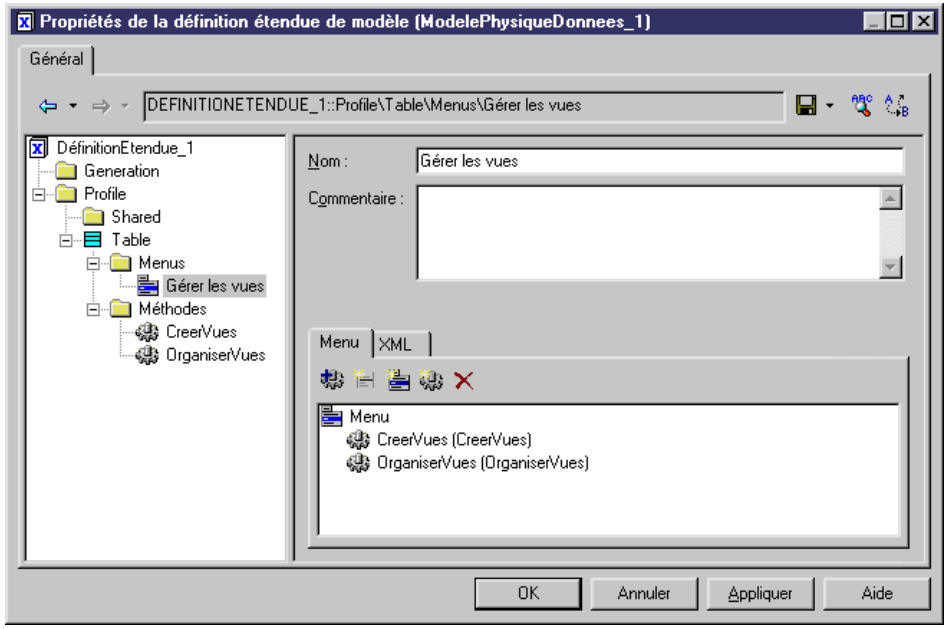
Vous pouvez ajouter des menus dans l'interface de PowerAMC et les remplir avec des commandes qui appellent des fonctions de méthode ou des transformations.

Pour plus d'informations sur les méthodes et transformations, voir *Méthodes (Profile)* à la page 108 et *Transformations et profils de transformation (Profile)* à la page 121.

Les menus peuvent être ajoutées dans les menus Fichiers, Outils et Aide de PowerAMC lorsqu'ils sont définis sur la métaclasse Model ou Diagram, ou bien sur le menu contextuel de symboles de diagramme ou d'éléments de l'Explorateur d'objets. Les menus définis dans une métaclasse parent sont hérités par ses enfants. Par exemple, vous pouvez généraliser un menu contextuel en le définissant sur une métaclasse parent telles que BaseObject.

1. Pointez sur une métaclasse, un stéréotype ou un critère, cliquez le bouton droit de la souris puis sélectionnez **Nouveau > Menu**.
2. Saisissez un nom et un commentaire (ainsi, dans le cas des métaclasses Model ou Diagram, qu'un emplacement).

3. Utilisez les outils du sous-onglet **Menu** pour créer les éléments de votre menu (voir *Ajout de commandes et autres éléments dans votre menu* à la page 112).



4. Cliquez sur **Appliquer** pour enregistrer vos modifications.

Propriétés d'un menu

Vous spécifiez les propriétés pour une menu en sélectionnant l'entrée correspondante dans l'Editeur de ressources.

Propriété	Description
Nom	Nom Spécifie le nom du menu. Ce nom ne s'affiche pas dans le menu.
Commentaire	Fournit une description du menu.
Emplacement	[métaclasses Model et Diagram uniquement] Spécifie l'emplacement auquel le menu sera affiché. Vous pouvez choisir : • Menu Fichier>Exporter • Menu Aide • Menu contextuel d'un objet • Menu Outils Les menus créés sur d'autres métaclasses ne sont disponibles que dans le menu contextuel, et la zone **Emplacement** n'est alors pas disponible.

Propriété	Description
Onglet Menu	Ce sous-onglet met à votre disposition des outils pour ajouter des éléments à votre menu (voir *Ajout de commandes et autres éléments dans votre menu* à la page 112).
XML	Ce sous-onglet affiche le code XML généré à partir du sous-onglet **Menu**.

Ajout de commandes et autres éléments dans votre menu

Vous pouvez utiliser les outils suivants dans l'onglet Menu pour créer des menus et commandes :

Vous pouvez modifier l'ordre des éléments dans le menu par glisser-déposer. Pour placer une commande dans un sous-menu, faites-la glisser dans ce sous-menu.

Outil	Fonction
	Affiche une boîte de dialogue de sélection permettant de sélectionner une ou plusieurs méthodes ou transformations afin de créer des commandes associées. Cette liste est limitée aux méthodes et transformations définies dans la métaclasse courante et ses métaclasses parent Lorsque vous cliquez sur OK, chaque méthode sélectionnée est ajoutée dans votre menu avec le format : <Libellé> (<Nom de méthode/transformation>). L'élément *libellé* est le nom de la commande tel qui s'affichera dans le menu. Vous pouvez définir une touche de raccourci en ajoutant une perluète immédiatement avant la touche de raccourci, ce caractère s'affiche souligné dans le menu. Les méthodes ou transformations associées aux commandes de menu ne sont pas synchronisées avec celles définies dans une métaclasse. Si vous modifiez le nom ou le script d'une méthode ou transformation, vous devez utiliser l'outil de recherche pour localiser et mettre à jour toutes les commandes qui utilisent cette méthode ou transformation.
	Créer un séparateur - Crée un séparateur sous l'élément sélectionné.
	Créer un sous-menu - Crée un sous-menu sous l'élément sélectionné.
	Supprime l'élément sélectionné dans l'arborescence de menu.

Exemple : Ouverture d'une boîte de dialogue à partir d'un menu

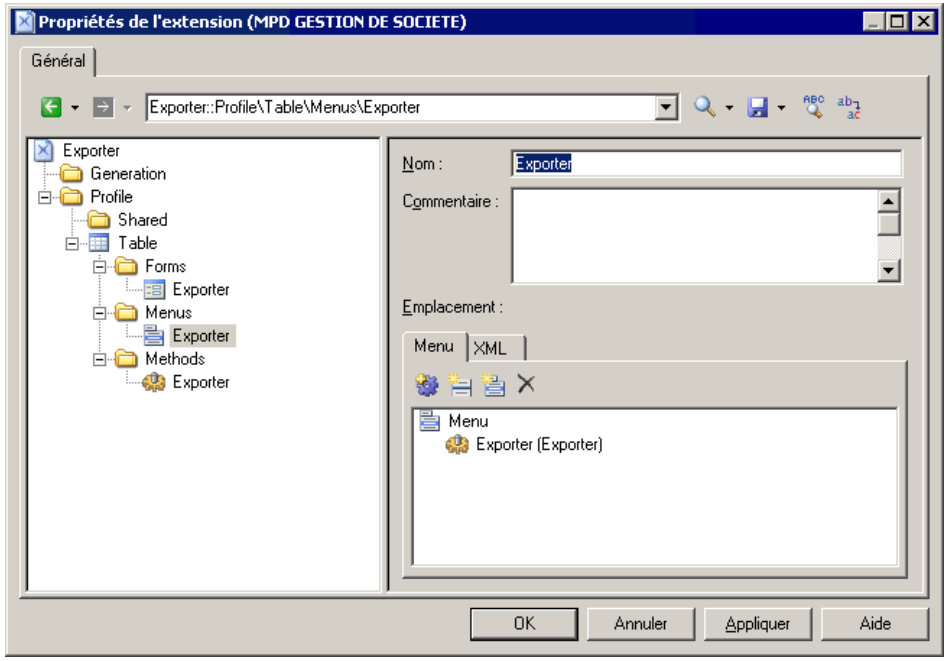
Dans cet exemple nous allons créer une commande de menu permettant d'exporter des propriétés d'objet dans un fichier XML par l'intermédiaire d'une boîte de dialogue.

1. Créez un nouveau fichier d'extension (voir *Création d'un fichier d'extension* à la page 26) dans un MPD et ajoutez la métaclasse `Table` (voir *Ajout d'une métaclasse dans un profil* à la page 51).

2. Pointez sur la métaclasse `Table`, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau** > **Formulaire**. Saisissez `Exporter` dans la zone **Nom**, puis sélectionnez `Boîte de dialogue` dans la liste **Type**.
3. Cliquez sur l'outil **Ajouter une zone d'édition** pour ajouter un contrôle de zone d'édition, puis nommez-le `Nom de fichier`.
4. Pointez sur la métaclasse `Table`, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau** > **Méthode**. Saisissez `Exporter` dans la zone **Nom**, cliquez sur l'onglet **Script de méthode**, puis saisissez le code suivant :

```
Sub %Method%(obj)
' Exporter un objet dans un fichier
' Créer une boîte de dialogue pour saisir le nom du fichier
Dim dlg
Set dlg = obj.CreateCustomDialog("%CurrentTargetCode%.Exporter")
    If not dlg is Nothing Then
    ' Initialiser la valeur du contrôle de nom de fichier
    dlg.SetValue "Nom de fichier", "c:\temp\MonFichier.xml"

    ' Show dialog
    If dlg.ShowDialog() Then
       ' Récupérer la valeur du client pour le contrôle de nom de
fichier
        Dim filename
        filename = dlg.GetValue("Nom de fichier")

        ' Traiter l'algorithme d'exportation...
        ' (Code d'exportation non inclus dans cet exemple)

        Output "Exportation de l'objet  " + obj.Name + " vers le
fichier " + filename
    End If

    ' Libérer la boîte de dialogue
    dlg.Delete
    Set dlg = Nothing
End If
End Sub
```

5. Pointez sur la métaclasse `Table`, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau** > **Menu**. Saisissez `Exporter` dans la zone **Nom**, puis cliquez sur l'outil **Ajouter une commande** et sélectionnez la méthode `Exporter` :



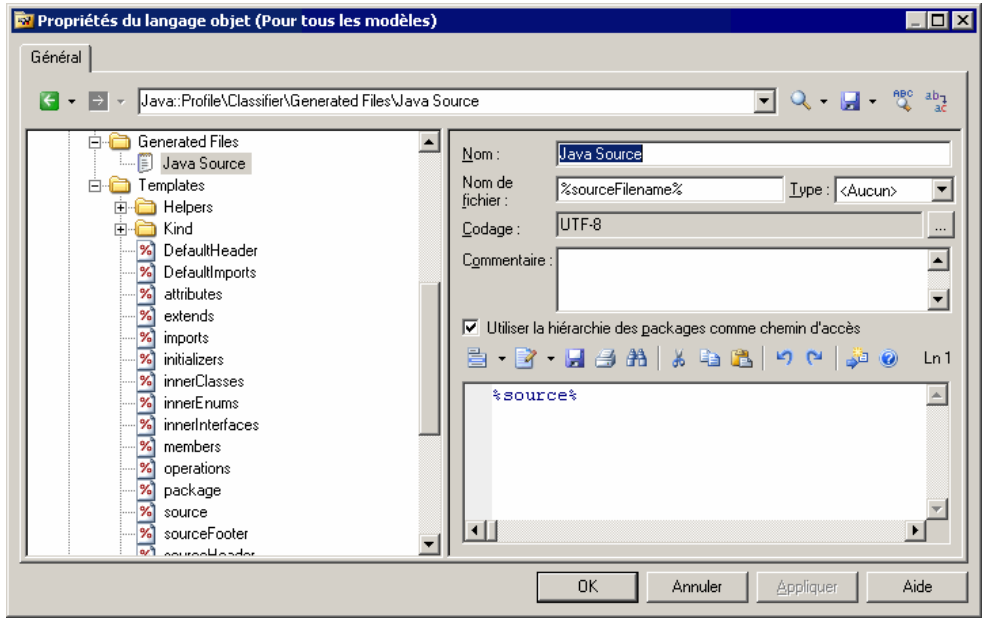
6. Cliquez sur **OK** pour enregistrer vos modifications et revenir au modèle. Lorsque vous pointez sur une table dans le diagramme ou dans l'Explorateur d'objets et cliquez le bouton droit de la souris, la commande **Exporter** est disponible dans le menu contextuel.

Templates et fichiers générés (Profile)

Vous pouvez définir des templates et des fichiers générés pour les métaclasses, stéréotypes et critères. Si un template s'applique à toutes les métaclasses, vous devez le créer dans la catégorie Shared.

Le langage de génération par template (GTL, Generation Template Language) permet de générer des fichier pour les métaclasses et le scripting (voir *Chapitre 5, Personnalisation de la génération à l'aide du langage de génération par template* à la page 285). Vous créez un template à l'aide du GTL, en utilisant des variables qui permettent d'accéder aux propriétés de l'objet courant ou de n'importe quel autre objet dans le modèle.

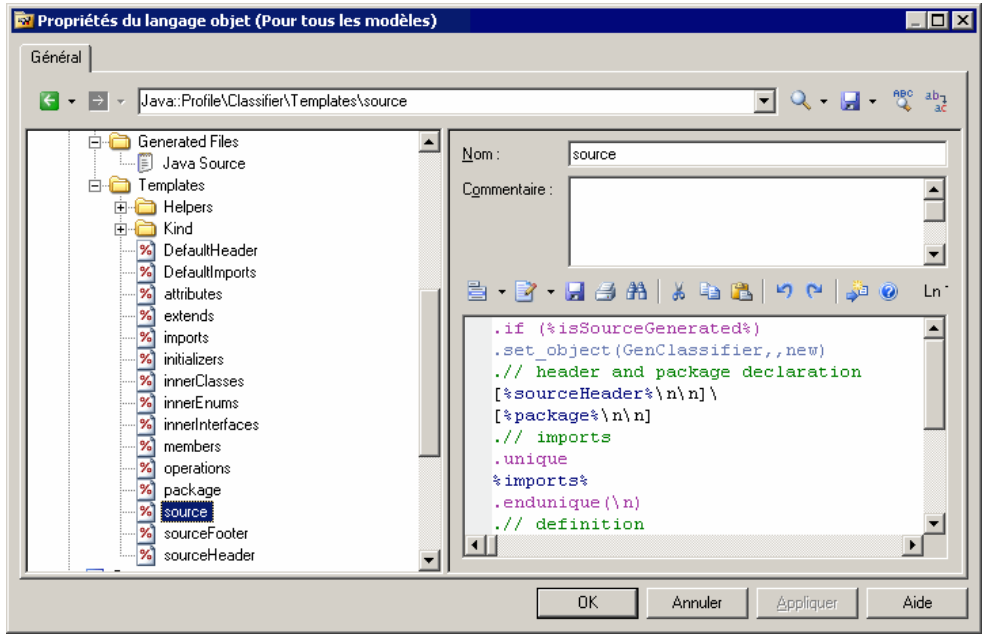
Dans l'exemple suivant, la catégorie Generated Files pour les classificateurs contient une entrée `Java Source`, qui contient elle-même une référence au template `%source%`. Lorsque le fichier est généré pour un classificateur donné ou pour les instances d'un classificateur avec un stéréotype ou critère sélectionné, il a le nom spécifié dans la zone **Nom de fichier**, et aura le contenu généré par ce template :



Remarque : Si vous placez votre curseur entre les signes pourcent qui encadrent le nom de ce template ou de tout autre template et appuyez sur **F12**, vous passez directement au template référencé ou, si plusieurs templates portent le même nom, une boîte de dialogue Résultats s'affiche pour vous permettre de sélectionner le template vers lequel vous souhaitez naviguer.

Le template référencé, `source`, contient du code de langage de génération par template qui permet de générer le contenu du fichier, y compris les références aux autres templates appelés :

- `%isSourceGenerated%`
- `%sourceHeader%`
- `%package%`
- `%imports%`



Création d'un template

Les templates peuvent être créés dans la catégorie Shared lorsque vous les appliquez à toutes les métaclasses. Ils peuvent également être créés au niveau de la métaclasse ou pour un stéréotype ou critère donné.

Remarque : Dans les version précédentes de PowerAMC, vous pouviez lier l'utilisation d'un template particulier à l'existence d'un stéréotype à l'aide de la syntaxe template name <<stereotype>>. Vous pouvez maintenant créer un template sous le stéréotype particulier dans le profil.

Vous pouvez utiliser l'outil Parcourir pour trouver tous les templates portant le même nom. Pour ce faire, ouvrez un template, placez le curseur sur un nom de template (entre les caractères %) et cliquez sur Parcourir (ou appuyez sur F12). Vous affichez ainsi une fenêtre Résultats qui répertorie tous les templates préfixés par le nom de leur métaclasse. Vous pouvez double-cliquer sur un template dans la fenêtre Résultats pour localiser sa définition dans l'éditeur de ressources.

1. Pointez sur une métaclasse, un stéréotype ou un critère, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau > Template**.
2. Saisissez un nom dans la zone Nom. Il est déconseillé d'utiliser des espaces dans les noms de template.
3. [facultatif] Saisissez un commentaire dans la zone Commentaire afin d'expliquer le rôle du template.

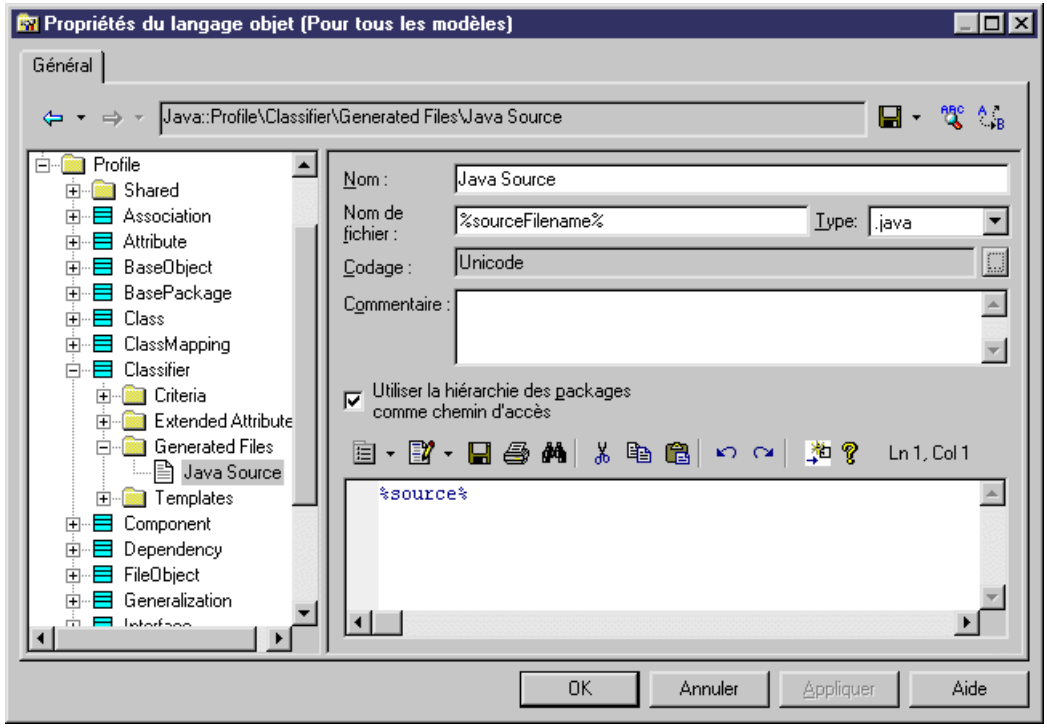
4. Saisissez le corps du template à l'aide du GTL dans la zone centrale.

Création d'un fichier généré

La catégorie Generated Files contient une entrée pour chaque type de fichier généré pour les métaclasses, stéréotypes ou critères. Seuls les fichiers définis pour les objets appartenant directement à une collection de modèle ou de package seront générés. Les sous-objets, tels que les attributs, les colonnes ou les paramètres, ne prennent pas en charge la génération de fichier, mais il peut être intéressant d'examiner le code généré pour ces sous-objet dans leur volet Aperçu.

Remarque : Si une extension attachée au modèle contient un nom de fichier généré identique à celui défini sur le fichier de ressource principal, c'est le fichier généré de l'extension qui sera généré.

1. Pointez sur une métaclasse, un stéréotype ou un critère, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau > Fichier généré**.
2. Saisissez un nom dans la zone Nom, saisissez un nom de fichier, et sélectionnez un type de fichier pour activer la coloration syntaxique.
3. Saisissez le template pour le contenu du fichier généré dans la zone de texte.



4. Cliquez sur Appliquer pour enregistrer vos modifications.

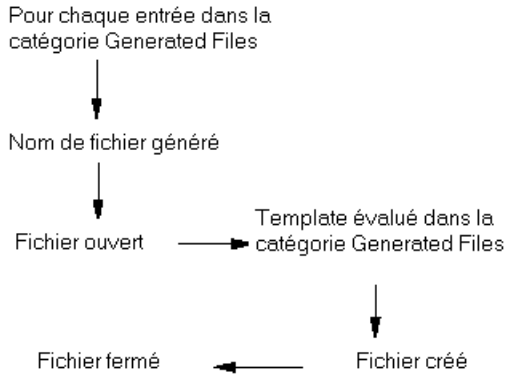
Les propriétés d'un fichier généré sont les suivantes :

Propriété	Description
Nom	Nom de l'entrée du fichier généré dans l'éditeur de ressources.
Nom de fichier	Spécifie le nom du fichier qui sera généré. Cette zone peut contenir des variables du langage de génération par template. Par exemple, pour générer un fichier XML avec le code de l'objet pour son nom, vous devez spécifier `%code%.xml`. Si vous laissez cette zone à vide, aucun fichier n'est généré, mais vous pouvez voir le code produit dans l'onglet **Aperçu** de la feuille de propriétés de l'objet. Si cette zone contient une extension reconnue, le code s'affiche avec l'éditeur de langage correspondant et la coloration syntaxique.
Type	Spécifie le type de fichier afin d'activer la coloration syntaxique appropriée pour la fenêtre d'aperçu.
Codage	Spécifie le format de codage du fichier généré. Cliquez sur l'outil Points de suspension à droite de la zone pour choisir un codage alternatif dans la boîte de dialogue Format de codage pour le texte en sortie, dans laquelle vous pouvez spécifier les options suivantes : • Codage - Format de codage pour le fichier généré • Annuler si perte de caractère - Spécifie que la génération doit être interrompue si des caractères ne sont pas identifiés et risquent d'être perdus dans le codage courant
Commentaire	Spécifie des informations supplémentaires relatives aux fichiers générés.
Utiliser la hiérarchie des packages comme chemin d'accès	Indique que la hiérarchie de packages doit être utilisée pour générer la hiérarchie des répertoires de fichiers.
Template du fichier généré (zone de texte)	Spécifie le template du fichier à générer. Vous pouvez spécifier le template directement ici ou faire référence à un template défini ailleurs dans le profil (voir *Chapitre 5, Personnalisation de la génération à l'aide du langage de génération par template* à la page 285).

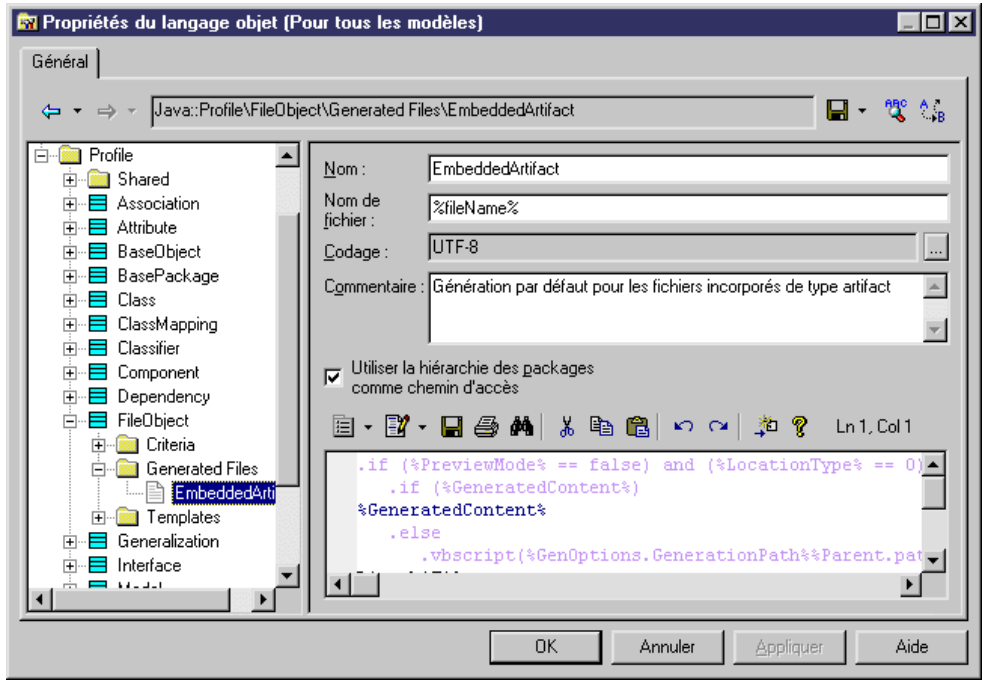
Exemples de fichiers générés

Un fichier est généré pour chaque entrée de la catégorie `Generated File` située sous une métaclasse, un stéréotype ou critère, à condition que la zone Nom de fichier ne soit pas laissée à blanc. Si aucun nom de fichier n'est spécifié, ou si seul un suffixe de nom de fichier est spécifié, le contenu du fichier généré peut tout de même être affiché dans l'onglet **Aperçu** de la feuille de propriétés de l'objet.

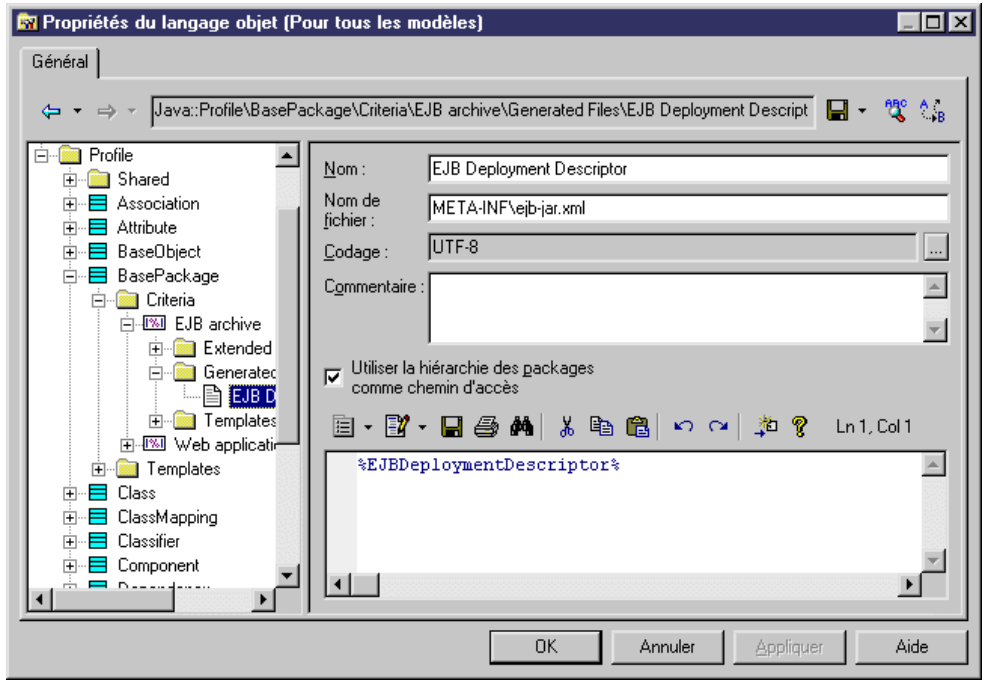
Le mécanisme de la génération de fichiers est le suivant pour chaque objet ayant une entrée Generated Files qui n'est pas vide :



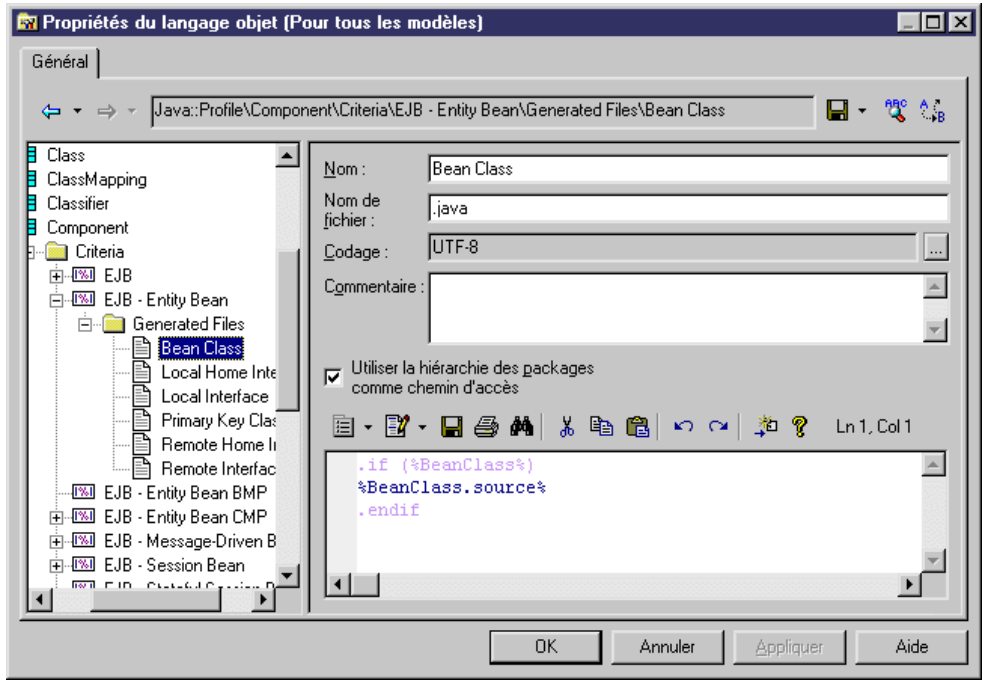
Dans l'exemple suivant, la catégorie `Generated Files` pour les objets fichier dans Java contient l'entrée `EmbeddedArtifact` qui s'applique à tous les fichiers incorporés de type Artifact à générer, et la zone **Nom de fichier** contient un template pour le nom du fichier à générer. La zone de texte située dans la partie inférieure affiche le code du template du fichier à générer :



Dans cet exemple, un fichier appelé ejb-jar.xml situé dans le dossier META-INF est généré :



Dans cet exemple, aucun fichier n'est généré car la zone **Nom de fichier** ne contient qu'un suffixe commençant par un . (point). Le contenu du fichier est uniquement disponible sur l'onglet **Aperçu** de la feuille de propriétés de composant (EJB - Entity Bean) :



Transformations et profils de transformation (Profile)

Une transformation définit un jeu d'actions à exécuter lors d'une génération ou sur demande. Vous définissez une transformation dans la catégorie Profile d'une extension sur une métaclasse, sur un stéréotype ou sur un critère.

Vous définissez une transformation lorsque vous avez devez :

- Modifier des objets dans un but particulier. Par exemple, vous pouvez créer une transformation dans un MOO qui convertit les classes <<control>> en composants.
- Modifier des objets en conservant la possibilité de revenir sur cette modification. Cette fonctionnalité peut s'avérer très utile dans le cas d'une ingénierie par va-et-vient. Supposez que vous généreriez un MPD à partir d'un MOO afin de créer une correspondance O/R. Si le MOO source contient des composants, vous pouvez créer des transformations qui convertissent des classes à partir de composants afin de générer facilement des tables dans un MPD et de les mettre en correspondance. Toutefois, lorsque vous mettez à jour le MOO source à partir du MPD généré, vous pouvez utiliser une autre transformation qui va automatiquement recréer les composants à partir des classes.

Les transformations peuvent être utilisées :

- Dans un profil de transformation (voir *Création d'un profil de transformation* à la page 125) lors de la génération de modèle, ou à la demande. Pour plus d'informations, voir

Guide des fonctionnalités générales > Liaison et synchronisation de modèles > Mise en correspondance d'objets.

- Sous forme de commande dans un menu défini par l'utilisateur (voir *Menus (Profile)* à la page 110)

Les transformations sont utilisées dans PowerAMC pour mettre en oeuvre une *Model Driven Architecture (MDA, architecture orientée modèle)*. MDA est une démarche de développement proposée par l'OMG (Object Management Group) et qui sépare la logique métiers d'une application des moyens technologiques utilisés pour la mettre en oeuvre. Le but étant d'améliorer l'intégration et l'interopérabilité des applications et ainsi réduire le temps et les efforts passés dans le développement et la maintenance de l'application.

Le développement MDA utilise la modélisation UML pour décrire une application à différents niveaux de détails, en commençant par la construction d'un modèle libre de toute plate-forme (*PIM,* platform-independent model) qui permet de modéliser la fonctionnalité et la logique métiers de base, puis en finissant par un modèle lié à une plate-forme spécifique (*PSM, platform-specific model*) qui inclut les technologies d'implémentation (comme CORBA, .NET, ou Java). Entre le PIM initial et le PSM final, il peut y avoir d'autres PIM ou PSM intermédiaires qui fournissent différents niveaux d'amélioration.

PowerAMC permet de créer le PIM initial et de l'améliorer progressivement au travers de différents modèles contenant chacun des niveaux d'implémentation et d'information technologiques toujours plus élevés. Vous pouvez définir des transformations qui vont générer une version plus aboutie d'un modèle, basé sur la plate-forme cible désirée. Lorsque des changements sont effectués sur le PIM, ils peuvent être répercutés en cascade dans ses modèles générés.

Les transformations peuvent également être utilisées pour appliquer des motifs de modélisation à des objets dans le modèle.

1. Pointez sur une métaclasse ou un stéréotype, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau > Transformation** dans le menu contextuel.
2. Saisissez les propriétés appropriées, incluant un script de transformation.

Propriétés d'une transformation

Une transformation a les propriétés suivantes :

Onglet Général

L'onglet Général contient les propriétés suivantes :

Propriété	Description
Nom	Nom de la transformation. Prenez soin de spécifier des noms explicites afin de les identifier plus facilement dans les listes de sélection.

Propriété	Description
Commentaire	Informations supplémentaires relatives à la transformation utilisées pour expliquer le script

Onglet Script de la transformation

L'onglet Script de la transformation contient les propriétés suivantes :

Nom	Définition
CopyObject	Duplique un objet existant et définir une source pour l'objet dupliqué. Paramètres : • source : objet à dupliquer • étiquette [facultatif] : identificateur Renvoie : Une copie du nouvel objet
SetSource	Définit l'objet source d'un objet généré. Il est recommandé de toujours définir l'objet source afin de garder trace de l'origine d'un objet généré. Paramètres : • source : objet source • cible : objet cible • étiquette [facultatif] : identifier Renvoie :
GetSource	Récupère l'objet source d'un objet généré. Paramètres : • cible : objet cible • étiquette [facultatif] : identifier Renvoie : Objet source
GetTarget	Récupérer l'objet cible d'un objet source Paramètres : • source : objet source • étiquette [facultatif] : identifier Renvoie : Objet cible

Puisqu'un objet source peut être transformé et avoir plusieurs cibles, vous pouvez rencontrer des problèmes pour identifier l'origine d'un objet, tout particulièrement dans la boîte de dialogue de fusion. Le mécanisme suivant est utilisé pour aider à identifier l'origine d'un objet :

- Si l'objet source est transformé en un seul objet, la transformation est utilisée comme identificateur interne de l'objet cible.
- Si l'objet source est transformé en plusieurs objets, vous pouvez définir une *balise* particulière afin d'identifier le résultat de la transformation. Vous devez spécifier uniquement des caractères alphanumériques, et il est recommandé d'utiliser une valeur "stable" comme un stéréotype, qui ne risque pas d'être modifiée lors de générations successives.

Par exemple, MOO1 contient la classe Customer, à laquelle vous appliquez un script de transformation pour créer un EJB. Deux nouvelles classes sont créées à partir de la classe source, une pour l'interface home, et l'autre pour l'interface remote. Dans le script de transformation, vous devez affecter une étiquette "home" pour l'interface home, et "remote" pour l'interface remote. L'étiquette s'affiche entre signes <> dans l'onglet Version pour un objet, en regard de l'objet source.

Dans l'exemple suivant, le mécanisme de l'étiquette est utilisé pour identifier les classes attachées à un composant :

```
 'setting tag for all classes attached to component
 for each Clss in myComponent.Classes
  if clss <> obj then
   trfm.SetSource obj,Clss," GenatedFromEJB"+ obj.name +"target"
+Clss.Name
   For each ope in clss.Operations
     if Ope.Name = Clss.Code Then 'then it is a constructor _Bean
operation
      trfm.SetSource obj,Ope," GenatedFromEJB"+ obj.name +"target"
+Ope.Name
     end if
     if Ope.Name = Clss.Name Then 'then it is a constructor operation
      trfm.SetSource obj,Ope," GenatedFromEJB"+ obj.name +"target"
+Ope.Name
     end if
     if Ope.name = "equals" Then 'then it is an equals operation and
should be tagged
      trfm.SetSource obj,Ope," GenatedFromEJB"+ obj.name +"target"
+Ope.Name
     end if
   next
   end if
 next
```

Le script de transformation ne requiert pas autant de vérifications que les scripts standards, qui imposent de vérifier le contenu d'un modèle pour éviter les erreurs, car les transformations sont toujours mises en oeuvre dans un modèle temporaire ne contenant aucun objet. Ce modèle temporaire sera ensuite fusionné avec le modèle cible de génération si l'option de conservation de modifications est activée lors de la mise à jour.

Si vous créez une transformation en utilisant un script existant, vous pouvez supprimer ces contrôles.

Les objets transformation interne ne s'affichent pas dans l'interface de PowerAMC. Ils sont créés comme objets temporaires et passés au script de sorte que l'utilisateur puisse accéder aux fonctions helper mais aussi pour enregistrer l'exécution d'une séquence de transformations afin de pouvoir les exécuter ultérieurement.

Les objets transformation interne sont préservés lorsque les transformations sont utilisées par la fonctionnalité Appliquer les transformations ou dans un menu. En effet, lorsque vous mettez à jour un modèle (le régénérez) dans lequel ce type de transformations a été exécuté, les transformations doivent être exécutées à nouveau dans le modèle source afin de préserver l'équivalence entre les modèles source et cible.

Par exemple, MCD1 contient l'entité A, vous générez un MOO à partir de MCD1 et la classe B est créée. Vous appliquez des transformations à la classe B dans MOO1, ce qui crée la classe C. Vous souhaitez ensuite régénérer MCD1 et mettre à jour MOO1 : la classe B sera générée à partir de l'entité A mais la classe C est manquante dans le modèle généré, ce qui risquerait de se manifester par une différence dans la boîte de dialogue de fusion. Toutefois, grâce aux objets transformation interne, les transformations qui ont été exécutées dans le MOO généré sont ré-exécutées et vous obtenez la classe C et les modèles à fusionner sont encore plus similaires qu'avant.

Onglets Script global et Dépendances

L'onglet Script global permet d'accéder aux définitions partagées par toutes les fonctions VBScript définies dans le profil, et l'onglet Dépendances répertorie les profils de transformation dans lesquels la transformation est utilisée.

Création d'un profil de transformation

Un profil de transformation est un groupe de transformations utilisé au cours d'une génération de modèle lorsque vous souhaitez appliquer des modifications aux objets dans les modèles source ou cible.

Vous définissez un profil de transformation dans la catégorie Transformation Profiles d'une extension (voir *Transformations et profils de transformation (Profile)* à la page 121). Chaque profil est identifié par le modèle dans lequel le fichier d'extension courant est défini, par un type de modèle, une famille et une sous-famille.

1. [si la catégorie Transformation Profiles est absente] Pointez sur le noeud racine, cliquez le bouton droit de la souris, sélectionnez Ajouter des transformations dans le menu contextuel, sélectionnez Transformation Profiles, puis cliquez sur OK pour créer ce dossier sous le noeud racine.
2. Pointez sur le dossier Transformation Profiles, cliquez le bouton droit de la souris, puis sélectionnez Nouveau dans le menu contextuel. Un nouveau profil de transformation est créé dans le dossier.
3. Définissez les propriétés appropriées, puis ajoutez une ou plusieurs transformations en utilisant l'outil Ajouter des transformations situé sur les onglets Pré-génération ou Post-

génération. Ces transformations doivent avoir été préalablement définies dans la catégorie Profile\Transformations de l'extension courante.

Propriété d'un profil de transformation

Vous définissez un profil de transformation à l'aide des propriétés suivantes :

Propriété	Description
Nom	Nom du profil de transformation
Commentaire	Informations supplémentaires relatives au profil de transformation
Type de modèle	[facultatif] Spécifie le type de modèle avec lequel le profil de transformation peut être utilisé. C'est une façon de filtrer les profils lors de la génération. Par exemple, si vous sélectionnez MOO alors que l'extension courante se trouve dans un MPD, le profil de transformation peut être utilisé lors de la génération d'un MPD vers un MOO ou lors du reverse engineering d'un MOO vers un MPD
Famille et sous-famille	[facultatif] Si le type du modèle prend en charge un fichier de ressource cible, vous pouvez également définir une famille et une sous-famille pour filtrer l'affichage des profils dans la boîte de dialogue de génération. La famille est utilisée pour établir un lien entre le fichier de ressource d'un modèle et une extension. Lorsque la famille du fichier de ressource correspond à la famille de l'extension, cela suggère que l'extension complète le fichier de ressource
Pré-génération	L'onglet Pré-génération affiche une liste de transformations à exécuter avant la génération de modèle. Ces transformations sont exécutées lorsque le modèle courant au sein duquel vous avez créé l'extension est le modèle source et que les contraintes définies dans les zones Type de modèle, Famille et Sous-famille sont respectées. Tout objet créé lors de la pré-génération est automatiquement ajouté à la liste des objets utilisés dans la génération. Ces changements du modèle source sont temporaires et sont annulés à la fin de la génération. Par exemple, vous pouvez définir un profil de transformation à l'aide d'une transformation qui annule la création des EJB à partir des classes avant de générer un MOO dans un MPD, ce afin d'établir une meilleure correspondance entre les classes et les tables lors de la génération
Post-génération	L'onglet Post-génération du profil affiche une liste ordonnée de transformations qui sont exécutées à l'issue de la génération. Ces transformations sont exécutées après la génération, lorsque le modèle courant au sein duquel vous avez créé l'extension, est le modèle cible. Par exemple, vous pouvez définir un profil de transformation avec une transformation qui applique automatiquement les conventions de dénomination appropriées au modèle généré.

Chapitre 3

Fichiers de définition pour les langage objet, de processus et XML

Une fichier distinct est fourni pour chaque langage de MOO, MPM ou MSX pris en charge par PowerAMC, ce fichier définit la syntaxe et les règles de génération d'objets et de mise en oeuvre des stéréotypes, types de données, scripts et constantes pour le langage. Vous devez sélectionner un fichier de définition de langage lorsque vous créez un MOO, MPM ou MSX.

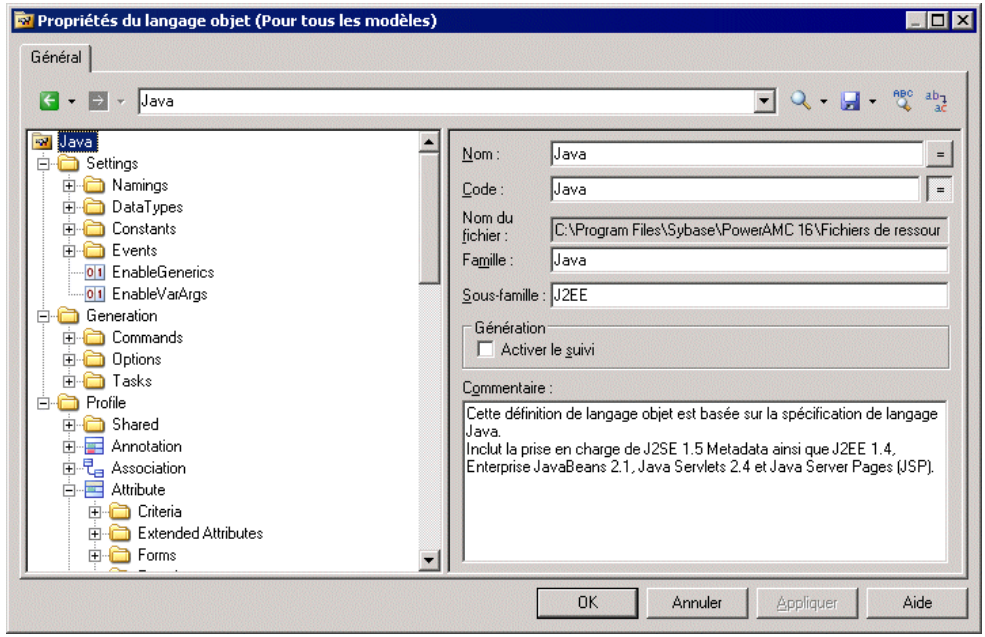
Les autres types de fichiers de définition de langage sont expliqués dans ce chapitre :

- MOO - Fichiers de définition de langage objet (.xol)
- MPM – Fichiers de définition de langage de processus (.xpl)
- MSX - Fichiers de définition de langage XML (.xsl)

Remarque : Le MPD utilise une forme différente de fichier de définition (voir *Chapitre 4, Fichiers de définition de SGBD* à la page 139), et les autres types de modèle n'ont pas de fichiers de définition, mais peuvent être étendus à l'aide de fichiers d'extension (voir *Chapitre 2, Fichiers d'extension* à la page 23).

Tous les langages cible ont la même structure de catégories de base, mais les détails et les valeurs des entrées diffèrent d'un langage à l'autre :

- Settings - contient des catégories Data Types, Constants, Namings et Events permettant de personnaliser et de gérer les fonctionnalités de génération. Les types des entrées de ces catégorie varient en fonction du type de fichier de ressources.
- Generation - contient des commandes, options et tâches de génération.
- Profile - contient des extensions définies sur les métaclasses.



Le noeud racine de chaque fichier contient les propriétés suivantes Pro:

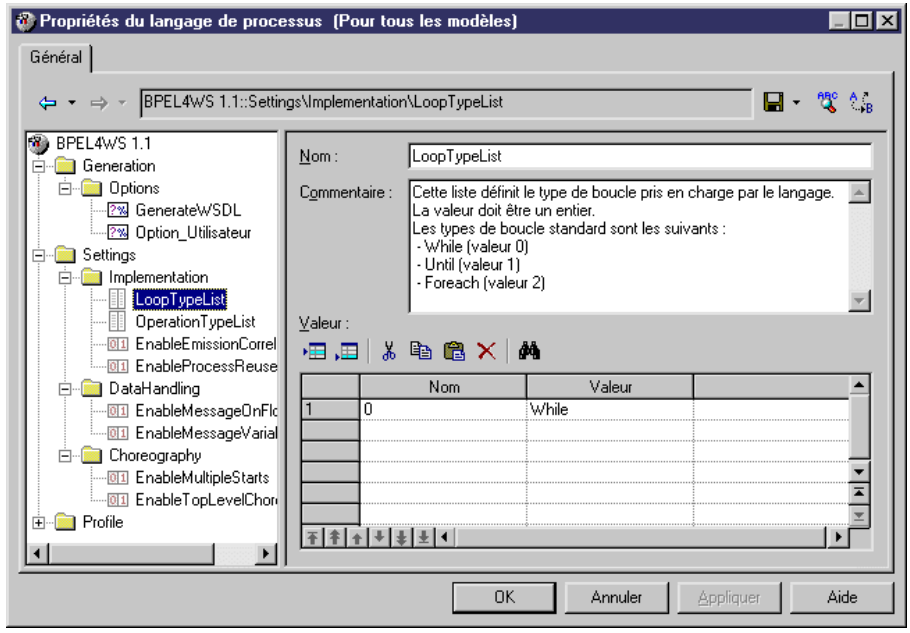
Propriété	Description
Nom	Spécifie le nom du langage cible.
Code	Spécifie le code du langage cible.
Nom de fichier	[lecture seule] Spécifie le chemin du fichier .xol, xpl ou .xsl. Si le langage cible a été copié dans votre modèle, cette zone est vide.
Version	[lecture seule] Spécifie la version du référentiel si la ressource est partagée via le référentiel
Famille	Active certaines fonctionnalités absentes par défaut dans le modèle. Par exemple, les langages objet des familles Java, XML, IDL et PowerBuilder® prennent en charge le reverse engineering.
Sous-famille	Permet d'affiner les fonctionnalités définies pour une famille particulière ; par exemple, dans la famille Java, la sous-famille J2EE permet de gérer les Entreprise Java beans ou de rendre possible la création de servlets et de JSP

Propriété	Description
Activer le suivi	Permet de prévisualiser les templates utilisés lors de la génération. Avant même de lancer la génération, vous pouvez afficher la page Aperçu de l'objet impliqué dans la génération pour voir ses templates et utiliser l'outil Réactualiser pour les afficher. Lorsque vous double-cliquez sur une ligne de suivi dans la page Aperçu, la définition de template correspondante s'affiche dans l'éditeur de ressources, dans la catégorie Profile\Objet\Templates. Le code du template peut être affiché dans des couleurs distinctes (voir le paragraphe Coloration syntaxique dans la section *Catégorie Generated Files* à la page 118).
Commentaire	Spécifie des informations supplémentaires sur le langage objet.

Catégorie Settings : langage de processus

La catégorie Settings contient les éléments suivants, utilisés pour contrôler les types de données, constantes, noms et catégories d'événements et pour personnaliser et gérer les fonctionnalités de génération de MPM :

- *Implementation* – [MPM exécutable uniquement] Rassemble les options qui influencent les possibilités de mise en oeuvre du processus. Les constantes suivantes sont définies par défaut :
  - *LoopTypeList* - Cette liste définit le type de boucle pris en charge par le langage. La valeur doit être un entier
  - *OperationTypeList* - Cette liste définit le type d'opération pris en charge par le langage. Une opération d'un type non pris en charge ne peut pas être associée à un processus. La valeur doit être un entier
  - *EnableEmissionCorrelation* - Ce paramètre permet la définition d'une corrélation pour un message émis
  - *EnableProcessReuse* - Ce paramètre permet à un processus d'être mis en oeuvre par un autre processus
  - *AutomaticInvokeMode* - Ce paramètre indique si le type d'action d'un processus mis en oeuvre par une opération peut être automatiquement déduit du type d'opération. Les valeurs possibles sont les suivantes :
    - 0 (valeur par défaut). Le type d'action ne peut pas être déduit et doit être spécifié
    - 1. Le langage impose au processus de recevoir une opération Demande-Réponse et une opération Sens unique et d'appeler une opération Demande-Réponse et une opération Notification
    - 2. Le langage s'assure qu'une opération Sollicitation-Réponse et une opération Notification est toujours reçue par le processus tandis que les opérations Demande-Réponse et Sens unique sont toujours appelées par le processus.

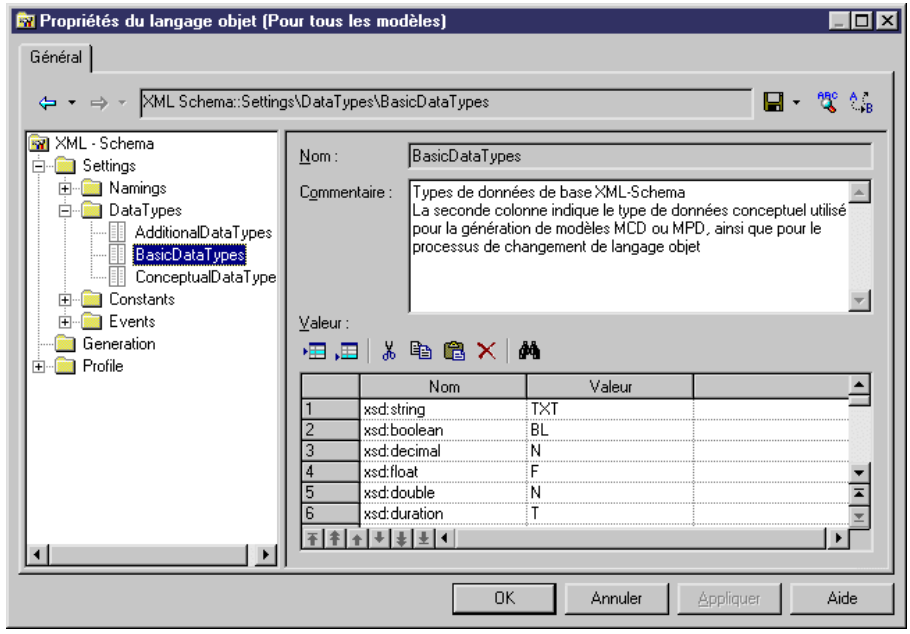


- *DataHandling* - [MPM exécutable uniquement] Rassemble des options relatives à la gestion des données dans le langage. Les valeurs constantes suivantes sont définies par défaut :
  - *EnableMessageOnFlow* - Indique si un format de message peut ou non être associé à un flux. La valeur par défaut est Oui
  - *EnableMessageVariable* - Permet à une variable de stocker la totalité d'un format de message. Dans ce cas, le format de message apparaîtra dans la liste Type de données de la variable
- *Choreography* - Rassemble des objets qui permettent de modéliser le graphique des activités (début, fin, décision, synchronisation, transition...) Contient les constantes suivantes définies par défaut :
  - *EnableMultipleStarts* - Lorsque défini à Non, ce paramètre vérifie qu'un processus composite ne comporte pas plusieurs débuts
  - *EnableTopLevelChoreography* - Lorsque défini à Non, ce paramètre vérifie qu'aucun flux ou objet de chorégraphie (début, fin, décision...) n'est défini directement sous le modèle ou sous un package. Ces objets peuvent être définis uniquement sous un processus composite

Catégorie Settings : langage objet

La catégorie Settings contient les éléments suivants, utilisés pour contrôler les types de données, constantes, noms et catégories d'événements et pour personnaliser et gérer les fonctionnalités de génération de MOO :

- *Data Types* - Table permettant de faire correspondre des types de données internes avec des types de données de langage objet. Les types de données suivants sont définis par défaut :
  - *BasicDataTypes* – liste les types de données de langage objet les plus utilisés. La colonne Valeur indique le type de données conceptuel utilisé pour la génération de MCD et de MPD.
  - *ConceptualDataTypes* – liste les types de données internes de PowerAMC. La colonne Valeur indique le type de données de langage objet utilisé pour la génération des modèles MCD et MPD.
  - *AdditionalDataTypes* – liste les types de données supplémentaires ajoutés dans les listes de types de données. Peut être utilisé pour ajouter ou modifier vos propres types de données. La colonne Valeur indique le type de données conceptuel utilisé pour la génération des modèles MCD et MPD.
  - *DefaultDataType* – spécifie le type de données par défaut.



- *Constants* - contient une table de correspondances entre les constantes suivantes et leurs valeurs par défaut : Null, True, False, Void, Bool.

- *Namings* - contient des paramètres qui influent sur ce qui sera inclus dans les fichiers que vous générez à partir d'un MOO :
  - *GetterName* - Nom et valeur pour les opérations getter
  - *GetterCode* - Code et valeur pour les opérations getter
  - *SetterName* - Nom et valeur pour les opérations setter
  - *SetterCode* - Code et valeur pour les opérations setter
  - *IllegalChar* - Liste des caractères illégaux dans le langage objet courant. Cette liste définit le contenu de la zone Caractères interdits dans (**Outils** > **Options du modèle** > **Convention de dénomination**). Par exemple, `"/!=<>"" ' ( )"`
- *Events* - définit des événements standard sur les opérations. Cette catégorie peut contenir des événements existants par défaut tels que les constructeur et destructeur, en fonction du langage objet. Un événement est lié à une opération. Le contenu de la catégorie Events est affiché dans la liste Evénement dans les feuilles de propriétés d'opération. Il décrit les événements qui peuvent être utilisés par une opération. Dans PowerBuilder, par exemple, la catégorie Events est utilisée pour associer les opérations aux événements PowerBuilder.

Catégorie Settings : langage XML

La catégorie Settings contient la sous-catégorie Data types qui montre la correspondance entre les types de données internes et ceux du langage XML.

Les types de données suivants sont définis par défaut :

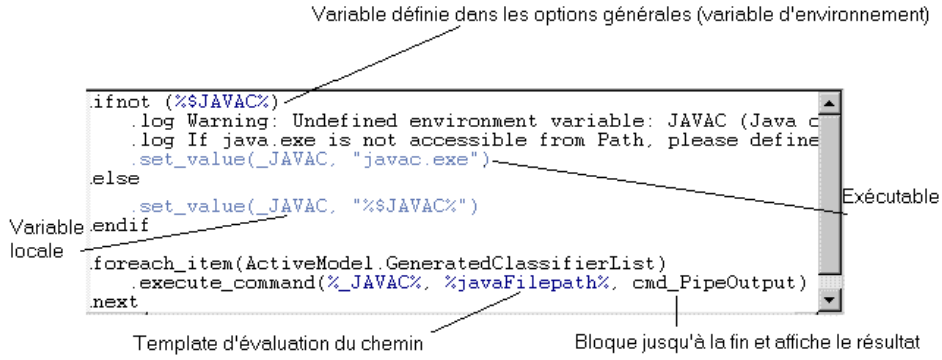
- ConceptualDataTypes - La colonne Valeur indique le type de données de langage XML utilisé pour la génération des modèles. Les types de données conceptuels sont les types de données internes de PowerAMC, et ils ne peuvent pas être modifiés
- XsmDataTypes- Types de données pour les générations depuis le modèle XML

Catégorie Generation

La catégorie Generation contient des catégories et des entrées permettant de définir et d'activer un processus de génération.

Les sous-catégories suivantes sont disponibles :

- *Commands* - contient des commandes génération, qui peuvent être exécutées à la fin du processus de génération, après la génération de tous les fichiers. Ces commandes sont écrites dans le langage de génération par template (voir *Chapitre 5, Personnalisation de la génération à l'aide du langage de génération par template* à la page 285), et doivent être incluses au sein de tâches à évoquer.



- *Options* – contient des options, disponibles sur l'onglet Options de la boîte de dialogue de génération, dont les valeurs peuvent être testées par des templates ou des commandes génération. Vous pouvez créer des options qui prennent des valeurs booléennes, des chaînes ou des listes. La valeur d'une option peut être appelée dans un template à l'aide de la syntaxe suivante :

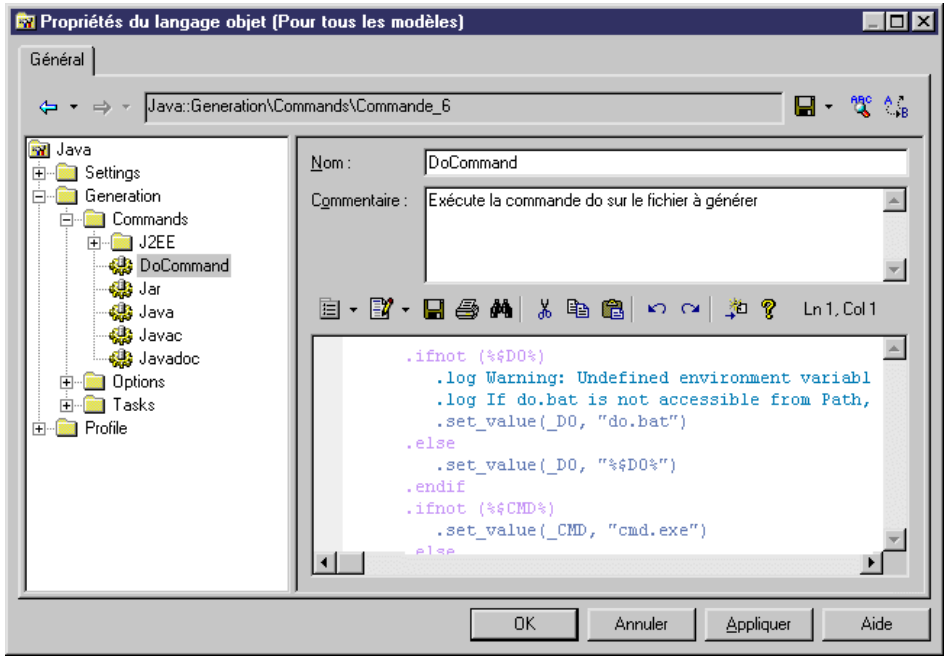
  ```
  '%' 'GenOptions.'<nom-option> '%'
  ```

  Par exemple, pour une option booléenne nommée GenerateComment, %GenOptions.GenerateComment% va être évalué en true ou false dans un template, selon la valeur spécifiée dans l'onglet Options de la boîte de dialogue de génération.
- *Tasks* – contient des tâches, disponibles sur l'onglet Tâches de la boîte de dialogue de génération, et qui contient la liste des commandes de génération (voir ci-avant). Lorsqu'une tâche est sélectionné dans l'onglet Tâches, les commandes incluses dans la tâches sont extraites et leurs templates évalués et exécutés.

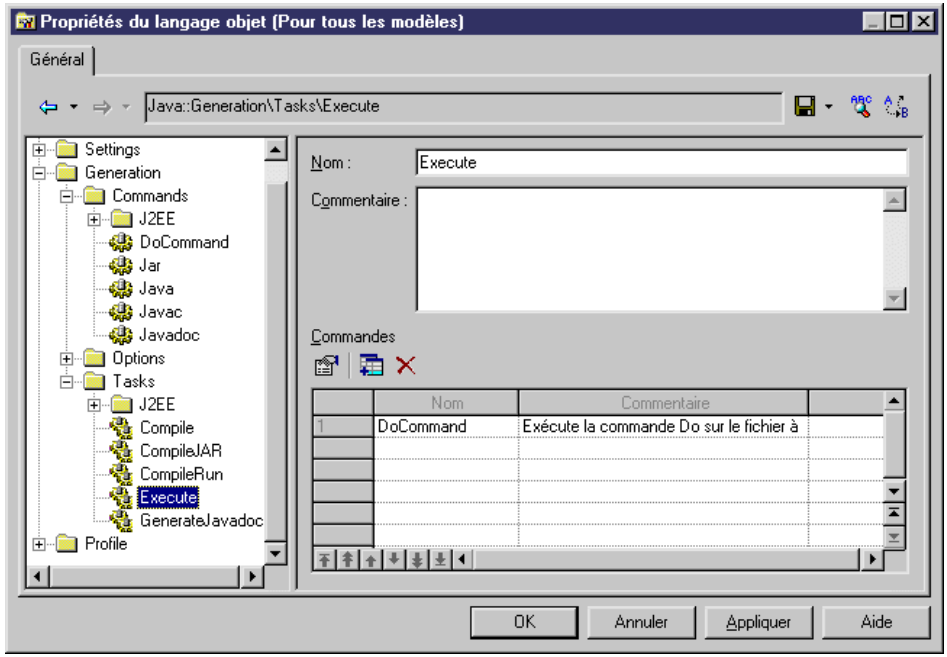
Exemple : Ajout d'une commande et d'une tâche de génération

Dans cet exemple, nous ajoutons une commande de génération et une tâche associée dans le langage objet Java :

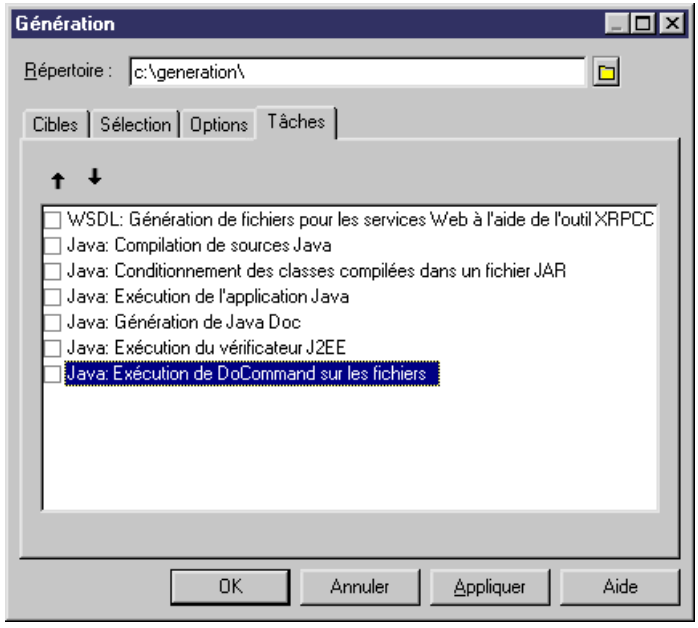
1. Créez un nouveau MOO pour Java, puis sélectionnez **Langage > Editer le langage objet courant** pour afficher le contenu du fichier de ressources Java.
2. Développez la catégorie Generation, double-cliquez sur la catégorie Commands, puis sélectionnez Nouveau dans le menu contextuel pour créer une nouvelle commande.
3. Nommez la commande DoCommand, puis saisissez le template approprié :



4. Pointez sur la catégorie Tasks, cliquez le bouton droit de la souris, puis sélectionnez Nouveau dans le menu contextuel, afin de créer une nouvelle tâche. Nommez cette tâchez Execute, cliquez sur l'outil Ajouter des commandes, sélectionnez DoCommand dans la liste, puis cliquez sur OK pour ajouter cette commande à la nouvelle tâche :



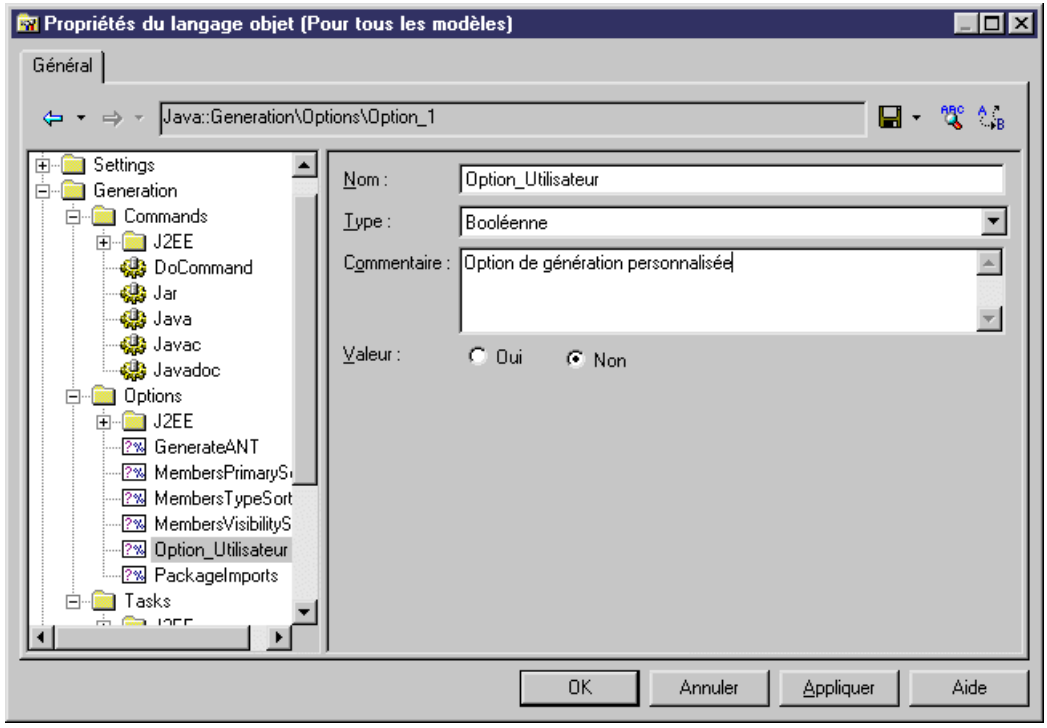
5. Cliquez sur OK pour enregistrer vos modifications et revenir au modèle. Sélectionnez **Langage > Générer du code Java** pour afficher la boîte de dialogue Génération, puis cliquez sur l'onglet Tâches. La nouvelle tâche est répertoriée sur l'onglet, sous son commentaire (ou sous son nom, si aucun commentaire n'a été défini) :



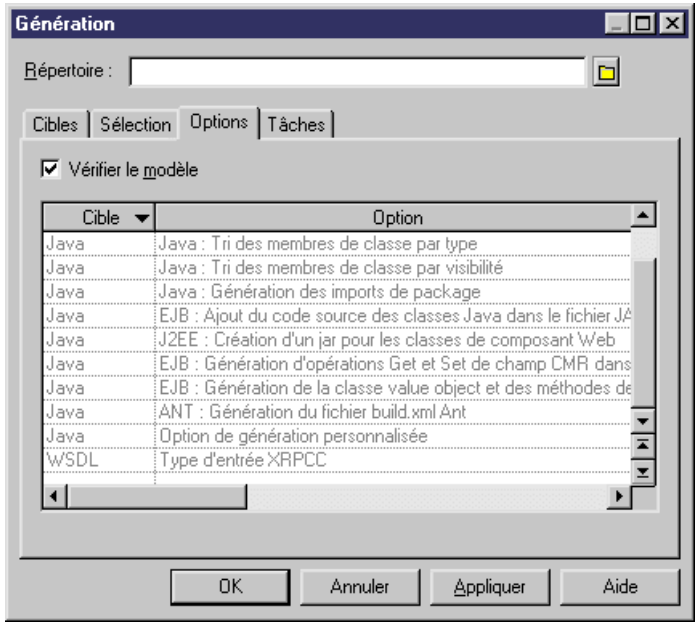
Exemple 2 : Ajout d'une option de génération

Dans ce exemple, nous ajoutons une option de génération au langage objet Java.

1. Sélectionnez **Langage > Editer le langage objet courant** pour afficher le contenu du fichier de ressources Java.
2. Développez la catégorie Generation, double-cliquez sur la catégorie Options, puis sélectionnez Nouveau dans le menu contextuel pour créer une nouvelle option :



3. Cliquez sur OK pour enregistrer vos modifications et revenir au modèle. Sélectionnez **Langage > Générer du code Java** pour afficher la boîte de dialogue Génération, puis cliquez sur l'onglet Options. La nouvelle option est répertoriée sur l'onglet, sous son commentaire (ou sous son nom, si aucun commentaire n'a été défini) :



Remarque : Pour obtenir des informations détaillées sur la création ou la modification de templates de génération, voir *Chapitre 5, Personnalisation de la génération à l'aide du langage de génération par template* à la page 285. Nous vous recommandons de commencer par lire ce chapitre afin de vous familiariser avec les concepts et fonctionnalités du processus de génération.

Catégorie Profile (fichiers de définition)

La catégorie Profile d'un fichier de définition de langage contient les catégories Stereotypes, Extended attributes, Methods, etc, afin de permettre d'étendre les métaclasses définies dans le métamodèle PowerAMC.

Dans les langages objet, la catégorie `Shared/Extended Attribute Types` contient différents attributs utilisés pour contrôler la prise en charge du langage objet dans PowerAMC La variable **ObjectContainer** spécifie un conteneur par défaut pour les associations de mise en oeuvre. Cet attribut a une liste éditable de valeurs possibles pour chaque langage objet, à partir de laquelle vous pouvez sélectionner une valeur par défaut pour votre langage. Vous pouvez, si besoin est, passer outre cette valeur par défaut en utilisant l'option de modèle **Conteneur d'association par défaut**.

Pour obtenir des informations détaillées sur la catégorie Profile, voir *Chapitre 2, Fichiers d'extension* à la page 23.

Chapitre 4 Fichiers de définition de SGBD

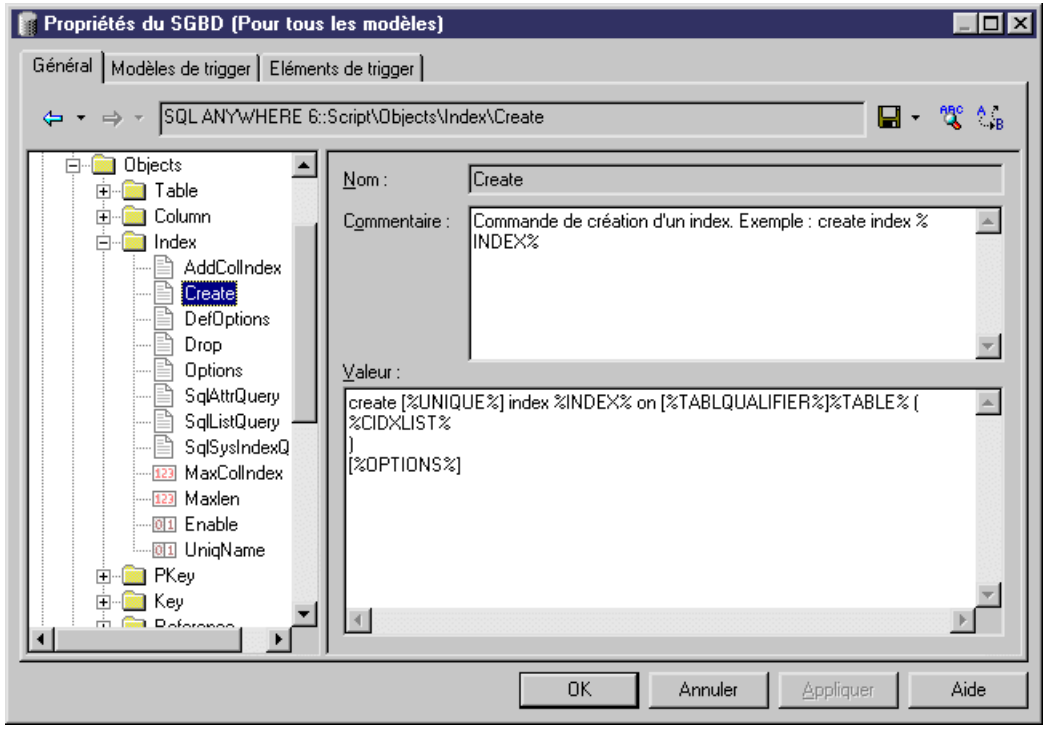
Un fichier de définition de SGBD est un fichier de ressource PowerAMC qui fournit à PowerAMC les informations nécessaires pour modéliser ou générer un SGBD, ou bien pour procéder à son reverse engineering.

PowerAMC fournit des fichiers de définition pour la plupart des SGBD les plus connus. Les fichiers de définition de SGBD sont situés dans le sous-répertoire /Fichiers de ressources/SGBD, et ont un suffixe .xdb.

Remarque : Les modifications apportées à un fichier de définition de SGBD peuvent changer la façon dont se comportent les fonctions de PowerAMC, tout particulièrement lors de la génération de scripts. Créez des copies de sauvegarde de vos définitions de bases de données, et testez scrupuleusement chaque script généré avant de l'exécuter.

Afficher votre fichier de définition de SGBD cible dans l'Editeur de ressources

Vous pouvez consulter ou modifier un fichier de définition de SGBD en utilisant l'Editeur de ressource. Lorsque vous sélectionnez une *catégorie* ou un élément dans le volet gauche, le nom, la valeur et le commentaire associé sont affichés du côté droit de la boîte de dialogue.



Sélectionnez **SGBD > Editer le SGBD courant**.

La boîte de dialogue Propriétés du SGBD s'affiche.

Remarque : Ne modifiez pas les fichiers de SGBD fournis avec PowerAMC. Pour chaque SGBD que vous souhaitez modifier, créez un nouveau SGBD correspondant. Pour ce faire, créez le nouveau SGBD à partir de la boîte de dialogue Liste des SGBD, définissez un nom, puis sélectionnez le SGBD d'origine dans la liste Copier depuis. Pour plus d'informations, voir "Utilisation de l'éditeur de ressources" dans le chapitre Fichiers de ressources et métamodèle public de PowerAMC.

Pour plus d'informations sur l'utilisation de l'éditeur, voir *Ouverture de fichiers de ressources dans l'éditeur* à la page 3.

Structure du fichier de définition de SGBD

Tous les fichiers de définition de SGBD ont la même structure composée de catégories, chacune pouvant contenir des éléments ou d'autres catégories. Les éléments, et leurs valeurs, sont différents dans chaque SGBD. Chaque élément n'est présent que s'il est pertinent dans un SGBD particulier. Chaque valeur est une instruction SQL ou un autre paramètre pouvant définir comment modéliser, générer pour le SGBD, ou lui faire subir un reverse engineering.

Chaque fichier de SGBD a la structure suivante :

- *General* - contient des informations générales sur la base de données, sans catégorie (voir *Catégorie General* à la page 160). Tous les éléments définis dans la catégorie General s'appliquent à tous les objets de la base de données.
- *Script* - utilisé pour la génération et le reverse engineering. Contient les sous-catégories suivantes :
  - *SQL* - contient les sous-catégories suivantes, chacune d'entre elles contenant des éléments dont les valeurs définissent une syntaxe générale pour la base de données :
    - *Syntax* - paramètres généraux relatifs à la syntaxe SQL (voir *Catégorie Syntax* à la page 161)
    - *Format* - paramètres relatifs aux caractères admis (voir *Catégorie Format* à la page 162)
    - *File* - éléments de texte header, footer et usage utilisés lors de la génération (voir *Catégorie File* à la page 165)
    - *Keywords* - liste des mots réservés SQL et des fonctions (voir *Catégorie Keywords* à la page 167)
  - *Objects* - contient des commandes permettant de créer, supprimer ou modifier tous les objets dans la base de données. Inclut également des commandes définissant le comportement de l'objet, ses valeurs par défaut et les requêtes SQL nécessaires, les options de reverse engineering, etc. (voir *Catégorie Script/Objects* à la page 168).
  - *Data Type* - contient la liste des types de données valides pour le SGBD spécifié et les types correspondants dans PowerAMC (voir *Catégorie Script/Data Type* à la page 230)
  - *Customize* - Extrait des informations depuis les fichiers de définition de SGBD PowerAMC Version 6. N'est plus utilisé dans les version ultérieures.
- *ODBC* - présent uniquement si le SGBD ne prend pas en charge les instructions standard pour la génération. Dans ce cas, la catégorie ODBC contient des éléments supplémentaires nécessaires pour la génération via une connexion directe.
- *Transformation Profiles* – contiennent des groupes de transformations utilisées lors de la génération de modèle lorsque vous devez appliquer des modifications à des objets source ou cible. Pour plus d'informations, voir *Transformations et profils de transformation (Profile)* à la page 121 et *Guide des fonctionnalités générales > Liaison et synchronisation de modèles > Génération de modèles et d'objets de modèle > Génération de modèles et d'objets de modèle > Fenêtre d'options de génération > Application de transformations*.

- *Profile* - permet de définir des types d'attributs étendus et des attributs étendus pour les objets de base de données. Pour plus d'informations, voir *Catégorie Profile* à la page 233.

Page de propriétés d'un SGBD

Un SGBD a une page de propriétés accessible lorsque vous cliquez sur le noeud racine dans l'arborescence. Les propriétés suivantes y sont définies :

Propriété	Description
Nom	Nom du SGBD. Ce nom doit être unique dans un modèle
Code	Code du SGBD. Ce code doit être unique dans un modèle
Nom de fichier	[lecture seule] Chemin d'accès et nom du fichier de SGBD.
Famille	Utilisé pour classifier un SGBD, ainsi que pour établir un lien entre différents fichiers de ressources de base de données. Par exemple, Sybase AS Anywhere et Sybase AS Enterprise appartiennent à la famille SQL Server. Les triggers ne sont pas effacés lorsque vous changez de base de données cible au sein d'une même famille. Une interface de fusion permet de fusionner des modèles de la même famille.
Commentaire	Informations supplémentaires relatives au SGBD

Modèles de triggers, éléments de modèle de trigger et modèles de procédure

Les modèles de trigger de SGBD, éléments modèle de trigger et modèles de procédure sont accessibles via les onglet de la fenêtre Editeur de ressource. En outre, dans le cas d'Oracle, il existe un onglet pour les templates de package de base de données.

Les modèles pour les procédures stockées sont définis sous la catégorie Procédure dans l'arborescence de SGBD.

Pour plus d'informations, voir *Modélisation des données > Construction de modèles de données > Triggers et procédures*.

Gestion de la génération et du reverse engineering

PowerAMC prend en charge la génération et le reverse engineering à la fois par *scripts* et via des connexions *directes à la base de données*.

Dans cette section :

- *Instruction* est utilisé pour définir une syntaxe SQL. Les instructions contiennent des variables qui seront évaluées lors de la génération et le reverse engineering de script.
- *Requête* est réservé pour décrire le reverse engineering direct de base de données

Les instructions pour la génération de script, le reverse engineering de script et la génération directe de base de données sont identiques, alors que le reverse engineering direct de base de données peut requérir des requêtes particulières.

Les processus de génération et de reverse engineering peuvent être définis comme suit :

- *Génération* - les instructions sont analysées et les variables évaluées et remplacées par leur valeur effective prise dans le modèle courant. Les mêmes instructions sont utilisées pour la génération de script et pour la génération directe.
- *Reverse engineering* – peut être effectué par :
  - *Script* - PowerAMC analyse le script et identifie les différentes instructions à l'aide de leur caractère de fin (défini dans Script\Sql\Syntax). Chaque instruction individuelle est "associée" avec une instruction existante dans le fichier de définition de SGBD afin de valider les variables dans les instructions récupérées via reverse engineering sous la forme d'éléments dans un modèle PowerAMC.
  - *Connexion directe à la base de données* - des requêtes spéciales sont utilisées pour récupérer des informations dans les tables système de la base de données. Chaque colonne d'un jeu de résultats est associée à une variable. Un en-tête de script spécifie l'association entre les colonnes du jeu de résultats et la variable. Les valeurs des enregistrements renvoyés sont stockées dans ces variables, qui sont alors validées comme valeurs d'attribut d'objet.

Pour plus d'informations sur les variables, voir *Chaînes et variables facultatives* à la page 248.

Catégorie Script

La catégorie Script contient les types d'éléments suivants :

- *Instructions de génération et de reverse engineering* - utilisées pour la génération et le reverse engineering à l'aide de script ou direct. Par exemple, l'instruction standard pour la création d'un index se présente comme suit :

```
create index %INDEX%
```

Ces instructions diffèrent d'un SGBD à l'autre SGBD. Par exemple, dans Oracle 9i, l'instruction de création d'index contient la définition d'un propriétaire :

```
create [%UNIQUE%?%UNIQUE% :[%INDEXTYPE% ]]index [%QUALIFIER%]%INDEX%
on [%CLUSTER%?cluster C_%TABLE%:[%TABLQUALIFIER%]%TABLE% (
 %CIDXLIST%
)]
[%OPTIONS%]
```

Les types d'instructions de génération et de reverse engineering suivants sont disponibles :

- Drop pour supprimer un objet
- Options pour définir les options physiques d'un objet
- ConstName pour définir le modèle de nom de contrainte pour les vérifications d'objet

- *Instructions de modification* - utilisées pour modifier les attributs d'objets existants. La plupart commencent par le mot "Modify", mais certaines incluent Rename ou AlterTableFooter.
  L'instruction pour créer une *clé* dépend de l'emplacement de création de la clé. Si la clé se trouve dans une table, elle sera créée avec un ordre de génération, si elle est créée hors de la table, il s'agira d'un ordre de modification de la table.
- *Eléments de définition de base de données* – utilisés pour personnaliser l'interface PowerAMC et son comportement en fonction des fonctionnalités de base de données. Par exemple, l'élément Maxlen dans la catégorie table doit être défini en fonction de la longueur maximale de code tolérée pour une table dans la base de données courante. Permission, EnableOwner, AllowedADT sont d'autres exemples d'éléments définis pour adapter PowerAMC au SGBD courant.
- *Requêtes de reverse engineering direct* - la plupart commencent par "Sql". Par exemple, SqlListQuery extrait une liste d'objets, et SqlOptsQuery permet de procéder au reverse engineering des options physiques. Pour plus d'informations, voir *Reverse engineering direct de base de données* à la page 148.

Catégorie ODBC

La catégorie ODBC contient des éléments pour la génération directe de base de données lorsque le SGBD ne prend pas en charge les instructions de génération définies dans la catégorie Script.

Par exemple, l'échange de données entre PowerAMC et MSACCESS fonctionne à l'aide de scripts VB et non de SQL, c'est pourquoi ces instructions sont situées dans la catégorie ODBC. Vous devez utiliser un programme spécial (access.mdb) pour convertir ces scripts en objets de base de données MSACCESS.

Génération de script

Les instructions de génération de script sont disponibles dans la catégorie Script, sous les différentes catégories d'objet. Par exemple, dans Sybase ASA 8, l'instruction Create de la catégorie Table se présente comme suit :

```
create table [%QUALIFIER%]%TABLE%
(
  %TABLDEFN%
)
[%OPTIONS%]
```

Cette instruction contient les paramètres de création de la table ainsi que le nom de son propriétaire et ses options physiques.

Mécanisme d'extension

Vous pouvez étendre des instructions de génération de script pour compléter la génération en utilisant les *instructions d'extension*. Le mécanisme d'extension permet de générer les instructions immédiatement avant ou après les instructions Create, Drop et Modify, et d'extraire ces instructions lors du reverse engineering.

Pour plus d'informations sur le reverse engineering d'instructions supplémentaires, reportez-vous à la section *Reverse engineering de script* à la page 147.

Langage de génération par template (GTL)

Les instructions d'extension sont définies à l'aide du mécanisme de langage de génération par template (GTL) PowerAMC.

Une instruction d'extension peut contenir :

- Une référence à d'autres *instructions* qui sera évaluée lors de la génération. Ces éléments sont des éléments de texte qui doivent être définis dans la même catégorie que les objet des instructions d'extension
- Des *variables* utilisées pour évaluer des propriétés d'objet et des attributs étendus. Les variables sont encadrées par des caractères %
- Des *macros*, telles que ".if", fournissant des structures de programmation générique pour tester des variables. Remarque : nous vous recommandons d'éviter d'utiliser des macros du GTL dans les scripts de génération, car elles ne peuvent pas être reconstituées lors d'un reverse engineering par script. La génération et le reverse engineering via une connexion directe ne sont pas concernés par cette limitation.

Pour plus d'informations sur le langage de génération par template, voir *Chapitre 5, Personnalisation de la génération à l'aide du langage de génération par template* à la page 285.

Lors de la génération, les instructions et variables sont évaluées et le résultat est ajouté au script global.

Exemple 1

L'instruction d'extension *AfterCreate* est définie dans la catégorie Table pour compléter l'instruction Create de la table, en ajoutant des partitions à la table si la valeur de l'attribut étendu de la table le requiert.

AfterCreate est défini dans la syntaxe de GTL comme suit :

```
.if (%ExtTablePartition% > 1)
%CreatePartition%
go
.endif
```

La macro .if est utilisée pour évaluer la variable %ExtTablePartition%. Cette variable est un attribut étendu qui contient le nombre de partitions de la table. Si la valeur de %ExtTablePartition% est supérieure à 1, %CreatePartition% sera généré suivi de "go". %CreatePartition% est une instruction définie dans la catégorie Table comme suit :

```
alter table [%QUALIFIER%]%TABLE%
 partition %ExtTablePartition%
```

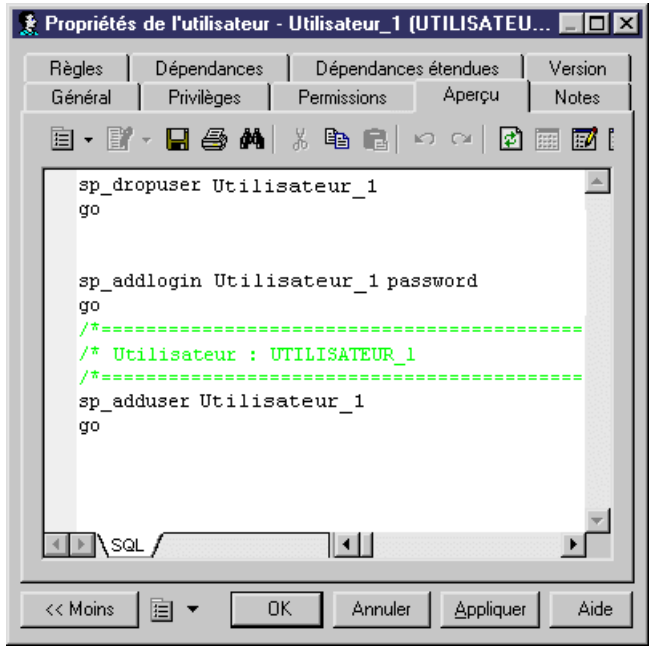
%CreatePartition% génère l'instruction de création du nombre de partitions de table spécifié dans %ExtTablePartition%.

Exemple 2

Vous créez dans Sybase ASE une instruction étendue pour créer automatiquement un login d'utilisateur avant l'exécution de l'instruction Create user. L'instruction BeforeCreate se présente comme suit :

```
sp_addlogin %Name% %Password%
go
```

Le login généré automatiquement aura le même nom que l'utilisateur et son mot de passe. Vous pouvez afficher un aperçu de l'instruction dans la feuille de propriétés de l'objet, l'instruction BeforeCreate apparaît avant l'instruction de création de l'utilisateur :



Instructions Modify

Vous pouvez également ajouter des instructions BeforeModify et AfterModify aux instructions *modify* standard.

Les instructions Modify sont exécutées afin de synchroniser la base de données avec la structure créée dans le MPD. Par défaut, la fonctionnalité de modification de base de données ne prend pas en compte les attributs étendus lorsqu'elle compare les changements effectués sur le modèle depuis la dernière génération. Vous pouvez contourner cette règle en ajoutant des attributs étendus dans l'élément de liste *ModifiableAttributes*. Les attributs étendus définis dans cette liste seront pris en compte dans la boîte de dialogue de fusion lors de la synchronisation de base de données.

Pour détecter qu'une valeur d'attribut étendu a été modifiée, vous pouvez utiliser les variables suivantes :

- %OLDOBJECT% pour accéder à l'ancienne valeur de l'objet
- %NEWOBJECT% pour accéder à la nouvelle valeur de l'objet

Par exemple, vous pouvez vérifier que la valeur de l'attribut étendu ExtTablePartition a été modifiée à l'aide de la syntaxe de GTL suivante :

```
.if (%OLDOBJECT.ExtTablePartition% != %NEWOBJECT.ExtTablePartition%)
```

Si la valeur d'attribut étendu a été modifiée, une instruction étendue sera générée pour mettre à jour la base de données. Dans la syntaxe de Sybase ASE, l'instruction étendue ModifyPartition se présente comme suit, car en cas de changement de partition vous devez supprimer la précédente partition avant de la recréer :

```
.if (%OLDOBJECT.ExtTablePartition% != %NEWOBJECT.ExtTablePartition%)
 .if (%NEWOBJECT.ExtTablePartition% > 1)
  .if (%OLDOBJECT.ExtTablePartition% > 1)
%DropPartition%
  .endif
%CreatePartition%
 .else
%DropPartition%
 .endif
.endif
```

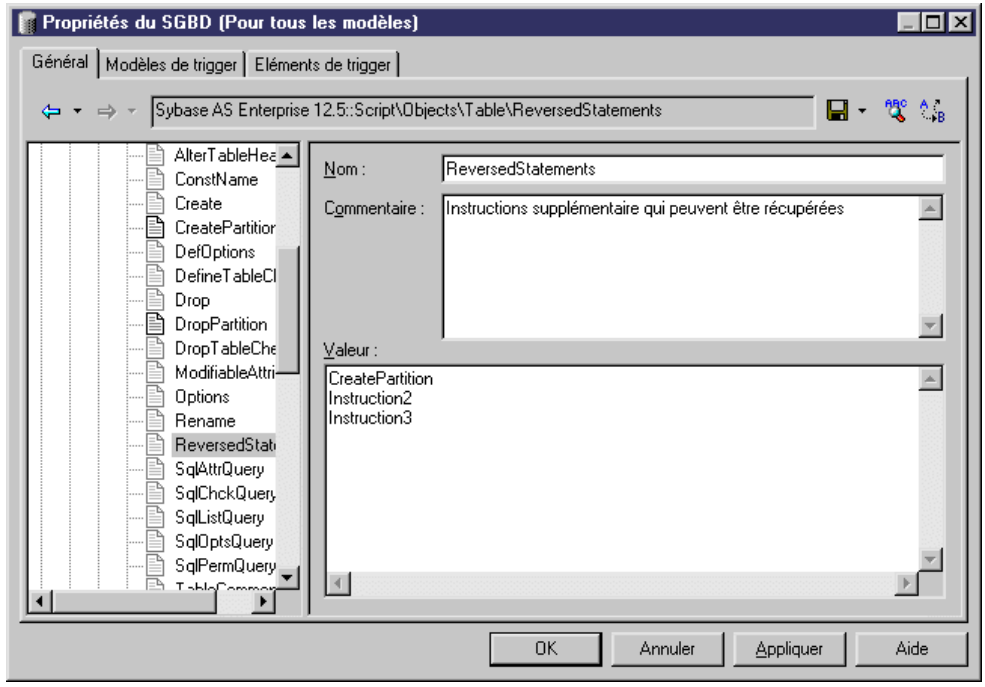
Pour plus d'informations sur le langage de génération par template (GTL) PowerAMC, voir *Chapitre 5, Personnalisation de la génération à l'aide du langage de génération par template* à la page 285.

Reverse engineering de script

Les mêmes instructions sont utilisées pour la génération et le reverse engineering.

Si vous utilisez le mécanisme d'extension pour la génération de script, vous devez déclarer les instructions dans l'élément de liste *ReversedStatements* afin qu'elles puissent être correctement traitées par le reverse engineering. Saisissez une instruction par ligne dans la liste ReversedStatement.

Par exemple, l'instruction d'extension AfterCreate utilise l'instruction CreatePartition. Cet élément de texte doit être déclaré dans ReversedStatements pour être correctement traité par le reverse engineering. Vous pouvez déclarer d'autres instructions de la façon suivante :



Génération directe de base de données

Le plus souvent, la génération directe utilise les mêmes instructions que la génération de script. Toutefois, lorsque le SGBD ne prend pas en charge la syntaxe SQL standard, des instructions de génération spéciales sont définies dans la catégorie ODBC. C'est notamment le cas pour MSACCESS qui a besoin de scripts VB pour créer des objets de base de données à l'aide de la génération directe.

Ces instructions sont définies dans la catégorie ODBC du SGBD.

Reverse engineering direct de base de données

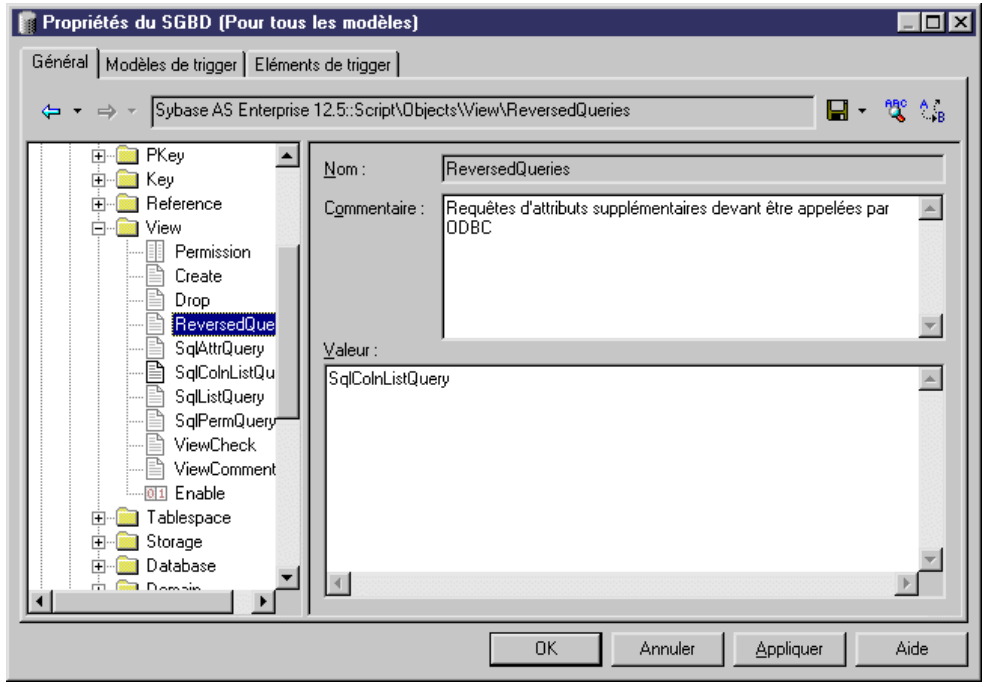
Le SGBD contient des requêtes de reverse engineering direct permettant d'extraire des objets (tables, colonnes, etc.) de la base de données.

La plupart des requêtes sont nommées sur le modèle "Sql...Query".

Elément	Description
SqlListQuery	Dresse la liste des objets pouvant être sélectionnés dans la zone Sélection. `SqlListQuery` extrait les objets et remplit la fenêtre de reverse engineering. Par la suite, chacune des autres requêtes ci-dessous est exécutée pour chaque objet sélectionné. Si `SqlListQuery` n'est pas défini, des fonctions ODBC standard sont utilisées pour extraire les objets. `SqlAttrQuery`, `SqlOptsQuery` etc. seront ensuite exécutées, si elles ont été définies. `SqlListQuery` doit extraire le plus petit nombre de colonnes possible car le processus fait une utilisation intensive de la mémoire
SqlAttrQuery	Procède au reverse engineering d'attributs d'objets. `SqlAttrQuery` peut ne pas être nécessaire si `SqlListQuery` peut extraire toutes les informations nécessaires. Par exemple, dans Sybase Adaptive Server® Anywhere 6, `TablespaceListQuery` suffit pour extraire toutes les informations requises pour l'utilisation dans un MPD
SqlOptsQuery	Procède au reverse engineering des options physiques
SqlListChildrenQuery	Procède au reverse engineering des objets enfant, par exemple des colonnes d'un index ou d'une clé particulière, des jointures d'une référence spécifique
SqlSysIndexQuery	Procède au reverse engineering des index système créés par la base de données
SqlChckQuery	Procède au reverse engineering des contraintes relatives aux vérifications d'objet
SqlPermQuery	Procède au reverse engineering de permissions sur les objets

Vous pouvez définir des requêtes supplémentaires pour récupérer plusieurs attributs lors du reverse engineering direct, ce afin d'éviter de charger *SqlListQuery* avec des requêtes pour extraire des attributs non pris en charge par *SqlAttrQuery*, ou des objets non sélectionnés pour le reverse engineering. Ces requêtes supplémentaires doivent être répertoriées dans l'élément *ReversedQueries*. Par exemple, *SqlColnListQuery* est utilisé exclusivement pour récupérer des colonnes de vue. Cette requête doit être déclarée dans l'élément ReversedQueries pour être prise en compte lors du reverse engineering.

Remarque : les requêtes étendues ne doivent pas être définies dans l'élément *ReversedQueries*. Pour plus d'informations sur ReversedQueries, voir la section *Mécanisme d'extension pour les requêtes de reverse engineering direct* à la page 152.



Structure de requête

Chaque colonne d'un jeu de résultats est associée à une variable. Un en-tête de script spécifie l'association entre les colonnes du jeu de résultats et la variable. Les valeurs des enregistrements renvoyés sont stockées dans ces variables, qui sont alors validées comme valeurs d'attribut d'objet.

L'en-tête de script est contenu entre accolades { }. Ces variables sont répertoriées entre crochets, et sont séparées les unes des autres par une virgule. Il existe une colonne pour chaque variable dans l'instruction Select qui suit l'en-tête.

Par exemple :

```
{OWNER, @OBJTCODE, SCRIPT, @OBJTLABL}
SELECT U.USER_NAME, P.PROC_NAME, P.PROC_DEFN, P.REMARKS
FROM SYSUSERPERMS U,SYSPROCEDURE P
WHERE [%SCHEMA% ? U.USER_NAME='%SCHEMA%' AND] P.CREATOR=U.USER_ID
ORDER BY U.USER_NAME
```

La liste des variables possibles correspond à la liste des variables établie dans la section *Variables de MPD* à la page 247.

Chaque partie de l'en-tête (séparée par des virgules) est associée aux informations suivantes :

- Nom de la variable (obligatoire). Voir l'exemple dans *Traitement avec des noms de variable*
- Le mot clé `ID` suit chaque nom de variable. ID signifie que la variable fait partie de l'identifiant
- Le mot clé `...` (points de suspension) signifie que la variable doit être concaténée pour toutes les lignes renvoyées par la requête SQL et ayant les mêmes valeurs pour les colonnes d'ID
- `Retrieved_value = PD.value` répertorie l'association entre une valeur extraite et une valeur PowerAMC. Une table de conversion permet de convertir chaque valeur de l'enregistrement (table système) en une autre valeur (dans PowerAMC). Ce mécanisme est un mécanisme alternatif. Voir l'exemple dans *Traitement avec une table de conversion*

La seule information obligatoire est le nom de variable. Toutes les autres informations sont facultatives. Les mots clés `ID` et `...` (points de suspension) sont mutuellement exclusifs.

Traitement avec des noms de variable

```
{TABLE ID, ISPKEY ID, CONSTNAME ID, COLUMNS ...}
select
 t.table_name,
 1,
 null,
 c.column_name + ', ',
 c.column_id
from
 systable t,
 syscolumn c
where
etc..
```

Dans ce script, l'identifiant est défini comme TABLE + ISKEY+ CONSTNAME.

Dans les lignes de résultat renvoyées par le script SQL, les valeurs du quatrième champ sont concaténées dans le champ COLUMNS tant que ces valeurs d'ID sont identiques.

```
SQL Result set
Table1,1,null,'col1,'
Table1,1,null,'col2,'
Table1,1,null,'col3,'
Table2,1,null,'col4,'
In PowerDesigner memory
Table1,1,null,'col1,col2,col3'
Table2,1,null,'col4'
```

Dans l'exemple, COLUMNS va contenir la liste des colonnes séparées par des virgules. PowerAMC va traiter le contenu du champ COLUMNS pour supprimer la dernière virgule.

Traitement avec une table de conversion

La syntaxe insérée immédiatement derrière un champ dans l'en-tête est la suivante :

```
(SQL value1 = PowerDesigner value1, SQL value2 = PowerDesigner
value2, * = PowerDesigner value3)
```

dans laquelle * représente toutes les autres valeurs.

Par exemple :

```
{ADT, OWNER, TYPE(25=JAVA , 26=JAVA)}
SELECT t.type_name, u.user_name,  t.domain_id
FROM sysusertype t, sysuserperms u
WHERE [u.user_name = '%SCHEMA%' AND]
(domain_id = 25 OR domain_id = 26) AND
t.creator = u.user_id
```

Dans cet exemple, lorsque la requête SQL renvoie la valeur 25 ou 26, elle est remplacée par JAVA dans la variable TYPE.

Mécanisme d'extension pour les requêtes de reverse engineering direct

Lors du reverse engineering, PowerAMC exécute des requêtes permettant d'extraire des informations des colonnes des tables système. Le résultat d'une requête est mis en correspondance avec les variables internes PowerAMC via l'en-tête de la requête. Lorsque les tables système d'un SGBD stockent des informations dans des colonnes avec LONG, BLOB, TEXT et d'autres types de données incompatibles, il est impossible de concaténer ces informations dans une chaîne.

Vous pouvez contourner cette limitation en utilisant le mot clé *EX* et en créant des requêtes et des variables personnalisées dans les requêtes de reverse engineering existantes à l'aide de la syntaxe suivante :

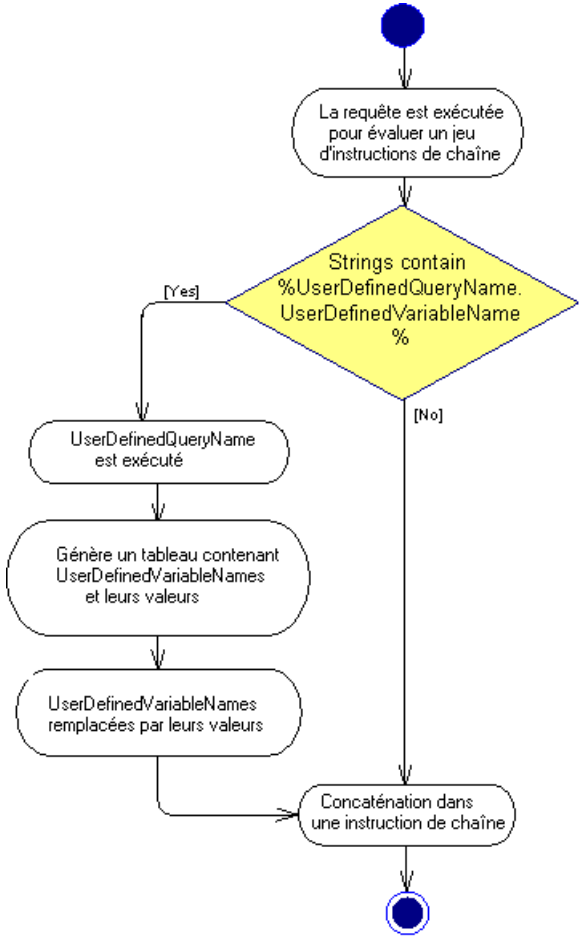
```
%UserDefinedQueryName.UserDefinedVariableName%
```

Ces variables définies par l'utilisateur seront évaluées par des requêtes séparées définies par l'utilisateur.

Dans l'exemple suivant, il est indiqué que OPTIONS contient une requête personnalisée, et nous pouvons constater dans le corps de la requête que l'option 'global partition by range' contient une requête personnalisée appelée :'SqlPartIndexDef', qui recherche les valeurs des variables 'i.owner' et 'i.index_name':

```
{OWNER, TABLE, CONSTNAME, OPTIONS EX}

select
 c.owner,
 c.table_name,
 c.constraint_name,
    ...
    'global partition by range
        (%SqlPartIndexDef.'||i.owner||i.index_name||'%)',
    ...
```

Le graphique suivant illustre le processus d'évaluation de variable lors du reverse engineering :



Remarque : Les requêtes étendues ne doivent pas être définies dans l'entrée ReversedQueries.

Etape 1

Une requête est exécutée pour évaluer les variables dans un jeu d'instructions de chaîne.

Si l'en-tête de la requête contient le mot clé EX, PowerAMC recherche les requêtes et les variables définies par l'utilisateur à évaluer. Les variables définies par l'utilisateur sont créées pour être remplies de données provenant des colonnes de type de données LONG/BLOB/TEXT....

Vous pouvez créer des requêtes définies par l'utilisateur dans une requête de reverse engineering direct. Assurez-vous que chaque requête a un nom unique.

Etape 2
L'exécution de la requête définie par l'utilisateur doit générer un jeu de résultats numérotés contenant autant de paires de variable définie par l'utilisateur (sans %) et de valeur de variable que nécessaire, s'il existe des variables à évaluer.

Par exemple, dans le jeu de résultats suivant, la requête a renvoyé trois lignes et 4 colonnes par ligne :

Variable 1	1	Variable 2	2
Variable 3	3	Variable 4	4
Variable 5	5	Variable 6	6

Etape 3
Les noms des variables définies par l'utilisateur sont remplacés par leurs valeurs.

Les sections suivantes expliquent les requêtes utilisateur définies pour remédier aux limitations du reverse engineering.

Reverse engineering direct d'options physiques

Lors du reverse engineering, les options physiques sont concaténées dans une seule instruction de chaîne. Toutefois, lorsque les tables système d'une base de données sont partitionnées (comme dans Oracle) ou fragmentées (comme dans Informix), les partitions/fragments partagent les mêmes attributs logiques, mais leurs propriétés physiques, telles que les spécifications de stockage, sont conservées dans chaque partition/fragment de la base de données. Les colonnes dans les partitions/fragments ont un type de données (LONG) qui permet le stockage de grandes quantités d'informations binaires non structurées.

Les options physiques dans ces colonnes ne pouvant pas être concaténées dans une instruction de chaîne lors du reverse engineering, `SqlOptsQuery` (catégorie Tables dans le SGBD) contient un appel à une requête définie par l'utilisateur qui va évaluer ces options physiques.

Dans Informix SQL 9, `SqlOptsQuery` est fourni par défaut avec les requêtes et variables utilisateur suivantes (le code suivant est un sous-ensemble de `SqlOptsQuery`) :

```
select
   t.owner,
   t.tabname,
   '%SqlFragQuery.FragSprt'||f.evalpos||'% %FragExpr'||f.evalpos||'%
in %FragDbsp'||f.evalpos||'% ',
   f.evalpos
from
   informix.systables t,
   informix.sysfragments f
where
   t.partnum = 0
   and t.tabid=f.tabid
```

```
[  and t.owner = '%SCHEMA%']
[  and t.tabname='%TABLE%']
```

A l'issue de l'exécution de `SqlOptsQuery`, la requête définie par l'utilisateur `SqlFragQuery` est exécutée pour évaluer `FragDbsp` n, `FragExpr` n, et `FragSprt` n. n représente `evalpos` qui définit la position du fragment dans la liste de fragmentation. n permet d'affecter des noms uniques aux variables, quel que soit le nombre de fragments définis dans la table.

`FragDbsp` n, `FragExpr` n, et `FragSprt` n sont des variables utilisateur qui seront évaluées pour récupérer des informations concernant les options physiques des fragments dans la base de données :

Variable utilisateur	Options physiques
FragDbsp n	Emplacement du fragment pour le fragment n
FragExpr n	Expression du fragment pour le fragment n
FragSprt n	Séparateur du fragment pour le fragment n

`SqlFragQuery` est défini comme suit :

```
{A, a(E="expression", R="round robin", H="hash"), B, b, C, c, D,
d(0="", *=",")}
select
   'FragDbsp'||f.evalpos, f.dbspace,
   'FragExpr'||f.evalpos, f.exprtext,
   'FragSprt'||f.evalpos, f.evalpos
from
   informix.systables t,
   informix.sysfragments f
where
   t.partnum = 0
   and f.fragtype='T'
   and t.tabid=f.tabid
[  and t.owner = '%SCHEMA%']
[  and t.tabname='%TABLE%']
```

L'en-tête de `SqlFragQuery` contient les noms de variable suivants.

```
{A, a(E="expression", R="round robin", H="hash"), B, b, C, c, D,
d(0="", *=",")}
```

Seules les règles de conversion définies entre crochets seront utilisées lors de la concaténation de chaîne : "FragSprt0", qui contient 0 (f.evalpos), sera remplacé par " ", et "FragSprt1", qui contient 1, sera remplacé par ","

`SqlFragQuery` génère un jeu de résultats numérotés contenant autant de paires de nom de variable utilisateur (sans %) et de valeurs de variable que nécessaire, s'il existe de nombreuses variables à évaluer.

Les noms de variable définies par l'utilisateur sont remplacés par leur valeur dans l'instruction de chaîne pour les options physiques des fragments dans la base de données.

Reverse engineering direct d'index basés sur une fonction

Dans Oracle 8i et versions ultérieures, vous pouvez créer des index basés sur des fonctions et des expressions qui impliquent une ou plusieurs colonnes dans la table en cours d'indexation. Un index basé sur une fonction précalcule la valeur de la fonction ou de l'expression et la stocke dans l'index. La fonction ou l'expression va remplacer la colonne d'index dans la définition de l'index.

Une colonne d'index avec une expression est stockée dans les tables système ayant un type de données LONG qui ne peut pas être concaténé dans une instruction de chaîne lors du reverse engineering.

Pour contourner cette limitation, `SqlListQuery` (catégorie Index dans le SGBD) contient un appel vers la requête définie par l'utilisateur `SqlExpression` utilisée pour récupérer l'expression d'index dans une colonne ayant le type de données LONG et pour concaténer cette valeur dans une instruction de chaîne (le code suivant est un sous-ensemble de `SqlListQuery`) :

```
select
   '%SCHEMA%',
   i.table_name,
   i.index_name,
   decode(i.index_type, 'BITMAP', 'bitmap', ''),
   decode(substr(c.column_name, 1, 6), 'SYS_NC',
'%SqlExpression.Xpr'||i.table_name||i.index_name||
c.column_position||'%', c.column_name)||' '||c.descend||', ',
   c.column_position
from
   user_indexes i,
   user_ind_columns c
where
   c.table_name=i.table_name
   and c.index_name=i.index_name
[  and i.table_owner='%SCHEMA%']
[  and i.table_name='%TABLE%']
[  and i.index_name='%INDEX%']
```

L'exécution de `SqlListQuery` appelle l'exécution de la requête définie par l'utilisateur `SqlExpression`.

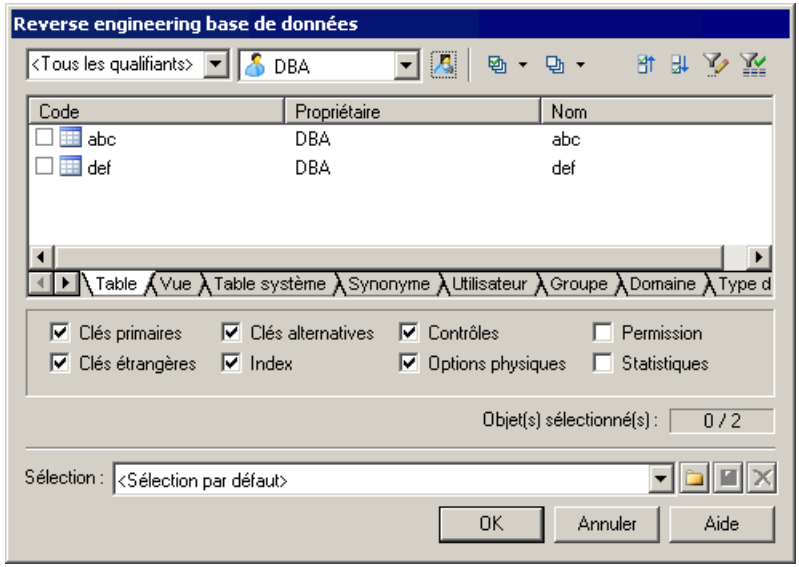
`SqlExpression` est suivi d'une variable définie par l'utilisateur comme suit :

```
{VAR, VAL}

select
   'Xpr'||table_name||index_name||column_position,
   column_expression
from
   all_ind_expressions
where 1=1
[  and table_owner='%SCHEMA%']
[  and table_name='%TABLE%']
```

Le nom de la variable définie par l'utilisateur est unique, il s'agit du résultat de la concaténation de "Xpr", du nom de table, du nom d'index et de la position de colonne.

Qualifiants et reverse engineering direct

Le qualifiant d'objet est affiché dans la liste dans l'angle supérieur gauche de la boîte de dialogue Reverse engineering de base de données. Vous pouvez utiliser un qualifiant pour sélectionner les objets sur lesquels faire porter le reverse engineering.



Vous pouvez ajouter une section relative aux qualifiants lorsque vous personnalisez votre SGBD. Cette section doit contenir les entrées suivantes :

- enable: YES/NO
- SqlListQuery (script) : cette entrée contient la requête SQL qui est exécutée pour extraire la liste des qualifiants. Vous ne devez pas ajouter d'en-tête à cette requête

L'effet de ces entrées est affiché dans le tableau ci-dessous :

Activé	SqlListQuery présent ?	Résultat
Yes	Yes	Les qualifiants sont disponibles et peuvent être sélectionnés. Sélectionnez-en si nécessaire. Vous pouvez également saisir le nom d'un qualifiant. SqlListQuery est exécuté pour remplir la liste des qualifiants
	No	Seule la valeur par défaut (Tous les qualifiants) est sélectionnée. Vous pouvez également saisir le nom d'un qualifiant
No	No	La liste est grisée

Exemple

Dans Adaptive Server Anywhere 7, une requête de qualifiant typique se présente comme suit :

```
.Qualifier.SqlListQuery :
select dbspace_name from sysfile
```

Génération et reverse engineering d'objets étendus

Certains SGBD incluent des objets qui ne peuvent pas être représentés par les objets du modèle PowerAMC standard. Toutefois, vous pouvez travailler avec ces objets, les générer ou procéder à leur reverse engineering via des objets étendus. Pour ce faire, vous devez commencer par créer un objet étendu, puis définir ses scripts de génération et de reverse engineering.

Création d'un objet étendu

Vous pouvez créer un objet étendu dans un SGBD.

1. Sélectionnez **SGBD > Editer le SGBD courant** pour afficher la feuille de propriétés du SGBD, puis développez la catégorie **Profile** dans le volet de gauche.
2. S'il n'existe pas d'entrée ExtendedObject dans cette catégorie, vous devez la créer en pointant sur **Profile**, en cliquant le bouton droit de la souris, puis en sélectionnant **Ajouter des métaclasses** dans le menu contextuel. Dans la fenêtre Sélection des métaclasses, cliquez sur le sous-onglet **PdCommon**, sélectionnez **ExtendedObject**, puis cliquez sur **OK** pour ajouter cette métaclasse dans la liste des objets.
3. Pointez sur l'entrée **ExtendedObject**, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau > Stéréotype** dans le menu contextuel pour créer un nouveau stéréotype, qui sera utilisé pour définir votre nouvel objet.
4. Spécifiez le nom de votre nouvel objet et cochez la case **Utiliser comme métaclasse**. Ce nouvel objet apparaîtra ainsi dans les menus de PowerAMC et fera l'objet de sa propre section dans l'Explorateur d'objets.

   Vous pouvez ajouter des attributs à l'objet, créer des templates pour définir sa forme pour la génération et le reverse engineering, et produire des formulaires personnalisés à utiliser dans des feuilles de propriétés. Pour plus d'informations, voir *Chapitre 2, Fichiers d'extension* à la page 23.

Une fois que vous avez défini votre objet, vous devez activer sa génération.

Définition de scripts de génération et de reverse engineering pour un objet étendu

Vous pouvez définir des scripts de génération et de reverse engineering pour un objet étendu.

1. Pointez sur l'entrée Script/Objects, cliquez le bouton droit de la souris, puis sélectionnez Ajouter des éléments dans le menu contextuel afin d'afficher une boîte de dialogue de sélection qui répertorie tous les objets disponibles dans le modèle.
2. Sélectionnez le nouvel objet étendu dans la liste, puis cliquez sur OK pour l'ajouter à la liste des objets.
3. Pointez sur l'entrée du nouvel objet, cliquez le bouton droit de la souris, puis sélectionnez Ajouter des éléments dans le menu contextuel afin d'afficher une boîte de dialogue de sélection qui répertorie tous les éléments de script qui peuvent être ajoutés à un objet étendu.
4. Pour activer la génération et le reverse engineering de l'objet, vous devez au minimum sélectionner les éléments suivants :
   - Create
   - Drop
   - AlterStatementList
   - SqlAttrQuery
   - SqlListQuery
5. Cliquez sur OK pour ajouter ces éléments de script à votre objet. Vous allez devoir spécifier des valeurs pour chacun de ces éléments. Pour plus d'informations, et pour obtenir de l'aide sur la syntaxe, reportez-vous à la section *Eléments communs aux différents objets* à la page 171.
6. Votre objet est maintenant disponible pour la génération et le reverse engineering. Vous pouvez également contrôler l'ordre dans lequel cet objet, ainsi que les autres objets, seront générés. Pour plus d'informations, voir la section *GenerationOrder – personnalisation de l'ordre de génération des objets* à la page 169.

Ajout de scripts avant ou après la génération ou le reverse engineering

Vous pouvez spécifier des scripts à utiliser avant ou après la génération ou le reverse engineering de base de données.

1. Ouvrez le dossier Profile. S'il n'y a pas d'élément pour Model, pointez sur le dossier Profile, cliquez le bouton droit de la souris, puis sélectionnez Ajouter des métaclasses dans le menu contextuel afin d'afficher une boîte de dialogue de sélection de métaclasse.
2. Sur le sous-onglet PdPDM, sélectionnez Model, puis cliquez sur OK pour revenir à l'éditeur de propriétés de SGBD. L'élément Model s'affiche maintenant dans le dossier Profile.

3. Pointez sur l'élément Model, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau > Gestionnaire d'événement** dans le menu contextuel pour afficher la boîte de dialogue Sélection.
4. Sélectionnez un ou plusieurs des gestionnaires d'événement suivants, en fonction de l'emplacement auquel vous souhaitez ajouter le script :
   - BeforeDatabaseGenerate
   - AfterDatabaseGenerate
   - BeforeDatabaseReverseEngineer
   - AfterDatabaseReverseEngineer
5. Cliquez sur OK pour revenir à l'éditeur de propriétés de SGBD. Les gestionnaires d'événement sélectionnés s'affichent maintenant sous l'élément Model.
6. Sélectionnez successivement les différents gestionnaires d'événement appropriés, cliquez sur leur onglet Script du gestionnaire d'événement, puis saisissez le script souhaité.
7. Cliquez sur OK pour confirmer vos changements et revenir au modèle.

Catégorie General

La catégorie General se trouve immédiatement sous la racine et contient les éléments suivantes :

Elément	Description
EnableCheck	Détermine si la génération des paramètres de contrôle est autorisée ou non. Les valeurs possibles sont les suivantes : • Yes - Paramètres de contrôle générés • No - Toutes les variables liées aux paramètres de contrôle ne seront pas évaluées lors des processus de génération et de reverse engineering
Enable Constname	Spécifie si des noms de contraintes sont utilisés lors de la génération. Les valeurs possibles sont les suivantes : • Yes - Les noms de contrainte sont utilisés lors de la génération • No - Les noms de contrainte ne sont pas utilisés
EnableIntegrity	Spécifie si le SGBD contient des contrainte d'intégrité. Les valeurs possibles sont les suivantes : • Yes - Les cases relatives aux clés primaires, clés alternatives et clés étrangères sont disponibles pour la génération et la modification de base de données • No - Les cases relatives aux clés primaires, clés alternatives et clés étrangères sont grisées et non disponibles pour la génération et la modification de base de données

Elément	Description
EnableMulti Check	Spécifie si la génération de plusieurs paramètres de contrôle pour les tables et colonnes est autorisée ou non. Les valeurs possibles sont les suivantes : • Yes - Plusieurs paramètres de contrôle sont générés. La première contrainte dans le script correspond à la concaténation de toutes les règles de validation, les autres contraintes correspondent à chaque règle de gestion de contrainte attachée à un objet • No - Toutes les règles de gestion (validation et contrainte) sont concaténées dans une même expression de contrainte
SqlSupport	Spécifie si la syntaxe SQL est admise. Les valeurs possibles sont les suivantes : • Yes - La syntaxe SQL est admise et l'aperçu SQL disponible • No - La syntaxe SQL n'est pas admise et l'aperçu SQL n'est pas disponible
UniqConst Name	Spécifie si les noms de contrainte uniques pour les objets sont ou non autorisés. Les valeurs possibles sont les suivantes : • Yes - Tous les noms de contrainte doivent être uniques dans la base de données, y compris les noms d'index • No - Les noms de contrainte doivent être uniques pour un objet La vérification de modèle prend en compte cette entrée lors de la vérification de nom de contrainte.

Catégorie Script/SQL

La catégorie SQL est située dans la catégorie **Racine > Script**. Ses sous-catégories définissent la syntaxe SQL pour le SGBD

Catégorie Syntax

La catégorie Syntax est située dans la catégorie **Racine > Script > SQL**, et contient les éléments suivants qui définissent la syntaxe spécifique du SGBD :

Elément	Description
BlockComment	Spécifie que le caractère est utilisé pour encadrer un commentaire portant sur plusieurs lignes. Exemple : `/*  */`
Block Terminator	Spécifie le caractère de fin de bloc, utilisé pour terminer les instructions telles que les instructions portant sur les triggers et procédures stockées.

Elément	Description
Delimiter	Spécifie le caractère de séparation de champs.
Identifier Delimiter	Spécifie le caractère de séparation d'identifiant. Lorsque les délimiteurs de début et de fin sont différents, ils doivent être séparés par un espace.
LineComment	Spécifie le caractère utilisé pour encadrer un commentaire d'une seule ligne. Exemple : `%%`
Quote	Spécifie le caractère utilisé pour encadrer les valeurs de chaîne. Le même caractère (apostrophe ou guillemet) doit être utilisé dans les onglets de paramètres de contrôle pour encadrer les mots réservés utilisés comme valeur par défaut.
SqlContinue	Spécifie le caractère de suite. Certaines bases de données requièrent un caractère de suite lorsqu'une instruction dépasse une ligne. Pour connaître le caractère approprié, reportez-vous à la documentation relative à votre SGBD. Ce caractère est attaché à chaque ligne, juste avant le caractère de saut de ligne.
Terminator	Spécifie le caractère de fin d'instruction, utilisé pour terminer les instructions telles que les instructions de création de table, de vue ou d'index, ou bien pour les instructions d'ouverture/fermeture de base de données. Si aucune valeur n'est spécifiée, c'est `BlockTerminator` qui est utilisé.
UseBlockTerm	Spécifie la syntaxe d'utilisation du `BlockTerminator`. Les valeurs possibles sont les suivantes : • Yes - `BlockTerminator` est toujours utilisé • No - `BlockTerminator` est utilisé pour les triggers et les procédures stockées uniquement

Catégorie Format

La catégorie Format est située dans la catégorie **Racine > Script > SQL**, et contient les éléments suivants qui définissent la mise en forme du script :

Elément	Description
AddQuote	Détermine si les codes d'objet sont systématiquement placés entre apostrophes ou guillemets lors de la génération. Les valeurs possibles sont les suivantes : • Yes – Les codes d'objet sont systématiquement placés entre apostrophes ou guillemets lors de la génération • No - Les codes d'objet sont générés sans apostrophes ou guillemets

Elément	Description
CaseSensitivity UsingQuote	Détermine si la sensibilité à la casse est gérée à l'aide de guillemets. Vous devez définir cette valeur booléenne à Yes si le SGBD que vous utilisez nécessite des guillemets pour préserver la casse des codes d'objet.
Format de date et d'heure	Voir *Format de date et d'heure* à la page 164.
EnableOwner Prefix / EnableDtbs Prefix	Spécifie que les codes d'objet peuvent être préfixés par le nom du propriétaire de l'objet, le nom de la base de données, ou les deux, à l'aide de la variable %QUALIFIER%. Les valeurs possibles sont les suivantes : • Yes – active les cases Préfixe de base de données et Préfixe de propriétaire dans la boîte de dialogue de génération. Sélectionnez l'une de ces options, ou les deux, pour préfixer les objets. Si vous sélectionnez les deux, le propriétaire et la base de données sont concaténés lors de l'évaluation de %QUALIFIER%. • No - Les options Préfixe de base de données et Préfixe de propriétaire sont indisponibles

IllegalChar	[génération uniquement] Spécifie les caractères incorrects pour les noms. Si le code contient un caractère illégal, il est défini entre guillemets lors de la génération.
	Exemple :
	`+-*/!=<>'"()`
	Si le nom de la table est "SALES+PROFITS", l'instruction create générée sera :
	`CREATE TABLE "SALES+PROFITS"`
	Des guillemets sont placés de part et d'autre du nom de table pour indiquer qu'un caractère incorrect est utilisé. Lors du reverse engineering, tout caractère illégal est considéré comme séparateur à moins qu'il ne soit situé dans un nom entre guillemets.

LowerCase Only	Lorsque vous générez un script, tous les objets sont générés en minuscules indépendamment des conventions de dénomination du modèle et des codes de MPD. Les valeurs possibles sont les suivantes : • Yes - Force tous les caractères du script généré en minuscules • No - Génère tout le script sans changer la façon dont les objets sont écrits dans le modèle
MaxScriptLen	Spécifie la longueur maximale d'une ligne de script.

Elément	Description
UpperCase Only	Lorsque vous générez un script, tous les objets sont générés en majuscules indépendamment des conventions de dénomination du modèle et des codes de MPD. Les valeurs possibles sont les suivantes : • Yes - Force tous les caractères du script généré en majuscules • No - Génère tout le script sans changer la façon dont les objets sont écrits dans le modèle Notez que `UpperCaseOnly` et `LowerCaseOnly` sont mutuellement exclusifs. Dans le cas ou ces deux éléments sont activés, le script est écrit en *minuscules*.

Format de date et d'heure

Vous pouvez personnaliser le format de date et d'heure pour la génération de données de test par script ou directe en utilisant des éléments de SGBD contenus dans la catégorie Format.

PowerAMC utilise la table de correspondance `PhysDataType` dans la catégorie Script \Data types afin de convertir les types de données physiques des colonnes en types de données conceptuels car les entrées de SGBD sont liées aux types de données conceptuels.

Exemple pour Sybase AS Anywhere 7 :

Type de données physique	Type de données conceptuel	Entrée de SGBD utilisée pour SQL	Entrée de SGBD utilisée pour la connexion directe
datetime	DT	DateTimeFormat	OdbcDateTimeFormat
timestamp	TS	DateTimeFormat	OdbcDateTimeFormat
date	D	DateFormat	OdbcDateFormat
time	T	TimeFormat	OdbcTimeFormat

Si vous souhaitez personnaliser le format de date et d'heure pour votre génération de données de test, vous devez vérifier le type de données des colonnes dans votre SGBD, puis trouver le type de données conceptuel correspondant afin de savoir quelle entrée personnaliser dans votre SGBD. Par exemple, si les colonnes utilisent les données Date & heure dans votre modèle, vous devez personnaliser l'entrée DateTimeFormat dans votre SGBD.

Le format par défaut pour la date et l'heure est le suivant :

- SQL: `'yyyy-mm-dd HH:MM:SS'`
- Connexion directe : `{ts 'yyyy-mm-dd HH:MM:SS'}`

Dans lequel :

Format	Description
yyyy	Année sur quatre chiffres
yy	Année sur deux chiffres
mm	Mois
dd	Jour
HH	Heure
MM	Minute
SS	Seconde

Par exemple, vous pouvez définir la valeur suivante pour l'entrée DateTimeFormat pour SQL : `yy-mm-dd HH:MM.` Pour la connexion directe, cette entrée doit avoir la valeur suivante : `{ts 'yy-mm-dd HH:MM'}.`

Catégorie File

La catégorie File est située dans la catégorie **Racine > Script > SQL**, et contient les éléments suivants qui définissent la mise en forme du script :

Elément	Description
AlterHeader	Spécifie le texte d'en-tête pour un script de génération de base de données.
AlterFooter	Spécifie le texte de fin pour un script de génération de base de données.
EnableMulti File	Spécifie que l'utilisation de scripts multiples est admise. Les valeurs possibles sont les suivantes : • Yes – Active la case à cocher Un seul fichier dans les boîtes de dialogue de génération de base de données, de génération de triggers et procédures et de modification de base de données. Si vous désélectionnez cette option, un script distinct est créé pour chaque table (portant le même nom que la table, et avec un suffixe défini dans l'élément `TableExt`), et un script global récapitule tous les éléments de script de table individuels. • La case à cocher Un seul fichier est indisponible, et un seul script inclut toutes les commandes. Le nom de fichier du script global est personnalisable dans la zone Nom de fichier des boîtes de dialogue de génération ou de modification et le suffixe est spécifié dans l'élément `ScriptExt`. Le nom par défaut pour le script global est CREBAS pour la génération de base de données, CRETRG pour la génération des triggers et procédures stockées, et ALTER pour la modification de base de données.
Footer	Spécifie le texte de fin pour un script de génération de base de données.

Elément	Description
Header	Spécifie le texte d'en-tête pour un script de génération de base de données.
ScriptExt	Spécifie le suffixe de script par défaut lorsque vous générez une base de données ou modifiez une base de données pour la première fois. Exemple : `sql`
StartCommand	Spécifie l'instruction d'exécution d'un script. Utilisé dans le fichier d'en-tête d'une génération multifichiers afin d'appeler tous les autres fichiers générés depuis le fichier d'en-tête. Exemple (Sybase ASE 11) : `isql %NAMESCRIPT%` Correspond à la variable %STARTCMD% (voir *Variables de MPD* à la page 247).
TableExt	Spécifie le suffixe des scripts utilisés pour générer chaque table lorsque l'élément EnableMultiFile est activé et que la case "Un seul fichier" n'est pas cochée dans la boîte de dialogue de génération ou de modification. Exemple : `sql`
TrgFooter	Spécifie le texte de fin pour un script (génération de triggers et de procédures).
TrgHeader	Spécifie le texte d'en-tête pour un script (modification de base de données).
TrgUsage1	[lorsque vous utilisez un seul script] Spécifie le texte à afficher dans la fenêtre Résultats à la fin de la génération de trigger et de procédure.
TrgUsage2	[lorsque vous utilisez plusieurs scripts] Spécifie le texte à afficher dans la fenêtre Résultats à la fin de la génération de trigger et de procédure.
TriggerExt	Spécifie le suffixe du script principal lorsque vous générez des triggers et des procédures stockées pour la première fois. Exemple : `trg`
Usage1	[lorsque vous utilisez un seul script] Spécifie le texte à afficher dans la fenêtre Résultats à la fin de la génération de la base de données.
Usage2	[lorsque vous utilisez plusieurs scripts] Spécifie le texte à afficher dans la fenêtre Résultats à la fin de la génération de base de données.

Catégorie Keywords

La catégorie Keywords est située dans la catégorie **Racine > Script > SQL**, et contient les éléments suivants qui réservent des mots clés.

Les listes des fonctions et opérateurs SQL sont utilisées pour remplir l'éditeur de code SQL de PowerAMC SQL afin de proposer des listes de fonctions disponibles pour aider à la saisie de code SQL.

Elément	Description

CharFunc	Spécifie une liste de fonctions SQL à utiliser avec des caractères et des chaînes.
	Exemple :
	`char()`
	`charindex()`
	`char_length() etc`

Commit	Spécifie une instruction de validation de transaction par une connexion directe.

ConvertFunc	Spécifie une liste de fonctions SQL à utiliser pour convertir des valeurs entre hex et integer et pour gérer les chaînes.
	Exemple :
	`convert()`
	`hextoint()`
	`inttohex() etc`

DateFunc	Spécifie une liste de fonctions SQL à utiliser avec les dates.
	Exemple :
	`dateadd()`
	`datediff()`
	`datename() etc`

GroupFunc	Spécifie une liste de fonctions SQL à utiliser avec des mots clés de regroupement.
	Exemple :
	`avg()`
	`count()`
	`max() etc`

ListOperators	Spécifie une liste d'opérateurs SQL à utiliser pour comparer des valeurs, des booléens et divers opérateurs sémantiques.
	Exemple :
	`=`
	`!=`
	`not like etc`

Elément	Description

NumberFunc	Spécifie une liste de fonctions SQL à utiliser avec des nombres.
	Exemple :
	`abs()`
	`acos()`
	`asin() etc`

OtherFunc	Spécifie une liste de fonctions SQL à utiliser pour les estimations, les concaténations et les vérifications SQL.
	Exemple :
	`db_id()`
	`db_name()`
	`host_id() etc`

Reserved Default	Spécifie une liste de mots clés qui peuvent être utilisés comme valeurs par défaut. Si un mot réservé est utilisé comme valeur par défaut, il n'est pas encadré d'apostrophes.
	Exemple (SQL Anywhere® 10) - USER est une valeur par défaut réservée :
	`Create table CUSTOMER (`
	`Username varchar(30) default USER`
	`)`
	Lorsque vous exécutez ce script, CURRENT DATE est reconnu comme valeur par défaut réservée.

ReservedWord	Spécifie une liste de mots réservés. Si un mot réservé est utilisé comme code d'objet, il est placé entre apostrophes lors de la génération (en utilisant les apostrophes spécifiés dans **SGBD** > **Script** > **SQL** > **Syntax** > > **Quote**).

Catégorie Script/Objects

La catégorie Objects est située dans la catégorie **Racine** > **Script** > **SQL** (ainsi, éventuellement, que sous **Racine** > **ODBC** > **SQL**), et contient les éléments suivants qui définissent les objets de base de données qui seront disponibles dans votre modèle.

Commandes pour tous les objets

Les commandes suivantes sont situées sous les catégories **Racine** > **Script** > **Objects** et **Racine** > **ODBC** > **Objects**, et s'appliquent à tous les objets.

MaxConstLen - définition d'une longueur maximale pour le nom de contrainte

Commande permettant de définir la longueur maximale de nom de contrainte prise en charge par la base de données cible. Cette valeur est mise en oeuvre dans la vérification de modèle et

produit une erreur si le code dépasse la valeur définie. Le nom de contrainte est également tronqué au moment de la génération.

Remarque : PowerAMC a une longueur maximale de 254 caractères pour les noms de contrainte. Si votre base de données prend en charge des noms de contrainte plus longs, vous devez définir les noms de contrainte de sorte qu'ils se conforment à la limite de 254 caractères.

EnableOption - activation des options physiques

Commande permettant d'activer les options physiques pour le modèle, les tables, les index, les clés alternatives et autres objets qui sont pris en charge par le SGBD cible. Elle contrôle également la disponibilité de l'onglet Options d'une feuille de propriétés d'objet.

Les valeurs possibles sont les suivantes :

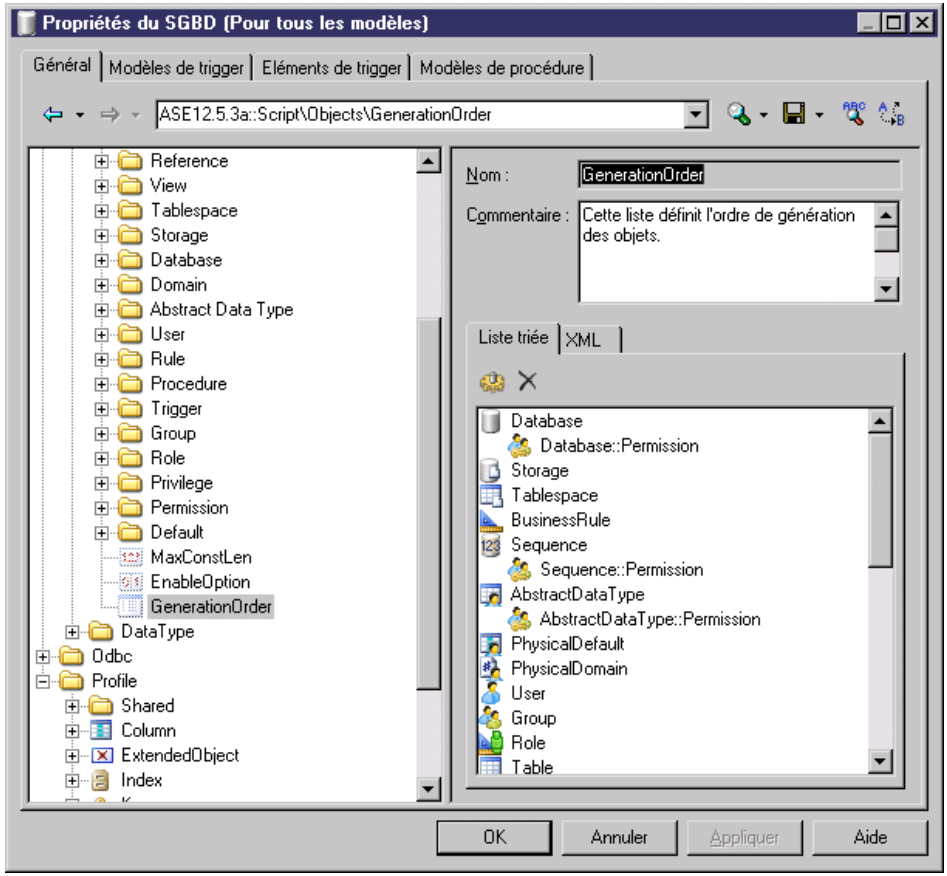
- Yes - L'onglet Options physiques est disponible dans la feuille de propriétés de l'objet.
- No - L'onglet Options physiques n'est pas disponible dans la feuille de propriétés de l'objet.

Pour plus d'informations, voir *Options physiques* à la page 240

GenerationOrder – personnalisation de l'ordre de génération des objets

Commande permettant de spécifier l'ordre de génération des objets. Cette commande est désactivée par défaut.

1. Pointez sur l'entrée Script/Objects, cliquez le bouton droit de la souris, puis sélectionnez Ajouter des éléments dans le menu contextuel afin d'afficher une boîte de dialogue de sélection qui répertorie tous les objets disponibles dans le modèle.
2. Cochez la case GenerationOrder, puis cliquez sur OK. La commande GenerationOrder est activée et ajoutée à la fin de la liste de la catégorie Objects.
3. Cliquez sur l'élément GenerationOrder pour afficher ses propriétés :



4. Vous pouvez faire glisser des entrées dans l'onglet Liste triée afin de spécifier l'ordre dans lequel vous souhaitez que les objets soient créés.
5. Notez que tous les types d'objet ne sont pas inclus dans cette liste par défaut. Vous pouvez ajouter et retirer des éléments de cette liste en utilisant les outils disponibles sur l'onglet. Si un objet ne figure pas dans la liste, il sera généré malgré tout, mais uniquement après les objets présents dans la liste. Les sous-objets, tels que "Sequence::Permissions", peuvent être placés directement sous leur objet parent dans la liste (ils seront affichés en retrait pour illustrer la parenté) ou séparément, auquel cas ils sont affichés sans mise en retrait.
6. Cliquez sur OK pour confirmer vos modifications et revenir au modèle.

Remarque : Par défaut, les objets étendus (voir *Génération et reverse engineering d'objets étendus* à la page 158) ne sont pas automatiquement inclus dans cette liste, et sont générés après tous les autres objets. Pour promouvoir ces objets dans l'ordre de génération, ajoutez-les dans la liste en utilisant les outils de l'onglet, et placez-les à la position souhaitée pour la génération.

Eléments communs aux différents objets

Les éléments suivants sont disponible dans différents objets situés dans la catégorie **Racine** > **Script** > **Objects**.

Elément	Description
Add	Spécifie l'instruction requise pour ajouter l'objet dans l'instruction de création d'un autre objet. Exemple (ajout d'une colonne) : `%20:COLUMN% %30:DATATYPE% [default %DEFAULT%] [%IDENTITY%?identity:[%NULL%][%NOTNULL%]] [[constraint %CONSTNAME%] check (%CONSTRAINT%)]`
AfterCreate/ AfterDrop/ AfterModify	Spécifie les instructions étendues exécutées après les principales instructions Create, Drop ou Modify. Pour plus d'informations, voir *Génération de script* à la page 144.
Alter	Spécifie l'instruction requise pour modifier l'objet.
AlterDBIgnored	Spécifie une liste d'attributs qui doivent être ignorés lors d'une comparaison avant le lancement d'une mise à jour de base de données.
AlterStatementList	Spécifie une liste d'attributs qui, lorsqu'ils sont modifiés, doivent provoquer l'émission d'une instruction alter. Chaque attribut dans la liste est mis en correspondance avec l'instruction alter à utiliser.
BeforeCreate/ BeforeDrop/ BeforeModify	Spécifie les instructions étendues exécutées avant les principales instructions Create, Drop ou Modify. Pour plus d'informations, voir *Génération de script* à la page 144.
ConstName	Spécifie un template de nom de contrainte pour l'objet. Le template contrôle la façon dont le nom de l'objet sera généré. Le template s'applique à tous les objets pour lesquels vous n'avez pas défini de nom de contrainte individuel. Le nom de contrainte qui sera appliqué à un objet est affiché dans sa feuille de propriétés. Exemples (ASE 15) : • Table : CKT_%.U26:TABLE% • Colonne : CKC_%.U17:COLUMN%_%.U8:TABLE% • Clé primaire : PK_%.U27:TABLE%
Create	[génération et reverse engineering] Spécifie l'instruction requise pour créer l'objet. Exemple : `create table %TABLE%`

Elément	Description
DefOptions	Spécifie les valeurs par défaut pour les options physiques qui seront appliquées à tous les objets. Ces valeurs doivent respecter la syntaxe SQL. Exemple : `in default_tablespace` Pour plus d'informations, voir *Options physiques* à la page 240.

Drop	Spécifie l'instruction requise pour supprimer l'objet.
	Exemple (SQL Anywhere 10) :
	`if exists( select 1 from sys.systable`
	`  where table_name=%.q:TABLE%`
	`  and table_type in ('BASE', 'GBL TEMP')[%QUALIFIER%?`
	`  and creator=user_id(%.q:OWNER%)]`
	`) then drop table [%QUALIFIER%]%TABLE%`
	`end if`

Enable	Spécifie si un objet est pris en charge.

EnableOwner	Active la définition des propriétaires pour l'objet. Le propriétaire de l'objet peut ne pas être le propriétaire de la table parent. Les valeurs possibles sont les suivantes :
	• Yes - La liste Propriétaire s'affiche dans la feuille de propriétés de l'objet.
	• No – Les propriétaires ne sont pas pris en charge pour l'objet.
	Notez que dans le cas d'un propriétaire d'index, vous devez vous assurer que l'instructions Create prend en compte le propriétaire de la table et de l'index. Par exemple, dans Oracle 9i, l'instruction Create d'un index se présente comme suit :
	`create [%UNIQUE%?%UNIQUE% :[%INDEXTYPE% ]]index [%QUALIFIER%]%INDEX% on [%CLUSTER%?cluster C_%TABLE%:[%TABLQUALIFIER%]%TABLE% (`
	`  %CIDXLIST%`
	`)]`
	`[%OPTIONS%]`
	%QUALIFIER% fait référence à l'objet courant (index) et %TABLQUALIFIER% fait référence à la table parent de l'index.

EnableSynonym	Active la prise en charge des synonymes pour l'objet.
Footer	Spécifie la fin de l'objet. Le contenu est inséré directement après chaque instructions `create objet`.
Header	Spécifie l'en-tête de l'objet. Le contenu est inséré directement avant chaque instructions `create objet`.

Elément	Description
MaxConstLen	Spécifie la longueur maximale de nom de contrainte prise en charge pour l'objet dans la base de données cible, où cette valeur est différente de la valeur par défaut. Voir aussi *MaxConstLen - définition d'une longueur maximale pour le nom de contrainte* à la page 168).
MaxLen	Spécifie la longueur maximale de code pour un objet. Cette valeur est mise en oeuvre lors de la vérification de modèle et produit une erreur si le code dépasse la valeur définie. Le code d'objet est également tronqué au moment de la génération.
Modifiable Attributes	Spécifie une liste d'attributs étendus qui seront pris en compte dans la boîte de dialogue de fusion lors de la synchronisation de base de données. Pour plus d'informations, voir *Génération de script* à la page 144. Exemple (ASE 12.5) : `ExtTablePartition`
Options	Spécifie les objets physiques relatives à la création d'un objet. Exemple (ASA 6) : `in %s : category=tablespace` Pour plus d'informations, voir *Options physiques* à la page 240.

Permission	Spécifie une liste de permissions disponibles pour l'objet. La première colonne est le nom SQL de la permission (SELECT, par exemple), et la seconde colonne est le nom abrégé qui s'affiche dans le titre des colonnes de grille.
	Exemple (permissions sur les tables dans ASE 15) :
	`SELECT / Sel`
	`INSER / Ins`
	`DELETE / Del`
	`UPDATE / Upd`
	`REFERENCES / Ref`

Reversed Queries	Spécifie une liste de requêtes d'attribut supplémentaires à appeler lors du reverse engineering directe. Pour plus d'informations, voir *Reverse engineering direct de base de données* à la page 148.
Reversed Statements	Spécifie une liste d'instructions supplémentaires qui seront récupérées par reverse engineering. Pour plus d'informations, voir *Reverse engineering de script* à la page 147.

Elément	Description

SqlAttrQuery	Spécifie une requête SQL permettant d'extraire des information supplémentaires sur les objets récupérés via reverse engineering par `SQLListQuery`.
	
	Exemple (Join Index in Oracle 10g) :
	
	`{OWNER ID, JIDX ID,  JIDXWHERE ...}`
	`select index_owner, index_name,`
	`outer_table_owner || '.' || outer_table_name || '.'`
	`|| outer_table_column || '=' || inner_table_owner`
	`|| '.' || inner_table_name || '.' || inner_ta-`
	`ble_column || ','`
	`from all_join_ind_columns`
	`where 1=1`
	`[  and index_owner=%.q:OWNER%]`
	`[  and index_name=%.q:JIDX%]`

SqlListQuery	Spécifie une requête SQL permettant de répertorier des objets dans la boîte de dialogue de reverse engineering. La requête est exécutée pour renseigner les variables d'en-tête et créer des objets en mémoire.
	
	Exemple (Dimension dans Oracle 10g) :
	
	`{ OWNER, DIMENSION }`
	`select d.owner, d.dimension_name`
	`from sys.all_dimensions d`
	`where 1=1`
	`[  and d.dimension_name=%.q:DIMENSION%]`
	`[  and d.owner=%.q:SCHEMA%]`
	`order by d.owner, d.dimension_name`

SqlOptsQuery	Spécifie une requête SQL permettant d'extraire les options physiques d'objet sur les objets récupérés via reverse engineering par `SqlListQuery`. Le résultat de la requête va renseigner la variable %OPTIONS% et doit respecter la syntaxe SQL.
	
	Exemple (Table in SQL Anywhere 10) :
	
	`{OWNER, TABLE, OPTIONS}`
	`select u.user_name, t.table_name,`
	`  'in '+ f.dbspace_name`
	`from sys.sysuserperms u`
	`  join sys.systab t on (t.creator = u.user_id)`
	`  join sys.sysfile f on (f.file_id = t.file_id)`
	`where f.dbspace_name <> 'SYSTEM'`
	`  and t.table_type in (1, 3, 4)`
	`[  and t.table_name = %.q:TABLE%]`
	`[  and u.user_name = %.q:OWNER%]`

Elément	Description

SqlPermQuery	Spécifie une requête SQL permettant de procéder au reverse engineering de permissions accordées sur des tables.
	Exemple (Procédure dans SQL Anywhere 10) :
	`{ GRANTEE, PERMISSION}`
	`select`
	`u.user_name grantee, 'EXECUTE'`
	`from sysuserperms u, sysprocedure s, sysprocperm p`
	`where (s.proc_name = %.q:PROC% ) and`
	`(s.proc_id = p.proc_id) and`
	`(u.user_id = p.grantee)`

Variable par défaut

Dans une colonne, si la variable par défaut est de type texte ou chaîne, la requête doit extraire la valeur de la variable par défaut entre apostrophes. La plupart des SGBD ajoutent ces apostrophes à la valeur de la variable par défaut. Si le SGBD que vous utilisez n'ajoute pas les apostrophes automatiquement, vous devez les spécifier dans les différentes requêtes à l'aide de la variable par défaut.

Par exemple, dans IBM DB2 UDB 8 pour OS/390, la ligne suivante a été ajoutée dans SqlListQuery afin d'ajouter des apostrophes à la valeur de la variable par défaut :

```
...
 case(default) when '1' then '''' concat defaultvalue concat ''''
when '5' then '''' concat defaultvalue concat '''' else defaultvalue
end,
...
```

Table

La catégorie Table est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les tables sont modélisées pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des tables : • AfterCreate, AfterDrop, AfterModify • BeforeCreate, BeforeDrop, BeforeModify • ConstName • Create, Drop • Enable, EnableSynonym • Header, Footer • Maxlen, MaxConstLen • ModifiableAttributes • Options, DefOptions • Permission • ReversedQueries, ReversedStatements • SqlAttrQuery, SqlListQuery, SqlOptsQuery, SqlPermQuery Pour obtenir une description de chacun de ces éléments, voir *Eléments communs aux différents objets* à la page 171.

AddTableCheck	Spécifie une instruction permettant de personnaliser le script pour modifier les contraintes de table au sein d'une instruction `alter table`.
	Exemple (SQL Anywhere 10) :
	`alter table [%QUALIFIER%]%TABLE%`
	`  add [constraint %CONSTNAME% ]check (%.A:CONSTRAINT%)`

AllowedADT	Spécifie une liste de types de données abstraits sur lesquels une table peut être basée. Cette liste remplit la zone Basé sur de la feuille de propriétés de table. Vous pouvez affecter des types de données abstraits objet aux tables. La table utilise les propriétés du type de données abstrait et les attributs du type de données abstrait deviennent des colonnes de la table. Exemple (Oracle 10g) : `OBJECT`

Elément	Description
AlterTable Footer	Spécifie une instruction qui doit être placée après les instructions `alter table` (et avant le caractère de fin). Exemple : `AlterTableFooter = /* End of alter statement */`
AlterTable Header	Spécifie une instruction qui doit être placée avant les instructions `alter table`. Vous pouvez placer un en-tête alter table dans vos scripts pour les documenter ou dans le cadre d'une logique d'initialisation. Exemple : `AlterTableHeader = /* Table name: %TABLE% */`
DefineTable Check	Spécifie une instruction permettant de personnaliser le script des contraintes de table (vérifications) au sein d'une instruction `create table`. Exemple : `check (%CONSTRAINT%)`

DropTable Check	Spécifie une instruction permettant de supprimer une vérification de table dans une instruction `alter table`.
	Exemple :
	`alter table [%QUALIFIER%]%TABLE%`
	`  delete check`

InsertIdentityOff	Spécifie une instruction permettant d'activer l'insertion de données dans une table contenant une colonne d'identité. Exemple (ASE 15) : `set identity_insert [%QUALIFIER%]%@OBJTCODE% off`
InsertIdentityOn	Spécifie une instruction permettant de désactiver l'insertion de données dans une table contenant une colonne d'identité. Exemple (ASE 15) : `set identity_insert [%QUALIFIER%]%@OBJTCODE% on`

Elément	Description
Rename	[modification] Spécifie une instruction permettant de renommer une table. Si cet élément n'est pas spécifié, le processus de modification de base de données supprime les contraintes de clé étrangère, crée une nouvelle table avec le nouveau nom, insère les lignes de l'ancienne table dans la nouvelle table, et crée les index et contraintes sur la nouvelle table à l'aide de tables temporaires. Exemple (Oracle 10g) : `rename %OLDTABL% to %NEWTABL%` La variable %OLDTABL% est le code de la table avant qu'elle ne soit renommée. La variable %NEWTABL% est le nouveau code de la table.

SqlChckQuery	Spécifie une requête SQL Query permettant de procéder au reverse engineering des vérifications de table.
	
	Exemple (SQL Anywhere 10) :
	
	`{OWNER, TABLE, CONSTNAME, CONSTRAINT}`
	`select u.user_name, t.table_name,`
	`k.constraint_name,`
	`case(lcase(left(h.check_defn, 5))) when 'check'`
	`then substring(h.check_defn, 6) else h.check_defn`
	`end`
	`from sys.sysconstraint k`
	`join sys.syscheck h on (h.check_id = k.cons-`
	`traint_id)`
	`join sys.systab t on (t.object_id = k.table_ob-`
	`ject_id)`
	`join sys.sysuserperms u on (u.user_id = t.creator)`
	`where k.constraint_type = 'T'`
	`and t.table_type in (1, 3, 4)`
	`[ and u.user_name = %.q:OWNER%]`
	`[ and t.table_name = %.q:TABLE%]`
	`order by 1, 2, 3`

Elément	Description

SqlListRefr Tables	Spécifie une requête SQL utilisée pour répertorier les tables référencées par une table.
	Exemple (Oracle 10g) :
	`{OWNER, TABLE, POWNER, PARENT}`
	`select c.owner, c.table_name, r.owner,`
	`r.table_name`
	`from sys.all_constraints c,`
	`sys.all_constraints r`
	`where (c.constraint_type = 'R' and c.r_cons-`
	`traint_name = r.constraint_name and c.r_owner =`
	`r.owner)`
	`[ and c.owner = %.q:SCHEMA%]`
	`[ and c.table_name = %.q:TABLE%]`
	`union select c.owner, c.table_name,`
	`r.owner, r.table_name`
	`from sys.all_constraints c,`
	`sys.all_constraints r`
	`where (r.constraint_type = 'R' and r.r_cons-`
	`traint_name = c.constraint_name and r.r_owner =`
	`c.owner)`
	`[ and c.owner = %.q:SCHEMA%]`
	`[ and c.table_name = %.q:TABLE%]`

SqlListSchema	Spécifie une requête utilisée pour extraire un schéma enregistré dans la base de données. Cet élément est utilisé avec les tables de type XML (une référence à un document XML stocké dans la base de données). Lorsque vous définissez une table XML, vous devez extraire les documents XML enregistrés dans la base de données afin d'affecter un document à la table, ce qui se fait à l'aide de la requête SqlListSchema. Exemple (Oracle 10g) : `SELECT schema_url FROM dba_xml_schemas`
SqlStatistics	Spécifie une requête SQL utilisée pour procéder au reverse engineering des statistiques de colonne et de table. Voir SqlStatistics dans *Column* à la page 181.
SqlXMLTable	Spécifie une sous-requête utilisée pour améliorer les performances de SqlAttrQuery (voir *Eléments communs aux différents objets* à la page 171).

Elément	Description
TableComment	[génération et reverse engineering] Spécifie une instruction permettant d'ajouter un commentaire de table. Si cet élément n'est pas spécifié, la case à cocher Commentaire sur les sous-onglets Tables et Vues de la boîte de dialogue de génération de base de données n'est pas disponible. Exemple (Oracle 10g) : `comment on table [%QUALIFIER%]%TABLE% is %.q:COMMENT%` La variable %TABLE% représente le nom de la table tel que défini dans la boîte de dialogue Liste des tables, ou dans la feuille de propriétés de table. La variable %COMMENT% représente le commentaire défini dans la zone Commentaire de la feuille de propriétés de table.
TypeList	Spécifie une liste de types (par exemple, SGBD : relationnel, objet, XML) pour les tables. Cette liste remplit la liste Type dans la feuille de propriétés de table. Le type XML doit être utilisé avec l'élément SqlListSchema.
UniqConstraint Name	Spécifie si l'utilisation du même nom pour un index et une contrainte sur une même table est possible. Les valeurs possibles sont les suivantes : • Yes – Le nom de contrainte et le nom d'index de la table doivent être différents, ce qui sera contrôlé pendant la vérification du modèle • No - Le nom de contrainte et le nom d'index de la table peuvent être identiques

Column

La catégorie Column est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les colonnes sont modélisées pour votre SGBD.

Elément	Description
[Common items]	Les éléments suivants communs aux différents objets peuvent être définis pour des colonnes : • Add • AfterCreate, AfterDrop, AfterModify • BeforeCreate, BeforeDrop, BeforeModify • ConstName • Create, Drop • Enable • Maxlen, MaxConstLen • ModifiableAttributes • Options, DefOptions • Permission • ReversedQueries, ReversedStatements • SqlAttrQuery, SqlListQuery, SqlOptsQuery, SqlPermQuery Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.

AddColnCheck	Spécifie une instruction permettant de personnaliser le script afin de modifier les contraintes de colonne au sein d'une instruction alter table.
	Exemple (Oracle 10g) :
	`alter table [%QUALIFIER%]%TABLE%`
	` add [constraint %CONSTNAME%] check (%.A:CONSTRAINT%)`

AlterTableAdd Default	Spécifie une instruction permettant de définir la valeur par défaut d'une colonne dans une instruction alter. Exemple (SQL Server 2005) : `[[ constraint %ExtDeftConstName%] default %DEFAULT% ]for %COLUMN%`

Elément	Description
AltEnableAdd ColnChk	Spécifie si une contrainte de vérification de colonne, construite à partir des paramètres de contrôle de la colonne, peut ou non être ajoutée dans une table à l'aide de l'instruction `alter table`. Les valeurs possibles sont les suivantes : • Yes - `AddColnChck` peut être utilisé pour modifier la contrainte de vérification de colonne dans une instruction `alter table`. • No - PowerAMC copie les données dans une table temporaire avant de recréer la table avec les nouvelles contraintes. Voir aussi `AddColnChck`.
AltEnableTS Copy	Permet l'utilisation de colonnes timestamp dans les instructions insert.
Bind	Spécifie une instruction permettant de lier une règle à une colonne. Exemple (ASE 15) : `[%R%?[exec ]][execute ]sp_bindrule [%R%?['[%QUALIFIER%]%RULE%'][[%QUALIFIER%]%RULE%]:['[%QUALIFIER%]%RULE%']], '%TABLE%.%COLUMN%'`
CheckNull	Spécifie si une colonne peut être NULL.
Column Comment	Spécifie une instruction permettant d'ajouter un commentaire à une colonne. Exemple : `comment on column [%QUALIFIER%]%TABLE%.%COLUMN% is %.q:COMMENT%`
DefineColn Check	Spécifie une instruction permettant de personnaliser le script des contraintes de colonne (vérifications) au sein d'une instruction `create table`. Cette instruction est appelée si les instructions create, add, ou alter contiennent %CONSTDEFN%. Exemple : `[constraint %CONSTNAME%] check (%CONSTRAINT%)`

DropColnChck	Spécifie une instruction permettant de supprimer une vérification de colonne dans une instruction `alter table`. Cette instruction est utilisée dans le script de modification de base de données lorsque les paramètres de contrôle ont été supprimés d'une colonne.
	Si `DropColnChck` est vide, PowerAMC copie les données dans une table temporaire avant de recréer la table avec des nouvelles contraintes.
	Exemple (SQL Anywhere 10) :
	`alter table [%QUALIFIER%]%TABLE%`
	`drop constraint %CONSTNAME%`

Elément	Description

DropColnComp	Spécifie une instruction permettant de supprimer une expression calculée de colonne dans une instruction alter table.
	Exemple (SQL Anywhere 10) :
	`alter table [%QUALIFIER%]%TABLE%`
	`alter %COLUMN% drop compute`

DropDefault Constraint	Spécifie une instruction permettant de supprimer une contrainte liée à une colonne définie avec une valeur par défaut
	Exemple (SQL Server 2005) :
	`[%ExtDeftConstName%?alter table [%QUALIFIER%]%TABLE%`
	`drop constraint %ExtDeftConstName%]`

EnableBindRule	Spécifie si les règles de gestion peuvent être liées à des colonnes pour les paramètres de contrôle. Les valeurs possibles sont les suivantes : • Yes - Les éléments Create et Bind de l'objet Rule sont générés • No - La vérification est générée dans la commande Add de colonne
Enable Computed-Coln	Spécifie si les colonnes calculées peuvent être utilisées.

Elément	Description

EnableDefault	Spécifie si les valeurs par défaut prédéfinies sont admises. Les valeurs possibles sont les suivantes :
	• Yes - La valeur par défaut (si elle est définie) est générée pour les colonnes. Elle peut être définie dans les paramètres de contrôle pour chaque colonne. La variable %DEFAULT% contient la valeur par défaut. La case Valeur par défaut pour les colonnes peut être cochée dans les sous-onglets Tables et Vues de la boîte de dialogue de génération de base de données
	• No - La valeur par défaut ne peut pas être générée, et la case Valeur par défaut est indisponible.
	Exemple (AS IQ 12.6) :
	EnableDefault est activé et la valeur par défaut pour la colonne EMPFUNC est Technical Engineer. Le script généré se présente comme suit :
	`create table EMPLOYEE`
	`(`
	`  EMPNUM  numeric(5)    not null,`
	`  EMP_EMPNUM  numeric(5)      ,`
	`  DIVNUM  numeric(5)    not null,`
	`  EMPFNAM   char(30)      ,`
	`  EMPLNAM   char(30)    not null,`
	`  EMPFUNC   char(30)`
	`  default 'Technical Engineer',`
	`  EMPSAL  numeric(8,2)      ,`
	`  primary key (EMPNUM)`
	`);`

Elément	Description
EnableIdentity	Spécifie si le mot clé Identity est pris en charge. Les colonnes Identity sont des compteurs séquentiels gérés par la base de données (par exemple, Sybase et Microsoft SQL Server). Les valeurs possibles sont les suivantes : • Yes - Active la case à cocher Identity dans la feuille de propriétés de la colonne. • No - La case à cocher Identity n'est pas disponible. Lorsque la case Identity est cochée, le mot clé Identity est généré dans le script après le type de données de la colonne. Une colonne Identity ne peut pas être NULL : lorsque vous cochez la case Identity, la case Obligatoire est automatiquement cochée. PowerAMC s'assure que : • Une seule colonne Identity peut être définie par table • Une clé étrangère ne peut pas être une colonne Identity • Identity n'est pris en charge que pour certains types de données. Si la case Identity est cochée pour une colonne dont le type de données n'est pas pris en charge, ce type de données est changé en *numeric*. Si le type de données d'une colonne d'identité est changé en en type de données non pris en charge, PowerAMC affiche un message d'erreur. Notez que, lors de la génération, la variable %IDENTITY% contient la valeur "identity" mais vous pouvez la changer facilement, si nécessaire, en utilisant la syntaxe suivante : `[%IDENTITY%?new identity keyword]`
EnableNotNull WithDflt	Spécifie si les valeurs par défaut sont affectées aux colonnes contenant des valeurs NULL. Les valeurs possibles sont les suivantes : • Yes - La case à cocher With Default est activée dans la feuille de propriétés d'une colonne. Lorsque vous cochez cette case, une valeur par défaut est affectée à une colonne lorsqu'une valeur Null est insérée. • No - La case à cocher With Default n'est pas disponible.

Elément	Description

ModifyColn Chck	Spécifie une instruction permettant de modifier une vérification de colonne dans une instruction `alter table`. Cette instruction est utilisée dans le script de modification de base de données lorsque les paramètres de contrôle d'une colonne ont été modifiés dans la table.
	Si `AddColnChck` est vide, PowerAMC copie les données dans une table temporaire avant de recréer la table avec des nouvelles contraintes.
	Exemple (AS IQ 12.6) :
	`alter table [%QUALIFIER%]%TABLE%`
	`modify %COLUMN% check (%.A:CONSTRAINT%)`
	La variable %COLUMN% est le nom de la colonne définie dans la feuille de propriétés de table. La variable % CONSTRAINT % est la contrainte de vérification construite à partir des nouveaux paramètres de contrôle.
	`AltEnableAddColnChk` doit être défini à YES pour permettre l'utilisation de cette instruction.

ModifyColn Comp	Spécifie une instruction permettant de modifier une expression calculée pour une colonne dans une instruction alter table.
	Exemple (ASA 6) :
	`alter table [%QUALIFIER%]%TABLE%`
	`alter %COLUMN% set compute (%COMPUTE%)`

ModifyColnDflt	Spécifie une instruction permettant de modifier une valeur par défaut de colonne dans une instruction `alter table`. Cette instruction est utilisée dans le script de modification de base de données lorsque la valeur par défaut d'une colonne a été modifiée dans la table.
	Si `ModifyColnDflt` est vide, PowerAMC copie les données dans une table temporaire avant de recréer la table avec des nouvelles contraintes.
	Exemple (ASE 15) :
	`alter table [%QUALIFIER%]%TABLE%`
	`replace %COLUMN% default %DEFAULT%`
	La variable %COLUMN% représente le nom de la colonne tel que défini dans la feuille de propriétés de table. La variable %DEFAULT% représente la valeur par défaut de la colonnes modifiée.

ModifyColnNull	Spécifie une instruction permettant de modifier l'état NULL/non-NULL d'une colonne dans une instruction `alter table`.
	Exemple (Oracle 10g) :
	`alter table [%QUALIFIER%]%TABLE%`
	`modify %COLUMN% %MAND%`

Elément	Description

ModifyColumn	Spécifie une instruction permettant de modifier une colonne. Cette instruction diffère de l'instruction `alter table`, et est utilisée dans un script de modification de base de données lorsque la définition de colonne a été modifiée.
	Exemple (SQL Anywhere 10) :
	`alter table [%QUALIFIER%]%TABLE%`
	`modify %COLUMN% %DATATYPE% %NOTNULL%`

NullRequired	Spécifie le statut obligatoire d'une colonne. Cet élément est utilisé avec la variable de colonne NULLNOTNULL, qui peut prendre la valeur "null" "not null" ou une valeur vide. Pour plus d'informations, voir *Gestion des valeurs Null* à la page 188.

Rename	Spécifie une instruction permettant de renommer une colonne dans une instruction `alter table`.
	Exemple (Oracle 10g) :
	`alter table [%QUALIFIER%]%TABLE%`
	`rename column %OLDCOLN% to %NEWCOLN%`

SqlChckQuery	Spécifie une requête SQL permettant de procéder au reverse engineering de paramètres de contrôle de colonne. Le résultat doit être conforme à la syntaxe SQL appropriée.
	Exemple (SQL Anywhere 10) :
	`{OWNER, TABLE, COLUMN, CONSTNAME, CONSTRAINT}`
	`select u.user_name, t.table_name,`
	`c.column_name, k.constraint_name,`
	`case(lcase(left(h.check_defn, 5))) when 'check'`
	`then substring(h.check_defn, 6) else h.check_defn`
	`end`
	`from sys.sysconstraint k`
	`join sys.syscheck h on (h.check_id = k.cons-`
	`traint_id)`
	`join sys.systab t on (t.object_id = k.table_ob-`
	`ject_id)`
	`join sys.sysuserperms u on (u.user_id = t.creator)`
	`join sys.syscolumn c on (c.object_id = k.ref_ob-`
	`ject_id)`
	`where k.constraint_type = 'C'`
	`[ and u.user_name=%.q:OWNER%]`
	`[ and t.table_name=%.q:TABLE%]`
	`[ and c.column_name=%.q:COLUMN%]`
	`order by 1, 2, 3, 4`

Elément	Description

SqlStatistics	Spécifie une requête SQL permettant de procéder au reverse engineering des statistiques de colonne et de table.
	Exemple (ASE 15) :
	`[%ISLONGDTTP%?{ AverageLength }`
	`select [%ISLONGDTTP%?[%ISSTRDTTP%?`
	`avg(char_length(%COLUMN%)):avg(datalength(%COLUMN`
	`%))]:null] as average_length`
	`from [%QUALIFIER%]%TABLE%`
	`:{ NullValuesRate, DistinctValues, AverageLength }`
	`select`
	`[%ISMAND%?null:(count(*) - count(%COLUMN%)) * 100 /`
	`count(*)]  as null_values,`
	`[%ISMAND%?null:count(distinct %COLUMN%)]  as dis-`
	`tinct_values,`
	`[%ISVARDTTP%?[%ISSTRDTTP%?avg(char_length(%COLUMN`
	`%)):avg(datalength(%COLUMN%))]:null] as avera-`
	`ge_length`
	`from [%QUALIFIER%]%TABLE%]`

Unbind	Spécifie une instruction permettant de faire en sorte qu'une règle ne soit plus liée à une colonne.
	Exemple (ASE 15) :
	`[%R%?[exec ]][execute ]sp_unbindrule '%TABLE%.%CO-`
	`LUMN%'`

Gestion des valeurs Null

L'élément NullRequired spécifie le caractère obligatoire d'une colonne. Cet élément est utilisé avec la variable de colonne NULLNOTNULL, qui peut prendre la valeur "null" "not null" ou une valeur vide. Les combinaisons suivantes sont possibles :

Lorsque la colonne est obligatoire

"not null" est systématiquement généré lorsque NullRequired est défini à True ou False comme illustré dans l'exemple suivant :

```
create domain DOMN_MAND char(33) not null;
create domain DOMN_NULL char(33)     null;

create table TABLE_1
(
 COLN_MAND_1 char(33)  not null,
 COLN_MAND_2 DOMN_MAND not null,
 COLN_MAND_3 DOMN_NULL not null,
);
```

Lorsque la colonne n'est pas obligatoire

- Si NullRequired est défini à True, "null" est généré. L'entrée NullRequired doit être utilisée dans ASE par exemple, puisque la possibilité d'être null ou non y est une option de base de données, et que les mots clés "null" ou "not null" sont requis.

  Dans l'exemple suivant, toutes les valeurs "null" sont générées :

```
create domain DOMN_MAND char(33) not null;
create domain DOMN_MAND char(33)     null;

create table TABLE_1
(
 COLN_NULL_1 char(33)      null,
 COLN_NULL_2 DOMN_NULL     null,
 COLN_NULL_3 DOMN_MAND     null
)
```

- Si NullRequired est défini à False, une chaîne vide est générée. Toutefois, si une colonne attachée à un domaine obligatoire devient non obligatoire, "null" sera généré.

  Dans l'exemple suivant, "null" est généré uniquement pour COLUMN_NULL3 car cette colonne utilise le domaine obligatoire, les autres colonnes générant une chaîne vide :

```
create domain DOMN_MAND char(33) not null;
create domain DOMN_NULL char(33)     null;

create table TABLE_1
(
 COLUMN_NULL1 char(33)          ,
 COLUMN_NULL2 DOMN_NULL         ,
 COLUMN_NULL3 DOMN_MAND     null
);
```

Index

La catégorie Index est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les index sont modélisés pour votre SGBD.

Elément	Description
[Common items]	Les éléments suivants communs aux différents objets peuvent être définis pour des index : • Add • AfterCreate, AfterDrop, AfterModify • BeforeCreate, BeforeDrop, BeforeModify • Create, Drop • Enable, EnableOwner • Header, Footer • Maxlen • ModifiableAttributes • Options, DefOptions • ReversedQueries • ReversedStatements • SqlAttrQuery, SqlListQuery, SqlOptsQuery Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.
AddColIndex	Spécifie une instruction permettant d'ajouter une colonne dans l'instruction `Create Index`. Ce paramètre définit chaque colonne dans la liste des colonnes de l'instruction `Create Index`. Exemple (ASE 15) : `%COLUMN%[ %ASC%]` %COLUMN% représente le code de la colonne tel que défini dans la liste des colonnes de la table. %ASC% représente ASC (ordre ascendant) ou DESC (ordre descendant) en fonction de l'état du bouton radio Tri pour la colonne d'index.
Cluster	Spécifie la valeur qui doit être affectée au mot clé Cluster. Si ce paramètre est vide, la valeur par défaut de la variable %CLUSTER% est CLUSTER.
CreateBefore Key	Contrôle l'ordre de génération des index et des clés. Les valeurs possibles sont les suivantes : • Yes – Les index sont générés avant les clés. • No – Les index sont générés après les clés.

Elément	Description
DefIndexType	Spécifie le type par défaut d'un index. Exemple (DB2) : `Type2`
DefineIndex Column	Spécifie la colonne d'un index.
EnableAscDesc	Active la propriété Tri dans les feuilles de propriétés d'index, qui permet de trier par ordre ascendant ou descendant. Les valeurs possibles sont les suivantes : • Yes – La propriété Tri est activée pour les index, avec Ascendant sélectionné par défaut. La variable %ASC% est calculée. Le mot clé ASC ou DESC est généré lorsque vous créez ou modifiez la base de données • No – Le tri d'index n'est pas pris en charge. Exemple (SQL Anywhere 10) : Un index de clé primaire est créé sur la table TASK, avec la colonne PRONUM triée par ordre ascendant et la colonne TSKNAME par ordre descendant : `create index IX_TASK on TASK (PRONUM asc, TSKNAME desc);`
EnableCluster	Permet la création d'index cluster. Les valeurs possibles sont les suivantes : • Yes - case à cocher Cluster s'affiche dans la feuille de propriétés d'index. • No – L'index ne prend pas en charge les index cluster.
EnableFunction	Permet la création d'index basés sur des fonctions. Les valeurs possibles sont les suivantes : • Yes - Vous pouvez définir des expressions pour les index. • No – L'index ne prend pas en charge les expressions.
IndexComment	Spécifie une instruction permettant d'ajouter un commentaire à un index. Exemple (SQL Anywhere 10) : `comment on index [%QUALIFIER%]%TABLE%.%INDEX% is %.q:COMMENT%`

Elément	Description

IndexType	Spécifie une liste de types d'index disponibles.
	Exemple (IQ 12.6) :
	`CMP`
	`HG`
	`HNG`
	`LF`
	`WD`
	`DATE`
	`TIME`
	`DTTM`

MandIndexType	Spécifie si le type d'index est obligatoire pour les index. Les valeurs possibles sont les suivantes : • Yes – Le type d'index est obligatoire. • No - Le type d'index n'est pas obligatoire.
MaxColIndex	Spécifie le nombre maximum de colonnes pouvant être incluses dans un index. Cette valeur est utilisée lors de la vérification de modèle.

SqlSysIndex Query	Spécifie une requête SQL utilisée pour répertorier les index système créés par la base de données. Ces index sont exclus lors du reverse engineering.
	Exemple (AS IQ 12.6) :
	`{OWNER, TABLE, INDEX, INDEXTYPE}`
	`select u.user_name, t.table_name, i.index_name, i.index_type`
	`from sysindex i, systable t, sysuserperms u`
	`where t.table_id = i.table_id`
	`and u.user_id = t.creator`
	`and i.index_owner != 'USER'`
	`[and u.user_name=%.q:OWNER%]`
	`[and t.table_name=%.q:TABLE%]`
	`union`
	`select u.user_name, t.table_name, i.index_name, i.index_type`
	`from sysindex i, systable t, sysuserperms u`
	`where t.table_id = i.table_id`
	`and u.user_id = t.creator`
	`and i.index_type = 'SA'`
	`[and u.user_name=%.q:OWNER%]`
	`[and t.table_name=%.q:TABLE%]`

UniqName	Spécifie si les noms d'index doivent être uniques dans la portée globale de la base de données. Les valeurs possibles sont les suivantes : • Yes – Les noms d'index doivent être uniques dans la portée globale de la base de données. • No – Les noms d'index doivent être uniques pour chaque objet

Pkey

La catégorie Pkey est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les clés primaires sont modélisées pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des clés primaires : • Add • ConstName • Create, Drop • Enable • Options, DefOptions • ReversedQueries Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.
EnableCluster	Spécifie si les contraintes clustered sont permises sur les clés primaires. • Yes - Les contraintes clustered sont permises. • No - Les contraintes clustered ne sont pas permises.
PkAutoIndex	Détermine si une instruction `Create Index` est générée pour chaque instruction de clé primaire. Les valeurs possibles sont les suivantes : • Yes - Génère automatiquement un index de clé primaire lorsque vous générez l'instruction de clé primaire. Si vous cochez la case Clé primaire sous Création d'index, la case Primaire est automatiquement décochée sous Création de table, et réciproquement. • No - Ne génère pas automatiquement les index de clé primaire. Les cases Clé primaire et Création d'index peuvent être cochées simultanément.
PKeyComment	Spécifie une instruction permettant d'ajouter un commentaire de clé primaire.

Elément	Description

UseSpPrimKey	Spécifie l'utilisation de l'instruction `Sp_primarykey` pour générer des clés primaires. Pour une base de données qui prend en charge la procédure de mise en oeuvre de définition de clé, vous pouvez tester la valeur de la variable correspondante %USE_SP_PKEY% et choisir entre la création d'une clé dans la table et le lancement d'une procédure. Les valeurs possibles sont les suivantes :
	• Yes - L'instruction `Sp_primarykey` est utilisée pour générer des clés primaires.
	• No - Les clés primaires sont générées séparément dans une instruction `alter table`.
	Exemple (ASE 15) :
	Si UseSpPrimKey est activé, l'élément Add pour Pkey contient :
	`UseSpPrimKey = YES`
	`Add entry of`
	`[%USE_SP_PKEY%?[execute] sp_primarykey %TABLE%, %PKEYCOLUMNS%`
	`:alter table [%QUALIFIER%]%TABLE%`
	`add [constraint %CONSTNAME%] primary key [%IsClustered%] (%PKEYCOLUMNS%)`
	`[%OPTIONS%]]`

Key

La catégorie Key est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les clés sont modélisées pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des clés : • Add • AfterCreate, AfterDrop, AfterModify • BeforeCreate, BeforeDrop, BeforeModify • ConstName • Create, Drop • Enable • MaxConstLen • ModifiableAttributes • Options, DefOptions • ReversedQueries, ReversedStatements • SqlAttrQuery, SqlListQuery, SqlOptsQuery Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.
AKeyComment	Spécifie une instruction permettant d'ajouter un commentaire à une clé alternative.
AllowNullable Coln	Spécifie si des colonnes non obligatoires sont permises. Les valeurs possibles sont les suivantes : • Yes - Colonne pouvant être non obligatoires admises. • No - Seules les colonnes obligatoires sont admises.
EnableCluster	Spécifie si les contraintes clustered sont permises sur les clés alternatives. • Yes - Les contraintes Clustered sont admises. • No - Les contraintes Clustered ne sont pas admises.

Elément	Description

SqlAkeyIndex	Spécifie une requête de reverse engineering permettant d'obtenir les index de clé alternative d'une table via connexion directe.
	Exemple (SQL Anywhere 10) :
	`select distinct  i.index_name`
	`from sys.sysuserperms u`
	`  join sys.systable t on`
	`  (t.creator=u.user_id)`
	`  join sys.sysindex i on`
	`  (i.table_id=t.table_id)`
	`where i."unique" not in ('Y', 'N')`
	`[  and t.table_name = %.q:TABLE%]`
	`[  and u.user_name = %.q:SCHEMA%]`

UniqConstAuto Index	Détermine si une instruction `Create Index` est générée pour chaque instruction de clé. Les valeurs possibles sont les suivantes : • Yes - Génère automatiquement un index de clé alternative au sein de l'instruction de clé alternative. Si vous cochez la case Clé alternative pour la création d'index lorsque vous générez ou modifiez une base de données, la case Clé alternative pour la table est automatiquement décochée, et vice versa. • No - Les index de clé alternative ne sont pas générés automatiquement. Les cases Clé alternative et Création d'index ne peuvent pas être cochées simultanément.

Reference

La catégorie Reference est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les références sont modélisées pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des références : • Add • AfterCreate, AfterDrop, AfterModify • BeforeCreate, BeforeDrop, BeforeModify • ConstName • Create, Drop • Enable • MaxConstLen • ModifiableAttributes • ReversedQueries, ReversedStatements • SqlAttrQuery, SqlListQuery Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.
CheckOn Commit	Spécifie que le test d'intégrité référentielle est uniquement effectué après COMMIT. Contient le mot clé utilisé pour spécifier une référence avec l'option CheckOnCommit. Exemple : `CHECK ON COMMIT`
DclDelIntegrity	Spécifie une liste de contraintes d'intégrité référentielle déclarative pour la suppressions admises. La liste peut contenir n'importe laquelle des valeurs suivantes, ou toutes ces valeurs, qui contrôlent la disponibilité des options correspondantes sur l'onglet Intégrité des feuilles de propriétés de référence : • RESTRICT • CASCADE • SET NULL • SET DEFAULT

Elément	Description
DclUpdIntegrity	Spécifie une liste de contraintes d'intégrité référentielle déclarative pour la modification admises. La liste peut contenir n'importe laquelle des valeurs suivantes, ou toutes ces valeurs, qui contrôlent la disponibilité des options correspondantes sur l'onglet Intégrité des feuilles de propriétés de référence : • RESTRICT • CASCADE • SET NULL • SET DEFAULT
DefineJoin	Spécifie une instruction permettant de définir une jointure pour une référence. Il s'agit d'un autre moyen pour définir le contenu de l'instruction `create reference`, et cela correspond à la variable %JOINS%. En règle générale, le script `create`pour une référence utilise les variables %CKEYCOLUMNS% et %PKEYCOLUMNS% qui contiennent des colonnes enfant et parent séparées par une virgule. Si vous utilisez %JOINS%, vous pouvez faire référence à chaque paire de colonnes parent-enfant séparément. Une boucle est exécutée sur la jointure pour chaque paire de colonnes parent-enfant, ce qui permet d'utiliser une syntaxe mélangeant PK et FK. Exemple (Access 2000) : `P=%PK% F=%FK%`
EnableChange JoinOrder	Spécifie si, lorsqu'une référence est liée à une clé, comme affiché sur l'onglet Jointures d'une feuille de propriétés de référence, l'organisation automatique de l'ordre des jointures est disponible. Les valeurs possibles sont les suivantes : • Yes - L'ordre de jointure peut être établi automatiquement ou manuellement à l'aide de l'option Organisation automatique de l'ordre des jointures. Lorsque vous cochez la case de cette fonctionnalité, vous triez la liste en fonction de l'ordre des colonnes de clé (les boutons de déplacement ne sont pas disponibles). En décochant cette case, vous pouvez modifier manuellement l'ordre de jointure à l'aide des boutons de déplacement (qui sont alors disponibles. • No - La case Organisation automatique de l'ordre des jointures est grisée et non disponible.
EnableCluster	Spécifie si les contraintes clustered sont permises sur les clé étrangères. • Yes - Les contraintes clustered sont permises. • No - Les contraintes clustered ne sont pas permises.

Elément	Description
EnablefKey Name	Spécifie le rôle de clé étrangère admis lors de la génération de base de données. Les valeurs possibles sont les suivantes : • Yes - Le code de la référence est utilisé comme rôle pour la clé étrangère. • No - Le rôle n'est pas admis pour la clé étrangère.
FKAutoIndex	Détermine si une instruction `Create Index` est générée pour chaque instruction de clé étrangère. Les valeurs possibles sont les suivantes : • Yes - Génère automatiquement un index de clé étrangère au sein de l'instruction de clé étrangère. Si vous cochez la case Clé étrangère pour la création d'index lorsque vous générez ou modifiez une base de données, la case Clé étrangère pour la table est automatiquement décochée, et vice-versa. • No – Les index de clé alternative ne sont pas générés automatiquement. Les cases Clé étrangère et Création d'index ne peuvent pas être cochées simultanément.
FKeyComment	Spécifie une instruction permettant d'ajouter un commentaire de clé alternative.

SqlListChildren Query	Spécifie une requête SQL utilisée pour répertorier les jointures dans une référence.
	Exemple (Oracle 10g) :
	`{CKEYCOLUMN, FKEYCOLUMN}`
	`[%ISODBCUSER%?select`
	`p.column_name, f.column_name`
	`from sys.user_cons_columns f,`
	`sys.all_cons_columns p`
	`where f.position = p.position`
	`and f.table_name=%.q:TABLE%`
	`[ and p.owner=%.q:POWNER%]`
	`and p.table_name=%.q:PARENT%`
	`and f.constraint_name=%.q:FKCONSTRAINT%`
	`and p.constraint_name=%.q:PKCONSTRAINT%`
	`order by f.position`
	`:select p.column_name, f.column_name`
	`from sys.all_cons_columns f,`
	`sys.all_cons_columns p`
	`where f.position = p.position`
	`and f.owner=%.q:SCHEMA%`
	`and f.table_name=%.q:TABLE%`
	`[ and p.owner=%.q:POWNER%]`
	`and p.table_name=%.q:PARENT%`
	`and f.constraint_name=%.q:FKCONSTRAINT%`
	`and p.constraint_name=%.q:PKCONSTRAINT%`
	`order by f.position]`

Elément	Description
UseSpFornKey	Spécifie l'utilisation de l'instruction `Sp_foreignkey` pour générer une clé étrangère. Les valeurs possibles sont les suivantes : • Yes - L'instruction `Sp_foreignkey` est utilisée pour créer des références. • No - Les clés étrangères sont générées séparément dans une instruction `alter table` en utilisant l'ordre Create de la référence. Voir aussi `UseSpPrimKey` (*Pkey* à la page 193).

View

La catégorie View est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les vues sont modélisées pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des vues : • AfterCreate, AfterDrop, AfterModify • BeforeCreate, BeforeDrop, BeforeModify • Create, Drop • Enable, EnableSynonym • Header, Footer • ModifiableAttributes • Options • Permission • ReversedQueries, ReversedStatements • SqlAttrQuery, SqlListQuery, SqlOptsQuery, SqlPermQuery Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.
EnableIndex	Spécifie une liste de types de vue pour lesquels un index de vue est disponible. Exemple (Oracle 10g) : `MATERIALIZED`

Elément	Description
SqlListSchema	Spécifie une requête utilisée pour extraire les schémas enregistrés dans la base de données. Cet élément est utilisé avec des vues de type XML (une référence à un document XML stocké dans la base de données). Lorsque vous définissez une vue XML, vous devez pouvoir extraire les documents XML enregistrés dans la base de données afin d'affecter un document à la vue, ce qui se fait en utilisant la requête SqlListSchema. Exemple (Oracle 10g) : `SELECT schema_url FROM dba_xml_schemas`
SqlXMLView	Spécifie une sous-requête utilisée pour améliorer la performance de SqlAttrQuery.
TypeList	Spécifie une liste de types (par exemple, SGBD : relationnel, objet, XML) pour les vues. Cette liste remplit la liste Type de la feuille de propriétés de vue. Le type XML doit être utilisé avec l'élément SqlListSchema.

ViewCheck	Spécifie si la case With Check Option est disponible dans la feuille de propriétés de la vue. Si la case est cochée et que le paramètre `ViewCheck` n'est pas vide, la valeur de `ViewCheck` est générée à la fin de l'instruction select de la vue et avant le caractère de fin.
	Exemple (SQL Anywhere 10) :
	Si ViewCheck est défini à la valeur with check option, le script généré se présente comme suit :
	`create view TEST as`
	`select CUSTOMER.CUSNUM, CUSTOMER.CUSNAME, CUSTOMER.CUSTEL`
	`from CUSTOMER`
	`with check option;`

ViewComment	Spécifie une instruction permettant d'ajouter un commentaire de vue. Si ce paramètre est vide, la case Commentaire située sous Vue dans les options de la boîte de dialogue de génération de base de données n'est pas disponible.
	Exemple (Oracle 10g) :
	`[%VIEWSTYLE%=view? comment on table [%QUALIFIER%]%VIEW% is`
	`%.q:COMMENT%]`

ViewStyle	Spécifie le type d'utilisation de la vue. La valeur définie est affichée dans la liste Utilisation sur la feuille de propriétés de la vue. Exemple (Oracle 10g) : `materialized view`

Tablespace

La catégorie Tablespace est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les tablespaces sont modélisés pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des tablespaces : • AfterCreate, AfterDrop, AfterModify • BeforeCreate, BeforeDrop, BeforeModify • Create, Drop • Enable • ModifiableAttributes • Options, DefOptions • ReversedQueries, ReversedStatements • SqlAttrQuery, SqlListQuery, SqlOptsQuery Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.
Tablespace Comment	Spécifie une instruction permettant d'ajouter commentaire au tablespace.

Storage

La catégorie Storage est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les storages sont modélisés pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des storages: • AfterCreate, AfterDrop, AfterModify • BeforeCreate, BeforeDrop, BeforeModify • Create, Drop • Enable • ModifiableAttributes • Options, DefOptions • ReversedQueries, ReversedStatements • SqlAttrQuery, SqlListQuery Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.

Elément	Description
Storage Comment	Spécifie une instruction permettant d'ajouter commentaire à un storage.

Database

La catégorie Database est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les bases de données sont modélisées pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des bases de données : • AfterCreate, AfterDrop, AfterModify • BeforeCreate, BeforeDrop, BeforeModify • Create, Drop • Enable • ModifiableAttributes • Options, DefOptions • Permission • ReversedQueries, ReversedStatements • SqlAttrQuery, SqlListQuery, SqlOptsQuery, SqlPermQuery Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.
BeforeCreate Database	Contrôle l'ordre dans lequel les bases de données, les tablespaces et les storages sont générés. Les valeurs possibles sont les suivantes : • Yes – [valeur par défaut] Les instructions Create Tablespace et Create Storage sont générées avant l'instruction Create Database. • No - Les instructions Create Tablespace et Create Storage sont générées après l'instruction Create Database
CloseDatabase	Spécifie la commande permettant de fermer la base de données. Si ce paramètre est vide, l'option Base de données/Fermeture sur l'onglet Options de la boîte de dialogue Génération d'une base de données n'est pas disponible.
EnableMany Databases	Permet la prise en charge de plusieurs bases de données dans le même modèle.

Elément	Description
OpenDatabase	Spécifie la commande permettant d'ouvrir la base de données. Si ce paramètre est vide, l'option Base de données/Ouverture sur l'onglet Options de la boîte de dialogue Génération d'une base de données n'est pas disponible. Exemple (ASE 15) : `use %DATABASE%` La variable %DATABASE% représente le code de la base de données associée au modèle généré.

Domain

La catégorie Domain est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les domaines sont modélisés pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des domaines : • AfterCreate, AfterDrop, AfterModify • BeforeCreate, BeforeDrop, BeforeModify • Create, Drop • Enable, EnableOwner • Maxlen • ModifiableAttributes • ReversedQueries, ReversedStatements • SqlAttrQuery, SqlListQuery Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.
Bind	Spécifie la syntaxe pour lier une règle de gestion à un domaine. Exemple (ASE 15) : `[%R%?[exec ]][execute ]sp_bindrule [%R%?['[%QUALIFIER%]%RULE%']][[%QUALIFIER%]%RULE%]:['[%QUALIFIER%]%RULE%']], %DOMAIN%`
EnableBindRule	Spécifie si les règles de gestion peuvent être liées aux domaines pour les paramètres de contrôle. Les valeurs possibles sont les suivantes : • Yes - Les entrées Create et Bind de l'objet Rule sont générées • No - Le contrôle inclus dans la commande d'ajout du domaine est généré

Elément	Description
EnableCheck	Spécifie si les paramètres de contrôle sont générés. Cet élément est testé lors de la génération de colonne. Si l'option Type utilisateur est sélectionnée pour les colonnes dans la boîte de dialogue de génération, et si EnableCheck est défini à Yes pour les domaines, alors les paramètres de contrôle ne sont pas générés pour les colonnes, puisque la colonne est associée à un domaine ayant des paramètres de contrôle. Lorsque les contrôles portant sur la colonne divergent de ceux sur le domaine, les contrôles sur la colonne sont générés. Les valeurs possibles sont les suivantes : • Yes - Les paramètres de contrôle sont générés • No - Toutes les variables liées aux paramètres de contrôle ne seront pas évaluées lors de la génération et du reverse engineering
EnableDefault	Spécifie si les valeurs par défaut sont générées. Les valeurs possibles sont les suivantes : • Yes - Les valeurs par défaut définies pour les domaines sont générées. La valeur par défaut peut être définie dans les paramètres de contrôle. La variable %DEFAULT% contient la valeur par défaut • No - Les valeurs par défaut ne sont pas générées
SqlListDefault Query	Spécifie une requête SQL permettant de récupérer et de répertorier les valeurs par défaut du domaine dans les tables système lors du reverse engineering.
UddtComment	Spécifie une instruction permettant d'ajouter commentaire de type de données utilisateur.
Unbind	Spécifie la syntaxe pour dissocier une règle de gestion d'un domaine. Exemple (ASE 15) : `[%R%?[exec ]][execute ]sp_unbindrule %DOMAIN%`

Abstract Data Type

La catégorie Abstract Data Type est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les types de données abstraits sont modélisés pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des types de données abstraits : • AfterCreate, AfterDrop, AfterModify • BeforeCreate, BeforeDrop, BeforeModify • Create, Drop • Enable • ModifiableAttributes • Permission • ReversedQueries, ReversedStatements • SqlAttrQuery, SqlListQuery, SqlPermQuery Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.
ADTComment	Spécifie une instruction permettant d'ajouter un commentaire de type de données abstrait.

AllowedADT	Spécifie une liste des types de données abstraits qui peuvent être utilisés comme types de données pour un type de données abstrait.
	Exemple (Oracle 10g) :
	`OBJECT`
	`TABLE`
	`VARRAY`

Authorizations	Spécifie une liste des utilisateurs capables d'appeler les types de données abstraits.

CreateBody	Spécifie une instruction permettant de créer un corps de type de données abstrait.
	Exemple (Oracle 10g) :
	`create [or replace ]type body [%QUALIFIER%]%ADT%`
	`[.O:[as][is]]`
	`%ADTBODY%`
	`end;`

Elément	Description
EnableAdtOn Coln	Spécifie si types de données abstraits sont activés pour les colonnes. Les valeurs possibles sont les suivantes : • Yes - Les types de données abstraits sont ajoutés à la liste des types de colonne à condition d'avoir le type valide. • No - Les types de données abstraits ne sont pas admis pour les colonnes.
EnableAdtOn Domn	Spécifie si types de données abstraits sont activés pour les domaines. Les valeurs possibles sont les suivantes : • Yes - Les types de données abstraits sont ajoutés dans la listes des types de domaines, à condition qu'ils aient un type correct • No - Les types de données abstraits ne sont pas admis pour les domaines
Enable Inheritance	Active l'héritage pour les types de données abstraits.
Install	Spécifie une instruction permettant d'installer une classe Java comme classe de données abstraite (dans ASA, types de données abstraits sont installés et retirés plutôt que créés et supprimés). Cet élément équivaut à une instruction `create`. Exemple (SQL Anywhere 10) : `install JAVA UPDATE from file %.q:FILE%`
JavaData	Spécifie une liste de mécanismes d'instanciation pour les types de données abstraits SQL Java.
Remove	Spécifie une instruction permettant d'installer une classe Java comme classe de données abstraite. Exemple (SQL Anywhere 10) : `remove JAVA class %ADT%`

Abstract Data Type Attribute

La catégorie Abstract Data Types Attribute est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les attributs de type de données abstrait sont modélisés pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour les attributs de type de données abstrait : • Add • AfterCreate, AfterDrop, AfterModify • BeforeCreate, BeforeDrop, BeforeModify • Create, Drop, Modify • ModifiableAttributes • ReversedQueries, ReversedStatements • SqlListQuery Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.

AllowedADT	Spécifie une liste de types de données abstraits qui peuvent être utilisés comme types de données pour attributs de type de données abstrait.
	Exemple (Oracle 10g) :
	`OBJECT`
	`TABLE`
	`VARRAY`
	Si vous sélectionnez le type OBJECT pour un type de données abstrait, un onglet Attributs s'affiche dans la feuille de propriétés de type de données abstrait, vous permettant de spécifier les attributs du type de données d'objet.

User

La catégorie User est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les utilisateurs sont modélisés pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des utilisateurs : • AfterCreate, AfterDrop, AfterModify • BeforeCreate, BeforeDrop, BeforeModify • Create, Drop • Enable • Maxlen • ModifiableAttributes • Options, DefOptions • ReversedQueries, ReversedStatements • SqlAttrQuery, SqlListQuery, SqlOptsQuery, SqlPermQuery Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.
UserComment	Spécifie une instruction permettant d'ajouter commentaire à une utilisateur.

Rule

La catégorie Rule est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les règles sont modélisées pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des règles : • AfterCreate, AfterDrop, AfterModify • BeforeCreate, BeforeDrop, BeforeModify • Create, Drop • Enable • Maxlen • ModifiableAttributes • ReversedQueries, ReversedStatements • SqlAttrQuery, SqlListQuery Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.

Elément	Description

ColnDefault Name	Spécifie le nom d'un défaut pour une colonne. Cet élément est utilisé avec les SGBD qui ne prennent pas en charge les paramètres de contrôle sur les colonnes. Lorsqu'une colonne a une valeur par défaut spécifique définie dans ses paramètres de contrôle, un nom est créé pour cette valeur par défaut.
	La variable correspondante est %DEFAULTNAME%.
	Exemple (ASE 15) :
	`D_%.19:COLUMN%_%.8:TABLE%`
	La colonne Employee function `EMPFUNC` de la table EMPLOYEE a la valeur par défaut, `Technical Engineer.` Le nom par défaut de la colonne, `D_EMPFUNC_EMPLOYEE`, est créé :
	`create default D_EMPFUNC_EMPLOYEE`
	`as 'Technical Engineer'`
	`go`
	`execute sp_bindefault D_EMPFUNC_EMPLOYEE, "EMPLOYEE.EMPFUNC"`
	`go`

ColnRuleName	Spécifie le nom d'une règle pour une colonne. Cet élément est utilisé avec des SGBD qui ne prennent pas en charge les paramètres de contrôle sur les colonnes. Lorsqu'une colonne a une règle spécifique définie sur ses paramètres de contrôle, un nom est créé pour cette règle.
	La variable correspondante est %RULE%.
	Exemple (ASE 15) :
	`R_%.19:COLUMN%_%.8:TABLE%`
	La colonne Speciality (TEASPE) de la table Team a une liste de valeurs définie dans ses paramètres de contrôle : Industry, Military, Nuclear, Bank, Marketing :
	Le nom de règle suivant, R_TEASPE_TEAM, est créé et associé à la colonne TEASPE :
	`create rule R_TEASPE_TEAM`
	`as @TEASPE in ('Industry','Military','Nuclear','Bank','Marketing')`
	`go`
	`execute sp_bindrule R_TEASPE_TEAM, "TEAM.TEASPE"`
	`go`

MaxDefaultLen	Spécifie la longueur maximum prise en charge par le SGBD pour le nom par défaut de la colonne.
RuleComment	Spécifie une instruction permettant d'ajouter un commentaire à la règle.

Elément	Description

UddtDefault Name	Spécifie le nom par défaut pour un type de données utilisateur. Cet élément est utilisé avec les SGBD qui ne prennent pas en charge les paramètres de contrôle sur les types de données utilisateur. Lorsqu'un type de données utilisateur a une valeur par défaut spécifique définie dans ses paramètres de contrôle, un nom est créé pour cette valeur par défaut.
	La variable correspondante est %DEFAULTNAME%.
	Exemple (ASE 15) :
	`D_%.28:DOMAIN%`
	Le domaine `FunctionList` a une valeur par défaut définie dans ses paramètres de contrôle : `Technical Engineer`. Le script SQL suivant va générer un nom par défaut pour cette valeur par défaut :
	`create default D_FunctionList`
	`as 'Technical Engineer'`
	`go`

UddtRuleName	Spécifie le nom d'une règle pour un type de données utilisateur. Cet élément est utilisé avec les SGBD qui ne prennent pas en charge les paramètres de contrôle sur les types de données utilisateur. Lorsqu'un type de données utilisateur a une règle spécifique définie dans ses paramètres de contrôle, un nom est créé pour cette règle.
	La variable correspondante est %RULE%.
	Exemple (ASE 15) :
	`R_%.28:DOMAIN%`
	Le domaine `FunctionList` a une valeur par défaut définie dans ses paramètres de contrôle : `Technical Engineer`. Le SQL suivant va générer un nom par défaut pour cette valeur par défaut :
	`create rule R_Domain_speciality`
	`as (@Domain_speciality in ('Industry','Milita-`
	`ry','Nuclear','Bank','Marketing'))`
	`go`
	`execute sp_bindrule R_Domain_speciality, T_Do-`
	`main_speciality`
	`go`

Procedure

La catégorie Procedure est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les procédures sont modélisées pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des procédures : • AfterCreate, AfterDrop, AfterModify • BeforeCreate, BeforeDrop, BeforeModify • Create, Drop • Enable, EnableOwner, EnableSynonym • Maxlen • ModifiableAttributes • Permission • ReversedQueries, ReversedStatements • SqlAttrQuery, SqlListQuery, SqlPermQuery Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.
CreateFunc	Spécifie l'instruction permettant la création d'une fonction. Exemple (SQL Anywhere 10) : `create function [%QUALIFIER%]%FUNC%[%PROCPRMS%? ([%PROCPRMS%])] %TRGDEFN%`
CustomFunc	Spécifie l'instruction permettant la création d'une fonction utilisateur, une forme de procédure qui renvoie une valeur à l'environnement appelant à utilisateur dans des requêtes et dans d'autres instructions SQL. Exemple (SQL Anywhere 10) : `create function [%QUALIFIER%]%FUNC% (<arg> <type>) RETURNS <type> begin end`
CustomProc	Spécifie l'instruction permettant la création d'une procédure stockée. Exemple (SQL Anywhere 10) : `create procedure [%QUALIFIER%]%PROC% (IN <arg> <type>) begin end`

Elément	Description
DropFunc	Spécifie l'instruction permettant de supprimer une fonction. Exemple (SQL Anywhere 10) : `if exists(select 1 from sys.sysprocedure where proc_name = %.q:FUNC%[ and user_name(creator) = %.q:OWNER%]) then drop function [%QUALIFIER%]%FUNC% end if`
EnableFunc	Spécifie si les fonctions sont admises. Les fonctions sont des formes de procédure qui renvoient une valeur à l'environnement appelant à utiliser dans des requêtes et d'autres instructions SQL.
Function Comment	Spécifie une instruction permettant d'ajouter un commentaire à une fonction.
ImplementationType	Spécifie une liste de types de modèle de procédure disponibles.
MaxFuncLen	Spécifie la longueur maximum du nom d'une fonction.
Procedure Comment	Spécifie une instruction permettant d'ajouter un commentaire à une procédure.

Trigger

La catégorie Trigger est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les triggers sont modélisés pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des triggers : • AfterCreate, AfterDrop, AfterModify • BeforeCreate, BeforeDrop, BeforeModify • Create, Drop • Enable, EnableOwner • Maxlen • ModifiableAttributes • ReversedQueries, ReversedStatements • SqlAttrQuery, SqlListQuery Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.
DefaultTrigger Name	Spécifie un modèle pour définir les noms de trigger par défaut. Exemple (SQL Anywhere 10) : `%TEMPLATE%_%.L:TABLE%`

Elément	Description
EnableMulti Trigger	Permet l'utilisation de plusieurs triggers par type.

Event	Spécifie une liste d'attributs d'événement de trigger pour remplir la liste Evénement sur l'onglet Définition des feuilles de propriétés de trigger.
	Exemple :
	`Delete`
	`Insert`
	`Update`

EventDelimiter	Spécifie un caractère pour séparer plusieurs événements de trigger.
ImplementationType	Spécifie une liste de types de modèle de trigger disponibles.

Time	Spécifie une liste d'attributs de moment de trigger permettant de remplir la liste Moment sur l'onglet Définition des feuilles de propriétés de trigger.
	Exemple :
	`Before`
	`After`

Trigger Comment	Spécifie une instruction permettant d'ajouter un commentaire à un trigger.
UniqName	Spécifie si les noms de trigger doivent être uniques dans la portée globale de la base de données. Les valeurs possibles sont les suivantes : • Yes – Les noms de trigger doivent être uniques dans la portée globale de la base de données. • No – Les noms de trigger doivent être uniques pour chaque objet

Elément	Description

UseErrorMsg Table	Spécifie une macro pour accéder aux messages d'erreur de trigger depuis une table de messages dans votre base de données.
	
	Permet d'utiliser le bouton Défini par l'utilisateur sur l'onglet Messages d'erreur de la boîte de dialogue de régénération des triggers (voir *Modélisation des données > Construction de modèles de données > Triggers et procédures > Génération de triggers et de procédures > Création et génération de messages d'erreur personnalisés*).
	
	Si un numéro d'erreur dans le script de trigger correspond à un numéro d'erreur dans la table de messages, le message d'erreur par défaut de la macro .ERROR est remplacé par votre message.
	
	Exemple (ASE 15) :
	`begin`
	`select @errno = %ERRNO%,`
	`@errmsg = %MSGTXT%`
	`from %MSGTAB%`
	`where %MSGNO% = %ERRNO%`
	`goto error`
	`end`
	
	Où :
	• %ERRNO% - paramètre de numéro d'erreur pour la macro .ERROR macro
	• %ERRMSG% - paramètre de texte de message d'erreur pour la macro .ERROR
	• %MSGTAB% - nom de la table de messages
	• %MSGNO% - nom de la colonne qui stocke le numéro de message d'erreur
	• %MSGTXT% - nom de la colonne du texte de message d'erreur
	
	Voir aussi UseErrorMsgText.

Elément	Description

UseErrorMsg Text	Spécifie une macro permettant d'accéder aux messages d'erreur du trigger depuis la définition du modèle de trigger.
	Permet d'utiliser l'option Standard sur l'onglet Messages d'erreur de la boîte de dialogue Régénération de trigger.
	Le numéro d'erreur et le message définis dans la définition de modèle sont utilisés.
	Exemple (ASE 15) :
	`begin`
	`select @errno  = %ERRNO%,`
	`@errmsg = %MSGTXT%`
	`goto error`
	`end`
	Voir aussi UseErrorMsgTable.

ViewTime	Spécifie une liste de moments disponibles pour le trigger sur la vue.

DBMS Trigger

La catégorie DBMS Trigger est située dans la catégorie**Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les triggers de SGBD sont modélisés pour votre SGBD.

Item	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des triggers de SGBD : • AfterCreate, AfterDrop, AfterModify • BeforeCreate, BeforeDrop, BeforeModify • Create, Drop • Alter, AlterStatementList, AlterDBIgnored • Enable, EnableOwner • Header, Footer • Maxlen • ModifiableAttributes • ReversedQueries, ReversedStatements • SqlAttrQuery, SqlListQuery Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.
EventDelimiter	Spécifie un caractère qui sépare les différents événements de trigger.

Item	Description
Events_*scope*	Spécifie une liste d'attributs d'événement de trigger qui remplissent la liste Evénement sur l'onglet Définition de la feuille de propriétés d'un trigger en fonction de la *portée* sélectionnée, par exemple, Schema, Database, Server.
Scope	Spécifie une liste de portées disponibles pour le trigger de SGBD. Chaque portée doit avoir un élément Events_*portée* associé.

Time	Spécifie une liste d'attributs de moment de déclenchement pour renseigner la liste Moment sur l'onglet Définition de la feuille de propriétés d'un trigger.
	Exemple :
	`Before`
	`After`

Trigger Comment	Spécifie une liste instruction permettant d'ajouter un commentaire de trigger.

Join Index

La catégorie Join Index est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les join indexes sont modélisés pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des join indexes : • Add • AfterCreate, AfterDrop, AfterModify • BeforeCreate, BeforeDrop, BeforeModify • Create, Drop • Enable, EnableOwner • Header, Footer • Maxlen • ModifiableAttributes • Options, DefOptions • ReversedQueries, ReversedStatements • SqlAttrQuery, SqlListQuery, SqlOptsQuery Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.
AddJoin	Spécifie l'instruction SQL utilisée pour définir des jointures pour les join indexes. Exemple : `Table1.coln1 = Table2.coln2`

Elément	Description
EnableJidxColn	Permet la prise en charge de l'attachement de plusieurs colonnes à un join index. Dans Oracle 9i, on appelle cet attachement un join index bitmap.
JoinIndex Comment	Spécifie une instruction permettant d'ajouter un commentaire à un join index.

Qualifier

La catégorie Qualifier est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les qualifiants sont modélisés pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des qualifiants : • Enable • ReversedQueries • SqlListQuery Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.
Label	Spécifie un libellé pour <Tout> dans la liste de sélection de qualifiant.

Sequence

La catégorie Sequence est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les séquences sont modélisées pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des séquence : • AfterCreate, AfterDrop, AfterModify • BeforeCreate, BeforeDrop, BeforeModify • Create, Drop • Enable, EnableOwner, EnableSynonym • Maxlen • ModifiableAttributes • Options, DefOptions • Permission • ReversedQueries, ReversedStatements • SqlAttrQuery, SqlListQuery, SqlPermQuery Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.
Rename	Spécifie la commande permettant de renommer une séquence. Exemple (Oracle 10g) : `rename %OLDNAME% to %NEWNAME%`
Sequence Comment	Spécifie une instruction permettant d'ajouter un commentaire à une séquence.

Synonym

La catégorie Synonym est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les synonymes sont modélisés pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des synonymes : • Create, Drop • Enable, EnableSynonym • Maxlen • ReversedQueries • SqlAttrQuery, SqlListQuery Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.
EnableAlias	Spécifie si les synonymes peuvent avoir un type d'alias.

Group

La catégorie Group est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les groupes sont modélisés pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des groupes : • AfterCreate, AfterDrop, AfterModify • BeforeCreate, BeforeDrop, BeforeModify • Create, Drop • Enable • Maxlen • ModifiableAttributes • ReversedQueries, ReversedStatements • SqlAttrQuery, SqlListQuery, SqlPermQuery Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.
Bind	Spécifie une commande permettant d'ajouter un utilisateur à un groupe. Exemple (SQL Anywhere 10) : `grant membership in group %GROUP% to %USER%`

Elément	Description
Group Comment	Spécifie une instruction permettant d'ajouter commentaire à un groupe.
ObjectOwner	Permet aux groupes d'être propriétaire d'objets.

SqlListChildren Query	Spécifie une requête SQL permettant de répertorier les membres d'un groupe.
	Exemple (ASE 15) :
	`{GROUP ID, MEMBER}`
	`select g.name, u.name`
	`from`
	`[%CATALOG%.]dbo.sysusers u, [%CATALOG%.]dbo.sysusers g`
	`where`
	`u.suid > 0 and`
	`u.gid = g.gid and`
	`g.gid = g.uid`
	`order by 1, 2`

Unbind	Spécifie une commande permettant de supprimer un utilisateur d'un groupe. Exemple (SQL Anywhere 10) : `revoke membership in group %GROUP% from %USER%`

Role

La catégorie Role est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les rôles sont modélisés pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des rôles : • AfterCreate, AfterDrop, AfterModify • BeforeCreate, BeforeDrop, BeforeModify • Create, Drop • Enable • Maxlen • ModifiableAttributes • ReversedQueries, ReversedStatements • SqlAttrQuery, SqlListQuery, SqlPermQuery Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.

Elément	Description
Bind	Spécifie une commande permettant d'ajouter un rôle à un utilisateur ou à une autre rôle. Exemple (ASE 15) : `grant role %ROLE% to %USER%`

SqlListChildren Query	Spécifie une requête SQL permettant de répertorier les membres d'un groupe.
	Exemple (ASE 15) :
	`{ ROLE ID, MEMBER }`
	`SELECT r.name, u.name`
	`FROM`
	`master.dbo.sysloginroles l,`
	`[%CATALOG%.]dbo.sysroles s,`
	`[%CATALOG%.]dbo.sysusers u,`
	`[%CATALOG%.]dbo.sysusers r`
	`where`
	`l.suid = u.suid`
	`and s.id =l.srid`
	`and r.uid = s.lrid`

Unbind	Spécifie une commande permettant de supprimer un rôle d'un utilisateur ou d'un autre rôle.

DB Package

La catégorie DB Package est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les packages de base de données sont modélisés pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des packages de base de données : • AfterCreate, AfterDrop, AfterModify • BeforeCreate, BeforeDrop, BeforeModify • Create, Drop • Enable, EnableSynonym • Maxlen • ModifiableAttributes • Permission • ReversedQueries, ReversedStatements • SqlAttrQuery, SqlListQuery, SqlPermQuery Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.

Elément	Description
Authorizations	Spécifie une liste des utilisateurs pouvant appeler des packages de base de données.

CreateBody	Spécifie un modèle permettant de définir le corps d'un package de base de données. Cette instruction est utilisée dans l'instruction d'extension After-Create.
	Exemple (Oracle 10g) :
	`create [or replace ]package body [%QUALIFIER%] %DBPACKAGE% [.O:[as][is]][%IsPragma% ? pragma serially_reusable]`
	` %DBPACKAGEBODY%`
	`[begin`
	` %DBPACKAGEINIT%`
	`]end[ %DBPACKAGE%];`

Sous-objets de DB Package

Les catégories suivantes sont situées sous la catégorie **Racine** > **Script** > **Objects** :

- DB Package Procedure
- DB Package Variable
- DB Package Type
- DB Package Cursor
- DB Package Exception
- DB Package Pragma

Chacune contient la plupart des éléments suivants qui définissent la façon dont les sous-objets de packages de base de données sont modélisés pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour les sous-objets de packages de base de données : • Add • ReversedQueries Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.

DBProcedure Body	[procédure de package de base de données uniquement] Spécifie un modèle permettant de définir le corps de la procédure de package dans l'onglet Définition de sa feuille de propriétés.
	Exemple (Oracle 10g) :
	`begin`
	`end`

Elément	Description

ParameterTypes	[procédure et curseur de package de base de données uniquement] Spécifie les types disponibles pour les procédures ou curseurs.
	Exemple (Oracle 10g : procédure) :
	`in`
	`in nocopy`
	`in out`
	`in out nocopy`
	`out`
	`out nocopy`

Parameter

La catégorie Parameter est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les paramètres sont modélisés pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des paramètres : • Add • ReversedQueries Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.

Privilege

La catégorie Privilege est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les privilèges sont modélisés pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des privilèges : • AfterCreate, AfterDrop, AfterModify • BeforeCreate, BeforeDrop, BeforeModify • Create, Drop • Enable • ModifiableAttributes • ReversedQueries, ReversedStatements Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.

Elément	Description
GrantOption	Spécifie l'option d'octroi pour une instruction portant sur les privilèges. Exemple (Oracle 10g) : `with admin option`
RevokeInherited	Permet de révoquer des privilèges hérités des groupes et des rôles.
RevokeOption	Spécifie l'option de révocation pour une instruction portant sur les privilèges.

System	Spécifie une liste de privilèges système disponibles.
	Exemple (ASE 15) :
	`CREATE DATABASE`
	`CREATE DEFAULT`
	`CREATE PROCEDURE`
	`CREATE TRIGGER`
	`CREATE RULE`
	`CREATE TABLE`
	`CREATE VIEW`

Permission

La catégorie Permission est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les permissions sont modélisées pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des permissions : • Create, Drop • Enable • ReversedQueries • SqlListQuery Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.
GrantOption	Spécifie l'option d'octroi pour une instruction portant sur les permissions. Exemple (ASE 15) : `with grant option`
RevokeInherited	Permet de révoquer les permissions héritées pour les groupes et rôles.

Elément	Description
RevokeOption	Spécifie l'option de révocation pour une instruction portant sur les permissions. Exemple (ASE 15) : `cascade`

Default

La catégorie Default est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les défauts sont modélisés pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des défauts : • AfterCreate, AfterDrop, AfterModify • BeforeCreate, BeforeDrop, BeforeModify • Create, Drop • Enable, EnableOwner • Maxlen • ModifiableAttributes • ReversedQueries, ReversedStatements • SqlAttrQuery, SqlListQuery Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.

Bind	Spécifie la commande permettant de lier un objet par défaut à un domaine ou à une colonne.
	Lorsqu'un domaine ou une colonne utilise un défaut, une instruction *binddefault* est générée après l'instruction de création du domaine ou de la table. Dans l'exemple suivant, la colonne Address dans la table Customer utilise le défaut CITYDFLT :
	`create table CUSTOMER (`
	`  ADDRESS  char(10)  null`
	`)`
	`sp_bindefault CITYDFLT, 'CUSTOMER.ADDRESS'`
	Si le domaine ou la colonne utilise une valeur de défaut directement saisie dans la liste Défaut, la valeur de défaut est déclarée sur la ligne de création de la colonne :
	`ADDRESS  char(10)  default 'StdAddr' null`

PublicOwner	Permet à PUBLIC de posséder des synonymes publics.

Elément	Description
Unbind	Spécifie la commande permettant de dissocier un défaut d'un domaine ou d'une colonne. Exemple (ASE 15) : `[%R%?[exec ]][execute ]sp_unbindefault %.q:BOUND_OBJECT%`

Web Service et Web Operation

Les catégories Web Service et Web Operation sont situées sous **Racine > Script > Objects**, et peuvent contenir les éléments suivants qui définissent la façon dont les services Web et les opérations Web sont modélisés pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des services Web et les opérations Web : • AfterCreate, AfterDrop, AfterModify • Alter • BeforeCreate, BeforeDrop, BeforeModify • Create, Drop • Enable, EnableOwner • Header, Footer • MaxConstLen (opérations Web uniquement) • Maxlen • ModifiableAttributes • ReversedQueries, ReversedStatements • SqlAttrQuery, SqlListQuery Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.
Enable Namespace	Spécifie si les espaces de noms sont pris en charge.
EnableSecurity	Spécifie si les options de sécurité sont prises en charge.

OperationType List	[opération Web uniquement] Spécifie une liste de types d'opération de service Web.
	Exemple (DB2 UDB 8.x CS) :
	`query`
	`update`
	`storeXML`
	`retrieveXML`
	`call`

Elément	Description

ServiceTypeList	[service Web uniquement] Spécifie une liste de types de services Web.
	Exemple (SQL Anywhere 10) :
	`RAW`
	`HTML`
	`XML`
	`DISH`

UniqName	Spécifie si les noms d'opération de service Web peuvent être uniques dans la base de données.
WebService Comment/ WebOperation Comment	Spécifie la syntaxe permettant d'ajouter un commentaire à un service Web pi à une opération de service Web.

Web Parameter

La catégorie Web Parameter est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les paramètres Web sont modélisés pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des paramètres Web : • Add • Enable Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.
EnableDefault	Permet d'utiliser des valeurs par défaut pour les paramètres de service Web.
ParameterDttp List	Spécifie une liste de types de données qui peuvent être utilisées comme paramètres de service Web.

Result Column

La catégories Result Column est située sous **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les colonnes de résultat sont modélisés pour votre SGBD.

Elément	Description
ResultColumn DttpList	Spécifie une liste de types de données qui peuvent être utilisés pour les colonnes de résultat.

Dimension

La catégorie Dimension est située dans la catégorie **Racine** > **Script** > **Objects**, et peut contenir les éléments suivants qui définissent la façon dont les dimensions sont modélisées pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des dimensions : • AfterCreate, AfterDrop, AfterModify • Alter • BeforeCreate, BeforeDrop, BeforeModify • Create, Drop • Enable • Header, Footer • Maxlen • ReversedQueries • SqlAttrQuery, SqlListQuery Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.
AddAttr Hierarchy	Spécifie la syntaxe permettant de définir une liste d'attributs de hiérarchie. Exemple (Oracle 10g) : `child of %DIMNATTRHIER%`
AddAttribute	Spécifie la syntaxe permettant de définir un attribut. Exemple (Oracle 10g) : `attribute %DIMNATTR% determines [.O:[(%DIMNDEPCOLN-LIST%)][%DIMNDEPCOLN%]]`
AddHierarchy	Spécifie la syntaxe permettant de définir une hiérarchie de dimensions. Exemple (Oracle 10g) : `hierarchy %DIMNHIER% ( %DIMNATTRHIERFIRST% %DIMNATTRHIERLIST%)`
AddJoin Hierarchy	Spécifie la syntaxe permettant de définir une liste de jointures pour les attributs de hiérarchie. Exemple (Oracle 10g) : join key [.O:[(%DIMNKEYLIST%)][%DIMNKEYLIST%]] references %DIMNPARENTLEVEL%

Elément	Description
AddLevel	Spécifie la syntaxe pour le niveau de dimension (attribut). Exemple (Oracle 10g) : level %DIMNATTR% is [.O:[(%DIMNCOLNLIST%)][%DIMNTABL%.%DIMNCOLN%]]

Extended Object

La catégorie Extended Object est située dans la catégorie **Racine > Script > Objects**, et peut contenir les éléments suivants qui définissent la façon dont les objets étendus sont modélisés pour votre SGBD.

Elément	Description
[Eléments communs]	Les éléments suivants communs aux différents objets peuvent être définis pour des objets étendus : • AfterCreate, AfterDrop, AfterModify • BeforeCreate, BeforeDrop, BeforeModify • Create, Drop • EnableSynonym • Header, Footer • ModifiableAttributes • ReversedQueries, ReversedStatements • SqlAttrQuery, SqlListQuery Pour obtenir une description de chacun de ces éléments communs, voir *Eléments communs aux différents objets* à la page 171.
AlterStatement List	Spécifie une liste d'éléments de texte représentant des instructions modifiant les attributs correspondants
Comment	Spécifie la syntaxe permettant d'ajouter un commentaire à un objet étendu.

Catégorie Script/Data Type

La catégorie Data Type fournit des correspondances afin de permettre à PowerAMC de gérer correctement les types de données spécifiques aux SGBD.

Les variables suivantes sont utilisées dans de nombreuses entrées :

- %n - Longueur du type de données
- %s - Taille du type de données
- %p - Précision du type de données

Elément	Description
AmcdAmcdType	Répertorie les correspondances utilisées afin de convertir des types de données spécialisés (tels que `XML`, `IVL`, `MEDIA`, etc) en types de données PowerAMC standard. Ces correspondances sont utilisées afin d'aider la conversion d'un SGBD à l'autre, lorsque le nouveau SGBD ne prend pas en charge un ou plusieurs de ces types spécialisés. Par exemple, si le type de données `XML` n'est pas pris en charge, c'est `TXT` qui est utilisés.
AmcdDataType	Répertorie les correspondances utilisées afin de convertir des types de données PowerAMC (**internes**) en types de données de types de SGBD (**modèle physique**). Ces correspondances sont utilisées pendant la génération d'un MCD vers un MPD ainsi que lorsque vous utilisez la commande **Changer de SGBD courant**. Exemples (ASE 15) : • Le type de données `A%n` de PowerAMC est converti en type de données `char(%n)` pour ASE 15. • Le type de données `VA%n` de PowerAMC est converti en type de données `varchar(%n)` pour ASE 15.
PhysDataType	Répertorie les correspondances entre les types de données de SGBD (**modèle physique**) et les types de données de PowerAMC (**internes**). Ces correspondances sont utilisées pendant la génération d'un MPD vers un MCD ainsi que lorsque vous utilisez la commande **Changer de SGBD courant**. Exemples (ASE 15) : • Le type de données `sysname` d'ASE 15 est converti en type de données `VA30` pour PowerAMC. • Le type de données `integer` d'ASE 15 est converti en type de données `I` pour PowerAMC.
PhysDttpSize	Répertorie les tailles de stockage pour les types de données de SGBD. Ces valeurs sont utilisées lors de l'estimation de la taille d'une base de données. Exemples (ASE 15) : • Le type de données `smallmoney` d'ASE 15 requiert 8 octets de mémoire. • Le type de données `smalldatetime` d'ASE 15 requiert 4 octets de mémoire.

Elément	Description
OdbcPhysData Type	Répertorie les correspondances permettant de convertir les types de données de base de données directe (**ODBC**) en type de données de SGBD (**modèle physique**) lors du reverse engineering de base de données. Ces correspondances sont utilisées lorsque la façon de stocker les types de données dans la base de données diffère de la notation de SGBD (le plus souvent à cause de l'inclusion d'une taille par défaut). Exemples (ASE 15) : • Un type de données `float(8)` dans une base de données ASE 15 est récupéré comme `float`. • Un type de données `decimal(30,6)` dans une base de données ASE 15 est récupéré comme `decimal`.
PhysOdbcData Type	Répertorie les correspondances de types de données de SGBD (**modèle physique**) en types de données de base de données (**ODBC**) à utiliser lors de la mise à jour et du reverse engineering d'une base de données. Ces correspondances sont utilisées lorsque des types de données équivalents d'un point de vue fonctionnel mais qui diffèrent de ceux spécifiés dans le MPD sont trouvés dans une base de données existante, ce afin d'éviter l'affichage de différences inutiles et non pertinentes dans la boîte de dialogue de fusion. Exemples (ASE 15) : • Un type de données `unichar` est traité comme équivalant à un type de données `unichar(1)` dans une base de données ASE 15. • Un type de donnéesA `float(1)` est traité comme équivalant à un type de données `float(4)` dans une base de données ASE 15.
PhysLogADT Type	Répertorie les correspondances permettant de convertir les types de données abstraits de SGBD (**modèle physique**) en types de données abstraits (**internes**) de PowerAMC. Ces correspondances sont utilisées afin de renseigner la zone **Type** et d'afficher les propriétés appropriées dans les feuilles de propriétés de type de données abstrait ainsi que lorsque vous utilisez la commande **Changer de SGBD courant**. Exemples (Oracle 11g) : • Le type de données abstrait Oracle 11g `VARRAY` est converti en type de données `Array` pour PowerAMC. • Le type de données abstrait Oracle 11g `SQLJ_OBJECT` est converti en type de données `JavaObject` pour PowerAMC.

Elément	Description
LogPhysADT Type	Répertorie les correspondances permettant de convertir les types de données abstraits de PowerAMC (**internes**) en types de données abstraits de SGBD (**modèle physique**). Ces correspondances sont utilisées avec la commande **Changer de SGBD courant**. Exemples (Oracle 11g) : • Le type de données abstrait `List` de PowerAMC est converti en un type de données `TABLE` dans Oracle 11g. • Le type de données abstrait `Object` de PowerAMC est converti en un type de données `OBJECT` dans Oracle 11g.
AllowedADT	Répertorie les types de données abstraits qui peuvent être utilisés comme types pour les colonnes et les domaines dans le SGBD. Exemple (ASE 15) : • JAVA
HostDataType	Répertorie les correspondances permettant de convertir les types de données de SGBD (**modèle physique**) en types de données permis comme paramètres de procédure (**Trigger**). Ces correspondances sont utilisées afin de renseigner la zone **Type de données** dans les feuilles de propriétés de paramètre de procédure de type de données abstrait Exemples (Oracle 11g) : • Le type de données `DEC` de Oracle 11g est converti en type de données `number`. • Le type de données `SMALLINT` de Oracle 11g est converti en type de données `integer`.

Catégorie Profile (SGBD)

La catégorie Profile permet d'étendre les objets standard de PowerAMC. Vous pouvez affiner la définition, le comportement et l'affichage des objets existants en créant des attributs étendus, des stéréotypes, des critères, des formulaires, des symboles, des fichiers générés, etc, et ajouter de nouveaux objets en créant et stéréotypant des objets étendus et des sous-objets.

Vous pouvez ajouter des extensions soit dans :

- vos fichiers de définition de SGBD - vous devez en effectuer une sauvegarde avant de les modifier.

- un fichier d'extension séparé - que vous attachez au modèle.

Pour obtenir des informations détaillées sur l'utilisation des profils, y compris sur l'ajout d'attributs étendus et d'objets, voir *Chapitre 2, Fichiers d'extension* à la page 23.

Utilisation d'attributs étendus lors de la génération

Les attributs étendus peuvent être pris en compte lors de la génération. Chaque valeur d'attribut étendu peut être utilisée comme une variable qui peut être référencée dans les scripts définis dans la catégorie Script.

Certains SGBD incluent des attributs étendus prédéfinis. Par exemple, dans PostgreSQL, les domaines incluent les attributs étendus par défaut utilisés pour la création de types de données utilisateur.

Vous pouvez créer autant d'attributs étendus que nécessaire, pour chaque objet pris en charge par le SGBD.

Remarque : Les noms des variables PowerAMC tiennent compte de la casse des caractères. Le nom d'une variable doit correspondre à la casse près au nom d'attribut étendu.

Exemple

Par exemple, dans DB2 UDB 7 OS/390, l'attribut étendu `WhereNotNull` permet d'ajouter une clause qui impose l'unicité des noms d'index s'ils ne sont pas nuls.

Dans l'instruction `Create index`, `WhereNotNull` est évaluée comme suit :

```
create [%INDEXTYPE% ][%UNIQUE% [%WhereNotNull%?where not
null ]]index [%QUALIFIER%]%INDEX% on [%TABLQUALIFIER%]%TABLE% (
            %CIDXLIST%
)
[%OPTIONS%]
```

Si le nom d'index est unique, et si vous avez défini le type de l'attribut étendu `WhereNotNull` comme True, la clause "where not null" est insérée dans le script.

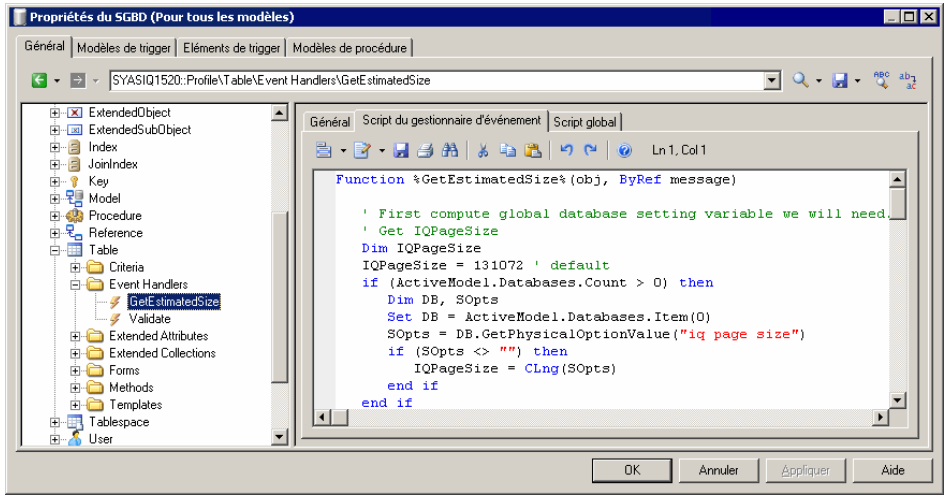
Dans l'élément `SqlListQuery` :

```
{OWNER, TABLE, INDEX, INDEXTYPE, UNIQUE, INDEXKEY, CLUSTER,
WhereNotNull}

select
 tbcreator,
 tbname,
 name,
 case indextype when '2' then 'type 2' else 'type 1' end,
 case uniquerule when 'D' then '' else 'unique' end,
 case uniquerule when 'P' then 'primary' when 'U' then 'unique' else
'' end,
 case clustering when 'Y' then 'cluster' else '' end,
 case uniquerule when 'N' then 'TRUE' else 'FALSE' end
from
 sysibm.sysindexes
where 1=1
[  and tbname=%.q:TABLE%]
[  and tbcreator=%.q:OWNER%]
[  and dbname=%.q:CATALOG%]
order by
 1 ,2 ,3
```

Modification du mécanisme d'estimation de taille de base de données

Par défaut, le mécanisme d'estimation de la taille de base de données utilise des algorithmes standard afin de calculer les tailles des tablespaces, tables, colonnes et index, et les additionne afin de fournir une indication de la taille que la base de données va requérir. Vous pouvez choisir de ne pas utiliser cet algorithme pour l'un ou plusieurs de ces types d'objet lors du calcul ou d'inclure des objets supplémentaires dans le calcul en ajoutant le gestionnaire d'événement `GetEstimatedSize` sur l'objet approprié dans la catégorie `Profile` et en saisissant un script afin de calculer sa taille.

1. Sélectionnez **SGBD > Editer le SGBD courant** pour afficher le fichier de définition du SGBD, puis développez la catégorie Profile.
2. Pointez sur la métaclasse pour laquelle vous souhaitez fournir un script afin de calculer la taille d'objet, sélectionnez **Nouveau > Gestionnaire d'événement** pour afficher une boîte de dialogue de sélection, sélectionnez le gestionnaire d'événement `GetEstimatedSize`, puis cliquez sur **OK** afin de l'ajouter à la métaclasse.
3. Cliquez sur l'onglet **Script du gestionnaire d'événement** dans le volet de droite, et saisissez le code approprié pour calculer la taille de l'objet choisi.



Dans l'exemple suivant, nous examinons des extraits du gestionnaire d'événement `GetEstimatedSize` défini sur la métaclasse `Table` afin d'estimer la taille de la base de données en calculant la taille de chaque table comme taille totale de toutes ses colonnes plus la taille de tous ses index.

Remarque : Pour obtenir des exemples du gestionnaire d'événement `GetEstimatedSize` utilisé dans Table et les autres métaclasses, voir les fichiers de définition de SGBD Sybase IQ v15.2 et HP Neoview R2.4.

Dans ce premier extrait du script, la fonction `GetEstimatedSize` s'ouvre et la taille de chaque table est obtenue en bouclant sur la taille de chacune de ses colonnes. Le calcul effectif de la taille de la colonne est effectué par la ligne :

```
ColSize = C.GetEstimatedSize(message, false)
```

, qui appelle le gestionnaire d'événement `GetEstimatedSize` sur la métaclasse `Column` (voir *Appel du gestionnaire d'événement GetEstimatedSize sur une autre métaclasse* à la page 238) :

```
Function %GetEstimatedSize%(obj, ByRef message)

' First compute global database setting variable we will need.

' Get table size and keep column size for future use
   Dim ColSizes, TblSize, ColSize, C
   Set ColSizes = CreateObject("Scripting.Dictionary")

   TblSize = 0 ' May be changed to take into account table
definition initial size.

  for each C in obj.Columns

     ' Start browsing table columns and use event handler defined
on column metaclass (if it exists).
```

```
      ColSize = C.GetEstimatedSize(message, false)

        ' Store column size in the map for future use in indexes.
      ColSizes.Add C, ColSize

        ' Increase the table global size.
      TblSize = TblSize + ColSize
   next
   Dim RawDataSize
   RawDataSize = BlockSize * int(obj.Number * TblSize / BlockSize)
    ' At this point, the RawDataSize is the size of table in
database.
```

Ensuite, la taille des index de table est calculée directement dans le script sans appeler de gestionnaire d'événement sur la métaclasse Index, la ligne qui produit les tailles d'index est mise en forme et la taille des index est ajoutée à la taille totale de la base de données :

```
' Now calculate index sizes. Set up variables to store indexes
sizes.
   Dim X, XMsg, XDataSize
   XMsg = ""
   for each X in obj.Indexes
      XDataSize = 0
      ' Browsing index columns and get their size added in
XDataSize
      For each C in X.IndexColumns
         XDataSize = XDataSize + ColSizes.Item(C.Column)
      next
      XDataSize = BlockSize * int(obj.Number * XDataSize /
BlockSize)

        ' Format the display message in order to get size
information in output and result list.
      XMsg = XMsg & CStr(XDataSize) & "|" & X.ObjectID & vbCrLf

      ' Add the index size to table size.
      RawDataSize = RawDataSize + XDataSize
   next
```

Pour finir, l'information de taille est mise en forme pour sortie (voir *Mise en forme du résultat d'une estimation de taille de base de données* à la page 238). Chaque table est imprimée sur une ligne distincte à la fois dans les fenêtres Résultats et Liste de résultats, et sa taille totale incluant toutes les colonnes et index est fournie :

```
    ' set the global message to table size and all indexes
(separate with carriage return).
   message = CStr(RawDataSize) & "||" & obj.ShortDescription &
vbCrLf & XMsg

   %GetEstimatedSize% = RawDataSize

End Function
```

Une fois que toutes les tables ont été traitées, PowerAMC calcule et imprimer la taille totale estimée de la base de données.

Appel du gestionnaire d'événement GetEstimatedSize sur une autre métaclasse

Vous pouvez appeler un gestionnaire d'événement `GetEstimatedSize` défini sur une autre métaclasse afin d'utiliser cette taille dans votre calcul. Pour exemple, vous pouvez définir `GetEstimatedSize` sur la métaclasse `Table`, et faire un appel vers le gestionnaire `GetEstimatedSize` défini sur les métaclasses `Column` et `Index` afin d'utiliser ces tailles pour calculer la taille totale de la table.

Le syntaxe de la fonction est la suivante, avec *message* qui représente le nom de votre variable contenant les résultats à imprimer :

```
GetEstimatedSize(message[,true|false])
```

En général, nous vous conseillons d'utiliser la fonction sous la forme suivante :

```
GetEstimatedSize(message, false)
```

L'utilisation du paramètre `false` (qui est la valeur par défaut, mais qui est affiché ici pour plus de clarté) signifie que l'on appelle le gestionnaire d'événement `GetEstimatedSize` sur d'autres métaclasses, et qu'on utilise le mécanisme par défaut uniquement si le gestionnaire d'événement n'est pas disponible.

Si vous affectez la valeur true au paramètre, vous forcez l'utilisation du mécanisme par défaut pour le calcul de la taille des objets (uniquement possible pour les tables, colonnes et join indexes) :

```
GetEstimatedSize(message, true)
```

Mise en forme du résultat d'une estimation de taille de base de données

Vous pouvez mettre en forme le résultat pour votre estimation de taille de base de données. Les sous-objets (par exemple les colonnes et les index) contenus dans une table sont affichés en décalé, et vous pouvez imprimer des informations supplémentaires après le total.

La syntaxe du résultat est la suivante :

```
[taille-objet][:compartiment]|[ObjectID][|libellé]
```

où :

- *taille-objet* - représente la taille de l'objet.
- *compartiment* - représente un nombre d'un chiffre, qui indique que la taille de l'objet doit être exclue de la taille totale de la base de données et doit être imprimée après le calcul de la taille de la base de données. Par exemple, vous pouvez décider d'inclure la taille des tables individuelles dans votre calcul de la taille de la base de données et d'imprimer les tailles des tablespaces hors du calcul.
- `ObjectID` - n'est pas nécessaire pour les objets, tels que les tables, mais est requis pour les sous-objets, si vous souhaitez les imprimer dans la Liste de résultats.
- *libellé* - toute chaîne identifiante appropriée, le plus souvent défini à `ShortDescription`, qui imprime le type et le nom de l'objet sélectionné.

Par exemple, dans le gestionnaire d'événement défini sur la métaclasse `Table` (ayant calculé et stocké la taille d'une table, la taille de toutes les colonnes de type LONG contenues dans la taille, ainsi que la taille de chaque index dans la table), nous créons un message variable pour imprimer cette information. Nous commençons par imprimer une ligne donnant la taille de la table :

```
message = CStr(TableSize) & "||" & objTable.ShortDescription & vbCrLf
```

Nous ajoutons ensuite une ligne qui imprime la taille totale de toutes les colonnes de type LONG de la table :

```
message = message & CStr(LongSize) & "||Colonnes de type LONG" &
vbCrLf
```

Nous ajoutons ensuite une ligne qui imprime la taille de chaque index dans la table :

```
message = message & CStr(IndexSize) & "|" & objIndex.ObjectID &
vbCrLf
```

Dans le gestionnaire d'événement défini sur la métaclasse `Tablespace` (en ayant calculé et stocké la taille d'un tablespace), nous créons un message variable pour imprimer cette information après avoir imprimé le calcul de la taille de base de données.

Nous commençons par remplacer l'introduction par défaut de ce second compartiment :

```
message = ":1||Des tables sont allouées aux tablespaces suivants :"
```

Nous ajoutons une ligne qui imprime la taille de chaque tablespace dans la table :

```
message = message + CStr(tablespaceSize) & ":1||" &
objTablespace.ShortDescription
```

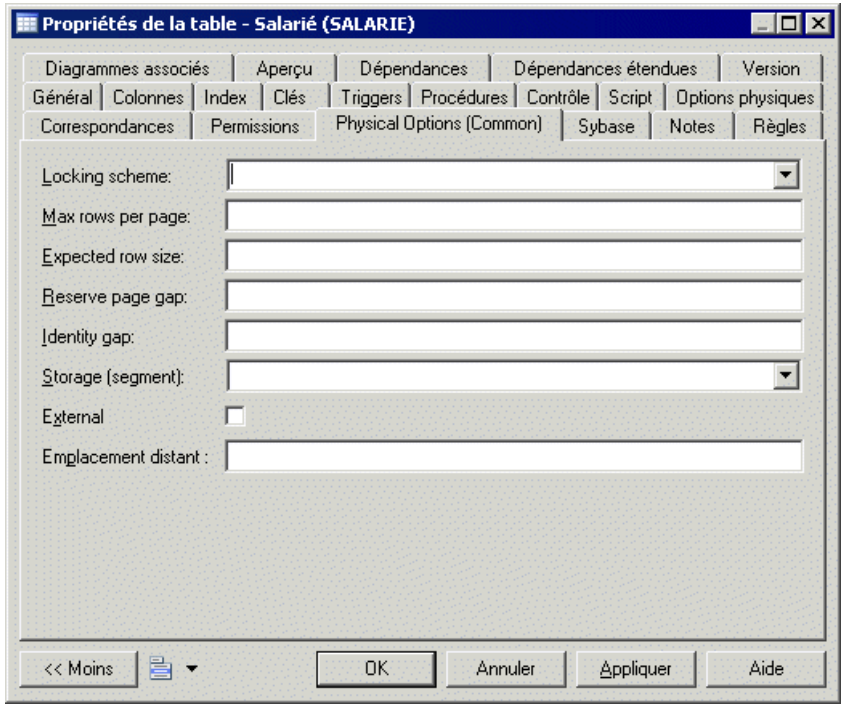
Le résultat se présente comme suit :

```
Estimation de la taille de la base de données "Sales"...

 Nombre    Taille estimée      Objet
-------    --------------      -------------------------------------

 10,000           6096 Ko      Table 'Invoices'
                                 Colonnes de type LONG (35 KB)
                                 Index 'customerFKeyIndex' (976 KB)
                                 Index 'descriptionIndex' (1976 KB)

                            [...etc...]

 Des tables sont allouées aux tablespaces suivants :

           Taille estimée      Objet
           --------------      -------------------------------------
                  6096 Ko      Tablespace 'mainStorage'

                            [...etc...]
```

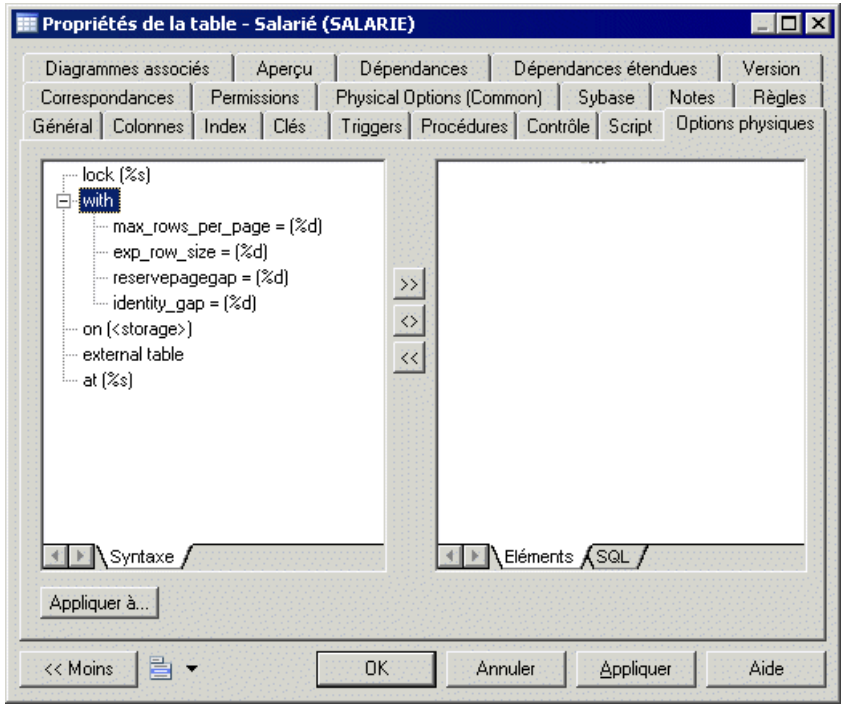
Options physiques

Dans certains SGBD, les options physiques sont utilisées pour spécifier comment un objet est optimisé ou stocké dans la base de données. Vous définissez des options physiques dans les feuilles de propriétés d'objet en utilisant les onglets suivants :

- Physical Options (common) – affiche les options physiques les plus couramment définies pour l'objet dans format de formulaire standard :



- Options physiques – affiche toutes les options physiques pour l'objet sous la forme d'une arborescence :



Remarque : L'onglet Physical Options (common) est configurable et les options qui y sont affichées sont associées à des attributs étendus. Vous pouvez ajouter des options supplémentaires sur cet onglet ou ajouter votre propre onglet en associant ces options à des attributs étendus. Pour plus d'informations, voir *Ajout d'options physiques de SGBD dans vos formulaires* à la page 85.

Pour plus d'informations sur la définition d'options physiques, voir *Modélisation des données > Construction de modèles de données > Mise en oeuvre physique > Options physiques*.

Syntaxe des options physiques

Si les options physiques sont prises en charge pour un objet, elles sont stockées dans l'entrée Options sous l'objet dans la catégorie Script/Object du fichier de ressource de SGBD.

Pour plus d'informations, voir *Eléments communs aux différents objets* à la page 171. Les valeurs par défaut sont stockées dans l'entrée DefOptions.

Lors de la génération, les options sélectionnées dans le modèle pour chaque objet sont stockées sous la forme d'une chaîne SQL dans la variable %OPTIONS%, qui doit s'afficher à la fin de l'élément auquel elle appartient et ne doit être suivie de rien. L'exemple suivant illustre la syntaxe correcte :

```
create table
[%OPTIONS%]
```

Lors du reverse engineering par script, la section de la requête SQL déterminée comme constituant les options physiques est stockée dans %OPTIONS%, et sera analysée lorsque nécessaire par la feuille de propriétés d'un objet.

Lors du reverse engineering direct, l'instruction SQL `SqlOptsQuery` est exécutée pour récupérer les options physiques stockées dans %OPTIONS% afin de les analyser lorsque nécessaire par la feuille de propriétés d'un objet.

Vous pouvez utiliser les variables PowerAMC (voir *Variables de MPD* à la page 247) afin de définir les options physiques pour un objet. Par exemple, dans Oracle, vous pouvez définir la variable suivante pour un cluster afin que ce cluster prenne le même nom que la table.

```
Cluster %TABLE%
```

Définition d'options physiques spécifiées par une valeur

Les éléments d'option contiennent du texte utilisé pour afficher l'option sur les onglets d'options physiques. Les éléments d'option peuvent contenir des variables %d ou %s pour permettre à l'utilisateur de spécifier une valeur. Par exemple :

```
with max_rows_per_page=%d
on %s: category=storage
```

- la variable %d - requiert une valeur numérique
- variable %s - requiert une valeur de chaîne

Les variables incluses entre les signes % (%--%) ne sont pas admises dans les options physiques.

Vous pouvez spécifier une contrainte (une liste de valeurs, des valeurs par défaut, la valeur doit être un storage ou un tablespace, certaines lignes peuvent être groupées) sur n'importe quelle ligne contenant une variable. Ces contraintes sont introduites par une virgule directement derrière l'option physique et séparées par des virgules.

Exemple

`With max_rows_per_page` est une option physique d'index pour Sybase AS Enterprise 11.x. Cette option limite le nombre de lignes par page de données. La syntaxe se présente comme suit :

```
with max_row_per_page = x
```

L'option `with max_rows_per_page` s'affiche sur l'onglet Options avec la valeur par défaut zéro (0) :

Cette option est définie dans le fichier de définition de SGBD comme suit :

```
with max_rows_per_page=%d
on %s : category=storage
```

Les variables `%d` et `%s` doivent se trouver en dernière position et ne doivent pas être suivies d'autres options.

Options physiques sans nom

Une ligne dans une entrée d'option doit comporter un nom afin de pouvoir être identifiée par PowerAMC. Si une option physique est dépourvue de nom, vous devez ajouter un nom entre des signes inférieur et supérieur <> avant l'option.

Pour définir un segment dans Sybase AS Enterprise 11, la syntaxe appropriée est la suivante :

```
sp_addsegment segmentname, databasename, devicename
```

`segmentname` correspond au code de storage défini dans PowerAMC. `databasename` correspond au code de modèle. Ces deux entrées sont automatiquement générées. `devicename` doit être saisi par l'utilisateur, il devient une option.

Dans SYSTEM11, cette option est définie comme suit :

```
Create = execute sp_addsegment %STORAGE%, %DATABASE%, %OPTIONS%
OPTIONS = <devname> %s
```

Une option physique dépourvue de nom doit être suivie de la variable `%d` ou `%s`.

Définition d'une valeur par défaut pour une option physique

Une option physique peut avoir une valeur par défaut, spécifiée par le mot clé `Default= x`, placé après le nom de l'option ou après la valeur `%d` ou `%s`, et séparé par un signe deux points.

Exemple

La valeur par défaut pour `max_row_per_page` est 0. Dans Sybase Adaptive Server® Enterprise 11, cette valeur par défaut pour l'objet index est définie comme suit :

```
max_rows_per_page=%d : default=0
```

Définition d'une liste de valeurs pour une option physique

Lorsque vous utilisez les variables `%d` et `%s`, une valeur d'option physique peut correspondre à une liste d'options possibles, spécifiées par le mot clé `list= x | y`, placé après le nom d'option ou après la valeur `%d` ou `%s`, et séparé par un signe deux points. Les valeurs possibles sont séparées par le caractère `|`.

Par exemple, l'option `dup_prow` d'un index Sybase ASE 11 a deux options mutuellement exclusives pour créer un index non-unique clustered :

```
IndexOption =
<duprow> %s: list=ignore_dup_row | allow_dup_row
```

Une liste avec les valeurs est affichée sur l'onglet Options physiques.

Remarque : Si `Default=` et `List=` sont utilisés simultanément, ils doivent être séparés par une virgule. Par exemple : IndexOption = <duprow> %s: `default= ignore_dup_row`, list=ignore_dup_row | allow_dup_row

Définition d'une option physique pour un tablespace ou un storage

Une option physique peut utiliser le code d'un tablespace ou d'un storage : `category=tablespace` ou `category=storage` construit une liste avec tous les codes de tablespace ou de storage définis dans la boîte de dialogue Liste des tablespaces ou Liste des storages.

Par exemple, dans Sybase ASE 11, l'option `on segmentname` spécifie que l'index est créé sur le segment spécifié. Un segment ASE correspond à un storage PowerAMC. La syntaxe se présente comme suit :

```
on segmentname
```

La valeur par défaut pour l'index est définie dans l'élément d'option comme suit :

```
on %s: category=storage
```

Une liste de valeurs est affichée sur les onglets Options physiques.

Syntaxe d'option physique composite

Une option physique composite est une option physique qui inclut d'autres options dépendantes. Ces options sont sélectionnées simultanément dans le volet droit de l'onglet d'options physiques.

La syntaxe standard pour les options physiques se présente comme suit :

```
with : composite=yes, separator=yes, parenthesis=no
{
fillfactor=%d : default=0
max_rows_per_page=%d : default=0
}
```

L'option physique `With` inclut les autres options entre accolades { }, séparées par une virgule. Pour définir une option composite, vous devez utiliser le mot clé composite.

Mot clé	Valeur et résultat
composite	Les valeurs possibles sont les suivantes : • yes - des accolades sont utilisées pour définir une option physique composite • no – les accolades ne peuvent pas être utilisées
separator	Les valeurs possibles sont les suivantes : • yes - les options sont séparées par une virgule • no [valeur par défaut] - les options sont dépourvues de caractère séparateur

Mot clé	Valeur et résultat
parenthesis	Les valeurs possibles sont les suivantes : • yes - l'option composite est délimitée par des parenthèses qui incluent toutes les autres options, par exemple : with (max_row_per_page=0, ignore_dup_key) • no [valeur par défaut] - rien ne délimite l'option composite
nextmand	Les valeurs possibles sont les suivantes : • yes - la prochaine ligne dans l'option physique est obligatoire. • no - vous ne serez pas en mesure de procéder à la génération/au reverse engineering de l'intégralité de l'option physique composite
prevmand	Les valeurs possibles sont les suivantes : • yes - la ligne précédente dans l'option physique est obligatoire • no - vous ne serez pas en mesure de procéder à la génération/au reverse engineering de l'intégralité de l'option physique composite
chldmand	Les valeurs possibles sont les suivantes : • yes - il doit y avoir au moins une ligne enfant • no – les enfants ne sont pas obligatoires

category	Les valeurs possibles sont les suivantes :
	• tablespace - l'élément est lié à un tablespace
	• storage - l'élément est lié à un storage
	`storage : category=storage, composite=yes, separator=no, parenthesis=yes`
	`{`
	Remarque : Dans Oracle, l'option physique composite `storage` est utilisée comme modèle pour définir toutes les valeurs de storage dans une entrée de storage. Ceci vous évite d'avoir à définir des valeurs indépendamment chaque fois que vous devez utiliser les mêmes valeurs dans une clause de storage. Par conséquence, l'option physique Oracle n'inclut pas le nom de storage (%s) :

list	Liste dans laquelle des valeurs sont séparées par un trait vertical (|)
dquoted	Les valeurs possibles sont les suivantes : • yes - la valeur est placée entre guillemets (" ") • no - la valeur n'est pas placée entre guillemets (" ")

Mot clé	Valeur et résultat
squoted	Les valeurs possibles sont les suivantes : • yes - la valeur est placée entre apostrophes ("" "") • no - la valeur n'est pas placée entre apostrophes (' ')
enabledbprefix	Les valeurs possibles sont les suivantes : • yes - le nom de base de données est utilisé comme préfixe (voir les options de tablespace dans DB2 OS/390) • no - le nom de base de données n'est pas utilisé comme préfixe

`Default=` et/ou `List=` peut également être utilisé avec les mots clés `composite=`, `separator=` et `parenthesis=`. `Category=` peut être utilisé avec les trois mots clés d'une option composite.

Exemple

Les options relatives aux index IBM DB2 contiennent l'option composite suivante :

```
<using_block> : composite=yes
{
 using vcat %s
 using stogroup %s : category=storage, composite=yes
 {
  priqty %d : default=12
  secqty %d
  erase %s : default=no, list=yes | no
 }
```

Répétitions d'options

Certaines bases de données répètent un bloc d'options, groupées dans une option composite. Dans ce cas, la définition composite contient le mot clé `multiple` :

```
with: composite=yes, multiple=yes
```

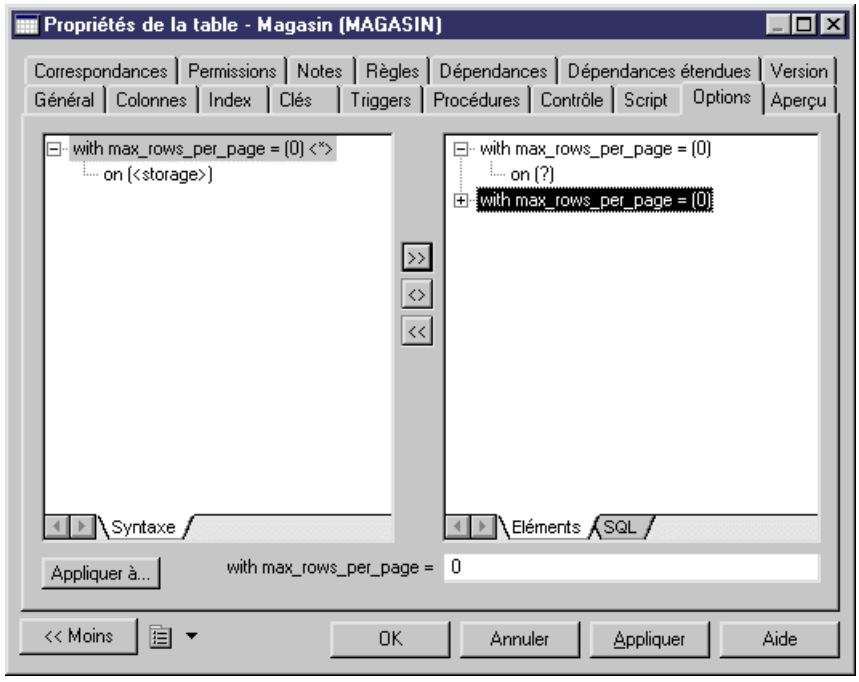
Par exemple, les options de fragmentation Informix peuvent être répétées *n* fois comme illustré ci-après :

```
IndexOption =
fragment by expression : composite=yes, separator=yes
{
 <list> : composite=yes, multiple=yes
 {
  <frag-expression> %s
  in %s : category=storage
 }
 remainder in %s : category=storage
}
```

La sous-option `<list>` est utilisée pour éviter d'avoir à répéter le mot clé `fragment` avec chaque nouveau bloc d'options.

Lorsque vous répétez une option composite, cette option s'affiche avec <*> dans le volet des options physiques disponibles (volet gauche) sur l'onglet des options physiques.

```
max_rows_per_page=0 <*>
```



Vous pouvez ajouter l'option composite dans le volet droit plusieurs fois en utilisant le bouton Ajouter entre les volets, sur l'onglet d'options physiques.

Si la sélection se trouve sur l'option composite dans le volet droit et que vous cliquez sur la même option composite dans le volet gauche afin de l'ajouter, une boîte de message vous demande si vous souhaitez réutiliser l'option sélectionnée. Si vous cliquez sur Non, l'option composite est ajoutée dans le volet droit comme nouvelle ligne.

Variables et macros de MPD

Les requêtes SQL enregistrées dans les éléments du fichier de définition de SGBD utilisent diverses variables de MPD. Ces variables sont remplacées par des valeurs de votre modèle lors de la génération de scripts, et sont évaluées pour créer des objets PowerAMC lors du reverse engineering.

Les variables PowerAMC sont écrites entre signes pourcent (%).

Exemple

```
CreateTable = create table %TABLE%
```

L'évaluation des variables dépend des paramètres et du contexte. Par exemple, la variable %COLUMN% ne peut pas être utilisée dans un paramètre CreateTablespace, car elle n'est valide que dans le contexte d'un paramètre de colonne.

Lorsque vous référencez des attributs d'objet, vous pouvez utiliser les variables suivantes, ou les noms publics disponibles via le métamodèle PowerAMC (voir *Chapitre 5, Personnalisation de la génération à l'aide du langage de génération par template* à la page 285 et *Chapitre 1, Utilisation des fichiers de ressources PowerAMC* à la page 1.

Test des valeurs de variables à l'aide des opérateurs [ ]

Vous pouvez utiliser des crochets [ ] afin de tester l'existence ou la valeur d'une variable.

Vous pouvez utiliser les crochets pour :

- Inclure des chaînes et variables facultatives, ou des listes de chaînes et de variables dans la syntaxe des instructions SQL : [%*variable*%]
- Tester la valeur d'une variable et insérer ou reconsidérer une valeur en fonction du résultat du test : [%*variable*%? *true* : *false*]
- Tester le contenu d'une variable [%*variable*%=*constante*? *true* : *false*]

Variable	Génération
[%*variable*%]	Teste l'existence de la variable. Génération : Généré uniquement si *variable* existe et n'a pas la valeur NO ou FALSE. Reverse engineering : Evalué si l'analyseur détecte une instruction SQL correspondant à la variable et n'ayant pas la valeur NO ou FALSE.
[%*variable*%? *true* : *false*]	Teste l'existence de la variable et permet un résultat conditionnel. Génération : *true* est généré si *variable* existe et n'a pas la valeur NO ou FALSE. Dans le cas contraire, c'est *false* qui est généré. Reverse engineering : Si l'analyseur détecte *variable* et que cette dernière n'a pas la valeur NO ou FALSE, le reverse engineering renvoie *true*. Dans le cas contraire, le reverse engineering renvoie *false*. *variable* est défini à True ou False, selon le cas.

Variable	Génération
[%*variable*%=*constant*? *true* : *false*]	Teste l'existence de la variable et permet un résultat conditionnel. Génération : si *variable* est égal à *constant*, *true* est généré. Dans le cas contraire, c'est *false* qui est généré. Reverse engineering : Si l'analyseur détecte que *variable* est égal à *constant*, le reverse engineering renvoie *true*. Dans le cas contraire, le reverse engineering renvoie *false*.
[.Z: [*item1*] [*item2*]...]	Spécifie que les *items* n'ont pas un ordre significatif. Génération : .Z est ignoré Reverse engineering : Les *items* peuvent être récupérés par reverse engineering dans l'ordre où ils sont rencontrés.
[.O: [*item1*] [*item2*]...]	Spécifie que les *items* sont synonymes, l'un seul d'entre eux pouvant être produit. Génération : Seul le premier *item* répertorié est généré. Reverse engineering : L'analyseur du reverse engineering doit trouver l'un des *items* pour valider toute l'instruction.

Exemples

- `[%OPTIONS%]`

  Si %OPTIONS% (options physiques pour les objets visibles dans la feuille de propriétés d'objet) existe et n'a pas la valeur NO ou FALSE, il est généré avec la valeur de %OPTIONS%.
- `[default %DEFAULT%]`

  Si l'instruction `default 10` est rencontrée lors du reverse engineering, %DEFAULT% se voit attribuer la valeur 10, mais l'instruction n'est pas obligatoire et le reverse engineering continue même en son absence. Dans le cadre de la génération de script, si %DEFAULT% a la valeur 10, il est généré comme `default 10`, dans le cas contraire rien n'est généré pour le bloc.
- `[%MAND%? not null : null ]`

  Si %MAND% est évalué comme true ou contient une valeur autre que False ou NO, il est généré comme `not null`. Dans le cas contraire, il est généré comme `null`.
- `[%DELCONST%=RESTRICT?:[on delete %DELCONST%]]`

  Si %DELCONST% contient la valeur RESTRICT, il est généré comme `on delete RESTRICT`.
- `%COLUMN% %DATATYPE%[.Z: [%NOTNULL%][%DEFAULT%]]`

  En raison de la présence de la variable .Z, les deux instructions suivantes seront correctement récupérées par le reverse engineering et ce, même si les attributs de colonne ne sont pas dans le même ordre :

- `Create table abc (a integer not null default 99)`
- `Create table abc (a integer default 99 not null)`

- `[.O:[procedure][proc]]`

  Cette instruction va générer `procedure`. Lors du reverse engineering, l'analyseur syntaxique va faire correspondre les mots clés `procedure` ou `proc`.

- **Remarque :** Une chaîne entre crochets est systématiquement générée. Dans le cadre du reverse engineering, le fait de placer une chaîne entre crochets signifie qu'elle est facultative et que son absence ne va pas annuler le reverse engineering de l'instruction.

```
create [or replace] view %VIEW% as %SQL%
```

Une script contenant soit `create` ou `create or replace` sera correctement récupéré par reverse engineering car `or replace` est facultatif.

Mise en forme des valeurs de variable

Vous pouvez spécifier le format pour les valeurs de variable. Par exemple, vous pouvez forcer des valeurs en minuscules ou majuscules, tronquer ces valeurs ou les placer entre guillemets.

Vous devez incorporer les options de format dans la syntaxe de variable comme suit :

```
%[[?][-][width][.[-]precision][c][H][F][U|L][T][M][q][Q]:]<varname>
%
```

Les options de format des variables sont les suivantes :

Option	Description
?	Champ obligatoire, si une valeur nulle est renvoyée, l'appel de conversion échoue
n (n, étant un entier)	Ajoute des espaces ou des zéros à droite pour remplir la largeur et justifier à gauche
-*n*	Ajoute des espaces ou des zéros à gauche pour remplir la largeur et justifier à droite
width	Copie le nombre minimal spécifié de caractères dans la mémoire tampon de sortie
.[-]precision	Copie le nombre maximal spécifié de caractères dans la mémoire tampon de sortie
.L	Force les caractères en minuscules
.U	Force les caractères en majuscules
.F	Combiné avec L et U, applique des conversions au premier caractère
.T	Les espaces de début et de fin sont supprimés de la variable
.H	Convertit le nombre en hexadécimal
.c	Force la majuscule à la première lettre ainsi que des minuscules aux autres lettres du mot

Option	Description
.*n*	Tronque la valeur pour ne conserver que les *n* premiers caractères
.-*n*	Tronque la valeur pour ne conserver que les *n* derniers caractères
M	Extrait une partie du nom de la variable, cette option utilise les paramètres de largeur et de précision pour identifier la partie à extraire
q	Place la variable entre apostrophes
Q	Place la variable entre guillemets

Vous pouvez combiner les codes de format. Par exemple, %.U8:CHILD% met en forme le code de la table enfant avec un maximum de huit caractères majuscules.

Examples

Les exemples suivants montrent les codes de format incorporés dans la syntaxe de variable pour le modèle de nom de contrainte des clés primaires, en utilisant une table nommée CUSTOMER_PRIORITY :

Format	Utilisation
.L	Minuscules. Exemple : `PK_%.L:TABLE%` Résultat : `PK_customer_priority`
.U*n*	Majuscules + texte de variable justifié à droite jusqu'à une longueur fixe, *n* représente le nombre de caractères. Exemple : `PK_%.U12:TABLE%` Résultat : `PK_CUSTOMER_PRI`
.T	Supprimer les espaces de début et de fin de la variable. Exemple : `PK_%.T:TABLE%` Résultat : `PK_customer_priority`
.*n*	Longueur maximum dans laquelle *n* représente le nombre de caractères. Exemple : `PK_%.8:TABLE%` Résultat : `PK_Customer`

Format	Utilisation
-n	Complète le résultat avec des espaces à droite pour afficher une longueur fixe dans laquelle *n* représente le nombre de caractères. Exemple : `PK_%-20:TABLE%` Résultat : `PK_ Customer_priority`
M	Extrait une partie de la variable. Exemple : `PK%3.4M:TABLE%` Résultat : `PK_CUST`

Variables communes pour les objets

Ces variables peuvent être utilisées pour tous les objets qui prennent en charge ces concepts.

Variable	Commentaire
%COMMENT%	Commentaire de l'objet ou son nom (si aucun commentaire n'est défini)
%OWNER%	Code généré de l'utilisateur possédant l'objet ou son parent. Vous ne devez pas utiliser cette variable pour les requêtes sur les objets répertoriés dans les boîtes de dialogue de reverse engineering direct, car leur propriétaire n'est alors pas encore défini.
%DBPREFIX%	Préfixe de base de données des objets (nom de la base + '.' si la base de données est définie)
%QUALIFIER%	Qualifiant de l'objet entier (préfixe de la base + préfixe du propriétaire)
%OPTIONS%	Texte SQL définissant les options physiques pour l'objet
%OPTIONSEX%	Texte analysé définissant les options physiques de l'objet
%CONSTNAME%	Nom de contrainte de l'objet
%CONSTRAINT%	Corps de la contrainte SQL de l'objet. Ex : (A <= 0) AND (A >= 10)
%CONSTDEFN%	Définition de contrainte de colonne. Ex : contraint C1 checks (A>=0) AND (A<=10)
%RULES%	Concaténation d'expression serveur des règles de gestion associée à l'objet
%NAMEISCODE%	True si le nom et le code de l'objet (table, colonne, index) sont identiques (spécifique AS/400)
%TABLQUALIFIER%	Qualifiant de la table parent (préfixe de base de données + préfixe de propriétaire)
%TABLOWNER%	Code généré de l'utilisateur propriétaire de la table parent

Les variables suivantes sont disponibles pour tous les objets nommés :

Variable	Commentaire
%@OBJTNAME%	Nom de l'objet
%@OBJTCODE%	Code de l'objet
%@OBJTLABL%	Commentaire de l'objet
%@OBJTDESC%	Description de l'objet

Les variables de métadonnées suivantes sont disponibles:

Variable	Commentaire
@CLSSNAME	Nom localiser pour une classe d'objet. Par exemple : Table, Vue, Colonne, Index
@CLSSCODE	Code de la classe d'objet. Par exemple : TABL, VIEW, COLN, INDX

Variables pour les tables et les vues

PowerAMC peut utiliser des variables lors de la génération et du reverse-engineering tables et vues.

Les variables suivantes sont disponibles pour les tables :

Variable	Commentaire
%TABLE%	Code généré pour la table
%TNAME%	Nom de la table
%TCODE%	Code de la table
%TLABL%	Commentaire de la table
%PKEYCOLUMNS%	Liste des colonnes de clé primaire. Ex : A, B
%TABLDEFN%	Corps complet de la définition de table. Contient la définition des colonnes, des contrôles et des clés
%CLASS%	Nom de type de données abstrait
%CLASSOWNER%	Propriétaire de l'objet classe
%CLASSQUALIFIER%	Qualifiant de l'objet classe
%CLUSTERCOLUMNS%	Liste des colonnes utilisées pour un cluster
%INDXDEFN%	Définition d'index de table
%TABLTYPE%	Type de table

Les variables suivantes sont disponibles pour les vues :

Variable	Commentaire
%VIEW%	Code généré pour la vue
%VIEWNAME%	Nom de la vue
%VIEWCODE%	Code de la vue
%VIEWCOLN%	Liste des colonnes de la vue. Ex : "A, B, C"
%SQL%	Texte SQL de la vue. Ex : Select * from T1
%VIEWCHECK%	Contient le mot clé "with check option" si cette option est sélectionnée dans la vue
%SCRIPT%	Commande complète de création de la vue. Ex : create view V1 as select * from T1
%VIEWSTYLE%	Style de la vue : view, snapshot, materialized view
%ISVIEW%	True s'il s'agit d'une vue (et non pas d'un snapshot)
%USAGE%	Read-only=0, Updatable=1, Check option=2

Les variables suivantes sont disponibles pour les tables et vues :

Variable	Commentaire
%XMLELEMENT%	Elément contenu dans le schéma XML
%XMLSCHEMA%	Schéma XML

Variables pour les colonnes, domaines et contraintes

PowerAMC peut utiliser des variables lors de la génération et du reverse-engineering de colonnes, domaines et contraintes. Les variables des tables parent sont également disponibles.

Les variables suivantes sont disponibles pour les colonnes :

Variable	Commentaire
%COLUMN%	Code généré pour la colonne
%COLNNO%	Position de la colonne dans la liste des colonnes de la table
%COLNNAME%	Nom de la colonne
%COLNCODE%	Code de la colonne
%PRIMARY%	Contient le mot clé "primaire" si la colonne est une colonne de clé primaire

Variable	Commentaire
%ISPKEY%	TRUE si la colonne fait partie d'une clé primaire
%ISAKEY%	TRUE si la colonne fait partie d'une clé alternative
%FOREIGN%	TRUE si la colonne fait partie d'une clé étrangère
%COMPUTE%	Calcul du texte de la contrainte
%PREVCOLN%	Code de la colonne précédente dans la liste des colonnes de la table
%NEXTCOLN%	Code de la colonne suivante dans la liste des colonnes de la table
%NULLNOTNULL%	Statut obligatoire d'une colonne. Cette variable est systématiquement utilisée avec NullRequired, voir *Gestion des valeurs Null* à la page 188
%PKEYCLUSTER%	Mot clé CLUSTER pour la clé primaire lorsqu'elle est définie sur la même ligne
%AKEYCLUSTER%	Mot clé CLUSTER pour la clé alternative lorsqu'elle est définie sur la même ligne
%AVERAGELENGTH%	Longueur moyenne
%ISVARDTTP%	TRUE si le type de données de la colonne a une longueur variable
%ISLONGDTTP%	TRUE si le type de données de la colonne a un type de données long mais qu'il ne s'agit ni d'une image ni d'un blob
%ISBLOBDTTP%	TRUE si le type de données de la colonne est une image ou un blob
%ISSTRDTTP%	TRUE si le type de données de la colonne contient des caractères

Les variables suivantes sont disponibles pour les domaines :

Variable	Commentaire
%DOMAIN%	Code généré du domaine (disponible également pour les colonnes)
%DEFAULTNAME%	Nom de l'objet par défaut associé au domaine (spécifique à SQL Server)

Les variables suivantes sont disponibles pour les contraintes :

Variable	Commentaire
%UNIT%	Attribut Unité des paramètre de contrôle
%FORMAT%	Attribut Format des paramètre de contrôle

Variable	Commentaire
%DATATYPE%	Type de données. Ex : int, char(10) ou numeric(8, 2)
%DTTPCODE%	Code du type de données. Ex : int, char ou numeric
%LENGTH%	Longueur du type de données. Ex : 0, 10 ou 8
%PREC%	Précision du type de données. Ex : 0, 0 ou 2
%ISRDONLY%	TRUE si l'attribut Lecture seule est sélectionné dans les paramètres de contrôle standard
%DEFAULT%	Valeur par défaut
%MINVAL%	Minimum value
%MAXVAL%	Valeur maximum
%VALUES%	Liste des valeurs. Ex : (0, 1, 2, 3, 4, 5)
%LISTVAL%	Contrainte SQL associée à la liste des valeurs. Ex : C1 in (0, 1, 2, 3, 4, 5)
%MINMAX%	Contrainte SQL associée aux valeurs minimale et maximale. Ex : (C1 <= 0) AND (C1 >= 5)
%ISMAND%	TRUE si le domaine ou la colonne est obligatoire
%MAND%	Contient le mot clé "null" ou "not null" selon la valeur de l'attribut Obligatoire
%NULL%	Contient le mot clé "null" si le domaine ou la colonne est obligatoire
%NOTNULL%	Contient le mot clé "not null" si le domaine ou la colonne est obligatoire
%IDENTITY%	Mot clé "identity" si le domaine ou la colonne est de type Identity (spécifique Sybase)
%WITHDEFAULT%	Mot clé "with default" si le domaine ou la colonne est de type With default
%ISUPPERVAL%	TRUE si l'attribut Majuscules est sélectionné dans les paramètres de contrôle standard
%ISLOWERVAL%	TRUE si l'attribut Minuscules est sélectionné dans les paramètres de contrôle standard
%UPPER%	Contrainte SQL associée aux valeurs en majuscules uniquement
%LOWER%	Contrainte SQL associée aux valeurs en minuscules uniquement

Variable	Commentaire
%CASE%	Contrainte SQL associée aux casses (majus, minus, initiale majus, etc)

Variables pour les clés

PowerAMC peut utiliser des variables lors de la génération et du reverse-engineering des clés.

Nom de la variable	Commentaire
%COLUMNS% ou %COLNLIST%	Liste des colonnes de la clé. Ex : "A, B, C"
%ISPKEY%	TRUE lorsque la clé est une clé primaire de table
%PKEY%	Nom de contrainte de clé primaire
%AKEY%	Nom de contrainte de clé alternative
%KEY%	Nom de contrainte de la clé
%ISMULTICOLN%	True si la clé porte sur plusieurs colonnes
%CLUSTER%	Mot clé cluster

Variables pour les index et colonnes d'index

PowerAMC peut utiliser des variables lors de la génération et du reverse-engineering des index et colonnes d'index.

Les variables suivantes sont disponibles pour les index :

Variable	Commentaire
%INDEX%	Code généré de l'index
%TABLE%	Code généré du parent d'un index, il peut s'agir d'une table ou d'une table de requête (vue)
%INDEXNAME%	Nom de d'index
%INDEXCODE%	Code d'index
%UNIQUE%	Contient le mot clé "unique" lorsque l'index est unique
%INDEXTYPE%	Contient le type d'index (disponible uniquement pour certains SGBD)
%CIDXLIST%	Liste des codes d'index avec séparateur, sur la même ligne. Exemple : A asc, B desc, C asc
%INDEXKEY%	Contient le mot clé "primary", "unique" ou "foreign" en fonction de l'origine de l'index

Variable	Commentaire
%CLUSTER%	Contient le mot clé "cluster" si l'index est de type cluster
%INDXDEFN%	Utilisée pour définir un index au sein d'une définition de table

Les variables suivantes sont disponibles pour les colonnes d'index :

Variable	Commentaire
%ASC%	Contient le mot clé "ASC" ou "DESC" en fonction de l'ordre de tri
%ISASC%	TRUE si l'ordre de tri de la colonne est ascendant

Variables pour les références et les colonnes de référence

PowerAMC peut utiliser des variables lors de la génération et du reverse-engineering des références et colonnes de référence.

Les variables suivantes sont disponibles pour les références :

Variable	Commentaire
%REFR%	Code généré de la référence
%PARENT%	Code généré de la table parent
%PNAME%	Nom de la table parent
%PCODE%	Code de la table parent
%PQUALIFIER%	Qualifiant de la table parent. Voir aussi QUALIFIER.
%CHILD%	Code généré de la table enfant
%CNAME%	Nom de la table enfant
%CCODE%	Code de la table enfant
%CQUALIFIER%	Qualifiant de la table enfant. Voir aussi QUALIFIER.
%REFRNAME%	Nom de la référence
%REFRCODE%	Code de la référence
%FKCONSTRAINT%	Nom de contrainte de clé étrangère (référence)
%PKCONSTRAINT%	Nom de contrainte de la clé primaire utilisée pour faire référence à l'objet
%CKEYCOLUMNS%	Liste des colonnes de clé parent. Ex : C1, C2, C3
%FKEYCOLUMNS%	Liste des colonnes de clé étrangère. Ex : C1, C2, C3

Variable	Commentaire
%UPDCONST%	Contient des mots clés de contrainte déclarative pour les modifications : "restrict", "cascade", "set null" ou "set default"
%DELCONST%	Contient des mots clés de contrainte déclarative pour les suppressions : "restrict", "cascade", "set null" ou "set default"
%MINCARD%	Cardinalité minimale
%MAXCARD%	Cardinalité maximale
%POWNER%	Nom du propriétaire de la table parent
%COWNER%	Nom du propriétaire de la table enfant
%CHCKONCMMT%	TRUE si vous avez coché la case "check on commit" pour la référence (spécifique à ASA 6.0)
%REFRNO%	Numéro de référence dans la collection de référence de la table enfant
%JOINS%	Jointures de référence

Les variables suivantes sont disponibles pour les colonnes de référence :

Variable	Comment
%CKEYCOLUMN%	Code généré de la colonne de table parent (clé primaire)
%FKEYCOLUMN%	Code généré de la colonne de table enfant (clé étrangère)
%PK%	Code généré de la colonne de clé primaire
%PKNAME%	Nom de la colonne de clé primaire
%FK%	Code généré de la colonne de clé étrangère
%FKNAME%	Nom de la colonne de clé étrangère
%AK%	Code de la colonne de clé alternative (identique à PK)
%AKNAME%	Nom de la colonne de clé alternative (identique à PKNAME)
%COLTYPE%	Type de données de la colonne de clé primaire
%COLTYPENOOWNER%	Propriétaire de la colonne de clé primaire
%DEFAULT%	Valeur par défaut de la colonne de clé étrangère
%HOSTCOLTYPE%	Type de données de colonne de clé primaire utilisé pour la déclaration de procédure. Par exemple : without length

Variables pour les triggers et procédures

PowerAMC peut utiliser des variables lors de la génération et du reverse-engineering des triggers et procédures. Les variables de table parent sont également disponibles.

Les variables suivantes sont disponibles pour les triggers :

Variable	Commentaire
%ORDER%	Numéro d'ordre du trigger (si le SGBD prend en charge plusieurs trigger de même type)
%TRIGGER%	Code généré du trigger
%TRGTYPE%	Type de trigger. Contient les mots clés "beforeinsert", "afterupdate", etc.
%TRGEVENT%	Evénement déclencheur. Contient les mots clés "insert", "update", "delete"
%TRGTIME%	Moment du déclenchement. Contient les mots clés NULL, "before", "after"
%REFNO%	Numéro d'ordre de référence dans la liste des références
%ERRNO%	Numéro d'erreur pour une erreur standard
%ERRMSG%	Message d'erreur pour une erreur standard
%MSGTAB%	Nom de la table contenant des messages définis par l'utilisateur
%MSGNO%	Nom de la colonne contenant des numéros d'erreur dans un tableau d'erreurs défini par l'utilisateur
%MSGTXT%	Code de la colonne contenant des numéros d'erreur dans un tableau d'erreurs défini par l'utilisateur
%SCRIPT%	Script SQL du trigger ou de la procédure
%TRGBODY%	Corps du trigger (uniquement pour le reverse engineering direct de Oracle)
%TRGDESC%	Description du trigger (uniquement pour le reverse engineering direct de Oracle)
%TRGDEFN%	Définition de trigger
%TRGSCOPE%	Portée du trigger (Mots clés : database, schema, all server)
%TRGSCOPEOWNER%	Propriétaire de la portée du trigger
%TRGSCOPEQUALIFIER%	Propriétaire de la portée du trigger plus tiret

Les variables suivantes sont disponibles pour les procédures :

Variable	Commentaire
%PROC%	Code générer de la procédure (également disponible pour le trigger lorsque ce dernier est mis en oeuvre à l'aide d'une procédure)
%FUNC%	Code généré de la procédure sur la procédure est une fonction (avec une valeur de résultat)
%PROCPRMS%	Liste des paramètres de la procédure

Variables pour les règles

PowerAMC peut utiliser des variables lors de la génération et du reverse-engineering des règles.

Nom de la variable	Commentaire
%RULE%	Code généré pour la règle
%RULENAME%	Nom de la règle
%RULECODE%	Code de la règle
%RULECEXPR%	Expression client de la règle
%RULESEXPR%	Expression serveur de la règle

Variables pour les séquences

PowerAMC peut utiliser des variables lors de la génération et du reverse-engineering des séquences.

Nom de la variable	Commentaire
%SQNC%	Nom de la séquence
%SQNCOWNER%	Nom du propriétaire de la séquence

Variables pour les synonymes

PowerAMC peut utiliser des variables lors de la génération et du reverse-engineering des synonymes.

Variable	Commentaire
%SYNONYM%	Code généré du synonyme
%BASEOBJECT%	Objet de base du synonyme
%BASEOWNER%	Propriétaire de l'objet de base
%BASEQUALIFIER%	Qualifiant de l'objet de base

Variable	Commentaire
%VISIBILITY%	Private (défaut) ou public
%SYNMTYPE%	Synonyme d'alias (DB2 uniquement)
%ISPRIVATE%	True pour un synonyme privé
%ISPUBLIC%	True pour un synonyme public

Variables pour les tablespaces et les storages

PowerAMC peut utiliser des variables lors de la génération et du reverse-engineering des tablespaces et storages.

Nom de la variable	Commentaire
%TABLESPACE%	Code généré pour le tablespace
%STORAGE%	Code généré pour le storage

Variables pour les types de données abstraits

PowerAMC peut utiliser des variables lors de la génération et du reverse-engineering des types de données abstraits et leurs objets enfant.

Les variables suivantes sont disponibles pour les types de données abstraits :

Variable	Commentaire
%ADT%	Code généré du type de données abstrait
%TYPE%	Type du type de données abstrait. Contient des mots clés tels que "array", "list", ...
%SIZE%	Taille du type de données abstrait
%FILE%	Fichier Java du type de données abstrait
%ISARRAY%	TRUE si le type de données abstrait est de type Array
%ISLIST%	TRUE si le type de données abstrait est de type List
%ISSTRUCT%	TRUE si le type de données abstrait est de type Structure
%ISOBJECT%	TRUE si le type de données abstrait est de type Object
%ISJAVAOBJECT%	TRUE si le type de données abstrait est de type JAVA object
%ISJAVA%	TRUE si le type de données abstrait est de type classe JAVA
%ADTDEF%	Contient la définition du type de données abstrait
%ADTBODY%	Corps du type de données abstrait

Variable	Commentaire
%SUPERADT%	Supertype du type de données abstrait
%ADTNOTFINAL%	Type de données abstrait final
%ADTABSTRACT%	Type de données abstrait instanciable
%ADTHEADER%	Corps du type de données abstrait avec ODBC
%ADTTEXT%	Spéc du type de données abstrait avec ODBC
%SUPERQUALIFIER%	Qualifiant du supertype du type de données abstrait
%SUPEROWNER%	Propriétaire du supertype du type de données abstrait
%ADTAUTH%	Autorisation du supertype du type de données abstrait
%ADTJAVANAME%	Nom JAVA du type du type de données abstrait
%ADTJAVADATA%	Données JAVA du type du type de données abstrait
%ADTATTRDEF%	Partie relative aux attributs de la définition du type de données abstrait
%ADTMETHDEF%	Partie relative aux méthodes de la définition du type de données abstrait

Les variables suivantes sont disponibles pour les attributs de type de données abstrait :

Variable	Commentaire
%ADTATTR%	Code généré de l'attribut de type de données abstrait
%ATTRJAVANAME%	Nom Java de l'attribut du type de données abstrait

Les variables suivantes sont disponibles pour les procédures de type de données abstrait :

Variable	Commentaire
%ADTPROC%	Code de procédure
%PROCTYPE%	Type de procédure (constructor, order, map)
%PROCFUNC%	Type de procédure (procedure, function)

Variable	Commentaire
%PROCDEFN%	Corps de procédure (begin... end)
%PROCRETURN%	Type de résultat de procédure
%PARAM%	Paramètres de procédure
%PROCNOTFINAL%	Procédure finale
%PROCSTATIC%	Membre de procédure
%PROCABSTRACT%	Procédure instanciable
%SUPERPROC%	Superprocédure de procédure
%ISCONSTRUCTOR%	True si la procédure est un constructeur
%PROCJAVANAME%	Nom JAVA de procédure
%ISJAVAVAR%	True si la procédure est mise en correspondance avec une variable JAVA statique
%ISSPEC%	True dans les spécifications, non défini dans le corps

Variables pour les join indexes (IQ)

PowerAMC peut utiliser des variables lors de la génération et du reverse-engineering des join indexes IQ.

Nom de la variable	Commentaire
%JIDX%	Code généré pour le join index
%JIDXDEFN%	Corps complet des définitions de join index
%REFRLIST%	Liste des références (pour la connexion directe)
%RFJNLIST%	Liste des références de jointure (pour la connexion directe)
%FACTQUALIFIER%	Qualifiant de la table de fait
%JIDXFACT%	Fait (table de base)
%JIDXCOLN%	Liste de colonnes
%JIDXFROM%	Clause From

Nom de la variable	Commentaire
%JIDXWHERE%	Clause Where

Variables pour ASE & SQL Server

PowerAMC peut utiliser des variables lors de la génération et du reverse-engineering des objets pour ASE et SQL Server.

Nom de la variable	Commentaire
%RULENAME%	Nom de la règle associée au domaine
%DEFAULTNAME%	Nom de l'objet par défaut associé au domaine
%USE_SP_PKEY%	Utilise sp_primary_key pour créer des clés primaires
%USE_SP_FKEY%	Utilise sp_foreign_key pour créer des clés étrangères

Variables pour la synchronisation de base de données

PowerAMC peut utiliser des variables lors de la génération et du reverse-engineering des objets lors de la synchronisation de base de données.

Nom de la variable	Commentaire
%OLDOWNER%	Nom de l'ancien propriétaire de l'objet. Voir aussi OWNER
%NEWOWNER%	Nom du nouveau propriétaire de l'objet. Voir aussi OWNER
%OLDQUALIFIER%	Ancien qualifiant de l'objet. Voir aussi QUALIFIER
%NEWQUALIFIER%	Nouveau qualifiant de l'objet. Voir aussi QUALIFIER
%OLDTABL%	Ancien code de la table
%NEWTABL%	Nouveau code de la table
%OLDCOLN%	Ancien code de la colonne
%NEWCOLN%	Nouveau code de la colonne
%OLDNAME%	Ancien code de la séquence
%NEWNAME%	Nouveau code de la séquence

Variables pours les packages de base de données et leurs objets enfant

PowerAMC peut utiliser des variables lors de la génération et du reverse-engineering des packages de base de données et de leurs objets enfant.

Les variables suivantes sont disponibles pour les packages de base de données :

Variable	Commentaire
%DBPACKAGE%	Code généré du package de base de données
%DBPACKAGECODE%	Code d'initialisation à la fin du package
%DBPACKAGESPEC%	Spécification du package de base de données
%DBPACKAGEBODY%	Corps du package de base de données
%DBPACKAGEINIT%	Code d'initialisation du package de base de données
%DBPACKAGEPRIV%	Autorisation du package de base de données (ancien privilège)
%DBPACKAGEAUTH%	Autorisation du package de base de données
%DBPACKAGEPUBLIC%	True pour un sous-objet public
%DBPACKAGETEXT%	Corps du package de base de données avec ODBC
%DBPACKAGEHEADER%	Spéc du package de base de données avec ODBC

Les variables suivantes sont disponibles pour les procédures de package de base de données :

Variable	Commentaire
%DBPKPROC%	Code de procédure
%DBPKPROCTYPE%	Type de procédure (procedure, function)
%DBPKPROCCODE%	Corps de procédure (begin... end)
%DBPKPROCRETURN%	Type de résultat de procédure
%DBPKPROCPARAM%	Paramètres de procédure

Les variables suivantes sont disponibles pour les variables de package de base de données :

Variable	Commentaire
%DBPFVAR%	Code de la variables
%DBPFVARTYPE%	Type de la variables
%DBPFVARCONST%	Variable de type constant
%DBPFVARVALUE%	Valeur par défaut pour la constante

Les variables suivantes sont disponibles pour les types de package de base de données :

Variable	Commentaire
%DBPKTYPE%	Code du type
%DBPKTYPEVAR%	Liste de variables
%DBPKISSUBTYPE%	True si le type est un sous-type

Les variables suivantes sont disponibles pour les curseurs de package de base de données :

Variable	Commentaire
%DBPKCURSOR%	Code du curseur
%DBPKCURSORRE-TURN%	Type de résultat du curseur
%DBPKCURSORQUERY%	Requête du curseur
%DBPKCURSORPARAM%	Paramètre du curseur

Les variables suivantes sont disponibles pour exceptions de package de base de données :

Variable	Commentaire
%DBPKEXEC%	Code de l'exception

Les variables suivantes sont disponibles pour les paramètres de package de base de données :

Variable	Commentaire
%DBPKPARM%	Code du paramètre
%DBPKPARMTYPE%	Type du paramètre
%DBPKPARMDTTP%	Type de données du paramètre
%DBPKPARMDEFAULT%	Valeur par défaut du paramètre

Les variables suivantes sont disponibles pour les pragma de package de base de données :

Variable	Commentaire
%DBPKPRAGMA%	Directive de pragma
%DBPKPRAGMAOBJ%	Directive de pragma sur objet

Variable	Commentaire
%DBPKPRAGMAPARAM%	Paramètre de directive de pragma

Variables pour la sécurité dans la base de données

PowerAMC peut utiliser des variables lors de la génération et du reverse-engineering des objets de sécurité de base de données.

Nom de la variable	Commentaire
%PRIVLIST%	Liste des privilèges pour une commande grant
%REVPRIVLIST%	Liste des privilèges pour une commande revoke
%PERMLIST%	Liste des permissions pour une commande grant
%REVPERMLIST%	Liste des permissions pour une commande revoke
%COLNPERMISSION%	Permissions sur une liste de colonnes particulière
%BITMAPCOLN%	Bitmap de colonnes spécifiques avec des permissions
%USER%	Nom de l'utilisateur
%GROUP%	Nom du groupe
%ROLE%	Nom du rôle
%GRANTEE%	Nom générique utilisé pour concevoir un utilisateur, un groupe ou un rôle
%PASSWORD%	Mot de passe pour un utilisateur, un groupe ou un rôle
%OBJECT%	Objets de base de données (table, vue, colonne, etc.)
%PERMISSION%	Commande SQL grant/revoke pour un objet de base de données
%PRIVILEGE%	Commande SQL grant/revoke pour un ID (utilisateur, groupe ou rôle)
%GRANTOPTION%	Option pour grant : with grant option / with admin option
%REVOKEOPTION%	Option pour revoke : with cascade
%GRANTOR%	Utilisateur qui accorde la permission
%MEMBER%	Membre d'un groupe ou membre d'un rôle
%GROUPS%	Liste de groupes séparés par le délimiteur
%MEMBERS%	Liste de membres (utilisateurs ou rôles) d'un groupe ou d'un rôle séparés par le délimiteur
%ROLES%	Liste des rôles parent d'un utilisateur ou d'un rôle

Nom de la variable	Commentaire
%SCHEMADEFN%	Définition de schéma

Variables pour les défauts

PowerAMC peut utiliser des variables lors de la génération et du reverse-engineering des défauts.

Variable	Commentaire
%BOUND_OBJECT%	Objet lié

Variables pour les services Web

PowerAMC peut utiliser des variables lors de la génération et du reverse-engineering des séquences.

Les variables suivantes sont disponibles pour les services Web :

Variable	Commentaire
%WEBSERVICENAME%	Seul code généré du service Web
%WEBSERVICE%	Code généré du chemin local de service Web
%WEBSERVICETYPE%	Type de service Web
%WEBSERVICESQL%	Instruction SQL
%WEBSERVICELOCAL-PATH%	Chemin local

Les variables suivantes sont disponibles pour les opérations de service Web :

Variable	Commentaire
%WEBOPERATIONNA-ME%	Seul code généré de l'opération Web
%WEBOPERATION%	Code généré de l'opération, service et chemin local
%WEBOPERATIONTYPE%	Type d'opération Web
%WEBOPERATIONSQL%	Instruction SQL
%WEBOPERATIONPA-RAM%	Liste des paramètres d'opération Web

Les variables suivantes sont disponibles pour la sécurité de service Web :

Variable	Commentaire
%WEBUSER%	Utilisateur de connexion requis pour le service Web
%WEBCNCTSECURED%	Connexion sécurisée
%WEBAUTHREQUIRED%	Authorisation requise

Les variables suivantes sont disponibles pour les paramètres de service Web:

Variable	Commentaire
%WEBPARAM%	Liste des paramètres Web
%WEBPARAMNAME%	Nom du paramètre Web
%WEBPARAMTYPE%	Type du paramètre Web
%WEBPARAMDTTP%	Type de données du paramètre Web
%WEBPARAMDEFAULT%	Valeur par défaut du paramètre Web

Variables pour les dimensions

PowerAMC peut utiliser des variables lors de la génération et du reverse-engineering des dimensions.

Variable	Commentaire
%DIMENSION%	Code généré de la dimension
%DIMNDEF%	Définition de la dimension
%DIMNATTR%	Attribut (niveau) de la dimension
%DIMNOWNERTABL%	Propriétaire de la table de niveau
%DIMNTABL%	Table de niveau
%DIMNCOLN%	Colonne de niveau
%DIMNCOLNLIST%	Liste des colonnes de niveau
%DIMNHIER%	Hiérarchie de dimensions
%DIMNKEY%	Liste des colonnes de clé enfant
%DIMNKEYLIST%	Liste des colonnes de clé enfant
%DIMNLEVELLIST%	Liste des niveaux de la hiérarchie
%DIMNATTRHIER%	Attribut de la hiérarchie

Variable	Commentaire
%DIMNATTRHIERFIRST%	Premier attribut de la hiérarchie
%DIMNATTRHIERLIST%	Liste des attributs de la hiérarchie
%DIMNPARENTLEVEL%	Niveau parent de la hiérarchie
%DIMNDEPATTR%	Attribut de dimension avec dépendant
%DIMNDEPCOLN%	Colonne dépendante
%DIMNDEPCOLNLIST%	Liste des colonnes dépendantes

Variables pour les objets étendus

PowerAMC peut utiliser des variables lors de la génération et du reverse-engineering des objets étendus.

Variable	Commentaire
%EXTENDEDOBJECT%	Code généré de l'objet étendu
%EXTENDEDSUBOBJECT%	Code généré du sous-objet étendu
%EXTSUBOBJTPARENT%	Code généré du parent du sous-objet étendu
%EXTSUBOBJTPARENTOWNER%	Code généré du propriétaire du sous-objet étendu
%EXTSUBOBJTPARENTQUALIFIER%	Qualifiant de l'objet parent (préfixe de base de données et préfixe de propriétaire)
%EXTOBJECTDEFN%	Corps complet de la définition de l'objet étendu. Contient les définitions de la collection étendue répertoriée dans l'élément de SGBD DefinitionContent.

Variables pour les métadonnées

PowerAMC peut utiliser des variables lors de la génération et du reverse-engineering des métadonnées.

Nom de la variable	Commentaire
%@CLSSNAME%	Nom localisé de la classe de l'objet. Ex : Table, View, Column, Index
%@CLSSCODE%	Code de la classe de l'objet. Ex : TABL, VIEW, COLN, INDX

Variables pour le reverse engineering

PowerAMC peut utiliser des variables lors du reverse-engineering d'objets.

Nom de la variable	Commentaire
%R%	Défini à TRUE lors du reverse engineering
%S%	Permet de sauter un mot. La chaîne est analysée pour le reverse engineering, mais pas générée
%D%	Permet de sauter une valeur numérique. La valeur numérique est analysée pour le reverse engineering, mais pas générée
%A%	Permet de sauter tout le texte. Le texte est analysé pour le reverse engineering, mais pas généré
%ISODBCUSER%	True si l'utilisateur courant est l'utilisateur connecté
%CATALOG%	Nom du catalogue à utiliser dans des requêtes de reverse engineering direct
%SCHEMA%	Variable qui représente un login utilisateur et l'objet appartenant à cet utilisateur dans la base de données. Vous devez utiliser cette variable pour les requêtes sur les objets répertoriés dans les boîtes de dialogue de reverse engineering direct, car leur propriétaire n'est pas encore défini. Une fois le propriétaire d'un objet défini, vous pouvez utiliser SCHEMA ou OWNER
%SIZE%	Taille du type de données de la colonne ou du domaine. Utilisé pour le reverse engineering direct, lorsque la longueur n'est pas définie dans les tables système
%VALUE%	Une valeur de la liste des valeurs dans une colonne ou dans un domaine
%PERMISSION%	Permet de procéder au reverse engineering de permissions définies sur des objets de base de données
%PRIVILEGE%	Permet de procéder au reverse engineering de privilèges définis sur un utilisateur, un groupe ou un rôle

Variables pour la génération de bases de données, de triggers et de procédures

PowerAMC peut utiliser des variables lors de la génération des bases de données, des triggers et des procédures.

Nom de la variable	Commentaire
%DATE%	Date et heure de génération
%USER%	Nom de login de l'utilisateur exécutant la génération

Nom de la variable	Commentaire
%PATHSCRIPT%	Emplacement pour la génération du fichier script
%NAMESCRIPT%	Nom du fichier script dans lequel les commandes SQL doivent être écrites
%STARTCMD%	Description des modalités d'exécution du script généré
%ISUPPER%	TRUE si l'option de génération Majuscules est sélectionnée
%ISLOWER%	TRUE si l'option de génération Minuscules est sélectionnée
%DBMSNAME%	Nom du SGBD associé au modèle généré
%DATABASE%	Code de la base de données associée au modèle généré
%DATASOURCE%	Nom de la source de données associée au script généré
%USE_SP_PKEY%	Utilise la clé primaire de la procédure stockée pour créer des clés primaires (spécifique à SQL Server)
%USE_SP_FKEY%	Utilise la clé étrangère de procédure stockée pour créer des clés primaires (spécifique à SQL Server)

Macro .AKCOLN

Répète une instruction pour chaque colonne de clé alternative d'une table

Syntaxe

.AKCOLN("*instruction*","*préfixe*","*suffixe*","*dernier_suffixe*", "condition")

Argument	Description
instruction	Instruction à répéter pour chaque colonne
préfixe	Préfixe pour chaque nouvelle ligne
suffixe	Suffixe pour chaque nouvelle ligne
dernier suffixe	Suffixe de la dernière ligne
condition	Code de clé alternative (si l'argument de condition est laissé à blanc, la macro renvoie une instruction pour chaque clé alternative dans la table)

Exemple

Dans un trigger pour la table ECRIT, la macro suivante :

```
message .AKCOLN("'%COLUMN% is an alternate key column'","", "", "",
"AKEY1")
```

Génère le script de trigger qui suit :

```
message 'TA_ORDER is an alternate key column',
```

Remarque : La macro AKCOLN n'accepte que la variable %COLUMN% pour les colonnes.

Macro .ALLCOL

Répète une instruction pour chaque colonne d'une table

Syntaxe

.ALLCOL("*instruction***","***préfixe***","***suffixe***","***dernier_suffixe***")**

Argument	Description
instruction	Instruction à répéter pour chaque colonne
préfixe	Préfixe pour chaque nouvelle ligne
suffixe	Suffixe pour chaque nouvelle ligne
dernier suffixe	Suffixe de la dernière ligne

Exemple

Dans un trigger pour la table AUTEUR, la macro suivante :

```
.ALLCOL("%COLUMN% %COLTYPE%","","","","")
```

Génère le script de trigger qui suit :

```
AU_ID char(12),
AU_LNAME varchar(40),
AU_FNAME varchar(40),
AU_BIOGRAPH long varchar,
AU_ADVANCE numeric(8,2),
AU_ADDRESS varchar(80),
CITY varchar(20),
STATE char(2),
POSTALCODE char(5),
AU_PHONE char(12);
```

Macro .DEFINE

Définit une variable et initialise sa valeur.

Syntaxe

.DEFINE "*variable***" "***valeur***"**

Argument	Description
variable	Nom de la variable (sans signe %)
valeur	Valeur de la variable (peut inclure une autre variable encadrée de signes %)

Exemple

Dans un trigger pour la table AUTEUR, la macro suivante :

```
.DEFINE "TRIGGER" "T_%TABLE%"
message 'Error: Trigger(%TRIGGER%) of table %TABLE%'
```

Génère le script de trigger qui suit :

```
message 'Error: Trigger(T_AUTHOR) of table AUTHOR';
```

Macro .DEFINEIF

Définit une variable et initialise sa valeur si le résultat du test n'est pas NULL.

Syntaxe

.DEFINEIF "*valeur_test*""*variable*" "*valeur*"

Argument	Description
valeur_test	Valeur à tester
variable	Nom de la variable (sans signe %)
valeur	Valeur de la variable (peut inclure une autre variable encadrée de signes %)

Exemple

Par exemple, pour définir une variable spécifiant un type de données par défaut :

```
%DEFAULT%
.DEFINEIF "%DEFAULT%" "_DEFLT"" "%DEFAULT%"
Add %COLUMN% %DATATYPE% %_DEFLT%
```

Macro .ERROR

Gère les erreurs.

Syntaxe

.ERROR (*noerr* "*msgerr*"**)**

Argument	Description
noerr	Numéro de l'erreur
msgerr	Message d'erreur

Exemple

```
.ERROR(-20001, "Parent does not exist, cannot insert child")
```

Macro .FKCOLN

Répète une instruction pour chaque colonne de clé étrangère d'une table.

Syntaxe

```
.FKCOLN("instruction","préfixe","suffixe","dernier_suffixe")
```

Argument	Description
instruction	Instruction à répéter pour chaque colonne
préfixe	Préfixe pour chaque nouvelle ligne
suffixe	Suffixe pour chaque nouvelle ligne
dernier suffixe	Suffixe de la dernière ligne

Exemple

Dans un trigger pour la table ECRIT, la macro suivante :

```
message .FKCOLN("'%COLUMN% is a foreign key column'","",",",";")
```

Génère le script de trigger qui suit :

```
message 'AU_ID is a foreign key column,
TITLE_ISBN is a foreign key column;'
```

Remarque : La macro FKCOLN n'accepte que la variable %COLUMN% pour les colonnes.

Macro .FOREACH_CHILD

Répète une instruction pour chaque référence père-à-enfant contenue dans la table courante et qui remplit une condition.

Syntaxe

```
.FOREACH_CHILD ("condition")

"instruction"

.ENDFOR
```

Argument	Description
condition	Condition relative à la référence (voir ci-dessous)
instruction	Instruction à répéter

Condition	Sélectionne
UPDATE RESTRICT	Restrict pour une modification
UPDATE CASCADE	Cascade pour une modification
UPDATE SETNULL	Set null pour une modification
UPDATE SETDEFAULT	Set default pour une modification
DELETE RESTRICT	Restrict pour une suppression
DELETE CASCADE	Cascade pour une suppression
DELETE SETNULL	Set null pour une suppression
DELETE SETDEFAULT	Set default pour une suppression

Exemple

Dans un trigger pour la table TITRE, la macro suivante :

```
.FOREACH_CHILD("DELETE RESTRICT")
--  Cannot delete parent "%PARENT%" if children still exist in
"%CHILD%"
.ENDFOR
```

Génère le script de trigger qui suit :

```
--  Cannot delete parent "TITLE" if children still exist in
"ROYSCHED"
--  Cannot delete parent "TITLE" if children still exist in "SALE"
--  Cannot delete parent "TITLE" if children still exist in
"TITLEAUTHOR"
```

Macro .FOREACH_COLUMN

Répète une instruction pour chaque colonne de la table courante qui remplit une condition.

Syntaxe

```
.FOREACH_COLUMN ("condition")

"instruction"

.ENDFOR
```

Argument	Description
condition	Condition relative aux colonnes (voir ci-dessous)
instruction	Instruction à répéter

Condition	Sélectionne
vide	Toutes les colonnes
PKCOLN	Colonnes de clé primaire
FKCOLN	Colonnes de clé étrangère
AKCOLN	Colonnes de clé alternative
NMFCOL	Colonnes non-modifiables (pour lesquelles le paramètre de contrôle Non modifiable est sélectionné)
INCOLN	Colonnes de trigger (colonnes de clé primaire, colonnes de clé étrangère et colonnes non modifiables)

Exemple

Dans un trigger pour la table TITRE, la macro suivante :

```
.FOREACH_COLUMN("NMFCOL")
-- "%COLUMN%" cannot be modified
.ENDFOR
```

Génère le script de trigger qui suit :

```
-- "TITLE_ISBN" cannot be modified
-- "PUB_ID" cannot be modified
```

Macro .FOREACH_PARENT

Répète une instruction pour chaque référence enfant-à-père contenue dans la table courante et qui remplit une condition.

Syntaxe

```
.FOREACH_PARENT ("condition")
"instruction"
.ENDFOR
```

Argument	Description
condition	Condition de la référence (voir ci-dessous)
instruction	Instruction à répéter

Condition	Sélectionne les références définies avec ...
vide	Toutes les références
FKNULL	Clés étrangères non-obligatoires

Condition	Sélectionne les références définies avec ...
FKNOTNULL	Clés étrangères obligatoires
FKCANTCHG	Clés étrangères non-modifiables

Exemple

Dans un trigger pour la table VENTE, la macro suivante :

```
.FOREACH_PARENT("FKCANTCHG")
--  Cannot modify parent code of "%PARENT%" in child "%CHILD%"
.ENDFOR
```

Génère le script de trigger qui suit :

```
--  Cannot modify parent code of "STORE" in child "SALE"
--  Cannot modify parent code of "TITLE" in child "SALE"
```

Macro .INCOLN

Répète une instruction pour chaque colonne de clé primaire, colonne de clé étrangère, colonne de clé alternative ou colonne non-modifiable contenue dans une table.

Syntaxe

```
.INCOLN("instruction","préfixe","suffixe","dernier_suffixe")
```

Argument	Description
instruction	Instruction à répéter pour chaque colonne
préfixe	Préfixe pour chaque nouvelle ligne
suffixe	Suffixe pour chaque nouvelle ligne
dernier suffixe	Suffixe pour la dernière ligne

Exemple

Dans un trigger pour la table TITRE, la macro suivante :

```
.INCOLN("%COLUMN% %COLTYPE%","",",",";")
```

Génère le script de trigger qui suit :

```
TITLE_ISBN char(12),
PUB_ID char(12);
```

Macro .JOIN

Répète une instruction pour un couple de colonnes dans une jointure.

Syntaxe

```
.JOIN("instruction","préfixe","suffixe","dernier_suffixe")
```

Argument	Description
instruction	Instruction à répéter pour chaque colonne
préfixe	Préfixe pour chaque nouvelle ligne
suffixe	Suffixe pour chaque nouvelle ligne
dernier suffixe	Suffixe pour la dernière ligne

Exemple

Dans un trigger pour la table TITRE, la macro suivante :

```
.FOREACH_PARENT()
where .JOIN("%PK%=%FK%", " and", "", ";")
message 'Reference %REFR% links table %PARENT% to %CHILD%'
 .ENDFOR
```

Génère le script de trigger qui suit :

```
message 'Reference TITLE_PUB links table PUBLISHER to TITLE
```

Remarque : La macro JOIN n'accepte que les variables %PK%, %AK% et %FK% pour les colonnes.

Macro .NMFCOL

Répète une instruction pour chaque colonne non-modifiable d'une table. Les colonnes non-modifiables sont celles pour lesquelles le paramètre de contrôle Non modifiable est sélectionné.

Syntaxe

```
.NMFCOL("instruction","préfixe","suffixe","dernier_suffixe")
```

Argument	Description
instruction	Instruction à répéter pour chaque colonne
préfixe	Préfixe pour chaque nouvelle ligne
suffixe	Suffixe pour chaque nouvelle ligne
dernier suffixe	Suffixe pour la dernière ligne

Exemple
Dans un trigger pour la table TITRE, la macro suivante :

```
.NMFCOL("%COLUMN% %COLTYPE%","","","";")
```

Génère le script de trigger qui suit :

```
TITLE_ISBN char(12),
PUB_ID char(12);
```

Macro .PKCOLN

Répète une instruction pour chaque colonne de clé primaire d'une table.

Syntaxe

```
.PKCOLN("instruction","préfixe","suffixe","dernier_suffixe")
```

Argument	Description
instruction	Instruction à répéter pour chaque colonne
préfixe	Préfixe pour chaque nouvelle ligne
suffixe	Suffixe pour chaque nouvelle ligne
dernier suffixe	Suffixe pour la dernière ligne

Exemple
Dans un trigger pour la table ECRIT, la macro suivante :

```
message .PKCOLN("'%COLUMN% is a primary key column'","","","";")
```

Génère le script de trigger qui suit :

```
message 'AU_ID is a primary key column',
    'TITLE_ISBN is a primary key column';
```

Remarque : La macro FKCOLN n'accepte que la variable %COLUMN% pour les colonnes.

Macros .CLIENTEXPRESSION et .SERVEREXPRESSION

Utilise l'expression client et/ou serveur d'une règle de gestion dans le script du modèle de trigger, de l'élément de modèle de trigger, du trigger et de la procédure.

Syntaxe

```
.CLIENTEXPRESSION(code de la règle de gestion)
```

```
.SERVEREXPRESSION(code de la règle de gestion)
```

Exemple

La règle de gestion CONTROLE_DATE_ACTIVITE comporte l'expression serveur suivante :

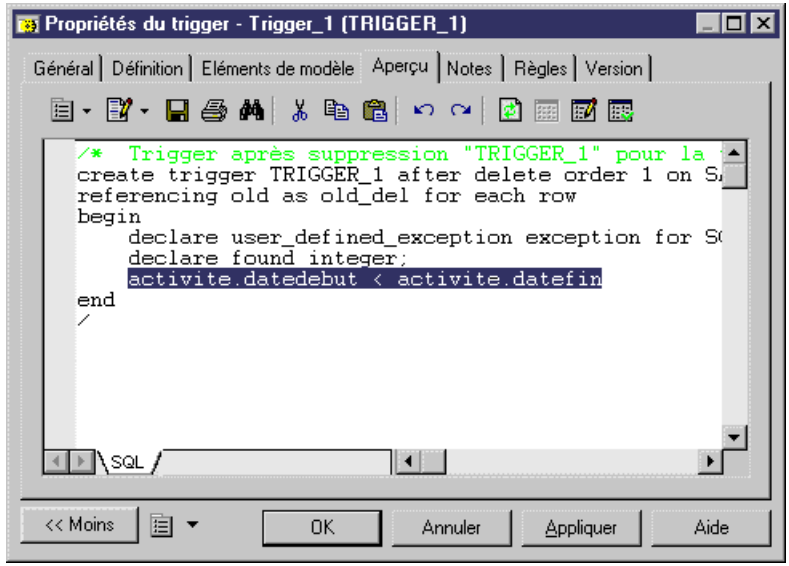
```
activity.begindate < activity.enddate
```

Dans un trigger basé sur le modèle de trigger AfterDeleteTrigger, vous saisissez la macro suivante dans l'onglet Définition du trigger :

```
.SERVEREXPRESSION(ACTIVITY_DATE_CONTROL)
```

Vous générez ainsi le script de trigger suivant :

```
activity.begindate < activity.enddate
end
```



Macro .SQLXML

Représente une requête SQL/XML dans la définition d'un trigger, d'une procédure ou d'une fonction.

Vous pouvez utiliser l'un des outils suivants :

- L'outil *Insérer une macro SQL/XML* affiche une boîte de sélection dans laquelle vous choisissez un élément global dans un modèle XML. Ce modèle XML doit être ouvert dans l'espace de travail, être mis en correspondance avec un MPD, et avoir le fichier d'extension SQL/XML attaché. Cliquez sur OK dans la boîte de dialogue. La macro SQLXML apparaît alors dans le code de la définition, avec le code du modèle XML (facultatif) ainsi que le code de l'élément globaL.

- L'outil *Macros*, qui permet de sélectionner *.SQLXML( )* dans la liste. La macro SQLXML apparaît vide dans le code de la définition. Vous devez spécifier entre les parenthèses le code d'un modèle XML (facultatif), suivi des signes :: puis du code d'un élément global. Le modèle XML dans lequel vous choisissez un élément global doit être ouvert dans l'espace de travail, mis en correspondance avec un modèle XML et avoir le fichier d'extension SQL/XML attaché.

A l'issue de la génération, la macro SQLXML est remplacée par la requête SQL/XML de l'élément global.

Syntaxe

```
.SQLXML(code d'un modèle XML::code d'un élément global)
```

Remarque : le code d'un modèle XML est facultatif.

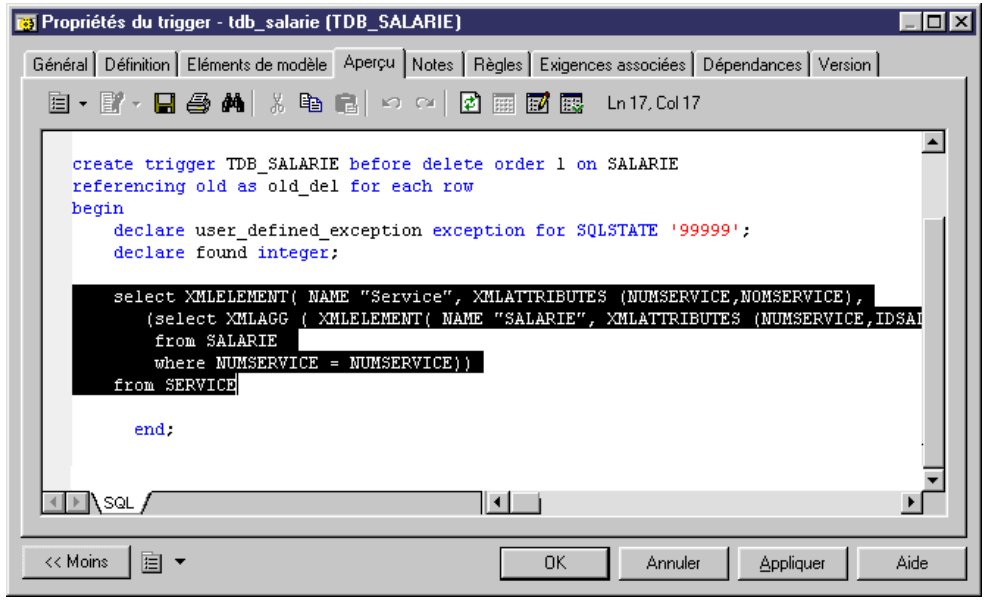
Exemple

Dans un trigger pour la table SALARIE, la macro suivante :

```
.SQLXML(PersonnelSociété::SERVICE)
```

Génère le script de trigger suivant :

```
select XMLELEMENT( NAME "Service", XMLATTRIBUTES
(NUMSERVICE,NOMSERVICE),
   (select XMLAGG ( XMLELEMENT( NAME "SALARIE", XMLATTRIBUTES
(NUMSERVICE,IDSALARIE,PRENOM,NOM)) )
    from SALARIE
    where NUMSERVICE = NUMSERVICE))
from SERVICE
```


Propriétés du trigger - tdb_salarie (TDB_SALARIE)
Général
Définition
Eléments de modèle
Aperçu
Notes
Règles
Exigences associées
Dépendances
Version
Ln 17, Col 17
create trigger TDB_SALARIE before delete order 1 on SALARIE
referencing old as old_del for each row
begin
declare user_defined_exception exception for SQLSTATE '99999';
declare found integer;
select XMLELEMENT(NAME "Service", XMLATTRIBUTES (NUMSERVICE,NOMSERVICE),
(select XMLAGG (XMLELEMENT(NAME "SALARIE", XMLATTRIBUTES (NUMSERVICE,IDSAI
from SALARIE
where NUMSERVICE = NUMSERVICE))
from SERVICE
end;
SQL
<< Moins
OK
Annuler
Appliquer
Aide

Chapitre 5

Personnalisation de la génération à l'aide du langage de génération par template

Le langage de génération par template (GTL, Generation Template Language) PowerAMC est un langage de génération de texte basé sur des templates qui est utilisé pour générer du texte pour les métaclasses définies dans le métamodèle PowerAMC, ainsi que sur toutes les extensions définies dans le profil du modèle.

Chaque template est associé à une métaclasse donnée (par exemple un attribut d'entité de MCD, une table de MPD ou une opération de MOO). Vous pouvez définir autant de templates que vous le souhaitez pour chaque métaclasse, et ceux-ci seront disponibles pour tous les objets (instances) de la métaclasse. Par exemple, pour examiner le jeu de templates utilisé afin de générer du code pour les opérations dans un MOO pour Java, ouvrez le langage Java dans l'Editeur de ressources et développer la catégorie Profile\Operation\Templates.

Lorsque vous générez un modèle, PowerAMC détermine pour quelles métaclasses des fichiers doivent être générés, et crée un fichier pour chaque instance de la métaclasse, en appliquant les templates appropriés et en résolvant les variables.

Le langage de génération par template est un langage orienté objet, et il prend en charge l'héritage et le polymorphisme afin de permettre que le code soit réutilisable et modifiable. Les macros fournissent les structures de programmation génériques permettant de tester des variables et de procéder à l'itération dans les collections, etc.

Un template de GTL peut contenir du texte, des macros et des variables, et il peut référencer :

- des attributs de métamodèle, tels que le nom d'une classe ou le type de données d'un attribut
- des collections, telles que la liste des attributs d'une classe ou des colonnes d'une table
- d'autres éléments du modèle, tels que les variables d'environnement

Les templates de GTL peuvent être soit :

- Des templates simples - qui peuvent contenir du texte, des variables et des blocs conditionnels, mais ne peuvent pas contenir de macros. Par exemple :
  ```
  %Visibility% %DataType% %Code%
  ```
  Lorsque ce template est évalué, les trois variables Visibility, DataType, et Code sont résolues aux valeurs de ces propriétés pour l'objet.
- Des templates complexes - qui peuvent contenir n'importe quel élément provenant d'un template simple, ainsi que des macros. Par exemple :

```
.if (%isInner% == false) and ((%Visibility% == +)
                                or (%Visibility% == *))
    [%sourceHeader%\n\n]\
    [%definition%\n\n]
    .foreach_item(ChildDependencies)
        [%isSameFile%?%InfluentObject.definition%\n\n]
    .next
    [%sourceFooter%\n]
.endif
```

Ce template commence par une macro .if qui teste les valeurs des propriétés isInner et Visibility. Plusieurs variables sont encadrées par des crochets, qui font en sorte que le texte qu'ils encadrent (dans le cas présent, les caractères de passage à la ligne) ne sera pas généré si la variable est évaluée à void. La macro .foreach_item boucle sur tous les membres de la collection ChildDependencies.

Création d'un template et d'un fichier généré

Les templates de GTL sont souvent utilisés pour générer des fichiers. Si votre template doit être utilisés pour la génération, il doit être référencé dans un fichier généré.

1. Dans l'Editeur de ressources, pointez sur une métaclasse dans la catégorie Profile, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau** > **Template** dans le menu contextuel.

   Il est conseillé de nommer vos templates en utilisant la présentation headless camelCase, (commençant par une minuscule), ce afin d'éviter les risques de conflit avec les noms de propriété et de collections qui, par convention, utilisent la présentation full CamelCase.
2. Pointez sur la métaclasse, cliquez à nouveau le bouton droit de la souris, puis sélectionnez **Nouveau** > **Fichier généré** dans le menu contextuel.
3. Insérez le nom du dans le fichier généré entre signes pourcent. Par exemple :

   ```
   %monTemplate%
   ```

Accès aux propriétés des objets

Les propriétés d'objet sont traitées comme des variables, et placées entre signes pourcent, comme suit :

```
%variable%
```

Template d'exemple :

```
Ce fichier est généré pour %Name% sous forme de %Shape% %Color%.
```

Résultat :

```
Ce fichier est généré pour MonObjet sous forme de Triangle Rouge.
```

Pour plus d'informations, voir *Membres d'objet* à la page 290.

Définition du format du résultat

Pour contrôler le format du résultat, insérez les options de format entre les signes pourcent, avant la variable, comme suit :

```
%.format:variable%
```

Template d'exemple :

Le template suivant modifie le format de la variable Name pour le mettre en majuscules et le placer entre guillemets.

```
Ce fichier est généré pour %.UQ:Name% sous forme de %.L:Shape%
%.L:Color%.
```

Résultat :

```
Ce fichier est généré pour "MONGADGET" sous forme de triangle rouge.
```

Pour plus d'informations, voir *Options de formatage des variables* à la page 293.

Utilisation des blocs conditionnels

Si vous avez du texte qui ne doit apparaître que si une variable est résolue en valeur non -null, vous devez le regrouper entre crochets.

Template d'exemple :

```
[This line is generated if "Exist" is not null: %Exist%]
This line is generated even if "Exist" is null: %Exist%
```

Résultat (si Exist est null) :

```
This line is generated even if "Exist" is null:
```

Résultat (si Exist n'est pas null) :

```
This line is generated if "Exist" is not null: Y
This line is generated even if "Exist" is null: Y
```

Pour plus d'informations, voir *Blocs conditionnels* à la page 291.

Accès aux collections de sous-objets

Les tables ont plusieurs colonnes, les classes ont plusieurs attributs et opérations. Pour procéder à l'itération de telles collections d'objets associés, utilisez une macro, telle que .foreach_item.

Exemple :

```
%Name% contient les widgets suivants :
.foreach_item(Widgets)
    \n\t%Name% (%Shape% %Color%)
.next
```

Résultat :

```
MyObject contient les widgets suivants :
    Widget1 (triangle rouge)
    Widget1 (carré jaune)
    Widget1 (cercle vert)
```

Pour plus d'informations, voir *Membres de collection* à la page 291.

Accès aux variables globales

Vous pouvez insérer des informations telles que votre nom d'utilisateur et la date courante en utilisant les variables globales.

Template d'exemple :

```
Ce fichier a été généré par %CurrentUser% le %CurrentDate%.
```

Résultat :

```
Ce fichier a été généré par jsmith le jeudi 23 décembre 2010
14:37:45.
```

Pour plus d'informations, voir *Variables globales* à la page 291.

Guide de référence des variables du langage de génération par template

Les variables sont des valeurs qualifiées encadrées de signes % et éventuellement précédées d'option de format. Au moment de l'évaluation, elles sont remplacées par leur valeur correspondance dans la portée de conversion active.

Une variable peut avoir le type suivant :

- Attribut d'un objet
- Membre d'une collection ou d'une collection étendue
- Un template
- Une variable d'environnement

Par exemple, la variable %Name% d'une interface peut être directement évaluée par une macro et remplacée par le nom de l'interface dans le fichier généré.

Remarque : Attention, la casse des caractères est prise en compte pour les noms de variable. La première lettre d'un nom de variable doit être une majuscule, comme dans %Code%.

Syntaxe des variables

Les variables suivantes sont représentées avec leur syntaxe possible :

variable-block:

```
%[.formatting-options:]variable%
```

variable

```
[outer-scope.][variable-object.][object-scope.]object-member
[outer-scope.][variable-object.][collection-scope.]collection-
member
[outer-scope.]local-variable
[outer-scope.]global-variable
```

object-member:

```
volatile-attribute
property
[target-code::]extended-attribute
[target-code::][metaclass-name::]template-name[(parameter-list)]
[*]+local-value[(parameter-list)]
```

object-member-object =

```
objecttype-property
oid-valued-object-member
this
```

collection-member

```
First
IsEmpty
Count
```

collection-member-object =

```
First
```

local-variable

```
local-object
[*]local-value
```

global-variable

```
global-object
global-value
$environment variable
```

variable-object

```
global-object
local-object
```

outer-scope

```
[outer-scope.]Outer
```

object-scope

```
[object-scope.]object-member-object
collection-scope.collection-member-object
```

collection-scope

```
[object-scope.]collection
[object-scope.]semi-colon-terminated-oid-valued object-member
```

Pour plus d'informations sur les collections étendues, voir *Collections et compositions étendues (Profile)* à la page 72.

Membres d'objet

Un membre d'objet peut être un attribut volatile, une propriété standard, un template ou un attribut étendu. Il peut y avoir trois types de propriété standard : boolean, string ou object. La valeur d'une propriété standard peut être :

- 'true' ou 'false' s'il s'agit d'une propriété de type boolean
- OID d'objet 'null' ou 'notnull' s'il s'agit d'une propriété de type object

La valeur d'un template est le résultat de sa conversion (remarquez qu'un template peut être défini par rapport à lui-même, c'est-à-dire de façon récursive).

La valeur d'un attribut étendu peut également être un template, auquel cas elle est convertie. Ceci permet de définir les templates sur une base objet (instance) plutôt que sur une base métaclasse.

Pour éviter les conflits de nom lorsque l'évaluation d'un template s'étend sur plusieurs cibles, il est possible de préfixer à la fois les attributs étendus et les templates par le code de leur cible parent. Par exemple : %Java::strictfp% ou %C++::definition%

Les noms de template peuvent également être préfixés par le nom de leur métaclasse parent. Ceci vous permet d'invoquer un template redéfini, en contournant de fait le mécanisme de résolution de template dynamique. Par exemple : %Classifier::definition%

Vous avez également la possibilité de spécifier une liste de paramètres. Les valeurs de paramètre ne doivent pas contenir de caractères % et être séparées par des virgules. Les paramètres sont transmis sous forme de variables locales @1, @2, @3... définies dans la portée de la conversion du template.

Si le template MyTemplate est défini de la façon suivante :

```
Parameter1 = %@1%
Parameter2 = %@2%
```

L'évaluation de %MyTemplate(MyParam1, MyParam2)% va produire :

```
Parameter1 = MyParam1
Parameter2 = MyParam2
```

Membres de collection

Chaque objet peut avoir une ou plusieurs collections qui contiennent les objets avec lesquels ils interagissent. Par exemple, une table a des collections de colonnes, d'index, de règles de gestion, etc.

Les collections sont représentées dans le métamodèle PowerAMC (voir *Chapitre 1, Utilisation des fichiers de ressources PowerAMC* à la page 1) par le biais d'associations entre les objets, avec des rôles nommés d'après le nom des collections.

Les membres de collection disponibles sont les suivants :

Nom	Type	Description
First	Object	Renvoie le premier élément de la collection
IsEmpty	Boolean	Permet de tester si une collection est vide. True si la collection est vide, false dans le cas contraire
Count	Integer	Nombre d'éléments de la collection

Remarque : Count est tout particulièrement utile pour définir des critères basés sur la taille de la collection, par exemple (Attributes.Count>=10).

Blocs conditionnels

Les blocs conditionnels peuvent être utilisés pour spécifier différents templates en fonction de la valeur d'une variable. Il existe deux types de bloc conditionnels différents :

Le premier type est similaire à C et aux expressions ternaires Java. Si la valeur de la variable est false, null, ou la chaîne null, le second template, s'il est spécifié, est évalué. Dans le cas contraire, c'est le premier template qui est évalué :

[*variable* ? *template-simple* [: *template-simple*]]

Le second type est converti si et uniquement si la valeur de la variable n'est pas la chaîne null :

[*texte variable texte*]

Exemple : déclaration d'attribut en Java :

```
%Visibility% %DataType% %Code% [= %InitialValue%]
```

Variables globales

Les variables globales sont disponibles quelle que soit la portée courante. Certaines variables spécifiques au langage de génération par template sont répertoriées dans le tableau suivant :

Nom	Type	Description
ActiveModel	Object	Modèle actif

Nom	Type	Description
GenOptions	struct	Permet d'accéder aux options de génération définies par l'utilisateur
PreviewMode	boolean	True si en mode Aperçu, false en mode de génération de fichier
CurrentDate	String	Date et heure système courante, mises en forme en fonction des paramètres régionaux en vigueur
CurrentUser	String	Login utilisateur courant
NewUUID	String	Renvoie un nouvel UUID

Variables locales

Vous pouvez définir des variables locales à l'aide des macros .set_object et .set_value

Pour plus d'informations, voir *Macrod .set_object et .set_value* à la page 325. Les variables locales ne sont visibles que dans la portée dans laquelle elles sont définies ainsi qu'à l'intérieur de leurs portées internes.

Les attributs volatils peuvent être définis via les macros .set_object et .set_value.

Si la portée est une portée d'objet :
Un attribut volatile est défini. Cet attribut sera disponible sur l'objet correspondant et ce, quelle que soit la hiérarchie de la portée. Les attributs volatiles masquent les attributs standard. Une fois définis, ils restent disponibles jusqu'à la fin du processus de génération courant.

Le mot clé "this" renvoie une portée d'objet et permet de définir des attributs volatiles sur l'objet qui est actif dans la portée courante.

Si la portée est une portée de template :
Une variable locale standard est définie.

Exemples :

```
.set_value(this.key, %Code%-%ObjectID%)
```

Définit l'attribut volatile key sur l'objet courant

```
eg. .set_object(this.baseClass,
ChildGeneralizations.First.ParentObject)
```

Définit l'attribut volatile baseClass de type d'objet sur l'objet courant.

Opérateur de déréférencement
Les variables définies via la macro set_object sont appelées objet local, tandis que celles définies à l'aide de la macro set_value sont appelées valeurs locales. L'opérateur de déréférencement * peut être appliqué aux valeurs locales.

L'opérateur * permet d'évaluer la variable dont le nom est la valeur de la variable locale spécifiée.

```
%[.formatting-options:]*local-variable%
```

Par exemple, le code suivant :

```
.set_value(i, Code)
%*i%
```

Equivaut à :

```
%Code%
```

Options de formatage des variables

Vous pouvez incorporer des options de format dans la syntaxe d'une variable comme suit :

```
%.format:variable%
```

Les options de format pour les variables sont les suivantes :

Option option	Description
n	Extrait les n premiers caractères. Des espaces ou des zéros sont ajoutés à gauche pour compléter la largeur et justifier le résultat à droite
-n	Extrait les n derniers caractères. Des espaces ou des zéros sont ajoutés à droite pour compléter la largeur et justifier le résultat à gauche
L	Convertit les caractères en minuscules
U	Convertit les caractères en majuscules
c	Initiale majuscule et les autres caractères en minuscules
A	Supprime automatiquement le retrait à droite et aligne le texte sur le bord gauche
D	Renvoie la valeur lisible d'un attribut, telle qu'elle est affichée dans l'interface lorsque cette valeur diffère de la représentation interne de cet attribut. Dans l'exemple suivant, la valeur de l'attribut Visibility est stockée en interne sous la forme "+", mais s'affiche sous la forme "public" dans la feuille de propriétés : %Visibility% = + %.D:Visibility% = public
F	Combiné avec L et U, applique la conversion sur le premier caractère.
T	Les espaces de gauche et de droite sont supprimés de la variable
q	Place la variable entre apostrophes
Q	Place la variable entre guillemets
X	Ignore les caractères interdits pour XML
E	[obsolète – utiliser l'opérateur !, voir *Opérateurs* à la page 294]

Vous pouvez combiner les codes de format. Par exemple, %.U8:CHILD% convertit les 8 premiers caractères du code de la table CHILD en majuscules.

Opérateurs du langage de génération par template

Le langage de génération par template prend en charge des opérateurs arithmétiques standard ainsi que certains opérateurs de template avancés.

Les opérateurs arithmétiques et logiques suivants sont pris en charge, dans lesquels *x* et *y* peuvent être des nombres ou des templates dont la résolution produit des nombres :

Opérateur	Description
%+(*x*,*y*)%	Opérateur d'addition
%-(*x*,*y*)%	Opérateur de soustraction
%*(*x*,*y*)%	Opérateur de multiplication
%/(*x*,*y*)%	Opérateur de division
%&(*x*,*y*)%	Opérateur logique `and`

Dans cet exemple, le template contenu dans la colonne de gauche produit le résultat affiché dans la colonne de droite :

Template	Résultat

`Base number= %Number%`
`Number+1= %+(Number,1)%`
`Number-1= %-(Number,1)%`
`Number*2= %*(Number,2)%`
`Number/2= %/(Number,2)%`
`Number&1= %&(Number,1)%`
	`Base number= 4`
	`Number+1= 5`
	`Number-1= 3`
	`Number*2= 8`
	`Number/2= 2`
	`Number&1= 0`

Les opérateurs de template avancés suivants sont également pris en charge :

Opérateur	Description
*	Opérateur de déréférencement - La syntaxe [*]+*valeur-locale* [(*liste-param*)] renvoie le membre d'objet défini par l'évaluation de [*]+ *valeur-locale*. Si le membre d'objet fourni est un template, une liste de paramètres peut être spécifiée. Le fait d'appliquer l'opérateur astérisque correspond à une double évaluation (l'opérateur * agit comme un opérateur de déréférencement). Si une variable locale est défini sous la forme : `.set_value(C, Code)`, alors `%C%` va renvoyer "Code" et `%*C%` va renvoyer le résultat de l'évaluation de %Code%. En d'autres termes, `%*C%` peut être considéré comme `%(%C%)%` (cette dernière syntaxe étant incorrecte).

Opérateur	Description
!	Opérateur d'évaluation - Evalue le résultat de l'évaluation de la variable comme un template. Par exemple, vous définissez un commentaire contenant une variable telle que %Code%. Lorsque vous utilisez l'opérateur ! dans %!Comment%, la valeur réelle de %Code% est substituée au bloc de variable. Sans opérateur !, la variable n'est pas évaluée. L'opérateur ! peut être utilisé plusieurs fois. Par exemple : `%!!template%` Ceci produit le résultat de l'évaluation de l'évaluation de l'évaluation du template 'template'

?	L'opérateur ? est utilisé pour tester l'existence d'un template, d'une variable locale ou d'un attribut volatile ou étendu. Il renvoie "true" si la variable existe, "false" dans le cas contraire
	Par exemple, si custname est défini alors que custid ne l'est pas, alors le template :
	`.set_value(foo, tt)`
	`%custname?%`
	`%custid?%`
	Renvoie le résultat suivant :
	`true`
	`false`

+	L'opérateur + est utilisé pour tester si une propriété d'objet est visible dans l'interface. Par exemple, vous pouvez tester si la zone Type est affichée dans l'onglet Général de la feuille de propriétés d'une base de données dans le Modèle de Fluidité de l'Information, ce qui indique si le fichier XEM Replication Server est attaché au modèle. Le template %Database.Type+% produira false si aucun fichier XEM n'est associé au modèle courant.

Portée de la conversion

La portée de la conversion définit le contexte d'évaluation d'un template, en déterminant l'objet auquel le template est appliqué. La portée peut changer lors de la conversion d'un template, mais un seul objet peut être actif à la fois.

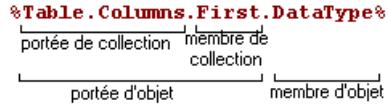
La portée initiale est toujours la métaclasse sur laquelle le template est défini. Tous les attributs de métamodèle et les collections définies sur la métaclasse d'objet active et ses parents sont visibles, de même que les attributs étendus et templates correspondants.

Vous pouvez modifier la portée en utilisant le caractère '.' (point), qui se comporte comme un opérateur d'indirection comme dans le langage de programmation Java, le côté droit correspond à un membre de l'objet référencé par le côté gauche.

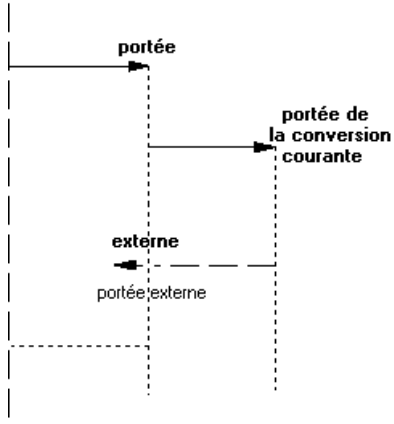
Les types de portée suivants sont disponibles :

- Portée de l'objet - Pour accéder aux membres d'un objet qui n'est pas actif dans la portée de la conversion courante, il est possible de spécifier une portée d'objet.
- Collection scope - Pour accéder aux membre d'une collection, vous pouvez spécifier une portée de collection. Pour plus d'informations sur les collections d'objet, voir *Chapitre 1, Utilisation des fichiers de ressources PowerAMC* à la page 1.

  Par exemple :



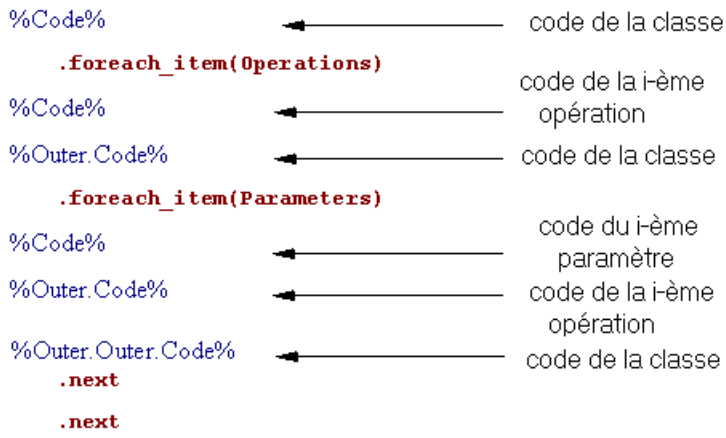
- Portée externe - Une portée externe peut être accessible à l'aide du mot clé Outer. Les règles suivantes s'appliquent :
  - Lorsqu'une portée est créée, l'ancienne portée devient la portée externe.
  - Lorsqu'une portée est quittée, sa portée externe est restaurée comme étant la portée de conversion courante



De nouvelles portées peuvent être créées lors de l'évaluation d'un template qui force l'objet à changer. Par exemple, la macro foreach_item (voir *Macro .foreach_item* à la page 316) qui permet l'itération sur les collections définit une nouvelle portée, de même que la macro foreach_line (voir *Macro .foreach_line* à la page 317). La portée externe est restaurée à la fin du bloc.

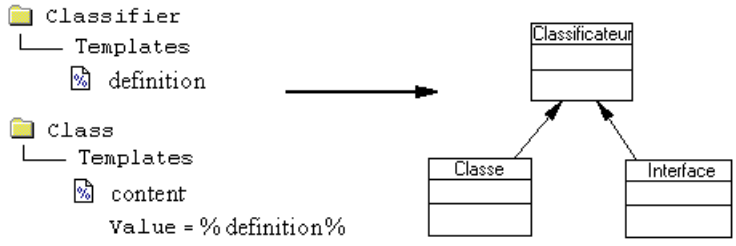
Les portées imbriquées forment une hiérarchie qui peut être affichée sous la forme d'une arborescence, la portée de plus haut niveau étant la racine.

L'exemple suivant montre le mécanisme de la portée à l'aide d'un template de classe :



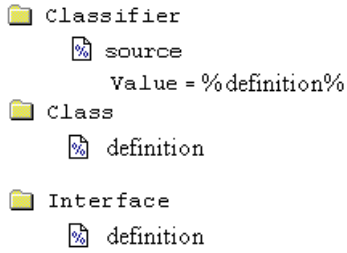
Héritage et polymorphisme

Les templates sont définis par rapport à une métaclasse donnée et sont hérités par et disponibles pour les enfants de cette métaclasse. Dans l'exemple suivant, le template de définition spécifié sur la métaclasse parent est utilisé dans l'évaluation du template de contenu sur la métaclasse enfant.

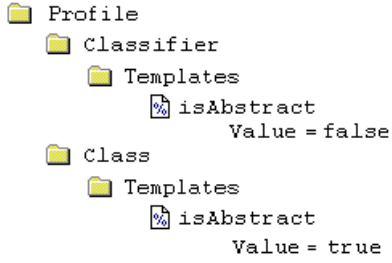


GTL prend en charge les concepts orientés objet suivants comme faisant partie de l'héritage :

- *Polymorphisme* - Les templates sont liés de façon dynamique. En d'autres termes, le choix du template à évaluer est effectué au moment de la conversion. Le polymorphisme permet au code de template défini sur un classificateur d'utiliser des templates définis sur ses enfants (classe, interface), le template utilisé n'a pas à être défini sur la métaclasse parent. Combinée avec les héritages, cette fonctionnalité permet de partager le code de template. Dans l'exemple suivant, le contenu de %definition% dépend de la nature de l'objet traité (classe ou interface) :



- *Redéfinition de template* - Un template défini par rapport à une métaclasse peut être redéfini sur une classe enfant. Le template défini sur l'enfant remplace le template définir sur le parent pour les objets de métaclasses enfant. Vous pouvez visualiser le parent rédéfini en utilisant la commande Afficher la super-définition dans le menu contextuel de la classe enfant, et spécifier l'utilisation du template parent en utilisant l'opérateur qualifiant "::". Par exemple :



  Le même nom de template "isAbstract" est utilisé dans deux catégories différentes : Classifier et Class. "false" est la valeur d'origine qui a été redéfinie par la nouvelle valeur "true". Vous pouvez récupérer la valeur d'origine en utilisant la syntaxe suivante : <metaclassName::template>, dans ce cas :

```
%isAbstract%
%Classifier::isAbstract%
```

- *Surcharge de template* - Vous pouvez avoir plusieurs définitions du même template qui s'appliquent à différentes conditions. Les templates peuvent également être définis sous les critères et stéréotypes (voir *Critères (Profile)* à la page 57 et *Stéréotypes (Profile)* à la page 53), et les conditions correspondantes sont combinées. Lors de la conversion, chaque condition est évaluée et le template approprié (ou, en l'absence de correspondance, le template par défaut) est appliqué. Par exemple:

```
nom-template-complet = (syntaxe1)  <nom-template>
                                   (syntaxe2)  <nom-template>'<<' stereotype '>>'
                                   (syntaxe3)  <nom-template>'<' <condition-simple> '>'
nom-template             = <texte>
```

Conversion d'un raccourci

Les raccourcis sont déréférencés lors de la conversion : la portée de l'objet cible remplace la portée du raccourci.

Par exemple, le fichier généré suivant défini dans la métaclasse package fournit la liste des classes contenues dans le package. Si un raccourci vers une classe est trouvé, le code de son objet cible suivi de (Raccourci) est généré, suivi par l'ID de l'objet parent, puis par l'ID du raccourci, ce qui montre clairement que la portée du raccourci est remplacée par la portée de l'objet cible du raccourci :

```
.foreach_item(Classes)
 .if (%IsShortcut%)
%Code% (Shortcut)
oid = %ObjectID%
shortcut oid = %Shortcut.ObjectID%
 .else
%Code%
%Shortcut%
 .endif
.next(\n)
```

Ce comportement est l'inverse de celui du VB Script, dans lequel la conversion des raccourcis récupère le raccourci lui-même.

Si vous souhaitez générer le raccourci lui-même plutôt que l'objet auquel il fait référence, vous pouvez utiliser la variable %Shortcut%.

Raccourci externe

Si le modèle cible d'un raccourci externe n'est pas ouvert, une boîte de dialogue de confirmation apparaît pour vous permettre d'ouvrir le modèle cible. Vous pouvez utiliser la macro set_interactive_mode pour changer ce comportement. Cette macro permet de décider si l'exécution du GTL doit s'effectuer avec des interactions avec l'utilisateur ou non.

Pour plus d'informations sur la macro set_interactive_mode, reportez-vous à la section *Macro .set_interactive_mode* à la page 324.

Séquences d'échappement

Des séquences d'échappement sont des séquences de caractères spécifiques utilisées pour configurer la présentation du fichier généré.

Les séquences d'échappement suivantes peuvent être utilisées dans des templates :

Séquence d'échappement	Description
\n	Caractère de passage à la ligne, crée une nouvelle ligne
\t	Caractère de tabulation, crée une tabulation

Séquence d'échappement	Description
\\	Crée une barre oblique inverse
\ au début d'une ligne	Crée un caractère de suite (ignore la nouvelle ligne)
. au début d'une ligne	Ignore la ligne
.. au début d'une ligne	Crée un caractère point (pour générer une macro)
%%	Crée un caractère pourcent

Pour plus d'informations sur les séquences d'échappement, reportez-vous à la section *Utilisation de nouvelles lignes dans la chaîne d'en-tête et de fin* à la page 301.

Partage de templates

Dans le mécanisme du langage de génération par template, vous pouvez partager des conditions, des templates et des sous-templates afin de faciliter la maintenance du langage et le rendre plus lisible.

Partage de conditions

Un template peut contenir une expression de condition. Vous avez également la possibilité de créer des templates pour partager des expressions de condition longues et fastidieuses :

Nom de template	Valeur de template
%ConditionVariable%	.bool (condition)

Au lieu de répéter la condition dans d'autres templates, vous utilisez simplement %ConditionVariable% dans la macro conditionnelle :

```
.if (%ConditionVariable%)
```

Exemple

Le template %isInner% contient une condition qui renvoie true si le classificateur est interne à un autre classificateur.

```
.bool (%ContainerClassifier%!=null)
```

Ce template est utilisé dans le template %QualifiedCode% permettant de définir le code qualifié du classificateur :

```
.if (%isInner%)
    %ContainerClassifier.QualifiedCode%::%Code%
.else
    %Code%
.endif
```

Utilisation des templates récursifs

Un template récursif est template qui est défini par rapport à lui-même.

Exemple

Considérons trois X, Y et Z. X est interne à Y, et Y est interne à Z.

La variable %topContainerCode% est définie pour extraire la valeur du conteneur parent d'une classe.

La valeur du template est la suivante :

```
.if (%isInner%)
    %ContainerClassifier.topContainerCode%
.else
    %Code%
.endif
```

Si la classe est interne pour une autre classe, %topContainerCode% est appliqué à la classe conteneur de la classe courante (%ContainerClassifier.topContainerCode%).

Si la classe n'est pas une classe interne, le code de la classe est généré.

Utilisation de nouvelles lignes dans la chaîne d'en-tête et de fin

Les chaînes d'en-tête et de fin sont utiles car elles ne sont générées que lorsque nécessaire, cela est particulièrement utile lorsque vous utilisez de nouvelles lignes '\n'. Elles sont ajoutées respectivement au début et à la fin du code généré, la chaîne d'en-tête et la chaîne de fin n'apparaissant pas dans le code généré.

Exemple

Vous souhaitez générer le nom d'une classe et ses attributs sous le format suivant (une ligne vide entre les attributs et la classe) :

```
Attribute 1  attr1
Attribute 2  attr2

Class
```

Vous pouvez insérer le séparateur "\n" après l'instruction .foreach pour vous assurer que chaque attribut s'affiche dans une ligne séparée. Vous pouvez également ajouter "\n\n " après l'instruction .endfor pour insérer une ligne vide après la liste d'attributs et avant le mot "Class".

```
.foreach (Attribute) ("\n")
Attribute %Code%
.endfor ("\n\n")
Class
```

Exemple supplémentaire

Considérons une classe nommée *Nurse*, ayant pour code de classe Nurse, et dotée de deux attributs :

Attribut	Type de données	Valeur initiale
NurseName	String	__
NurseGender	Char	'F'

Les templates suivants sont fournis à titre d'exemple, avec le texte généré pour chacune d'entre eux, ainsi qu'une description de chaque résultat :

Template 1

```
class "%Code%" {
 // Attributes
 .foreach_item(Attributes)
 %DataType% %Code%
  .if (%InitialValue%)
 = %InitialValue%
  .endif
 .next
 // Operations
 .foreach_item(Operations)
 %ReturnType% %Code%(...)
 .next
}
```

Texte généré 1

```
class "Nurse" {
 // Attributes String nurseName char nurseGender = 'F' // Operations}
```

Description 1

Au-dessous du code de classe, le code est généré sur une ligne. Il s'agit d'un exemple d'une macro de bloc (macro .if, .endif).

Template 2 (nouvelle ligne)

```
class "%Code%" {
 // Attributes
 .foreach_item(Attributes)
 %DataType% %Code%
  .if (%InitialValue%)
 = %InitialValue%
  .endif
 .next(\n)
 // Operations
 .foreach_item(Operations)
 %ReturnType% %Code%(...)
 .next(\n)
}
```

Texte généré 2

```
class "Nurse" {
```

```
 // Attributes String nurseName

 char nurseGender = 'F' // Operations}
```

Description 2

String nurseName et char nurseGender se trouvent sur deux lignes distinctes

Dans Template 1, String nurseName et char nurseGender se trouvaient sur la même ligne, alors que dans Template 2, l'ajout de \n à la fin de .next(\n) place String nurseName et char nurseGender sur deux lignes distinctes.

En outre, // Operations est affiché dans le résultat et ce, même en l'absence d'opération (voir Description 3).

Template 3 (blanc)

```
class "%Code%" {
 .foreach_item(Attributes, // Attributes\n,\n)
 %DataType% %Code%
  .if (%InitialValue%)
 = %InitialValue%
  .endif
 .next(\n)
 .foreach_item(Operations, // Operations\n,\n)
 %ReturnType% %Code%(...)
 .next(\n)
}
```

Texte généré 3

```
class "Nurse" {// Attributes

 String nurseName

 char nurseGender = 'F'

}
```

Description 3

L'espace entre *.foreach_item(Attributes,* et *// Attributes\n,\n)* n'est pas généré, comme indiqué dans le résultat suivant : class "Nurse" {// Attributes au lieu de .... { // Attributes

// Operations n'est pas affiché dans le résultat car il est placé dans la macro .foreach_item. Il est placé dans l'en-tête de la macro à cet effet.

Template 4 (blanc)

```
class "%Code%" {\n
 .foreach_item(Attributes," // Attributes\n",\n)
 %DataType% %Code%[ = %InitialValue%]
 .next(\n)
 .foreach_item(Operations," // Operations\n",\n)
 %ReturnType% %Code%(...)
```

```
 .next(\n)
}
```

Texte généré 4

```
class "Nurse" {
 // Attributes
 String nurseName
 char nurseGender = 'F'
}
```

Description 4

Le caractères guillemet ("") dans " // Attributes\n" permet d'insérer un espace comme indiqué dans le résultat : // Attributes

---

Remarque : La nouvelle ligne qui précède immédiatement une macro est ignorée, de même que celle qui la suit, comme dans l'exemple suivant :

Jack .set_value(v, John) Paul yields: JackPaul

instead of: Jack Paul

Utilisation du passage de paramètres

Vous pouvez passer des paramètres dans, hors ou dans/hors un template via des variables locales en tirant parti des portées de traduction. Vous pouvez accéder à des paramètres avec la variable %@<number>%.

Exemple

Templates de classe :

Template 1

```
<show> template
<<<
Class "%Code%" attributes :
// Public
%publicAttributes%

// Protected
%protectedAttributes%

// Private
%privateAttributes%
>>>
```

Template 2

```
<publicAttributes> template
<<<
```

```
.foreach_item(Attributes)
 .if (%Visibility% == +)
 %DataType %Code%
 .endif
.next(\n)
>>>
```

Template 3

```
<protectedAttributes> template
<<<
.foreach_item(Attributes)
 .if (%Visibility% == #)
 %DataType %Code%
 .endif
.next(\n)
>>>
```

Template 4

```
<privateAttributes> template
<<<
.foreach_item(Attributes)
 .if (%Visibility% == -)
 %DataType %Code%
 .endif
.next(\n)
>>>
```

Pour améliorer la lisibilité et rendre le code encore plus réutilisable, ces quatre templates peuvent être écrits dans deux templates à l'aide de paramètres :

Premier template

```
<show> template
<<<
Class "%Code%" attributes :
// Public
%attributes(+)%

// Protected
%attributes(#)%

// Private
%attributes(-)%
>>>
```

Second template

```
<attributes> template
<<<
.foreach_item(Attributes)
 .if (%Visibility% == %@1%)
 %DataType %Code%
 .endif
```

```
.next(\n)
>>>
```

Description
Le premier paramètre dans cet exemple %attributes(+, ou #, ou -)% peut être accessible via la variable %@1%, le second, s'il existe, est accessible via la variable %@2% variable, etc...

Messages d'erreur

Les messages d'erreur interrompent la génération du fichier dans lequel des erreurs ont été trouvées. Ces erreurs sont affichées dans l'onglet Aperçu de la feuille de propriétés de l'objet correspondant.

Les messages d'erreur ont le format suivant :

```
cible::catg-chemin nom-complet-template(numéro-ligne)
métaclasse-objet-actif code-objet-actif):
    type-erreur message-erreur
```

Vous pouvez rencontrer les types d'erreur suivants :

- Erreurs de syntaxe
- Erreurs de conversion

Erreurs de syntaxe

Vous pouvez rencontrer les erreurs de syntaxe suivantes :

Message d'erreur	Description et correction
Erreur de syntaxe dans la condition	Erreur de syntaxe dans une expression booléenne
.endif attendu	Ajoutez un .endif
.else sans .if correspondant	Ajoutez un .if au .else
.endif sans .if correspondant	Ajoutez un .if au .endif
.next attendu	Ajoutez un .next
.end%s attendu	Ajoutez un .end%s (par exemple, .endunique, .endreplace, ...)
.end%s sans .%s correspondant	Ajoutez un .*macro* au .end*macro*
.next sans .foreach correspondant	Ajoutez un .foreach au .next
Parenthèses manquantes ou non appariées	Corrigez les éventuelles accolades non appariées
Paramètre inattendus : *params-supplémentaires*	Supprimez les paramètres nécessaires
Macro inconnue	La macro n'est pas valide

Message d'erreur	Description et correction
.execute_command incorrect syntax	La syntaxe appropriée s'affiche dans l'onglet Aperçu, ou bien dans la fenêtre Devrait être : .execute_command(executable [,*arguments*[,{cmd_ShellExecute|cmd_PipeOutput}]])
Syntaxe incorrecte pour Change_dir	La syntaxe doit être : .change_dir(*path*)
Syntaxe incorrecte pour convert_name	La syntaxe doit être : .convert_name(*name*)
Syntaxe incorrecte pour convert_code	La syntaxe doit être : .convert_code(*code*)
Syntaxe incorrecte pour set_object	La syntaxe doit être : .set_object(*local-var-name* [,[*scope.*] *portée-objet* [,{new|update}]])
Syntaxe incorrecte pour set_value	La syntaxe doit être : .set_value(*local-var-name*,*template-simple*[,{new|update}])
Syntaxe incorrecte pour execute_vbscript	La syntaxe doit être : .execute_vbscript(*script-file* [,*script-input_params*])

Erreurs de conversion

Les erreurs de conversion sont des erreurs d'évaluation sur une variable lorsque vous évaluez un template.

Vous pouvez rencontrer les erreurs de conversion suivantes :

Message d'erreur de conversion	Description et correction
Collection non résolue : *collection*	Collection inconnue
Membre non résolu : *membre*	Membre inconnu
Aucune portée externe	Utilisation incorrecte du mot clé
Objet null	Se produit lors d'une tentative d'accès à un membre d'un objet null
Variable objet attendue : *objet*	Se produit lorsqu'une chaîne est utilisée à la place d'un objet
Erreur d'exécution VBScript	Erreur de script VB
Blocage détecté	Blocage dû à une boucle infinie

Guide de référence des macros du langage de génération par template

Les macros peuvent être utilisées pour exprimer la logique, et pour boucler sur des collections d'objets. Chaque mot clé de macro doit être précédé d'un caractère "." (point) et doit être le

premier caractère, autre qu'un espace, sur une ligne. Prenez soin de respecter la syntaxe des macros en termes de passage à la ligne.

Vous pouvez définir une macro dans un template, ou dans une commande.

Il existe trois types de macros :

- Les *macros simples* sont les macros qui ne sont constituées que d'une seule ligne.
- Les *macros de bloc* se composent d'un mot clé de début et d'un mot clé de fin délimitant un bloc auquel la macro est appliquée. Leur structure se présente comme suit :

```
.nom-macro [(paramètres)]
    bloc-entrée
.endnom-macro [(fin)]
```

- Les *macros de boucle* sont utilisées pour l'itération. A chaque itération, une nouvelle portée est créée. Le template spécifié dans le bloc est converti simultanément conformément à la portée d'itération.

```
.foreach_nom-macro [(paramètres[,en-tête[,fin]])]
    template-complexe
.next[(séparateur)]
```

Remarque : Les paramètres de macro peuvent être délimités par des guillemets. Les délimiteurs sont requis lorsque la valeur du paramètre inclut des virgules, des accolades et des espaces de début ou de fin. La séquence d'échappement pour les guillemets au sein d'un paramètre est \".

Pour pouvez utiliser les macros suivantes :

- *Macros conditionnelles et macro de boucle/itératives* :
  - *Macro .if* à la page 320
  - *Macro .foreach_item* à la page 316 – permet l'itération sur les collections d'objets
  - *Macro .foreach_line* à la page 317 – permet l'itération sur les lignes
  - *Macro .foreach_part* à la page 318 –permet l'itération sur les parties
  - *Macro .break* à la page 310 – interrompt la boucle
- *Macros d'affectation* - définit une variable locale ou un type valeur ainsi que des attributs volatiles :
  - *Macros .set_object et .set_value* à la page 325
  - *Macro .unset* à la page 326
- *Macros de résultats et de signalisation d'erreurs* :
  - *Macro .log* à la page 321
  - *Macros .error et .warning* à la page 314
- *Macros de commandes* - uniquement disponibles dans le contexte de l'exécution d'une commande générique :
  - *Macro .vbscript* à la page 327 - incorpore du code VB script dans un template
  - *Macro .execute_vbscript* à la page 315 - lance l'exécutions de scripts VB

Macro .abort_command

Cette macro stoppe l'exécution de la commande. Elle est disponible pour exécuter des commandes de génération uniquement, et peut être combinée aux macros standard du langage de génération par template lorsque vous définissez des commandes.

Exemple :

```
.if %_JAVAC%
    .execute (%_JAVAC%,%FileName%)
.else
    .abort_command
.endif
```

Macro .block

La macro .block est utilisée pour ajouter un en-tête et/ou une fin à son contenu lorsque ce dernier n'est pas vide.

```
.block [(en-tête)]
    bloc-entrée
.endblock[(fin)]
```

Les paramètres suivants sont disponibles :

Parameter	Description
en-tête	[facultatif] Généré avant le résultat, s'il y en un. Type : Template simple
bloc-entrée	Paramètre utilisé pour spécifier du texte Type : Template complexe
fin	[facultatif] Ajouté au résultat, s'il y en a un Type : Text

Le résultat est la concaténation de *en-tête*, l'évaluation de *bloc-entrée* et *fin*.

Exemple :

```
.block (<b>)
The current text is in bold
.endblock (</b>)
```

Macro .bool

Cette macro renvoie 'true' or 'false' en fonction de la valeur de la condition spécifiée.

```
.bool (condition)
```

Les paramètres suivants sont disponibles :

Paramètre	Description
condition	Condition à évaluer Type : Condition

Exemple :

```
.bool(%.3:Code%= =ejb)
```

Macro .break

Cette macro peut être utilisé pour sortir des boucles .foreach.

```
.break
```

Exemple :

```
.set_value(_hasMain, false, new)
.foreach_item(Operations)
   .if (%Code% == main)
      .set_value(_hasMain, true)
      .break
   .endif
.next
%_hasMain%
```

Macro .change_dir

Cette macro change le répertoire courant. Elle est disponible pour exécuter des commandes de génération uniquement, et peut être combinée aux macros standard du langage de génération par template lorsque vous définissez des commandes.

```
.change_dir (chemin)
```

Les paramètres suivants sont disponibles :

Paramètre	Description
chemin	Nouveau répertoire courant Type : Template simple (séquences d'échappement ignorées)

Exemple :

```
.change_dir(C:\temp)
```

Macro .collection

Cette macro renvoie une collection d'objets basée sur la portée et la conditions spécifiées. Les collections sont représentées comme concaténation de l'OID terminé par un point virgule.

```
.collection (portée-collection [,filtre])
```

Les paramètres suivants sont disponibles :

Paramètre	Description
portée-collection	Portée sur laquelle l'itération doit être effectuée. Type : <template-simple> renvoyant une portée de collection
filtre	[facultatif] Condition de filtre Type : condition

Exemple :

La macro suivante renvoie un sous-ensemble d'attributs définis sur le classificateur courant et dont le code commence par une lettre comprise entre a et e.

```
.object(Attributes, (%.1:Code% >= a) and (%.1:Code% <= e))
```

Résultat :

```
C3ADA38A-994C-4E15-91B2-08A6121A514C;58CE2951-7782-49BB-
B1BB-55380F63A8C9;F522C0AE-4080-41C2-83A6-2A2803336560;
```

Macro .comment et macro .//

La macro .comment et la macro .// sont utiles pour insérer des commentaires dans un template. Les lignes qui commencent par .// ou par .comment sont ignorées lors de la génération.

Exemple :

```
.// This is a comment
.comment This is also a comment
```

Macros .convert_name et .convert_code

Ces macros macros convertissent le nom d'un objet en son code (ou l'inverse).

Utilisez la syntaxe suivante pour convertir un nom en code :

```
.convert_name (expression[,"séparateur"[,"motif_séparateur"],case])
```

Utilisez la syntaxe suivante pour convertir un code en nom :

```
.convert_code (expression[,"séparateur"[,"motif_séparateur"]])
```

Les paramètres suivants sont disponibles :

Paramètre	Description
expression	Spécifie le texte à convertir. Dans le cas de .convert_name, il s'agit le plus souvent de la variable %Name% et peut inclure un suffixe ou un préfixe. Type : Template simple
séparateur	[facultatif] Caractère généré chaque fois qu'un séparateur déclaré dans **motif_séparateur** est trouvé dans le code. Par exemple, "_" (tiret bas). Type : Texte
motif_séparateur	[facultatif]] Déclaration des différents séparateurs qui peuvent exister dans un nom, et qui seront remplacés par **séparateur**. Vous pouvez déclarer plusieurs séparateurs, par exemple "_ " et "-" Type : Texte
casse	[facultatif pour .convert_name uniquement] Spécifie la casse dans laquelle convertir le code. Vous pouvez choisir l'une des valeurs suivantes : • firstLowerWord - Premier mot en minuscules, première lettre des mots suivants en majuscule • FirstUpperChar - Première lettre de chaque mot en majuscule • lower_case - Tous les mots en minuscules et séparés par un tiret bas • UPPER_CASE - Tous les mots en majuscules et séparés par un tiret bas

Dans l'exemple suivant, la macro .convert_name est ajoutée depuis le dossier Profile \Column dans une nouvelle entrée Generated Files :

```
.foreach_item(Columns)
 %Name%,
 .foreach_part(%Name%)
  .convert_name(%CurrentPart%)
 .next("_")
.next(\n)
```

Remarque : Ces macros peuvent être utilisées pour effectuer des conversions afin d'appliquer des conventions de dénomination dans votre modèle. Pour plus d'informations, voir *Guide des fonctionnalités générales > L'interface de PowerAMC > Objets > Propriétés d'un objet > Conventions de dénomination*.

Macro .create_path

Cette macro créé un chemin spécifié si ce dernier n'existe pas.

```
.create_path (chemin)
```

Les paramètres suivants sont disponibles :

Paramètre	Description
chemin	Chemin à créer Type : Template simple (séquences d'échappement ignorées)

Exemple :

```
.create_path(C:\temp)
```

Macro .delete

Supprime toutes les instances de la chaîne *chaîne-suppr* dans *bloc-entrée-suppr*.

```
.delete (del-string)
    block-input
.enddelete
```

Cette macro est particulièrement utile lorsque vous travaillez avec des conventions de dénomination (voir *Guide des fonctionnalités générales > L'interface de PowerAMC > Objets > Propriétés d'un objet > Conventions de dénomination*).

Les paramètres suivants sont disponibles :

Paramètre	Description
chaîne-suppr	Chaîne à supprimer dans le bloc. Type : Texte
bloc-entrée-suppr	Paramètre est utilisé pour spécifier du texte. Type : Template complexe

Exemple :

Dans l'exemple suivant, GetCustomerName est converti en CustomerName:

```
.delete( get )
    GetCustomerName
.enddelete
```

Dans l'exemple suivant, la variable %Code% a la valeur m_myMember et est convertie en myMember :

```
.delete(m_)
    %Code%
.enddelete
```

Macros .error et .warning

Ces macros sont utilisées pour émettre des messages d'erreur et des avertissements lors de la conversion. Les erreurs interrompent la génération, tandis que les avertissements sont émis à titre d'information uniquement et peuvent être déclenchés si une incohérence est détectée lorsque vous appliquez le template sur un objet particulier. Les messages sont affichés à la fois dans le volet Aperçu et dans la fenêtre Résultats.

Utilisez la syntaxe suivante pour insérer un message d'erreur :

```
.error message
```

Utilisez la syntaxe suivante pour insérer un message d'avertissement:

```
.warning message
```

Les paramètres suivants sont disponibles :

Paramètre	Description
message	Message d'erreur Type : Template simple

Exemple :

```
.error no initial value supplied for attribute %Code% of class
%Parent.Code%
```

Macro .execute_command

Cette macro est utilisée pour lancer des exécutables sous forme de processus séparés. Elle est disponible pour exécuter des commandes de génération uniquement, et peut être combinée aux macros standard du langage de génération par template lorsque vous définissez des commandes.

```
.execute_command (cmd [,args [,mode]])
```

Les paramètres suivants sont disponibles :

Paramètre	Description
cmd	Chemin d'accès d'exécutable Type : Template simple (séquences d'échappement ignorées)
args	[facultatif] Arguments pour l'exécutable Type : Template simple (séquences d'échappement ignorées)
mode	[facultatif] Vous pouvez choisir l'une des valeurs suivantes : • cmd_ShellExecute - est exécuté comme processus indépendant • cmd_PipeOutput - bloque jusqu'à la fin de l'exécution, puis montre le résultat de l'exécutable dans la fenêtre Résultats

Remarquez que si une commande .execute_command échoue pour une raison quelconque (exécutables non trouvés, ou bien résultat envoyé vers stderr), l'exécution de la commande est interrompue.

Exemple :

```
.execute_command(notepad, file1.txt, cmd_ShellExecute)
```

Macro .execute_vbscript

Cette macro est utilisée pour exécuter un script VB spécifié dans un fichier séparé.

```
.execute_vbscript (fichier-vbs [,paramètre-script])
```

Les paramètres suivants sont disponibles :

Paramètre	Description
fichier-vbs	Chemin d'accès du fichier VB script Type : Template simple (séquences d'échappement ignorées)
paramètre-script	[facultatif] Paramètre passé au script via la propriété globale ScriptInputParameters. Type : Template simple

Le résultat est la valeur de la propriété globale ScriptResult.

Exemple :

```
.execute_vbscript(C:\samples\vbs\login.vbs, %username%)
```

Remarque : l'objet actif de la portée de conversion courante est accessible via la collection ActiveSelection en tant que ActiveSelection.Item(0).

Pour plus d'informations sur ActiveSelection, voir *Propriétés globales* à la page 355.

Macro .foreach_item

Cette macro est utilisée pour l'itération dans les collections d'objet :

```
.foreach_item (collection [,en-tête [,fin [,condition
[,comparaison]]]])
    template-complexe
.next [(séparateur)]
```

Le template spécifié au sein du bloc est converti sur tous les objets contenus dans la collection spécifiée.

Si une comparaison est spécifiée, les éléments de la collection sont pré-triés en fonction de la règle correspondante avant leur itération.

Les paramètres suivants sont disponibles :

Paramètre	Description
collection	Collection sur laquelle l'itération est effectuée Type : Template simple
en-tête	[facultatif] Généré avant le résultat, s'il y en a un Type : Texte
fin	[facultatif] Ajouté au résultat, s'il y en a un Type : Texte
condition	[facultatif] Si spécifié, seuls les objets qui satisfont la condition sont considérés lors de l'itération Type : Condition simple
comparaison	[facultatif] est évalué au sein d'une portée dans laquelle deux objets locaux respectivement nommés 'Item1' et 'Item2' sont définis. Ils correspondent aux éléments dans la collection. <comparaison> doit être évalué comme true si Item1 doit être placé après Item2 dans l'itération Type : Condition simple
template-complexe	Template à appliquer à chaque élément. Type : Template complexe
séparateur	[facultatif] Générer entre deux évaluations de <template-complexe> non vides Type : Texte

Remarque : Les paramètres de macro peuvent être délimités par des guillemets. Les délimiteurs sont requis lorsque la valeur du paramètre inclut des virgules, des accolades et des espaces de début ou de fin. La séquence d'échappement pour les guillemets au sein d'un paramètre est `\"`.

Exemple :

Attribut	Type de données	Valeur initiale
cust_name	String	__
cust_foreign	Boolean	false

```
.foreach_item(Attributes,,,,%Item1.Code% >= %Item2.Code%))
    Attribute %Code%[ = %InitialValue%];
.next(\n)
```

Le résultat est le suivant :

Attribute cust_foreign = false

Attribute cust_name;

Remarque
Les quatre virgules après (Attributes,,,, signifient que tous les paramètres (en-tête, fin, condition et comparison) sont sautés.

Macro .foreach_line

La macro foreach_line est une macro simple qui procède à l'itération sur les lignes du template de saisie spécifié comme premier argument pour la macro. Le template spécifié dans le bloc est converti pour chaque ligne de l'entrée. Cette macro crée une nouvelle portée avec la variable locale CurrentLine. Cette dernière est définie dans le bloc comme étant la i-ème ligne du template en entrée dans l'itération i.

```
.foreach_line (input [,en-tête [,fin]])
    template-complexe
.next [(séparateur)]
```

Les paramètres suivants sont disponibles :

Paramètre	Description
input	Texte en entrée, sur lequel l'itération est effectuée Type : Template simple
en-tête	[facultatif] Généré avant le résultat, s'il y en a un Type : Texte
fin	[facultatif] Ajouté au résultat, s'il y en a un Type : Texte
template-complexe	Template à appliquer à chaque ligne. Type : Template complexe

Paramètre	Description
séparateur	[facultatif] Généré entre évaluations non vides de *template-complexe* Type : Texte

Exemple :

```
.foreach_line(%Comment%)
// %CurrentLine%
.next(\n)
```

Macro .foreach_part

Cette macro permet une itération et une transformation des parties du template d'entrée, avec des parties délimitées par un motif séparateur.

```
.foreach_part (expression [,"sépateur" [,en-tête [,fin]]])
    template-simple
.next[(séparateur)]
```

Cette macro crée une nouvelle portée dans laquelle la variable locale CurrentPart est définie comme la i-ème partie du template en entrée à l'itération i. La variable locale Separator contient le séparateur suivant.

Cette macro est souvent utilisé afin d'appliquer des conventions de dénomination (voir *Guide des fonctionnalités générales > L'interface de PowerAMC > Objets > Propriétés d'un objet > Conventions de dénomination*).

Les paramètres suivants sont disponibles :

Paramètre	Description
entrée	Texte en entrée sur lequel l'itération est effectuée Type : Template simple

Paramètre	Description
motif-séparateur	Séparateurs de caractères et de mots • Tout caractère spécifié dans le motif peut être utilisé comme séparateur • [<c1> - <c2>] spécifie un caractère au sein de la plage définie entre les caractères <c1> et <c2> Par exemple, le motif suivant " -_,[A-Z]" spécifie que chaque partie peut être séparée par un espace, un tiret, un trait de soulignement, une virgule ou un caractère compris entre A et Z (majuscule). Par défaut, le <motif-séparateur> est initialisé avec le motif (). Si le motif spécifié est vide, le motif est initialisé à l'aide de la valeur par défaut. Un séparateur >séparateur> peut être concaténé entre chaque partie. Les expressions <en-tête> et <fin> peuvent être ajoutées respectivement au début ou à la fin de l'expression générée. Il existe deux types de séparateur : • Séparateur de caractères : pour chaque séparateur de caractères, le séparateur spécifié dans la prochaine instruction de la macro est renvoyé (même pour des séparateurs consécutifs) • Séparateur de mots : ils sont spécifiés en tant qu'intervalles, par exemple [A-Z] spécifie que toutes les lettres majuscules sont des séparateurs. Pour un séparateur de mots, aucun séparateur (spécifié dans la prochaine instruction) n'est renvoyé Valeur par défaut : " -_,\t" Type : Texte
en-tête	[facultatif] Généré avant le résultat, s'il y en a un Type : Texte
fin	[facultatif] Ajouté au résultat, s'il y en a un Type : Texte
template-simple	Template à appliquer à chaque partie. Type : Template complexe
séparateur	[facultatif] Généré entre évaluations non vides de *template-complexe* Type : Texte

Exemples :

Convertit un nom en code de classe (conventions de dénomination Java). Dans l'exemple suivant, la variable *%Name%* équivaut à Employee shareholder', et est convertie en EmployeeShareholder :

```
.foreach_part (%Name%, " _-'")
    %.FU:CurrentPart%
.next
```

Convertit un nom en code d'attribut de classe (conventions de dénomination Java). Dans l'exemple suivant, la variable *%Name%* équivaut à Employee shareholder, et est convertie en EmployeeShareholder :

```
.set_value(_First, true, new)
.foreach_part(%Name%,"' _-'")
    .if (%_First%)
        %.L:CurrentPart%
        .set_value(_First, false, update)
    .else
        %.FU:CurrentPart%
    .endif
.next
```

Macro .if

La macro if est utilisée pour la génération conditionnelle, et a la syntaxe suivante :

```
.if[not] condition
        template-complexe
    [(.elsif[not] condition
        template-complexe)*]
    [.else
        template-complexe]
.endif [(fin)]
```

Les paramètres suivants sont disponibles :

Paramètre	Description
condition	La condition à évaluer, sous la forme : `variable [opérateur comparaison]` *opérateur* peut être ==, =, <=, >=, < ou bien >. Si les deux opérandes sont des entiers, les opérateurs <, >, >= et <= procèdent à des comparaisons d'entiers, dans le cas contraire, ils procèdent à une comparaison de chaînes qui prend en compte des nombres incorporés (exemple : Class_10 est supérieur à Class_2). *comparaison* peut être : • Un template simple • "*text*" • true • false • null • notnull Si aucun opérateur ni condition n'est spécifié, la condition est évaluée à true à moins que la valeur de la variable ne soit false, null ou la chaîne null. Vous pouvez enchaîner des conditions en utilisant les opérateurs logiques and ou or. Type : Template simple
template-complexe	Template à appliquer si la condition est satisfaite. Type : Template complexe
fin	Ajouté au résultat, s'il y en a un Type : Texte

Macro .log

Cette macro consigne un message dans l'onglet Génération de la fenêtre Résultats, située dans la partie inférieure de la fenêtre principale. Elle est disponible pour exécuter des commandes de génération uniquement, et peut être combinée aux macros standard du langage de génération par template lorsque vous définissez des commandes.

```
.log message
```

Les paramètres suivants sont disponibles :

Paramètre	Description
message	Message à consigner Type : Template simple

Exemple :

```
.log undefined environment variable: JAVAC
```

Macros .lowercase et .uppercase

Ces macros convertissent des blocs de texte dans la casse spécifiée.

```
.lowercase
    block-input
.endlowercase
```

and

```
.uppercase
    block-input
.enduppercase
```

Ces macros sont particulièrement utiles lorsque vous travaillez avec les conventions de dénomination (voir *Guide des fonctionnalités générales > L'interface de PowerAMC > Objets > Propriétés d'un objet > Conventions de dénomination*).

Les paramètres suivants sont disponibles :

Paramètre	Description
bloc-entrée	Un paramètre utilisé pour spécifier du texte Type : Template complexe

Dans l'exemple suivant, la variable %Comment% a pour valeur HELLO WORLD, qui est convertie en hello world.

```
.lowercase
    %Comment%
.endlowercase
```

Macro .object

Cette macro renvoie une collection d'objets en fonction de la portée et de la condition spécifiées. Les références d'objet sont représentées sous forme d'OID ; par exemple : E40D4254-DA4A-4FB6-AEF6-3E7B41A41AD1.

```
object = .object (scope:simple-template [,filter])
```

Les paramètres suivants sont disponibles :

Paramètre	Description
scope	Collection sur laquelle l'itération doit être effectuée, la macro va renvoyer le premier objet correspondant dans la collection Type : Template simple qui renvoie un objet ou une portée de collection
template-simple	Template à évaluer. Type : Template simple

Paramètre	Description
filter	Condition de filtre Type : condition

La macro suivante renvoie le premier attribut dans la collection définie sur le classificateur courant dont le code commence par une lettre comprise entre a et e (incluses).

```
.object(Attributes, (%.1:Code% >= a) and (%.1:Code% <= e))
```

Dans l'exemple suivant, le template `::myPackage2` est défini comme suit :

```
.object(ActiveModel.Packages, %Name% == MyPackage2)
```

et le template `OOM.Model::MyTemplate` est défini comme suit :

```
.foreach_item(myPackage2.Classes)
%Code%
.next(\n)
```

Dans `OOM.Model M = { OOM.Package MyPackage1, OOM.Package MyPackage2 { OOM.Class C1, OOM.Class C2} }`, le template `OOM.Model::MyTemplate` est évalué à :

```
C1
C2
```

Dans l'exemple suivant, ce template dans un MFI renvoie la première connexion d'accès aux données pour le processus associé à la publication courante :

```
.object(Process.DataConnections, %AccessType% == "RO")
```

Macro .replace

La macro .replace remplace toutes les occurrences d'une chaîne par une autre chaîne dans un bloc de texte.

Cette macro est particulièrement utile lorsque vous travaillez sur les conventions de dénomination.

Pour plus d'informations sur les conventions de dénomination, voir *Guide des fonctionnalités générales > L'interface de PowerAMC > Objets > Propriétés d'un objet > Conventions de dénomination*.

La macro .replace remplace l'ancienne chaîne <ancienne-chaîne> par la chaîne <nouvelle-chaîne> dans le bloc de texte <bloc-entrée>.

```
.replace (old-string,new-string)
    block-input
.endreplace
```

Les paramètres suivants sont disponibles :

Paramètre	Description
ancienne-chaîne	Chaîne à remplacer. Type : Texte
nouvelle-chaîne	Chaîne qui remplace *ancienne-chaîne*. Type : Texte
bloc-entrée	Un paramètre utilisé pour spécifier du texte. Type : Template complexe

Résultat

Le résultat est que toutes les occurrences de la chaîne ancienne-chaîne sont remplacées par des instances de la chaîne nouvelle-chaîne dans le bloc spécifié.

Dans l'exemple suivant, 'GetCustomerName' est converti en 'SetCustomerName'.

```
.replace( get , set )
GetCustomerName
.endreplace
```

Dans l'exemple suivant, la variable %Name% a pour valeur 'Customer Factory' et est convertie en 'Customer_Factory'.

```
.replace(" ", "_")
%Name%
.endreplace
```

Macro .set_interactive_mode

Cette macro permet de décider si l'exécution du GTL doit s'effectuer avec des interactions de l'utilisateur ou non.

```
.set_interactive_mode(mode)
```

Les modes suivants sont pris en charge :

- im_Batch - N'affiche aucune boîte de dialogue et utilise systématiquement les valeurs par défaut
- im_Dialog - Affiche des boîtes de dialogue d'information et de confirmation qui requièrent une action de l'utilisateur pour poursuivre l'exécution du script
- im_Abort - N'affiche jamais les boîtes de dialogue et abandonne l'exécution du script au lieu d'utiliser les valeurs par défaut à chaque fois qu'un dialogue s'impose

Vous pouvez utiliser cette macro lorsque votre modèle contient des raccourcis externes. Si le modèle cible d'un raccourci externe est fermé et que vous utilisez le mode im_Dialog, une boîte de dialogue s'affiche pour vous permettre d'ouvrir le modèle cible.

Macros .set_object et .set_value

Ces macros sont utilisées pour définir une variable locale de type objet (objet local) ou un type de valeur.

```
.set_object ( [portée.] nom [,ref-objet [,mode]])
```

La variable est une référence à l'objet spécifié à l'aide du second argument.

```
.set_value ( [portée.] nom, valeur [,mode])
```

La valeur de la variable est définie pour être la valeur du template converti spécifiée comme second argument.

Les paramètres suivants sont disponibles :

Paramètre	Description
portée	[facultatif] Portée qualifiante. Type : Template simple qui renvoie une portée d'objet soit de collection
nom	Nom de variable Type : Simple-template
ref-objet[.set_object uniquement]	[facultatif] Décrit une référence d'objet. S'il n'est pas spécifié ou s'il s'agit d'une chaîne vide, la variable est une référence à l'objet actif dans la portée de conversion courante Type : [*portée.*]*portée-objet*]
valeur[.set_value uniquement]	Valeur Type : Template simple (séquences d'échappement ignorées)
mode	[facultatif] Spécifie le mode de création. Vous pouvez choisir entre : • new - (Re)définit la variable dans la portée courante • update – [défaut] Si une variable du même nom existe déjà, celle-ci est modifiée, dans le cas contraire, une nouvelle variable est créée • newifundef - Définit la variable dans la portée courante si elle n'a pas été définie dans une portée externe, dans le cas contraire, rien ne se passe

Exemple :

```
.set_object(Attribute1, Attributes.First)
```

Exemple :

```
.set_value(FirstAttributeCode, %Attributes.First.Code%)
```

Remarque : Lorsque vous spécifiez une nouvelle variable, il est recommandé de spécifier 'new' comme troisième argument pour vous assurer qu'une nouvelle variable soit créée dans la portée courante.

Macro .unique

L'objet de la macro unique est de définir un bloc dans lequel l'unicité de chaque ligne du texte généré est garantie. Cette macro peut être utile pour calculer les importations, inclusions, typedefs ou de déclarations anticipées dans des langages tels que Java, C++ ou C#.

```
.unique
    bloc-entrée
.endunique[(fin)]
```

Le résultat est le bloc fourni en entrée dans lequel chaque ligne redondante a été supprimée.

Les paramètres suivants sont disponibles :

Paramètre	Description
bloc-entrée	Un paramètre utilisé pour spécifier du texte Type : Template complexe
fin	[facultatif] Ajouté au résultat, s'il y en a un Type : Texte

Exemple :

```
.unique
    import java.util.*;
    import java.lang.String;
    %imports%
.endunique
```

Macro .unset

Permet d'annuler la définition de variables locales et d'attributs volatiles définis à l'aide des macros .set_value et .set_object.

```
.unset([portée.]nom)
```

Les paramètres suivants sont disponibles :

Paramètre	Description
portée	[facultatif] Portée de qualification. Type : Template simple qui renvoie soit un objet soit une collection
nom	Nom de la variable locale ou de l'attribut volatile. Type : Template simple

Exemple :

```
.set_value(i, 1, new)
%i?%
```

```
.unset(i)
%i?%
```

La seconde ligne est vraie puisque la variable 'i' est définie tandis que la dernière ligne est fausse.

Macro .vbscript

La macro vbscript est utilisée pour incorporer du code VB script dans un template. Il s'agit d'une macro de bloc.

La syntaxe d'une macro vbscript est la suivante :

```
.vbscript [(liste-param-script)]
    bloc-entrée
.endvbscript [(fin)]
```

Le résultat de la valeur ScriptResultArray.

Les paramètres suivants sont disponibles :

Paramètre	Description
liste-param-script	Paramètres passés sur le script via le tableau ScriptInputArray. Type : Liste d'arguments de template-simple séparés par des virgules
bloc-entrée	Texte VB script Type : Texte
fin	Ajouté au résultat, s'il y en a un Type : Texte

Exemple :

```
.vbscript(hello, world)
ScriptResult = ScriptInputArray(0) + " " + ScriptInputArray(1)
.endvbscript
```

Le résultat est le suivant:

```
hello world
```

Remarque : l'objet actif de la portée de conversion courante est accessible via la collection ActiveSelection (voir *Propriétés globales* à la page 355) en tant que ActiveSelection.Item(0).

Chapitre 6

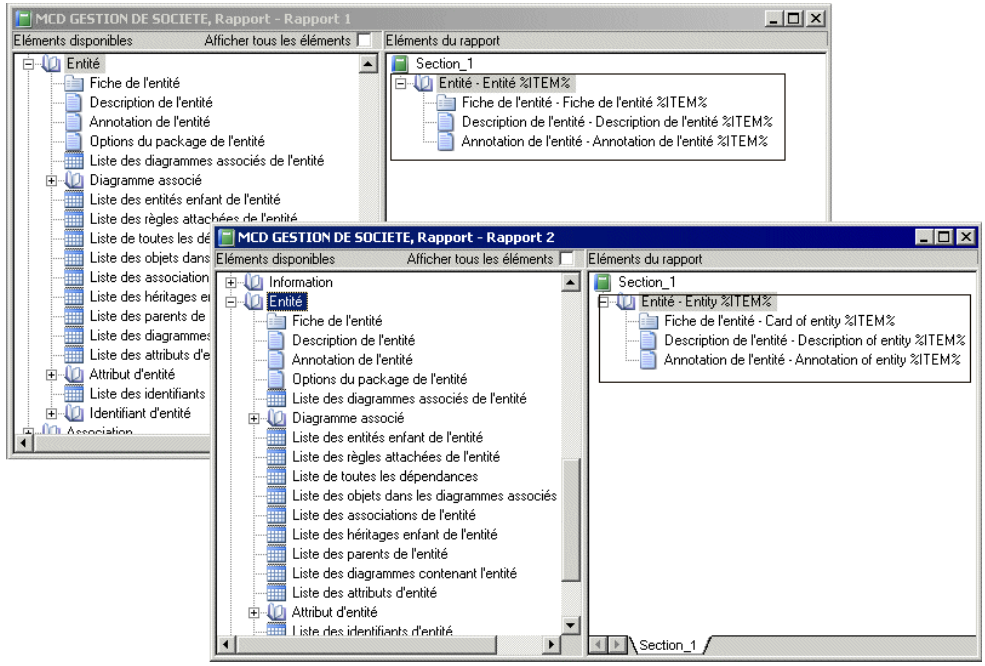
Traduction de rapports à l'aide des fichiers de ressource de langue de rapport

Un fichier de ressource de langue de rapport est un fichier au format XML enregistré avec une extension .XRL, qui contient tous les textes utilisés pour générer un rapport de modèle PowerAMC (par exemple, des titres de section de rapport, ou des noms d'objet de modèle et de leurs attributs (propriétés)) pour une langue particulière. Les fichiers de ressource de langue de rapport sont stockés dans le sous-répertoire Fichiers de ressources.

PowerAMC est livré avec une série de fichiers de ressource de langue de rapport en Français (langue par défaut), Anglais, Chinois simplifié et Chinois traditionnel. Vous pouvez éditer ces fichiers, ou les utiliser comme base pour créer vos propres fichiers .xrl afin de traduire les rapports dans d'autres langues.

Remarque : Lorsque vous créez un rapport, vous sélectionnez une langue de rapport pour afficher tous les textes imprimables traduits dans la langue donnée. Pour plus d'informations, voir *Guide des fonctionnalités générales > L'interface de PowerAMC > Rapports*.

Dans l'exemple suivant, Fiche de l'entité, Description de l'entité, et Annotation de l'entité sont affichés en Français et en Anglais, tels qu'ils seront affichés dans le volet Eléments de rapport :



Les fichiers de ressource de langue de rapport utilisent le langage de génération par template (GTL, Generation Template Language) PowerAMC afin de factoriser les traductions. Les templates d'éléments de rapport interagissent avec vos traductions des noms des objets de modèle et les variables linguistiques (qui gèrent les particularités syntaxiques telles que les formes plurielles et les articles définis) afin de générer automatiquement tous les éléments textuels d'un rapport.

Ce mécanisme, qui a été introduit avec la version 15 de PowerAMC, réduit de façon considérable (environ 60%) le nombre de chaîne qui doivent être traduites pour obtenir des rapports dans une nouvelle langue.

Par exemple, le titre de rapport Liste des relations de l'entité MonEntité est automatiquement généré comme suit :

- le template d'élément de rapport List - object collections (voir *Catégorie Report Titles* à la page 344) est traduit comme suit :

```
Liste des %@Value% %ParentMetaClass.OFTHECLSSNAME% %%PARENT%%
```

dans lequel les variables sont résolues comme suit :

- %@Value% - est remplacée par le type d'objet de la métaclasse (voir *Catégorie Object Attributes* à la page 341). Dans le cas présent, relations.
- %ParentMetaClass.OFTHECLSSNAME% %%PARENT%% - est remplacée par le type d'objet de la métaclasse parent, comme généré par la variable linguistique

OFTHECLSSNAME (voir *Catégorie Profile/Linguistic Variables* à la page 342). Dans le cas présent, l'entité.

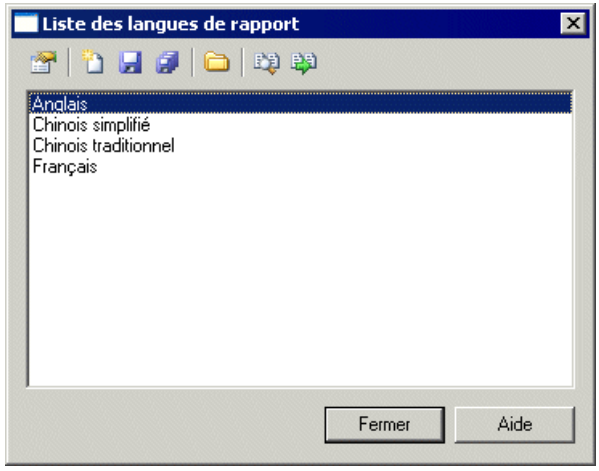
- %%PARENT%% - est remplacée par le nom de l'objet spécifique (voir *Catégorie Object Attributes* à la page 341). Dans le cas présent, MonEntité.

Pour plus d'informations sur les templates, voir *Chapitre 5, Personnalisation de la génération à l'aide du langage de génération par template* à la page 285.

Ouverture d'un fichier de ressource de langue de rapport

Vous pouvez affiche et éditer le contenu d'un fichier de ressource de langue de rapport dans l'Editeur de ressources.

1. Sélectionnez **Outils > Ressources > Langues de rapport** afin d'afficher la boîte de dialogue Liste des langues de rapport, qui affiche la liste des fichiers .xrl disponibles :



2. Sélectionnez une langue de rapport, puis cliquez sur l'outil Propriétés pour l'afficher dans l'Editeur de ressources.

Remarque : Vous pouvez ouvrir le fichier .xrl associé à un rapport ouvert dans l'Editeur de rapport en sélectionnant **Rapport > Propriétés** de rapport, puis en cliquant sur l'outil Editer le langage courant en regard de la liste des langues. Vous pouvez changer la langue de rapport en sélectionnant une autre langue dans la liste.

Pour plus d'informations sur les outils disponibles dans la boîte de dialogue Liste des langues de rapport, voir *Chapitre 1, Utilisation des fichiers de ressources PowerAMC* à la page 1.

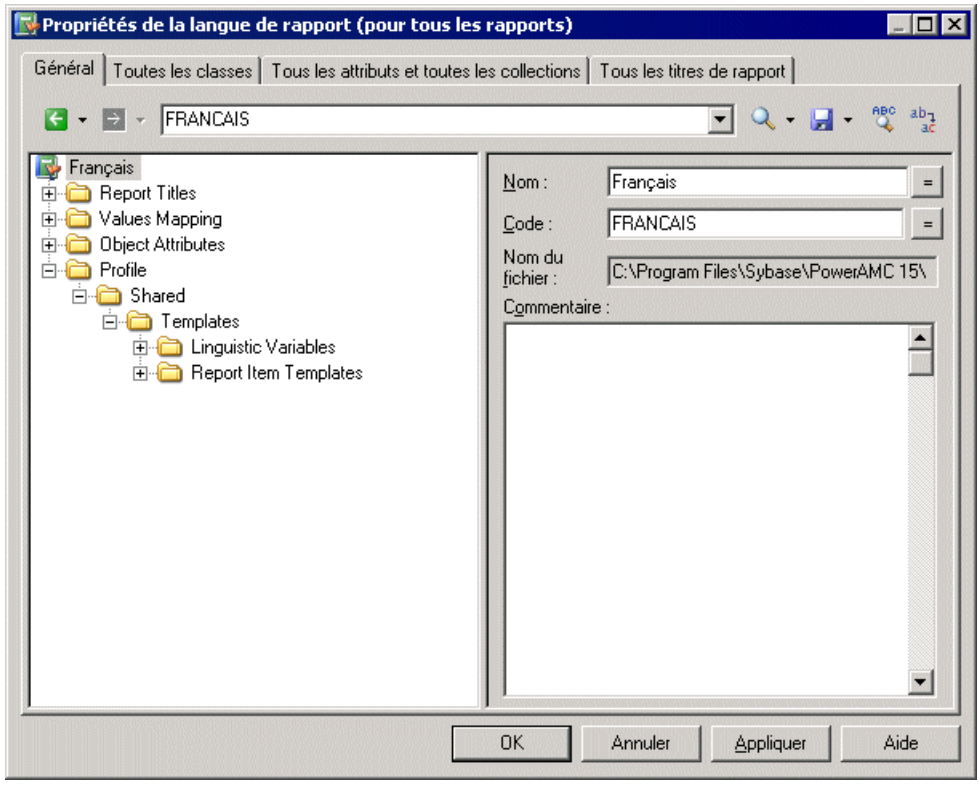
Création d'un fichier de ressource de langue de rapport pour une nouvelle langue

Vous pouvez traduire les titres de rapport et autres blocs de texte utilisés pour générer des rapports PowerAMC dans un nouveau langage.

1. Sélectionnez **Outils > Ressources > Langues de rapport** pour afficher la boîte de dialogue Liste des langues de rapport, qui affiche tous les fichiers de ressources de langue de rapport disponibles.
2. Cliquez sur l'outil Nouveau pour afficher la boîte de dialogue Nouvelle langue de rapport, et saisissez le nom que vous souhaitez voire affiché dans la boîte de dialogue Liste des langues de rapport.
3. [facultatif] Sélectionnez une langue de rapport dans la liste Copier depuis.
4. Cliquez sur OK pour afficher le contenu du nouveau fichier dans l'Editeur de langue de rapport.
5. Ouvrez la catégorie Values Mapping, puis traduisez chacune des valeurs de mot clé. Pour plus d'informations, voir *Catégorie Values Mapping* à la page 334.
6. Ouvrez la catégorie **Profile > Linguistic Variables** afin de créer les règles de grammaire nécessaires à l'évaluation correcte de templates d'élément de rapport. Pour plus d'informations, voir *Catégorie Profile/Linguistic Variables* à la page 342.
7. Ouvrez la catégorie **Profile > Report Items Templates**, et traduisez les différents templates. Pour plus d'informations, voir *Catégorie Profile/Report Item Templates* à la page 344. Lors de la traduction, vous pouvez découvrir d'autres variables linguistiques à créer (voir l'étape précédente).
8. Cliquez sur l'onglet Toutes les classes afin d'afficher une liste triable de toutes les métaclasses disponibles dans le métamodèle PowerAMC. Traduisez chaque nom de métaclasse. Pour plus d'informations, voir *Onglet Toutes les classe* à la page 346.
9. Cliquez sur l'onglet Tous les attributs et toutes les collections afin d'afficher une liste triable de tous les attributs et toutes les dimensions disponibles dans le métamodèle PowerAMC. Traduisez chaque nom d'attribut et de collection. Pour plus d'informations, voir *Onglet Tous les attributs et toutes les collections* à la page 347.
10. Cliquez sur l'onglet Tous les titres de rapport, puis passez en revue les titres de rapport générés. Pour plus d'informations, voir *Onglet Tous les titres de rapport* à la page 348. Notez que l'affichage de cet onglet peut prendre plusieurs secondes.
11. Cliquez sur l'outil Enregistrer, puis cliquez sur OK pour fermer l'Editeur de langue de rapport. Le fichier de ressource de langue de rapport peut maintenant être associé à un rapport.

Propriétés d'un fichier de ressource de langue de rapport

Tous les fichiers de ressource de langue de rapport peuvent être ouverts dans l'Editeur de ressources, et ils ont la même structure de base :



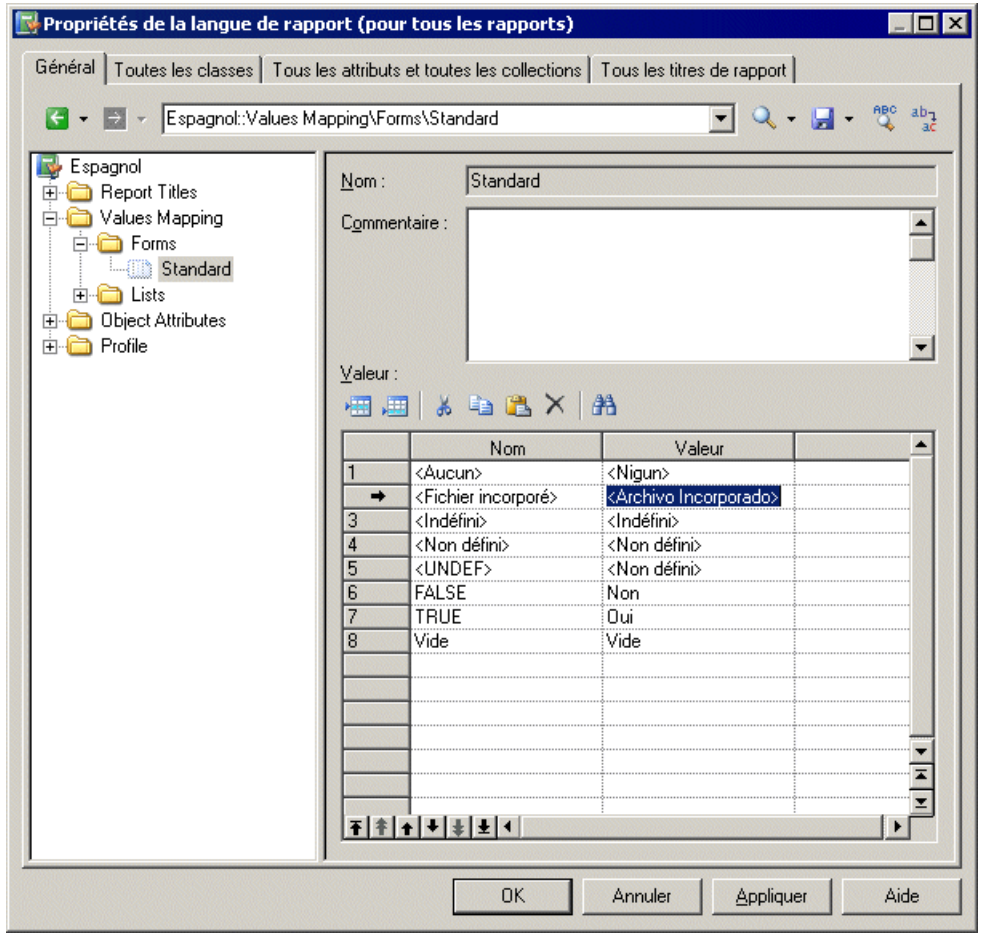
Pour plus d'informations sur l'Editeur de ressources, voir *Ouverture de fichiers de ressources dans l'Editeur de ressources* à la page 3.

Le noeud racine de chaque fichier contient les propriétés suivantes :

Propriété	Description
Nom	Spécifie le nom de la langue de rapport.
Code	Spécifie le code de la langue de rapport.
Nom du fichier	[lecture seule] Spécifie le chemin d'accès au fichier .xrl.
Commentaire	Spécifie une information supplémentaire relative à la langue de rapport.

Catégorie Values Mapping

La catégorie Values Mapping contient une liste de valeurs de mots clé (telles que Indéfini, Oui, Non, Faux ou Aucun) pour les propriétés d'objet affichées dans des fiches, contrôles et listes. Vous devez saisir une traduction dans la colonne Valeur pour chaque mot clé dans la colonne Nom :



Cette catégorie contient les sous-catégories suivantes :

Sous-catégorie	Description
Forms	Contient une table de correspondances Standard pour les mots clés des propriétés d'objet dans des fiches et contrôles, qui sont disponibles dans tous les modèles. Vous devez spécifier des traductions pour les valeurs de mots clé dans la colonne Valeur. Exemple : Embedded Files.
Lists	Contient une table de correspondances Standard pour les mots clés des propriétés d'objet dans des listes, qui sont disponibles dans tous les modèles. Vous devez spécifier des traductions pour les valeurs de mots clé dans la colonne Valeur. Exemple : True.

Vous pouvez créer de nouvelles tables de correspondances contenant des valeurs de mots clé spécifiques à des types d'objets de modèle particuliers.

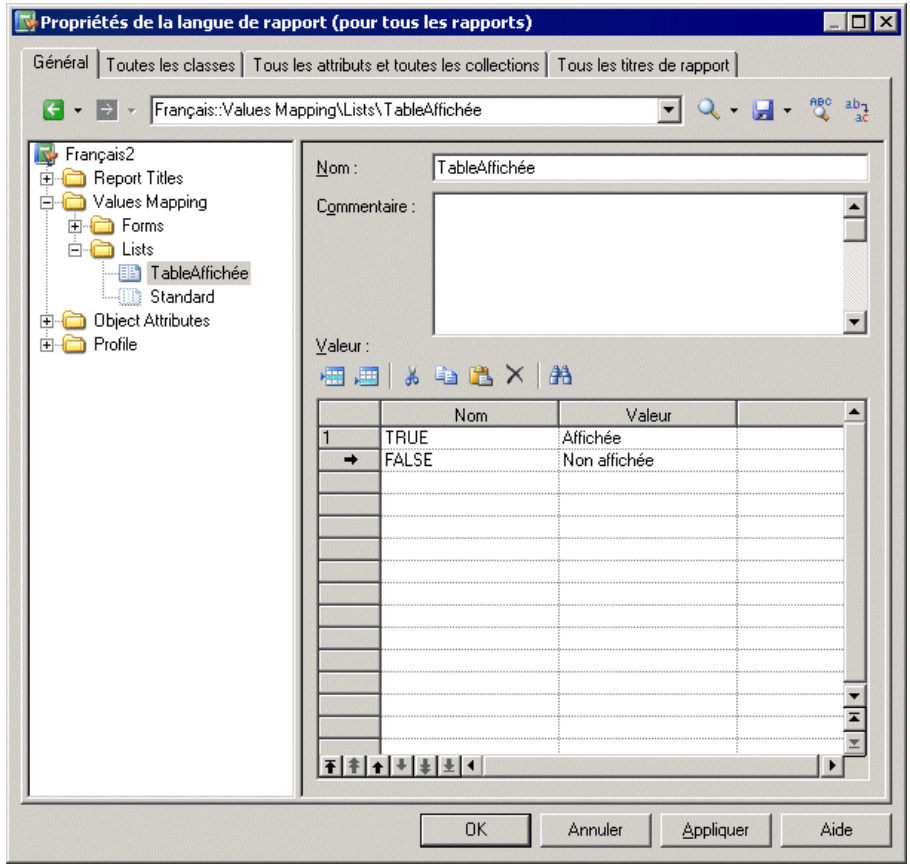
Exemple : Création d'une table de correspondances, et association de cette table à un objet de modèle particulier

Vous pouvez supplanter les valeurs contenues dans les tables de correspondance Standard pour un objet de modèle particulier en créant une nouvelle table de correspondances, et en l'associant à l'objet.

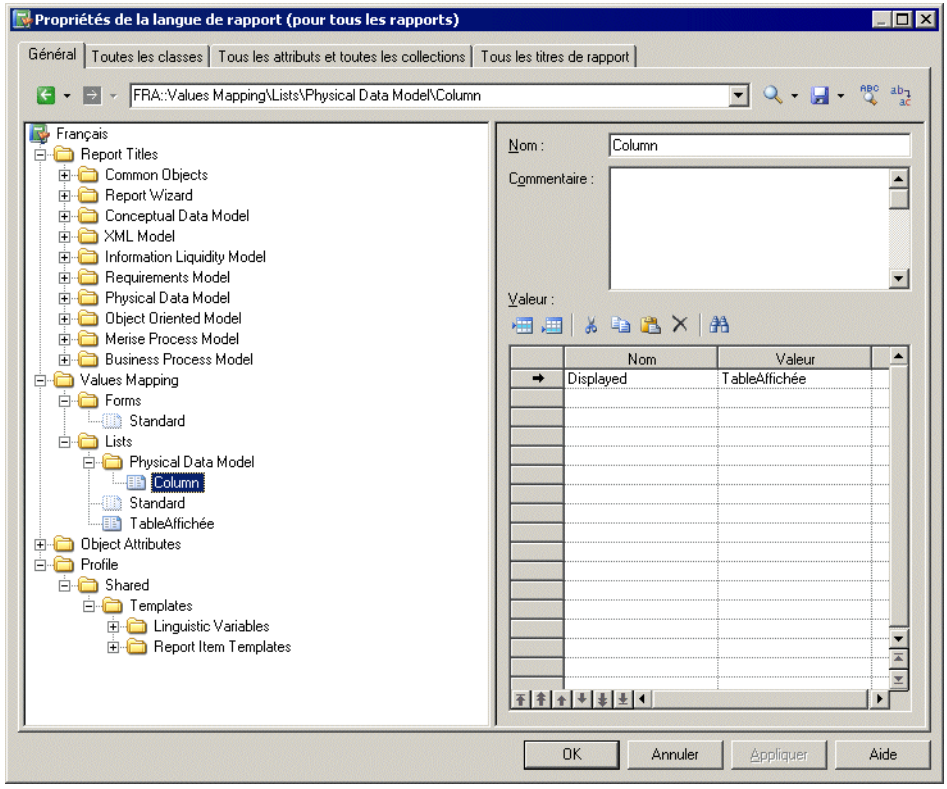
Dans l'exemple suivant, la table de correspondances TableAffichée est utilisée pour remplacer la table de correspondances Standard pour les colonnes de MPD afin de fournir des valeurs personnalisées pour la propriété Affichée, qui contrôle l'affichage de la colonne sélectionnée dans le symbole de table. Cette situation peut être résumée comme suit :

Nom	Valeur
TRUE	Affichée
FALSE	Non affichée

1. Ouvrez la valeur catégorie **Values Mapping > Lists**.
2. Pointez sur la catégorie Lists, cliquez le bouton droit de la souris, puis sélectionnez **Nouvel élément > Table de correspondance** afin de créer une nouvelle liste, et d'afficher sa feuille de propriétés.
3. Saisissez TableAffichée dans la zone Nom, puis saisissez les valeurs suivantes dans la liste Valeur, et appuyez sur Appliquer :
   - Nom : TRUE, Valeur : Affiché.
   - Nom :FALSE, Valeur : Non affiché.



4. Pointez sur la catégorie Lists, cliquez le bouton droit de la souris, puis sélectionnez **Nouvel élément > Categorie**, nommez la catégorie Physical Data Model, puis cliquez sur Appliquer.
5. Pour compléter la recréation de l'arborescence des attributs d'objet de MPD, pointez sur la nouvelle catégorie Physical Data Model, cliquez le bouton droit de la souris, puis sélectionnez **Nouvel élément > Table de correspondance**, nommez la catégorie Column, puis cliquez sur Appliquer.
6. Cliquez sur la colonne Nom pour créer une valeur et saisir Displayed, qui est le nom public de l'attribut de colonne de MPD (propriété).
7. Cliquez dans la colonne Value et saisissez TableAffichée pour spécifier la table de correspondances à utiliser pour cet attribut.



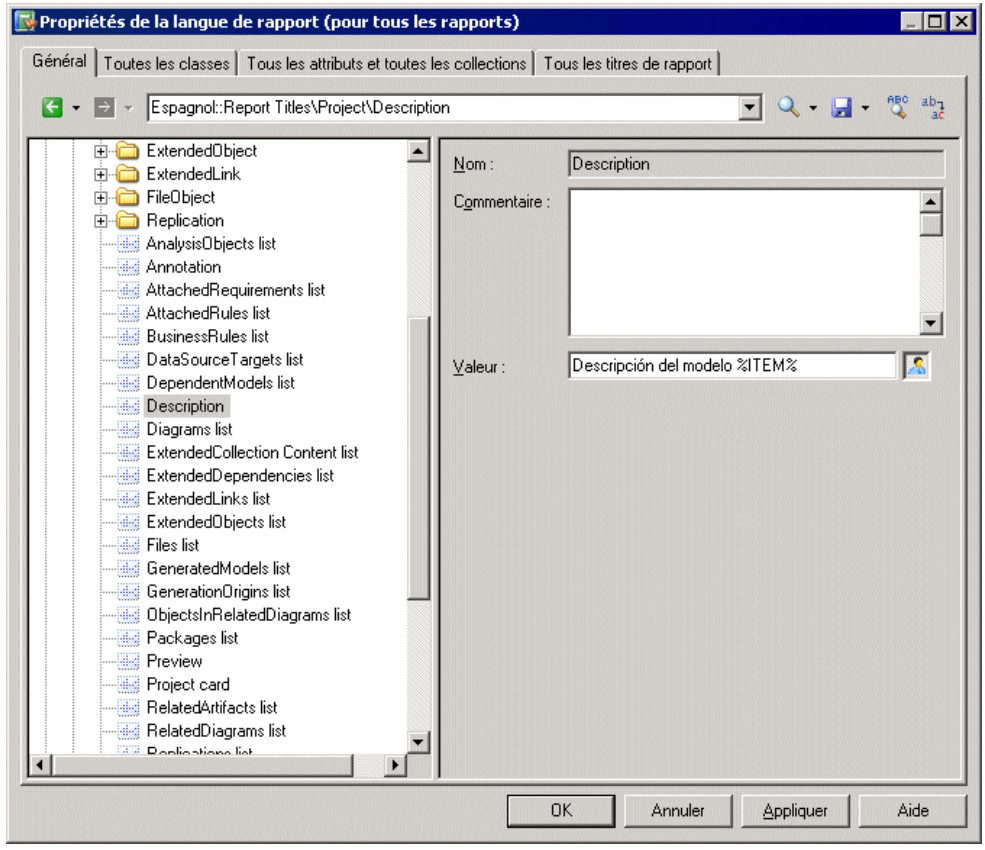
8. Cliquez sur Appliquer pour enregistrer vos modifications. Lorsque vous générez un rapport, la propriété Affichée sera affichée avec les valeurs spécifiées :

1 Liste des colonnes de table

Nom	*Code*	*Affiché*
Matricule	MATRICULE	Affichée
Nom	NOM	Affichée
Prénom	PRENOM	Affichée
Adresse	ADRESSE	Non affichée

Catégorie Report Titles

La catégorie Report Titles contient des traductions pour tous les titres de rapport possibles qui s'affichent dans le volet Eléments disponibles de l'Editeur de ressources, ceux qui sont générés avec l'Assistant Rapport, ainsi que différents textes.



Cette catégorie contient les sous-catégories suivantes :

Sous-catégorie	Description
Common Objects	Contient les textes disponibles pour tous les modèles. Vous devez fournir les traductions de ces textes ici. Exemple : HTMLNext fournit le texte pour le bouton Suivant dans un rapport HTML.

Sous-catégorie	Description
Report Wizard	Contient les titres de rapport générés à l'aide de l'Assistant Rapport. Vous devez fournir les traductions de ces textes ici. Exemple : Short description title fournit le texte pour une brève description lorsque vous générez un rapport à l'aide de l'Assistant Rapport.
[Modèles]	Contient les titres de rapport et autres textes disponibles pour chaque modèle. Ils sont automatiquement générés, mais vous pouvez passer outre leurs valeurs par défaut. Exemple : la liste DataTransformationTasks fournit le texte pour les tâches de transformation de données d'un processus de transformation donné dans le Modèle de Fluidité de l'Information.

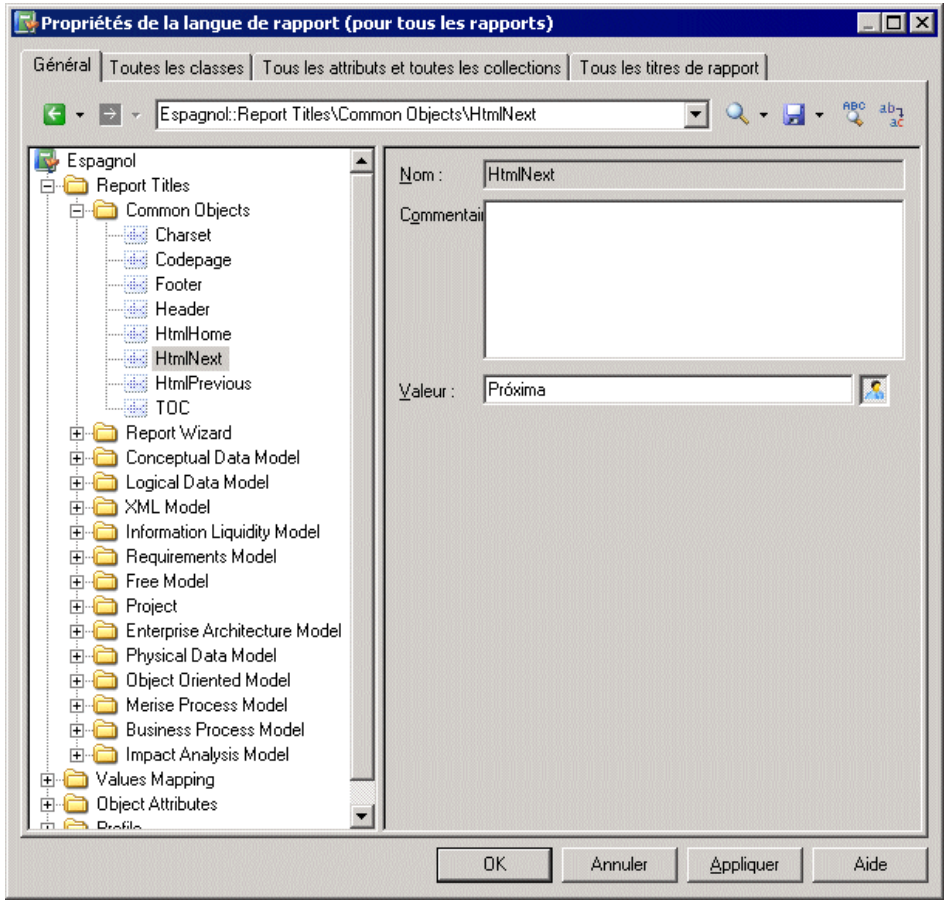
Par défaut (à l'exception des sous-catégories Common Objects et Report Wizard) ces traductions sont automatiquement générées dans les templates de la catégorie Profile (Voir *Catégorie Profile/Report Item Templates* à la page 344). Vous pouvez passer outre les valeurs générées automatiquement en saisissant votre propre texte dans la zone Valeur. Le bouton Défini par l'utilisateur est automatiquement enfoncé pour indiquer que la valeur n'est pas une valeur générée.

Remarque : L'onglet Tous les titres de rapport (voir *Onglet Tous les titres de rapport* à la page 348) affiche les mêmes traductions que celles présentes dans cette catégorie au sein d'une liste simple et triable. Cet onglet peut s'avérer plus pratique pour vérifier, et le cas échéant modifier, les traductions générées.

Exemple : Traduction du bouton Précédent d'un rapport HTML

Le bouton Précédent d'un rapport HTML est un objet commun disponible dans tous les modèles, et situé dans la catégorie Common Objects. Vous devez traduire ce texte manuellement avec les autres éléments dans cette catégorie, ainsi que dans la catégorie Report Wizard.

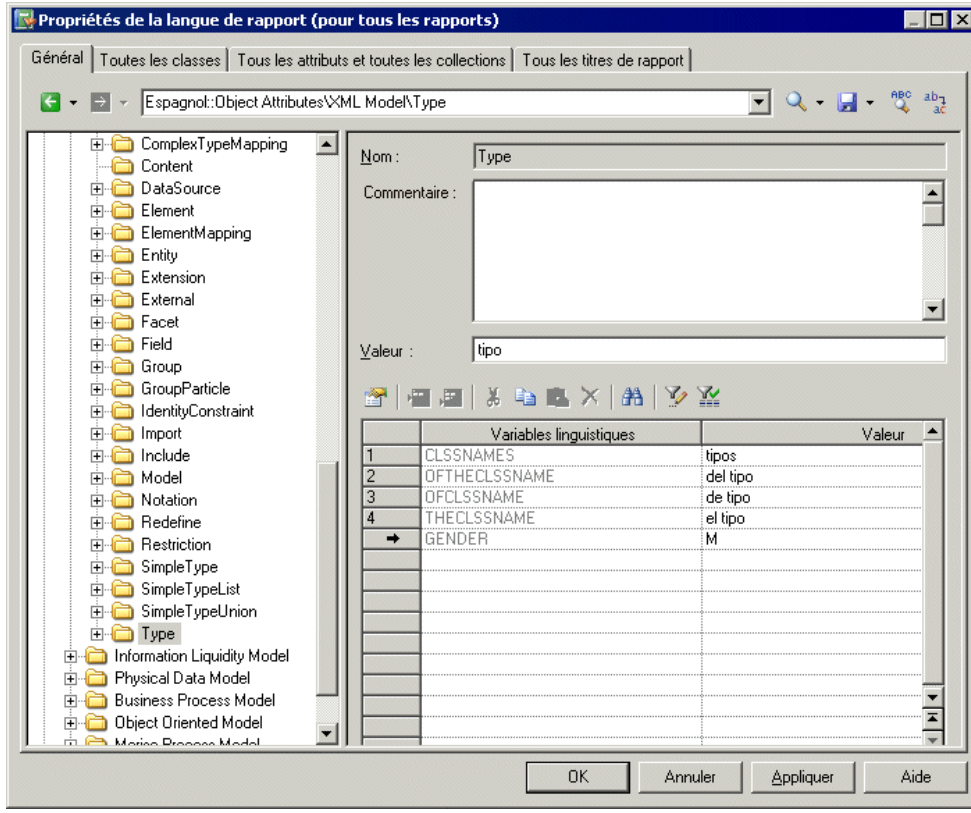
1. Ouvrez la catégorie **Report Titles** > **Common Objects**.
2. Cliquez sur l'entrée HtmlNext afin d'afficher ses propriétés, puis saisissez une traduction dans la zone Valeur. Le bouton Défini par l'utilisateur est automatiquement enfoncé pour indiquer qu'il ne s'agit pas d'une valeur générée.



3. Cliquez sur Appliquer pour enregistrer vos modifications.

Catégorie Object Attributes

La catégorie Object Attributes contient toutes les métaclasses, collections et attributs disponibles dans le métamodèle PowerAMC, organisés en arborescence :



Cette catégorie contient les sous-catégories suivantes :

Sous-catégorie	Description
[Modèles]	Contient les textes pour les métaclasses, collections et attributs disponibles pour chaque type de modèle, pour lesquels vous devez fournir des traductions. Exemple : Action fournit le texte pour l'attribut d'un processus dans le Modèle de Processus Métiers (MPM).

Sous-catégorie	Description
Common Objects	Contient les textes pour les métaclasses, collections et attributs disponibles pour tous les types de modèles, pour lesquels vous devez fournir des traductions. Exemple : Diagram fournit le texte pour un diagramme dans n'importe quel type de modèle.

Pour chaque élément dont le nom est donné, vous devez fournir une traduction dans la zone Nom. Cette valeur est extraite par les templates que vous avez spécifiés dans la catégorie Profile pour générer des titres de rapport par défaut (voir *Catégorie Report Titles* à la page 338).

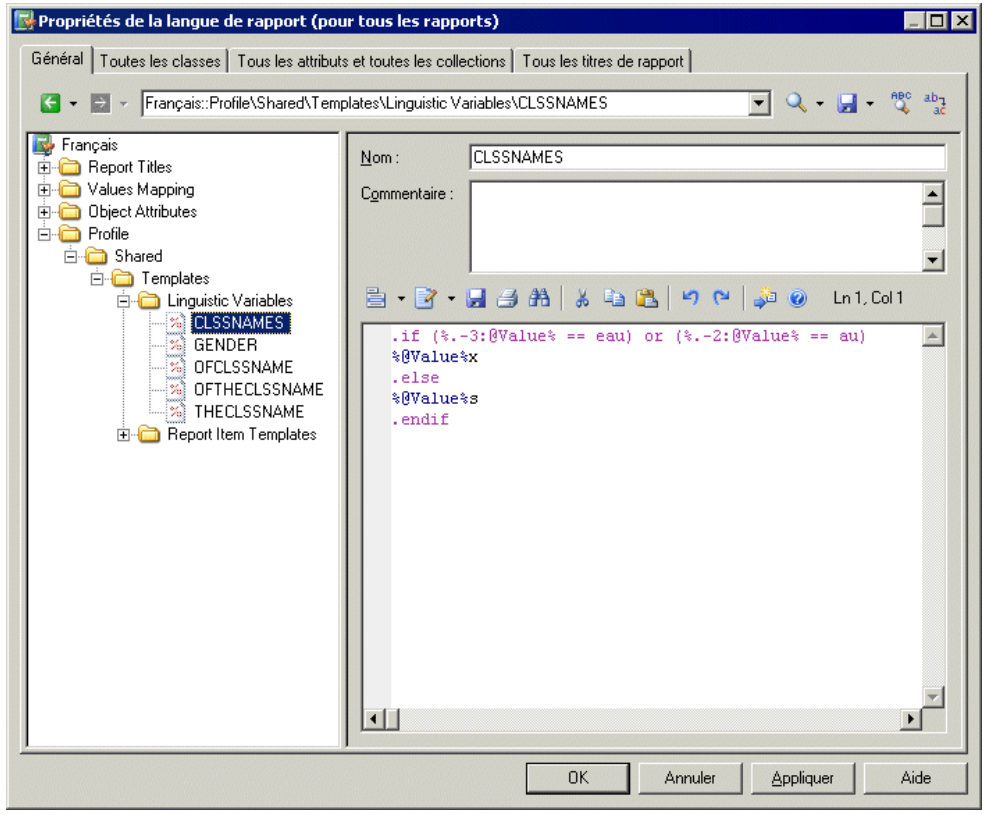
En ce qui concerne les métaclasses uniquement, les variables linguistiques que vous avez spécifiées (voir *Catégorie Profile/Linguistic Variables* à la page 342) sont répertoriées avec le résultat de leur application dans les traductions spécifiées dans la zone Valeur. Vous pouvez passer outre les valeurs générées automatiquement en saisissant votre propre texte dans la zone Valeur. Le bouton Défini par l'utilisateur est automatiquement enfoncé pour indiqué que la valeur n'est pas une valeur générée.

Remarque : Ces onglets affichent les mêmes traductions que celles affichées dans la catégorie Object Attributes au sein d'une liste simple et triable. Il peut s'avérer plus pratique de saisir les traductions dans ces onglets (voir *Onglet Toutes les classe* à la page 346 et *Onglet Tous les attributs et toutes les collections* à la page 347).

Catégorie Profile/Linguistic Variables

La catégorie Linguistic Variables contient des templates qui spécifient des règles de grammaire afin d'aider à construire les templates d'élément de rappor.

Vous pouvez utiliser cette catégorie pour définir par exemple les formes plurielles d'un nom, ainsi que l'article défini censé le précéder. Pour plus d'informations, voir *Catégorie Profile/Report Item Templates* à la page 344.



Le fait de spécifier les règles de grammaire appropriées pour votre langue et de les insérer dans vos templates d'élément de rapport améliore considérablement la génération de vos titres de rapport. Vous pouvez créer autant de variables que requis par votre langue.

Chaque variable linguistique et le résultat de son évaluation sont affichés pour chaque métaclasse dans la catégorie Object Attributes (voir *Catégorie Object Attributes* à la page 341).

Les exemples suivants montrent l'utilisation de règles de grammaire spécifiées sous forme de variables linguistiques afin de renseigner les templates d'éléments de rapport dans le fichier de ressource de langue de rapport Français :

- GENDER – Identifie comme féminin un nom de métaclasse %Value%, s'il se termine par "e" et comme masculin dans les autres cas :

```
.if (%.-1:@Value% == e)
F
.else
M
.endif
```

Par exemple : la table, la colonne, le trigger.

- CLSSNAMES – Crée un pluriel en ajoutant "x" à la fin du nom de métaclasse %Value%, s'il se termine par "eau" ou "au" et ajoute "s" dans les autres cas :

```
.if (%.-3:@Value% == eau) or (%.-2:@Value% == au)
%@Value%x
.else
%@Value%s
.endif
```

Par exemple : les tableaux, les tables, les entités.

- THECLSSNAME – Insère l'article défini avant le nom de la métaclasse %Value% en insérant " l' ", si ce nom commence par une voyelle, "le" s'il est masculin, et "la" dans le cas contraire :

```
.if (%.1U:@Value% == A) or (%.1U:@Value% == E) or (%.1U:@Value% == I)
or (%.1U:@Value% == O) or (%.1U:@Value% == U)
l'%@Value%
.elsif (%GENDER% == M)
le %@Value%
.else
la %@Value%
.endif
```

Par exemple : l'association, le package, la table.

- OFTHECLSSNAME – Insère la préposition "de" plus l'article défini avant le nom de la métaclasse %Value%, s'il commence par une voyelle ou s'il est féminin, dans le cas contraire insère "du".

```
.if (%.1U:@Value% == A) or (%.1U:@Value% == E) or (%.1U:@Value% == I)
or (%.1U:@Value% == O) or (%.1U:@Value% == U) or (%GENDER% == F)
de %THECLSSNAME%
.else
du %@Value%
.endif
```

Par exemple : de la table, du package.

- OFCLSSNAME – Insère la préposition " d' " avant le nom de métaclasse %Value%, s'il commence par une voyelle, et "de" dans le cas contraire.

```
.if (%.1U:@Value% == A) or (%.1U:@Value% == E) or (%.1U:@Value% == I)
or (%.1U:@Value% == O) or (%.1U:@Value% == U)
d'%@Value%
.else
de %@Value%
.endif
```

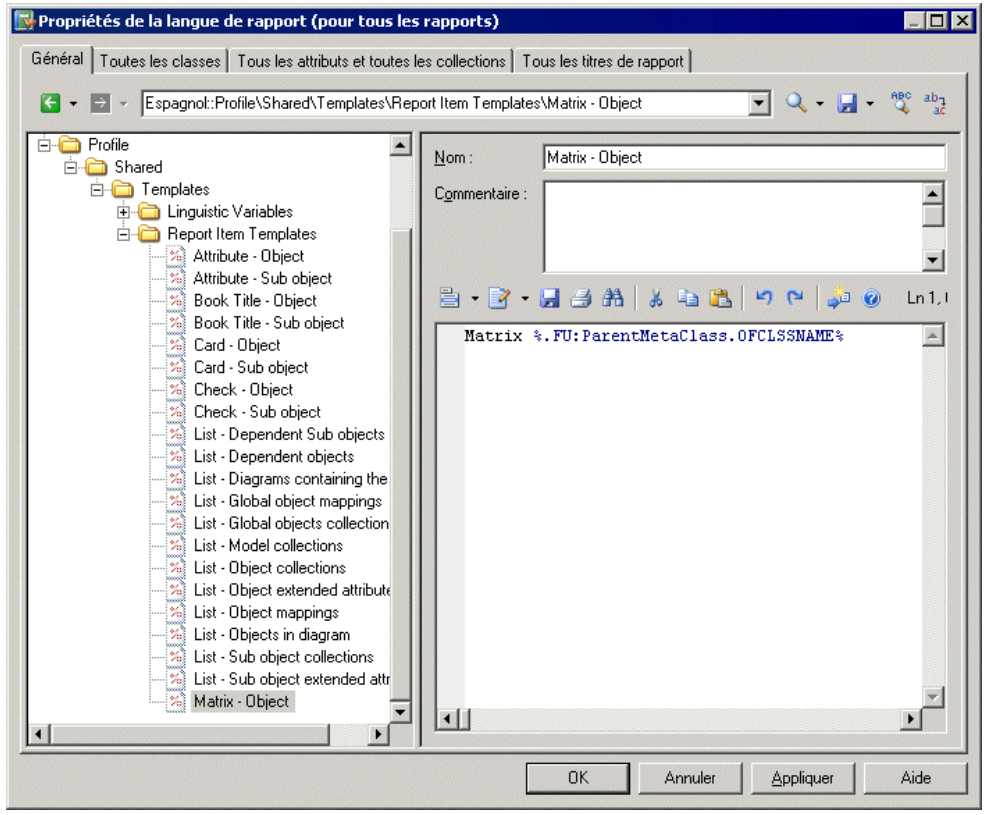
Par exemple : d'association, de table.

Catégorie Profile/Report Item Templates

La catégorie Report Item Templates contient un jeu de templates qui, en conjonction avec les traductions que vous allez fournir pour les noms de métaclasse, attribut et collections, sont

évalués pour générer automatiquement tous les titres de rapport possibles pour les éléments de rapport (livre, liste, fiche, etc.).

Pour plus d'informations, voir *Catégorie Object Attributes* à la page 341.



Vous devez fournir des traductions pour chaque template en saisissant votre propre texte. Les variables (telles que %text%) ne doivent pas être traduites.

Par exemple, la syntaxe du template pour la liste des sous-objets contenus dans une collection appartenant à un objet se présente comme suit :

```
List of %@Value% of the %ParentMetaClass.@Value% %%PARENT%%
```

Lorsque ce template est évalué, la variable %@Value% est remplacée par la valeur spécifiée dans la zone Valeur pour l'objet, %ParentMetaClass.@Value% est remplacé par la valeur spécifiée dans la zone Valeur du parent de l'objet, et %%PARENT%% est remplacé par le nom du parent de l'objet.

Dans cet exemple, vous traduisez ce template comme suit :

- Traduisez les éléments non-variable dans le template.

- Créez une variable linguistique OFTHECLSSNAME afin de spécifier la règle de grammaire utilisée dans le template (voir *Catégorie Profile/Linguistic Variables* à la page 342).

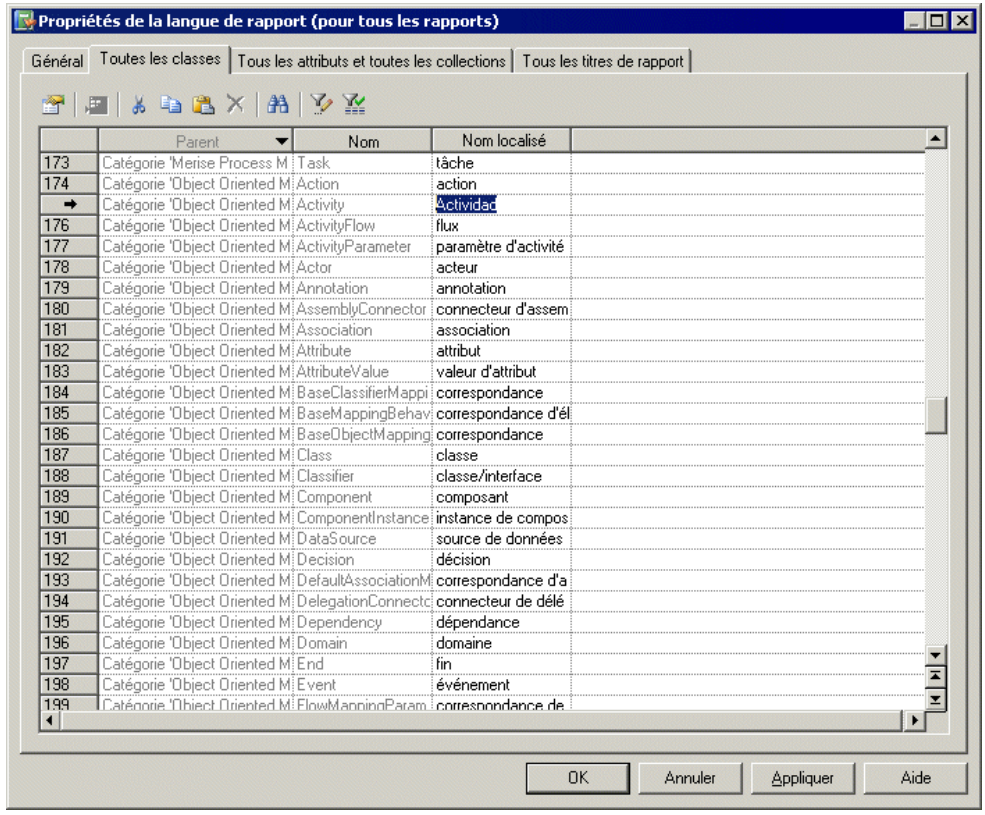
Ce template sera réutilisé pour créer des titres de rapport pour toutes les listes de sous-objets contenues dans une collection appartenant à un objet.

Vous ne pouvez pas supprimer des templates ou en créer de nouveaux.

Onglet Toutes les classes

L'onglet Toutes les classes répertorie toutes les métaclasses disponibles dans la catégorie Object Attributes disponibles dans la catégorie Object Attributes de l'onglet Général, mais sa présentation sous forme de tableau rend son utilisation plus simple.

Pour plus d'informations, voir *Catégorie Object Attributes* à la page 341.

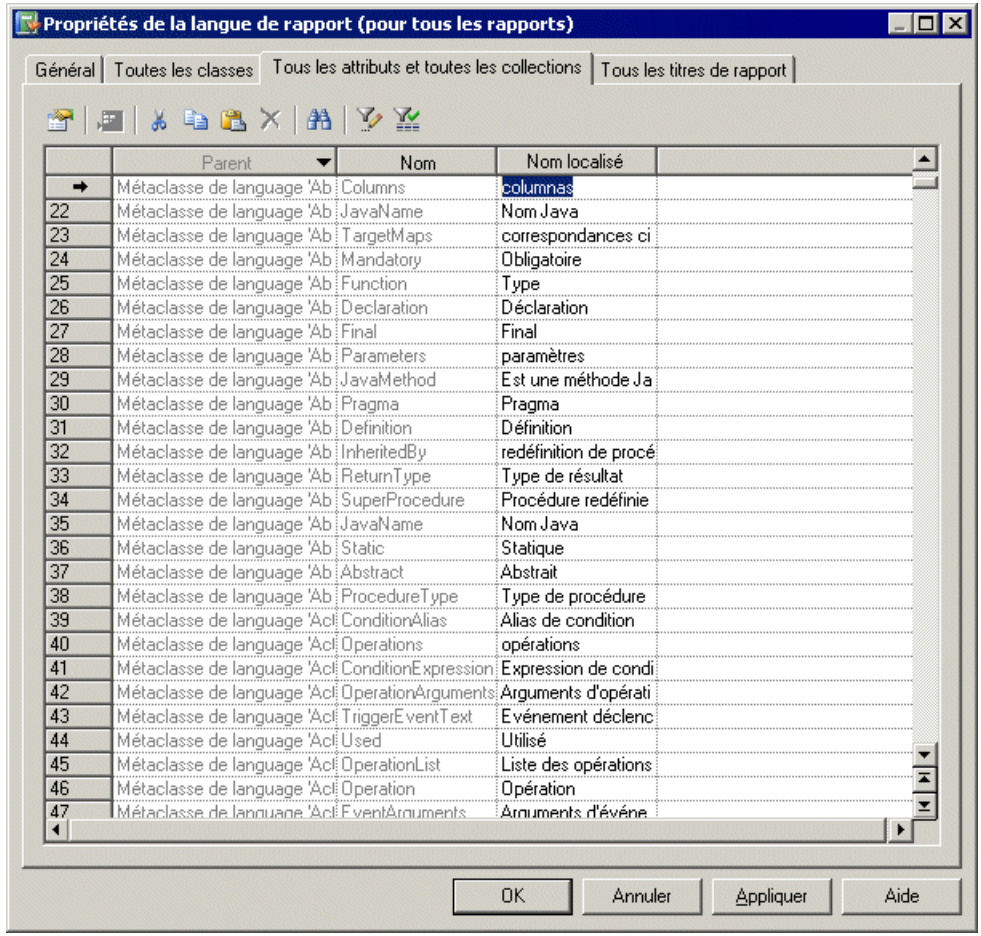


Pour chaque métaclasse répertoriée dans la colonne Nom, vous devez saisir une traduction dans la colonne Valeur. Vous pouvez trier la liste pour regrouper des objets par nom, et traiter les éléments identiques simultanément en sélectionnant plusieurs lignes.

Onglet Tous les attributs et toutes les collections

L'onglet Tous les attributs et toutes les collections répertorie toutes les collections et tous les attributs disponibles dans la catégorie Object Attributes de l'onglet Général, mais sa présentation sous forme de tableau rend son utilisation plus simple.

Pour plus d'informations, voir *Catégorie Object Attributes* à la page 341.

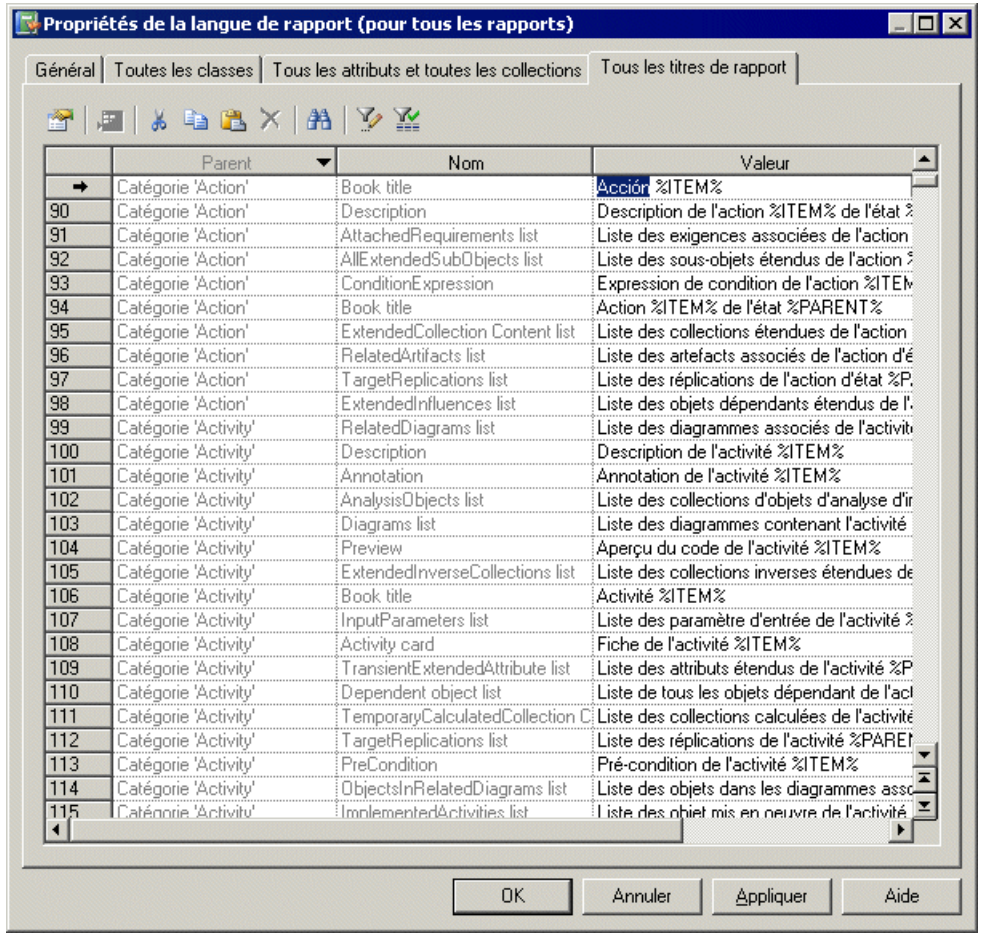


Pour chaque attribut ou chaque collection répertorié dans la colonne Nom, vous devez saisir une traduction dans la colonne Valeur. Vous pouvez trier la liste pour regrouper des objets par nom, et traiter les éléments identiques simultanément en sélectionnant plusieurs lignes.

Onglet Tous les titres de rapport

L'onglet Tous les titres de rapport répertorie tous les titres de rapport et autres textes disponibles dans la catégorie Report Titles de l'onglet Général, mais sa présentation sous forme de tableau rend son utilisation plus simple

Pour plus d'informations, voir *Catégorie Object Attributes* à la page 341.



Pour chaque rapport répertorié dans la colonne Nom, vous pouvez consulter ou modifier une traduction dans la colonne Valeur. Vous pouvez trier la liste pour regrouper des objets par nom, et traiter les éléments identiques simultanément en sélectionnant plusieurs lignes.

Chapitre 7

Pilotage de PowerAMC à l'aide de scripts

Lorsque vous manipulez des modèles de grande taille ou plusieurs modèles à la fois, il peut être fastidieux d'effectuer des tâches répétitives, telles que modifier des objets à l'aide de règles globales, importer ou générer des nouveaux formats ou encore vérifier des modèles.

Ces opérations peuvent être simplifiées à l'aide de scripts. Le scripting est largement utilisé dans différentes fonctionnalités PowerAMC. Par exemple, lorsque vous souhaitez :

- Créer des vérifications personnalisées, des gestionnaires d'événement, des transformations, des commandes et menus contextuels personnalisés (voir *Chapitre 2, Fichiers d'extension* à la page 23)
- Communiquer avec PowerAMC depuis une autre application (voir *Communication avec PowerAMC à l'aide de OLE Automation* à la page 409)
- Personnaliser les menus PowerAMC en ajoutant vos propres éléments de menu (voir *Personnalisation des menus PowerAMC à l'aide de compléments* à la page 414).
- Créer des macros VBscript et incorporer du code VBscript dans un template pour la génération (voir *Guide de référence des macros du langage de génération par template* à la page 307).

Vous pouvez accéder aux objets PowerAMC en utilisant un langage de script tel que Java, VBScript ou C#. Toutefois, le langage de script utilisé pour les exemples de ce chapitre est VBScript.

VBScript est un langage de script développé par Microsoft. PowerAMC fournit un support intégré pour le langage VBScript de Microsoft qui vous permet d'écrire et de lancer des scripts pour agir sur les objets du métamodèle de PowerAMC à l'aide de *propriétés* et de *méthodes*. Chacun des objets de PowerAMC peut être lu et modifié (création, modification ou suppression).

Accès aux objets du métamodèle PowerAMC

PowerAMC est livré avec un métamodèle publié sous la forme d'un Modèle Orienté-Objet (metamodel.moo) qui illustre la manière dont les métadonnées interagissent au sein du logiciel. Tous les objets dans le métamodèle de PowerAMC possèdent un nom et un code qui constituent le *nom public* des métadonnées. Un fichier d'aide au format HTML est également fourni pour vous permettre de retrouver les propriétés et les méthodes qui peuvent être utilisées pour accéder à chacun des objets de PowerAMC.

Pour plus d'informations sur les métadonnées, voir *Chapitre 1, Utilisation des fichiers de ressources PowerAMC* à la page 1.

PowerAMC fournit également des exemples de script que vous pouvez utiliser comme base pour créer vos propres scripts.

Le scripting vous permet d'effectuer n'importe quel type de manipulation de données mais vous pouvez également insérer et personnaliser des commandes dans le menu Outils qui vous permettront de lancer automatiquement vos propres scripts.

Objets

Les *objets* font référence à tous les types d'objets de PowerAMC. Il peut s'agir de :

- Objets de conception, tels que des tables, classes, processus ou colonnes.
- Diagrammes ou symboles.
- Objets fonctionnels, tels que le rapport ou le référentiel.

Un objet appartient à une métaclasse du métamodèle PowerAMC.

Chaque objet possède des propriétés, des collections et des méthodes qu'il hérite de sa métaclasse.

Les objets racine, tels que les modèles par exemples, sont créés ou récupérés en utilisant des méthodes globales. Pour plus d'informations, voir *Propriétés globales* à la page 355.

Les objets autres que les objets racine sont créés ou extraits à l'aide de collections. Par exemple, vous créez ces objets en utilisant une méthode Create sur des collections et les supprimez en utilisant une méthode Delete sur des collections. Pour plus d'informations, voir *Collections* à la page 351.

Vous pouvez parcourir le métamodèle de PowerAMC pour obtenir des informations au sujet des propriétés et des collections disponibles pour chacune des métaclasses.

Exemple

```
'Variables are not typed in VBScript. You create them and the
'location where you use them determines what they are
' get the current active model
Dim mdl ' the current model
Set mdl = ActiveModel
```

Propriétés

Une *propriété* désigne une information élémentaire disponible pour l'objet, telle que le nom, le code, le commentaire, etc.

Exemple

```
'How to get a property value in a variable from table 'Customer
Dim Table_name
'Assuming MyTable is a variable that already contains a 'table object
Get the name of MyTable in Table_name variable
Table_name = MyTable.name
'Display MyTable name in output window
output MyTable.name
```

```
'How to change a property value : change value for name 'of MyTable
MyTable.name = 'new name'
```

Collections

Une *collection* désigne un ensemble d'objets.

Le modèle est l'objet racine et les autres objets sont accessibles en parcourant la collection correspondante. Ces objets sont regroupés au sein de collections que l'on peut comparer aux noeuds d'objets apparaissant dans l'arborescence de l'Explorateur d'objets.

Si un objet CLIENT possède une collection, cela signifie que la collection contient la liste des objets avec lesquels l'objet CLIENT est en relation.

Certaines fonctions sont disponibles sur les collections. Vous pouvez :

- Parcourir une collection
- Obtenir le nombre d'objets que comporte une collection
- Créer un nouvel objet à l'intérieur d'une collection, s'il s'agit d'une collection de composition

Les collections peuvent être des types suivants :

- Read-only collections — Ce sont des collections que l'on peut uniquement parcourir.
- Unordered collections — Ce sont des collections pour lesquelles l'ordre des objets dans la liste n'a pas d'importance, par exemple la collection Relationships d'une entité de MCD est une collection non ordonnée.
- Ordered collections — Ce sont des collections pour lesquelles l'ordre des objets dans les propriétés ou les méthodes utilisées est défini par l'utilisateur et doit être respecté, par exemple la collection Columns d'une table de MPD est une collection ordonnée.
- Composition collections - Ce sont des collections pour lesquelles les objets appartiennent au propriétaire de la collection. Elles sont généralement affichéed dans Explorateur d'objets. Les collections qui ne sont pas des compositions peuvent également être accessibles via le scripting. Il peut s'agir par exemple de la liste des règles de gestion associées à une table ou à une classe et affichées dans l'onglet Règles de la feuille de propriétés ou dans la liste des objets affichée dans l'onglet Dépendances de la feuille de propriétés d'un objet.

Read-only collections (collections en lecture seule)

Models est la collection globale des modèles ouverts. C'est un exemple de collection en lecture seule.

Les propriété et méthodes disponibles pour les collections en lecture seule sont les suivantes :

Propriété ou méthode	Utilisation
Count As Long	Récupère le nombre d'objets contenus dans une collection.
Item(idx As Long = 0) As BaseObject	Récupère l'objet (item) au sein d'une collection pour un indice donné. Item(0) étant le premier objet.
MetaCollection As BaseObject	Récupère l'objet MetaCollection qui définit cette collection.
Kind As Long	Récupère le type d'objet que la collection peut contenir. Renvoie une constante prédéfinie telle que cls_.
Source As BaseObject	Récupère l'objet qui possède la collection.

Exemple :

```
'How to get the number of open models and display it
'in the output window
output Models.count
```

Unordered collections (collections non ordonnées

Toutes les méthodes et les propriétés des collections en lecture seule sont également disponibles pour les collections non ordonnées.

Les propriétés et les méthodes disponibles pour les collections non ordonnées sont les suivantes :

Propriété ou méthode	Utilisation
Add(obj As BaseObject)	Ajoute un objet en dernière position dans la collection.
Remove(obj As BaseObject, delete As Boolean = False)	Ote un objet donné d'une collection et, le cas échéant, le supprime.
CreateNew(kind As Long = 0) As BaseObject	Crée un objet d'un type donné, et l'ajoute à la fin de la collection. Si aucun type d'objet n'est spécifié, la valeur 0 est utilisée, ce qui signifie que la catégorie Kind de la collection sera utilisée. Voir le fichier d'aide sur les objets du métamodèle pour connaître les restrictions relatives à l'utilisation de cette méthode.
Clear(delete As Boolean = False)	Ote tous les objets de la collection et, le cas échéant, les supprime.

Exemple :

```
'remove table TEST from the active model
Set MyModel = ActiveModel
For each T in Mymodel.Tables
```

```
   If T.code = "TEST" then
       set MyTable = T
   End if
next
ActiveModel.Tables.Remove MyTable, true
```

Ordered collections (collections ordonnées)

Toutes les méthodes et les propriétés des collections en lecture seule et non ordonnées sont également disponibles pour les collections ordonnées.

Les propriétés et les méthodes disponibles pour les collections ordonnées sont les suivantes :

Propriété ou méthode	Utilisation
Insert(idx As Long = -1, obj As BaseObject)	Insère des objets dans une collection. Si aucun indice n'est fourni, l'indice –1 est utilisé par défaut. Cela signifie que l'objet est simplement ajouté en dernière position dans la collection.
RemoveAt(idx As Long = -1, delete As Boolean = False)	Retire l'objet de la collection en fonction de l'indice donné. Si aucun indice n'est fourni, l'indice –1 est utilisé par défaut. Cela signifie que l'objet est simplement ajouté en dernière position dans la collection (le cas échéant). L'objet peut également être supprimé.
Move(source As Long, dest As Long)	Déplace l'objet de son indice source vers son indice de destination.
CreateNewAt(idx As Long = -1, kind As Long = 0) As BaseObject	Crée un objet d'un type donné, et l'insère à un emplacement spécifié. Si aucun indice n'est fourni, l'indice -1 est utilisé, ce qui signifie que l'objet est simplement ajouté comme dernier objet de la collection. Si aucun type d'objet n'est spécifié, la valeur 0 est utilisée, elle signifie que la propriété Kind sera utilisée. Voir le fichier d'aide sur les objets du métamodèle pour plus d'information sur les restrictions d'utilisation de cette méthode.

Exemple :

```
'Move first column in last position
'Assuming the variable MyTable contains a table
MyTable.Columns.move(0,-1)
```

Composition collections (collections de composition)

Les collections de composition peuvent être ordonnées ou non ordonnées.

Toutes les méthodes et les propriétés des collections non ordonnées sont également disponibles pour les compositions non ordonnées.

Les propriétés et les méthodes disponibles pour les collections de composition non ordonnées sont les suivantes :

Propriété ou méthode	Utilisation
CreateNew(kind As Long = 0) As BaseObject	Crée un objet d'un type donné et l'ajoute en dernière position dans la collection. En l'absence de type d'objet spécifié, la valeur 0 est utilisée par défaut pour indiquer que la propriété Kind de la collection sera utilisée.

Toutes les méthodes et les propriétés des collections ordonnées sont également disponibles pour les compositions ordonnées.

Toutes les méthodes et les propriétés des compositions non ordonnées sont également disponibles pour les compositions ordonnées.

Les propriétés et les méthodes disponibles pour les collections de composition ordonnées sont les suivantes :

Propriété ou méthode	Utilisation
CreateNewAt(idx As Long = -1, kind As Long = 0) As BaseObject	Crée un objet d'un type donné et l'insère à une position donnée. En l'absence d'indice spécifié, l'indice – 1 est utilisé par défaut pour indiquer que l'objet est simplement ajouté en dernière position dans la collection. En l'absence de type d'objet spécifié, la valeur 0 est utilisée par défaut pour indiquer que la propriété Kind de la collection sera utilisée.

Ces méthodes peuvent être appelées en l'absence de type d'objet spécifié. Toutefois, cela n'est possible que lorsque la collection est fortement typée, c'est-à-dire que la collection doit contenir des objets d'un type non abstrait précis. Dans de tels cas, la propriété Kind de la collection correspond à une classe instanciable et la courte description de la collection désigne le nom du type d'objet.

Exemple :

La collection Columns d'une table est une collection de composition car vous pouvez créer des colonnes depuis cette collection. En revanche, la collection Columns d'une clé n'est pas une collection de composition car il est impossible de créer des objets (colonnes) depuis cette collection mais seulement possible de les lister.

```
'Create a new table in a model
'Assuming the variable MyModel contains a PDM
'Declare a new variable object MyTable
Dim MyTable
'Create a new table in MyModel
Set MyTable = MyModel.Tables.Createnew
```

```
'Create a new column in a table
'Declare a new variable object MyColumn
Dim MyColumn
'Create a new column in MyTable in 3rd position
Set MyTable = MyTable.Columns.CreateNewAt(2)
' the column is created with a default name and code
```

Remarque : Lorsque vous parcourez les collections d'un modèle pour récupérer ces objets, sachez que vous récupérerez aussi les raccourcis des objets de même type.

Propriétés globales

Les propriétés globales suivantes sont disponibles :

Type	Propriétés globales	Utilisation
Accesseur global	ActiveModel As BaseObject ActivePackage As BaseObject ActiveDiagram As BaseObject	Récupère le modèle, le package ou le diagramme correspondant à la vue active.
	ActiveSelection As ObjectSet	Collection en lecture seule qui permet de récupérer la liste des objets sélectionnés dans le diagramme actif.
	ActiveWorkspace As BaseObject	Récupère l'espace de travail (objet Workspace) de l'application.
	MetaModel As BaseObject	Récupère le métamodèle (objet MetaModel) de l'application.
	Models As ObjectSet	Collection en lecture seule qui permet de lister les modèles ouverts.
	RepositoryConnection As BaseObject	Récupère la connexion courante du référentiel qui est l'objet qui gère la connexion au serveur du référentiel, puis fournit un accès aux documents et objets stockés sous le référentiel.
Mode d'exécution	ValidationMode As Boolean	Active ou désactive le mode de validation (True/False).
	InteractiveMode As long	Gère l'intervention de l'utilisateur en affichant ou non des boîtes de dialogue à l'aide des constantes suivantes : im_+Batch, +Dialog ou +Abort.
Application	UserName As String	Récupère le nom de connexion de l'utilisateur.

Type	Propriétés globales	Utilisation
	Viewer As Boolean	Renvoie True si l'application en cours d'exécution est une version Visionneuse dotée de fonctionnalités limitées.
	Version As String	Renvoie la version de PowerAMC.
Spécifique à OLE	ShowMode As	Vérifie ou modifie le statut de visibilité de la fenêtre d'application principale de la façon suivant : • La valeur renvoyée est True si la fenêtre principale de l'application est visible et non réduite. • La valeur renvoyée est False dans le cas contraire.
	Locked As Boolean	Peut être défini à True pour assurer que l'application continue à s'exécuter après qu'un client OLE se soit déconnecté, dans le cas contraire l'application se ferme.

Exemple :

```
'Create a new table in a model
'Get the active model in MyModel variable
Set MyModel = ActiveModel
```

Vous pouvez utiliser deux types de mode d'exécution lorsque vous lancez un script dans l'éditeur. Vous pouvez spécifier une valeur par défaut pour chacun des modes :

- Validation mode (mode de validation)
- Interactive mode (mode interactif)

Mode de validation

Le mode de validation est activé par défaut (sa valeur est égale à True), mais vous pouvez choisir de désactiver temporairement ce mode en fixant sa valeur à False.

Etat	Constante	Code	Utilisation
Activé (valeur par défaut)	True	ValidationMode = True	Chaque fois que vous manipulez un objet de PowerAMC, toutes les méthodes internes de PowerAMC sont invoquées pour vérifier la validité de vos actions. Dans le cas d'une action non permise, une erreur survient. Ce mode est très utile pour déboguer mais limite nécessairement les performances du système.
Désactivé	False	ValidationMode = False	Vous l'utilisez lorsque vous souhaitez optimiser les performances de votre système ou parce que votre algorithme requiert un état temporairement invalide. Notez toutefois que dans ce cas les règles de validation telles que l'unicité du nom ou la nécessité pour un lien d'avoir des extrémités ne sont pas appliquées dans le modèle.

Exemple :

```
ValidationMode = true
```

Mode interactif

La constante Batch est la valeur par défaut dans le mode interactif.

Ce mode prend en charge les constantes suivantes :

Constant	Code	Description
im_Batch	InteractiveMode = im_Batch	N'affiche jamais les boîtes de dialogue et utilise toujours les valeurs par défaut. Cette constante s'utilise pour des scripts d'automatisation qui ne requièrent aucune action de l'utilisateur.
im_Dialog	InteractiveMode = im_Dialog	Affiche des boîtes de dialogue d'information et de confirmation qui requièrent une action de l'utilisateur pour poursuivre l'exécution du script.

Constant	Code	Description
im_Abort	InteractiveMode = im_Abort	N'affiche jamais les boîtes de dialogue et abandonne l'exécution du script au lieu d'utiliser les valeurs par défaut à chaque fois qu'un dialogue s'impose.

Déclaration Option Explicit

Nous vous recommandons d'utiliser la déclaration Option Explicit pour déclarer vos variables. Vous éviterez ainsi toute confusion dans l'écriture de votre code car cette option est désactivée par défaut dans VBScript.

Exemple :

```
Option Explicit
ValidationMode = True
InteractiveMode = im_Batch
' get the current active model
Dim mdl ' the current model
Set mdl = ActiveModel
```

Fonctions globales

Les fonctions globales suivantes sont disponibles :

Fonctions globales	Utilisation
CreateModel (modelkind As Long, filename As String = "", flags As Long =omf_Default) As BaseObject	Crée un nouveau modèle.
CreateModelFromTemplate (filename As String, flags As Long =omf_Default) As BaseObject	Crée un nouveau modèle à l'aide d'un template de modèle.
OpenModel (filename As String, flags As Long =omf_Default) As BaseObject	Ouvre un modèle existant
Output (message As String = "")	Inscrit un message dans l'onglet Script de la fenêtre Résultats dans la fenêtre principale de PowerAMC.
NewPoint (X As Long = 0, Y As Long = 0) As APoint	Crée un point pour positionner un symbole.
NewRect (Left As Long = 0, Top As Long = 0, Right As Long = 0, Bottom As Long = 0) As Arect	Crée un rectangle pour manipuler la position des symboles.

Fonctions globales	Utilisation
NewPtList () As PtList	Crée une liste de points pour positionner un lien.
NewGUID() As String	Crée un nouvel identifiant unique (Global Unique IDentifier, GUID). Ce nouvel identifiant est renvoyé sous la forme d'une chaîne de caractères dépourvue des signes de ponctuation l'entourant habituellement "{" "}".
IsKindOf(childkind As Long, parentkind As Long) As Boolean	Renvoie True si childkind correspond à une métaclasse dérivée de la métaclasse de type parentkind, False dans le cas contraire.
ExecuteCommand (cmd As String, Optional arglist As String, Optional mode As Long) As String	Ouvre une application externe.
Rtf2Ascii (rtf As String) As String	Supprime les marqueurs RTF (Rich-Text-File) au sein d'un texte au format RTF.
ConvertToUTF8 (InputFileName As String, OutputFileName As String)	Convertit le fichier <InputFileName> en UTF8 (8-bit Unicode Transformation Format qui est un format de conversion en unicode 8 bits dans lequel l'ordre des bits est initialisé par un marqueur d'ordre des bits - initial Byte Order Mark) et écrit le résultat de la conversion du fichier dans<OutputFileName>. Les deux noms de fichiers doivent être différents.
ConvertToUTF16 (InputFileName As String, OutputFileName As String)	Convertit le fichier <InputFileName> en UTF16 (16-bit Unicode Transformation Format Little Endian qui est un format de conversion en unicode 16 bits dans lequel l'ordre des bits est initialisé par un marqueur d'ordre des bits - initial Byte Order Mark) et écrit le résultat de la conversion du fichier dans<OutputFileName>. Les deux noms de fichiers doivent être différent.
EvaluateNamedPath (FileName As String, QueryIfUnknown As Boolean = True, FailOnError As Boolean = False) As String	Remplace une variable dans un chemin d'accès par son chemin nommé correspondant.
MapToNamedPath (FileName As String) As String	Remplace le chemin d'accès d'un fichier par le chemin nommé correspondant.

Fonctions globales	Utilisation
Progress(Key As String, InStatusBar Boolean = False) As BaseObject	Crée ou récupère un indicateur de progression donné.
BeginTransaction()	Démarre une nouvelle transaction.
CancelTransaction()	Annule la transaction en cours.
EndTransaction()	Valide la transaction en cours.

OpenModel(), CreateModel() et CreateModelFromTemplate flags

Les fonctions OpenModel, CreateModel et CreateModelFromTemplate utilisent les constantes globales suivantes :

Constant	Utilisation
Omf_Default	Comportement par défaut pour les fonctions OpenModel/CreateModel.
Omf_DontOpenView	Empêche l'ouverture de la fenêtre de diagramme par défaut pour les fonctions OpenModel/CreateModel/ CreateModelFromTemplate.
Omf_QueryType	Pour la fonction CreateModel UNIQUEMENT. Oblige à spécifier le type du diagramme initial.
Omf_NewFileLock	Pour la fonction CreateModel UNIQUEMENT. Crée et verrouille le fichier correspondant.
Omf_Hidden	Empêche le modèle de s'afficher dans l'espace de travail pour les fonctions OpenModel/CreateModel/ CreateModelFromTemplate.

Modes d'exécution des commandes

Les modes d'exécution des commandes utilisent les constantes globales suivantes :

Constant	Utilisation
cmd _ShellExec	Comportement par défaut. Laisse la session de MS-Windows exécuter la commande.
cmd _PipeOutput	Redirige le résultat de la commande dans l'onglet Général de la fenêtre Résultats de PowerAMC.
cmd _PipeResult	Récupère le résultat de la commande et le retourne sous forme de chaîne de caractères.
cmd_InternalScript	Indique que le premier paramètre de la fonction globale Execute Command est un fichier VBScript qui doit être exécuté comme un script interne plutôt que de laisser le système exécuter l'application associée au type de fichier.

Exemple :

```
'Create a new model and print its name in output window
CreateModel(PDOOm.cls_Model, "C:\Temp\Test.oom|Language=Java|
Diagram=SequenceDiagram")
Output ActiveModel.name
```

Constantes globales

Les constantes globales suivantes sont disponibles :

Constante globale	Utilisation
Version As String	Retourne la version de l'application.
HomeDirectory As String	Retourne le répertoire racine de l'application.
RegistryHome As String	Retourne le chemin racine du registre de l'application.
cls_... As Long	Identifie la classe d'un objet. Cette valeur est utilisée lorsque vous avez besoin de spécifier le type d'un objet dans la méthode de création par exemple. Elle est aussi utilisée par la méthode IsKindOf disponible sur tous les objets de PowerAMC.

Constantes d'ID de classe

Les constantes sont uniques au sein d'un modèle et sont utilisées pour identifier les classes d'objets dans chaque bibliothèque. Tous les identifiants de classes commencent par "cls_" suivi du nom public de l'objet. Par exemple cls_Process identifie la classe d'objet Processus en utilisant le nom public de l'objet.

Toutefois, lorsque vous manipulez plusieurs modèles à la fois, certaines constantes peuvent être communes, par exemple cls_Package.

Pour éviter toute confusion dans le code, vous devez préfixer le nom de la constante par le nom du module, par exemple PdOOM cls_Package. De même, lorsque vous créez un modèle, vous devez préfixer la constante cls_Model par le nom du module.

Méthode IsKindOf

Boolean avec une constante de classe pour vérifier qu'un objet hérite d'un type de classe donné.

Exemple :

Vous pouvez avoir un script doté d'une boucle qui parcourt la collection Classifiers d'un MOO et qui vérifie le type des objets rencontrés (dans le cas d'interfaces ou classes) pour leur appliquer un traitement différent selon leur type.

```
'Assuming the Activemodel is an OOM model
For each c in Activemodel.Classifiers
If c.IsKindOf(cls_Class) then
Output "Class " & c.name
ElsIf c.IsKindOf(cls_Interface) then
```

```
Output "Interface" & c.name
End If
Next
```

Exemple :

Toutes les collections d'un modèle peuvent contenir des objets d'un type donné mais également des raccourcis d'objets de même type. Vous pouvez avoir un script doté d'une boucle qui parcourt la collection Tables d'un MPD et qui vérifie le type des objets rencontrés (dans ce cas tables ou raccourcis) pour leur appliquer un traitement différent selon leur type.

```
For each t in Activemodel.Tables
If t.IsKindOf(cls_Table) then
Output t.name
End If
Next
```

Bibliothèques

Les bibliothèques sont disponibles pour chaque type de modèle ainsi que pour les fonctionnalités partagées de PowerAMC.

- PdBPM - Modèle de Processus Métiers (MPM)
- PdCDM - Modèle Conceptuel de Données (MCD)
- PdCommon - contient tous les objets communs à au moins deux modèles, ainsi que les classes abstraites du modèle. Par exemple, les règles de gestion, qui sont disponibles dans tous les modèles, et la classe BaseObject, à partir de laquelle tous les objets sont dérivés, sont définies dans ce package. Les autres packages de modèle sont liés à PdCommon via des liens de généralisation indiquant que chaque modèle hérite des objets communs du package PdCommon.
- PdEAM - Modèle d'Architecture d'Entreprise (MAE)
- PdFRM - Modèle libre (MLB
- PdILM - Modèle de Fluidité de l'Information (MFI)
- PdLDM - Modèle Logique de Données (MLD)
- PdMTM - Modèle des Traitements Merise
- PdOOM - Modèle Orienté Objet (MOO)
- PdPDM - Modèle Physique de Données (MPD)
- PdPRJ - Projet
- PdRMG - Référentiel
- PdRQM - Modèle de gestion des exigences (MGX)
- PdXSM - Modèle XML
- PdWSP - Espace de travail

Pour chaque bibliothèque, vous pouvez parcourir une liste de :

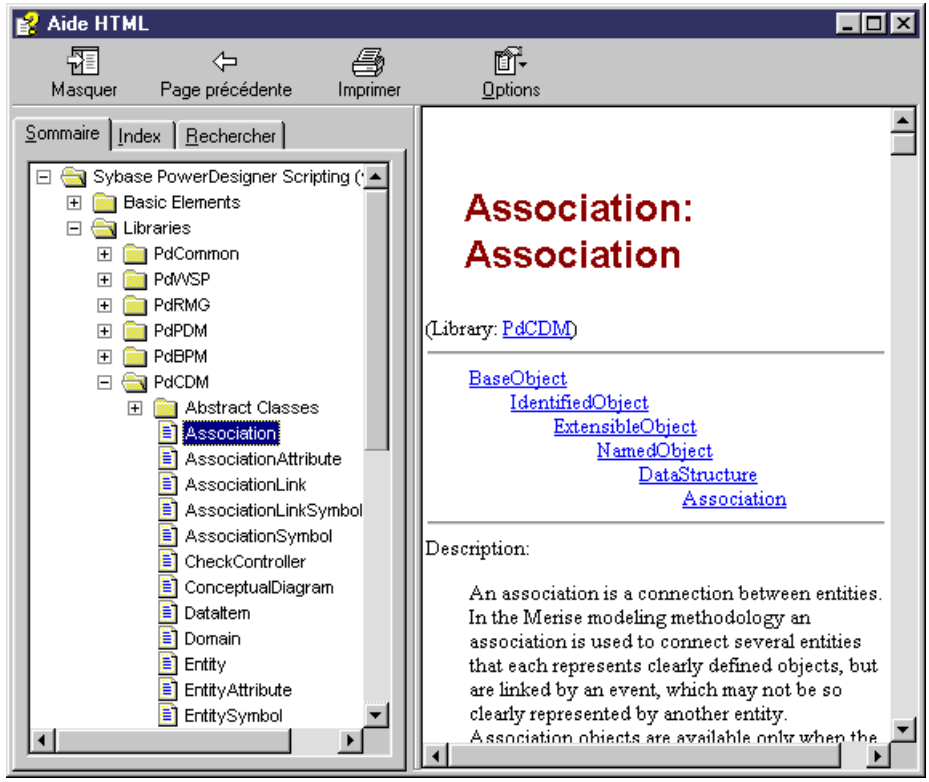
- *Abstract classes* (classes abstraites situées sous le noeud Abstract Classes). Ce sont des classes générales utilisées pour factoriser les attributs et les comportements. Elles ne sont pas visibles dans PowerAMC. Les classes instanciables héritent des classes abstraites.
- *Instanciable classes* (classes instanciables situées directement à la racine du noeud de chaque bibliothèque). Ce sont des classes spécifiques qui correspondent aux objets de l'interface. Elles possèdent des propriétés telles que le nom et le code, et elles héritent également des attributs et des comportements des classes abstraites via des liens de généralisation. Par exemple, NamedObject désigne la classe commune à la plupart des objets de modélisation de PowerAMC. Elle stocke des propriétés telles que le nom, le code, le commentaire, l'annotation et la description.

Pour plus d'informations sur les bibliothèques PowerAMC, voir *Chapitre 1, Utilisation des fichiers de ressources PowerAMC* à la page 1.

Utilisation du fichier d'aide sur les objets du métamodèle

PowerAMC fournit un fichier d'aide HTML compilé que vous pouvez ouvrir en sélectionnant **Aide > Aide sur les objets du Métamodèle** ou à partir de la boîte de dialogue de l'éditeur Edition/Exécution d'un script. Ce guide de référence est destiné à vous aider à vous familiariser avec les propriétés, les collections et les méthodes des objets de PowerAMC utilisables dans le scripting.

Le fichier d'aide sur les objets du métamodèle se compose de deux parties distinctes : l'arborescence de noeuds qui s'affiche sur le côté gauche et qui permet de naviguer à travers la hiérarchie des objets, et les descriptions correspondantes qui s'affichent à droite de l'arborescence :



Vous pouvez développer les noeuds suivants dans l'arborescence :

Noeuds	Ce que vous trouverez...
Basic Elements	Informations générales sur : • Des collections en lecture seule, ordonnées et non-ordonnées (voir *Collections* à la page 351) • Des types structurés (points, rectangles, liste de points) • Des propriétés globales (voir *Propriétés globales* à la page 355), des constantes (voir *Constantes globales* à la page 361), et des fonctions (voir *Fonctions globales* à la page 358)
Libraries	Bibliothèques portant sur les fonctionnalités courantes et sur chaque modèle (voir *Bibliothèques* à la page 362)
Appendix	Représentation hiérarchique du métamodèle de PowerAMC Liste des constantes utilisées pour identifier les objets de chaque bibliothèque.

Les objets de PowerAMC accessibles via VBScript correspondent aux objets de modélisation (tables, entités, classes, processus etc.) qui s'affichent dans l'interface utilisateur.

Pour chacun des objets de PowerAMC, vous pouvez parcourir une liste de :

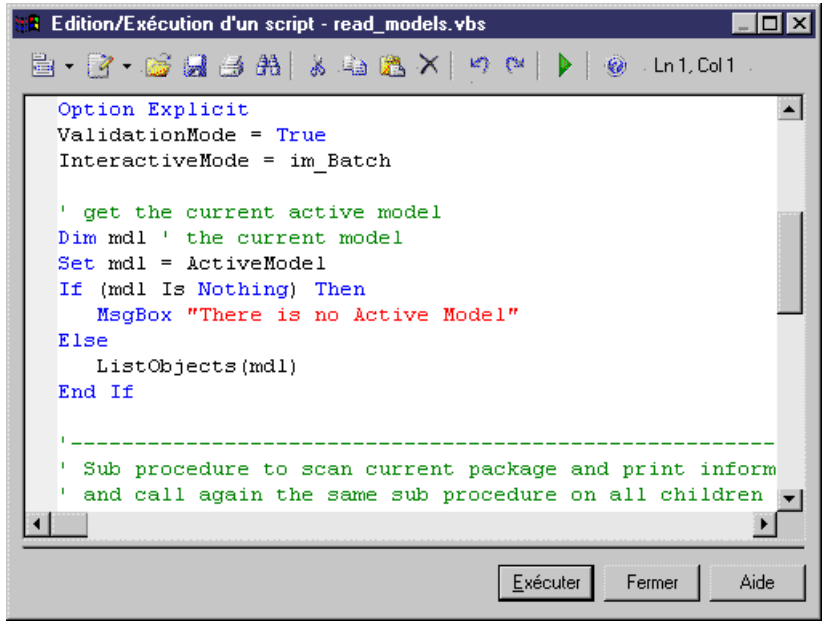
- Propriétés (exemple : Nom, Type de données, Transport)
- Collections en lecture seule, ordonnées et non-ordonnées (exemple : symboles, colonnes d'une table)
- Méthodes (exemple : Delete (), UpdateNamingOpts())

Utilisation de l'éditeur Edition/Exécution d'un script

L'éditeur Edition/Exécution d'un script fonctionne au sein de l'environnement de PowerAMC et fournit un accès à l'environnement de langage de script. Vous l'ouvrez depuis le menu **Outils > Exécuter des commandes**. L'éditeur de scripts est disponible quel que soit le type du modèle actif et même sans qu'aucun modèle ne soit ouvert.

Vous pouvez visualiser la date et l'heure à laquelle le script débute et prend fin dans l'onglet Script de la fenêtre Résultats située dans la partie inférieure de la fenêtre principale de PowerAMC, si vous avez utilisé la fonction globale Output.

L'éditeur Edition/Exécution d'un script à l'apparence suivante :



Vous pouvez utiliser les outils et raccourcis clavier suivants depuis la barre d'outils de l'éditeur Edition/Exécution d'un script :

Outil	Description	Raccourci clavier
	Menu de l'éditeur. Note : lorsque vous utilisez la fonctionnalité de recherche, le paramètre "Expression régulière" permet l'utilisation de caractères de remplacement dans l'expression de recherche.	**Maj+F11**
	Editer avec. Ouvre l'éditeur défini par défaut ou vous permet de sélectionner un autre éditeur en cliquant sur la flèche vers le bas en regard de cet outil.	**Ctrl+E**
	Exécuter. Exécute le script courant.	**F5**
	Guide de référence du script. Ouvre le fichier pdvbs12.chm.	**Ctrl+F1**

Pour plus d'informations sur la définition d'un éditeur par défaut, voir *Guide des fonctionnalités générales > L'interface de PowerAMC > Personnalisation de votre environnement de modélisation > Options générales > Définition d'un éditeur de texte*.

Signets dans le script

Dans la fenêtre de l'éditeur Edition/Exécution d'un script, vous pouvez ajouter et supprimer des signets à des endroits spécifiques du code puis naviguer de l'un à l'autre de ces signets :

Raccourci clavier	Description
Ctrl+F2	Ajoute un signet. Une marque de signet bleue s'affiche. Si vous renouvelez l'action à la même position, le signet est supprimé et la marque bleue disparaît.
F2	Passage au signet suivant.
Maj+F2	Passage au signet précédent.

Visual Basic

Si Visual Basic (VB) est installé sur votre machine, vous pouvez utiliser l'interface de VB pour l'écriture de vos scripts et avoir accès à sa fonctionnalité IntelliSense qui vérifie la validité de toutes les méthodes et les propriétés standard que vous invoquez et suggère des alternatives valides pour corriger votre code. Toutefois, l'éditeur Edition/Exécution d'un script reconnaît automatiquement les mots-clés de VBScript.

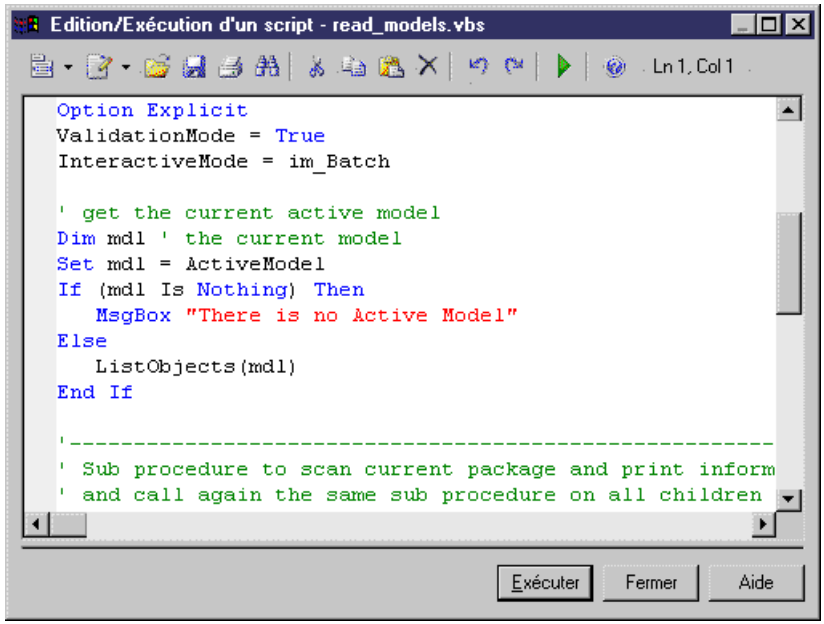
L'éditeur Edition/Exécution d'un script vous permet de :

- Créer un script
- Modifier un script
- Enregistrer un script
- Exécuter un script
- Utiliser un script d'exemple

Création d'un fichier VBScript

La boîte de dialogue Edition/Exécution d'un script permet de créer un fichier VBScript.

1. Sélectionnez **Outils > Exécuter des commandes > Editer/Exécuter le script** pour afficher la boîte de dialogue Edition/Exécution d'un script.
2. Saisissez les instructions de script directement dans la fenêtre de l'éditeur de script.



La syntaxe du script s'affiche comme dans Visual Basic.

Pour plus d'informations sur la syntaxe de VB, voir la documentation de *Visual Basic de Microsoft*.

Modification d'un fichier VBScript

La boîte de dialogue Edition/Exécution d'un script permet de modifier un fichier VBScript.

1. Ouvrez l'éditeur Edition/Exécution d'un script.
2. Cliquez sur l'outil Menu de l'éditeur et sélectionnez Ouvrir dans la liste.

   Une boîte de dialogue standard d'ouverture de fichiers s'affiche.
3. Sélectionnez un fichier VBScript (.VBS) et cliquez sur Ouvrir.

   Le fichier VBScript s'ouvre dans la fenêtre de l'éditeur Edition/Exécution d'un script. Vous pouvez alors le modifier.

Remarque : Vous pouvez insérer un fichier de script dans un script courant à l'aide de la commande Insérer du menu Menu de l'éditeur. Le script sera inséré au point d'insertion du curseur.

Enregistrement d'un fichier VBScript

Nous vous recommandons d'enregistrer votre modèle et votre fichier de script avant de l'exécuter.

1. Ouvrez l'éditeur Edition/Exécution d'un script.
2. Saisissez les instructions de script directement dans la fenêtre de l'éditeur de script.
3. Cliquez sur l'outil Menu de l'éditeur et sélectionnez Enregistrer dans la liste.

   ou

   Cliquez sur l'outil Enregistrer.

   Une boîte de dialogue standard d'enregistrement de fichiers s'affiche si votre script n'a jamais été enregistré auparavant.
4. Sélectionnez le répertoire dans lequel vous souhaitez enregistrer le fichier de script.
5. Saisissez un nom pour le fichier de script et cliquez sur Enregistrer.

Exécution d'un fichier VBScript

Vous pouvez exécuter un fichier VBScript à partir de PowerAMC.
Ouvrez un script et cliquez sur l'outil Exécuter ou sur le bouton Exécuter.

Le script s'exécute et vous pouvez voir la progression de son exécution dans la fenêtre Résultats situé dans la partie inférieure de la fenêtre principale de PowerAMC si vous avez au préalable utilisé la fonction globale Output qui permet d'afficher la progression d'une exécution et les erreurs dans l'onglet Script.

Si une erreur de compilation survient, une boîte de message s'affiche pour vous indiquer le type d'erreur. Une description brève de l'erreur s'affiche également dans le volet de résultats de la boîte de dialogue Edition/Exécution d'un script ; le curseur est placé en regard de l'erreur dans la fenêtre de l'éditeur.

L'éditeur Edition/Exécution d'un script prend en charge plusieurs niveaux de commandes Annuler et Répéter. Cependant, si vous exécutez un script qui modifie des objets dans plusieurs modèles, vous devez utilisez les commandes Annuler et Répéter dans chacun des modèles appelés par le script.

Remarque : Pour éviter les abandons d'applications, vous pouvez intercepter les erreurs à l'aide de la déclaration On Error Resume Next. Mais vous ne pouvez pas intercepter les erreurs à l'aide de cette déclaration lorsque vous utilisez le mode interactif avec la constante im_Abort.

Vous pouvez également insérer et personnaliser des commandes dans le menu Outils qui vous permettront de lancer automatiquement vos propres scripts.

Pour plus d'informations sur la personnalisation des commandes, voir *Personnalisation des menus PowerAMC à l'aide de compléments* à la page 414.

Utilisation des fichiers d'exemple VBScript

PowerAMC est livré avec des exemples de script que vous pouvez utiliser comme base pour écrire vos propres scripts. Il sont regroupés dans le répertoire d'installation VB Scripts de PowerAMC.

Ces scripts sont destinés à vous montrer une partie de l'étendue des types d'actions que vous pouvez effectuer sur les objets de PowerAMC à l'aide de VBScript. Ils sont également destinés à vous faciliter la rédaction de votre code dans vos propres scripts puisque vous pouvez aisément copier puis coller des parties de code du script d'exemple dans votre propre script.

Il est recommandé de conserver intacts les fichiers d'exemple de scripts et de n'utiliser que des copies.

Exemple de balayage d'un modèle

L'exemple suivant illustre un script doté d'une boucle qui balaye un modèle et ses sous-packages pour renvoyer des informations :

```
' Scan CDM Model and display objects information
' going down each package
Option Explicit
ValidationMode = True
InteractiveMode = im_Batch
' get the current active model
Dim mdl ' the current model
Set mdl = ActiveModel
If (mdl Is Nothing) Then
 MsgBox "There is no Active Model"
Else
 Dim fldr
 Set Fldr = ActiveDiagram.Parent
 ListObjects(fldr)
End If
' Sub procedure to scan current package and print information on
objects from current package
' and call again the same sub procedure on all children package
' of the current package
Private Sub ListObjects(fldr)
 output "Scanning " & fldr.code
 Dim obj ' running object
 For Each obj In fldr.children
  ' Calling sub procedure to print out information on the object
  DescribeObject obj
 Next
 ' go into the sub-packages
 Dim f ' running folder
 For Each f In fldr.Packages
  'calling sub procedure to scan children package
  ListObjects f
 Next
```

```
End Sub
' Sub procedure to print information on current object in output
Private Sub DescribeObject(CurrentObject)
 if CurrentObject.ClassName ="Association-Class link" then exit sub
 'output "Found "+CurrentObject.ClassName
 output "Found "+CurrentObject.ClassName+" """+CurrentObject.Name
+""", Created by "+CurrentObject.Creator+" On
"+Cstr(CurrentObject.CreationDate)
End Sub
```

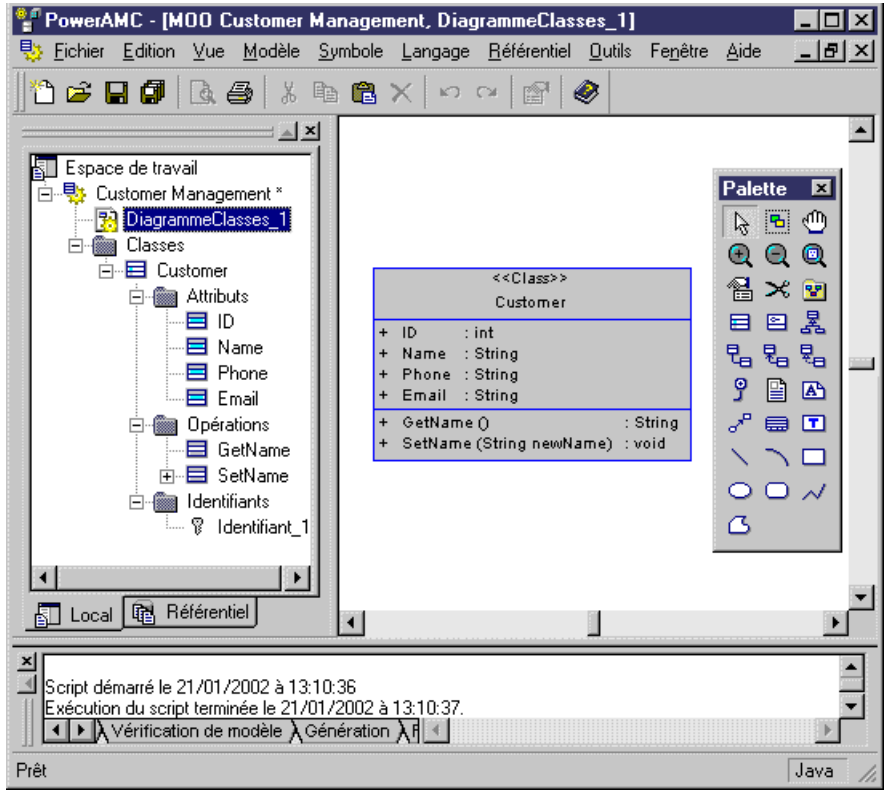
Exemple de création d'un modèle

L'exemple suivant illustre un script qui crée un nouveau MOO :

```
Option Explicit
' Initialization
' Set interactive mode to Batch
InteractiveMode = im_Batch
' Main function
' Create an OOM model with a class diagram
Dim Model
Set model = CreateModel(PdOOM.cls_Model, "|Diagram=ClassDiagram")
model.Name = "Customer Management"
model.Code = "CustomerManagement"
' Get the class diagram
Dim diagram
Set diagram = model.ClassDiagrams.Item(0)
' Create classes
CreateClasses model, diagram
' Create classes function
Function CreateClasses(model, diagram)
 ' Create a class
 Dim cls
 Set cls = model.CreateObject(PdOOM.cls_Class)
 cls.Name = "Customer"
 cls.Code = "Customer"
 cls.Comment = "Customer class"
 cls.Stereotype = "Class"
 cls.Description = "The customer class defines the attributes and
behaviors of a customer."
 ' Create attributes
 CreateAttributes cls
 ' Create methods
 CreateOperations cls
 ' Create a symbol for the class
 Dim sym
 Set sym = diagram.AttachObject(cls)
 CreateClasses = True
End Function
' Create attributes function
Function CreateAttributes(cls)
 Dim attr
 Set attr = cls.CreateObject(PdOOM.cls_Attribute)
 attr.Name = "ID"
 attr.Code = "ID"
 attr.DataType = "int"
```

```
 attr.Persistent = True
 attr.PersistentCode = "ID"
 attr.PersistentDataType = "I"
 attr.PrimaryIdentifier = True
 Set attr = cls.CreateObject(PdOOM.cls_Attribute)
 attr.Name = "Name"
 attr.Code = "Name"
 attr.DataType = "String"
 attr.Persistent = True
 attr.PersistentCode = "NAME"
 attr.PersistentDataType = "A30"
 Set attr = cls.CreateObject(PdOOM.cls_Attribute)
 attr.Name = "Phone"
 attr.Code = "Phone"
 attr.DataType = "String"
 attr.Persistent = True
 attr.PersistentCode = "PHONE"
 attr.PersistentDataType = "A20"
 Set attr = cls.CreateObject(PdOOM.cls_Attribute)
 attr.Name = "Email"
 attr.Code = "Email"
 attr.DataType = "String"
 attr.Persistent = True
 attr.PersistentCode = "EMAIL"
 attr.PersistentDataType = "A30"
 CreateAttributes = True
End Function
' Create operations function
Function CreateOperations(cls)
 Dim oper
 Set oper = cls.CreateObject(PdOOM.cls_Operation)
 oper.Name = "GetName"
 oper.Code = "GetName"
 oper.ReturnType = "String"
 Dim body
 body = "{" + vbCrLf
 body = body + " return Name;" + vbCrLf
 body = body + "}"
 oper.Body = body
 Set oper = cls.CreateObject(PdOOM.cls_Operation)
 oper.Name = "SetName"
 oper.Code = "SetName"
 oper.ReturnType = "void"
 Dim param
 Set param = oper.CreateObject(PdOOM.cls_Parameter)
 param.Name = "newName"
 param.Code = "newName"
 param.DataType = "String"
 body = "{" + vbCrLf
 body = body + " Name = newName;" + vbCrLf
 body = body + "}"
 oper.Body = body
 CreateOperations = True
End Function
```

Le script précédent produit le résultat suivant dans l'interface :



Tâches de base pouvant être réalisées à l'aide de scripts

Vous pouvez utiliser des scripts pour créer et ouvrir des modèles, ainsi que pour manipuler des objets et symboles dans PowerAMC.

Création d'un modèle à l'aide de scripts

Vous créez un modèle à l'aide de la fonction globale CreateModel (modelkind As Long, filename As String = "", flags As Long =omf_Default) As BaseObject et de la constante cls_Model préfixées par le nom du module pour identifier le type du modèle que vous souhaitez créer.

Notez que des arguments supplémentaires peuvent être spécifiés dans le paramètre filename en fonction du type de modèle (Language, DBMS, Copy, Diagram). L'argument Diagram utilise le nom public (public name) mais le nom localisé (celui figurant dans la boîte de dialogue de sélection d'une cible) est également accepté. Cependant, il n'est pas recommandé d'utiliser le nom localisé car votre script ne fonctionnera que dans la version localisée de PowerAMC.

Exemple

```
Option Explicit
' Call the CreateModel global function with the following parameters:
'   - The model kind is an Object Oriented Model (PdOOM.Cls_Model)
'   - The Language is enforced to be Analysis
'   - The first diagram will be a class diagram
'   - The language definition (for Analysis) is copied inside the
model
'   - The first diagram will not be opened in a window
'   - The new created model will not appear in the workspace
Dim NewModel
set NewModel = CreateModel(PdOOM.Cls_Model, "Language=Analysis|
Diagram=ClassDiagram|Copy", omf_DontOpenView Or omf_Hidden)
If NewModel is Nothing then
   msgbox "Fail to create UML Model", vbOkOnly, "Error"    ' Display
an error message box
Else
   output "The UML model has been successfully created"    ' Display a
message in the application output window
' Initialize model name and code
   NewModel.Name = "Sample Model"
   NewModel.Code = "Sample"
' Save the new model in a file
   NewModel.Save "c:\temp\MySampleModel.oom"
' Close the model
   NewModel.Close
' Release last reference to the model object to free memory
   Set NewModel = Nothing
End If
```

Ouvrir un modèle à l'aide de scripts

Vous ouvrez un modèle à l'aide de la fonction globale OpenModel (filename As String, flags As Long =omf_Default) As BaseObject.

Exemple

```
Option Explicit
' Call the OpenModel global function with the following parameters:
'   - The model file name
'   - The default diagram will not be opened in a window
'   - The opened model will not appear in the workspace
Dim ExistingModel, FileName
FileName = "c:\temp\MySampleModel.oom"
On Error Resume Next         ' Avoid generic scripting error message
like 'Invalid File Name
Set ExistingModel = OpenModel(FileName, omf_DontOpenView Or
omf_Hidden)
On Error Goto 0               ' Restore runtime error detection
If ExistingModel is nothing then
   msgbox "Fail to open UML Model:" + vbCrLf + FileName, vbOkOnly,
"Error"    ' Display an error message box
Else
```

```
  output "The UML model has been successfully opened"    ' Display a
message in the application output window
End If
```

Création d'un objet à l'aide de scripts

Il est recommandé de créer un objet directement à partir de la collection à laquelle il appartient afin d'obtenir immédiatement un état valide de l'objet. En revanche, vous créez certes l'objet, mais pas son symbole.

Vous pouvez également utiliser la méthode suivante : CreateObject(ByVal Kind As Long, ByVal ParentCol As String = "", ByVal Pos As Long = -1, ByVal Init As Boolean = -1) As BaseObject

Créer un objet dans un modèle

```
If not ExistingModel is Nothing Then
' Call the CreateNew() method on the collection that owns the object
   Dim MyClass
   Set MyClass = ExistingModel.Classes.CreateNew()
   If MyClass is nothing Then
     msgbox "Fail to create a class", vbOkOnly, "Error"    ' Display
an error message box
   Else
      output "The class objects has been successfully created"    '
Display a message in the application output window
     ' Initialize its name and code using a specific method
      ' that ensures naming conventions (Uppercase or lowercase
constraints,
      ' invalid characters...) are respected and that the name and
code
      ' are unique inside the model
      MyClass.SetNameAndCode "Customer", "cust"
      ' Initialize other properties directly
      MyClass.Comment = "Created by script"
      MyClass.Stereotype = "MyStereotype"
      MyClass.Final = true
      ' Create an attribute inside the class
      Dim MyAttr
      Set MyAttr = MyClass.Attributes.CreateNew()
      If not MyAttr is nothing Then
         output "The class attribute has been successfully created"
         MyAttr.SetNameAndCode "Name", "custName"
         MyAttr.DataType = "String"
         Set MyAttr = Nothing
      End If
      ' Reset the variable in order to avoid memory leaks
      Set MyClass = Nothing
   End If
End If
```

Créer un objet dans un package

```
If not ExistingModel is Nothing Then
   ' Create a package first
```

```
   Dim MyPckg
   Set MyPckg = ExistingModel.Packages.CreateNew()
   If not MyPckg is Nothing then
      output "The package has been successfully created"
      MyPckg.SetNameAndCode "All interfaces", "intf"
      ' Create an interface object inside the package
      Dim MyIntf
      Set MyIntf = MyPckg.Interfaces.CreateNew()
      If not MyIntf is Nothing then
         output "The interface object has been successfully created
inside the package"
         MyIntf.SetNameAndCode "Customer Interface", "custIntf"
         Set MyIntf = Nothing
      End If
      Set MyPckg = Nothing
   End If
End If
```

Création d'un symbole à l'aide de scripts

Vous créez le symbole d'un objet en attachant l'objet au diagramme actif à l'aide de la méthode suivante : AttachObject(ByVal Obj As BaseObject) As BaseObject.

Exemple

```
set symbol1 = ActiveDiagram.AttachObject(entity1)
```

Remarque : La méthode AttachObject peut également être utilisée pour créer un synonyme graphique ou un raccourci. Pour plus d'informations, voir les sections sur la création de synonymes graphiques et de raccourcis.

Affichage des symboles d'objets dans un diagramme à l'aide de scripts

Vous pouvez afficher le symbole d'un objet dans un diagramme à l'aide des méthodes suivantes:

- AttachObject(ByVal Obj As BaseObject) As BaseObject pour créer un symbole pour un objet non lien.
- AttachLinkObject(ByVal Link As BaseObject, ByVal Sym1 As BaseObject = NULL, ByVal Sym2 As BaseObject = NULL) As BaseObject pour créer un symbole pour un objet lien.
- AttachAllObjects() As Boolean pour créer un symbole pour chaque objet du package qui peut être affiché dans le diagramme courant.

Exemple

```
If not ExistingModel Is Nothing and not MyRealization Is Nothing Then
   ' Symbols are specific kind of objects that can be manipulated by
script
   ' We are now going to display the class, interface and realization
in the
   ' main diagram of the model and customize their presentation
```

```
   ' Retrieve main diagram
   Dim MyDiag
   Set MyDiag = ExistingModel.DefaultDiagram
   If not MyDiag is Nothing and
MyDiag.IsKindOf(PdOOM.Cls_ClassDiagram) Then
     ' Display the class, interface shortcut and realization link in
the diagram
      ' using default positions and display preferences
      Dim MyClassSym, MyIntfSym, MyRlzsSym
      Set MyClassSym = MyDiag.AttachObject(FoundClass)
      Set MyIntfSym  = MyDiag.AttachObject(IntfShct)
      Set MyRlzsSym  = MyDiag.AttachLinkObject(MyRealization,
MyClassSym, MyIntfSym)
      If not MyRlzsSym is Nothing Then
        output "Objects have been successfully displayed in diagram"
      End If
      ' Another way to do the same is the use of AttachAllObjects()
method:
      ' MyDiag.AttachAllObjects
      ' Changes class symbol format
      If not MyClassSym is nothing Then
         MyClassSym.BrushStyle = 1 ' Solid background (no gradient)
         MyClassSym.FillColor = RGB(255, 126, 126) ' Red background
color
         MyClassSym.LineColor = RGB(0, 0, 0) ' Black line color
         MyClassSym.LineWidth = 2 ' Double line width
         Dim Fonts
         Fonts = "ClassStereotype " + CStr(RGB(50, 50, 126)) + "
Arial,8,I"
       Fonts = Fonts + vbCrLf + "DISPNAME " + CStr(RGB(50, 50, 50)) +
" Arial,12,B"
        Fonts = Fonts + vbCrLf + "ClassAttribute " + CStr(RGB(150, 0,
0)) + " Arial,8,N"
         MyClassSym.FontList = Fonts ' Change font list
      End If
      ' Changes interface symbol position
      If not MyIntfSym is nothing Then
         Dim IntfPos
         Set IntfPos = MyIntfSym.Position
         If not IntfPos is Nothing Then
            IntfPos.x = IntfPos.x + 5000
            IntfPos.y = IntfPos.y + 5000
            MyIntfSym.Position = IntfPos
            Set IntfPos = Nothing
         End If
      End If
      ' Changes the link symbol corners
      If not MyRlzsSym is Nothing Then
         Dim CornerList, Point1, Point2
         Set CornerList = MyRlzsSym.ListOfPoints
         Set Point1 = CornerList.Item(0)
         Set Point2 = CornerList.Item(1)
         CornerList.InsertPoint 1, Max(Point1.x, Point2.x),
Min(Point1.y, Point2.y)
         Set CornerList = Nothing
         ' Max and Min are functions defined at end of this script
```

```
      End If
      ' Release the variables
      Set MyDiag = Nothing
      Set MyClassSym = Nothing
      Set MyIntfSym = Nothing
      Set MyRlzsSym = Nothing
   End If
End If
```

Positionnement d'un symbole à côté d'un autre à l'aide de scripts

Vous positionnez un symbole à côté d'un autre à l'aide des points X et Y (respectivement Abscisse et Ordonnée) et de la combinaison de la méthode (Position As Apoint) et de la fonction (NewPoint(X As Long = 0, Y As Long = 0) As Apoint)).

Exemple

```
Dim diag
Set diag = ActiveDiagram
Dim sym1, sym2
Set sym1 = diag.Symbols.Item(0)
Set sym2 = diag.Symbols.Item(1)
X1 = sym1.Position.X
Y1 = sym1.Position.Y
' Move symbols next to each other using a fixed arbitrary space
sym2.Position = NewPoint(X1+5000, Y1)
' Move symbols for them to be adjacent
sym2.Position = NewPoint(X1 + (sym1.Size.X+sym2.Size.X)/2, Y1)
```

Suppression d'un objet dans un modèle à l'aide de scripts

Vous supprimez un objet dans le modèle à l'aide de la méthode Delete As Boolean.

Exemple

```
If not ExistingModel is Nothing Then
   ' Create another class first
   Dim MyClassToDelete
   Set MyClassToDelete = ExistingModel.Packages.CreateNew()
   If not MyClassToDelete is Nothing then
      output "The second class has been successfully created"
      ' Just call Delete method to delete the object
      ' This will remove the object from the collection of model
classes
      MyClassToDelete.Delete
      ' The object is still alive but it has notified all other
     ' objects of its deletion. It is no more associated with other
objects.
      ' Its status is deleted
      If MyClassToDelete.IsDeleted() Then
         output "The second class has been successfully deleted"
      End If
      ' The reset of the VbScript variable will release the last
     ' reference to this object abd provoke the physical destruction
      ' and free the memory
```

```
      Set MyClassToDelete = Nothing
   End If
End If
```

Récupération d'un objet dans le modèle à l'aide de scripts

Le modèle suivant illustre comment vous pouvez récupérer un objet à l'aide de son code dans le modèle.

Exemple

```
' Call a function that is implemented just after in the script
Dim FoundIntf, FoundClass
Set FoundIntf = RetrieveByCode(ExistingModel, PDOOM.Cls_Interface,
"custIntf")
Set FoundClass = RetrieveByCode(ExistingModel, PDOOM.Cls_Class,
"cust")
If (not FoundIntf is nothing) and (not FoundClass is Nothing) Then
 output "The class and interface objects have been successfully
retrieved by their code"
End If
' Implement a method that retrieve an object by code
' The first parameter is the root folder on which the research begins
' The second parameter is the kind of object we are looking for
' The third parameter is the code of the object we are looking for
Function RetrieveByCode(RootObject, ObjectKind, CodeValue)
 ' Test root parameter
 If RootObject is nothing Then
  Exit Function       ' Root object is not defined
 End If
 If RootObject.IsShortcut() Then
  Exit Function       ' Root object is a shortcut
 End If
 If not RootObject.IsKindOf(Cls_BaseFolder) Then
  Exit Function       ' Root object is not a folder
 End If
 ' Loop on all objects in folder
 Dim SubObject
 For Each SubObject in RootObject.Children
  If SubObject.IsKindOf(ObjectKind) and SubObject.Code = CodeValue
Then
   Set RetrieveByCode = SubObject   ' Initialize return value
   Set SubObject = Nothing
   Exit Function
  End If
 Next
 Set SubObject = Nothing
 ' Recursive call on sub-folders
 Dim SubFolder
 For Each SubFolder in RootObject.CompositeObjects
  If not SubFolder.IsShortcut() Then
   Dim Found
   Set Found = RetrieveByCode(SubFolder, ObjectKind, CodeValue)
   If not Found Is Nothing Then
    Set RetrieveByCode = Found   ' Initialize return parameter
```

```
    Set Found = Nothing
    Set SubFolder = Nothing
    Exit Function
   End If
  End If
 Next
 Set SubFolder = Nothing
End Function
```

Création d'un raccourci dans un modèle à l'aide de scripts

Vous créez un raccourci dans le modèle à l'aide de la méthode CreateShortcut(ByVal NewPackage As BaseObject, ByVal ParentCol As String = "") As BaseObject.

Exemple

```
' We want to reuse at the model level the interface defined in the
package
' To do that, we need to create a shortcut of the interface at the
model level
Dim IntfShct
If not FoundIntf is Nothing and not ExistingModel Is Nothing Then
   ' Call the CreateShortcut() method and specify the model
   ' for the package where we want to create the shortcut
   Set IntfShct = FoundIntf.CreateShortcut(ExistingModel)
   If not IntfShct is nothing then
      output "The interface shortcut has been successfully created"
   End If
End If
```

Création d'un objet lien à l'aide de scripts

Vous créez un objet lien à l'aide de la méthode CreateNew(kind As Long = 0) As BaseObject, puis vous devez déclarer ses extrémités.

Exemple

```
Dim MyRealization
If (not ExistingModel Is Nothing) and (not FoundClass Is Nothing) and
(not IntfShct is Nothing) Then
   ' We are now going to create a realization link between the class
and the interface
   ' The link is an object like others with two mandatory attributes:
Object1 and Object2
   ' For oriented links, Object1 is the source and Object2 is the
destination
   Set MyRealization = ExistingModel.Realizations.CreateNew()
   If not MyRealization is Nothing then
      output "The realization link has been successfully created"
      ' Initialize both extremities
      Set MyRealization.Object1 = FoundClass
      Set MyRealization.Object2 = IntfShct
      ' Initialize Name and Code
      MyRealization.SetNameAndCode "Realize Main interface", "Main"
```

```
   End If
End If
```

Parcours d'une collection à l'aide de scripts

Toutes les collections peuvent être itérées à l'aide de la construction usuelle "For Each variable In collection".

Cette boucle commence par "For each <variable> in <collection>" et se termine par "Next".

La boucle est itérée sur chacun des objets de la collection. L'objet est disponible dans <variable>.

Exemple

```
'How to browse the collection of tables available on a model
Set MyModel = ActiveModel
'Assuming MyModel is a variable containing a PDM object.
For each T in MyModel.Tables
   'Variable T now contains a table from Tables collection of the
model
   Output T.name
Next
```

Manipulation d'objets dans une collection à l'aide de scripts

Dans l'exemple suivant, nous allons manipuler les objets dans une collection en créant des objets règles de gestion puis en les attachant à un objet classe. Pour cela, nous allons :

- Créer des objets règles de gestion
- Initialiser leurs attributs
- Récupérer les premiers objets dans la collection des attributs de classe
- Ajouter les règles créées au début et à la fin de la collection des règles associées
- Déplacer une règle à la fin de la collection des règles associées
- Oter une règle de la collection des règles associées

Exemple

```
If (not ExistingModel Is Nothing) and (not FoundClass Is Nothing)
Then
   ' We are going to create business rule objects and attached them to
the class
   ' Create first the business rule objects
   Dim Rule1, Rule2
   Set Rule1 = ExistingModel.BusinessRules.CreateNew()
   Set Rule2 = ExistingModel.BusinessRules.CreateNew()
   If (not Rule1 is Nothing) And (not Rule2 Is Nothing) Then
      output "Business Rule objects have been successfully created"
      ' Initialize rule attributes
      Rule1.SetNameAndCode "Mandatory Name", "mandatoryName"
      Rule1.ServerExpression = "The Name attribute cannot be empty"
      Rule2.SetNameAndCode "Unique Name", "uniqueName"
      Rule2.ServerExpression = "The Name attribute must be unique"
      ' Retrieve the first object in the class attributes collection
```

```
      Dim FirstAttr, AttrColl
      Set AttrColl = FoundClass.Attributes
      If not AttrColl is Nothing Then
         If not AttrColl.Count = 0 then
            Set FirstAttr = AttrColl.Item(0)
         End If
      End If
      Set AttrColl = Nothing
      If not FirstAttr is Nothing Then
         output "First class attribute successfully retrieved from
collection"
         ' Add Rule1 at end of attached rules collection
         FirstAttr.AttachedRules.Add Rule1
         ' Add Rule2 at the beginning of attached rules collection
         FirstAttr.AttachedRules.Insert 0, Rule2
         ' Move Rule2 at end of collection
         FirstAttr.AttachedRules.Move 1, 0
         ' Remove Rule1 from collection
         FirstAttr.AttachedRules.RemoveAt 0
         Set FirstAttr = Nothing
      End If
   End If
  Set Rule1 = Nothing
   Set Rule2 = Nothing
End If
```

Etendre le métamodèle à l'aide de scripts

Lorsque vous importez un fichier à l'aide de scripts, vous pouvez importer comme attributs étendus ou collections étendues certaines propriétés qui peuvent ne pas correspondre à des attributs standards.

Dans l'exemple suivants nous allons :

- Créer un nouveau fichier d'extension
- Initialiser des attributs d'extension de modèle
- Définir un nouveau stéréotype pour la métaclasse Class dans le profil
- Définir un attribut étendu pour ce stéréotype

Exemple

```
If not ExistingModel Is Nothing Then
 ' Création d'une extension
 Dim ModelExtension
 Set ModelExtension =
ExistingModel.ExtendedModelDefinitions.CreateNew()
 If not ModelExtension is Nothing Then
  output "Model extension successfully created in model"
  ' Initialisation des attributs d'extension de modèle
  ModelExtension.Name = "Extension for Import of XXX files"
  ModelExtension.Code = "importXXX"
  ModelExtension.Family = "Import"
  ' Définition d'un nouveau stéréotype pour la métaclasse Class dans
la section de profil
  Dim MySttp
```

```
  Set MySttp = ModelExtension.AddMetaExtension(PdOOM.Cls_Class,
Cls_StereotypeTargetItem)
  If not MySttp Is Nothing Then
   output "Stereotype extension successfully created in extension"
   MySttp.Name = "MyStereotype"
   MySttp.UseAsMetaClass = true ' The stereotype will behave as a new
metaclass (specific list and category in browser)
   ' Définition d'un attribut étendu pour ce stéréotype
   Dim MyExa
   Set MyExa =
MySttp.AddMetaExtension(Cls_ExtendedAttributeTargetItem)
   If not MyExa is Nothing Then
    output "Attribut étendu créé dans l'extension"
    MyExa.Name = "MonAttribut"
    MyExa.Comment = "attribut personnalisé provenant d'importation"
    MyExa.DataType = 10  ' Correspond à integer
    MyExa.Value = "-1" ' Valeur par défaut
    Set MyExa = Nothing
   End If
   ' Définition d'une collection étendue pour ce stéréotype
   Dim MyExCol
   Set MyExCol =
MySttp.AddMetaExtension(Cls_ExtendedCollectionTargetItem)
   If not MyExCol is Nothing Then
    output "Collection étendue créé dans l'extension"
    MyExCol.Name = "MaCollection"
    MyExCol.Comment = "collection personnalisée provenant
d'impotation"
    MyExCol.DestinationClassKind = PdOOM.Cls_class  ' La collection
ne peut stocker que des classes
    MyExCol.Destinationstereotype = "MonStereotype"  ' La collection
ne peut stocker que des classes avec le stéréotype "MonStereotype"
    Set MyExCol = Nothing
   End If
   Set MySttp = Nothing
  End If
  Set ModelExtension = Nothing
 End If
End If
```

Manipuler des propriétés étendues d'objets à l'aide de scripts

Vous pouvez dynamiquement récupérer et définir des propriétés étendues d'objets telles que les attributs et les collections à l'aide de scripts.

La syntaxe pour identifier une propriété d'objet est la suivante :

```
"<TargetCode>.<PropertyName>"
```

Par exemple, pour récupérer l'attribut étendu MyAttribute depuis l'objet importXXX, vous utilisez :

```
GetExtendedAttribute("importXXX.MyAttribute")
```

Notez que si le script est à l'intérieur d'un profil (par exemple dans un script de vérification personnalisée), vous pouvez utiliser la variable %CurrentTargetCode% à la place de TargetCode codé en dur, afin de favoriser la réutilisation de votre script.

Par exemple :

```
GetExtendedAttribute("%CurrentTargetCode%.MyAttribute")
```

Dans l'exemple suivant, nous allons :

- Modifier l'attribut étendu sur la classe
- Modifier la collection étendue sur la classe
- Ajouter la classe dans sa propre collection étendue en vue d'être utilisée comme une collection standard

Exemple

```
If (not ExistingModel Is Nothing) and (not FoundClass Is Nothing)
Then
 ' Modifier l'attribut étendu sur la classe
 Dim ExaName
 ExaName = "importXXX.MyAttribute"  ' nom d'attribut préfixé par le
code d'extension
 If FoundClass.HasExtendedAttribute(ExaName) Then
  output "Atrtribut étendu accessible"
  FoundClass.SetExtendedAttributeText ExaName, "1024"
  FoundClass.SetExtendedAttribute ExaName, 2048
  Dim valAsText, valAsInt
  valAsText = FoundClass.GetExtendedAttributeText(ExaName)
  valAsInt = FoundClass.GetExtendedAttribute(ExaName)
 End If
 ' Modifier la collection étendue sur la classe
 Dim ExColName, ExCol
 ExColName = "importXXX.MyCollection"  ' nom de collection préfixé
par le code d'extension
 Set ExCol = FoundClass.GetExtendedCollection(ExColName)
 If not ExCol is Nothing Then
  output "Collection étendue accessible"
  ' La collection étendue peut être utilisée comme une collection
standard
  ' par exemple, nous ajoutons la classe dans sa propre collection
étendue
  ExCol.Add FoundClass
  Set ExCol = Nothing
 End If
End If
```

Création d'un synonyme graphique à l'aide de scripts

Vous créez un synonyme graphique en attachant le même objet deux fois au même package.

Exemple

```
set diag = ActiveDiagram
set pack = ActivePackage
```

```
set class = pack.classes.createnew
set symbol1 = diag.AttachObject (class)
set symbol2 = diag.AttachObject (class)
```

Création d'une sélection d'objets à l'aide de scripts

L'objet Object Selection est un objet du modèle très utile pour sélectionner d'autres objets du modèle afin de leur appliquer un traitement spécifique. Vous pouvez par exemple ajouter des objets dans l'objet Object Selection pour les déplacer dans un autre package, et cela en une seule opération, plutôt que de répéter la même opération pour chacun des objets individuellement.

Toutes les fois que vous manipulez un ensemble d'objets dans l'interface utilisateur, vous devez utiliser l'objet Object Selection dans les scripts.

- Créer un objet Object Selection

Vous créez l'objet Object Selection depuis un modèle en utilisant la méthode CreateSelection : CreateSelection() As BaseObject.

Exemple

```
Set MySel = ActiveModel.CreateSelection
```

- Ajouter des objets individuellement

Vous pouvez ajouter des objets individuellement en ajoutant l'objet requis à la collection Objects.

Vous utilisez la méthode suivante de l'objet Object Selection : Add(obj As BaseObject)

Exemple

Ajout d'un objet dénommé Editeur :

```
MySel.Objects.Add(Editeur)
```

- Ajouter des objets d'un type donné

Vous pouvez ajouter tous les objets d'un type donné en utilisant la méthode suivante de l'objet Object Selection : AddObjects(ByVal RootPackage As BaseObject, ByVal ClassType As Long, ByVal IncludeShortcuts As Boolean = 0, ByVal Recursive As Boolean = 0).

RootPackage désigne le package à partir duquel les objets sont à ajouter.

ClassType désigne le type d'objet à ajouter.

IncludeShortcuts est le paramètre qui permet d'inclure des raccourcis.

Recursive est le paramètre qui permet de rechercher dans tous les sous-packages.

Exemple

Un ajout de classes sans inclusion de raccourcis et sans récursivité dans les sous-packages :

```
MySel.AddObjects(folder,cls_class)
```

- Retrait d'objets de la sélection courante

Vous pouvez retirer des objets de la sélection courante à l'aide de la méthode suivante de l'objet Object Selection : RemoveObjects(ByVal ClassType As Long, ByVal IncludeShortcuts As Boolean = -1)

Exemple
Retrait de toutes les classes et raccourcis de l'objet Object Selection :

```
MySel.RemoveObjects(cls_class, -1)
```

- Déplacement d'objets de la sélection courante vers un package cible

Vous pouvez déplacer les objets de la sélection courante vers un package cible à l'aide de la méthode suivante de l'objet Object Selection : MoveToPackage(ByVal TargetPackage As BaseObject)

Exemple
Déplacement d'objets de la sélection vers un package cible nommé Pack :

```
MySel.MoveToPackage Pack
```

- Copie d'objet de la sélection courante dans un package cible

Vous pouvez copier les objets de la sélection courante dans un package cible à l'aide la méthode suivante de l'objet Object Selection : CopyToPackage(ByVal TargetPackage As BaseObject)

Exemple
Copie d'objets de la sélection courante dans un package cible nommé Pack :

```
MySel.CopyToPackage Pack
```

- Filtrage d'une liste de sélection par stéréotype

Vous pouvez créer une sélection d'objets et filtrer cette sélection en utilisant un stéréotype. Vous devez pour ce faire utiliser la méthode suivante :

ShowObjectPicker(ByVal ClassNames As String = "", ByVal StereotypeFilter As String = "", ByVal DialogCaption As String = "", ByVal ShowEmpty As Boolean = True, ByVal InModel As Boolean = True) As BaseObject

Exemple
Affiche une boîte de dialogue de sélection permettant de sélectionner une transaction métiers :

```
If Not Fldr is Nothing then
         ' Create a selection object
         Set Sel = Mdl.CreateSelection
         If Not Sel is Nothing then
           'Show the object picker dialog for selecting a BT
            Set Impl = Sel.ShowObjectPicker ("Process",
```

```
"BinaryCollaboration", "Select a Binary Collaboration Process")
            ' Retrieve the selection
            If not Impl is Nothing Then
             If Impl.IsKindOf(PDBPM.Cls_Process) and Impl.Stereotype
= "BinaryCollaboration" then
                  Set Shct = Impl.CreateShortcut (Fldr)
                  If not Shct is Nothing Then
                     obj.Implementer = Shct
                        %Initialize% = True
                  End If
              End If
             End If
         End If
```

Création d'une extension à l'aide de script

Comme n'importe quel objet de PowerAMC, les extensions peuvent être lues, créées et modifiées à l'aide du scripting.

Pour obtenir des informations détaillées sur les extensions, voir *Chapitre 2, Fichiers d'extension* à la page 23.

Le script suivant illustre la façon dont vous pouvez *accéder* à une extension existante, la *parcourir*, y *créer* un attribut étendu au sein de la définition et enfin y *modifier* les valeurs d'attribut étendu. Une fonction est créée afin de permettre de développer les noeuds de catégories qui figurent dans l'arborescence de la boîte de dialogue Propriétés de l'extension.

Exemple

```
Dim M
Set M = ActiveModel
'Récupère la première extension dans le modèle actif
Dim X
Set X = M.ExtendedModelDefinitions.Item(0)
'Développer l'arborescence de catégories en utilisant la fonction
searchObject (détails plus bas)
Dim C
Set C = SearchObject (X.Categories, "Settings")
Set C = SearchObject (C.Categories, "Extended Attributes")
Set C = SearchObject (C.Categories, "Objects")
Set C = SearchObject (C.Categories, "Entity")
'Create extended attribute in the Entity category
Dim A
Set A = C.Categories.CreateNew (cls_ExtendedAttributeTargetItem)
'Définit les propriétés de l'attribut étendu
A.DataType = 10 'integer
A.Value = 10
A.Name = "Z"
A.Code = "Z"
'Récupère la première entité du modèle actif
Dim E
Set E = M.entities.Item(0)
'Récupère les valeurs de l'attribut étendu créé dans un boîte message
msgbox E.GetExtendedAttribute("X.Z")
```

```
'Change les valeurs de l'attribut étendu
E.SetExtendedAttribute "X.Z", 5
'Retrieve the modified values of the extended attribute in a message
box
msgbox E.GetExtendedAttribute("X.Z")
********************
'Détaille la fonction SearchObject qui permet de parcourir une
collection à partir de son nom et l'objet recherché
'* SUB SearchObject
Function SearchObject (Coll, Name)
'For example Coll = Categories and Name = Settings
 Dim Found, Object
 For Each Object in Coll
  If Object.Name = Name Then
   Set Found = Object
  End If
 Next
 Set SearchObject = Found
End Function
```

Mise en correspondance des objets à l'aide de scripts

Vous pouvez utiliser des scripts pour établir des correspondances entre des objets appartenant à des modèles hétérogènes.

Vous créez ou réutilisez une correspondance d'objet à l'aide de la méthode suivante sur l'objet DataSource et l'objet ClassifierMap : CreateMapping(ByVal Object As BaseObject) As BaseObject.

Exemple

Soit l'exemple suivant dans lequel un MOO (oom1) contient une classe (class_1) dotée de deux attributs (att1 et att2) et un MPD (pdm1) qui contient une table (table_1) dotée de deux colonnes (col1 et col2). Pour mettre en correspondance la classe et les attributs du MOO avec la table et les colonnes du MPD, vous devez effectuer les opérations suivantes :

- Créer une source de données dans le MOO

```
set ds = oom1.datasources.createnew
```

- Ajouter le MPD comme source pour la source de données

```
ds.AddSource pdm1
```

- Créer une correspondance pour class_1 et définir cette correspondance comme défaut pour class_1 (la source de donnée courante étant le défaut)

```
set map1 = ds.CreateMapping(class_1)
```

- Ajouter table_1 comme source pour class_1

```
map1.AddSource table_1
```

- Ajouter une correspondance pour att1

```
set attmap1 = map1.CreateMapping(att1)
```

- Définir col1 comme source pour att1

```
attmap1.AddSource col1
```

- Ajouter une correspondance pour att2

```
set attmap2 = map1.CreateMapping(att2)
```

- Définir col2 comme source pour att2

```
attmap.AddSource col2
```

Vous pouvez également retrouver la correspondance d'un objet à l'aide de la méthode suivante sur l'objet DataSource et l'objet ClassifierMap : GetMapping(ByVal Object As BaseObject) As BaseObject.

- Récupérer la correspondance de class_1

```
Set mymap = ds.GetMapping (class_1)
```

- Récupérer la correspondance de att1

```
Set mymap = map1.GetMapping (att1)
```

Pour plus d'informations sur la mise en correspondance d'objets, voir *Guide des fonctionnalités générales > Liaison et synchronisation de modèles > Mise en correspondance d'objets*.

Manipulation de bases de données à l'aide de script

Vous pouvez utiliser des scripts pour manipuler des bases de données dans PowerAMC.

Génération d'une base de données à l'aide de scripts

Lorsque vous devez générer une base de données en utilisant un script, vous pouvez utiliser les méthodes suivantes :

- GenerateDatabase(ByVal ObjectSelection As BaseObject = Nothing)
- GenerateTestData(ByVal ObjectSelection As BaseObject = Nothing)

Dans l'exemple ci-après, vous accomplissez les tâches suivantes :

- Ouvrir un modèle existant.
- Générer un script à partir du modèle
- Modifier le modèle
- Générer un script de base de données modifié
- Générer un jeu de données de test

Ouverture d'un modèle existant

Dans l'exemple ci-après, nous commençons par ouvrir un modèle existant (ASA 9) en utilisant la méthode suivante: OpenModel (filename As String, flags As Long =omf_Default) As BaseObject.

N'oubliez pas d'inclure la barre oblique inverse (\) finale dans le répertoire de destination.

Nous allons générer un script de base de données pour le modèle, modifier le modèle, générer un script de données modifié et générer un jeu de données de test en utilisant respectivement les méthodes suivantes :

- GenerateDatabaseScripts pModel
- ModifyModel pModel
- GenerateAlterScripts pModel
- GenerateTestDataScript pModel

Exemple :

```
Option Explicit
Const GenDir = "D:\temp\test\"
Const Modelfile = "D:\temp\phys.pdm"
Dim fso : Set fso = CreateObject("Scripting.FileSystemObject")
Start
Sub Start()
   dim pModel : Set pModel = OpenModel(Modelfile)
   If (pModel is Nothing) then
      Output "Unable to open the model"
      Exit Sub
   End if
End Sub
```

Génération d'un script pour le modèle

Vous générez ensuite un script pour ce modèle dans le dossier défini dans la constante "GenDir" en utilisant la méthode suivante : GenerateDatabase(ByVal ObjectSelection As BaseObject = Nothing).

Tout comme vous le feriez dans une boîte de dialogue de génération, vous devez définir un répertoire de génération ainsi que le nom du fichier sql avant de lancer la génération, comme dans l'exemple suivant.

Exemple :

```
Sub GenerateDatabaseScripts(pModel)
   Dim pOpts : Set pOpts = pModel.GetPackageOptions()
   InteractiveMode = im_Batch ' Avoid displaying generate window
   ' set generation options using model package options
   pOpts.GenerateODBC = False ' Force sql script generation rather
than
   ' ODBC
   pOpts.GenerationPathName = GenDir ' Define generation directory
   pOpts.GenerationScriptName = "script.sql" ' Define sql file name
```

```
   pModel.GenerateDatabase ' Launch the Generate Database feature
End Sub
```

Modification du modèle

Ensuite, vous modifiez le modèle en ajoutant une colonne à chaque table :

Exemple :

```
Sub ModifyModel(pModel)
   dim pTable, pCol
   ' Add a new column in each table
   For each pTable in pModel.Tables
      Set pCol = pTable.Columns.CreateNew()
      pCol.SetNameAndCode "az" & pTable.Name, "AZ" & pTable.Code
      pCol.Mandatory = False
   Next
End Sub
```

Génération d'un script de base de données modifié

Avant de générer le script de base de données modifiée, vous devez obtenir les options de package et changer les paramètres de génération, puis vous générez le script de base de données en conséquence.

Pour plus d'informations sur les options de génération, voir BasePhysicalPackageOptions dans le fichier d'aide sur les objets du métamodèle.

Exemple :

```
Sub GenerateAlterScripts(pModel)
   Dim pOpts : Set pOpts = pModel.GetPackageOptions()
   InteractiveMode = im_Batch ' Avoid displaying generate window
' set generation options using model package options
   pOpts.GenerateODBC = False ' Force sql script generation rather
than ODBC
pOpts.GenerationPathName = GenDir
   pOpts.DatabaseSynchronizationChoice = 0 'force already saved apm
as source
   pOpts.DatabaseSynchronizationArchive = GenDir & "model.apm"
   pOpts.GenerationScriptName = "alter.sql"
pModel.ModifyDatabase ' Launch the Modify Database feature
End Sub
```

Génération d'un jeu de données de test

Pour finir, vous générez un jeu de données de test :

Exemple :

```
Sub GenerateTestDataScript(pModel)
   Dim pOpts : Set pOpts = pModel.GetPackageOptions()
   InteractiveMode = im_Batch ' Avoid displaying generate window
' set generation options using model package options
   pOpts.TestDataGenerationByODBC = False ' Force sql script
generation rather than ODBC
   pOpts.TestDataGenerationDeleteOldData = False
```

```
pOpts.TestDataGenerationPathName = GenDir
   pOpts.TestDataGenerationScriptName = "Test.sql"
pModel.GenerateTestData ' Launch the Generate Test Data feature
End Sub
```

Génération d'une base de données via une connexion directe à l'aide de scripts

Vous pouvez générer une base de données via ODBC en utilisant un script.

Pour ce faire, vous commencez par vous connecter à la base de données en utilisant la méthode ConnectToDatabase(ByVal Dsn As String, ByVal User As String, ByVal Password As String) As Boolean depuis le modèle, puis vous configurez les options de génération et lancez la fonctionnalité de génération.

Pour plus d'informations sur les options de génération, voir BasePhysicalPackageOptions dans le fichier d'aide sur les objets du métamodèle.

Exemple :

```
Const cnxDSN = "ODBC:ASA 9.0 sample"
Const cnxUSR = "dba"
Const cnxPWD = "sql"

Const GenDir = "C:\temp\"
Const GenFile = "test.sql"
Const ModelFile = "C:\temp\phys.pdm"

set pModel = openModel(ModelFile)

   set pOpts=pModel.GetPackageOptions()

      pModel.ConnectToDatabase cnxDSN, cnxUSR, cnxPWD
      pOpts.GenerateODBC = true

   pOpts.GenerationPathName = GenDir
   pOpts.GenerationScriptName = 'script.sql"
   pModel.GenerateDatabase
```

Génération d'une base de données à l'aide de scripts en utilisant les paramètres et les sélections

Vous pouvez utiliser les paramètres et sélections dans le cadre du scripting avant de lancer la génération de base de données en utilisant les méthodes suivantes dans le modèle : UseSettings(ByVal Function As Long, ByVal Name As String = "") As Boolean et UseSelection(ByVal Function As Long, ByVal Name As String = "") As Boolean.

En utilisant le MPD d'exemple (Gestsoc.MPD) fourni dans le répertoire d'installation PowerAMC, et qui contient deux sélections :

- "Organisation" inclut les tables DIVISION, SALARIE, REGROUPE & EQUIPE.
- "Matériels" inclut les tables COMPOSE, MATERIEL, PROJET & UTILISE.

L'exemple suivant vous montre comment effectuer les tâches suivantes :

- Générer un premier script de ce modèle pour la sélection "Organisation".
- Générer un script de création de données de test pour les tables contenues dans cette sélection.
- Générer un second script de ce modèle pour la sélection "Matériels" ainsi qu'un script de création de données de test pour les tables qu'il contient en utilisant le second jeu de paramètres (setting2).

Exemple :

```
' Generated sql scripts will be created in 'GenDir' directory
' there names is the name of the used selection with extension ".sql"
for DDL scripts
' and extension "_td.sql" for DML scripts (for test data
generations).
Option Explicit

Const GenDir = "D:\temp\test\"

Const setting1 = "Tables & Views (with permissions)"
Const setting2 = "Triggers & Procedures (with permissions)"
Start EvaluateNamedPath("%_EXEMPLES%\gestsoc.mpd")

Sub Start(sModelPath)
   on error resume next
   dim pModel : Set pModel = OpenModel(sModelPath)
   If (pModel is Nothing) then
      Output "Unable to open model " & sModelPath
      Exit Sub
   End if

   GenerateDatabaseScripts pModel, "Organisation" setting1
   GenerateTestDataScript pModel, "Organisation" setting1

   GenerateDatabaseScripts pModel, "Matériels" setting2
   GenerateTestDataScript pModel, "Matériels" setting2
   pModel.Close
   on error goto 0
End Sub

Sub GenerateDatabaseScripts(pModel, sSelectionName, sSettingName)
   Dim pOpts : Set pOpts = pModel.GetPackageOptions()
   InteractiveMode = im_Batch ' Avoid displaying generate window
' set generation options using model package options
   pOpts.GenerateODBC = False ' Force sql script generation rather
than ODBC
   pOpts.GenerationPathName = GenDir
   pOpts.GenerationScriptName = sSelectionName & ".sql"
 ' Launch the Generate Database feature with selected objects
   pModel.UseSelection fct_DatabaseGeneration, sSelectionName
   pModel.UseSetting fct_DatabaseGeneration, sSettingName
   pModel.GenerateDatabase
End Sub

Sub GenerateTestDataScript(pModel, sSelectionName)
   Dim pOpts : Set pOpts = pModel.GetPackageOptions()
```

```
   InteractiveMode = im_Batch ' Avoid displaying generate window
   ' set generation options using model package options
   pOpts.TestDataGenerationByODBC = False ' Force sql script
generation rather than ODBC
   pOpts.TestDataGenerationDeleteOldData = False
   pOpts.TestDataGenerationPathName = GenDir
   pOpts.TestDataGenerationScriptName = sSelectionName & "_td.sql"
' Launch the Generate Test Data feature for selected objects
   pModel.UseSelection fct_TestDataGeneration, sSelectionName
   pModel.GenerateTestData
End Sub
```

Création de sélection et paramètre

Vous pouvez créer une sélection persistante qui peut être utilisée dans la boîte de dialogue de génération en transformant une sélection en sélection persistante.

Exemple :

```
Option Explicit
   Dim pActiveModel
   Set pActiveModel = ActiveModel

   Dim Selection, PrstSel
   Set Selection = pActiveModel.createselection
   Selection.AddActiveSelectionObjects

   Set PrstSel = Selection.CreatePersistentSelectionManager("test")
```

Reverse engineering d'une base de données à l'aide de scripts

Vous procédez au reverse engineering à l'aide de scripts en utilisant la méthode ReverseDatabase(ByVal Diagram As BaseObject = Nothing).

Dans l'exemple suivant, la base de données ODBC est récupérée dans un nouveau MPD.

Les premières lignes du script définissent les constantes utilisées :

- cnxDSN est la chaîne dsn ODBC le chemin d'un fichier dsn ODBC.
- cnxUSR est le nom d'utilisateur de connexion ODBC.
- cnxPWD est le mot de passe de connexion ODBC.

Exemple :

```
option explicit

' To use a user or system datasource, define constant with
"ODBC:<datasourcename>"
' -> Const cnxDSN = "ODBC:ASA 9.0 sample"
' To use a datasource file, define constant with the full path to the
DSN file
' -> Const cnxDSN = "\\romeo\public\DATABASES\_filedsn
\sybase_asa9_sample.dsn"

' use ODBC datasource
Const cnxDSN = "ODBC:ASA 9.0 sample"
```

```
Const cnxUSR = "dba"
Const cnxPWD = "sql"
Const GenDir = "C:\temp\"
Const filename = "D:\temp\phys.pdm"

' Call to main function with the newly created PDM
' This sample use an ASA9 database
Start CreateModel(PdPDM.cls_Model, "|DBMS=Sybase AS Anywhere 9")

Sub Start(pModel)

   If (pModel is Nothing) then
      output "Unable to create a physical model for selected DBMS"
      Exit Sub
   End If

   InteractiveMode = im_Batch

' Reverse database phase
' First connect to the database with connection parameters
   pModel.ConnectToDatabase cnxDSN, cnxUSR, cnxPWD
' Get the reverse option of the model
   Dim pOpt
   Set pOpt = pModel.GetPackageOptions()

' Force ODBC Reverse of all listed objects
   pOpt.ReversedScript = False
   pOpt.ReverseAllTables = true
   pOpt.ReverseAllViews = true
   pOpt.ReverseAllStorage = true
   pOpt.ReverseAllTablespace = true
   pOpt.ReverseAllDomain = true
   pOpt.ReverseAllUser = true
   pOpt.ReverseAllProcedures = true
   pOpt.ReverseAllTriggers = true
   pOpt.ReverseAllSystemTables = true
   pOpt.ReverseAllSynonyms = true
' Go !
   pModel.ReverseDatabase
   pModel.save(filename)
' close model at the end
   pModel.Close false
End Sub
```

Manipulation du référentiel à l'aide de scripts

PowerAMC permet d'accéder aux fonctionnalités via le scripting en utilisant la propriété globale RepositoryConnection as BaseObject.

Il permet de récupérer la connexion au référentiel courante, qui est l'objet qui gère la connexion au serveur de référentiel et fournit l'accès aux documents et objets stockés dans le référentiel.

L'objet *RepositoryConnection* équivaut au noeud racine dans l'Explorateur du référentiel.

Vous pouvez accéder aux documents du référentiel, mais vous ne pouvez pas accéder à l'administration des objets du référentiel, tels que les utilisateurs, groupes, configurations, branches, et listes de verrous.

En outre, seule la dernière version d'un document de référentiel est accessible via le scripting.

Connexion à la base de données du référentiel

Avant que vous ne puissiez établir une connexion à la base du référentiel à l'aide de scripts, votre station de travail doit comporter des définitions de référentiel, car il n'est pas possible de définir une nouvelle définition de référentiel via la fonctionnalité de scripting.

Pour récupérer la connexion au référentiel courante :

Utilisez le code suivant	Description
RepositoryConnection As BaseObject	Propriété globale qui gère la connexion à la base de données du référentiel.

Pour établir une connexion avec une base de données de référentiel :

Utilisez le code suivant	Description
Open (ByVal RepDef As String = "", ByVal User As String = "", ByVal Pass As String = "", ByVal DBUser As String = "", ByVal DBPass As String = "") As Boolean	Méthode portant sur RepositoryConnection qui permet d'établir une connexion au référentiel.

Pour se déconnecter du référentiel :

Utilisez le code suivant	Description
Close()	Méthode portant sur RepositoryConnection qui permet de se déconnecter de la base de données du référentiel.

Vous pouvez établir une connexion à la base du référentiel à l'aide de la méthode suivante sur la propriété globale RepositoryConnection : Open(ByVal RepDef As String = "", ByVal User As String = "", ByVal Pass As String = "", ByVal DBUser As String = "", ByVal DBPass As String = "") As Boolean.

Exemple

```
Dim C
Set C = RepositoryConnection
C.Open
```

Vous interrompez la connexion à la base du référentiel en utilisant la méthode suivante : Close().

Exemple

```
C.Close
```

Accès à un document du référentiel

Vous pouvez accéder aux documents du référentiel situés à la racine du référentiel en utilisant la collection ChildObjects (contenant à la fois des documents et des dossiers) ainsi que dans n'importe lequel de ses sous-dossiers.

Pour parcourir le référentiel à la recherche d'un document :

Utilisez le code suivant	Description
ChildObjects As ObjectCol	Collection sur la classe StoredObject qui gère l'accès aux documents du référentiel.

Pour mettre à jour une version de document :

Utilisez le code suivant	Description
Refresh()	Méthode sur la propriété RepositoryConnection qui permet de visualiser les nouveaux documents, mettre à jour les versions des documents existants ou de dissimuler les documents supprimés.

Pour rechercher un document :

Utilisez le code suivant	Description
FindInRepository() As BaseObject	Méthode sur la classe BaseModel qui permet de vérifier si un modèle a déjà été consolidé.

Les documents du référentiels sont les suivants :

Document du référentiel	Description
RepositoryModel	Contient tout type de modèle PowerAMC.
RepositoryReport	Contient des rapports multimodèle consolidés.
RepositoryDocument	Contient des fichiers autre que PowerAMC (texte, Word, ou Excel).
OtherRepositoryDocument	Contient des fichiers autre que PowerAMC définis à l'aide de l'interface de référentiel Java qui permet de définir des métamodèles.

Vous pouvez accéder à un document RepositoryModel et à ses sous-objets à l'aide de la collection suivante : ChildObjects As ObjectCol.

Exemple

```
' Retrieve the deepest folder under the connection
 Dim CurrentObject, LastFolder
 set LastFolder = Nothing
 for each CurrentObject in C.ChildObjects
 if CurrentObject.IsKindOf(cls_RepositoryFolder) then
   set LastFolder = CurrentObject
  end if
 next
```

La collection ChildObjects n'est pas automatiquement mise à jour lorsque le référentiel est modifié au cours d'une exécution de script. Pour rafraîchir toutes les collections, vous pouvez utiliser la méthode suivante : Refresh().

Exemple

```
C.Refresh
```

Vous pouvez déterminer si la consolidation d'un modèle a déjà été effectuée en utilisant la méthode suivante : FindInRepository() As BaseObject.

Exemple

```
Set repmodel = model.FindInRepository()
If repmodel Is Nothing Then
  ' Model was not consolidated yet...
 model.ConsolidateNew
Else
 ' Model was already consolidated...
 repmodel.Freeze
 model.Consolidate
End If
```

Extraction d'un document de référentiel

Vous pouvez extraire un document de référentiel à l'aide de scripts de l'une des façons suivantes :

- La façon générique qui est applicable à tout type de document
- La façon spécifique qui n'est applicable qu'aux documents RepositoryModel et RepositoryReport

Pour extraire tout type de document :

Utilisez le code suivant	Description
ExtractToFile(ByVal FileName As String, ByVal MergeMode As Long = 2, ByVal OpenMode As Boolean = -1, ByRef Actions As String = NULL, ByRef Conflicts As String = NULL) As BaseObject	Méthode sur la classe RepositoryModel qui permet d'extraire tout type de document.

Pour extraire un document PowerAMC :

Utilisez le code suivant	Description
UpdateFromRepository(ByVal MergeMode As Integer = 2, ByRef actions As String = NULL, ByRef conflicts As String = NULL) As Boolean	Méthode sur la classe BaseModel qui permet d'extraire des documents PowerAMC.

La façon générique

Pour extraire un document de référentiel, vous devez :

- Rechercher un document de référentiel à l'aide de la collection ChildObjects
- Extraire le document à l'aide de la méthode suivante : ExtractToFile (ByVal FileName As String, ByVal MergeMode As Long = 2, ByVal OpenMode As Boolean = -1, ByRef Actions As String = NULL, ByRef Conflicts As String = NULL) As BaseObject

Exemple

```
set C = RepositoryConnection
C.Open
Dim D, P
set P = Nothing
for each D in C.ChildObjects
if D.IsKindOf (cls_RepositoryModel) then
D.ExtractToFile ("C:\temp\OO.MOO")
end if
next
```

La façon spécifique :

Pour extraire un document RepositoryModel ou RepositoryReport, vous devez :

- Récupérer le document depuis le modèle local ou le rapport multimodèle, (à la condition qu'ils aient déjà été consolidés) à l'aide de la méthode suivante : UpdateFromRepository (ByVal MergeMode As Integer = 2, ByRef actions As String = NULL, ByRef conflicts As String = NULL) As Boolean

Exemple

```
set MyModel = OpenModel ("C:\temp\OO3.MOO")
MyModel.UpdateFromRepository
```

Consolidation d'un document de référentiel

Vous pouvez consolider un document de référentiel à l'aide de scripts de l'une des façons suivantes :

- La façon générique qui est applicable à tout type de document
- La façon spécifique qui n'est applicable qu'aux documents RepositoryModel et RepositoryReport

Pour consolider tout type de document :

Utilisez le code suivant	Description
ConsolidateDocument(ByVal FileName As String, ByVal MergeMode As Long = 2, ByRef Actions As String = NULL, ByRef Conflicts As String = NULL) As BaseObject	Méthode sur la classe RepositoryFolder qui permet de consolider tout type de document.

Pour consolider un document PowerAMC :

Utilisez le code suivant	Description
ConsolidateNew(ByVal RepositoryFolder As BaseObject, ByRef actions As String = NULL, ByRef conflicts As String = NULL) As BaseObject	Méthode sur la classe BaseModel qui permet de consolider des documents PowerAMC.
Consolidate(ByVal MergeMode As Integer = 2, ByRef actions As String = NULL, ByRef conflicts As String = NULL) As BaseObject	Méthode sur la classe BaseModel qui permet de consolider des versions de référentiel supplémentaires d'un document PowerAMC.

La façon générique

Pour consolider un document de référentiel, vous devez :

- Spécifier un nom de fichier lorsque vous utilisez la méthode suivante : ConsolidateDocument (ByVal FileName As String, ByVal MergeMode As Long = 2, ByRef Actions As String = NULL, ByRef Conflicts As String = NULL) As BaseObject)

Exemple :

```
set C = RepositoryConnection
C.open
C.ConsolidateDocument ("c:\temp\test.txt")
```

La façon spécifique

Pour consolider un document RepositoryModel ou RepositoryReport, vous pouvez utiliser l'une des méthodes suivantes :

- ConsolidateNew (ByVal RepositoryFolder As BaseObject, ByRef actions As String = NULL, ByRef conflicts As String = NULL) As BaseObject, pour consolider la première version de référentiel d'un document
- Consolidate (ByVal MergeMode As Integer = 2, ByRef actions As String = NULL, ByRef conflicts As String = NULL) As BaseObject, pour consolider les versions supplémentaires d'un document

Exemples :

```
Set model = CreateModel(PdOOM.cls_Model, "|Diagram=ClassDiagram")
set C = RepositoryConnection
```

```
C.Open
model.ConsolidateNew c
```

```
set C = RepositoryConnection
C.Open
model.Consolidate
```

Notions de base relatives au mode de résolution des conflits

Si vous mettez à jour un document qui a déjà été modifié depuis la dernière extraction ou consolidation, un conflit peut survenir.

Conflits de consolidation

Vous pouvez résoudre les conflits survenant lors de la consolidation d'un document de référentiel en spécifiant un mode de fusion en second paramètre de la méthode suivante : ConsolidateDocument(ByVal FileName As String, ByVal MergeMode As Long = 2, ByRef Actions As String = NULL, ByRef Conflicts As String = NULL) As BaseObject.

Ce paramètre (ByVal MergeMode As Long = 2) peut prendre les valeurs suivantes :

Value	Description
1	Remplace le document dans le référentiel par le document local sans conserver aucune modification effectuée dans le document de référentiel.
2 (valeur par défaut)	Tente de sélectionner automatiquement les actions de fusion par défaut en prenant en compte les dates de modifications des objets et annule la consolidation en cas de conflit (objets modifiés à la fois localement et dans le référentiel).
3	Sélectionne les actions de fusion par défaut, mais favorise toujours les changements locaux en cas de conflit au lieu d'annuler la consolidation.
4	Sélectionne les actions de fusion par défaut, et favorise les modifications du document de référentiel, en cas de conflit.

Les actions de fusion effectuées au cours de la consolidation et les conflits qui ont pu survenir peuvent être récupérés au sein de la chaîne de caractères spécifiée en troisième et quatrième paramètres : ByRef Actions As String = NULL and ByRef Conflicts As String = NULL.

Conflits d'extraction

Vous pouvez résoudre les conflits survenant lors de l'extraction d'un document de référentiel en spécifiant un mode de fusion en second paramètre de la méthode suivante : ExtractToFile(ByVal FileName As String, ByVal MergeMode As Long = 2, ByVal OpenMode As Boolean = -1, ByRef Actions As String = NULL, ByRef Conflicts As String = NULL) As BaseObject.

Ce paramètre (ByVal MergeMode As Long = 2) peut prendre les valeurs suivantes :

Valeur	Description
0	Extrait le document sans fusion, efface ainsi le document existant localement, le cas échéant, et met le document extrait en lecture-seule.
1	Extrait le document sans fusion, efface ainsi le document existant localement, le cas échéant.
2 (default value)	Tente de sélectionner automatiquement les actions de fusion par défaut en prenant en compte les dates de modifications des objets et annule l'extraction en cas de conflit (objets modifiés à la fois localement et dans le référentiel).
3	Sélectionne les actions de fusion par défaut, mais favorise toujours les changements locaux en cas de conflit au lieu d'annuler l'extraction.
4	Sélectionne les actions de fusion par défaut, et favorise les modifications du document de référentiel, en cas de conflit.

Les actions de fusion effectuées au cours de l'extraction et les conflits qui ont pu survenir peuvent être récupérées au sein de la chaîne de caractères spécifiée en quatrième et cinquième paramètres : ByRef Actions As String = NULL and ByRef Conflicts As String = NULL. Le troisième paramètre (ByVal OpenMode As Boolean = -1) vous permet de conserver ouvert le modèle extrait.

Gestion des versions d'un document

Vous pouvez gérer des versions de document à l'aide de scripts.

Pour geler et dégeler une version de document :

Utilisez le code suivant	Description
Freeze(ByVal Comment As String = "") As Boolean	Méthode sur la classe RepositoryDocumentBase qui permet de créer une version archivée d'un document.
Unfreeze() As Boolean	Méthode sur la classe RepositoryDocumentBase qui permet de modifier la version courante dans le référentiel pour refléter les changements effectués sur votre machine en local.

Par exemple :

```
MyDocument.Freeze "Update required"
```

```
MyDocument.Unfreeze
```

Pour verrouiller et déverrouiller une version de document :

Utilisez le code suivant	Description
Lock(ByVal Comment As String = "") As Boolean	Méthode sur la classe RepositoryDocumentBase qui permet d'empêcher d'autres utilisateurs de mettre à jour une version consolidée.
Unlock() As Boolean	Méthode sur la classe RepositoryDocumentBase qui permet à d'autres utilisateurs de mettre à jour une version consolidée.

Par exemple :

```
MyDocument.Lock "Protection required"
```

```
MyDocument.Unlock
```

Pour supprimer une version de document :

Utilisez le code suivant	Description
DeleteVersion() As Boolean	Méthode sur la classe RepositoryDocumentBase qui permet de supprimer une version de document.

Par exemple :

```
MyDocument.Delete
```

Gestion de l'explorateur du référentiel

L'explorateur de référentiel permet d'effectuer des opérations sur les dossiers à l'aide de scripts.

Pour créer un dossier :

Utilisez le code suivant	Description
CreateFolder(ByVal FolderName As String) As BaseObject	Méthode sur la classe RepositoryFolder qui permet de créer un nouveau dossier dans l'explorateur du référentiel.

Par exemple :

```
RepositoryConnection.CreateFolder("VBTest")
```

Pour supprimer un dossier vide :

Utilisez le code suivant	Description
DeleteEmptyFolder() As Boolean	Méthode sur la classe RepositoryFolder qui permet de supprimer un dossier vide dans l'explorateur du référentiel.

Pour plus d'informations sur les documents, voir *Accès à un document du référentiel* à la page 396.

Par exemple :

```
Dim C
Set C = RepositoryConnection
C.Open "MyRepDef"
' Retrieve the deepest folder under the connection
Dim D, P
 set P = Nothing
 for each D in C.ChildObjects
  if D.IsKindOf (cls_RepositoryFolder) then
    D.DeleteEmptyFolder
    c.refresh
  end if
 next
```

Gestion des rapports l'aide de scripts

Vous pouvez générer des rapports HTML et RTF à l'aide de VBScript, mais vous ne pouvez pas créer de rapport.

Accès à un rapport portant sur un modèle à l'aide de scripts

Vous pouvez parcourir un rapport portant sur un modèle à l'aide de la collection suivante sur la classe BaseModelReport : Reports As ObjectCol.

Exemple

```
set m = ActiveModel
For each Report in m.Reports
Output Report.name
```

Récupération d'un rapport multimodèle à l'aide de scripts

Vous pouvez récupérer un rapport multimodèle à l'aide de la fonction suivante : OpenModel(filename As String, flags As Long =omf_Default) As BaseObject

Exemple

```
OpenModel ("c:\temp\mmr1.mmr")
```

Génération d'un modèle HTML à l'aide de scripts

Vous pouvez générer en HTML un rapport pour un modèle ou un rapport multimodèle à l'aide de la méthode suivante sur la classe BaseModelReport : GenerateHTML(ByVal FileName As String) As Boolean.

Exemple

```
set m = ActiveModel
For each Report in m.Reports
   Filename = Report.name & ".htm"
   Report.GenerateHTML (filename)
Next
```

Génération d'un modèle RTF à l'aide de scripts

Vous pouvez générer en RTF un rapport pour un modèle ou un rapport multimodèle à l'aide de la méthode suivante sur la classe BaseModelReport : GenerateRTF(ByVal FileName As String) As Boolean

Exemple

```
set m = ActiveModel
For each Report in m.Reports
   Filename = Report.name & ".rtf"
   Report.GenerateRTF (filename)
Next
```

Accès aux métadonnées à l'aide de scripts

Vous pouvez accéder aux objets internes de PowerAMC et les manipuler à l'aide de Visual Basic Scripting. Les scripts permettent d'accéder aux propriétés, collections et méthodes d'objet et de les modifier en utilisant le nom public de ces objets.

Le métamodèle PowerAMC fournit des informations utiles relatives à ces objets :

Information	Description	Example
Nom public	Le nom et le code des objets du métamodèle sont les noms publics des objets internes de PowerAMC.	AssociationLinkSymbol ClassMapping CubeDimensionAssociation

Information	Description	Example
Collections d'objets	Vous pouvez identifier les collections d'une classe en observant les associations liées à cette classe dans le diagramme. Le rôle de chaque association est le nom de la collection.	Dans PdBPM, il existe une association entre les classes MessageFormat et MessageFlow. Le nom public de cette association est Format. Le rôle de cette association est Usedby, qui correspond à la collection de messages de la classe MessageFormat.
Attributs d'objet	Vous pouvez afficher les attributs d'une classe avec les attributs que cette classe hérite d'une autre classe via des liens de généralisation.	Dans PdCommon, dans le diagramme Common Instantiable Objects, vous pouvez afficher les objets BusinessRule, ExtendedDependency et FileObject avec leurs propres attributs, ainsi que les classes abstraites dont ils héritent les attributs via des liens de généralisation.
Opérations d'objet	Les opérations dans des classes d'un métamodèle correspondent aux méthodes objet utilisées dans VBS.	BaseModel contient l'opération Compare qui peut être utilisée dans VBS.
Stéréotype <<notScriptable>>	Objets qui ne prennent pas en charge les scripts VB qui ont le stéréotype <<notScriptable>>.	CheckModelInternalMessage FileReportItem

PowerAMC vous permet d'accéder aux métadonnées via VBScript à l'aide de la propriété globale MetaModel As BaseObject. Il n'existe qu'une seule instance du métamodèle à laquelle vous pouvez accéder depuis n'importe où par le biais de la propriété globale Application.MetaModel.

Cette fonctionnalité générique permet d'accéder à l'objet MetaModel de façon générique et implique un code neutre que vous pouvez utiliser pour tout type de modèle. Par exemple, vous pouvez l'utiliser pour chercher le dernier objet modifié dans un modèle donné.

Toutes les propriétés et collections des métadonnées sont en lecture seule.

Accès aux objets de métadonnées à l'aide de scripts

Vous pouvez accéder aux objets des métadonnées à l'aide de scripts :

Utilisez le code suivant	Description
MetaModel As BaseObject	Propriété globale. Point d'entrée pour accéder aux objets métadonnées.

Récupération de la version du métamodèle à l'aide de scripts

Vous pouvez extraire la version du métamodèle à l'aide de scripts :

Utilisez le code suivant	Description
Version As String	Propriété. Permet d'extraire la version du métamodèle.

Extraction des types de bibliothèques de métaclasse à l'aide de scripts

Vous pouvez extraire les types de bibliothèques de métaclasse disponibles à l'aide de scripts :

Utilisez le code suivant	Description
MetaLibrary	Collection. Permet d'extraire les types de bibliothèques de métaclasses disponibles pour un module donné.

Accès à la métaclasse d'un objet à l'aide de scripts

Vous pouvez utiliser des scripts pour accéder aux métaclasses d'objet.

Vous pouvez accéder à la métaclasse d'un objet à l'aide de scripts :

Utilisez le code suivant	Description
MetaClass As BaseObject	Propriété. Fournit l'accès à la métaclasse de chaque objet.

Vous pouvez accéder à la métaclasse d'un objet en utilisant son nom public à partir du métamodèle à l'aide de scripts :

Utilisez le code suivant	Description
GetMetaClassByPublicName (ByVal name As String) As BaseObject	Méthode. Fournit l'accès à la métaclasse d'un objet en utilisant son nom public.

Vous pouvez accéder au métaattribut et métacollection d'une métaclasse en utilisant son nom public (à partir de la métaclasse) :

Utilisez le code suivant	Description
GetMetaMemberByPublicName (ByVal name As String) As BaseObject	Méthode. Fournit l'accès à un métaattribut ou à une métacollection en utilisant son nom public.

Extraction des enfants d'une métaclasse à l'aide de scripts

Vous pouvez extraire les enfants d'une métaclasse à l'aide de scripts :

Utilisez le code suivant	Description
Children As ObjectSet	Collection. Lists the MetaClasses that inherit from the parent MetaClass.

Gestion de l'espace de travail à l'aide de scripts

L'objet *Workspace* correspond au noeud racine Espace de travail dans l'explorateur d'objets. PowerAMC vous permet d'accéder aux fonctionnalités de l'espace de travail courant en utilisant la propriété globale ActiveWorkspace As BaseObject.

Chargement, enregistrement et fermeture d'un espace de travail à l'aide de scripts

Les méthodes suivantes permettent de charger, enregistrer et fermer un espace de travail à l'aide de scripts:

Pour charger un espace de travail :

Utilisez le code suivant	Description
Load (ByVal filename As String = "") As Boolean	Charge l'espace de travail depuis un emplacement donné.

Pour enregistrer un espace de travail :

Utilisez le code suivant	Description
Save (ByVal filename As String = "") As Boolean	Enregistre l'espace de travail à un emplacement donné.

Pour fermer un espace de travail :

Utilisez le code suivant	Description
Close ()	Ferme l'espace de travail courant.

Manipulation du contenu d'un espace de travail à l'aide de scripts

Vous pouvez également manipuler le contenu d'un espace de travail à l'aide des éléments suivants :

- Le *WorkspaceDocument* qui correspond aux documents que vous pouvez ajouter dans l'espace de travail. Il peut contenir des *WorkspaceModel* (modèles attachés à un espace de travail) et des *WorkspaceFile* (fichiers externes attachés à un espace de travail).
- Le *WorkspaceFolder* qui correspond aux dossiers de l'espace de travail. Vous pouvez les créer, les supprimer et les renommer. Vous pouvez également ajouter des documents dans les dossiers.

Vous pouvez utiliser la méthode AddDocument(ByVal filename As String, ByVal position As Long = -1) As BaseObject sur un WorkspaceFolder pour ajouter des documents à l'espace de travail.

Exemple de manipulation d'espace de travail :

```
Option Explicit
' Close existing workspace and save it to Temp
Dim workspace, curentFolder
Set workspace = ActiveWorkspace
workspace.Load "%_EXAMPLES%\mywsp.sws"
Output "Saving current workspace to ""Example directory :
"+EvaluateNamedPath("%_EXAMPLES%\temp.sws")
workspace.Save "%_EXAMPLES%\Temp.SWS"
workspace.Close
workspace.Name = "VBS WSP"
workspace.FileName = "VBSWSP.SWS"
workspace.Load "%_EXAMPLES%\Temp.SWS"
dim Item, subitem
for each Item in workspace.children
   If item.IsKindOf(PdWsp.cls_WorkspaceFolder) Then
      ShowFolder (item)
      renameFolder item,"FolderToRename", "RenamedFolder"
      deleteFolder item,"FolderToDelete"
      curentFolder = item
   ElsIf item.IsKindOf(PdWsp.cls_WorkspaceModel) Then
   ElsIf item.IsKindOf(PdWsp.cls_WorkspaceFile) Then
   End if
next
 Dim subfolder
'insert folder in root
 Set subfolder =
workspace.Children.CreateNew(PdWsp.cls_WorkspaceFolder)
 subfolder.name = "Newfolder(VBS)"
 'insert folder in root at pos 6
 Set subfolder = workspace.Children.CreateNewAt(5,
PdWsp.cls_WorkspaceFolder)
 subfolder.name = "Newfolder(VBS)insertedAtPos5"'
   ' add a new folder in this folder
 Set subfolder =
subfolder.Children.CreateNew(PdWsp.cls_WorkspaceFolder)
 subfolder.name = "NewSubFolder(VBS)"
```

```
 subfolder.AddDocument EvaluateNamedPath("%_EXAMPLES%\pdmrep.rtf")
 subfolder.AddDocument EvaluateNamedPath("%_EXAMPLES%\cdmrep.rtf")
 subfolder.AddDocument EvaluateNamedPath("%_EXAMPLES%\project.pdm")
 subfolder.AddDocument EvaluateNamedPath("%_EXAMPLES%\demo.oom")
 dim lastmodel
 set lastmodel = subfolder.AddDocument
(EvaluateNamedPath("%_EXAMPLES%\Ordinateurs.fem"))
 lastmodel.open
 lastmodel.name = "Computers"
 lastmodel.close
 'detaching model from workspace
 lastmodel.delete
workspace.Save "%_EXAMPLES%\Final.SWS"
```

Communication avec PowerAMC à l'aide de OLE Automation

OLE Automation (ou Visual Basic for Application) est un moyen de communiquer avec PowerAMC à partir d'une autre application grâce à l'architecture COM au sein de la même application ou au sein d'autres applications. Vous pouvez écrire un programme à l'aide de n'importe quel langage prenant en charge COM, par exemple les macros de Word et Excel, VB, C++ ou PowerBuilder.

Des exemples de OLE Automation pour différents langages figurent dans le répertoire OLE Automation de PowerAMC.

Différences entre VBScript et OLE Automation

Les programmes VBScript et les programmes OLE Automation sont très similaires. Vous pouvez aisément créer des programmes VB ou VBA, si vous savez utiliser VBScript. Cependant il existe quelques différences. L'exemple de programme suivant révèle ce qui différencie OLE Automation de VBScript.

Programme VBScript

Le programme VBScript suivant vous permet de dénombrer les classes définies dans un MOO et d'afficher leur nombre dans la fenêtre Résultats de PowerAMC, puis de créer un autre MOO et d'afficher son nom dans cette même fenêtre.

Pour cela, les étapes suivantes sont nécessaires :

- Récupérer le modèle actif courant à l'aide de la fonction globale ActiveModel.
- Vérifier l'existence d'un MOO actif.
- Dénombrer les classes dans le MOO actif et afficher un message dans la fenêtre Résultats.
- Créer un nouveau MOO et afficher son nom dans la fenêtre Résultats.

```
'* Purpose:  This script displays the number of classes defined in an
OOM in the output window.
Option Explicit
' Main function
```

```
' Get the current active model
Dim model
Set model = ActiveModel
If model Is Nothing Then
   MsgBox "There is no current model."
ElsIf Not Model.IsKindOf(PdOOM.cls_Model) Then
   MsgBox "The current model is not an OOM model."
Else
   ' Display the number of classes
   Dim nbClass
   nbClass = model.Classes.Count
  Output "The model '" + model.Name + "' contains " + CStr(nbClass) +
" classes."
' Create a new OOM
   Dim model2
   set model2 = CreateModel(PdOOM.cls_Model)
   If Not model2 Is Nothing Then
      ' Copy the author name
      model2.author = model.author
      ' Display a message in the output window
      Output "Successfully created the model '" + model2.Name + "'."
   Else
      MsgBox "Cannot create an OOM."
   End If
End If
```

Programme OLE Automation

Pour faire de même avec un programme OLE Automation, vous devez le modifier de la manière suivante :

- Ajouter la définition de l'application PowerAMC.
- Invoquer la fonction CreateObject pour créer une instance de l'objet PowerAMC Application.
- Préfixer toutes les fonctions globales (ActiveModel, Output, CreateModel) par l'objet PowerAMC Application.
- Libérer l'objet PowerAMC Application.
- Spécifier des types pour les variables "model" et "model2".

```
'* Purpose:  This script displays the number of classes defined in an
OOM in the output window.
Option Explicit
' Main function
Sub VBTest()
   ' Defined the PowerDesigner Application object
   Dim PD As PdCommon.Application
   ' Get the PowerDesigner Application object
   Set PD = CreateObject("PowerDesigner.Application")
' Get the current active model
   Dim model As PdCommon.BaseModel
   Set model = PD.ActiveModel
   If model Is Nothing Then
      MsgBox "There is no current model."
   ElsIf Not model.IsKindOf(PdOOM.cls_Model) Then
      MsgBox "The current model is not an OOM model."
```

```
   Else
      ' Display the number of classes
      Dim nbClass
      nbClass = Model.Classes.Count
      PD.Output "The model '" + model.Name + "' contains " +
CStr(nbClass) + " classes."
' Create a new OOM
      Dim model2 As PdOOM.Class
      Set model2 = PD.CreateModel(PdOOM.cls_Model)
      If Not model2 Is Nothing Then
         ' Copy the author name
         model2.Author = Model.Author
         ' Display a message in the output window
        PD.Output "Successfully created the model '" + model2.Name +
"'."
      Else
         MsgBox "Cannot create an OOM."
      End If
   End If
' Release the PowerDesigner Application object
   Set PD = Nothing
End Sub
```

Préparation pour OLE Automation

Pour permettre à OLE Automation de communiquer avec PowerAMC, vous avez besoin d'effectuer les opérations suivantes :

- Créer une instance de l'objet PowerAMC Application.
- Préfixer toutes les fonctions globales par l'objet PowerAMC Application.
- Libérer l'objet PowerAMC Application avant de fermer le programme.
- Spécifier le type des objets toutes les fois où c'est possible (Dim obj As <ObjectType>).
- Adapter les constantes de classe d'objets au language utilisé lors de la création de l'objet.
- Ajouter des références aux bibliothèques de type d'objets que vous devez utiliser.

Création de l'objet PowerAMC Application

Le programme d'installation inscrit l'objet PowerAMC Application par défaut.

Vous devez vérifier que la variable retournée est vide.

Lorsque vous créez l'objet PowerAMC Application, l'instance courante de PowerAMC sera utilisée, autrement PowerAMC sera lancé.

Si PowerAMC est lancé lorsque vous créez l'objet PowerAMC Application, PowerAMC sera fermé lorsque vous libérerez l'objet PowerAMC Application.

Vous créez l'objet PowerAMC Application, à l'aide de la méthode suivante dans Visual Basic : CreateObject(ByVal Kind As Long, ByVal ParentCol As String = "", ByVal Pos As Long = -1, ByVal Init As Boolean = -1) As BaseObject

Exemple

```
' Defined the PowerDesigner Application object
   Dim PD As PdCommon.Application
   ' Get the PowerDesigner Application object
   Set PD = CreateObject("PowerDesigner.Application")
```

Numéro de version de PowerAMC

Si vous souhaitez vous assurer que l'application fonctionne avec une version particulière de PowerAMC, vous devez saisir le numéro de version dans les commandes de création de l'objet PowerAMC application :

```
' Defined the PowerDesigner Application object
   Dim PD As PdCommon.Application
   ' Get the PowerDesigner Application object
   Set PD = CreateObject("PowerDesigner.Application.x")
'x represents the version number
```

Si vous n'utilisez aucune fonctionnalité particulière de PowerAMC, votre application peut fonctionner avec n'importe quelle version de PowerAMC et il n'est pas utile de spécifier un numéro de version. Dans ce cas, c'est la dernière version installée qui est utilisée.

Remarque : Vous devez libérer l'objet PowerAMC Application avant de quitter l'application dans laquelle vous l'utilisez. Pour cela, vous utilisez la syntaxe suivante : Set Pd = Nothing.

Spécification du type d'objet

Lorsque vous créez des programmes VB ou VBA, il est fortement recommandé de spécifier le type des objets.

Par exemple il est préférable d'utiliser la syntaxe suivante :

```
Dim cls As PdOOM.Class
```

Plutôt que la syntaxe ci-dessous :

```
Dim cls
```

Si vous ne spécifiez pas le type des objets, vous pouvez rencontrer des problèmes lors de l'exécution du programme, qui peuvent s'avérer très difficiles à résoudre par la suite.

Raccourcis

Si le modèle contient des raccourcis, il est recommandé d'utiliser la déclaration suivante Dim obj as PdCommon.IdentifiedObject.

Si le modèle cible est fermé, vous obtiendrez une erreur d'exécution.

Adaptation de la syntaxe des constantes de classe d'objets au langage

Lorsque vous créez un objet à l'aide de VBScript, vous indiquez la constante de classe de l'objet à créer de la façon suivante :

```
Dim cls
Set cls = model.CreateObject(PdOOM.cls_Class)
```

Cette syntaxe fonctionne correctement pour VBScript, VBA et VB mais elle ne fonctionne pas pour les autres langages. En effet, les constantes de classe d'objets sont définies comme une énumération. Seuls les langages qui prennent en charge les énumérations définies en dehors d'une classe peuvent utiliser cette syntaxe.

Pour C# et VB.NET, vous pouvez utiliser la syntaxe suivante :

```
Dim cls As PdOOM.Class
Set cls = model.CreateObject(PdOOM.PdOOM_Classes.cls_Class)
'Where PdOOM_Classes is the name of the enumeration.
```

Pour les autres langages, tels que JavaScript ou PowerBuilder, vous devez définir des constantes qui représentent les objets que vous souhaitez créer.

Pour une liste exhaustive des constantes de classes, voir le fichier VBScriptConstants.vbs qui figure dans le répertoire OLE Automation de PowerAMC.

Ajout de références aux bibliothèques de type d'objet

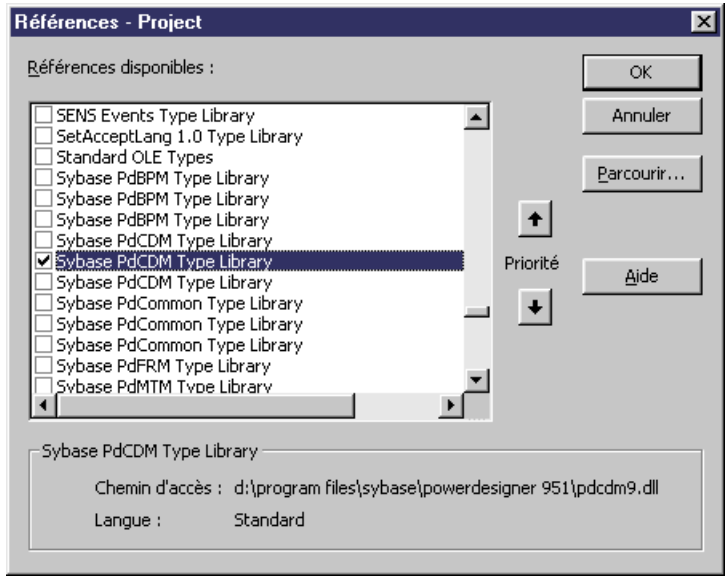
Vous devez ajouter des références aux bibliothèques des types d'objets de PowerAMC que vous souhaitez utiliser, telles que Sybase PdCommon, Sybase PdOOM, Sybase PdPDM, etc. pour les programmes VB, VBA, VB .NET et C# afin que les programmes reconnaissent les objets utilisés.

Pour ajouter des références aux bibliothèques de type d'objet dans un éditeur VBA :
Sélectionnez **Outils** > **Références**.

Pour ajouter des références aux bibliothèques de type d'objet dans un éditeur Visual Basic :
Sélectionnez **Projet** > **Références**.

Pour ajouter des références aux bibliothèques de type d'objet dans un éditeur C# et VB.NET :
Pointez sur le projet dans l'explorateur de projet, cliquez le bouton droit de la souris, puis sélectionnez Ajouter des références.

Exemple d'une fenêtre Références pour un programme VBA dans Word :



Personnalisation des menus PowerAMC à l'aide de compléments

Un complément est un module qui ajoute une fonctionnalité ou un service particulier au comportement standard de PowerAMC. Les compléments de PowerAMC vous permettent de personnaliser les menus de PowerAMC, en y ajoutant vos propres commandes. Vous pouvez personnaliser les menus suivants :

- Tous les menus contextuels des objets qui sont disponibles depuis l'Explorateur d'objets ou depuis un symbole dans le diagramme.
- La majorité des menus de chacun des modules à partir de chacun des types de diagrammes (par exemple : Importer, Exporter, Reverse engineering, Outils, Aide).

Vous pouvez ajouter les éléments de menus suivants :

- Commandes lançant un script de méthode défini à l'aide de VBScript.
- Sous-menus qui sont des menus apparaissant sous un élément de menu.
- Séparateur qui sont des lignes utilisées pour organiser les commandes dans les menus.

Vous pouvez utiliser les types de complément suivants pour créer des éléments de menu dans PowerAMC :

- Commandes personnalisées- pour lancer des programmes exécutables ou des scripts VB à l'aide de la boîte de dialogue Personnaliser les commandes depuis le menu Outils. Les commandes que vous définissez ne peuvent s'afficher comme des sous-menus que dans les éléments de menu Exécuter des commandes et dans les éléments de menu Importer et

Exporter du menu Fichier mais pas dans les menus contextuels des objets. Vous pouvez masquer l'affichage de ces sous-menus dans le menu, tout en conservant leur définition. Pour plus d'informations, voir *Création de commandes personnalisées dans le menu Outils* à la page 415.

- Fichiers de ressources – pour définir des commandes pour une cible particulière. Les méthodes et menus sont créés dans le fichier de ressource, dans la catégorie Profile située sous la métaclasse appropriée. Vous pouvez filtrer ces méthodes Pilotage de PowerAMC à l'aide de scripts et menus à l'aide d'un stéréotype ou d'un critère. Cependant le fichier de ressource doit toujours être associé au modèle pour que la commande définie puisse s'afficher. Pour plus d'informations, voir *Menus (Profile)* à la page 110.
- ActiveX – lorsque vous souhaitez mettre en oeuvre des une interaction plus complexe entre lui-même et PowerAMC, comme activer et désactiver des éléments de menus basés sur une sélection d'objets, interagir avec l'environnement d'affichage des fenêtres ou pour les plug-ins écrits dans d'autres langages tels que Visual Basic.NET or C++. Pour plus d'informations, voir *Création d'un complément ActiveX* à la page 423.
- Fichier XML – lorsque vous souhaitez définir plusieurs commandes qui seront toujours disponibles indépendamment de la cible que vous sélectionnez. Ce fichier XML contient un programme déclaratif simple avec un langage lié à un fichier .EXE ou à un script VB. Les commandes liées aux mêmes applications (par exemple ASE, IQ etc.) devraient être regroupées au sein du même fichier XML. Pour plus d'informations, voir *Création d'un complément fichier XML* à la page 425.

Remarque : La syntaxe XML d'un menu défini dans la page Menu de l'éditeur de ressources est la même pour un fichier XML et un complément ActiveX. Vous pouvez utiliser l'interface de l'éditeur de ressources pour visualiser dans la page XML la syntaxe d'un menu que vous avez créé dans la page Menu et qui vous aidera à construire la même syntaxe XML dans votre ActiveX ou fichier XML. Pour plus d'informations, voir *Création d'un complément fichier XML* à la page 425.

Création de commandes personnalisées dans le menu Outils

Vous pouvez créer vos propres éléments de menu depuis le menu Outils de PowerAMC pour accéder aux objets de PowerAMC en utilisant vos propres scripts.

Depuis le menu Outils de l'application, vous pouvez ajouter vos propres entrées de sous-menus qui vous permettront d'exécuter les commandes suivantes :

- Programmes exécutables
- Scripts VB

Vous pouvez également réunir des commandes au sein de sous-menus, modifier les commandes existantes et leur affecter des raccourcis clavier.

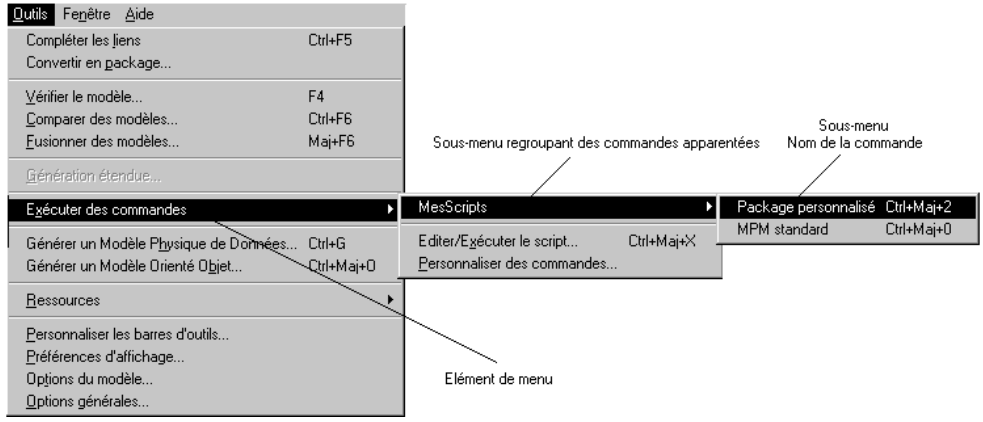
Définition d'une commande personnalisée

Vous pouvez définir des commandes dans la boîte de dialogue Personnaliser des commandes. Le nombre de commandes que vous pouvez définir est limité à 256.

Lorsque vous définissez une commande, le nom que vous saisissez pour la commande s'affiche comme une entrée de sous-menu de l'élément de menu Exécuter des commandes. Les noms des commandes apparaissent triées alphabétiquement.

Vous pouvez définir un contexte pour cette commande afin qu'elle soit dépendante du diagramme et ne s'affiche que dans les cas appropriés.

L'illustration suivante montre le résultat de définitions de commandes effectuées dans la boîte de dialogue Personnaliser des commandes.



Pour définir une commande, vous devez spécifier les informations suivantes dans la boîte de dialogue Personnalisation des commandes :

Définition de commande	Description
Nom	Nom de la commande qui s'affiche comme un sous-menu dans l'élément de menu Exécuter des commandes. Les noms sont exclusifs et peuvent contenir des touches d'accès rapide (&Génération Java s'affichera comme suit : Génération Java).
Sous-menu	Nom du sous-menu regroupant les commandes connexes. Il s'affiche dans l'élément de menu Exécuter des commandes. Vous pouvez sélectionner un sous-menu par défaut dans la liste (<Aucun>, Exporter, Génération, Importer, Reverse engineering, Vérification du modèle) ou créer votre propre sous-menu qui sera ajouté à la liste. Si vous sélectionnez <Aucun> ou laissez la zone vide, la commande que vous avez définie s'affichera directement dans le sous-menu de l'élément de menu Exécuter des commandes.

Définition de commande	Description
Contexte	Information facultative qui permet de définir l'affichage de la commande en fonction du diagramme ouvert. Si vous ne définissez aucun contexte pour la commande, celle-ci s'affichera dans l'élément de menu Exécuter des commandes quel que soit le diagramme ouvert et même lorsqu'il n'y a aucun diagramme actif.
Type	Type de la commande que vous sélectionnez dans la liste. Il peut s'agir d'un programme exécutable ou d'un script VB.
Ligne de commande	Chemin du fichier de commande. Le bouton Points de suspension vous permet de sélectionner un fichier ou tout autre argument. Si le fichier de commande est un script VB, vous pouvez cliquer sur le bouton Editer avec dans la barre d'outils pour ouvrir directement l'éditeur VBScript et visualiser ou modifier le script.
Commentaire	Libellé descriptif de la commande. Il s'affiche dans la barre d'état lorsque vous sélectionnez un nom de commande dans l'élément de menu Exécuter des commandes.
Afficher dans le menu	Permet d'afficher ou non le nom de la commande dans l'élément de menu Exécuter des commandes. Cela vous permet de désactiver une commande dans le menu sans supprimer pour autant la définition de la commande.
Touche de raccourci	Permet d'affecter un raccourci clavier à la commande. Vous pouvez en sélectionner un dans la liste. L'utilisation d'un raccourci clavier est exclusive.

Option Contexte

L'option Contexte vous permet d'afficher une commande personnalisée en fonction du diagramme courant, si les paramètres que vous avez déclarés dans sa définition lui correspondent.

Lorsqu'aucune correspondance n'est trouvée, la commande n'est pas disponible.

Lorsque vous cliquez sur le bouton Points de suspension dans la colonne Contexte de la boîte de dialogue Personnalisation des commandes, vous ouvrez la boîte de dialogue Définition du contexte dans laquelle vous êtes invité à sélectionner les paramètres facultatifs suivants :

Paramètre	Description
Modèle	Permet de sélectionner un type de modèle dans la liste Modèle.
Diagramme	Permet de sélectionner un type de diagramme pour le modèle sélectionné dans la liste Diagramme.

Paramètre	Description
Ressource cible	Permet de sélectionner ou de saisir un nom de fichier .XEM dans la liste Ressource Cible qui contient tous les modèles de définitions étendues définis pour le type de modèle sélectionné. Le bouton de recherche permet de sélectionner dans un autre répertoire d'autres types de ressources cibles telles que les XOL, XPL, XSL et XDB.

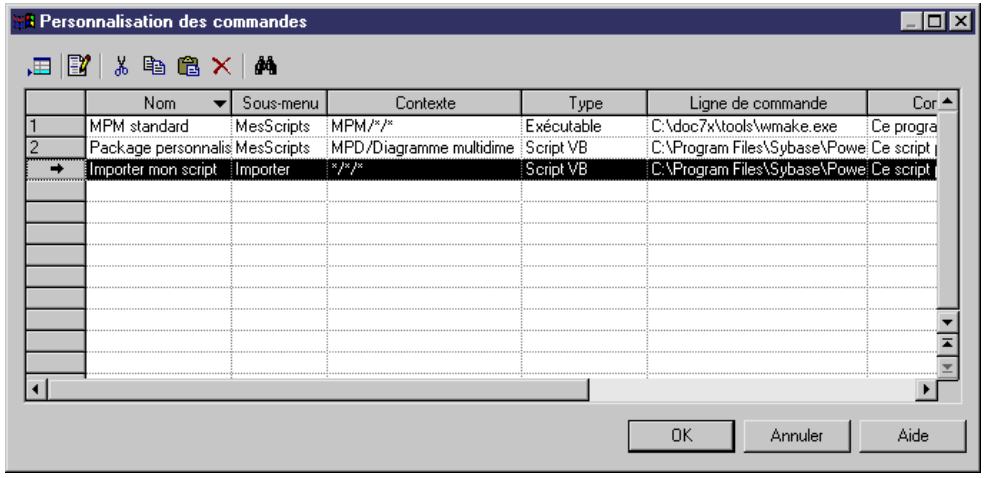
Les exemples suivants illustrent des définitions de contexte, tels qu'elles s'affichent dans la colonne Contexte de la boîte de dialogue Personnalisation des commandes :

Définition de contexte	Description
//*	Valeur par défaut. La commande s'affiche dans l'élément de menu Exécuter les commandes, quel que soit le diagramme ouvert et même lorsqu'il n'y a aucun diagramme actif.
MOO/*/*	La commande s'affiche dans l'élément de menu Exécuter les commandes toutes les fois où un MOO est ouvert, quel que soit le type du diagramme ouvert et la ressource cible sélectionnée.
MOO/Diagramme de classes/*	La commande s'affiche dans l'élément de menu Exécuter les commandes toutes les fois où un MOO est ouvert avec un diagramme de classe et quel que soit la ressource cible sélectionnée.
MOO/Diagramme de classes/Java	La commande s'affiche dans l'élément de menu Exécuter les commandes toutes les fois où un MOO est ouvert avec un diagramme de classe qui a pour ressource cible Java.

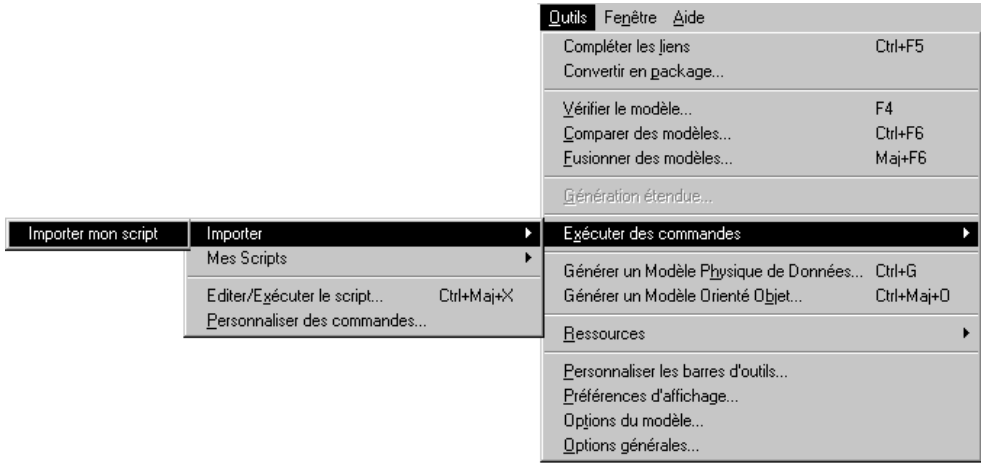
Sous-menus Importer/Exporter

Lorsque vous sélectionnez Importer ou Exporter dans la liste Sous-menu de la boîte de dialogue Personnalisation des commandes, ces commandes que vous avez définies s'affichent non seulement comme entrée de sous-menu de l'élément de menu Exécuter des commandes du menu Outils mais également comme entrée de sous-menu des éléments de menu Importer et Exporter du menu Fichier.

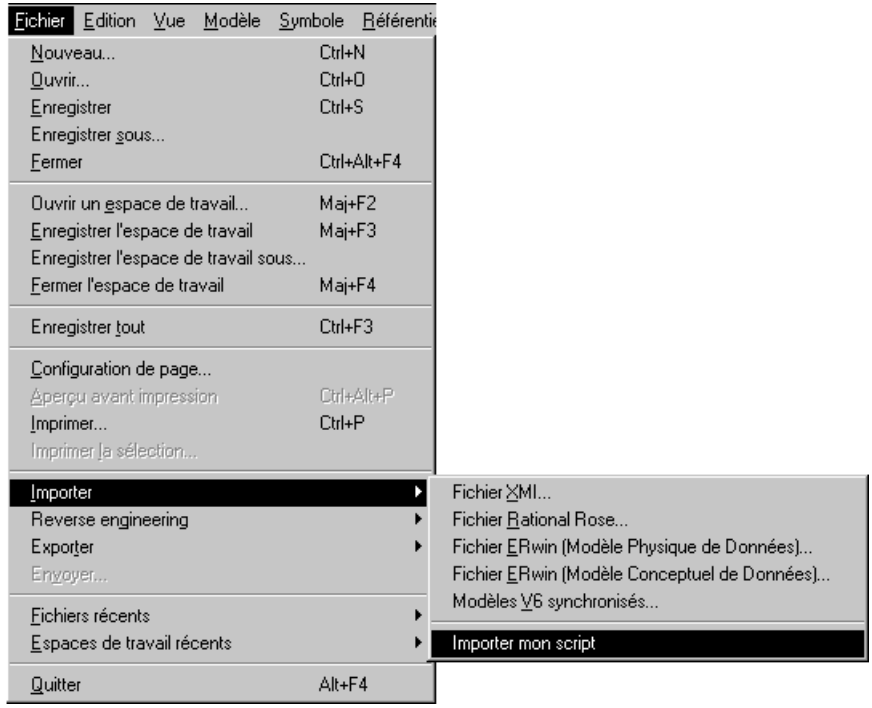
Par exemple vous définissez les commandes suivantes dans la boîte de dialogue Personnalisation des commandes :



La commande s'affiche comme suit dans le menu Outils :



La commande s'affiche comme suit dans le menu Fichier :



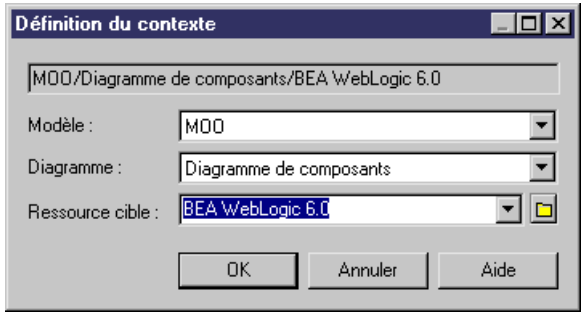
Définition d'une commande personnalisée

Vous pouvez définir vos propres commandes personnalisées.

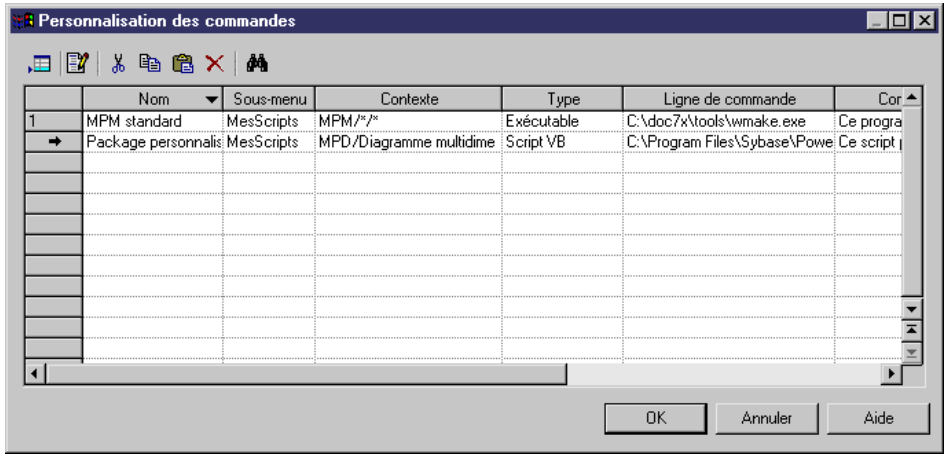
1. Sélectionnez **Outils > Exécuter des commandes > Personnaliser des commandes** pour afficher la boîte de dialogue Personnalisation des commandes.
2. Cliquez sur une ligne vide dans la liste.

   ou

   Cliquez sur l'outil Ajouter une ligne.
3. Saisissez un nom de commande dans la colonne Nom.
4. [Facultatif] Sélectionnez un sous-menu dans la liste de la colonne Sous-menu.
5. [Facultatif] Définissez un contexte en cliquant sur le bouton Points de suspension dans la colonne Contexte.



6. Sélectionnez un type dans la liste de la colonne Type.
7. Sélectionnez un fichier de commande ou un argument dans la colonne Ligne de commande.
8. [Facultatif] Saisissez un commentaire dans la colonne Commentaire.
9. Sélectionnez la case à cocher Afficher dans le menu pour afficher le nom de la commande dans le menu.
10. [Facultatif] Sélectionnez un raccourci clavier dans la liste de la colonne Touche de raccourci.
11. Cliquez sur OK.



Vous pouvez visualiser la commande que vous venez de définir en sélectionnant **Outils > Exécuter des commandes**.

Gestion des commandes personnalisées

Comprendre la manière dont les commandes personnalisées sont stockées dans PowerAMC vous permettra de brancher aisément vos programmes dans l'application au moment de les installer.

Stockage

Les commandes personnalisées sont enregistrées dans le Registre. Vous pouvez définir des valeurs pour les commandes personnalisées dans la catégorie CURRENT USER du Registre ou bien dans la catégorie LOCAL MACHINE du Registre.

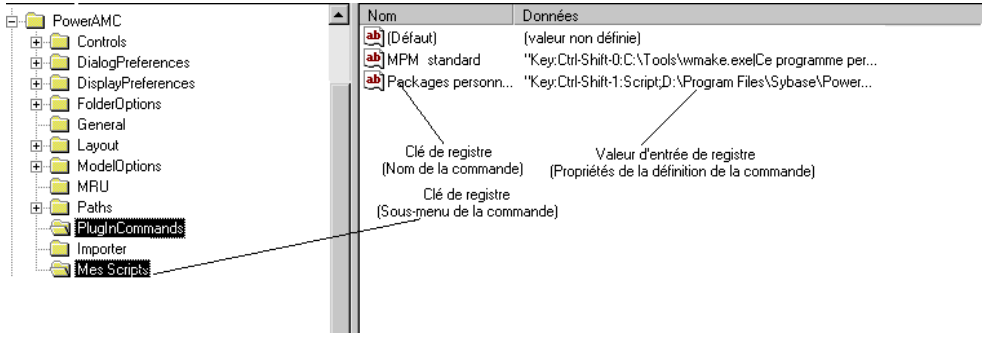
Si vous définissez des valeurs dans la catégorie LOCAL MACHINE du Registre, les commandes personnalisées sont disponibles pour tous les utilisateurs de la machine. Cependant, lorsque vous ôtez une commande personnalisée définie dans le Registre à partir de la boîte de dialogue Personnalisation des commandes, vous ôtez uniquement la ligne de la liste mais pas son entrée correspondante dans le Registre. Lorsque vous effectuez cette opération, la valeur par défaut (celle définie dans la catégorie LOCAL MACHINE du Registre) s'affiche à nouveau lorsque vous ré-ouvrez la boîte de dialogue.

La définition des commandes personnalisées peut se trouver dans :

- HKEY_CURRENT_USER\Software\Sybase\PowerAMC <version>\PlugInCommands.
- HKEY_LOCAL_MACHINE\Software\Sybase\PowerAMC <version> \PlugInCommands.

Chaque commande personnalisée est stockée dans une valeur chaîne distincte dans le Registre :

- Le nom de la commande personnalisée est une entrée de Registre portant le même nom que la commande.
- Le sous-menu de la commande personnalisée est une clé de Registre portant le même nom que le sous-menu.
- Les autres propriétés de la commande sont stockées dans le champ Données de l'entrée de Registre (valeur d'entrée de registre).



Format de définition

La syntaxe de l'entrée de Registre est la suivante :

[Hide:][Key:<key specification>:][Script:]<command>[|comment]

Notez qu'aucun des préfixes ci-dessus n'est localisé.

Mot-clé de la syntaxe	Description
Hide:	Définit la commande comme cachée
Key:<key specification>:	Permet d'affecter un raccourci clavier à la commande. C'est un champ facultatif. L'élément <key specification> peut inclure les préfixes facultatifs suivants, respectivement dans cet ordre : • **Ctrl-** pour l'indicateur CONTROL • **Shift-** pour l'indicateur MAJ Suivi d'un caractère unique compris entre les intervalles "0-9" (exemple : Ctrl-Shift-0).
Script:	Définit la commande à interpréter comme un script interne.
<Command>	Définit le nom de fichier ainsi que des arguments facultatifs de la commande. La commande est obligatoire et se termine par le caractère suivant '|'. Si vous souhaitez insérer le caractère '|' au sein d'une commande, vous devez doubler ce caractère.
Comment	Décrit la commande. C'est un champ facultatif.

Remarque : la boîte de dialogue Personnalisation des commandes prend uniquement en charge les raccourcis clavier compris dans l'intervalle suivant : "Ctrl-Maj-0" à "Ctrl-Maj-9". Si vous définissez des raccourcis clavier en dehors de cet intervalle, des conflits peuvent surgir avec les autres raccourcis clavier intégrés dans l'application et aboutir à des résultats imprévisibles. La réutilisation d'un même raccourci clavier pour deux commandes distinctes peut aussi aboutir à des résultats imprévisibles.

Création d'un complément ActiveX

Vous pouvez créer vos propres éléments de menu dans les menus de PowerAMC à l'aide d'un complément ActiveX.

Remarque : Pour pouvoir utiliser votre complément, enregistrez-le dans le répertoire Add-ins situé dans le répertoire d'installation de PowerAMC et activez-le via la fenêtre Options générales de PowerAMC (voir *Guide des fonctionnalités générales > L'interface de PowerAMC > Personnalisation de votre environnement de modélisation > Options générales > Gestion des compléments*).

L'ActiveX doit implémenter une interface spécifique nommée IPDAddIn pour devenir un complément dans Power AMC.

Cette interface définit les méthodes suivantes :

- HRESULT Initialize([in] IDispatch * pApplication)
- HRESULT Uninitialize()
- BSTR ProvideMenuItems([in] BSTR sMenu, [in] IDispatch *pObj)
- BOOL IsCommandSupported([in] BSTR sMenu, [in] IDispatch * pObject, [in] BSTR sCommandName)
- HRESULT DoCommand(in BSTR sMenu, in IDispatch *pObj, in BSTR sCommandName)

Ces méthodes sont invoquées par PowerAMC pour dialoguer avec les menus et exécuter les commandes définies par l'ActiveX.

Méthode Initialize / Uninitialize

La méthode Initialize permet d'initialiser la communication entre PowerAMC et l'ActiveX. PowerAMC démarre la communication en fournissant à l'ActiveX un pointeur vers son objet application. L'objet application vous permet de gérer l'environnement de PowerAMC (fenêtre de résultat, modèle actif etc.), il doit être enregistré pour référence ultérieure. Le type de l'objet application est défini dans la bibliothèque de type PdCommon.

La méthode Uninitialize est utilisée pour nettoyer des références vers les objets de PowerAMC. Cette méthode est invoquée lorsque PowerAMC est fermé et doit être utilisée pour libérer toutes les variables globales.

Méthode ProvideMenuItems

La méthode ProvideMenuItems retourne un texte XML qui décrit les éléments de menu à ajouter dans les menus de PowerAMC. La méthode est invoquée chaque fois que PowerAMC a besoin d'afficher un menu.

Lorsque vous pointez sur un symbole dans le diagramme, puis cliquez le bouton droit de la souris, cette méthode est invoquée deux fois : une fois pour l'objet et une fois pour le symbole. Ainsi, vous pouvez créer une méthode qui ne sera invoquée que sur les menus contextuels graphiques.

La méthode ProvideMenuItems est invoquée une fois lors de l'initialisation de PowerAMC pour remplir les menus Import et Reverse. Aucun objet n'est placé en paramètre dans la méthode à ce moment là.

Le texte XML qui décrit un menu peut utiliser les éléments (DTD) suivants :

```
<!ELEMENT Menu (Command | Separator | Popup)*>
<!ELEMENT Command>
<!ATTLIST Command
    Name    CDATA    #REQUIRED
    Caption    CDATA    #REQUIRED
>
<!ELEMENT Separator>
```

```
<!ELEMENT PopUp (Command | Separator | Popup)*>
<!ATTLIST PopUp
    Caption    CDATA    #REQUIRED
>
```

Exemple :

```
ProvideMenuItems ("Object", pModel)
```

Il résulte le texte suivant :

```
<MENU>
<POPUP Caption="&Perforce">
    <COMMAND Name="CheckIn" Caption="Check &In"/>
    <SEPARATOR/>
    <COMMAND Name="CheckOut" Caption="Check &Out"/>
</POPUP>
</MENU>
```

Remarque : cette syntaxe est la même que celle utilisée dans la création d'un menu à l'aide d'un fichier de ressource.

Remarque : Vous pouvez utiliser l'interface de l'éditeur de ressources pour visualiser dans la page XML la syntaxe d'un menu que vous avez créé dans la page Menu et qui vous aidera à construire la même syntaxe XML.

Pour plus d'informations sur la personnalisation des menus à l'aide d'un fichier de ressource, voir *Ajout de commandes et autres éléments dans votre menu* à la page 112.

Méthode IsCommandSupported

La méthode IsCommandSupported vous permet de désactiver dynamiquement les commandes définies dans un menu. La méthode doit renvoyer la valeur "true" pour activer une commande et "false" pour la désactiver.

Méthode DoCommand

La méthode DoCommand implémente l'exécution d'une commande désignée par son nom.

Exemple :

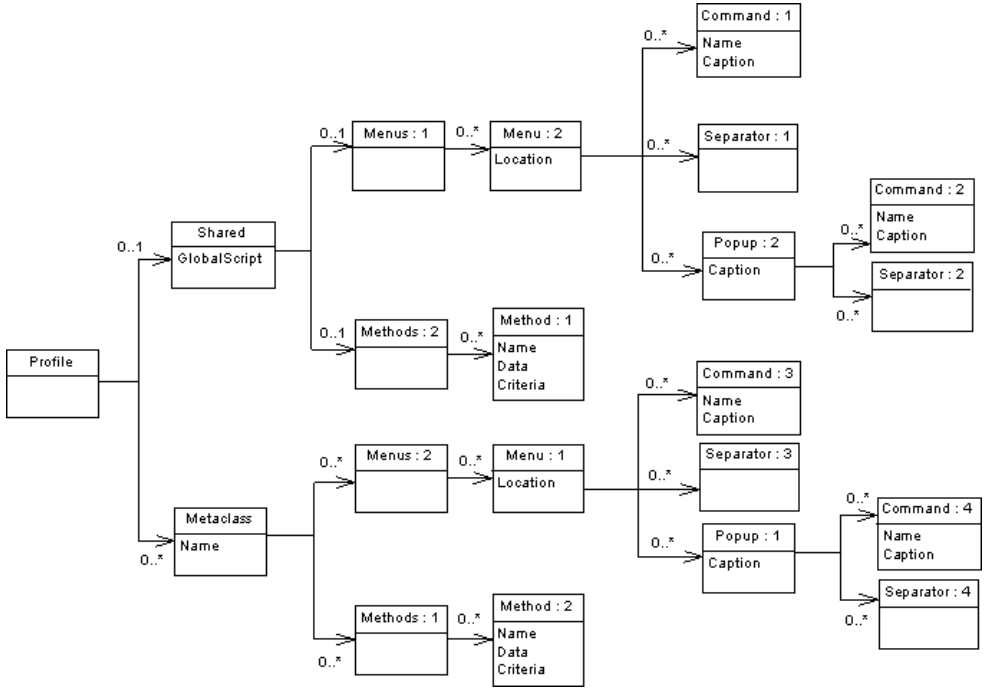
```
DoCommand ("Object", pModel, "CheckIn")
```

Création d'un complément fichier XML

Vous pouvez créer vos propres éléments de menu dans les menus de PowerAMC à l'aide d'un fichier XML

Remarque : Pour pouvoir utiliser votre complément, enregistrez-le dans le répertoire Add-ins situé dans le répertoire d'installation de PowerAMC et activez-le via la fenêtre Options générales de PowerAMC (voir *Guide des fonctionnalités générales > L'interface de PowerAMC > Personnalisation de votre environnement de modélisation > Options générales > Gestion des compléments*).

L'illustration suivante vous permet de comprendre la structure du fichier XML :



Profile est l'élément racine du complément fichier XML. Il contient les éléments suivants :

- Shared pour lesquels les menus et commandes sont définis
- Metaclass qui définit les menus et commandes pour une métaclasse particulière

<!ELEMENT Profile ((Shared)?, (Metaclass)*)>.

Shared

L'élément Shared définit les menus qui sont toujours disponibles et leurs méthodes associées (éléments Menus et Methods) et les méthodes partagées (attribut GlobalScript).

L'attribut GlobalScript est utilisé pour spécifier un script (VBS) global facultatif qui peut contenir des fonctions partagées.

L'élément Menus contient des menus qui sont toujours disponibles pour l'application. Vous pouvez spécifier un emplacement pour définir l'emplacement du menu. La définition de cet emplacement peut prendre les valeurs suivantes :

- FileImport
- File reverse
- Tools
- Help

Vous ne pouvez définir qu'un menu par emplacement.

Methods définit les méthodes utilisées dans les menus décrits au sein de l'élément Menus et qui sont toujours disponibles pour l'application.

Metaclass

L'élément Metaclass est utilisé pour spécifier des menus qui sont disponibles pour une métaclasse particulière de PowerAMC. Une métaclasse s'identifie par son nom. Vous devez utiliser le nom public.

L'élément Menus contient des menus disponibles pour une métaclasse.

L'élément Menu décrit un menu disponible pour une métaclasse. Il contient un ensemble de commandes, de séparateurs et de menus contextuels. Vous pouvez spécifier un emplacement pour définir l'emplacement du menu. Il peut prendre les valeurs suivantes :

- FileExport
- Tools
- Help
- Object

Object est la valeur par défaut pour l'attribut Location.

L'élément Methods contient un ensemble de méthodes disponibles pour une métaclasse.

L'élément Method définit une méthode. Une méthode s'identifie par un nom et un script VB.

L'élément Command définit un élément de menu commande. Son nom doit être équivalent au nom d'un élément "Method" pour pouvoir être implémenté.

L'élément Popup définit un élément de sous-menu qui peut contenir des commandes, des séparateurs et des menus contextuels.

Caption représente la valeur affichée dans le menu.

Un séparateur indique que vous souhaitez insérer une ligne dans le menu.

Exemple :

```
<?xml version="1.0" encoding="UTF-8"?>
<Profile>
    <Metaclass Name="PdOOM.Model">
    <Menus>
    <Menu Location="Tools">
        <Popup Caption="Perforce">
            <Command Name="CheckIn" Caption="Check In"/>
            <Separator/>
            <Command Name="CheckOut" Caption="Check Out"/>
        </Popup>
    </Menu>
    </Menus>
    <Methods>
        <Method Name="CheckIn">
Sub %Method%(obj)
execute_command( p4, submit %Filename%, cmd_PipeOutput)
End Sub
```

```
        </Method>
        <Method Name="CheckOut">
Sub %Method%(obj)
execute_command( p4, edit %Filename%, cmd_PipeOutput)
End Sub
        </Method>
    </Methods>
</Metaclass>
</Profile>
```

Une méthode définie sous une métaclasse est supposée avoir l'objet courant comme paramètre ; son nom est calculé à partir du nom d'attribut de la balise de la méthode.

Exemple :

```
<Method Name="ToInt" >
Sub %Method%(obj)
 Print obj
 ExecuteCommand(&quot;%MORPHEUS%\ToInt.vbs&quot;, &quot;&quot;,
cmd_InternalScript)
End Sub
```

Chaque nom de métaclasse doit avoir pour préfixe le nom public du type de bibliothèque auquel elle appartient, par exemple PdOOM.Class.

La notion d'héritage est prise en compte : un menu défini sur la métaclasse PdCommon.NamedObject sera disponible pour la métaclasse PdOOM.Class.

Vous ne pouvez définir qu'un menu par emplacement donné. Si vous définissez plusieurs emplacements, seul le dernier sera conservé.

Les menus définis sous l'élément Shared peuvent faire référence aux emplacements suivants : "FileImport" "Reverse" et "Help".

Ces menus ne peuvent faire référence qu'à des méthodes définies sous l'élément Shared et aucun objet n'est placé en paramètre dans ces méthodes définies sous Shared.

Exemple :

```
<?xml version="1.0" encoding="UTF-8"?>
<Profile>
    <Shared>
    <GlobalScript>
Option Explicit
Function Print (obj)
Output obj.classname &amp; &quot; &quot; &amp; obj.name
End Function
    /GlobalScript>
    </Shared>
    <Metaclass Name="PdOOM.Class">
    <Menus>
    <Menu>
        <Popup Caption="Transformation">
            <Command Name="ToInt" Caption="Convert to interface"/>
            <Separator/>
```

```
        </Popup>
    </Menu>
    </Menus>
    <Methods>
        <Method Name="ToInt" >
Sub %Method%(obj)
 Print obj
 ExecuteCommand(&quot;%MORPHEUS%\ToInt.vbs&quot;, &quot;&quot;,
cmd_InternalScript)
End Sub
        </Method>
    </Methods>
    </Metaclass>
</Profile>
```

Vous pouvez retrouver le DTD au sein du dossier Add-ins du répertoire d'installation de PowerAMC.

Remarque : cette syntaxe est la même que celle utilisée dans la création d'un menu à l'aide d'un fichier de ressource.

Remarque : Vous pouvez utiliser l'interface de l'éditeur de ressources pour visualiser dans la page XML la syntaxe d'un menu que vous avez créé dans la page Menu et qui vous aidera à construire la même syntaxe XML.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W