Personnalisation et extension de PowerAMC

SAP® Sybase® PowerAMC™ 16.5 SP03

Windows

ID DU DOCUMENT : DC20013-01-1653-01
DERNIERE REVISION : Novembre 2013

Table des matières

Chapitre 1 Fichiers de ressources PowerAMC

L'environnement SAP® Sybase® 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 vôtres afin de personnaliser et d'étendre le comportement de l'environnement.

Les types de fichiers de ressources suivants, basés sur ou étendant le métamodèle public PowerAMC, sont fournis :

- *Fichier de définition* : personnalisent le métamodèle pour définir les objets disponibles pour un SGBD ou langage particulier :
  - *Fichiers de définition de SGBD* (.xdb) - définissent un SGBD particulier dans le MPD (voir *Chapitre 4, Fichiers de définition de SGBD* à la page 129).
  - *Fichiers de définition de langage de processus, objet et XML* (.xpl, .xol et .xsl) – définissent un langage particulier dans le MPM, MOO ou MSX (voir *Chapitre 3, Fichiers de définition pour les langage objet, de processus et XML* à la page 117).
- *Fichiers d'extension* (.xem) - étendent la définition 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 pour un modèle (voir *Chapitre 2, Fichiers d'extension* à la page 11).
- *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 > Stockage, partage et documentation des modèles > 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 langue de rapport* à la page 309).
- *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 > Utilisation de profils pour contrôler l'interface de PowerAMC*).
- *Profils utilisateurs* (.upf) - stockent les préférences relatives aux options de modèle, options générales, préférences d'affichage, etc (voir *Guide des fonctionnalités générales > Modélisation avec 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 > Modélisation avec PowerAMC > Objets > Conventions de dénomination*).

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 ressource dans l'éditeur de ressources
	Nouveau - Crée un nouveau fichier de ressource en utilisant un fichier original comme modèle (voir *Création et copie de fichiers de ressources* à la page 7).
	Enregistrer - Enregistre le fichier de ressource sélectionné.
	Enregistrer tout - Enregistre tous les fichiers de ressources de la liste.
	Sélectionner un chemin - Spécifie les répertoires dans lesquels PowerAMC doit effectuer une recherche pour remplir la liste (voir *Spécification des répertoires sur lesquels faire porter la recherche des fichiers de ressources* à la page 7).
	Comparer - Permet de sélectionner deux fichiers de ressources à comparer.
	Fusionner - Permet de sélectionner deux fichiers de ressources à fusionner.
	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 d'un glossaire et d'une bibliothèque d'entreprise*.
	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.

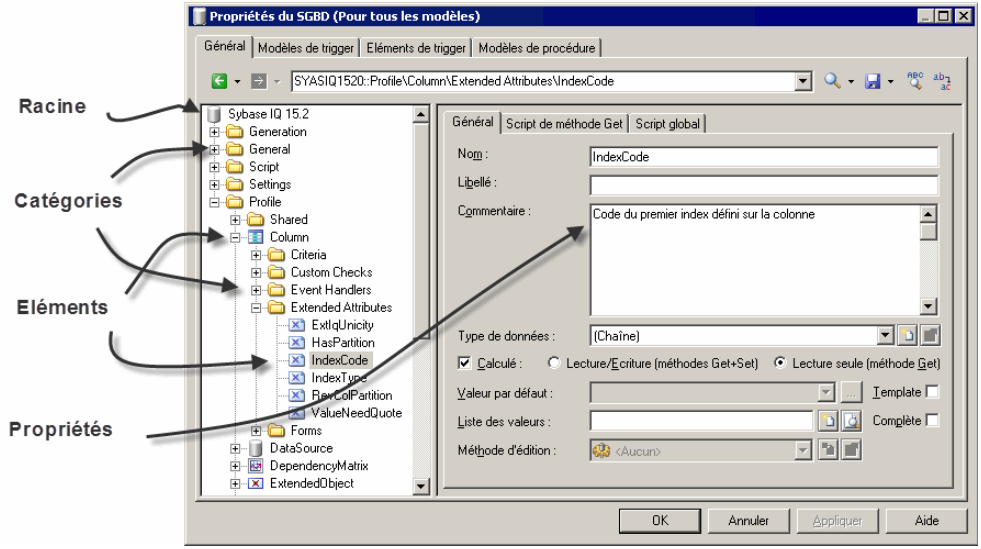
Pour afficher un fichier de définition utilisé par votre modèle :

- 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 ressource, sélectionnez **Outils > Ressources >** ***Type*** pour ouvrir la liste de fichiers de ressources appropriée, sélectionnez un fichier dans la liste, 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 jamais les originaux des 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

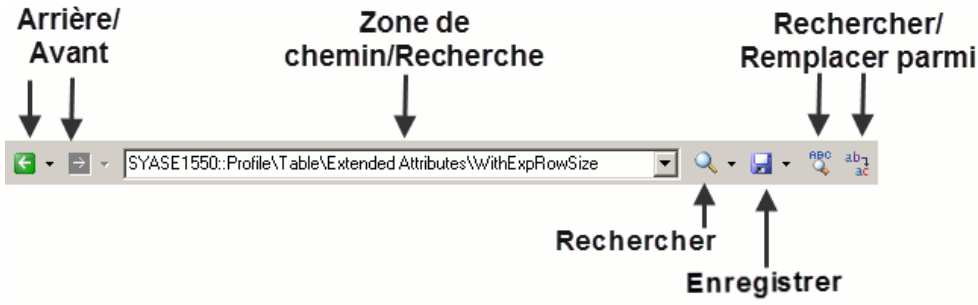
définition de SGBD collecte toutes les entrées liées à la génération et au reverse engineering de base de données.

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 depuis une référence contenue dans un autre template (voir *Templates (Profile)* à la page 94) ou une autre extension, placez votre curseur entre les signes pourcent et appuyez sur **F12**. Si une extension redéfinit un autre élément, pointez sur ce dernier, cliquez le bouton droit de la souris, puis sélectionnez **Afficher la super-définition** pour passer à l'élément redéfini.

Edition de fichiers de ressources

Vous pouvez ajouter des éléments dans l'Editeur de ressources en pointant sur une catégorie ou entrée dans l'arborescence et en cliquant le bouton droit de la souris.

Les options d'édition suivantes sont disponibles :

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

Option d'édition	Description
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'Editeur 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 ressource partagé sont disponibles pour tous les modèles qui utilisent cette ressource, tandis que celles que vous effectuez dans une ressource incorporée 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 ressource dans la liste de fichiers de ressources appropriée. Pour créer une copie d'un fichier de ressource existant, sélectionnez-le dans la zone **Copier depuis** de la boîte de dialogue **Nouveau...**.

Avertissement ! Chaque fichier de ressource 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 de fichiers de 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 12.

Spécification des répertoires sur lesquels faire porter la recherche des fichiers de ressources

Utilisez l'outil **Sélectionner un chemin** dans la barre d'outils de la liste de ressources pour spécifier les répertoires dans lesquels doit s'effectuer la recherche pour remplir la liste. Si vous prévoyez de modifier des fichiers de ressources standard ou de créer vos propres ressources, vous devez stocker ces fichiers dans un répertoire située hors du répertoire d'installation de PowerAMC.

Par défaut, seul le répertoire contenu dans le dossier `Program Files` contenant les fichiers de ressources standard s'affiche dans la liste, mais PowerAMC ne permet pas d'y enregistrer des modifications, et va proposer un lieu de remplacement si vous tentez de le faire, en ajoutant le répertoire sélectionné dans la liste. Vous pouvez ajouter des répertoires supplémentaires si nécessaire.

Remarque : Si vous avez créé ou modifié des fichiers de ressources dans `Program Files` avant la version 16.5, qui est la version dans laquelle cette règle a été introduite, vos fichiers

risquent de ne plus être disponibles dans la mesure où Windows Vista ou Windows 7 les stockent sur un miroir virtuel situé, par exemple dans `C:\Users\nomutilisateur\AppData\Local\VirtualStore\Program Files\Sybase\PowerAMC 16\Fichiers de ressources\SGBD`. Pour restaurer ces fichiers dans vos listes, vous pouvez également les déplacer vers un répertoire plus pratique, et ajouter leur emplacement dans votre liste à l'aide de l'outil **Sélectionner un chemin**.

Le premier répertoire dans la liste est l'emplacement par défaut, qui est proposé lorsque vous enregistrez un fichier. La racine de la bibliothèque appartenant à votre connexion de référentiel la plus récente est parcourue de façon récursive avant les répertoires situés dans la liste (voir *Guide des fonctionnalités générales > Administration de PowerAMC > Déploiement d'un glossaire et d'une bibliothèque d'entreprise*).

Remarque : Dans de rares cas, lorsque vous cherchez des fichiers de ressources pour résoudre des références cassées dans des modèles, les répertoires de la liste sont parcourus dans l'ordre, et la première instance de la ressource requise est utilisée.

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 des ressources du type approprié.
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 seconde 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 générales > Modélisation avec 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 des ressources du type approprié.
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 générales > Modélisation avec PowerAMC > Comparaison et fusion de modèles*.
5. Sélectionnez ou rejetez les actions de fusion proposées, puis cliquez sur **OK** pour effectuer la fusion.

Chapitre 2 Fichiers d'extension

Les fichiers d'extension (`*.xem`) permettent de personnaliser et d'étendre le métamodèle PowerAMC afin de prendre en charge vos besoins de modélisation particuliers. Vous pouvez définir des propriétés supplémentaires pour des types de données existants ou spécifier de tout nouveaux types d'objets, pour modifier l'interface de PowerAMC (en réorganisant et ajoutant des onglets de feuilles de propriétés, des outils de Boîte à outils et des commandes de menus), mais aussi afin de définir des cibles et options de génération supplémentaires.

Les fichiers d'extension ont un suffixe `.xem` et sont situés dans *`rép_installation`*`/Fichiers de ressources/Definitions etendues de modèle`.

Des listes de fichiers d'extension par type de modèle sont disponibles en sélectionnant **Outils > Ressources > Extensions > *type de modèle***. Pour plus d'informations sur les outils disponibles dans les listes de fichiers de ressources, voir *Chapitre 1, Fichiers de ressources PowerAMC* à la page 1.

Remarque : Les extensions, telle que l'extension Excel Import, peuvent être attachées à n'importe quel type de modèle, et sont disponibles dans la liste que vous affichez en sélectionnant **Outils > Ressources > Extensions > Tous les types de modèle**.

Chaque fichier d'extension contient les deux catégories de premier niveau :

- *Generation* - utilisée pour développer ou compléter la génération d'objet par défaut de PowerAMC (pour les MPM, MOO et MSX) ou pour une génération distincte. Pour plus d'informations, voir *Catégorie Generation* à la page 122.
- *Profile* - utilisé pour étendre les métaclasses dans le métamodèle PowerAMC :
  - Créez 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 métaclasses par stéréotype.
    - Critères – pour sous-classifier les métaclasses en évaluant des conditions.
  - Ajoutez de nouvelles propriétés aux objets et affichez-les :
    - Attributs étendus – pour fournir des métadonnées supplémentaires.
    - Collections et compositions étendues – pour permettre de lier manuellement des objets.
    - Collections calculées – pour lier automatiquement des objets.
    - Matrices de dépendances – pour montrer les connexions entre deux types d'objets.
    - Formulaires – pour modifier des feuilles de propriétés ou ajouter des boîtes de dialogue personnalisées.
    - Symboles personnalisés – pour changer l'apparence des objets dans le diagramme.

- Ajoutez des contraintes et des règles de validation sur les objets :
  - Vérifications personnalisées – pour tester la validité de vos modèles à la demande.
  - Gestionnaires d'événement – pour procéder à la validation ou appeler des méthodes automatiquement.
- Exécutez des commande sur les objets :
  - Méthodes – fragments de code VBScript à appeler au moyen de commandes de menus ou de boutons de formulaires.
  - Menus [pour les métaclasses et stéréotypes uniquement] – pour ajouter des commandes dans le menus PowerAMC.
- Générez des objets de nouvelles manières :
  - Templates – pour extraire du texte des propriétés d'objet.
  - Fichiers générés - pour assembler des templates pour l'aperçu et la génération des fichiers
  - Transformations – pour automatiser les changements sur les objets lors de la génération ou à la demande.
- Etablissez des correspondances entre différents métamodèles :
  - Générations d'objet - pour définir des correspondances entre les différents modules dans le métamodèle PowerAMC pour la génération intermodèle.
  - Importations XML - pour définir des correspondances entre un schéma XML et un module PowerAMC afin d'importer des fichiers XML sous la forme de modèles.

Remarque : Etant donné que vous pouvez attacher plusieurs fichiers de ressources 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 des extensions.

Création d'un fichier d'extension

Vous pouvez créer un fichier d'extension directement dans votre modèle ou dans la liste de fichiers d'extension appropriée.

Remarque : Les extensions, telles que l'extension Excel Import, qui peut être attachée à n'importe quel type de modèle, ne peuvent être créées que dans la liste d'extensions **Tous les types de modèle**. Pour plus d'informations sur la création d'un fichier d'extension à partir d'une liste de 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.

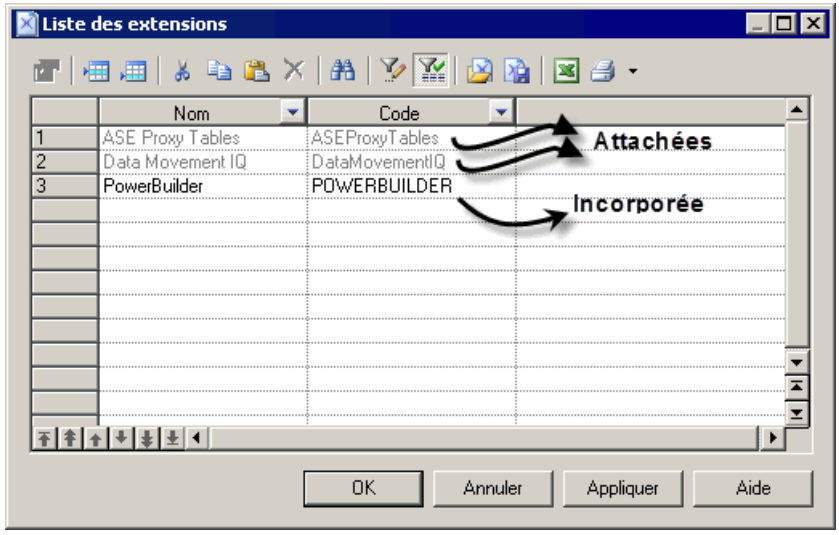
   Le nouveau fichier XEM est initialement incorporé dans votre modèle, et ne peut pas être partagé avec un autre modèle. Pour plus d'informations sur l'exportation de vos extensions et leur mise à disposition à des fins de partage, voir *Exportation d'un fichier d'extension incorporé à partager* à la page 14.

Attachement d'extensions à un modèle

Les extensions peuvent être stockées dans des fichiers `*.xem` que vous pouvez attacher à un ou plusieurs modèles. Vous pouvez attacher une ou plusieurs extensions à un modèle au moment de la création en utilisant le bouton **Sélection d'extensions** dans la boîte de dialogue Nouveau modèle. Vous pouvez attacher des fichiers d'extension ultérieurement à tout moment via la Liste des extensions.

1. Sélectionnez **Modèle > Extensions** pour ouvrir la Liste des extensions, qui répertorie les extensions attachées au modèle.
2. Cliquez sur l'outil **Attacher une extension** pour afficher la boîte de dialogue Sélection d'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 créer un lien dans le modèle vers le fichier spécifié. Pour copier le contenu du fichier d'extension et l'enregistrer dans votre modèle, cliquez sur le bouton **Incorporer la ressource dans le modèle** dans la barre d'outils. En incorporant un fichier ainsi, vous pouvez effectuer des modifications spécifiques à votre modèle sans affecter les autres modèles qui référencent 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 incorporez un fichier d'extension 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, qui répertorie les extensions attachées au modèle.
2. Sélectionnez une extension incorporée dans la liste, puis cliquez sur l'outil **Exporter une extension**.
3. Saisissez un nom et sélectionnez un répertoire pour enregistrer le fichier d'extension, puis cliquez sur **Enregistrer**.

Remarque : Pour que l'extension soit disponible afin d'être attachée à d'autres modèles, vous devez l'enregistrer dans un répertoire qui est répertorié par l'outil **Sélectionner un chemin** dans la liste d'extensions appropriée (voir *Spécification des répertoires sur lesquels faire porter la recherche des fichiers de ressources* à la page 7).

Propriétés d'un fichier d'extension

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

Le noeud racine de chaque fichier contient les propriétés suivantes :

Propriété	Description
Nom / Code	Spécifie le nom et le code du fichier d'extension, qui doit être unique au sein d'un modèle.
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 / Sous-famille	Restreint la disponibilité de l'extension à une famille cible particulière. Par exemple, lorsqu'un fichier d'extension a comme famille Java, Il n'est disponible que pour les cibles de la famille de langage objet JAVA. EJB 2.0 est une sous-famille de Java.
Rattachement automatique	Spécifie que le fichier d'extension correspondant sera automatiquement attaché 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 (voir *Templates (Profile)* à la page 94). 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'Editeur de ressources ouvre la définition correspondante dans la catégorie.

Propriété	Description
Compléter la génération de langage	[extensions de MPM, MOO, MSX] Spécifie que tout fichier généré (voir *Fichiers générés (Profile)* à la page 95) que vous définissez dans l'extension sera généré lorsque vous sélectionnerez la commande **Langage > Générer du code...** en plus des fichiers générés par défaut. Si vous donnez à un fichier généré dans votre extension le même nom que celui défini dans le fichier de définition de langage (voir *Chapitre 3, Fichiers de définition pour les langage objet, de processus et XML* à la page 117), le fichier de votre extension prévaut sur celui du fichier de définition de langage. Pour activer une génération de fichiers indépendante, vous devez désélectionner cette option, sélectionner l'option **Activer la sélection pour la génération de fichiers** pour au moins une métaclasse (voir *Métaclasses (Profile)* à la page 35), et ajouter au moins un fichier généré à la métaclasse (voir *Génération de vos fichiers dans le cadre d'une génération standard ou étendue* à la page 99). **Remarque :** PowerBuilder® ne prend pas en charge les fichiers XEM 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 122).
- Transformation Profile - Regroupes les transformations pour application au moment de la génération de modèle ou à la demande (voir *Transformations (Profile)* à la page 102).

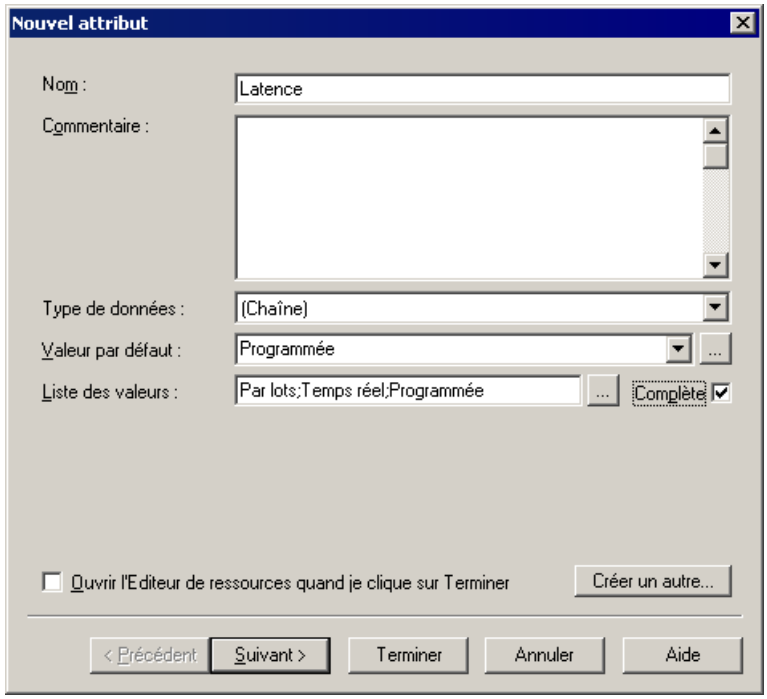
Exemple : Ajout d'un nouvel attribut à partir d'une feuille de propriétés

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**.



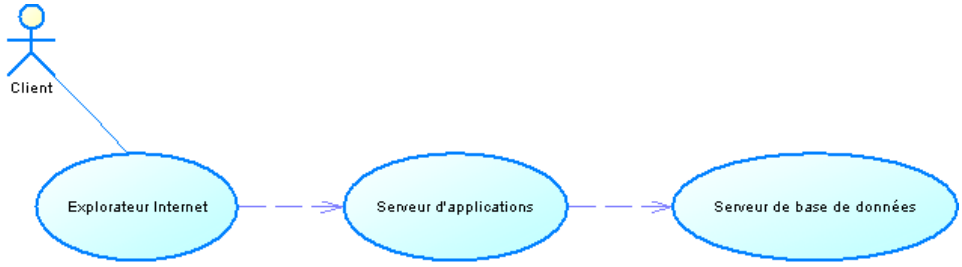
Exemple : Création d'extensions de diagramme de robustesse

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.

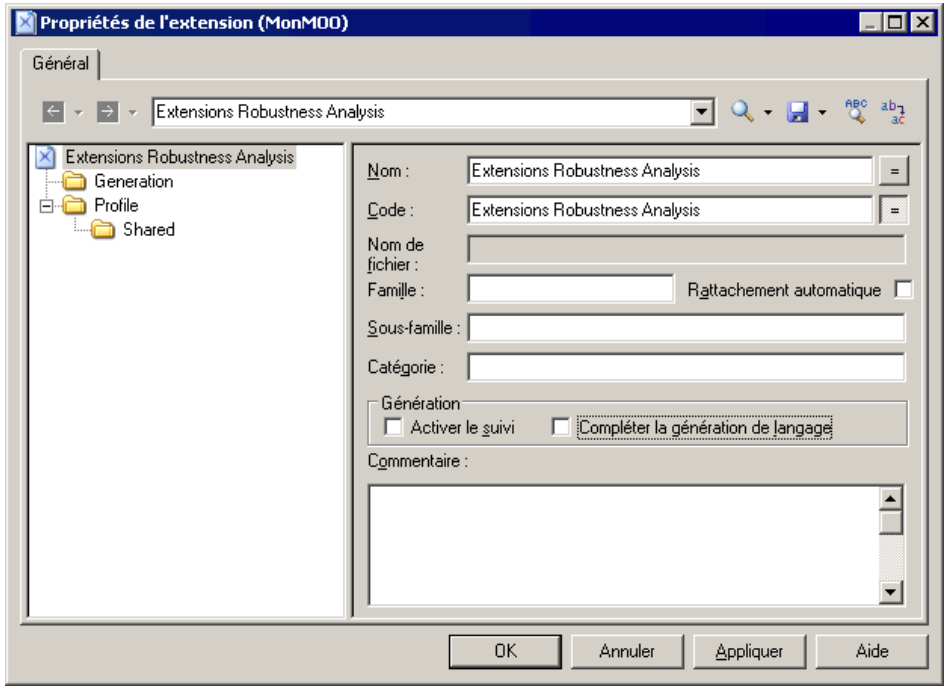
La création des extensions Robustness Analysis va nous permettre de vérifier des cas d'utilisation tels que le cas suivant, qui représente une transaction Web de base :



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.

La première étape de la définition d'extensions consiste à créer un fichier d'extension (.xem) pour les stocker :

1. Créez ou ouvrez un MOO, puis sélectionnez **Modèle > Extensions** pour afficher la liste des extensions attachée 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 :



Pour obtenir des informations détaillées sur la création de fichiers d'extension, voir *Création d'un fichier d'extension* à la page 12.

Création de nouveaux types d'objets à l'aide de stéréotypes

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

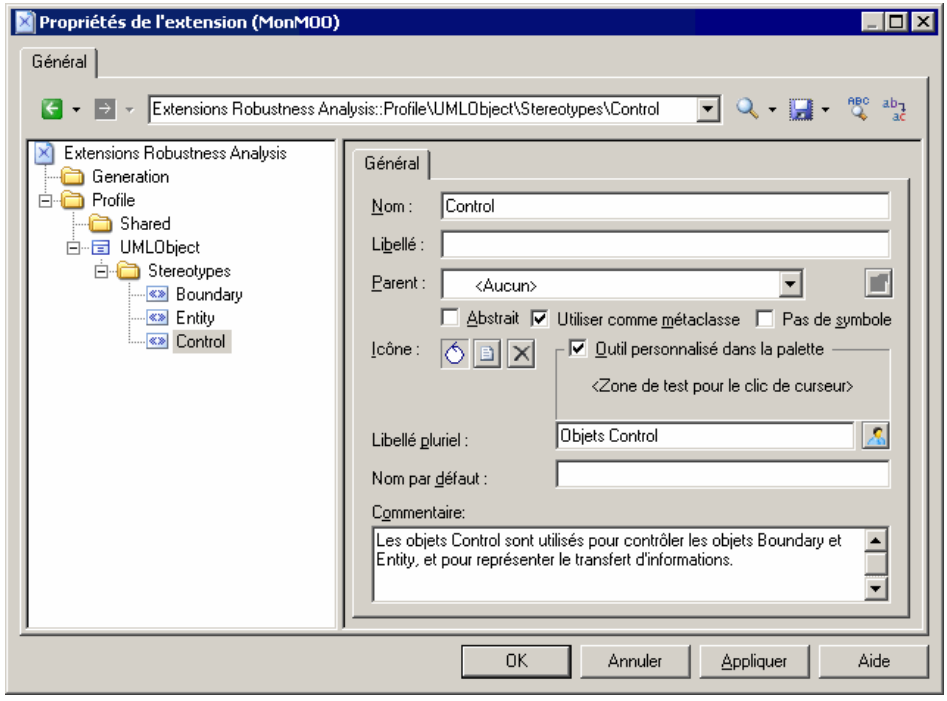
1. Pointez sur la catégorie Profile, cliquez le bouton droit de la souris, puis sélectionnez **Ajouter des métaclasses** pour afficher la boîte de dialogue Sélection de métaclasses.
2. Sélectionnez `UMLObject` sur l'onglet PdOOM, puis cliquez sur **OK** pour ajouter cette métaclasse au fichier d'extension.

   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 :

Stéréoty-pe	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 re-cherche.	entity.cur
Control	Les objets Control sont utilisés pour contrôler les objets Boundary et Entity, et représenter le transfert d'informa-tions.	control.cur



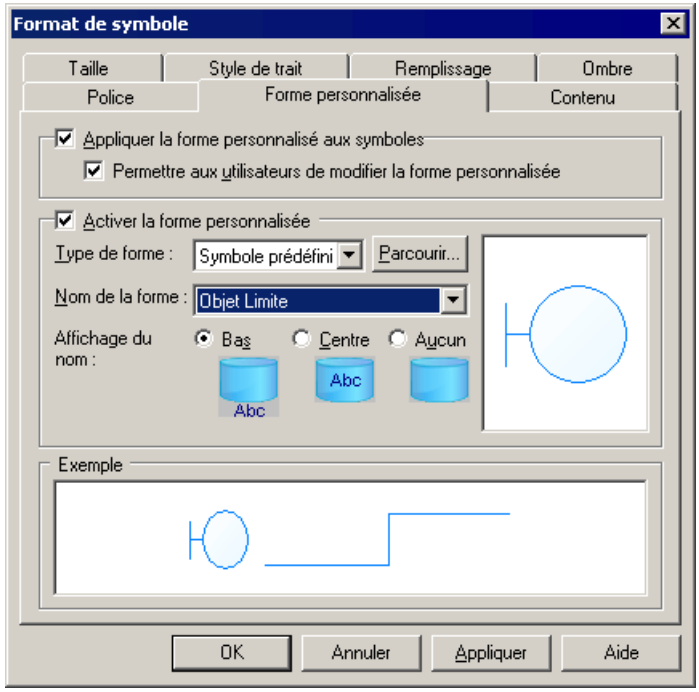
9. Cliquez sur **Appliquer** afin d'enregistrer vos modifications avant de poursuivre.

   Pour obtenir des informations détaillées sur la création de stéréotypes, voir *Stéréotypes (Profile)* à la page 40.

Spécification de symboles personnalisés pour les objets Robustness Analysis

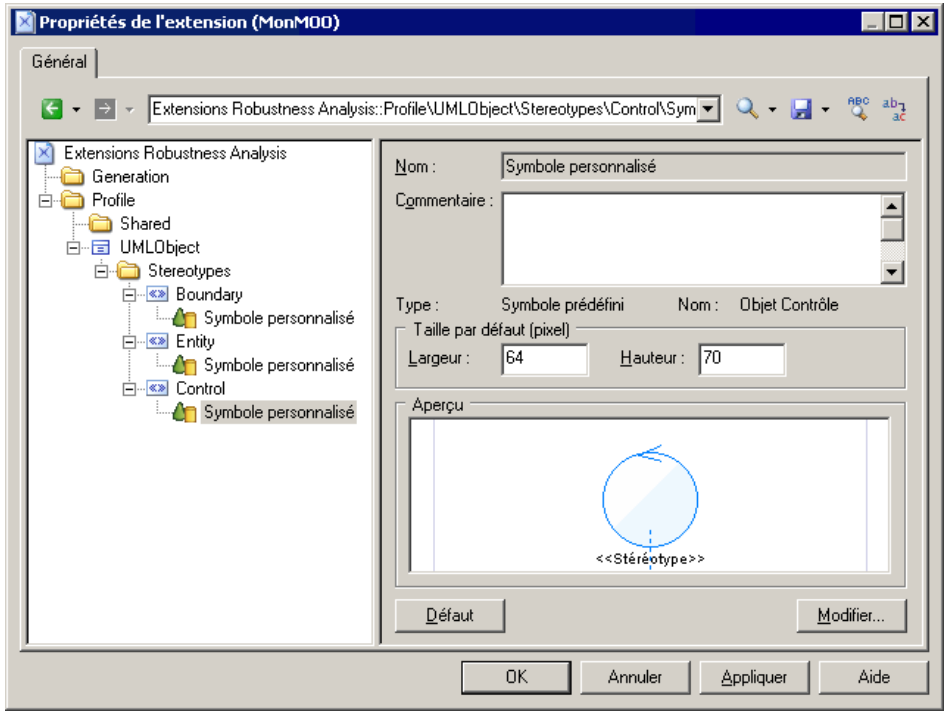
Nous allons spécifier des symboles de diagramme pour chacun de nos nouveaux objets Robustness Analysis en ajoutant des symboles personnalisés à nos nouveaux stéréotypes.

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.

   Pour obtenir des informations détaillées sur la création de symboles personnalisés, voir *Symboles personnalisés (Profile)* à la page 78.

Exemple : 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.

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

   ' 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 acteur
   ' Call CompareObjectKind() global function defined in Global
Script pane
   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
```

Remarque : Pour plus d'informations sur VBS, voir *Chapitre 7, Pilotage de PowerAMC à l'aide de scripts* à la page 331.

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

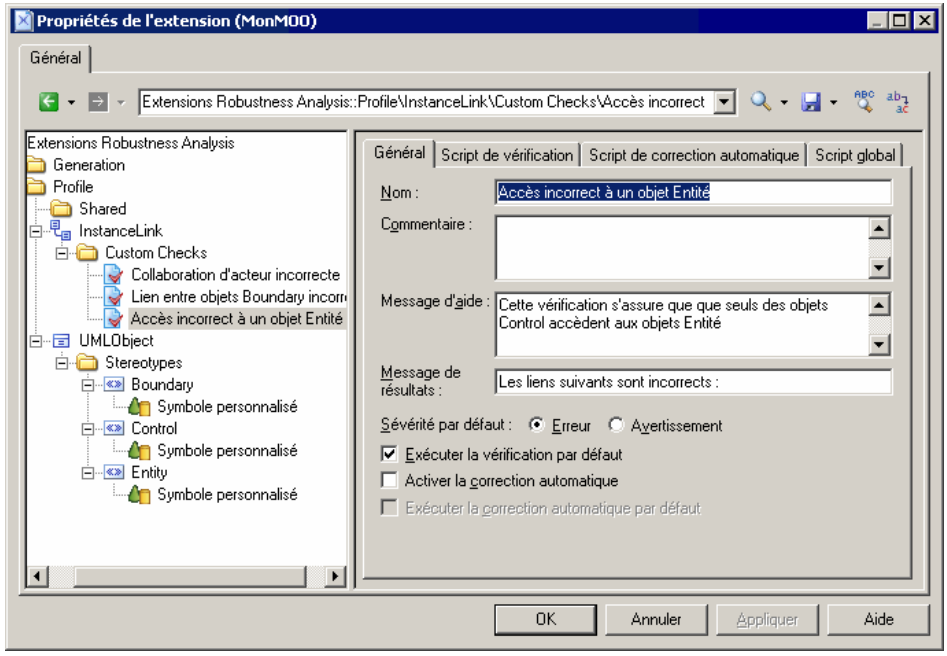
Script de vérification	`Function %Check%(link)`
	`  ' La valeur par défaut est True`
	`  %Check% = True`
	
	`  ' L'objet doit être un lien entre objets`
	`  If link is Nothing then`
	`  Exit Function`
	`  End if`
	`  If not link.IsKindOf(PdOOM.cls_InstanceLink) then`
	`  Exit Function`
	`  End If`
	
	`  ' 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	<pre>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</pre>



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

   Pour obtenir des informations détaillées sur la création de vérifications personnalisées *Vérifications personnalisées (Profile)* à la page 80.

Exemple : 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 de la métaclasse `Message` (pour extraire le numéro d'ordre du message, son nom, l'émetteur et le récepteur) et de la métaclasse `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%
```

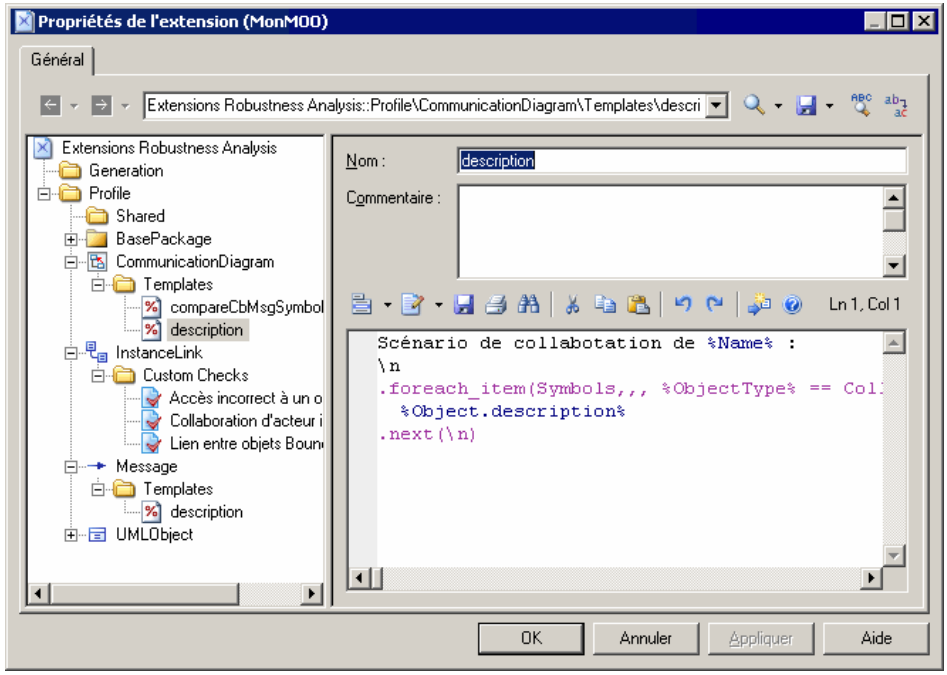
   La première ligne du template initialise la variable `_tabs`, et la macro `foreach_part` calcule le montant d'indentation approprié en bouclant sur chaque numéro d'ordre, et en ajoutant 3 espaces chaque fois qu'il trouve un point. La dernière ligne utilise cette variable afin de réaliser l'indentation,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

   Pour obtenir des informations détaillées sur les templates et le GTL, voir *Templates (Profile)* à la page 94 et *Chapitre 5, Personnalisation de la génération à l'aide du langage de génération par template* à la page 265.

Exemple : 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`.

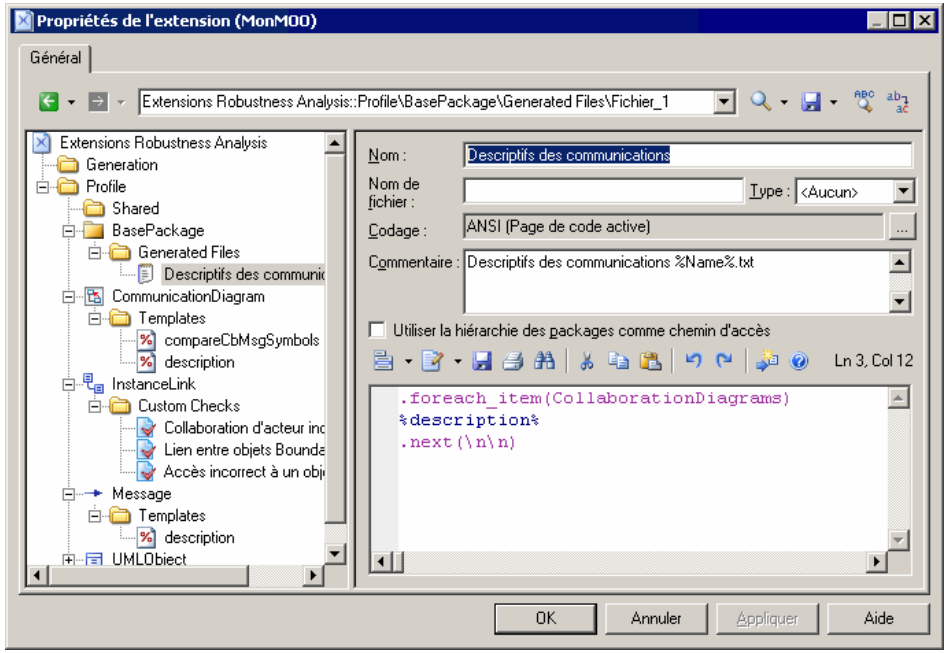
1. Pointez sur la catégorie Profile, cliquez le bouton droit de la souris, puis sélectionnez **Ajouter des métaclasses** pour afficher la boîte de dialogue Sélection de métaclasses, cliquez sur l'outil **Modifier le filtre des métaclasses**, sélectionnez `Afficher les métaclasses de modélisation abstraite`, 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'accès	[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.

   Pour obtenir des informations détaillées sur la création de fichiers générés, voir *Fichiers générés (Profile)* à la page 95.

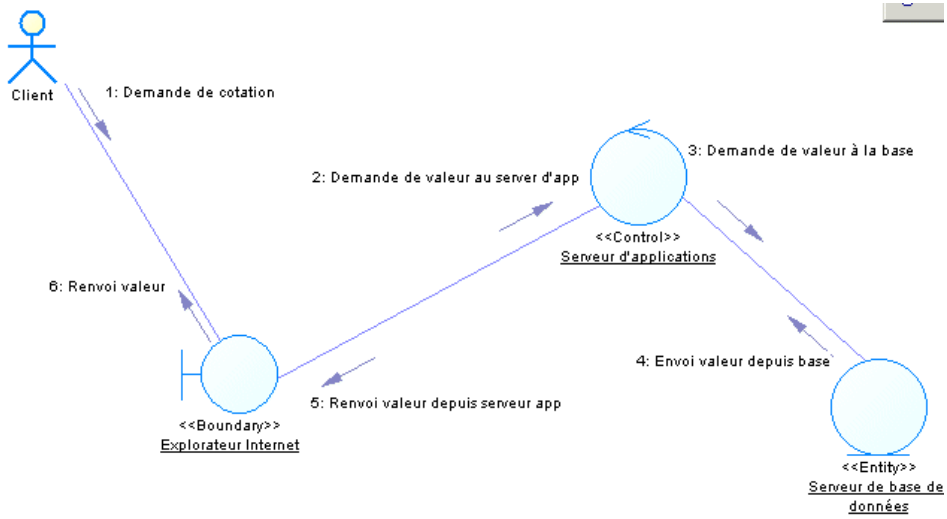
Exemple : Test des extensions Robustness Analysis

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

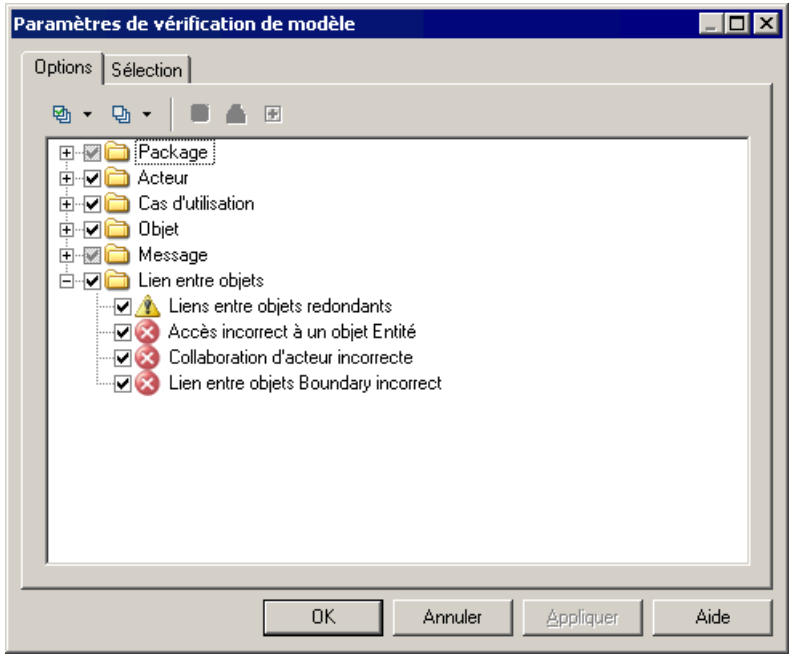
1. Pointez sur le noeud du diagramme dans l'Explorateur d'objets, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau > Diagramme de communication**.

   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 successivement 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



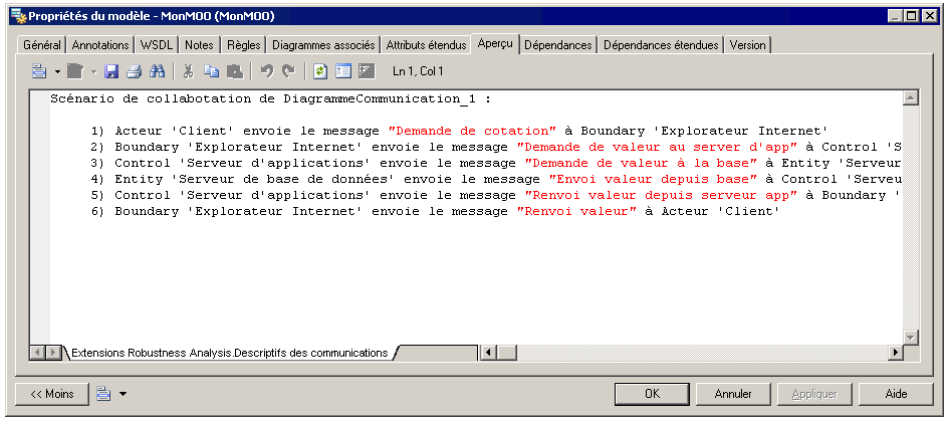
5. Sélectionnez **Outils > Vérifiez le modèle** pour afficher la boîte de dialogue Paramètres de vérification de modèle, dans laquelle les vérifications personnalisées que vous avez créées sont affichées dans la catégorie Lien entre objets :



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 :

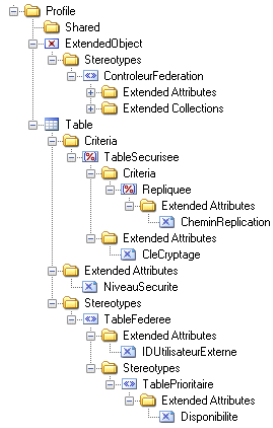


Métaclasses (Profile)

Les métaclasses sont définies dans le métamodèle PowerAMC et constituent la base pour vos extensions. Vous ajoutez une métaclasse dans la catégorie Profile lorsque vous devez l'étendre afin de modifier son comportement, ajouter de nouvelles propriétés, changer sa feuille de propriétés ou son symbole, ou même l'exclure de vos modèles.

Vous pouvez soit faire des extensions d'un type d'objet existant, soit créer un tout nouveau type d'objet de modélisation en ajoutant la métaclasse `ExtendedObject`, `ExtendedSubObject` ou `ExtendedLink` (voir *Objets, sous-objets et liens étendus (Profile)* à la page 39).

Dans l'exemple suivant, `FederationController` est un tout nouveau type d'objet créé en ajoutant la métaclasse `ExtendedObject` et en définissant un stéréotype sur cette dernière. Diverses spécialisations de la métaclasse `Table` sont définies par le biais des critères et stéréotypes :

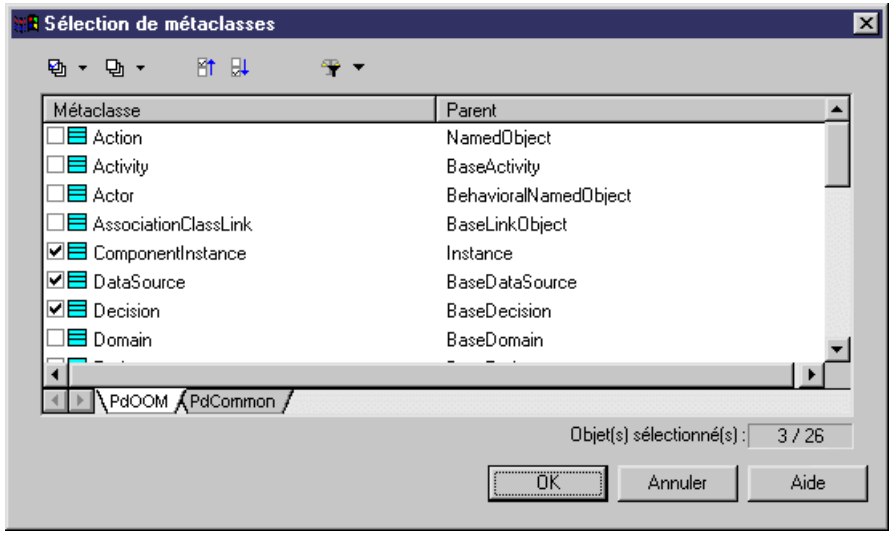


Les extensions sont héritées, de sorte que toute extension effectuée sur une métaclasse est disponible pour tous ses enfants stéréotypés, et ceux qui sont sujets aux critères. Les divers attributs étendus définis sous la métaclasse de table seront disponibles pour les instances de table en fonction des règles suivantes :

- `NiveauSecurite` - Toutes les tables.
- `CleCryptage` - Tables pour lesquelles le critère `TableSecurisee` est évalué à true.
- `CheminReplication` - Tables pour lesquelles les critères `TableSecurisee` et `Repliquee` sont tous les deux évalués à true.
- `IDUtilisateurExterne` - Tables qui ont le stéréotype `TableFederee` ou `TablePrioritaire`.
- `Disponibilite` - Tables qui ont le stéréotype `TablePrioritaire`.

Par exemple, une table qui a le stéréotype `TableFederee`, et pour laquelle le critère `TableSecurisee` est évalué à true affiche les attributs `NiveauSecurite`, `CleCryptage` et `IDUtilisateurExterne`, tandis qu'une table qui a le stéréotype `TablePrioritaire`, et pour laquelle les critères `TableSecurisee` et `Repliquee` sont tous les deux évalués à true, va afficher ces attributs, ainsi que les attributs `CheminReplication` et `Disponibilite`.

1. Pointez sur la catégorie Profile, cliquez le bouton droit de la souris, puis sélectionnez **Ajouter des 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.

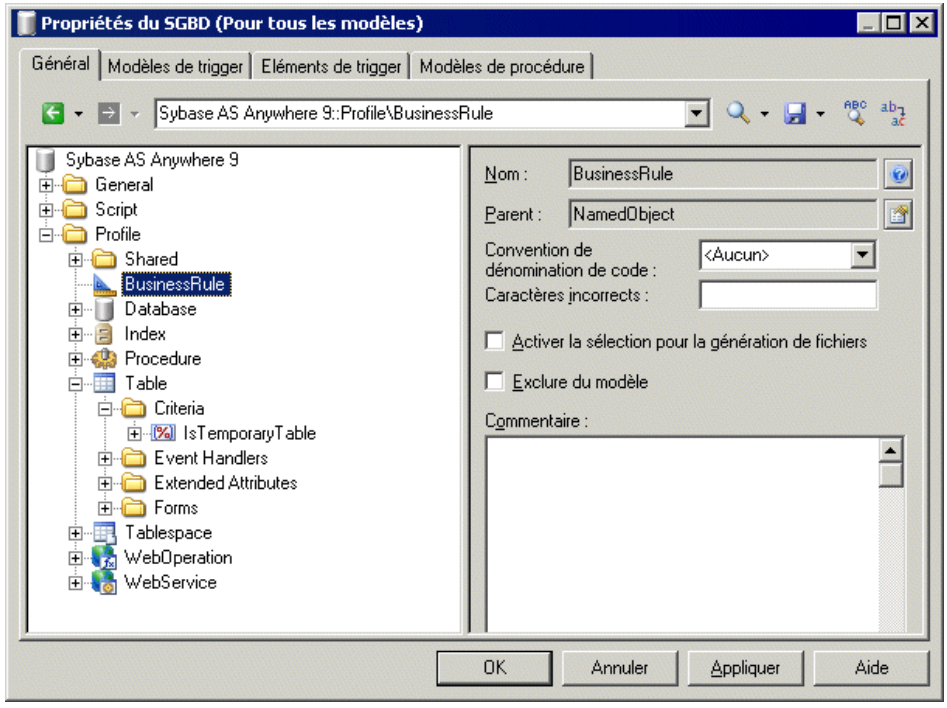


[facultatif] Utilisez l'outil **Modifier le filtre de métaclasse** pour afficher :

- Toutes les métaclasses.
- Les métaclasses concrètes - pour les types d'objets qui peuvent être créés dans un modèle, telles que `Class` ou `Interface`.
- Les métaclasses abstraites -qui ne sont jamais instanciées mais sont utilisées pour définir des extensions communes. Par exemple, ajoutez la métaclasse `Classifier` dans votre profil pour définir les extensions qui seront héritées à la fois par les classes et les interfaces.

Remarque : Pour plus d'information sur l'affichage et la navigation des métaclasses dans le métamodèle, voir *Chapitre 8, Métamodèle public PowerAMC* à la page 371.

3. Cliquez sur **OK** pour ajouter la métaclasse sélectionnée dans votre profil :



4. [facultatif] Renseignez les propriétés suivantes :

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.
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 • `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 > Modélisation avec PowerAMC > Objets > Conventions de dénomination*.

Propriété	Description
Caractères illégaux	[métaclasses concrètes dans les fichiers cible] 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 121).
Activer la sélection pour la génération de fichiers	Spécifie que les instances de la métaclasse peuvent être sélectionnées pour générer des fichier sur l'onglet **Sélection** de la boîte de dialogue Génération dans le cadre d'une génération étendue (voir *Génération de vos fichiers dans le cadre d'une génération standard ou étendue* à la page 99).
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 cette case pour la métaclasse `BusinessRule` afin de masquer les règles de gestion dans vos modèles. 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.

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 d'objet entièrement nouveaux dans vos modèle plutôt que des objets basés sur des objets PowerAMC existants. Ces objets ne s'affichent pas, par défaut, dans les modèles autres que le modèle libre sauf si vous les ajoutez dans une extension ou dans un autre fichier de ressource.

- Objets étendus – définit de nouveaux types d'objet qui peuvent être créés n'importe où.
- Sous-objets étendus – définit de nouveaux types d'objet enfant qui ne peuvent être créés que dans la feuille de propriétés de leur parent via une composition étendue (voir *Collections et compositions étendues (Profile)* à la page 54).
- Liens étendus – définit des nouveaux types de lien entre les objets.

1. Pointez sur la catégorie **Profile**, cliquez le bouton droit de la souris, puis sélectionnez **Ajouter des métaclasses**, puis 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** pour les ajouter dans 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 40 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 *Création de nouvelles métaclasses à l'aide de stéréotypes* à la page 42).
4. Cliquez sur **Appliquer** pour enregistrer les modifications.

Stéréotypes (Profile)

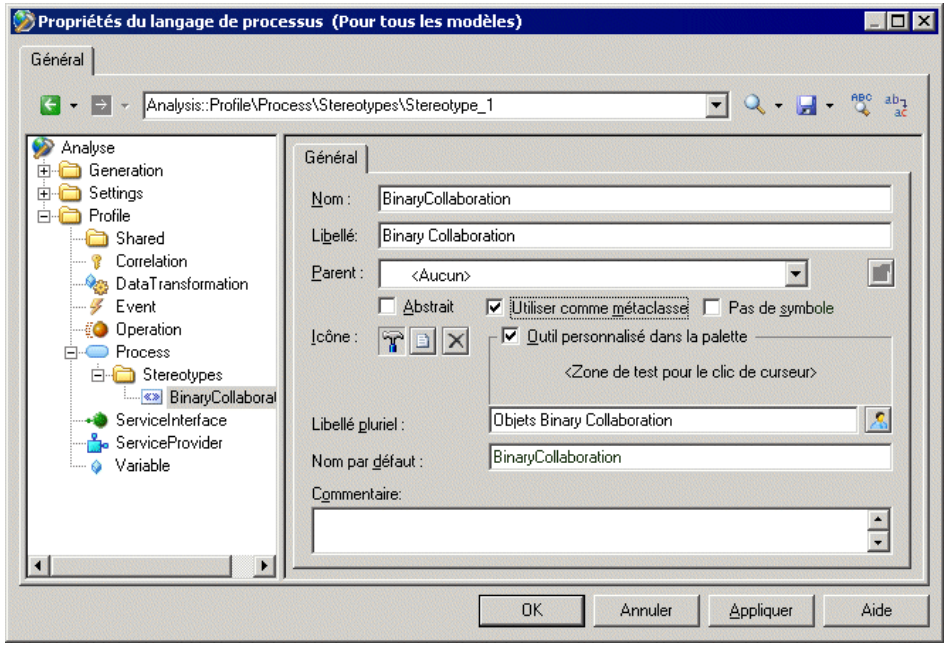
Les stéréotypes sous-classifient les métaclasses de sorte que les extensions sont appliquées aux objets uniquement s'ils portent le stéréotype. Les stéréotypes peuvent être promus au statut de métaclasses avec une liste particulière, une catégorie dans l'Explorateur d'objets, un symbole personnalisé et un outil de Boîte à outils.

Remarque : 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.

1. Pointez sur une métaclasse, un stéréotype ou un critère dans la catégorie Profile, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau > Stéréotype**.
2. Spécifiez les propriétés suivantes :

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.

Propriété	Description
Utiliser comme métaclasse	Promeut le stéréotype au mêle statut que les métaclasses PowerAMC standard, pour lui donner ses propres liste d'objets, catégorie dans l'Explorateur d'objets et onglet dans les boîtes de sélection multionglet telles que celles utilisées pour la génération (voir *Création de nouvelles métaclasses à l'aide de stéréotypes* à la page 42).
Pas de symbole	[disponible lorsque **Utiliser comme métaclasse** est sélectionné] Spécifie que les instances de la métaclasse stéréotypée ne peuvent pas être affichées dans un diagramme et qu'elle ne sont visibles que dans l'Explorateur d'objets. Désactive l'**Outil personnalisé dans la Boîte à outils**.
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`. **Remarque :** L'icône est utilisée pour identifier les objets dans l'Explorateur d'objets et ailleurs dans l'interface, mais pas comme symbole dans le diagramme. Pour spécifier un symbole de diagramme personnalisé, voir *Symboles personnalisés (Profile)* à la page 78.
Outil personnalisé dans la Boîte à outils	[disponible pour les objets qui prennent en charge les symboles] Spécifie un outil de Boîte à outils pour vous permettre de créer des objets dans un diagramme. Si vous ne sélectionnez pas cette option, les utilisateurs ne peuvent créer des objets portant ce stéréotype qu'à partir de l'Explorateur d'objets ou du menu **Modèle**. Les outils personnalisés s'affichent dans un groupe de Boîte à outils nommé en fonction du fichier de ressource dans lequel ils sont définis. **Remarque :** Si vous n'avez pas spécifié d'icône, l'outil aura par défaut une icône en forme de marteau.
Libellé pluriel	[disponible lorsque **Utiliser comme métaclasse** est sélectionné] Spécifie la forme plurielle du nom d'affichage qui s'affichera dans l'interface PowerAMC.
Nom par défaut	[disponible lorsque **Utiliser comme métaclasse** ou **Outil personnalisé dans la Boîte à outils** est sélectionné] 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	Fournit une description ou des informations supplémentaire sur le stéréotype.



Création de nouvelles métaclasses à l'aide de stéréotypes

Vous pouvez utiliser des stéréotypes afin de créer de nouveaux types d'objets qui se comportement comme des métaclasses PowerAMC standard ou bien pour avoir des objets avec des noms identiques mais des stéréotypes différents dans le même espace de noms (un stéréotype de métaclasse crée un sous-espace de noms dans la métaclasse courante).

Par exemple, voir *Création de nouveaux types d'objets à l'aide de stéréotypes* à la page 19.

Remarque : Les stéréotypes définis sur les sous-objets (comme les colonnes de table ou attributs d'entité) ne peuvent pas être promus au statut de métaclasse.

1. Créez un stéréotype sous la métaclasse sur laquelle vous souhaitez baser votre nouvelle métaclasse. Si le nouveau type d'objet ne partage pas de caractéristiques avec une métaclasses existante, utilisez la métaclasse `ExtendedObject`.

   Remarque : Si la métaclasse `ExtendedObject` ou l'autre métaclasse n'est pas visible, ajoutez-la en pointant sur la catégorie Profile, cliquant le bouton droit de la souris, puis en sélectionnant **Ajouter des métaclasses** (voir *Métaclasses (Profile)* à la page 35).
2. Dans la page de propriétés du stéréotype, sélectionnez **Utiliser comme métaclasse**.
3. [facultatif] Spécifiez une icône et un outil pour créer des instances du stéréotype de métaclasse.
4. Cliquez sur **Appliquer** pour enregistrer les modifications, puis ajoutez les attributs étendus et autres extensions appropriées sous le stéréotype.

Dans votre modèle, le stéréotype a :

- Une entrée de liste distincte dans le menu **Modèle** après la liste de sa métaclasse parent (et la liste de sa métaclasse parent n'affichera pas les objets ayant de le stéréotype d'objet). Les objets créés dans la nouvelle liste portent le nouveau stéréotype de métaclasse par défaut. Si vous changez le stéréotype, l'objet sera supprimé de la liste la prochaine fois que vous ouvrez cette dernière.
- Un dossier distinct dans l'Explorateur d'objets et une commande sous **Nouveau**, lorsque vous faites un clic droit sur le modèle ou sur un package.
- Des titres de feuille de propriétés basé sur le libellé de la métaclasse.
- Son propre onglet dans les boîtes de dialogue de sélection multionglet ainsi que celles utilisées pour la génération.

5. [facultatif : fichiers de définition de SGBD] Ajoutez le nouvel objet dans la catégorie `Script/Objects` et définissez les instructions SQL appropriées pour permettre sa génération et son reverse engineering (voir *Définition de la génération et du reverse engineering des nouvelles métaclasses* à la page 144).

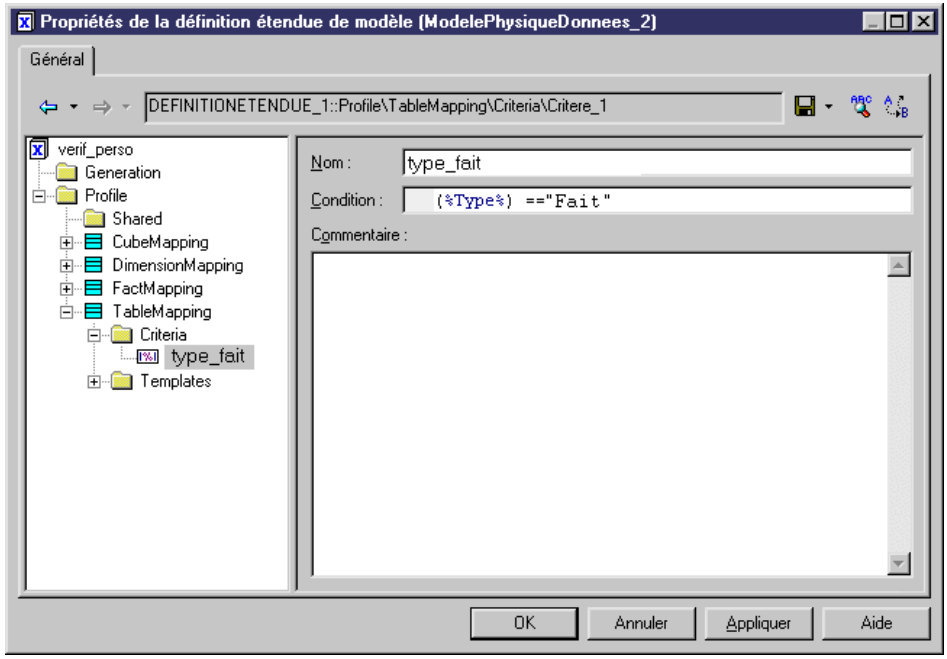
Critères (Profile)

Les critères sous-classifient les métaclasses de sorte que les extensions sont appliquées aux objets uniquement s'ils remplissent des conditions. Vous pouvez tester une instance d'objet sur plusieurs critères, et sous-critères, sa condition et les conditions spécifiées par ses parent doivent être satisfaites pour que ses extensions soient appliquées à l'instance.

1. Pointez sur une métaclasse, un stéréotype ou un critère dans la catégorie Profile, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau** > **Critère**.
2. Saisissez les propriétés suivantes :

Propriété	Description
Nom	Spécifie le nom du critère.

Propriété	Description
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 *Macro .if* à la page 297). 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 à `"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. Pour déplacer le critère sous un autre parent, sélectionnez ce parent dans la liste. Cliquez sur l'outil **Propriétés** pour ouvrir le parent et afficher ses propriétés.
Commentaire	Spécifie des informations supplémentaires relatives au critère.



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

Attributs étendus (Profile)

Les attributs étendus définissent des métadonnées supplémentaires à capturer pour les instances d'objet. Vous pouvez spécifier une valeur par défaut, permettre aux utilisateurs de saisir librement des données de type numérique, chaîne, ou autre (ou de sélectionner des objets), fournir une liste ouverte ou fermée de valeurs possibles ou calculer une valeur.

Remarque : Les attributs étendus sont répertoriés sur un onglet générique **Attributs étendus** sur la feuille de propriétés de l'objet, sauf si vous les insérez sur des formulaires (voir *Formulaires (Profile)* à la page 62). Si tous les attributs étendus sont alloués aux formulaires, la page générique n'est pas affichée.

1. Pointez sur une métaclasse, un stéréotype, ou un critère dans la catégorie Profile, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau > Attribut étendu**.
2. Spécifiez les propriétés suivantes :

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 - `TRUE` ou `False`. • Couleur - `xxx xxx xxx` où `x` est un entier compris entre 0 et 255. • Date ou Heure - votre format local tel que spécifié dans vos paramètres régionaux Windows • Fichier ou Répertoire - ne peuvent pas contenir `/ / /` ou l'un des caractères suivants : `?"<>|`. • Entier ou Réel - le format local approprié. • Hexadécimal - valeur hexadécimale. • Police - `nom de police,type de police,taille de police`. • Nom de police ou Style de police - une chaîne comportant de 1 à 50 caractères. • Taille de police - un entier compris entre 1 et 400. • Objet - un objet du type correct et, le cas échéant, avec le stéréotype approprié. Lorsque vous sélectionnez ce type, vous devez spécifier un **Type d'objet** et, le cas échéant, un **Stéréotype d'objet**, et vous pouvez également spécifier un **Nom de la collection inverse** (voir *Liaison d'objets à l'aide d'attributs étendus* à la page 54). • Mot de passe - aucune restriction. • Chaîne (monoligne) ou Texte (multiligne) - aucune restriction. Cochez la case **Valider** à droite de la liste pour forcer la validation des valeurs saisies pour l'attribut. 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 52).
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) Pour des exemples de scripts, voir *Scripts d'attributs calculés* à la page 50.
Valeur par défaut	[si non `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	[si pas `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	Spécifie 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 *Macros .object et .collection* à la page 299). 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 292) :
	`.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 52), cette zone est indisponible car les valeurs du type d'attribut étendu seront utilisées.

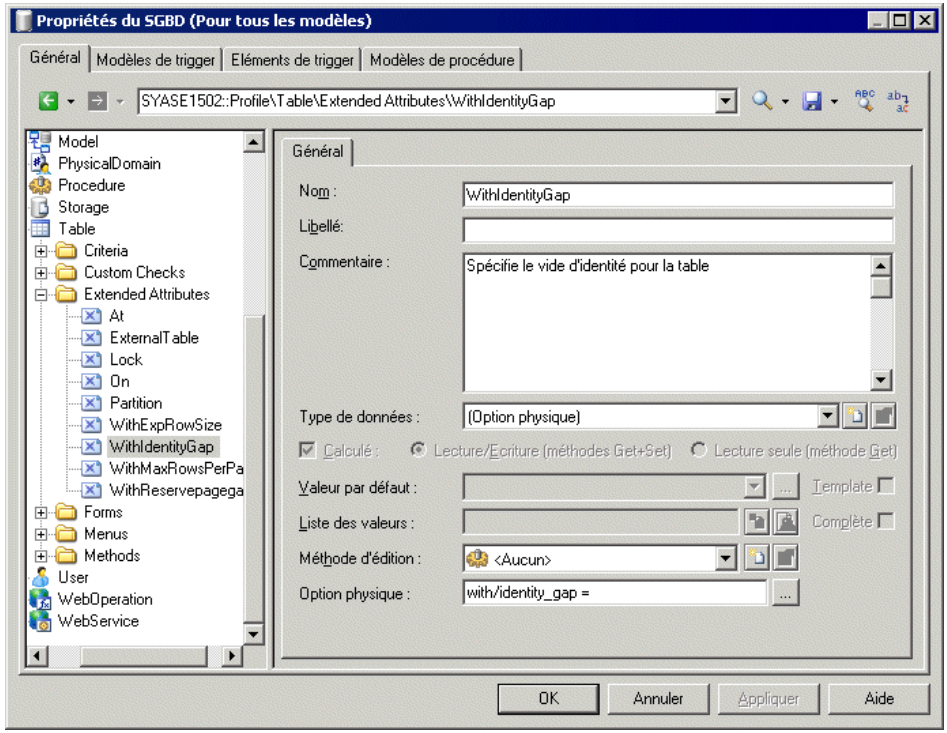
Complète	Spécifie que les valeurs possibles pour l'attribut sont définies dans la **Liste des valeurs**, et que l'utilisateur ne peut pas saisir d'autre valeur.

Propriété	Description

Méthode d'édition	[si pas `Complète`] 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`
	
	`     ' Affichage de la boîte de sélection d'objet sur la sélection`
	`      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`

Jeu d'icônes	Spécifie une jeu d'icônes à afficher sur des symboles d'objet à la place des valeurs d'attribut étendu (voir *Spécification d'icônes pour les valeurs d'attributs* à la page 53).
Format de texte	[pour les types 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.

Propriété	Description
Type / stéréotype d'objet / Nom de collection inverse	[pour les types de données `Objet` uniquement] Spécifie le type de l'objet que l'attribut contient (par exemple, Utilisateur, Table, Classe) ainsi, le cas échéant, qu'un stéréotype que les objets de ce type doivent avoir pour pouvoir être sélectionnés. Si l'option `Calculé` n'est pas sélectionnée, vous pouvez également spécifier le nom sous lequel les liens vers l'objet seront répertoriés sur l'onglet **Dépendances** 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 228.



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

Scripts d'attributs calculés

Vous pouvez créer des attributs étendus dont les valeurs dépendent des valeurs d'autres attributs. Ils peuvent être en lecture seule ou en lecteur et écriture.

Les scripts suivants fournissent un moyen de lire et d'écrire la valeur de l'attribut standard **Name** dans un attributs étendu calculé :

Script Get	Script Set

`Function %Get%(obj)`
`  %Get% = obj.GetAttribute("Name")`
`End Function`
	`Sub %Set%(obj, value)`
	`  obj.SetAttribute "Name", value`
	`End Sub`

Les scripts suivants lisent la valeur de l'attribut étendu calculé `FileGroup` depuis l'option physique `filegroup`, et l'écrivent dans l'attribut physique :

Script Get	Script Set

`Function %Get%(obj)`
`%Get% = obj.GetPhysicalOptionValue("on/<filegroup>")`
`End Function`
	`Sub %Set%(obj, value)`
	`obj.SetPhysicalOptionValue "on/<filegroup>", value`
	`End Sub`

Les scripts suivants lisent la valeur du nom de la base de données associée au MPD dans l'attribut étendu en lecture seule Database défini sur la métaclasse table :

Script Get	Script Set

`Function %Get%(obj)`
`%Get% = obj.GetAttribute("Parent.Database.Name")`
`End Function`
	[aucun]

Le script suivant évalue la valeur de l'attribut `Number` d'une table afin de définir l'attribut étendu booléen `BigTable` :

Script Get	Script Set

`Function %Get%(obj)`
`%Get% = obj.GetAttribute("Number") > 99999`
`End Function`
 	
Vous pouvez utiliser la syntaxe suivante afin de définir la valeur booléenne de façon plus explicite :
`Function %Get%(obj)`
`Dim value`
`If obj.GetAttribute("Number") > 99999 then`
`value = true`
`Else`
`value = false`
`End if`
`%Get% = value`
`End Function`
	[aucun]

Le script suivant évalue la valeur de l'attribut `Number` d'une table afin de définir un attribut étendu de type texte `TableSize` :

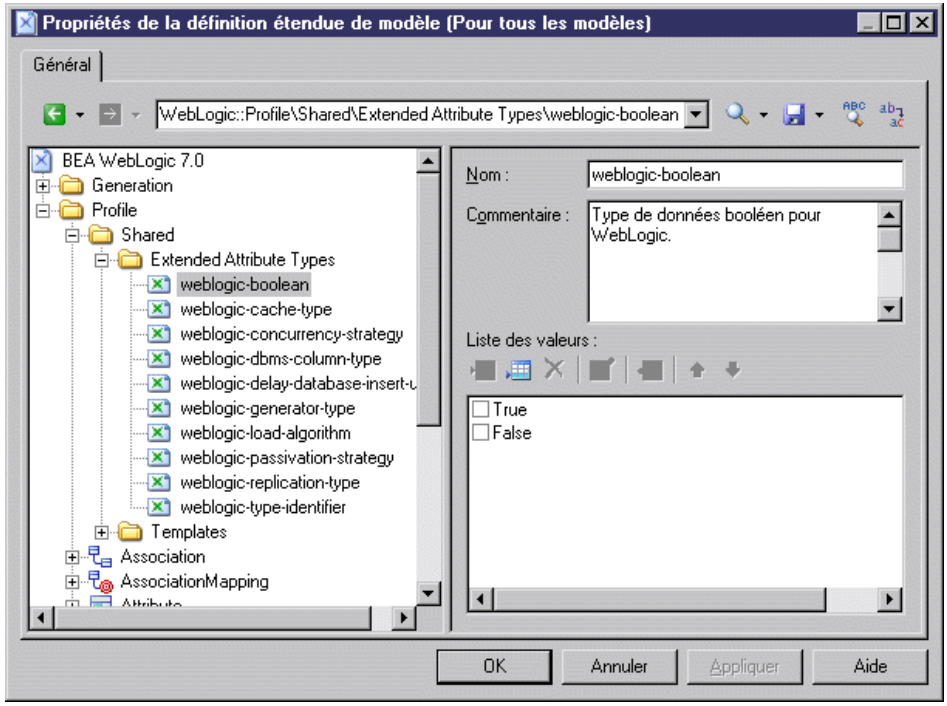
Script Get	Script Set

`Function %Get%(obj)`
`Dim value`
`If obj.GetAttribute("Number") < 10000 then`
`value = "Small"`
`ElseIf ((obj.GetAttribute("Number") > 9999) and (obj.GetAttribute("Number") < 100000)) then`
`value = "Medium"`
`ElseIf ((obj.GetAttribute("Number") > 99999) and (obj.GetAttribute("Number") < 1000000)) then`
`value = "Large"`
`Else`
`value = "Very Large"`
`End if`
`%Get% = value`
`End Function`
	[aucun]

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.



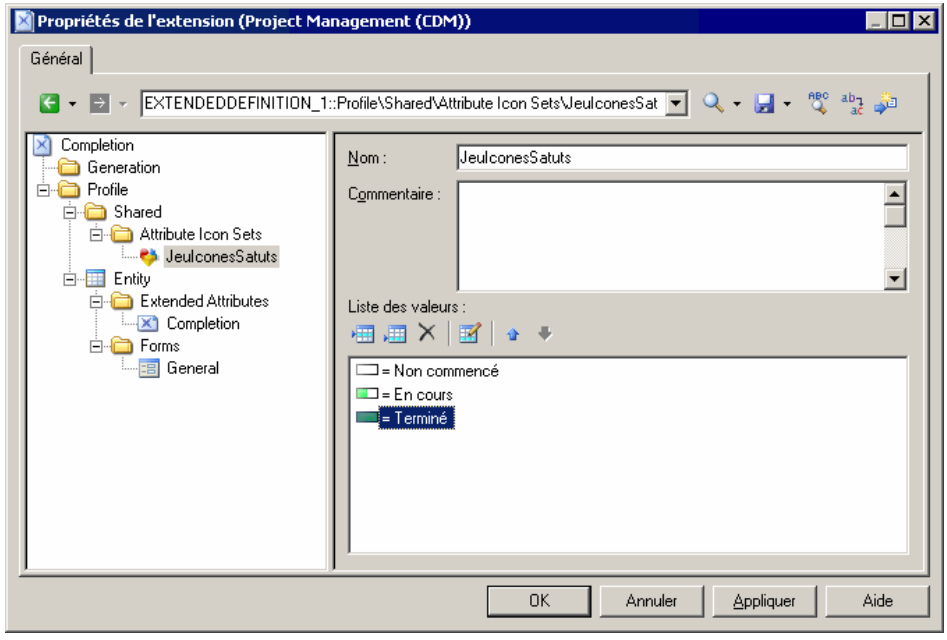
3. Cliquez sur **Appliquer** pour enregistrer vos modifications.

   Le nouveau type partagé est disponible pour n'importe quel attribut étendu dans la zone **Type de données**. Vous pouvez également définir une liste de valeurs pour un attribut étendu particulier directement dans cette zone (voir *Attributs étendus (Profile)* à la page 45).

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 (voir *Attributs étendus (Profile)* à la page 45).
2. Sélectionnez un type de données standard ou un type d'attribut étendu (voir *Création d'un type d'attribut étendu* à la page 52).
3. Si nécessaire, spécifiez une liste de valeurs possibles et une valeur par défaut.
4. 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 (=*).
5. 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.

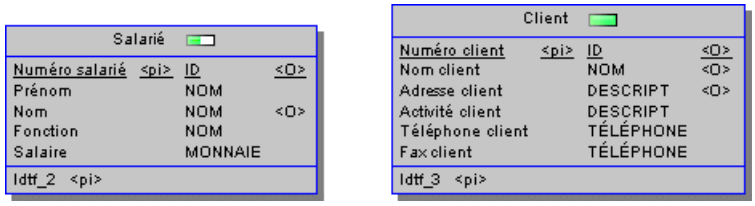


Remarque : Par défaut, la zone **Opérateur de filtre** est définie à =, et chaque icône correspond exactement à une valeur possible. Pour faire en sorte qu'une icône puisse correspondre à plusieurs valeurs, utilisez l'opérateur `Entre` ou un autre opérateur avec une **Valeur de filtre** appropriée. Par 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`

6. Si nécessaire, ajoutez l'attribut dans un formulaire (voir *Formulaires (Profile)* à la page 62), pour permettre aux utilisateurs de modifier sa valeur.
7. Cliquez sur **OK** pour enregistrer vos modifications et revenir au modèle.
8. 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 générales > Modélisation avec PowerAMC > Diagrammes, matrices et symboles > Préférences d'affichage*.

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



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

Spécifiez le type de données `[Objet]` afin de permettre aux utilisateurs de sélectionner un autre objet comme valeur pour un attribut. Vous devez spécifier un **Type d'objet** (métaclasse) vers lequel lier, et vous pouvez également spécifier le cas échéant un **Stéréotype d'objet** afin de filtrer les objets disponibles pour sélection, ainsi qu'un nom de **Collection inverse**, qui sera affiché dans l'onglet **Dépendances** de la feuille de propriétés de l'objet référencé.

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`. Lorsque vous définissez la propriété **Propriétaire** d'une table, celle-ci est répertoriée sur l'onglet **Dépendances** de la feuille de propriétés de l'utilisateur, sous le nom de collection inverse `Tables ayant un propriétaire`.

Collections et compositions étendues (Profile)

Les collections étendues définissent la possibilité d'associer une instance d'objet avec un groupe d'autres objets du type spécifié. Les compositions étendues définissent une connexion parent-enfant entre une instance d'objet et un groupe de sous-objets dérivés de la métaclasse `ExtendedSubObject`.

Pour les collections étendues, l'association entre les objets parent et enfant est relativement faible, de sorte que si vous copiez ou déplacez l'objet parent, les objets associés ne sont pas copiés ou déplacés, mais la connexion est préservée (si nécessaire, au moyen de raccourcis). Par exemple, vous pouvez associer des documents contenant des spécifications de cas d'utilisation avec les différents packages d'un modèle en créant une collection étendue sous la métaclasse `Package` et en spécifiant `FileObject` comme métaclasse cible.

Pour les compositions étendues, l'association est plus forte. Les sous-objets ne peuvent être créés que sous l'objet parent et sont déplacés ou copiés et/ou supprimés avec leur parent.

La collection ou composition est affichée sous la forme d'un nouvel onglet dans la feuille de propriétés de l'instance d'objet. Les feuilles de propriétés des objets référencés dans une collection contenant l'instance d'objet qui possède la collection sur leur onglet **Dépendances**.

1. Pointez sur une métaclasse, un stéréotype ou un critère dans la catégorie Profile, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau > Collection étendue** or **Composition étendue**.

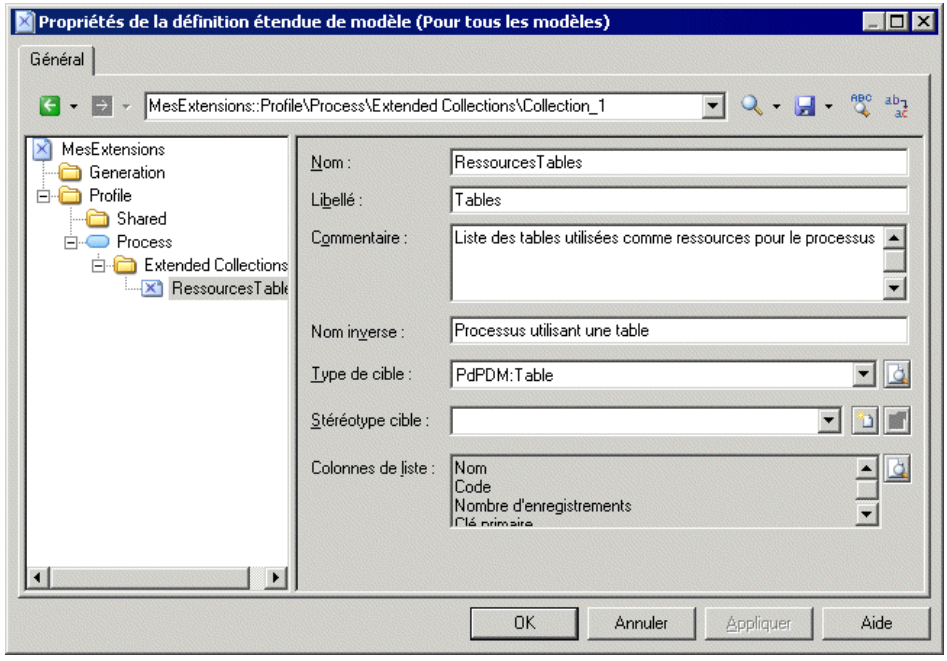
   Remarque : Si vous définissez la collection ou la composition sous un stéréotype ou un critère, son onglet s'affiche uniquement si l'instance de métaclasse porte le stéréotype ou correspond au critère approprié.

2. Spécifiez les propriétés suivantes :

Propriété	Description
Nom	Spécifie le nom de la collection ou composition étendue.
Libellé	Spécifie le nom d'affichage de la collection, qui sera utilisé comme nom pour l'onglet associé à la collection dans la feuille de propriétés de l'objet parent.
Commentaire	[facultatif] 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.
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

Propriété	Description
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 cliquer sur l'outil **Créer** à droite de cette zone pour en créer un nouveau.
Colonnes de ligne	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 pour ajouter ou supprimer des colonnes.

3. 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.

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

Collections calculées (Profile)

Les collections calculées définissent une connexion en lecture seule entre une instance d'objet et un groupe d'objets du type spécifié. La collection affiche un sous-onglet sur l'onglet **Dépendances** de la feuille de propriétés de l'objet. La logique de la collection est définie à l'aide de VBScript.

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.

Vous pouvez boucler sur les collections calculées en utilisant le langage de génération par templates (voir *Accès aux collections de sous-objets ou d'objets associés* à la page 267) Vous pouvez utiliser les collections calculées pour affiner l'analyser d'impact en évaluant plus précisément l'impact d'une modification. 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 ont le même type de données.

Remarque : Les collections calculées, à la différence des collections étendues (voir *Collections et compositions étendues (Profile)* à la page 54) ne peuvent pas être modifiées par l'utilisateur.

1. Pointez sur une métaclasse, un stéréotype ou un critère dans la catégorie Profile, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau > Collection calculée**.
2. Spécifiez les propriétés suivantes :

Propriété	Description
Nom	Spécifie le nom de la collection calculée à utiliser dans les scripts.
Libellé	Spécifie le nom d'affichage de la collection, qui s'affiche sous la forme du nom de l'onglet associé à la collection dans la feuille de propriétés de l'objet parent.
Commentaire	[facultatif] Décrit la collection calculée.
Type de cible	Spécifie 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** pour sélectionner une métaclasse.

Propriété	Description
Stéréotype cible	[facultatif] Spécifie un stéréotype pour filtrer le type de cible. Vous pouvez sélectionner un stéréotype existant dans la liste, ou en saisir un nouveau.
Liste des colonnes	Spécifie les colonnes affichées par défaut sur l'onglet de feuille de propriétés de la collection.

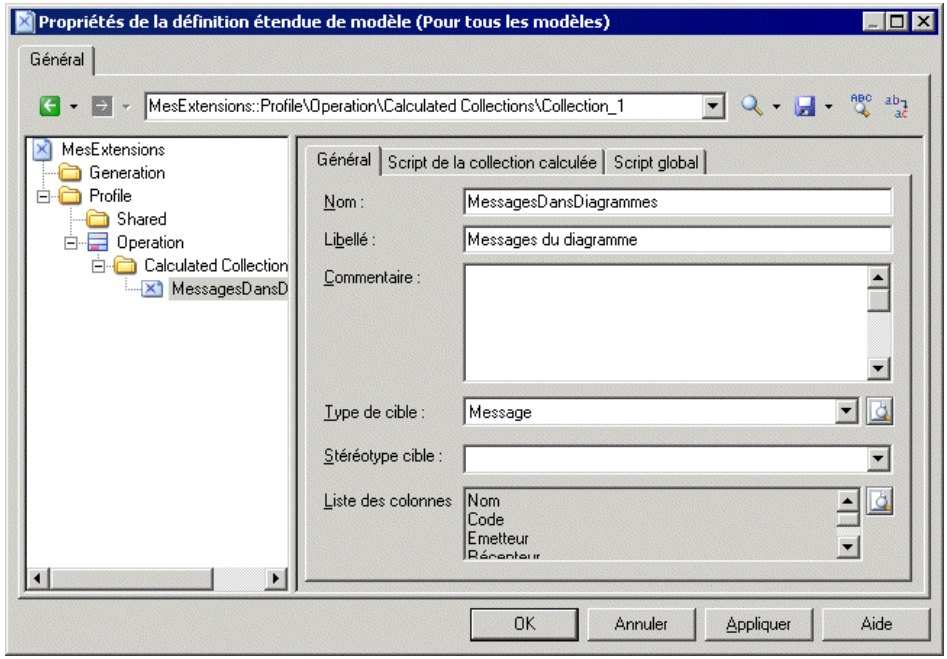
3. Cliquez sur l'onglet **Script de la collection calculée** et saisissez un script qui va calculer quels objets vont former la collection.

   Le script suivant recrée la liste sur le sous-onglet **Références sortantes** de l'onglet **Dépendances** sur la feuille de propriétés d'une table :

```
Function %Collection%(obj, coll) ' Requis
   dim r
   For each r in obj.outreferences
      coll.Add r                  ' Renseigne la collection
   Next
   %Collection% = True            ' Requis
End Function                      ' Requis
```

   Remarque : Vous pouvez réutiliser les fonctions de l'onglet **Script global** (voir *Script global (Profile)* à la page 115) mais vous devez savoir que si vous déclarez des *variables globales*, ces dernières ne seront pas réinitialisée chaque fois que la collection sera calculée, et qu'elles conserveront leur valeur jusqu'à ce que vous modifiiez le fichier de ressource, ou jusqu'à la fin de la session de PowerAMC. Ce comportement peut provoquer des erreurs, tout particulièrement si des variables référencent des objets qui peuvent être modifiés ou supprimés. Assurez-vous de réinitialiser la variable globale si vous ne souhaitez pas réutiliser la valeur de sa précédent exécution.

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



5. Pour visualiser la collection, affichez la feuille de propriétés d'un instance de métaclasse dans l'onglet **Dépendances**, puis sélectionnez le sous-onglet approprié.
6. [facultatif] Ajoutez la collection dans vos rapports sur des modèles. Les collections calculées sont automatiquement disponibles dans le nouvel Editeur de rapport sous la forme de listes sous le livre ou la liste de chaque métaclasse (voir *Guide des fonctionnalités générales > Stockage, partage et documentation des modèles > Rapports > Ancienne version de l'Editeur de rapport > Modification de la collection d'un élément*).

Matrices de dépendances (Profile)

Les matrices de dépendance permettent de passer en revue et de créer des liens entre tous types d'objet. Vous spécifiez une métaclasse pour les lignes de la matrice, et la même ou une autre métaclasse pour les colonnes. Le contenu des cellules est alors 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 :

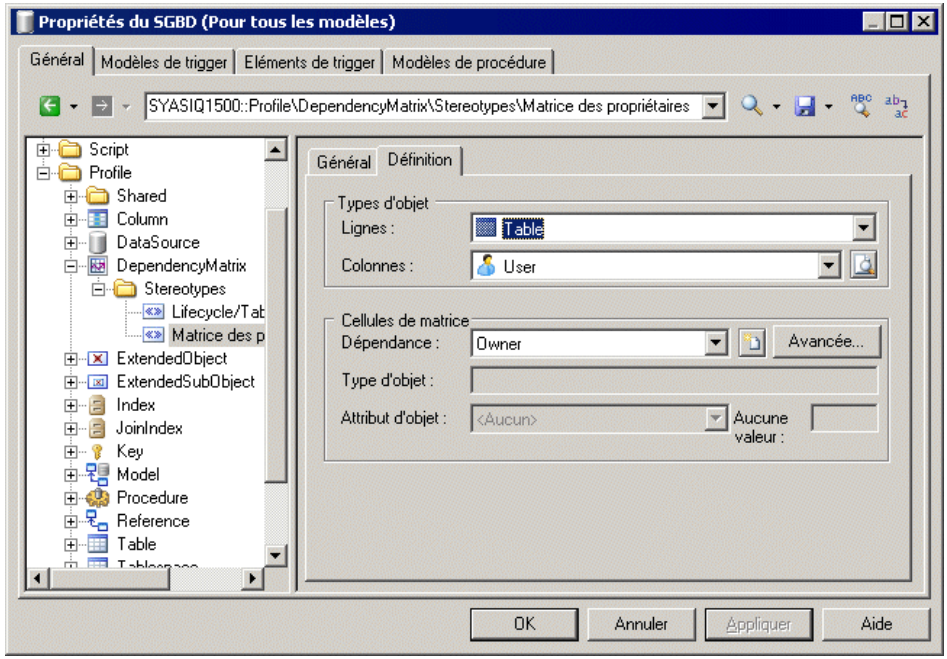
- Classes et autres classes de MOO – connectées par des liens d'association
- Tables et utilisateurs de MPD – connectées par la collection Propriétaire

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

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

1. Pointez sur la catégorie **Profile** cliquez le bouton droit de la souris, puis sélectionnez **Ajouter une matrice de dépendance** pour ajouter la métaclasse `DependencyMatrix` dans le profil et créer un stéréotype sous cette métaclasse, dans lequel vous allez définir les propriétés de la matrice.
2. Sur l'onglet **Général**, 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 et indiquer comment elles sont associées à l'aide des propriétés suivantes.

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. 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 54) reliant vos objets, ou sur le bouton **Avancée** pour spécifier un chemin de dépendance complexe (voir *Spécification des dépendances avancées* à la page 61). Pour certaines dépendances, le **Type d'objet** sur lequel la dépendance est basé sera affiché, et vous pouvez sélectionnez 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.



4. Cliquez sur **OK** pour enregistrer votre matrice et fermer l'Editeur 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 > Modélisation avec PowerAMC > Diagrammes, matrices et symboles > Matrices de dépendances*.

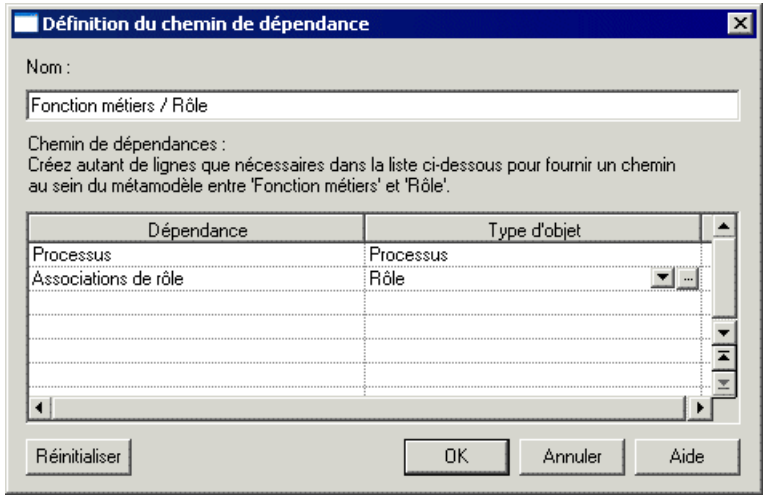
Spécification des dépendances avancées

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 :



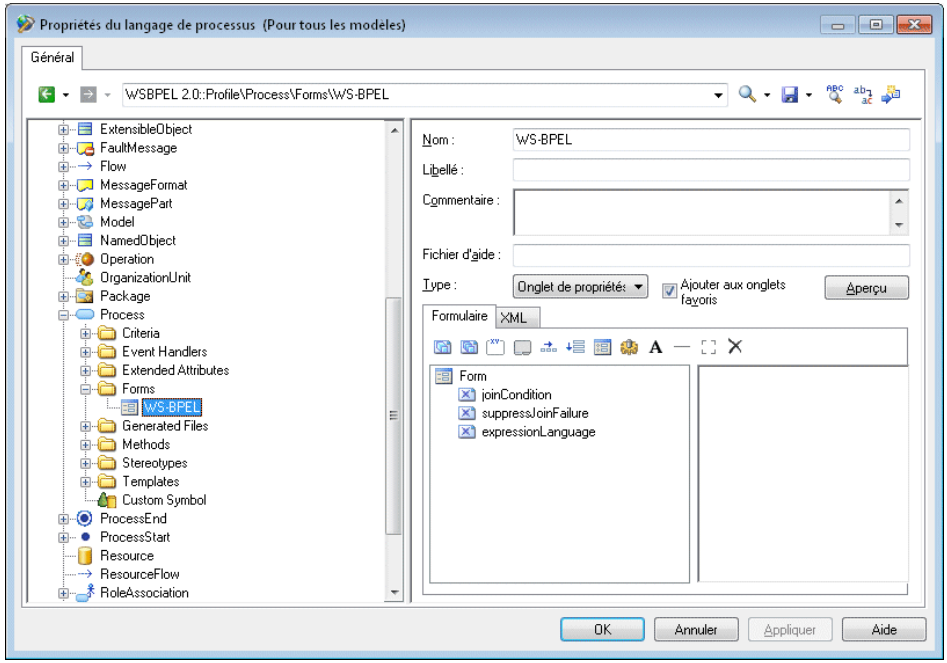
Formulaires (Profile)

Les formulaires présentent des attributs standard et étendus et les collections sous la forme d'onglets de feuille de propriétés ou peuvent être utilisés pour créer des boîtes de dialogue lancées depuis des menus ou des boutons de feuille de propriétés.

Remarque : A moins que vous ne les ajoutiez depuis un formulaire, les attributs les attributs étendus s'affichent par ordre alphabétique sur l'onglet **Attributs étendus** de la feuille de propriétés de l'objet. 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. Si vous associez tous vos attributs étendus à un formulaire, l'onglet **Attributs étendus** n'est pas affiché.

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

Remarque : Si vous définissez un onglet de propriétés sous un stéréotype ou un critère, cet onglet n'est affiché que lorsque l'instance de métaclasse porte le stéréotype ou répond au critère.



2. Spécifiez les propriétés suivantes :

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'onglet de feuille de propriétés ou dans la barre de titre de la boîte de dialogue.
Commentaire	Fournit des informations supplémentaires sur le formulaire.

Propriété	Description

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\Lisezmoi.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.

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 64).
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.

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

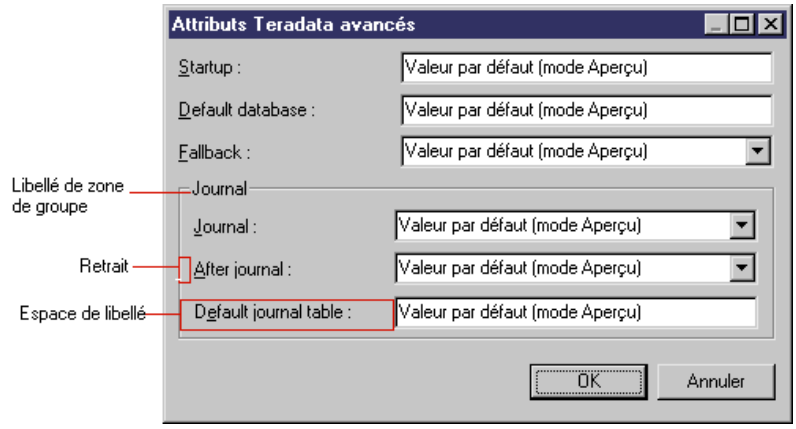
Vous insérez des contrôles dans votre formulaire en utilisant les outils de la barre d'outils située en haut de l'onglet Formulaire. Vous pouvez réorganiser les contrôles dans l'arborescence des contrôles en les faisant glisser. Pour placer un contrôle dans un conteneur de contrôle (zone de groupe ou 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.

Les outils suivants sont disponibles :

Outil	Description
	**Ajouter un attribut/une collection** – affiche une boîte de dialogue de sélection qui permet de sélectionner des attributs standard ou étendus ou des collections qui appartiennent à la métaclasse pour les insérer dans le formulaire. Si vous ne spécifiez aucune libellé, le nom d'attribut ou de collection est utilisé comme libellé de formulaire. Si vous avez saisi un commentaire, il s'affiche sous la forme d'info-bulle. 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. Les collections sont affichées sous la forme de grilles standard avec tous les outils appropriés.
	**Ajouter une zone de groupe** - insère une zone de groupe, destinée à contenir d'autres contrôles dans une zone nommée.
	**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/verticale** - insère une disposition horizontale ou verticale. Pour disposer les contrôles côte à côte, faites-les glisser sur la disposition horizontale dans la liste. Pour disposer les contrôles les uns au-dessus des autres, faites-les glisser sur la disposition verticale dans la liste. Les dispositions verticales et horizontales sont souvent utilisées en combinaison afin de fournir des colonnes de contrôle.
	**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 73).
	**Ajouter un bouton de méthode** - affiche une boîte de dialogue de sélection qui permet de sélectionner une ou plusieurs méthodes appartenant à la métaclasse à associer au formulaire via des boutons. Le fait de cliquer sur un bouton appelle la méthode. Si vous ne saisissez pas un libellé, le nom de méthode est utilisé comme libellé pour le bouton. Si vous avez saisi un commentaire, il s'affiche sous la forme d'info-bulle.
	**Ajouter une zone d'édition/zone d'édition multiligne** [boîtes de dialogue uniquement] insère une zone d'édition ou zone d'édition multiligne.
	**Ajouter une liste modifiable/zone de liste/case à cocher** [boîtes de dialogue uniquement] - insère une liste modifiable, une zone de liste ou case à cocher.
	**Ajouter du texte/une ligne de séparation/une zone d'espacement** - insère le contrôle décoratif approprié. La ligne de séparation est verticale lorsque le contrôle parent est une disposition verticale.
	**Supprimer** – supprime le contrôle sélectionné.

Sélectionnez un contrôle pour spécifier des propriétés qui contrôlent son format et son 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 92).
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 : • <Aucun> - 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 enfant.

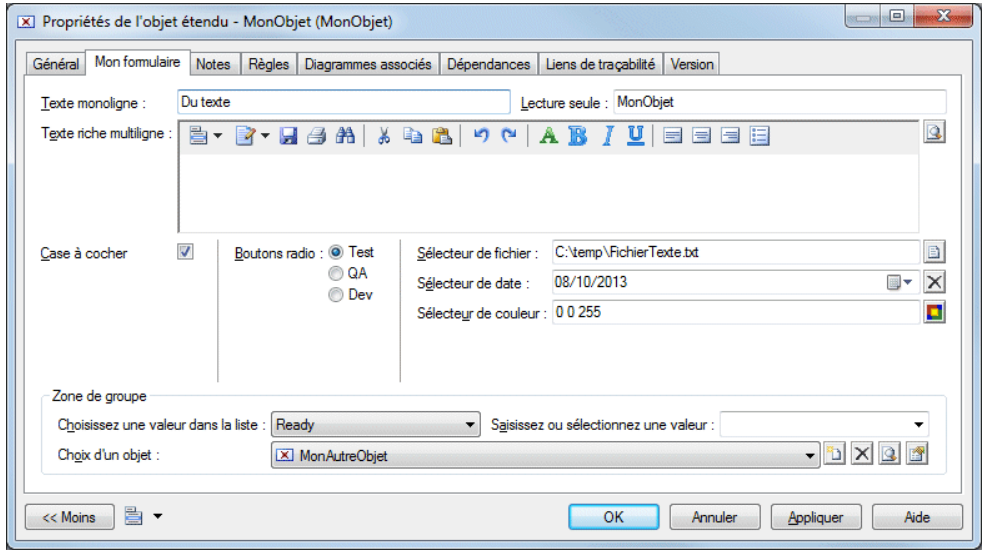
Propriété	Définition
Espace de libellé	[contrôles de conteneur uniquement] Spécifie l'espace en pixels réservé pour l'affichage des libellés des contrôles enfant entre le retrait du conteneur et les zones de contrôle. Pour aligner les contrôles sur les contrôles d'un conteneur précédent, saisissez une 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 *Attributs étendus (Profile)* à la page 45).
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 *Attributs étendus (Profile)* à la page 45).

Propriété	Définition
Liste exclusive	[liste modifiables] Spécifie que seules les valeurs définies dans la **Liste des valeurs** peuvent être saisies dans la liste modifiable.
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 / vertical	Spécifie que le contrôle peut être redimensionné horizontalement ou, dans le cas d'un contrôle multiligne, verticalement, lors que la feuille de propriétés ou la boîte de dialogue 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 entre une case à cocher ou une ligne ou une colonne de boutons radio, tandis que 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.

Exemple : Création de contrôles de formulaire courants

Dans cet exemple, nous vous indiquons comment créer les contrôles les plus courants pour présenter des attributs dans vos formulaires.

Le formulaire suivant contient certains des contrôles les plus fréquemment utilisés pour présenter des attributs dans vos formulaires :



Pour créer un contrôle, vous devez créer un attribut étendu (voir *Attributs étendus (Profile)* à la page 45), puis l'ajouter au formulaire (voir *Ajout d'attributs étendus et d'autres contrôles dans votre formulaire* à la page 64). Vous pouvez organiser vos contrôles à l'aide de dispositions horizontales et verticales, de séparations, d'espacements, de texte libre et de zones de groupe. PowerAMC va automatiquement ajouter les outil supplémentaires appropriés à vos contrôles, tels que les outils **Créer**, **Supprimer**, **Sélectionner**, et **Propriétés** à droite d'un contrôle d'objet.

Pour créer les contrôles affichés :

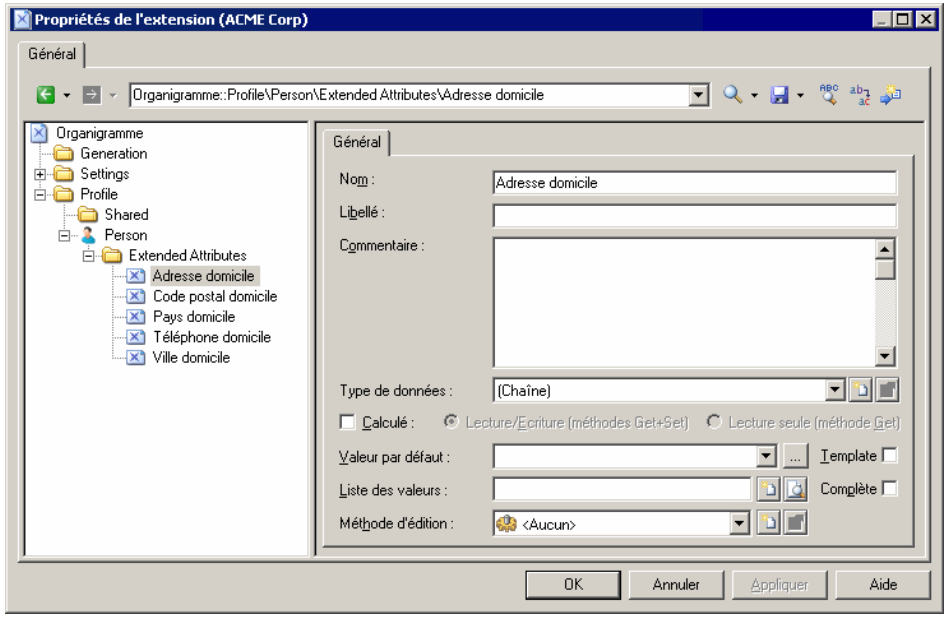
Contrôle	Requiert
Texte monoligne	Sélectionnez le type de données `Chaîne`, `Mot de passe` (masque les valeurs saisies), ou `Réel`, `Hexadécimal`, ou `Entier` pour votre attribut.
Lecture seule	Sélectionnez n'importe quel type de données, cochez les cases **Calculé** et **Lecteur seule (méthode Get)**, puis saisissez le script nécessaire pour calculer la valeur qui sera affichée.
Texte riche multiligne	Sélectionnez le type de données `Texte` et le format de texte `RTF`. Vous pouvez sélectionner d'autres formats de texte pour afficher différents types de code ou de texte simple.
Case à cocher	Sélectionnez le type de données `Booléen`. Utilisez la zone **Valeur par défaut** pour spécifier si la case doit être cochée ou non par défaut.
Boutons radio	Sélectionnez le type de données approprié, saisissez une liste de valeurs, puis sélectionnez l'option **Complète**. Lorsque vous ajoutez votre attribut sur le formulaire, sélectionnez colonne ou ligne dans la zone **Afficher**.

Contrôle	Requiert
Sélecteur de fichier, de date ou de couleur	Sélectionnez le type de données `Fichier`, `Date` ou `Couleur`.
Choisissez une valeur dans la liste ou Saisissez ou sélectionner une valeur	Sélectionnez le type de données approprié et saisissez une liste de valeurs pour permettre à l'utilisateur de sélectionner une valeur dans la liste ou de saisir leur propre valeur. Sélectionnez l'option **Complète** pour forcer l'utilisateur à sélectionner dans la liste.
Choix d'un objet	Sélectionnez le type de données `Objet` puis sélectionnez un type d'objet, ainsi le cas échéant qu'un stéréotype d'objet (voir *Liaison d'objets à l'aide d'attributs étendus* à la page 54).

Exemple : Création d'un onglet de feuille de propriétés

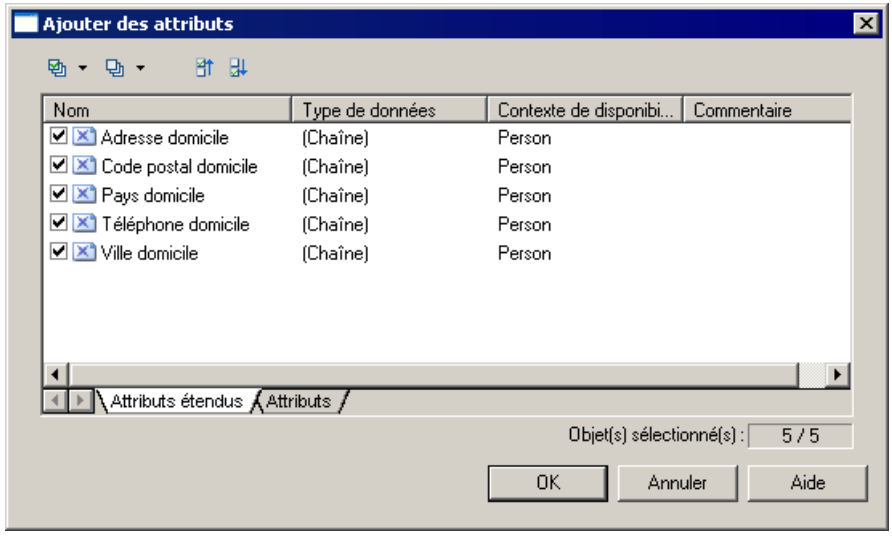
Dans cet exemple, nous allons créer un nouvel onglet de propriétés pour la métaclasse de MAE Person, ce afin d'afficher des attributs étendus que nous définissons afin d'y stocker des informations personnelles.

1. Créez un nouveau fichier d'extension (voir *Création d'un fichier d'extension* à la page 12) dans un MAE, ajoutez la métaclasse `Person` (voir *Métaclasses (Profile)* à la page 35), puis définissez cinq attributs étendus (voir *Attributs étendus (Profile)* à la page 45) destinés à contenir les détails relatifs à l'adresse du domicile :

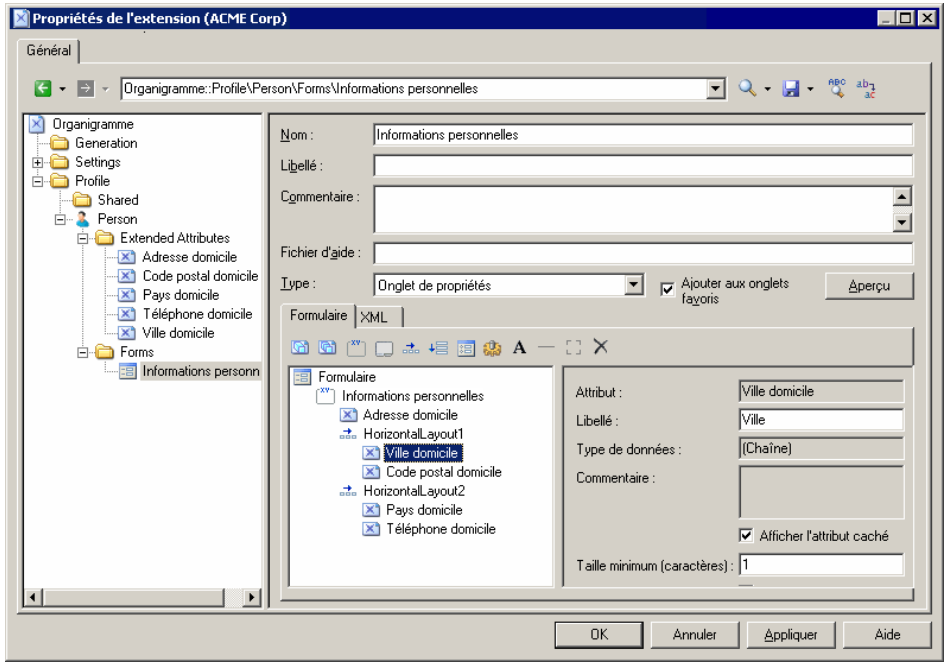


2. Pointez sur la métaclasse `Person`, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau > Formulaire**, saisissez `Informations personnelles` dans la zone

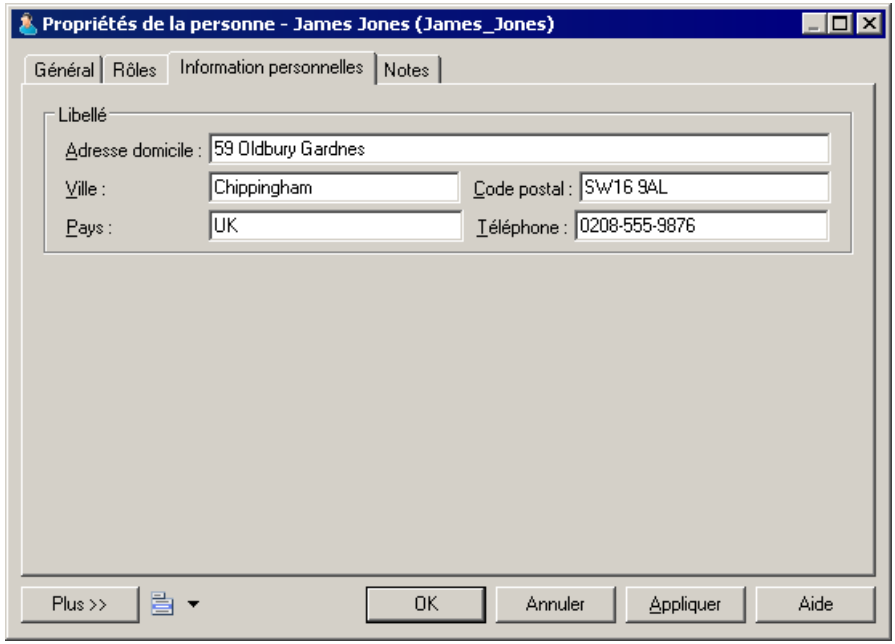
Nom, sélectionnez `Onglet de propriétés` dans la liste **Type**, puis cliquez sur **Ajouter un attribut** afin de sélectionner tous les nouveaux attributs étendus à inclure dans le formulaire :



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 :

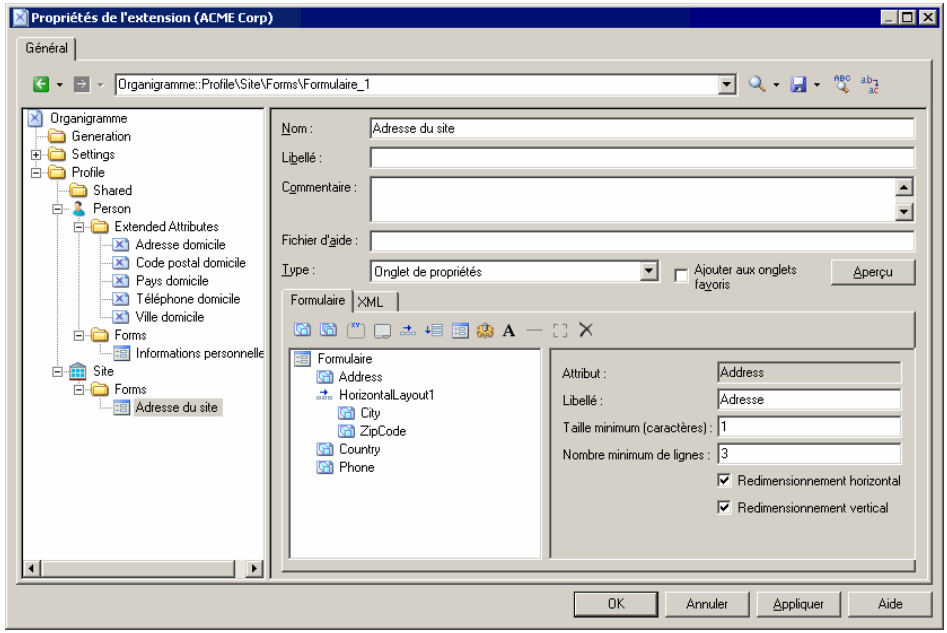


Exemple : Inclusion d'un formulaire dans un autre formulaire

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.

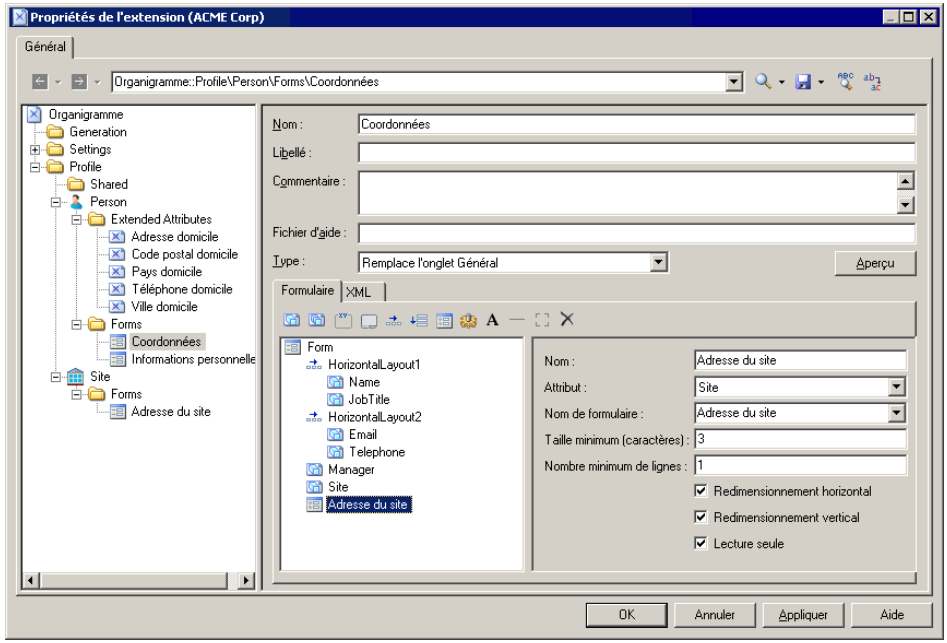
Cet exemple utilise le fichier d'extension créé dans *Exemple : Création d'un onglet de feuille de propriétés* à la page 70.

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 :

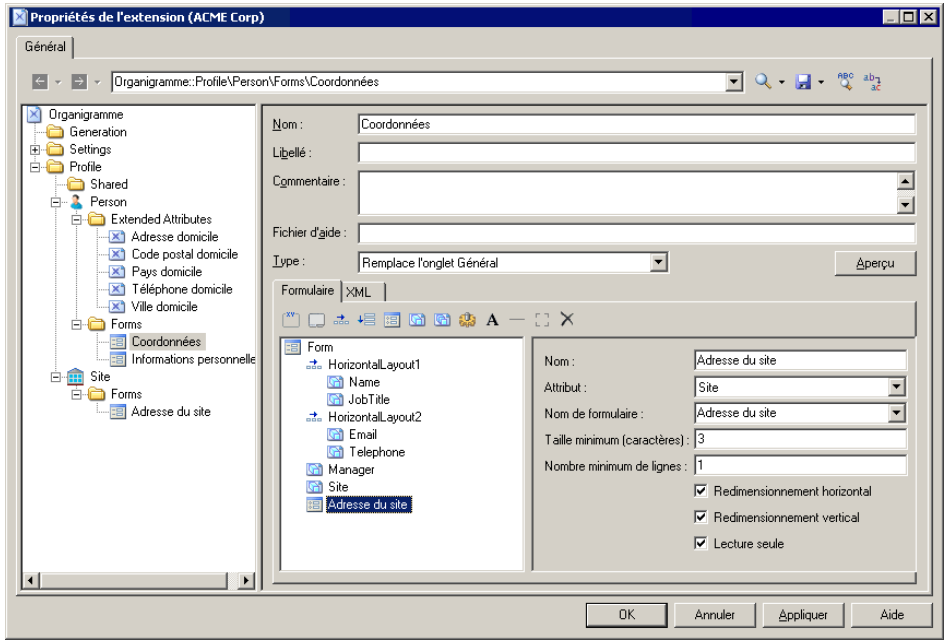


3. Créez un formulaire sous la métaclasse `Person`, sélectionnez `Remplace l'onglet Général` dans la liste **Type**, puis changez le nom en `Coordonnées`.
4. Supprimez les attributs non souhaités de la liste, et réorganisez les attributs restants que vous souhaitez voir s'afficher, y compris l'attribut `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 la zone **Nom du formulaire**. 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 :

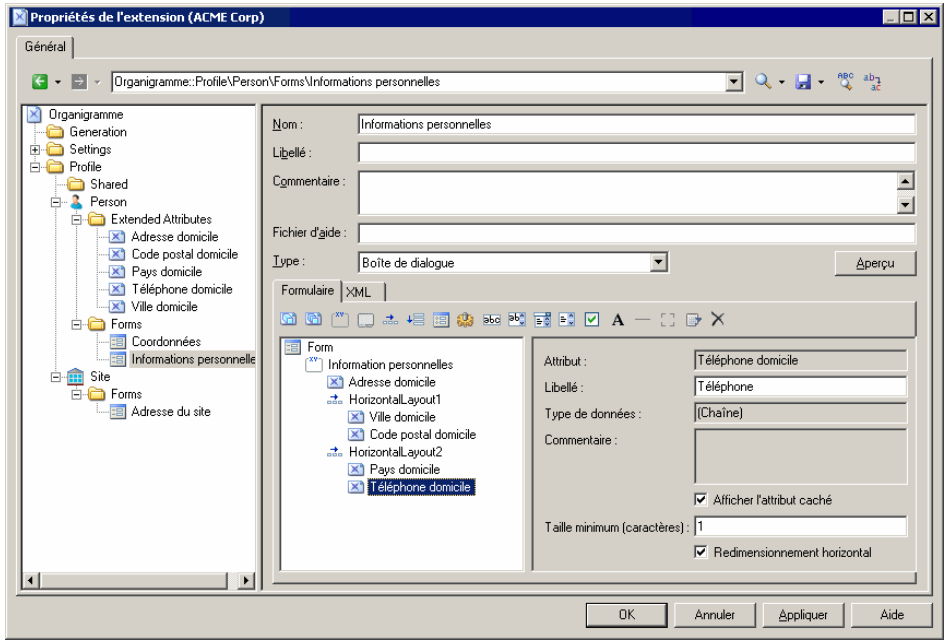


Exemple : Ouverture d'une boîte de dialogue à partir d'une feuille de propriétés

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 73.

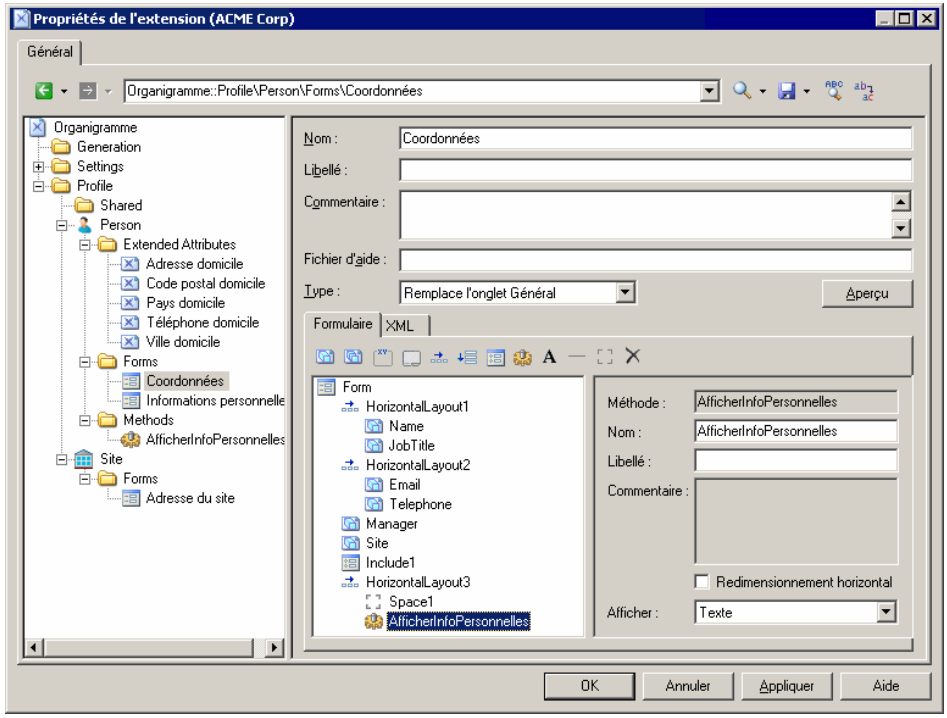
1. Affichez le formulaire `Informations personnelles` sous la métaclasse `Person`, puis sélectionnez `Boîte de dialogue` dans la zone **Type**, afin de la transformer d'onglet de feuille de propriétés en boîte de dialogue indépendante :



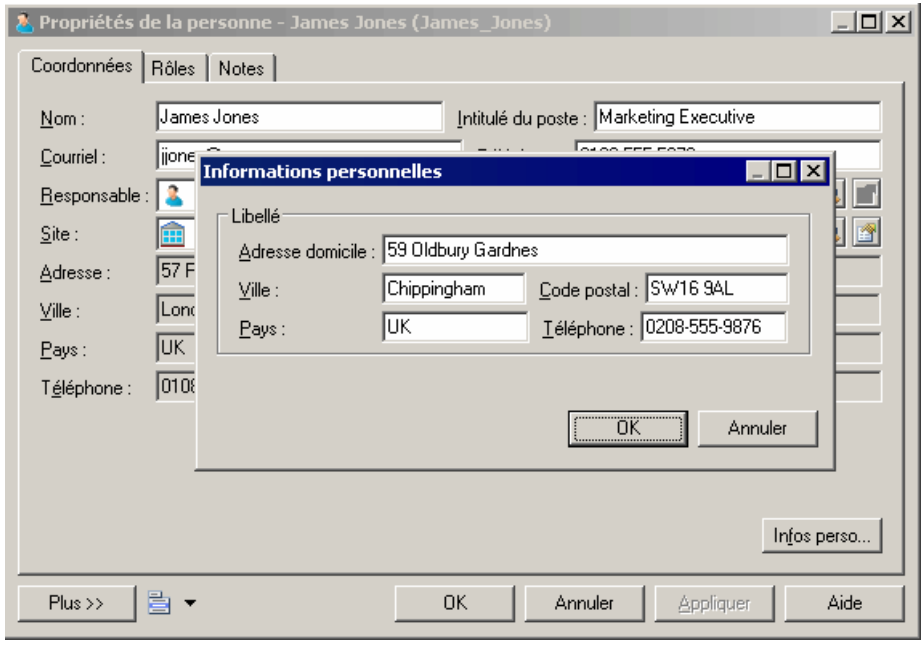
2. Pointez sur la métaclasse Person, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau > Méthode**. Saisissez le nom `AfficherInfoPersonnelles`, puis cliquez sur l'onglet **Script de méthode** et saisissez le script suivant :

```
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 :



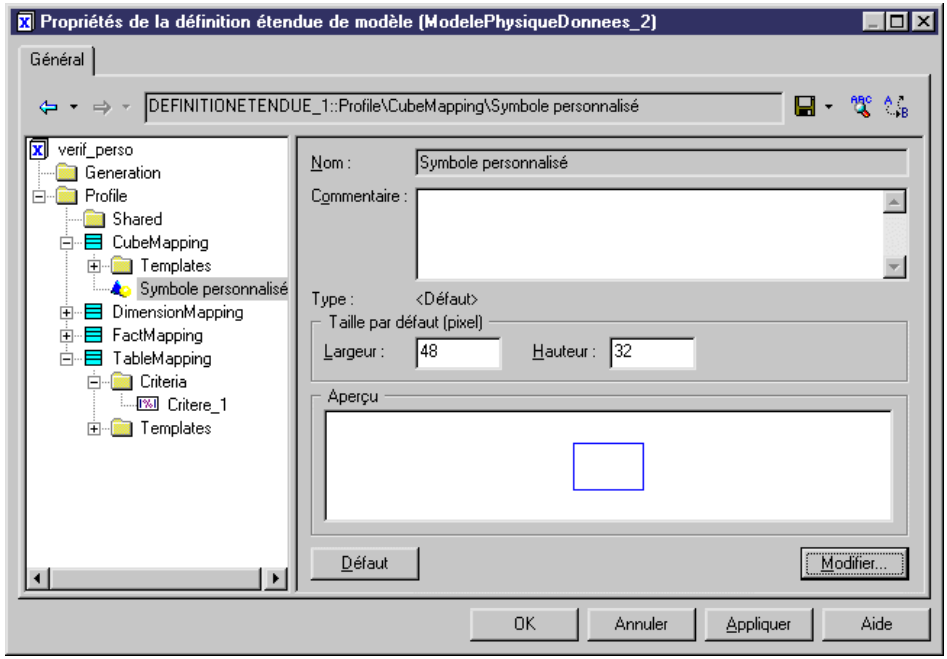
4. Saisissez `Infos perso...` dans la zone **Libellé**, puis cliquez sur **OK** pour enregistrer vos modifications et revenir au modèle. Dorénavant, lorsque vous affichez la feuille de propriétés d'une personne, l'onglet **Coordonnées** contient un bouton **Infos perso...** qui permet d'afficher la boîte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



Symboles personnalisés (Profile)

Les symboles personnalisés modifient l'apparence des symboles d'objet dans les diagrammes, ainsi que les informations qu'ils contiennent. Vous pouvez choisir d'imposer certains aspects du format et du contenu de symbole, tout en laissant aux utilisateurs la possibilité d'en modifier d'autres.

1. Pointez sur une métaclasse, un stéréotype ou un critère dans la catégorie Profile, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau** > **Symbole personnalisé**.



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.

   Remarque : Si vous personnalisez le style de ligne et les flèches d'un symbole de lien (par exemple, une référence de MPD), vos styles remplacent ceux sélectionnés dans la boîte de dialogue Préférences d'affichage, et risquent de provoquer confusion et incohérence dans le modèle. Pour assurer la cohérence dans un modèle gouverné par une notation, sélectionnez `Notation` pour les propriétés **Style** et **Flèche** sur l'onglet **Style de trait**.

   Pour plus d'informations sur la boîte de dialogue Format de symbole (y compris sur les options relatives aux options de symbole personnalisés qui permettent de contrôler les options de format par défaut pour le symbole et de décider onglet par onglet si les utilisateurs peuvent les éditer) voir *Guide des fonctionnalités générales > Modélisation avec 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)

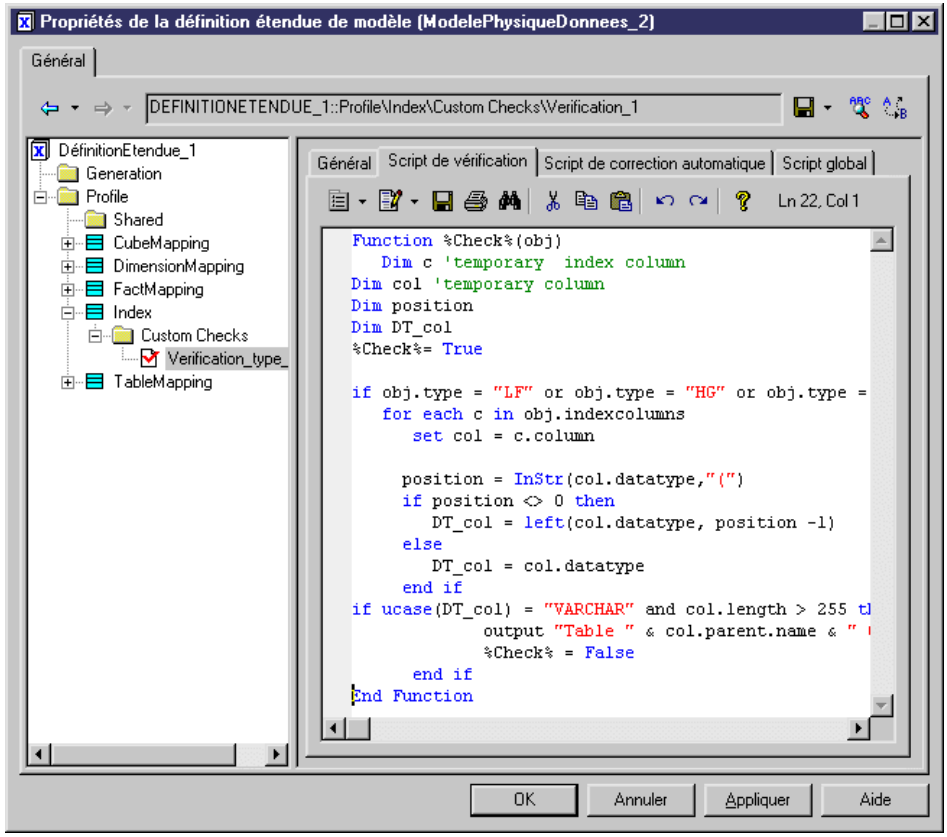
Les vérifications personnalisées définissent des règles supplémentaires pour valider le contenu de vos modèles. La logique des vérifications est définie avec VBScript. Les vérifications personnalisées s'affichent aux côtés des vérifications standard dans la boîte de dialogue **Paramètres de vérification de modèle**.

Les vérifications personnalisées s'affichent avec les vérifications standard dans la boîte de dialogue **Paramètres de vérification de modèle** (voir *Guide des fonctionnalités générales > Modélisation avec PowerAMC > Objets > Vérification de modèles*).

1. Pointez sur une métaclasse, un stéréotype ou un critère dans la catégorie Profile, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau > Vérification personnalisée**.
2. Spécifiez les propriétés suivantes :

Paramètre	Description
Nom	Spécifie le nom de la vérification personnalisée, qui 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 pour l'identifier de façon unique.
Commentaire	Fournit une description de la vérification personnalisée.
Message d'aide	Spécifie le texte à afficher dans la boîte de message qui s'ouvre lorsque l'utilisateur pointe sur la vérification, clique le bouton droit de la souris et sélectionne **Aide**.
Message de résultats	Spécifie le texte à afficher dans la fenêtre **Résultats** lors de l'exécution de la vérification.
Sévérité par défaut	Spécifie si la vérification est définie par défaut comme une erreur (problème majeur qui interrompt la génération) ou un avertissement (problème mineur ou simple recommandation).
Exécuter la vérification par défaut	Spécifie que la vérification est sélectionnée par défaut dans la boîte de dialogue **Paramètres de vérification de modèle**.
Exécuter la correction automatique	Spécifie qu'une correction automatique est disponible pour cette vérification (voir *Exemple : Correction automatique de MPD* à la page 83).
Exécuter la correction automatique par défaut	Spécifie que la correction automatique est exécutée par défaut.

3. Cliquez sur l'onglet **Script de vérification** et saisissez votre script (voir *Exemple : Vérification personnalisée de MPD* à la page 82. Vous pouvez accéder aux fonctions de bibliothèque partagées et aux attributs statiques définis pour réutilisation dans le fichier de ressource à partir de l'onglet **Script global** (voir *Script global (Profile)* à la page 115).



4. Si vous souhaitez définir une correction automatique, cliquez sur l'onglet **Script de correction automatique**, puis saisissez votre (voir *Exemple : Correction automatique de MPD* à la page 83.
5. Cliquez sur **Appliquer** pour enregistrer vos modifications.

   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. Vos vérifications personnalisées sont affichées dans la boîte de dialogue **Paramètres de vérification de modèle** avec les vérifications de modèle standard. Si des erreurs sont détectées lors de la vérification personnalisée, les actions suivantes sont proposées à l'utilisateur :
   - Ignorer- Saute le script problématique et continue avec les vérifications suivantes.
   - Ignorer tout - Saute ce script problématique ainsi que les autres qui pourraient se présenter et continue avec les vérifications suivantes.
   - Annuler - Arrête la vérification du modèle.
   - Déboguer - Arrête la vérification du modèle et affiche l'Editeur de ressources sur la ligne de script contenant le problème.

Exemple : Vérification personnalisée de MPD

Vous saisissez le script de la vérification personnalisée dans l'onglet **Script de vérification** en utilisant VBScript. Dans cet exemple, nous allons rédiger un script afin de vérifier que les index SAP® Sybase® IQ de type HG, HNG, CMP ou LF ne sont pas liés à des colonnes ayant un type de données VARCHAR d'une longueur supérieure à 255.

Ce script est initialisé avec la ligne suivante, que vous ne devez pas modifier :

```
Function %Check%(obj)
```

Au moment de l'exécution, la variable `%Check%` est remplacée par la concaténation des noms du fichier de ressource, de la métaclasse courante, du stéréotype ou critère ainsi que celui de la vérification elle-même défini dans l'onglet **Général**, les éventuels espaces sont remplacés par un trait de soulignement. Le paramètre `obj` contient l'objet vérifié.

Nous commençons par définir un certain nombre de variables après la définition de fonction par défaut :

```
Dim c 'colonne index temporaire
Dim col 'colonne temporaire
Dim position
Dim DT_col
```

Nous saisissons ensuite le corps de la fonction, qui commence par définir `%Check%` à true (ce qui signifie que l'objet passe le test) puis procède à l'itération sur chacune des colonnes associées à l'index et teste leur type de données. Si une colonne a un type varchar dont la longueur dépasse 255, le script produit un message et définit la vérification à false (l'objet n'a pas satisfait au test) :

```
%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 & ", colonne " & col.name & " :
Le type de données n'est pas compatible avec l'index " & obj.name & "
type " & obj.type
     %Check% = False
  end if
```

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 331.

Exemple : Correction automatique de MPD

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** en utilisant du code VBScript. Dans cet exemple, nous allons rédiger un script destiné à réparer un index Sybase IQ lié à un type de données invalide.

Le script est initialisé avec la ligne suivante, qui ne doit pas être modifiée :

```
Function %Fix%(obj, outmsg)
```

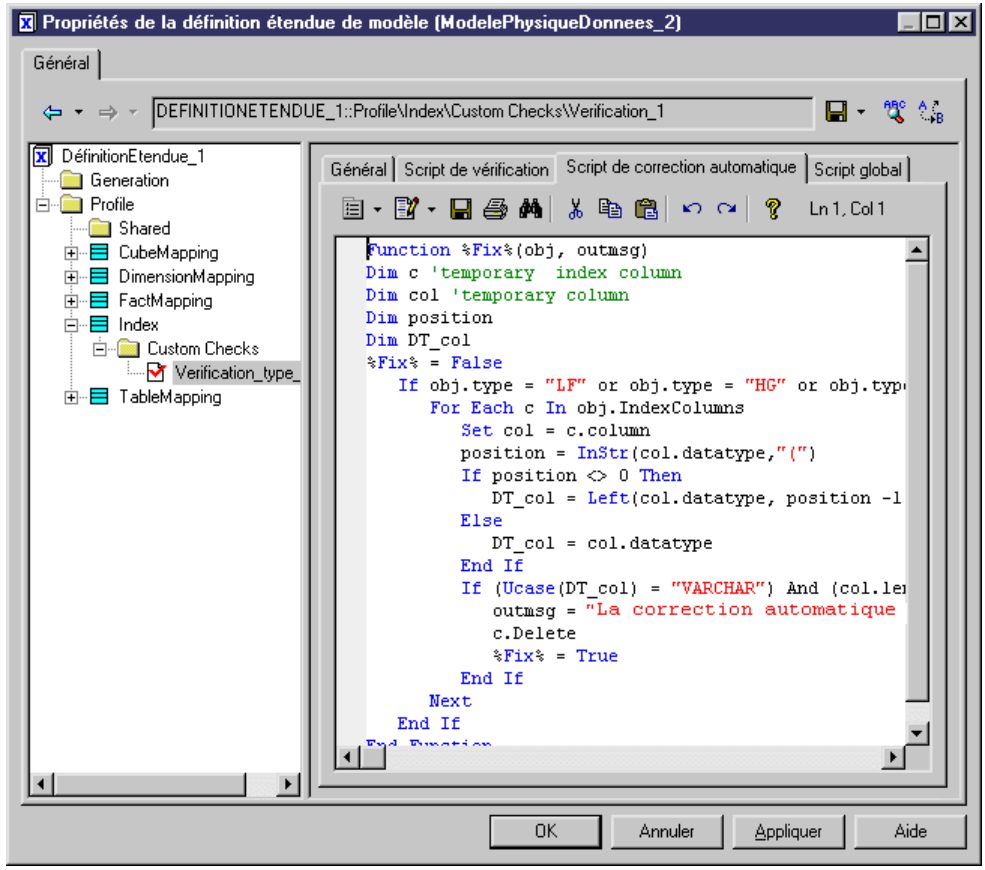
Au moment de l'exécution, la variable `%Fix%` est remplacée par le nom de la correction automatique. Le paramètre `obj` contient l'objet en cours de vérification et `outmsg`, le message à produire.

Nous allons commencer par définir un certain nombre de variables après la définition de fonction par défaut :

```
Dim c 'colonne index temporaire
Dim col 'colonne temporaire
Dim position
Dim DT_col
```

Ensuite, nous saisissons le corps de la fonction, qui commence en définissant la variable `%Fix%` à false (ce qui signifie que rien n'est fait), puis nous procédons à l'itération de toutes les colonnes associées à l'index et testons leur type de données. Si une colonne a un type varchar dont la longueur dépasse 255 caractères, le script produit un message, supprime la colonne de la collection des colonnes associées à l'index, et définit fix à true (la correction a été effectuée) :

```
%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 = "La correction automatique a supprimé la colonne " &
col.Name & " de l'index."
    c.Delete
    %Fix% = True
   End If
  Next
 End If
```



Gestionnaires d'événement (Profile)

Les gestionnaires d'événement définissent des règles de validation ou d'autres scripts à exécuter lorsqu'un événement se produit sur un objet. La logique du gestionnaire d'événement est définie avec VBScript. Les critères ne prennent pas en charge les gestionnaires d'événement.

1. Pointez sur une métaclasse, un stéréotype ou un critère dans la catégorie Profile, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau > Gestionnaire d'événement** pour afficher une boîte de sélection, qui répertorie les types de gestionnaire d'événement disponibles :

Gestionnaire d'événement	Description

CanCreate	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 le gestionnaire d'événement renvoie True sur un stéréotype, vous pouvez utiliser l'outil personnalisé pour créer l'objet stéréotypé et le stéréotype est disponible dans la liste Stéréotype de la feuille de propriétés d'objet. S'il renvoie True sur une métaclasse, 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.
	Remarque : Les gestionnaires d'événement `CanCreate` sont ignorés lors d'une importation de modèle ou d'un reverse engineering, car ils pourraient modifier le modèle et y créer des divergences par rapport au modèle d'origine.

Gestionnaire d'événement	Description

Initialize	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.
	`...`
	`' Recherche d'une requesting activity existante 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`
	`' Création d'une requesting activity si aucune n'a été trouvée`
	`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, l'initialisation sera lancée 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 renvoie True sur une métaclasse, elle sera lancée 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. S'il renvoie True sur un modèle, elle sera lancée 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

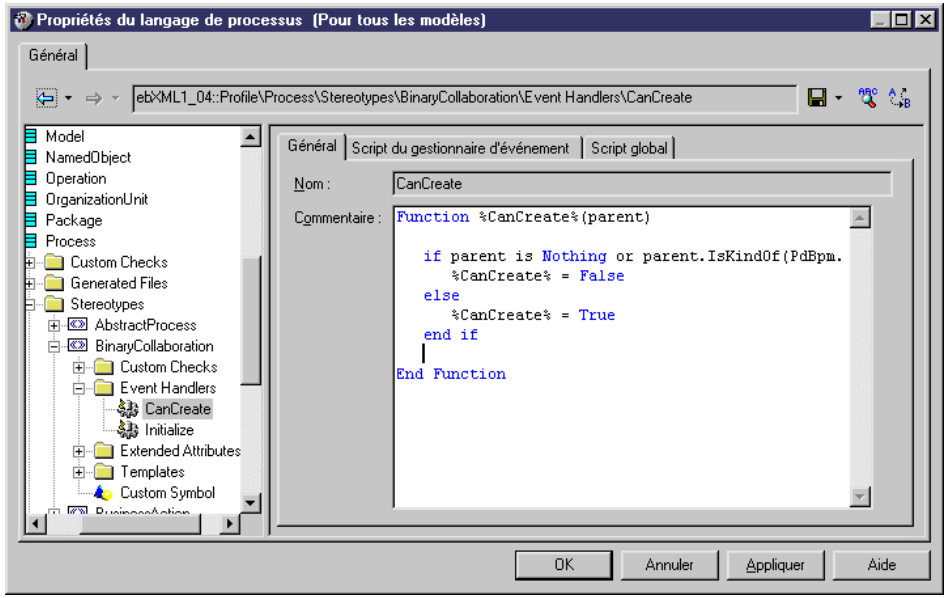
Validate	Valide les modifications apportées aux propriétés d'objet ou déclenche des mises à jour en cascade lorsque vous changez d'onglet ou que vous cliquez sur **OK** ou sur **Appliquer** dans une feuille de propriétés d'objet. Vous pouvez définir un message d'erreur qui apparaît si la condition n'est pas satisfaite en renseignant la variable de message et en définissant la variable `%Validate%` à False.
	
	Dans l'exemple suivant, le gestionnaire d'événement vérifie qu'un commentaire est ajouté dans la définition d'un objet :
	`Function %Validate%(obj, ByRef message)`
	` if obj.comment = "" then`
	`  %Validate% = False`
	`  message = "Le commentaire ne doit pas être vide"`
	` else`
	`  %Validate% = True`
	` end if`
	`End Function`

CanLinkKind	[objets de lien] Valide le type et le stéréotype des objets qui peuvent être liés comme extrémités source et destination lorsque vous créez un lien à l'aide d'un outil de la Boîte à outils ou que vous modifiez les extrémités de ligne dans une feuille de propriétés. Les paramètres `sourceStereotype` et `destinationStereotype` sont facultatifs.
	
	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, OnModelSave et OnModelClose	[modèles] Exécuté immédiatement après l'ouverture, l'enregistrement ou la fermeture d'un modèle.
OnLanguageChangeRequest, OnLanguageChanging et OnLanguageChanged	[modèles] 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 annulé. • Après après le changement de langage, mais avant que les transformations ne soient appliquées aux objets afin de les rendre conformes à la nouvelle définition de langage. • Après le changement de SGBD ou de fichier de définition de langage du modèle et la transformation des objets.

Gestionnaire d'événement	Description
OnNewFromTemplate	[modèles] Exécuté immédiatement après la création d'un modèle ou d'un projet à partir d'un template de modèle ou de projet.
BeforeDatabaseGenerate, AfterDatabaseGenerate, BeforeDatabaseReverseEngineer et AfterDatabaseReverseEngineer	[MPD] Exécuté immédiatement avant ou après la génération ou le reverse engineering d'une base de données (voir *Ajout de scripts avant ou après la génération ou le reverse engineering* à la page 145).
GetEstimatedSize	[MPD uniquement] Exécuté lorsque le mécanisme d'estimation de la taille de base de données est appelé (voir *Modification du mécanisme d'estimation de taille de base de données* à la page 219).

2. Sélectionnez un plusieurs gestionnaires d'événement, puis cliquez sur **OK** pour les ajouter.
3. Saisissez un nom et un commentaire pour identifier et document le gestionnaire d'événement.
4. Cliquez sur l'onglet **Script du gestionnaire d'événement** et saisissez un script pour définir le gestionnaire d'événement. Vous pouvez accéder à des fonctions de bibliothèques partagées et à des attributs statiques définis pour réutilisation dans le fichier de ressource depuis l'onglet **Script global** (voir *Script global (Profile)* à la page 115).



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

Exemple : Définition des valeurs par défaut pour les propriétés

Vous pouvez définir une valeur par défaut pour la plupart des propriétés d'objet via un gestionnaire d'événement `Initialize`.

1. Ajoutez la métaclasse appropriée dans votre profil (voir *Métaclasses (Profile)* à la page 35), et créez un gestionnaire d'événement de type `Initialize` sous elle.
2. Cliquez sur l'onglet **Script du gestionnaire d'événement** et modifiez le script afin de spécifier les valeur par défaut pour une ou plusieurs des propriétés du formulaire :

   ```
   obj.NomPropriété = Valeur
   ```

   Par exemple, le script suivant définit le stéréotype d'un héritage de MCD a `MonHeritage` et sa propriété **Générer les enfants** a la valeur `N'hériter que les attributs primaires` :

   ```
   Function %Initialize%(obj)
       obj.Stereotype = "MonHeritage"
       obj.InheritAll = False
       %Initialize% = True
   End Function
   ```

3. Cliquez sur **OK** pour enregistrer vos modifications et refermer l'éditeur de ressources.

   Dorénavant, lorsque vous créerez un héritage dans votre modèle, ces propriétés seront définies avec les valeurs par défaut.

Méthodes (Profile)

Les méthodes sont écrites en VBScript et effectuent des actions sur les objets lorsqu'elles sont appelées par d'autres extensions, telles que des commandes de menus ou des boutons de formulaires.

1. Pointez sur une métaclasse, un stéréotype ou un critère dans la catégorie Profile, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau** > **Méthode**.
2. Spécifiez les propriétés suivantes :

Propriété	Description
Nom	Spécifie le nom de la méthode.
Commentaire	Fournit des informations supplémentaires sur la méthode.

3. Cliquez sur l'onglet **Script de méthode**, puis saisissez le code VBscript. Le cas échéant, vous pouvez réutiliser les fonctions stockées dans l'onglet **Script global**.

   Pour plus d'informations sur l'utilisation de l'onglet **Script global**, voir *Exemple : Vérification personnalisée de MPD* à la page 82 et *Script global (Profile)* à la page 115.

L'exemple suivant, créé sous la métaclasse `Class`, convertit des classes en interfaces en copiant les propriétés de base et les opérations de la classe, en supprimant cette classe (pour éviter tout problème d'espace de noms), puis en créant la nouvelle interface.

```
Sub %Mthd%(obj)
 ' Conversion de 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 des opérations de la classe
 Dim Op
 For Each Op In obj.Operations
  ' ...
  Output Op.Name
 Next

 ' Destruction de la classe
 obj.Delete

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

Remarque : Ce 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 *Exemple : Ouverture d'une boîte de dialogue à partir d'un menu* à la page 92).

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

Menus (Profile)

Les menus spécifient des commandes qui doivent apparaître dans les menus **Fichier**, **Outils** et **Aide** standard ou dans les menus contextuels de PowerAMC.

1. Pointez sur une métaclasse, un stéréotype ou un critère dans la catégorie Profile, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau** > **Menu**.
2. Spécifiez les propriétés suivantes :

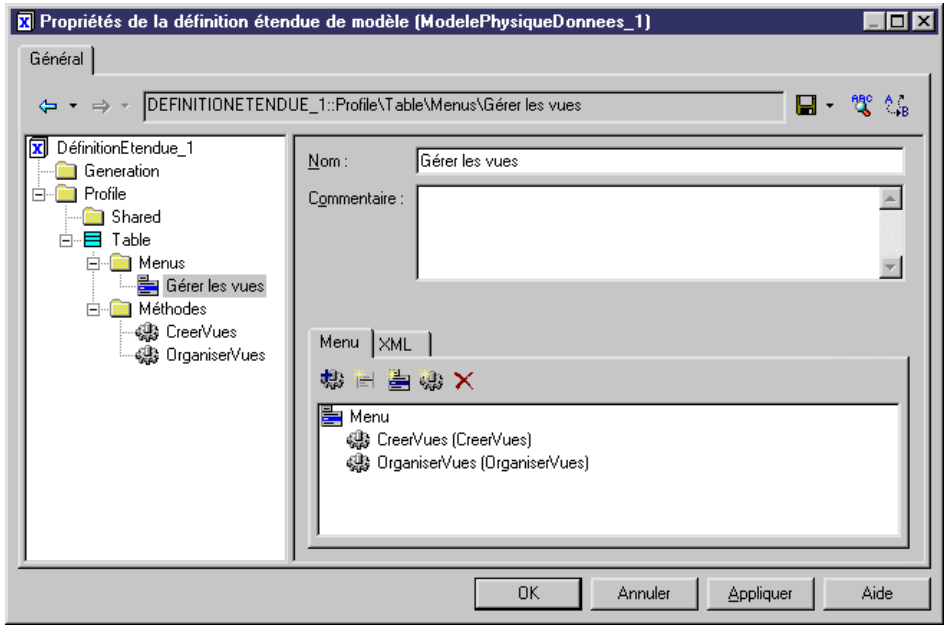
Propriété	Description
Nom	Spécifie le nom interne du menu. Ce nom ne s'affiche pas dans le menu.
Commentaire	Fournit une description du menu.

Propriété	Description
Emplacement	[modèle et diagramme uniquement] Spécifie où le menu sera affiché. Vous pouvez choisir l'une des valeurs suivantes : • Menu File > 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**.

3. Utilisez les outils du sous-onglet **Menu** pour créer des éléments dans votre menu :

Outil	Fonction
	Ajouter une commande - Affiche une boîte de dialogue de sélection qui répertorie les méthodes (voir *Méthodes (Profile)* à la page 89) et les transformations (voir *Transformations (Profile)* à la page 102) définies dans la métaclasse courante et ses métaclasses parent pour les ajouter au menu sous la forme de commandes. Sélectionnez une ou plusieurs entrées, puis cliquez sur **OK**. Les éléments sont ajoutés dans votre menu au format suivant : **EntréeMenu** （**NomMéthode/Transformation**） Vous pouvez modifier la partie **EntréeMenu** (et définir une touche de raccourci en ajoutant une perluète avant la lettre de raccourci) mais vous ne pouvez pas éditer la partie **NomMéthode/Transformation**. **Remarque :** Si vous modifiez le nom de la méthode ou de la transformation, vous devez mettre à jour les commandes en utilisant la méthode ou la transformation à la main, car le nom n'est pas automatiquement synchronisé. Vous pouvez utiliser l'outil **Remplacer parmi les éléments** pour localiser et mettre à jour ces commande.
	Ajouter un séparateur - Crée un séparateur de menu sous l'élément sélectionné.
	Ajouter un sous-menu - Crée un sous-menu sous l'élément sélectionné.
	Supprimer - Supprime l'élément sélectionné.

Vous pouvez réorganiser des éléments dans l'arborescence de menu en les faisant glisser. Pour placer un élément dans un sous-menu, faites-le glisser sur ce sous-menu.



4. [facultatif] Cliquez sur le sous-onglet **XML** pour consulter le code XML généré depuis le sous-onglet **Menu**.
5. Cliquez sur **Appliquer** pour enregistrer vos modifications.

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

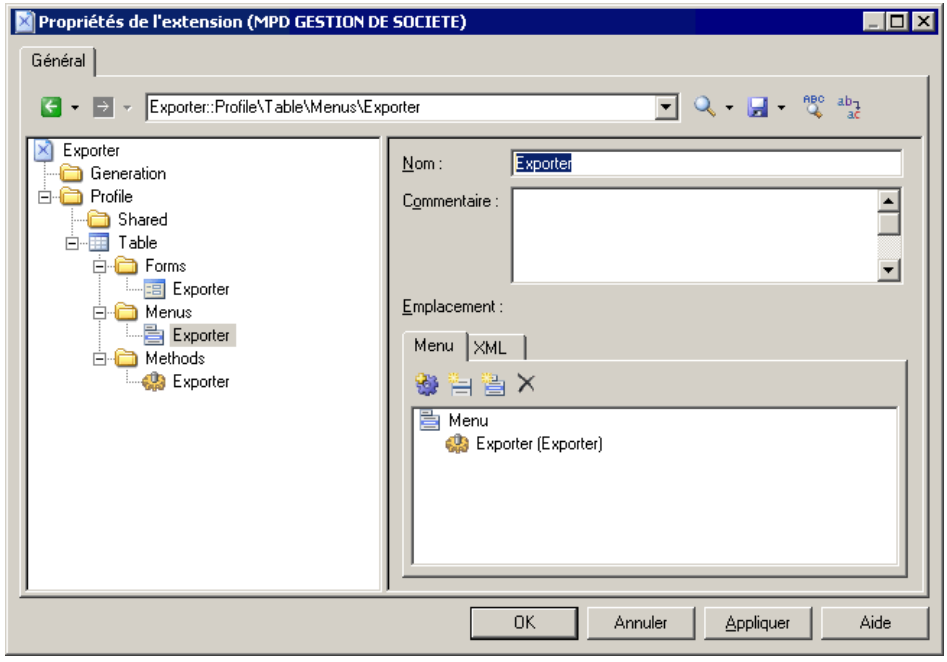
Dans cet exemple nous allons créer une commande de menu permettant d'exporter des propriétés d'objet dans un fichier XML par l'intermédiaire d'une boîte de dialogue.

1. Créez un nouveau fichier d'extension (voir *Création d'un fichier d'extension* à la page 12) dans un MPD, puis ajoutez la métaclasse `Table` (voir *Métaclasses (Profile)* à la page 35).
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 zone **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** et 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` :



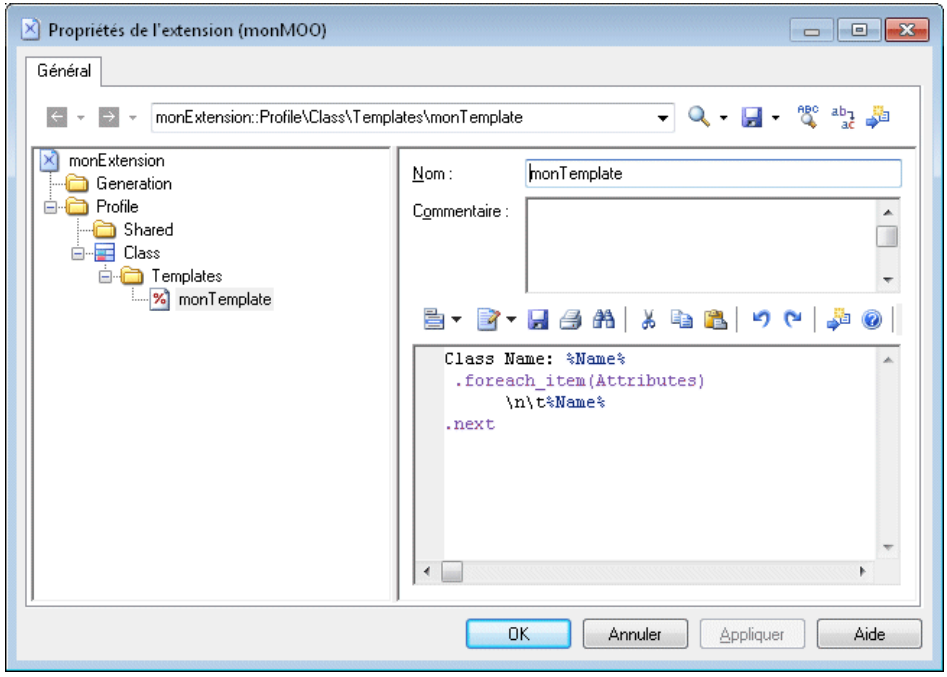
6. Cliquez sur **OK** pour enregistrer vos modifications et revenir au modèle. Lorsque vous pointez sur une table dans le diagramme ou dans l'Explorateur d'objets et cliquez le bouton droit de la souris, la commande **Exporter** est disponible dans le menu contextuel.

Templates (Profile)

Les templates du GTL extraient du texte des valeurs de propriétés PowerAMC afin de l'utiliser dans des fichiers générés ou d'autres contextes.

1. Pointez sur une métaclasse, un stéréotype ou un critère dans la catégorie Profile (ou dans la catégorie `Profile/Shared`, si le template s'applique à toutes les métaclasses), cliquez le bouton droit de la souris, puis sélectionnez **Nouveau** > **Template**.
2. Saisissez un nom pour le template. N'utilisez pas d'espace dans le nom et, par convention, les templates sont nommés en utilisant la casse headless camelCase (par exemple `monTemplate`).
3. [facultatif] Saisissez un commentaire dans la zone Commentaire afin d'expliquer le rôle du template.
4. Saisissez le code de GTL (voir *Chapitre 5, Personnalisation de la génération à l'aide du langage de génération par template* à la page 265) dans la zone de texte.

   Dans cet exemple, `monTemplate` est défini sur la métaclasse `Class`, et va générer le nom de la classe, suivi par une liste de ses attributs :



Fichiers générés (Profile)

Les fichiers générés assemblent des templates de GTL sous la forme de fichiers ou bien pour l'aperçu dans l'onglet **Aperçu** des feuilles de propriétés d'objet.

1. Pointez sur une métaclasse, un stéréotype ou un critère dans la catégorie Profile, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau > Fichier généré**.

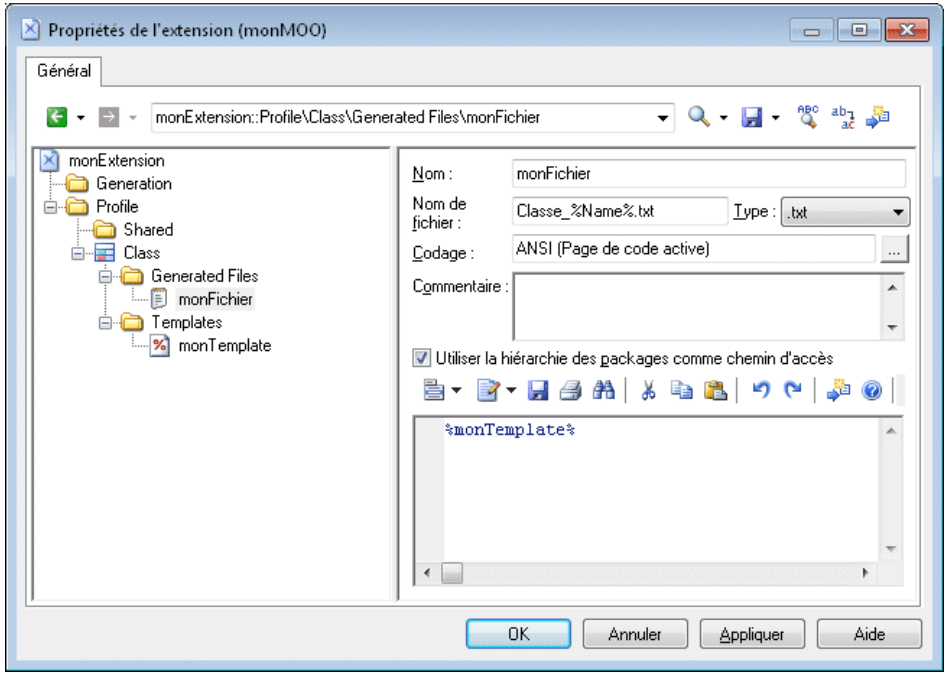
   Seuls les objets, tels que les tables ou les classes, prennent en charge la génération de fichier, mais vous pouvez toujours créer des fichiers générés pour les sous-objets, tels que les colonnes et les attributs, afin de prévisualiser le code généré pour eux sur l'onglet **Aperçu** de leur feuille de propriétés.
2. Spécifiez les propriétés suivantes :

Propriété	Description
Nom	Spécifie un nom pour l'entrée du fichier généré dans l'Editeur de ressources. Si une extension attachée au modèle contient un nom de fichier généré identique à celui définir dans le fichier de ressource principal, seul le fichier généré de l'extension sera généré.
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.

Propriété	Description
Utiliser la hiérarchie des packages comme chemin d'accès	Spécifie que la hiérarchie de packages doit être utilisée pour générer la hiérarchie des répertoires de fichiers.

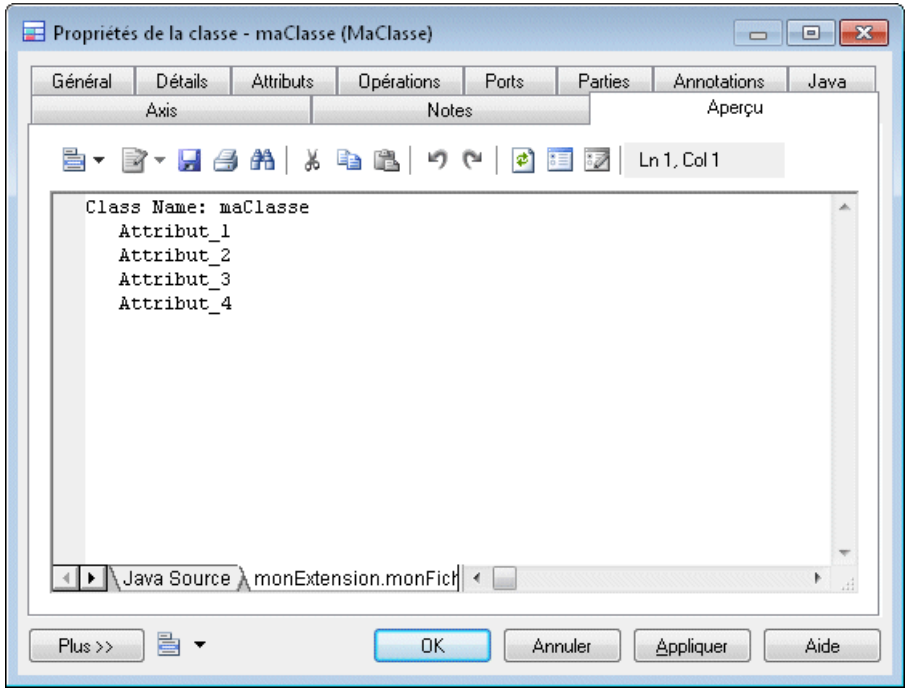
3. Saisissez le code de GTL (voir *Chapitre 5, Personnalisation de la génération à l'aide du langage de génération par template* à la page 265) ou le nom d'un template afin de remplir le fichier dans la zone de texte.

   Dans l'exemple suivant, un fichier généré est défini pour les classes de MOO. Un fichier sera généré pour chaque classe dans le modèle avec un nom dérivé du nom (`%Name%`) de la classe et qui contient le contenu généré depuis le template `%monTemplate%` (voir *Templates (Profile)* à la page 94) :



4. Cliquez sur **OK** pour enregistrer vos modifications et fermer l'Editeur de ressources.

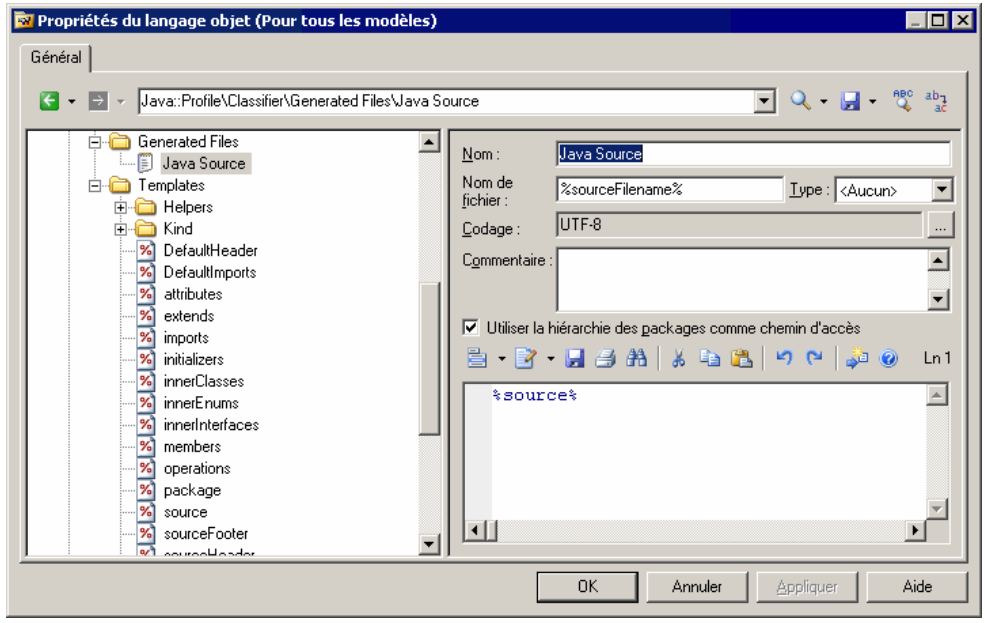
   Le fichier est immédiatement disponible en tant que sous-onglet sur l'onglet **Aperçu** de la feuille de propriétés de l'objet :



Exemple : Templates et fichiers générés Java

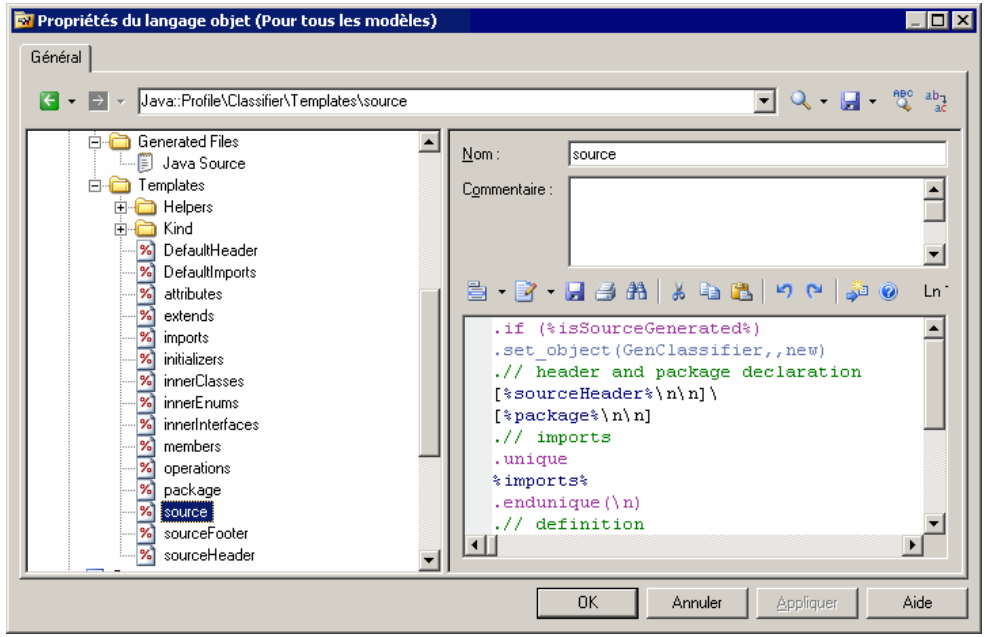
Les templates contiennent du code de GTL utilisé pour générer des fragments de texte depuis les valeurs de propriétés de PowerAMC, tandis que les fichiers générés sont utilisés pour assembler des templates pour la génération ou pour la prévisualisation dans l'onglet **Aperçu** d'une feuille de propriétés d'objet.

Dans l'exemple suivant, un fichier généré appelé `Java Source` est défini pour les classificateurs. Un fichier sera généré pour chaque classificateur dans le modèle avec un nom dérivé du template `%sourceFilename%` spécifié dans la zone **Nom de fichier**, et contenant le contenu généré depuis le template `%source%` :



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, y compris les références aux autres templates appelés `%isSourceGenerated%`, `%sourceHeader%`, `%package%` et `%imports%`:



Génération de vos fichiers dans le cadre d'une génération standard ou étendue

Vous pouvez utiliser des fichiers générés afin d'étendre la génération standard pour les objets de MPM, MPM et MSX ou pour créer une génération étendue distincte pour chaque type de modèle. Dans le cas des générations étendues, vous pouvez définir une commande de menu personnalisée.

Pour étendre les générations standard de MPM, MOO ou MSX depuis l'Editeur de ressources :

1. Cochez la case **Compléter la génération de langage** à la racine du fichier d'extension (voir *Propriétés d'un fichier d'extension* à la page 15) pour faire en sorte que le fichier d'extension puisse être sélectionnable dans la boîte de dialogue **Génération**, sur l'onglet **Cible**.
2. Définissez le fichier généré.
3. [facultatif] Définissez les options sous `Generation\Options` (voir *Exemple : Ajout d'une option de génération* à la page 123) pour les faire apparaître dans la boîte de dialogue **Génération**, sur l'onglet **Options**.
4. [facultatif] Définissez des commandes sous `Generation\Commands` et référencez ces commandes dans des tâches (voir *Exemple : Ajout d'une commande et d'une tâche de génération* à la page 125) pour les faire apparaître dans la boîte de dialogue **Génération**, sur l'onglet **Tâches**.

Vous pouvez également définir des génération de fichier distinctes hors de la génération de langage standard pour un MPD ou tout autre type de modèle et les rendre disponibles via la commande **Outils > Génération étendue**

1. [MOO, MPM et MSX uniquement] Décochez la case **Compléter la génération de langage** à la racine du fichier d'extension (voir *Propriétés d'un fichier d'extension* à la page 15).
2. Ajoutez les métaclasses appropriées dans la catégorie Profile, puis sélectionnez l'option **Activer la sélection pour la génération de fichiers** (voir *Métaclasses (Profile)* à la page 35) pour les métaclasses à partir desquelles vous souhaitez générer des fichiers.
3. Définissez les fichiers générés appropriés sous ces métaclasses.

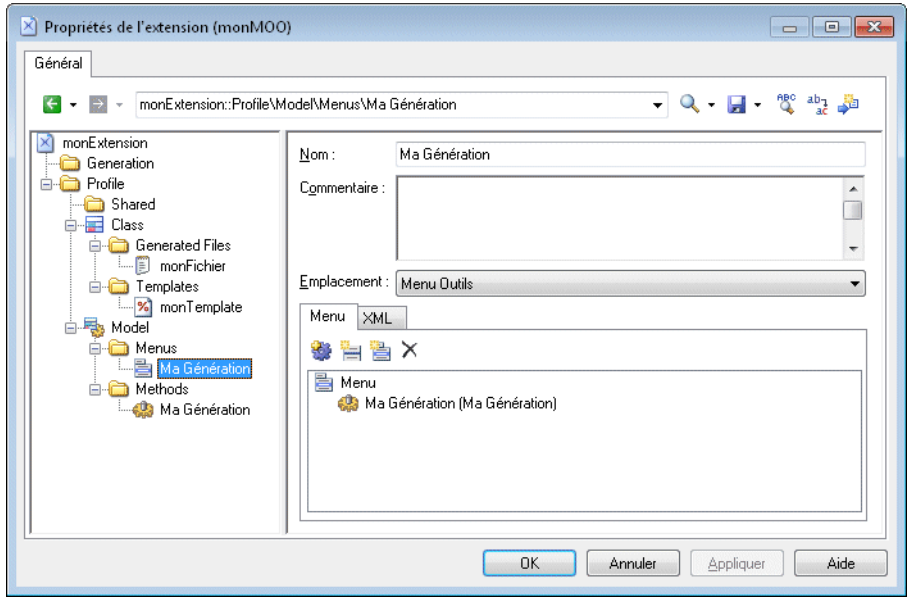
   La génération est immédiatement disponible dans l'onglet **Cibles** de la boîte de dialogue **Génération** lorsque vous sélectionnez **Outils > Génération étendue**.
4. [facultatif] Créez une commande dans le menu **Outils** pour accéder directement à votre génération étendue dans sa propre boîte de dialogue :
   a. Créez une méthode sous `Profile\Model` avec le nom que vous souhaitez donner à votre commande, puis saisissez le code suivant (où `extension` représente le code du fichier d'extension) :

```
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, "extension"

End Sub
```

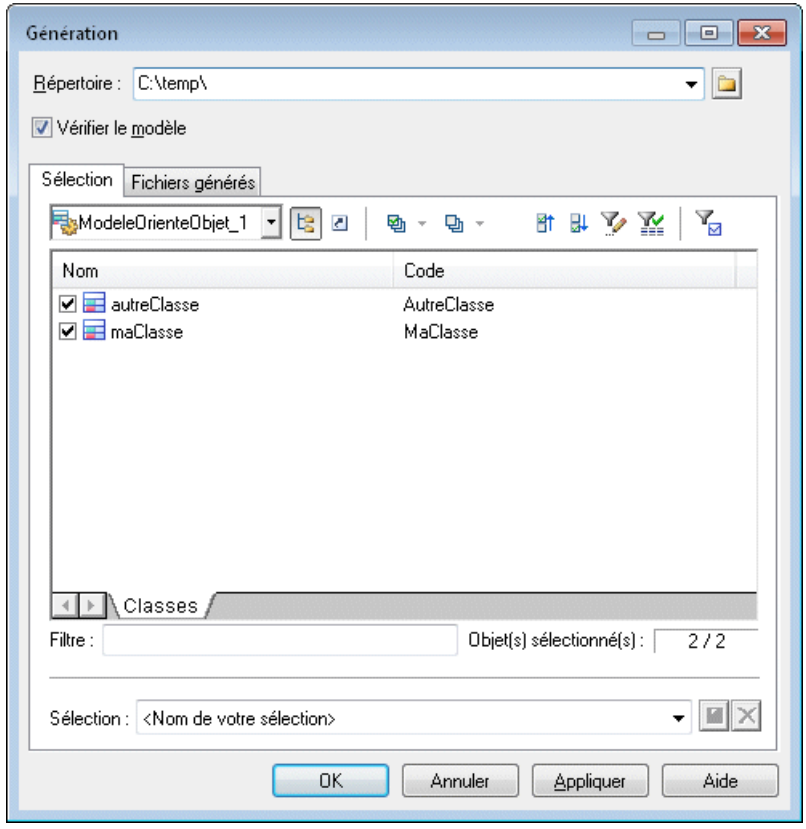
   Pour plus d'informations sur les méthodes, voir *Méthodes (Profile)* à la page 89.

   b. Créez une commande de menu sous `Profile\Model` et sélectionnez `Menu Outils` dans la liste **Emplacement** (voir *Menus (Profile)* à la page 90).

   c. Ajoutez la méthode dans le menu en utilisant l'outil **Ajouter une commande** :



d. Sélectionnez la commande spécifiée (par exemple, **Outils > Ma Génération**) afin d'ouvrir une boîte de dialogue **Génération**, qui n'a pas d'onglet **Cibles** :



Transformations (Profile)

Les transformations définissent des jeux d'actions pour modifier les objets avant ou après la génération, ou à la demande. Les transformations sont le plus souvent regroupées au sein de profils de transformation.

Les transformations peuvent être utilisées pour :

- Mettre en oeuvre une architecture *MDA (Model Driven Architecture)*, qui utilise la modélisation UML afin de décrire une application à différents niveaux de détails. PowerAMC permet de créer un *PIM (platform-independent model)* (modèle ne dépendant pas d'une plateforme, et qui permet de modéliser la fonctionnalité et la logique métiers de base) puis de l'affiner progressivement dans différents modèles contenant des niveaux croissants d'informations de mise en oeuvre et dépendantes de la technologie jusqu'à un *modèle spécifique à une plateforme* (PSM, platform-specific model). Vous pouvez définir des transformations qui vont générer une version affinée du modèle, basées sur la

plateforme cible souhaitée, et les changements apportés au PIM peuvent être répercutés en cascade aux modèles générés.
- Appliquer des motifs de modélisation à vos objets de modèle.
- 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 dans le cas d'une ingénierie par va-et-vient. Par exemple, si vous générez un MPD à partir d'un MOO afin de créer une correspondance O/R et que le MOO source contient des composants, vous pouvez pré-transformer, des composants en classes afin d'établir plus facilement des correspondances avec des tables de MPD. Lorsque vous mettez à jour le MOO source depuis le MPD généré, vous pouvez utiliser une post-transformation pour recréer les composants à partir des classes.

Les transformations peuvent être appelées à la demande (sélectionnez **Outil > Appliquer des transformations**), avant ou après la génération de modèle (voir *Guide des fonctionnalités générales > Liaison et synchronisation de modèles > Génération de modèles et d'objets de modèle*), ou via une commande personnalisée (voir *Menus (Profile)* à la page 90).

1. Pointez sur une métaclasse, un stéréotype ou un critère dans la catégorie Profile, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau > Transformation**.
2. Saisissez un **Nom** approprié, et le cas échéant un **Commentaire** pour préciser son rôle.
3. Sur l'onglet **Script de transformation**, saisissez du code VBscript pour effectuer la transformation.

   Dans cet exemple, qui est créé dans une extension attachée à un MCD sous la métaclasse `DataItem`, le script teste pour savoir si l'information a une liste de valeurs définie et, si tel est le cas (et s'il n'existe pas déjà un domaine avec la même liste de valeurs dans le MCD), crée un nouveau domaine avec la liste de valeurs :

```
Sub %Transformation%(obj, trfm)

   Dim list
   list = obj.ListOfValues
   if not list = "" then
      output "transforming " & cstr(obj)

      ' Vérifie si un tel domaine existe déjà
      Dim domn, found
      found = false
      for each domn in obj.Model.Domains
         if domn.ListOfValues = list then
            found = true
         end if
      next

      ' Crée un nouveau domaine
      if not found then
         set domn = obj.Model.Domains.CreateNew()
         domn.SetNameAndCode obj.Name, obj.Code
         domn.ListOfValues = list
```

```
      end if
   end if

End Sub
```

Si cette transformation peut être ajoutée dans un profil de transformation en tant que :

- Transformation pré-génération - La transformation est appelée dans la boîte de dialogue Options de génération. Les domaines sont créés temporairement dans le MCD avant la génération, puis générés dans le modèle cible (par exemple, dans un MPM).
- Transformation post-génération - La transformation peut être appelée dans la boîte de dialogue Options de génération (pour une génération MCD-MCD). Les domaines sont créés dans le MCD cible après génération. La transformation peut également être appelée à tout moment en sélectionnant **Outils > Appliquer les transformations** pour créer les domaines dans le modèle existant.

4. [facultatif] Consultez l'onglet **Script global** (voir *Script global (Profile)* à la page 115), qui permet d'accéder aux définitions partagées par toutes les fonctions VBscript définies dans le profil, et l'onglet **Dépendances**, qui répertorie les profils de transformation dans lesquels la transformation est utilisée.

Profils de transformation (Profile)

Un profil de transformation rassemble des transformations, et les rend disponibles lors de la génération de modèle, ou lorsque vous sélectionnez **Outils > Appliquer des transformations**.

1. [si la catégorie Transformation Profiles est absente] Pointez sur le noeud racine, cliquez le bouton droit de la souris, sélectionnez **Ajouter des éléments**, sélectionnez `Transformation Profiles`, puis cliquez sur **OK** pour créer ce dossier.
2. Pointez sur le dossier `Transformation Profiles`, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau** pour créer un profil de transformation.
3. Saisissez les propriétés appropriées :

Propriété	Description
Nom / Commentaire	Spécifient le nom du profil de transformation et fournit une explication sur son rôle.
Type de modèle / Famille / Sous-famille	[facultatif] Spécifient le type de modèle avec lequel le profil de transformation peut être utilisé lors de la génération et (si le type prend en charge un fichier de définition de langage) sa famille et sa sous-famille. Si une ou plusieurs de ces zones est renseignée, le profil ne sera affiché que si le modèle à générer est conforme à cette ou ces propriétés. Par exemple, si vous définissez la transformation dans un MPD ou une extension de MPD et spécifiez `Modèle Orienté Objet` et `Java`, le profil ne sera disponible que si vous sélectionnez de générer le MPD dans un MOO Java.

4. Cliquez sur l'onglet **Pré-génération**, puis cliquez sur l'outil **Ajouter des transformations** pour ajouter des transformations à effectuer avant la génération.

   Ces transformations sont exécutées avant la génération sur les objets contenus dans votre modèle source. Si des objets sont créés par ces transformations, ils sont automatiquement ajoutés dans la listes de objets à générer. Toute modification d'objets existant ou tout nouvel objet créés par ces transformations sont annulés après génération, de sorte que votre modèle revient à son état antérieur.
5. Cliquez sur l'onglet **Post-génération**, puis cliquez sur l'outil **Ajouter des transformations** pour ajouter des transformations à effectuer après la génération. Les transformations ajoutées sur cet onglet sont également rendues disponibles pour application hors du contexte d'une génération en sélectionnant **Outils** > **Appliquer les transformations**.

   Ces transformations sont exécutées sur les objets générés dans votre modèle cible.
6. Cliquez sur **Appliquer** pour enregistrer vos modifications.

Développement de scripts de transformation

Les scripts de transformation sont rédigés en VBScript en utilisant des méthodes spéciales. Les scripts de transformation ne requièrent pas autant de vérifications que les scripts standard, car les transformations sont toujours mises en oeuvre dans un modèle temporaire vide, qui est ensuite fusionné avec le modèle cible de génération.

Un objet source pouvait être transformé et avoir plusieurs cibles, il peut s'avérer difficile d'identifier l'origine d'un objet, particulièrement dans la boîte de dialogue de fusion. Le mécanisme suivant est utilisé pour vous aider à identifier l'origine d'un objet :

- Si l'objet source est transformé en un seul objet, la transformation est utilisée comme un identifiant 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 ne devez utiliser que des caractères alphanumériques, et nous vous recommandons d'utiliser une valeur "stable" telle qu'un stéréotype, qui ne sera pas modifié lors des générations successives.

Les méthodes suivants sont disponibles lorsque vous écrivez un script de transformation :

- `CopyObject(`**source** `[,`**balise**`])`

  Duplique un objet existant, définit une source pour l'objet dupliqué, et renvoie une copie du nouvel objet.
- `SetSource(`**source, cible** `[,`**balise**`])`

  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.
- `GetSource(`**cible** `[,`**balise**`])`

  Identifie l'objet source d'un objet généré.
- `GetTarget(`**source** `[,`**balise**`])`

Identifie l'objet cible d'un objet source.

Les objets transformation interne sont préservés lorsque les transformations sont utilisées par la fonctionnalité **Appliquer les transformations** ou via une commande de menu personnalisée, elles peuvent donc être exécutées à nouveau si vous mettez ensuite à jour (ou régénérez) le modèle. Par exemple, vous générez l'entité de MCD `A` dans une classe de MOO `B` puis appliquez une transformation à la classe `B` afin de créer la classe `C`. Si vous effectuez des modifications sur l'entité `A` et répétez la génération pour mettre à jour le MOO, la classe `B` est mise à jour et la transformation est automatiquement réappliquée pour mettre à jour la classe `C`.

Importations XML (Profile)

Les importations XML permettent de définir des correspondances entre un schéma XML et le métamodèle PowerAMC (et d'éventuelles extensions) pour permettre l'importation des fichiers XML conformes au schéma. Vous pouvez spécifier des scripts d'initialisation et de post-traitement afin de gérer la complexité dans l'importation.

Pour une présentation de la création, du déploiement et de l'utilisation des importations XML, voir *Guide des fonctionnalités générales > Modélisation avec PowerAMC > Objets > Importation d'objets à partir de fichiers XML*.

1. [Si la catégorie `XML Imports` n'est pas présente] Pointez sur le noeud racine, cliquez le bouton droit de la souris, puis sélectionnez **Ajouter des éléments**, sélectionnez `XML Imports`, puis cliquez sur **OK** pour créer ce dossier.
2. Pointez sur le dossier `XML Imports`, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau** pour créer une importation XML.
3. Saisissez les propriétés appropriées :

Propriété	Description
Nom	Spécifie le nom de l'importation, qui sera utilisé comme nom pour la commande d'importation sous **Fichier > Importer**.
Premier diagramme	Spécifie le premier diagramme qui doit être initialisé dans le modèle créé à partir du fichier importé.
Créer des symboles par défaut	Spécifie la création de symboles dans le diagramme pour les objets importés.
Suffixe de nom de fichier	Spécifie le suffixe de nom de fichier qui identifie les documents XML qui sont conformes au schéma.
Commentaire	Fournit une explication de l'importation ou d'autres informations supplémentaires.

4. Cliquez sur l'onglet **Schéma**, puis cliquez sur l'outil **Importer** afin de copier le schéma, avec les importations et les inclusions résolues, dans le fichier d'extension à des fins de mise en correspondance.

   Avertissement ! Si le schéma sélectionné est trop permissif et permet trop de hiérarchies d'objets, il se peut qu'il ne puisse pas être affiché entièrement dans l'Editeur de correspondances. Si vous disposez d'un fichier XML exemple à importer, vous pouvez l'importer à la place du schéma en cliquant sur **Importer depuis un exemple** et PowerAMC va déduire un schéma partiel depuis ce fichier. Notez que bien qu'un schéma obtenu ainsi puisse importer un fichier de données exemple, d'autres documents basés sur le même schéma peuvent ne pas être complets s'ils contiennent d'autres types d'objets (ou attributs ou collections) qui, bien que valides pour le schéma, ne se trouvaient pas dans le premier document.

   Vous pouvez cliquer sur l'outil **Afficher comme un modèle** pour ouvrir le schéma sous la forme d'un modèle XML schema.
5. [facultatif] Cliquez sur l'onglet **Extensions** et sélectionnez les fichiers d'extension contenant des extensions du métamodèle PowerAMC standard afin de fournir des métaclasses (voir *Objets, sous-objets et liens étendus (Profile)* à la page 39), attributs (voir *Attributs étendus (Profile)* à la page 45), et collections (voir *Collections et compositions étendues (Profile)* à la page 54) supplémentaires avec lesquels mettre en correspondance votre schéma XML.
   En attachant de cette manière des fichiers d'extension, vous pouvez réutiliser des extensions préalablement définies dans vos importations ou partager des extensions entre des importations. Vous avez également la possibilité de définir des extensions sous la catégorie Profile dans le fichier de ressource contenant la définition d'importation XML, ou de les créer de façon dynamique lors vous créez vos correspondances d'importation.
6. [facultatif] Cliquez sur l'onglet **Initialisation** et saisissez du code VBScript à exécuter lors de la création du modèle, avant d'importer le moindre objet. Vous pouvez accéder aux fonctions de bibliothèque partagées et aux attributs statiques définis pour réutilisation dans le fichier de ressource depuis l'onglet **Script global** (voir *Script global (Profile)* à la page 115).
7. [facultatif] Cliquez sur l'onglet **Post-traitement** et saisissez du code VBScript à exécuter une fois que tous les objets ont été importés.
8. Cliquez sur l'onglet **Général**, puis cliquez sur le bouton **Correspondances** pour définir des correspondances depuis les métaclasses identifiées dans votre schéma XML vers celles du métamodèle PowerAMC dans l'Editeur de correspondances (voir *Correspondances d'importation XML* à la page 107).
9. Cliquez sur **Appliquer** pour enregistrer vos modifications.

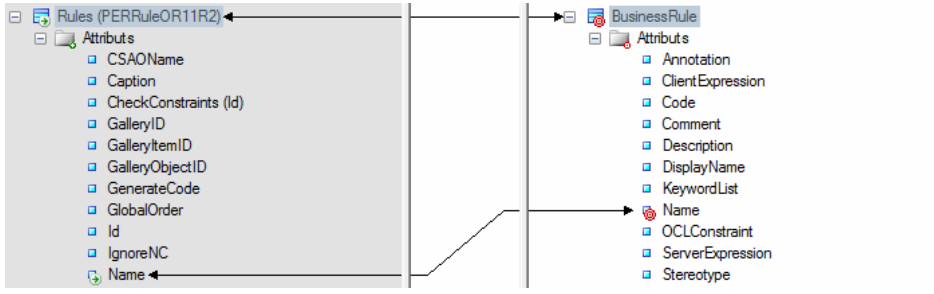
Correspondances d'importation XML

Vous contrôlez la façon dont les éléments définis dans un schéma XML sont importés en les mettant en correspondance de même que leurs attributs, compositions, et agrégations, avec des objets contenus dans le métamodèle PowerAMC. Le schéma XML est analysé et présenté

sous la forme d'une liste de métaclasses dans la partie gauche de l'Editeur de correspondances et le métamodèle PowerAMC (et ses éventuelles extensions) sont affichés dans la partie droite.

Remarque : Il n'est pas nécessaire de mettre en correspondance toutes les métaclasses (ou tout leur contenu), mais uniquement celles que vous souhaitez utiliser. Si le métamodèle PowerAMC ne contient pas les métaclasses, attributs, compositions ou agrégations appropriés pour établir les correspondances, vous pouvez les créer de façon dynamique ou enregistrer les correspondances existantes, fermer l'Editeur de correspondances, définir ou attacher les extensions appropriées, puis réouvrir l'Editeur de correspondances afin d'établir les correspondances manquantes.

1. Faites glisser une métaclasse externe sur une métaclasse PowerAMC pour créer une correspondance d'importation. Les éventuels attributs et collections externes sont automatiquement mis en correspondance avec les attributs PowerAMC qui ont le même nom :

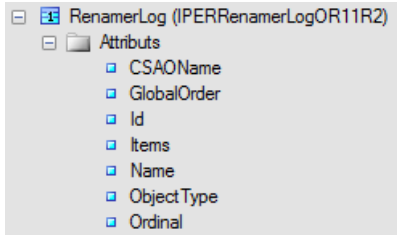


   Par défaut l'Editeur de correspondances répertorie les attributs et collections standard des métaclasses, qui sont normalement affichées dans les feuilles de propriétés d'objet, cliquez sur l'outil **Filtrer les propriétés**, puis sélectionnez `Afficher toutes les propriétés`. Vous pouvez également filtrer l'arborescence en utilisant les outils **Filtrer les correspondances** et **Filtrer les objets**.

   Remarque : Si aucune métaclasse appropriée n'existe, pour créer et mettre en correspondance une nouvelle métaclasse étendue basée sur la métaclasse `ExtendedObject`, faites glisser la métaclasse externe sur le noeud racine du métamodèle PowerAMC.

2. Faites glisser des attributs supplémentaires sous la métaclasse sur des attributs PowerAMC ayant des types de données compatibles afin de les mettre en correspondance. Les attributs sont contenus dans un dossier sous la métaclasse et représentent des propriétés individuelles telles que Name, Size, DimensionalType, contenant des valeurs de type booléen, texte , numérique ou ID d'objet :

PowerAMC identifie les métaclasses de sous-objet dans le schéma qui sont limitées à une seule instance et affichent en superposition un `1` sur leur icône. Les attributs situés sous de telles métaclasses sont traités comme appartenant à la métaclasse parent et peuvent être mis en correspondance avec les attributs situés sous l'objet PowerAMC avec lequel le parent est mis en correspondance :



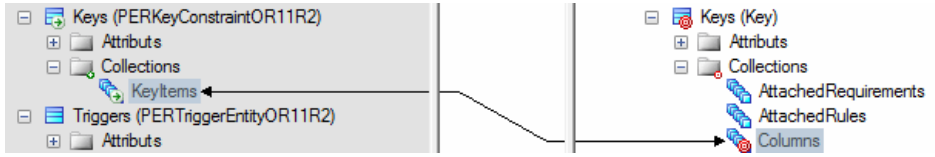
Remarque : Si aucun attribut approprié n'existe, pour créer et mettre en correspondance un nouvel attribut étendu, faites glisser l'attribut externe sur la métaclasse PowerAMC avec laquelle son parent est mis en correspondance.

3. Faites glisser des métaclasses de sous-objet (compositions) sous la métaclasse sur des compositions PowerAMC afin de créer des correspondances entre elles :

Les éventuels attributs situés sous la métaclasse de sous-objet sont automatiquement mis en correspondance avec les attributs PowerAMC qui portent le même nom. Mettez en correspondance les autres attributs de sous-objet appropriés.

Remarque : Dans certaines circonstances, il peut s'avérer approprié de mettre en correspondance une métaclasses de sous-objet externe et une métaclasse d'objet PowerAMC, et de telles correspondances sont permises.

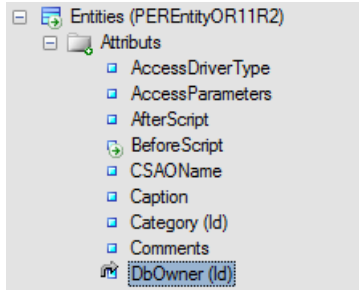
4. Faites glisser des collections externes (agrégations) sous la métaclasse sur des collections PowerAMC pour créer des correspondances entre elles :



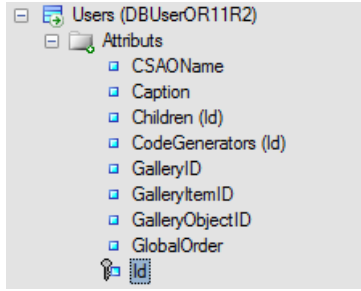
5. Dans certains schémas, il peut s'avérer nécessaire d'identifier des attributs comme des références et des identifiants pour lier une métaclasse à l'autre au moyen d'une agrégation :

   a) Pointez sur un attribut, cliquez le bouton droit de la souris, puis sélectionnez **Déclarer comme référence d'objet** afin de spécifier qu'il agit comme un pointeur vers un autre objet. De tels attributs ont souvent le type GUID, Token, ou NCName (PowerAMC

identifie automatiquement les attributs de type IDRef comme des références). Une flèche en angle arrondi est ajoutée à l'icône d'attribut :



b) Ouvrez la métaclasse vers laquelle la référence d'objet pointe, sélectionnez son attribut identifiant, cliquez le bouton droit de la souris et sélectionnez **Déclarer comme identifiant unique**. Un symbole de clé apparaît sur l'icône d'attribut :



c) L'attribut de référence d'objet peut maintenant être mis en correspondance avec un attribut PowerAMC de type objet (qui est également surmonté d'une flèche en angle arrondi) :

6. [facultatif] Sélectionnez une métaclasse et saisissez un script d'initialisation ou de post-traitement afin de modifier les objets lors de ou juste après la création (voir *Propriétés d'une correspondance de métamodèle* à la page 111).
7. [facultatif] Cliquez sur le modèle cible (noeud racine) afin d'afficher la liste de toutes les correspondances dans le volet **Correspondances** en bas de la boîte de dialogue, et utilisez les flèches en bas de la liste pour changer l'ordre dans lequel les objets sont générés afin de vous assurer que les dépendances sont respectées.

Remarque : Pour contrôler l'ordre dans lequel les attributs compositions, et agrégations sont importés dans les objets, sélectionnez la métaclasse cible pour afficher ses correspondances dans le volet **Correspondances**, puis utilisez les flèches en bas des listes sur les sous-onglets **Correspondance des attributs**, **Correspondances des collections** et **Correspondances des sous-objets**.

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

Propriétés d'une correspondance de métamodèle

Les propriétés d'une correspondance de métamodèle sont des correspondances entre objets de métamodèle, qui contrôlent la façon dont les objets sont importés ou générés. Les correspondances de métamodèle sont des sous-objets de l'objet de métamodèle PowerAMC sous lequel elles sont définies.

Pour afficher la feuille de propriétés d'une correspondance de métamodèle, sélectionnez la correspondance dans la liste située en haut du volet **Correspondances** de l'Editeur de correspondances ou de l'onglet **Correspondances** et la feuille de propriétés de l'objet parent, puis cliquez sur l'outil **Propriétés**.

Les onglets disponibles sur une feuille de propriétés particulière dépendent des objets mis en correspondance. L'onglet **Général** contient les propriétés suivantes :

Propriété	Description
Objet source	Spécifie l'objet de métamodèle mis en correspondance avec l'objet cible.
Objet cible	Spécifie l'objet de métamodèle mis en correspondance avec l'objet source. Cet objet est le parent de la correspondance elle-même.

Script de transformation	[correspondances de méta-attributs] Spécifie un script pour définir la valeur de l'attribut. Dans l'exemple suivant, provenant d'une importation XML, l'attribut `not-nullable` est importé dans l'attribut `Mandatory` et, vu que le sens de l'attribut est inversé, la valeur booléenne importée est définie comme l'opposé de la valeur source :
	`Sub %Set%(obj, sourceValue)`
	`  obj.SetAttribute "Mandatory", not sourceValue`
	`End Sub`
	Dans l'exemple suivant, provenant d'une génération d'objet, l'attribut `NumberID` est généré dans l'attribut `Comment` et une chaîne de texte est ajoutée en début de commentaire afin de clarifier l'origine de la valeur :
	`Function %AdjustValue%(sourceValue, sourceObject, targetObject)`
	`  Dim targetValue`
	`  targetValue = "The original process NumberID is " +cstr(sourceValue)`
	`  %AdjustValue% = targetValue`
	`End Function`

Les onglets suivants sont également disponibles pour les correspondances de métaclasses :

- **Initialisation** - Spécifie un script permettant d'initialiser la métaclasse à créer. Dans l'exemple suivant, la valeur de l'attribut Stereotype est définie à SimpleType :

```
Sub %Initialize%(obj)
   obj.Stereotype = "SimpleType"
End Sub
```

- **Correspondances d'attribut** - Répertorie les correspondances des attributs sous la métaclasse. Sélectionnez une correspondance, puis cliquez sur l'outil **Propriétés** pour afficher sa feuille de propriétés. Pour contrôler l'ordre dans lequel les attributs sont créés, afin de respecter les dépendances entre eux, utilisez les flèches situées en bas de la liste.
- **Correspondances de collection** - Répertorie les correspondances des sous la métaclasse.
- **Post-traitement** - Spécifie un script permettant de modifier la métaclasse après la création et l'exécution des correspondances. Dans l'exemple suivant, la valeur de l'attribut **Code** est copiée dans l'attribut **Name** :

```
Sub %PostProcess%(obj)
   ' Copie du code dans le nom
   obj.Name = obj.Code
End Sub
```

Propriétés des objets du métamodèle

Pour afficher les propriétés des métaclasses, méta-attributs et métacollections représentées dans l'Editeur de correspondances, double-cliquez sur le noeud d'objet dans l'Editeur de correspondances ou pointez sur ce noeud, cliquez le bouton droit de la souris, puis sélectionnez **Propriétés**.

L'onglet **Général** contient les propriétés suivantes :

Propriété	Description
Parent	[méta-attributs et métacollections] Spécifie la métaclasse à laquelle le méta-objet appartient.
Collection parent	[sous-objets/compositions] Spécifie le nom de la collection de composition qui contient les sous-objets sous l'objet parent.
Nom	Spécifie le nom de la métaclasse dans le métamodèle PowerAMC ou schéma XML.
Type de données	[méta-attributs] Spécifie le type de données de l'attribut.
Identifiant	[méta-attributs] Spécifie que l'attribut est utilisé pour identifier la métaclasse pour référencement par une autre métaclasse.
Référence / Chemin de référence	[méta-attributs et métacollections] Spécifie que l'attribut ou la collection est utilisé pour pointer vers une autre métaclasse afin de former une agrégation.
Singleton	[métaclasses] Spécifie que seule une instance de la métaclasse est possible sous chaque objet parent.
Commentaire	Fournit des informations supplémentaire sur le méta-objet.

Les onglets suivants sont également disponibles pour les métaclasses :

- **Attributs** - Répertorie les méta-attributs qui appartiennent à la métaclasse. Sélectionnez un attribut dans la liste, puis cliquez sur l'outil **Propriétés** pour afficher sa feuille de propriétés.
- **Collections** - Répertorie les métacollections qui appartiennent à la métaclasse. Sélectionnez une collection dans la liste, puis cliquez sur l'outil **Propriétés** pour afficher sa feuille de propriétés.

Générations d'objet (Profile)

Les générations d'objet permettent de définir des correspondances entre un type de modèle PowerAMC et un autre basé sur les deux métamodèles (et les éventuelles extensions) pour permettre la génération d'un des deux types d'objets.

Pour une présentation de la création, du déploiement et de l'utilisation des générations d'objets, voir *Guide des fonctionnalités générales > Liaison et synchronisation de modèles > Génération de modèles et d'objets de modèle > Génération d'objets de modèle > Définition de générations d'objet avancées*.

1. [si la catégorie `Object Generations` n'est pas présente] Pointez sur le noeud racine, cliquez le bouton droit de la souris, puis sélectionnez **Ajouter des éléments**, sélectionnez `Object Generations`, puis cliquez sur **OK** pour créer ce dossier.
2. Pointez sur le dossier `Object Generations`, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau** pour créer une génération d'objet.
3. Saisissez les propriétés appropriées :

Propriété	Description
Type de modèle cible	Spécifie le type de modèle qui sera créé ou mis à jour par la génération.
Nom de la commande	Spécifie le nom de la commande qui sera affichée dans l'interface sous **Outils > Générer des objets**. Cette zone est initialisée lorsque vous sélectionnez un type de modèle cible.
Commentaire	Fournit une description de la génération ou d'autres informations supplémentaires.

4. [facultatif] Cliquez sur l'onglet **Extensions source** et/ou **Extensions cible** et sélectionnez les fichiers d'extension contenant des attributs étendus, collections ou métaclasses à référencer dans vos correspondances.

   Le fait d'attacher des fichiers d'extension de cette façon permet de réutiliser des extensions déjà définies dans vos générations ou de partager des extensions entre générations. Vous pouvez également définir des extensions appropriées sous la catégorie Profile dans le fichier de ressource contenant la définition de génération.

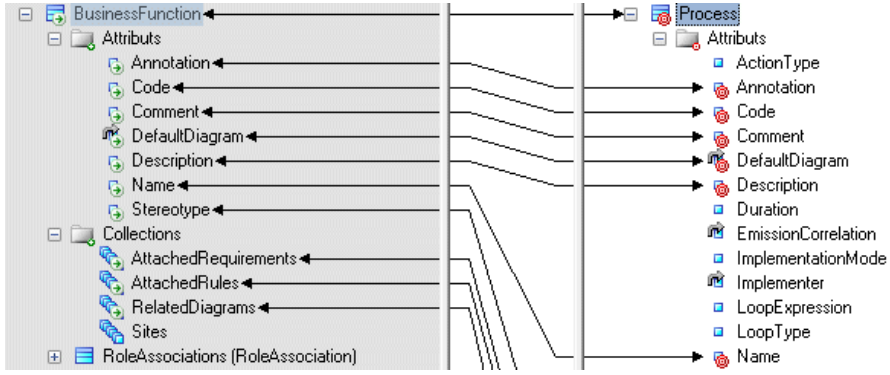
5. Cliquez sur l'onglet **Correspondance** pour définir les correspondances de vos métaclasses source vers vos métaclasses cible dans l'Editeur de correspondances (voir *Correspondances de génération intermodèle* à la page 114).
6. Cliquez sur **Appliquer** pour enregistrer vos modifications.

Correspondances de génération intermodèle

Vous contrôlez la façon dont les métaclasses d'un type de modèle PowerAMC seront générées dans des métaclasses d'un autre type de modèle en les mettant en correspondance, de même que leurs attributs et collections, dans l'Editeur de correspondances. Les extensions définies pour les métamodèles source ou cible sont affichées et disponibles pour mise en correspondance.

Remarque : Il n'est pas nécessaire de mettre en correspondance toutes les métaclasses (ou tout leur contenu), mais seules celles que vous souhaitez utiliser. Si le métamodèle PowerAMC ne contient pas de métaclasse, attribut, composition ou agrégation pouvant donner lieu à une correspondance, vous devez enregistrer les correspondances existantes, fermer l'Editeur de correspondances, définir ou attacher les extensions appropriées, puis réouvrir l'Editeur de correspondances pour les mettre en correspondance.

1. Faites glisser une métaclasse du volet Source situé à gauche vers le volet Cible situé à droite. Les éventuels attributs source sont automatiquement mis en correspondance avec des attributs cible qui portent le même nom :



Remarque : Par défaut, l'Editeur de correspondances répertorie les attributs et collections standard des métaclasses, qui sont affichés, par défaut, dans des feuilles de propriétés. Pour afficher toutes les propriétés disponibles, cliquez sur l'outil **Filtrer les propriétés**, puis sélectionnez `Afficher toutes les propriétés`. Vous pouvez également filtrer l'arborescence en utilisant les outils **Filtrer les correspondances** et **Filtrer les objets**.

2. Faites glisser des attributs source supplémentaires sur des attributs cible ayant des types de données compatibles afin de les mettre en correspondance. Les attributs sont contenus dans un dossier sous la métaclasse et représentent des propriétés individuelles telles que

Name, Size, DimensionalType, contenant des valeurs de type booléen, texte , numérique ou ID d'objet :

3. Faites glisser des métaclasses de sous-objet (compositions) sous la métaclasse vers des compositions cible pour créer des correspondances entre elles :

   Tous les attributs situés sous la métaclasse de sous-objet sont automatiquement mis en correspondance avec des attributs cible qui portent le même nom. Mettez en correspondance les autres attributs de sous-objet appropriés.

   Remarque : Dans certaines circonstances, il peut s'avérer approprié de mettre en correspondance une métaclasses de sous-objet source et une métaclasse d'objet cible, et de telles correspondances sont permises.

4. Faites glisser des collections source (agrégations) sous la métaclasse sur des collections cible pour créer des correspondances entre elles :
5. [facultatif] Sélectionnez une métaclasse et saisissez un script d'initialisation ou de post-traitement afin de modifier les objets ou à la création (voir *Propriétés d'une correspondance de métamodèle* à la page 111).
6. [facultatif] Cliquez sur le modèle cible (noeud racine) afin d'afficher la liste de toutes les correspondances dans le volet **Correspondances** en bas de la boîte de dialogue, et utilisez les flèches en bas de la liste pour changer l'ordre dans lequel les objets sont générés afin de vous assurer que les dépendances sont générées.

   Remarque : Pour contrôler l'ordre dans lequel les attributs compositions, et agrégations sont générés, sélectionnez la métaclasse cible pour afficher ses correspondances dans le volet **Correspondances**, puis utilisez les flèches en bas des listes sur les sous-onglets **Correspondance des attributs**, **Correspondances des collections** et **Correspondances des sous-objets**.

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

Script global (Profile)

Le profil contient un script global, que vous pouvez utiliser pour stocker des fonctions et des variables à réutiliser dans vos scripts définis pour des extensions.

Par exemple, on peut imaginer rédiger une fonction pour obtenir le type de données d'un élément et le réutiliser dans les exemples de script de vérification personnalisée et de correction automatique (voir *Vérifications personnalisées (Profile)* à la page 80.

La nouvelle fonction `DataTypeBase` est saisie sur l'onglet **Script global** 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
```

Le script pour la vérification (voir *Exemple : Vérification personnalisée de MPD* à la page 82 peut être réécrit pour appeler la fonction comme suit :

```
Function %Check%(obj)
Dim c 'colonne d index temporaire
 Dim col 'colonne temporaire
 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 & " : Le
type de données n'est pas compatible avec Index " & obj.name & " type
" & obj.type
    %Check% = False
   End If
  Next
 End If
End Function
```

Remarque : Les variables définies sur l'onglet **Script global** sont réinitialisées chaque fois qu'elles sont référencées dans un autre script.

Chapitre 3

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

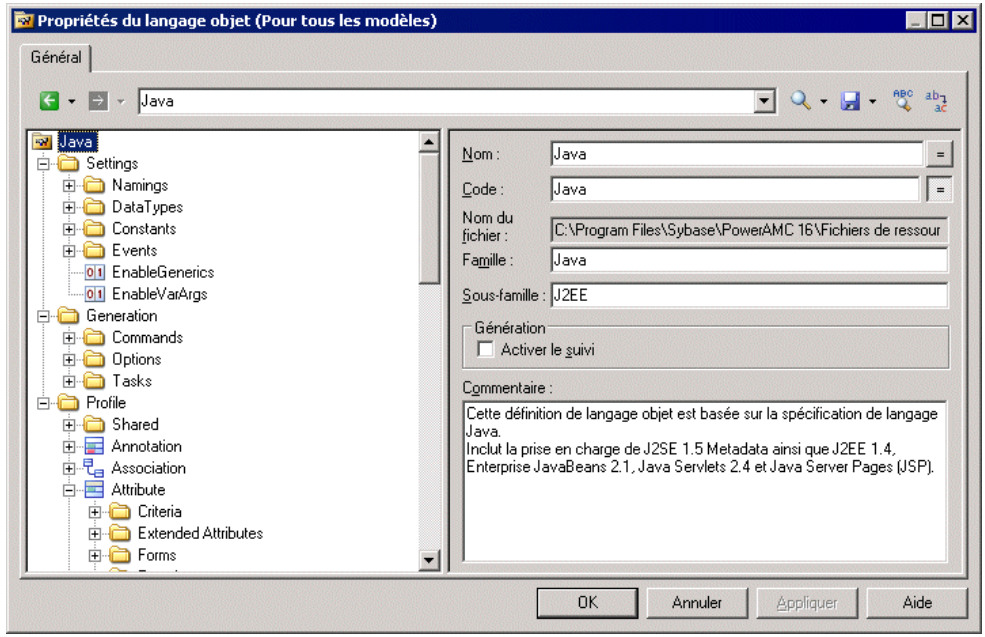
Les fichiers de définition de langage fournissent à PowerAMC les informations nécessaires pour modéliser, récupérer par reverse engineering et générer pour un langage orienté objet, de processus ou XML particulier. PowerAMC fournit des fichiers de définition pour de nombreux langages populaires. Vous sélectionnez un langage lorsque vous créez un MOO, MPM ou MSX.

Les fichiers de définition ont un suffixe `.xol`, `.xpl` ou `.xsl` et sont situés sous *répertoire_installation*`/Fichiers de ressources`. Pour afficher la liste des langages, sélectionnez **Outils > Ressources > Langages objet > , Langages de processus** ou **Langages XML**. Pour plus d'informations sur les outils disponibles dans les listes de fichiers de ressources, voir *Chapitre 1, Fichiers de ressources PowerAMC* à la page 1.

Remarque : Le MPD utilise une autre forme de fichier de définition (voir *Chapitre 4, Fichiers de définition de SGBD* à la page 129), et d'autres types de modèle n'ont aucun fichier de définition mais peuvent être étendus à l'aide de fichiers d'extension (voir *Chapitre 2, Fichiers d'extension* à la page 11).

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.



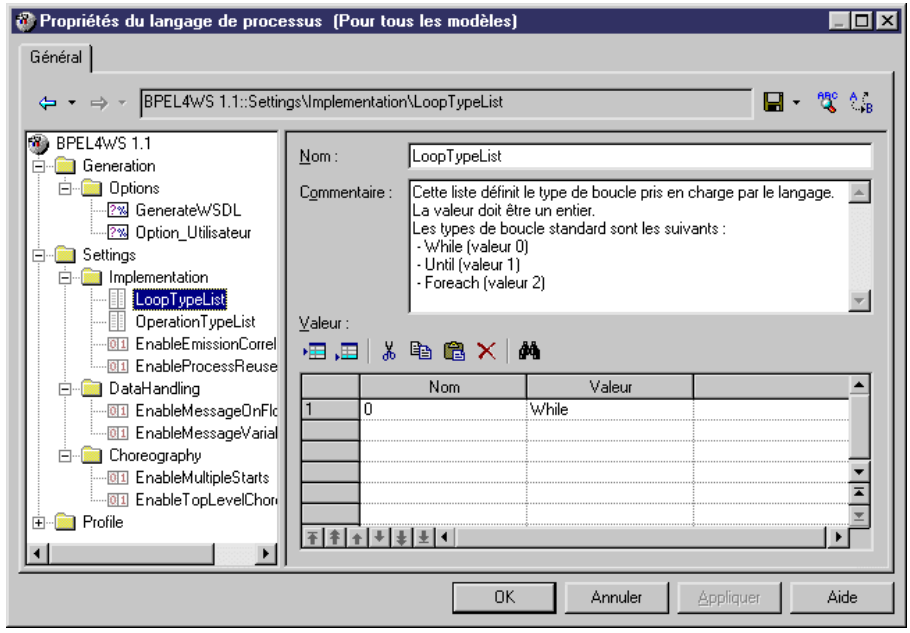
Le noeud racine de chaque fichier contient les propriétés suivantes Pro:

Propriété	Description
Nom / Code	Spécifie le nom et le code du fichier de définition de langage.
Nom de fichier	[lecteur seule] Spécifie le chemin du fichier de définition de langage. 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 / Sous-famille	Spécifie la famille et la sous-famille du langage, qui permet d'activer certaines fonctionnalités non-standard dans le modèle. Par exemple les langages objet des familles Java, XML, IDL et SAP® Sybase® PowerBuilder® prennent en charge le reverse engineering.
Activer le suivi	Permet de prévisualiser les templates utilisés lors de la génération (voir *Templates (Profile)* à la page 94). Avant même de lancer la génération, vous pouvez afficher la page **Aperçu** de la feuille de propriétés d'objet appropriée, et cliquer sur l'outil **Réactualiser** pour afficher les templates. Lorsque vous double-cliquez sur une ligne de suivi dans la page **Aperçu**, l'Editeur de ressources s'ouvre sur la définition de template correspondante.
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.

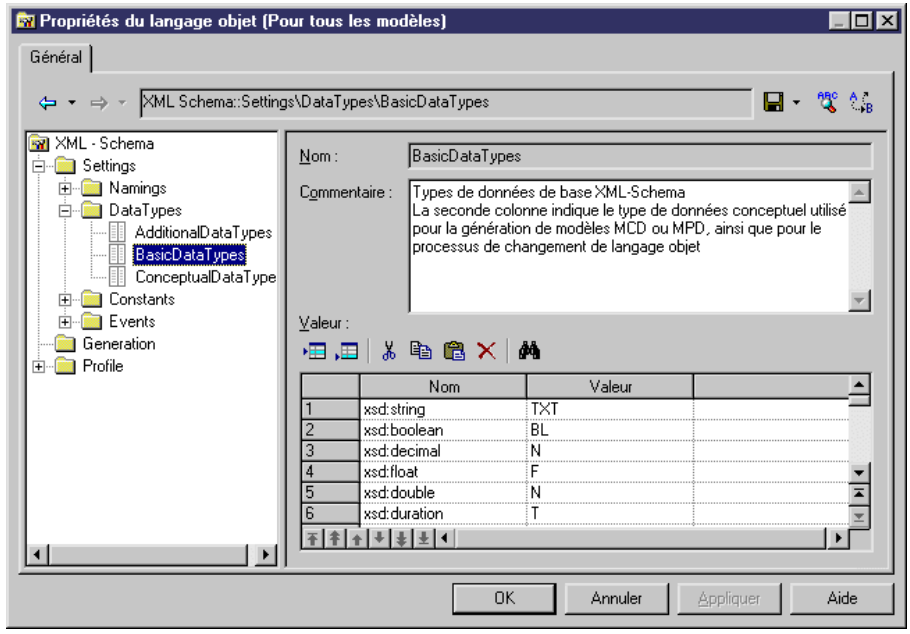


- *DataHandling* - [MPM exécutable uniquement] Rassemble des options relatives à la gestion des données dans le langage. Les valeurs constantes suivantes sont définies par défaut :
  - *EnableMessageOnFlow* - Indique si un format de message peut ou non être associé à un flux. La valeur par défaut est 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 > Conventions 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 :

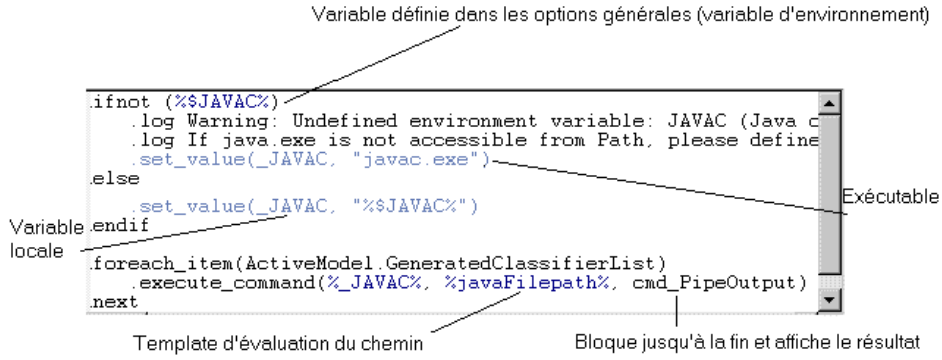
- ConceptualDataTypes - La colonne Valeur indique le type de données de langage XML utilisé pour la génération des modèles. Les types de données conceptuels sont les types de données internes de PowerAMC, et ils ne peuvent pas être modifiés.
- XsmDataTypes- Types de données pour les générations depuis 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 265), 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 :

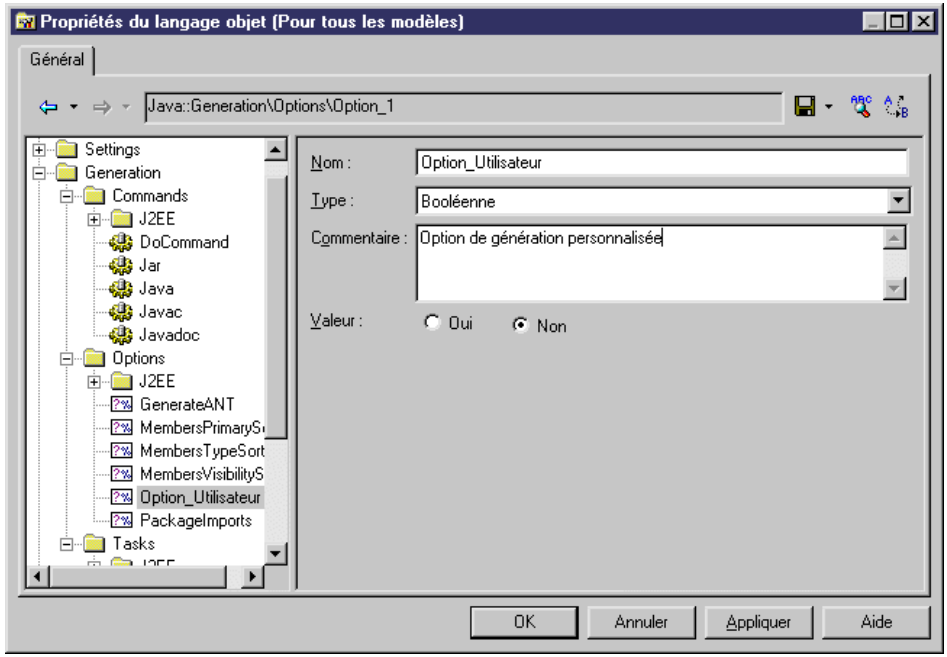
  ```
  %GenOptions.option%
  ```

  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.

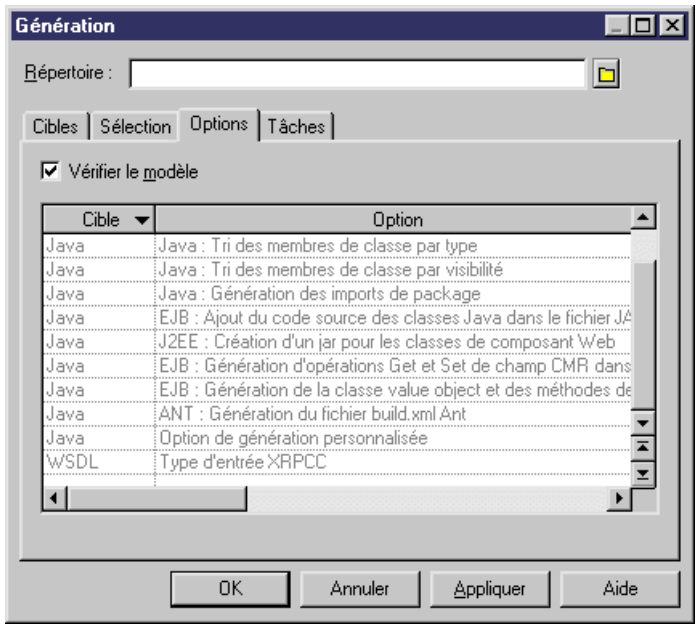
Exemple : Ajout d'une option de génération

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, pointez sur la catégorie Options, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau** :



3. Cliquez sur **OK** pour enregistrer vos modifications et revenir au modèle. Sélectionnez ensuite **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) :

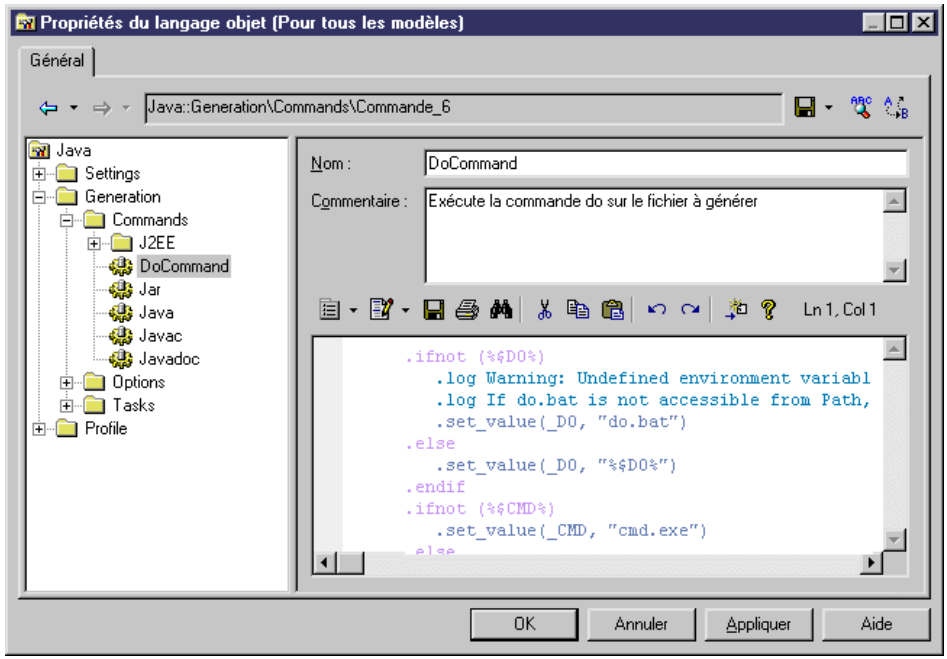


Remarque : Pour obtenir des informations détaillées sur la création ou la modification de templates de génération, voir *Chapitre 5, Personnalisation de la génération à l'aide du langage de génération par template* à la page 265.

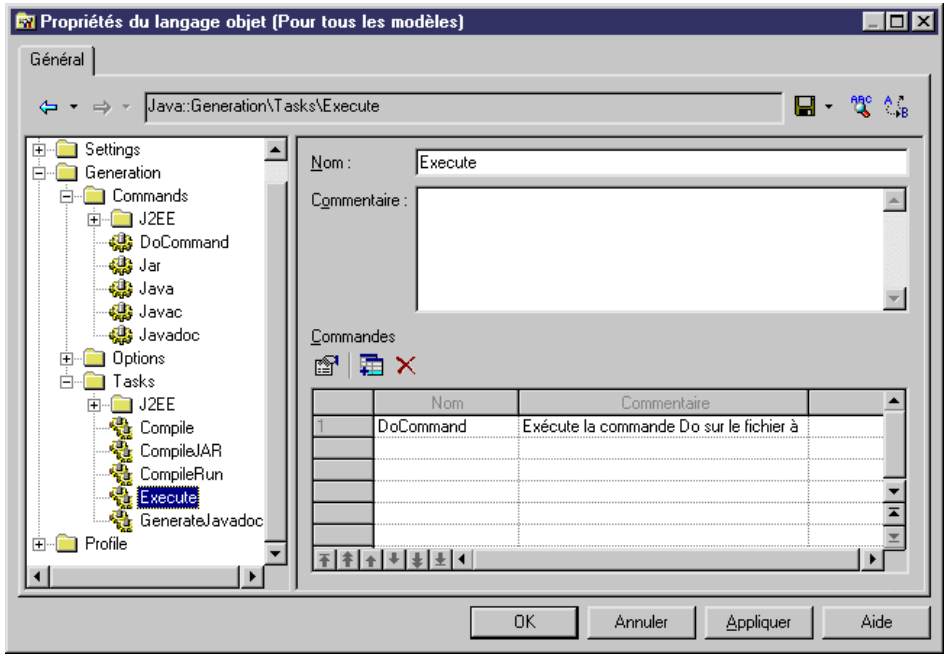
Exemple : Ajout d'une commande et d'une tâche de génération

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

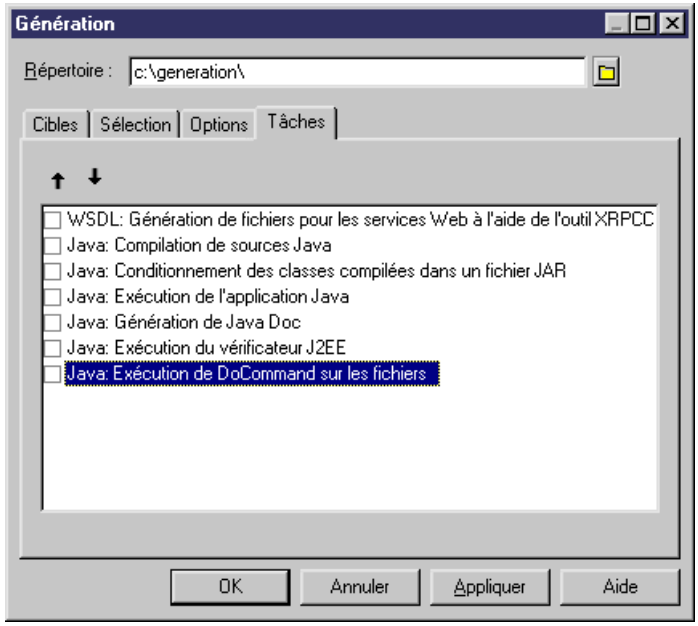
1. Créez un nouveau MOO pour Java, puis sélectionnez **Langage > Editer le langage objet courant**.
2. Développez la catégorie Generation, pointez sur la catégorie Commands, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau**.
3. Nommez la commande `DoCommand` et saisissez le template approprié :



4. Pointez sur la catégorie Tasks, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau**. Nommez la tâche `Execute`, cliquez sur l'outil **Ajouter des commandes**, sélectionnez `DoCommand` dans la liste, puis cliquez sur **OK** pour l'ajouter à la nouvelle tâche :



5. Cliquez sur **OK** pour enregistrer vos modifications et revenir au modèle. Sélectionnez. Sélectionnez ensuite **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) :



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 divers attributs utilisés pour contrôler la prise en charge du langage objet par PowerAMC. La variable **ObjectContainer** spécifie le conteneur par défaut pour la mise en oeuvre des associations. Cet attribut dispose d'une liste modifiable de valeurs possibles pour chaque langage objet, dans laquelle vous pouvez sélectionner une valeur par défaut pour votre langage. Vous pouvez le cas échéant redéfinir 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 11.

Chapitre 4 Fichiers de définition de SGBD

Un fichier de définition de SGBD fournit à PowerAMC les informations nécessaires pour la modélisation, le reverse engineering et la génération pour un SGBD particulier. PowerAMC fournit des fichiers de définition pour les SDGBD les plus populaires. Vous pouvez sélectionner un SGBD lorsque vous créez un MPD.

Les fichiers de définition de SGBD ont un suffixe `.xdb` et sont situés sous *répertoire_installation*`/Fichiers de ressources/SGBD`. Pour consulter la liste de SGBD, sélectionnez **Outils > Ressources > SGBD**. Pour plus d'informations sur les outils disponibles dans les listes de fichiers de ressources, voir *Chapitre 1, Fichiers de ressources PowerAMC* à la page 1.

Vous pouvez consulter ou modifier le fichier de définition de SGBD attaché à votre MPD dans l'Editeur de ressources en sélectionnant **SGBD > Editer le SGBD courant**. Lorsque vous sélectionnez une *catégorie* ou un élément dans le volet de gauche, les nom, valeur et commentaire associés sont affichés dans la partie droit de la boîte de dialogue.

Avertissement ! Les fichiers de ressource fournis avec PowerAMC dans le dossier `Program Files` ne peuvent pas être modifiés directement. Pour créer une copie à des fins d'édition, utilisez l'outil **Nouveau** dans la liste de fichiers de ressource, puis enregistrez-la à un autre emplacement. Pour inclure des fichiers de ressource provenant d'autres emplacements afin de les utiliser dans vos modèles, utilisez l'outil **Chemin** dans la liste des fichiers de ressource.

Chaque SGBD a la structure de fichiers suivantes :

- *General* - contient des informations générales sur la base de données, sans catégorie *Catégorie General (SGBD)* à la page 145). 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 *Syntax (catégorie de SGBD)* à la page 147)
    - *Format* - paramètres relatifs aux caractères admis (voir *Format (catégorie de SGBD)* à la page 148)
    - *File* - éléments de texte header, footer et usage utilisés lors de la génération (voir *File (catégorie de SGBD)* à la page 149)
    - *Keywords* - liste des mots réservés SQL et des fonctions (voir *Keywords (catégorie de SGBD)* à la page 151)
  - *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 (SGBD)* à la page 153).
    - *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 Category (SGBD)* à la page 214).
    - *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 (voir *Transformations (Profile)* à la page 102).
- *Profile* - permet de définir des types d'attributs étendus et des attributs étendus pour les objets de base de données (voir *Catégorie Profile (SGBD)* à la page 217).

Les propriétés suivantes sont disponibles à la racine d'un fichier de définition de SGBD :

Propriété	Description
Nom / Code	Nom et code du SGBD.
Nom de fichier	[lecture seule] Chemin d'accès et nom du fichier de SGBD.
Famille	Utilisé pour classifier un SGBD, et pour établir un lien entre différents fichiers de ressources de base de données. Par exemple SAP® Sybase® SQL Anywhere®, et SAP® Sybase® Adaptive Server® 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*

Génération et reverse engineering de base de données

PowerAMC prend en charge la génération et le reverse engineering de bases de données à la fois par scripts et via des connexions directes par l'intermédiaire des instructions et requêtes SQL stockées dans la catégorie `Script/Objects`. La génération et le reverse engineering de scripts et la génération via une connexion directe utilisent les mêmes instructions, tandis que le reverse engineering à partir d'une connexion directe utilise des requêtes séparées.

PowerAMC effectue la génération et le reverse engineering comme suit :

- Génération/Mise à jour de base de données - Chaque objet de modèle sélectionné est appliqué aux instructions dans la catégorie `Script/Objects`.
- Reverse engineering :
  - Script - PowerAMC analyse le script et identifie des instructions de création d'objet en les comparant aux instructions contenues dans le catégorie `Script/Objects`.
  - Connexion directe - PowerAMC utilise les requêtes de la catégorie `Script/Objects` afin d'extraire les informations des tables système de la base de données. Chaque colonne d'un jeu de résultats de requête est associée à une variable. L'en-tête de la requête 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 ensuite validées comme attributs d'objet.

Génération de script

PowerAMC peut générer un script SQL à partir d'un MPD pour créer ou modifier une base de données. Les instructions qui contrôlent la génération de script sont disponibles dans la catégorie `Script/Objects`.

Lorsque vous générez un script SQL, PowerAMC traite tour à tour chaque objet à créer, et lui applique l'instruction `Create` ou les autres instructions appropriées afin de créer ou de modifier cet objet :

- `Create` - Crée un nouvel objet.
- `Alter` / `Modify` - Modifie les attributs d'un objet existant.
- `Add` - Crée un nouveau sous-objet. Si des clés sont définies dans une table, elles seront créées à l'aide d'une instruction `Add`, mais si elles sont créées hors de la table, elles seront créées à l'aide d'une instruction `Modify`.
- `Rename` - Renommer un objet.
- `Drop` - Supprime un objet (à utiliser lorsqu'une instruction Alter n'est pas possible).
- *`Object`*`Comment` - Ajoute un commentaire sur l'objet.
- `Options` - Définit les options physiques pour un objet.

- `ConstName` - Définit le template de nom de contrainte pour les vérifications d'objet.

Par exemple, dans Sybase Adaptive Server® Enterprise 15.7, l'instruction `Create` dans la catégorie Table se présente comme suit :

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

Cette instruction contient les paramètres pour créer la table avec son propriétaire et ses options physiques à l'aide de variables (voir *Variables pour les tables et les vues* à la page 234) qui extraient les informations nécessaires des propriétés de l'objet. La variable `%TABLDEFN%` collecte les éléments `Add` dans les catégories Column, PKey, Key et Reference, et l'élément `AddTableCheck` dans la catégorie Table.

D'autres instructions dans les catégories d'objet sont utilisées afin de personnaliser l'interface et le comportement de PowerAMC en fonction des caractéristiques de la base de données, telles que `Maxlen`, `Permission`, `EnableOwner`et `AllowedADT`.

Extension de la génération à l'aide d'instructions Before et After

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

Les instructions d'extension sont rédigées en langage de génération par template (voir *Chapitre 5, Personnalisation de la génération à l'aide du langage de génération par template* à la page 265). Lors de la génération, les instructions et variables sont évaluées et le résultat est ajouté dans le script global.

Remarque : Nous vous recommandons d'éviter d'utiliser des macros du langage de génération par template (à l'exception de `.if`) dans des scripts de génération, car elles peuvent s'avérer impossibles à résoudre après un reverse engineering par script. Cette restriction ne porte pas sur la génération et le reverse engineering via une connexion directe.

Exemple - Ajout d'une instruction AfterCreate

L'instruction d'extension `AfterCreate` est définie dans la catégorie Table afin de compléter l'instruction `Create` de la table en ajoutant des partitions à la table si la valeur de l'attribut étendu de la partition le requiert :

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

La macro `.if` évalue la variable `%ExtTablePartition%`, qui est un attribut étendu qui contient le nombre de partitions de la table. Si la valeur est supérieure à 1, alors `%CreatePartition%`, défini dans la catégorie Table, sera généré comme suit :

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

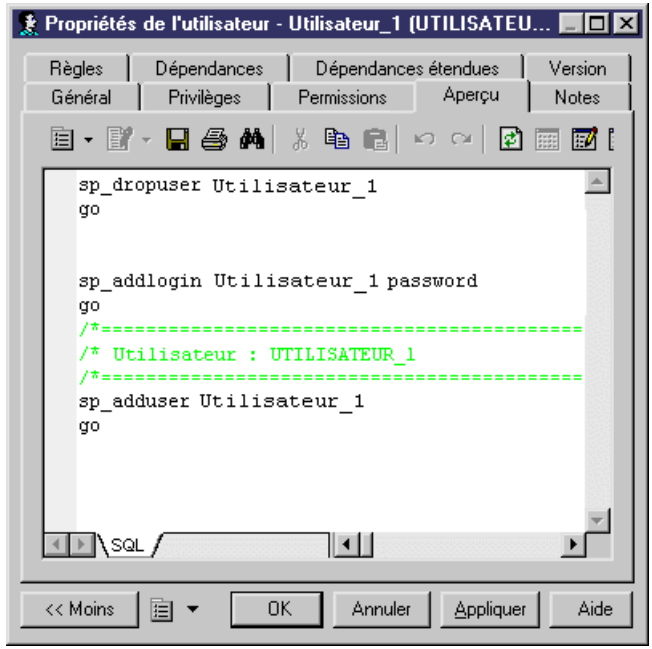
Cet élément génère l'instruction permettant de créer le nombre de partitions spécifié dans `%ExtTablePartition%`.

Exemple - Ajout d'une instruction BeforeCreate

L'instruction d'extension `BeforeCreate` est définie dans la catégorie User pour créer l'ID utilisateur d'un utilisateur avant l'exécution de l'instruction `Create` sur l'utilisateur :

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

L'ID utilisateur généré automatiquement aura le même nom et le même mot de passe que l'utilisateur. L'instruction `BeforeCreate` est affichée avant l'instruction de création de l'utilisateur dans l'**Aperçu** :



Exemple - 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 le schéma créé 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 dans le modèle depuis la dernière génération. Vous pouvez passer outre cette règle en ajoutant des attributs étendus dans la 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 une ancienne valeur de l'objet
- `%NEWOBJECT%` - pour accéder à une nouvelle valeur de l'objet

Par exemple, vous pouvez vérifier que la valeur de l'attribut étendu `ExtTablePartition` a été modifiée grâce à la syntaxe de langage de génération par template suivante :

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

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

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

Reverse engineering de script

PowerAMC permet de procéder au reverse engineering de scripts SQL dans un MPD. Les instructions qui contrôlent la génération du script sont disponibles dans la catégorie `Script/Objects`.

Lorsque vous procédez au reverse engineering de scripts SQL dans un MPD, PowerAMC compare tour à tour chaque instruction avec toutes les instructions `Create` définies dans le fichier de définition de SGBD et, pour chaque correspondance, extrait toutes les informations disponibles afin de créer ou de mettre à jour les objets de MPD.

Les instructions utilisées dans le reverse engineering de script sont les mêmes que celles utilisées pour la génération de script (voir *Génération de script* à la page 131).

Par exemple, dans Sybase IQ v15.2, l'instruction `Create` dans la catégorie Table se présente comme suit :

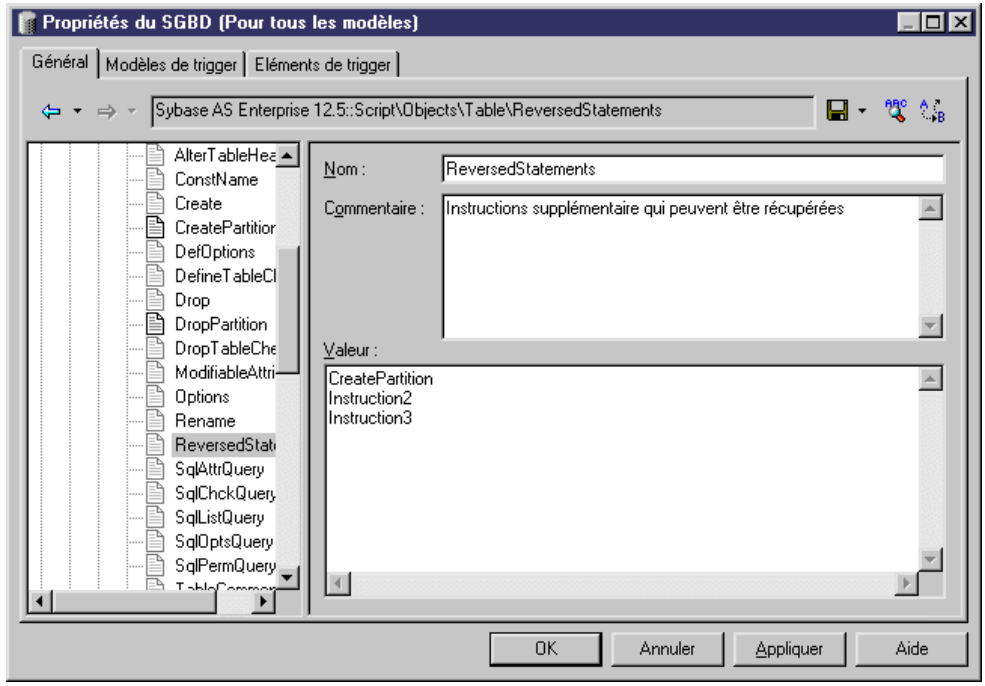
```
create[%ExtGlobalTemporaryTable%? global temporary] table
[%QUALIFIER%]%TABLE% (
   %TABLDEFN%
)[.Z:[[%R%?[.O:[in][on]] %DBSpace%:[%DBSpace%?
   in %DBSpaceGeneratedName%]]][
   on commit %OnCommit%][%NotTransactional%? not transactional][
   at %.q:At%][%R%?partition by range %RevPartition%:[%PartitionKey
```

```
%?[%hasLifecycle%?:
   partition by range (%PartitionKey.Code%)
   (
      %PartitionDef%
   )]]]
]
```

Cette instruction contient les paramètres pour la création de la table ainsi que son propriétaire et ses options physiques à l'aide de variables (voir *Variables pour les tables et les vues* à la page 234) qui extraient les information nécessaires des propriétés de l'objet.

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

Par exemple, l'instruction d'extension `AfterCreate` utilise `CreatePartition`, qui doit être déclarée dans `ReversedStatements` pour être correctement traitée par le reverse engineering :



Génération directe de base de données

PowerAMC peut générer ou modifier une base de données à partir d'un MPD via une connexion directe. Les instructions qui contrôlent la génération directe sont disponibles dans la catégorie `Script/Objects`, sauf lorsque le SGBD ne prend pas en charge la syntaxe SQL standard. Par exemple, MS Access, qui a besoin des scripts VB pour créer des objets de

base de données, a des instructions de génération spécifiques définies dans la catégorie ODBC.

Lorsque vous générez via une connexion directe, PowerAMC traite tour à tour chaque objet à créer, et lui applique l'instruction `Create` ou les autres instructions appropriées afin de créer ou de modifier cet objet.

Les instructions utilisées dans la génération directe sont les mêmes que ceux utilisés pour la génération de script (voir *Génération de script* à la page 131).

Reverse engineering direct de base de données

PowerAMC peut procéder au reverse engineering dans un MPD à partir d'une connexion directe à une base de données . Les requêtes qui contrôlent le reverse engineering direct sont disponibles dans la catégorie `Script/Objects`.

Les requêtes suivantes sont utilisées dans le reverse engineering via une connexion directe :

- `SqlListQuery` - Dresse la liste des objets disponibles pour renseigner la boîte de dialogue de reverse engineering. Cette requête est gourmande en termes de mémoire et doit extraire le plus petit nombre de colonnes possible. Si elle n'est pas définie, c'est `SqlAttrQuery` qui est utilisée pour renseigner la boîte de dialogue.
- `SqlAttrQuery` - Extrait les attributs de l'objet à récupérer. Cette requête n'est pas nécessaire si l'objet a peu d'attributs, et que `SqlListQuery` peut récupérer toutes les informations nécessaires, comme c'est le cas pour les tablespaces dans Sybase SQL Anywhere®.
- `SqlOptsQuery` - Extrait les options physiques à récupérer.
- `SqlListChildrenQuery` - Extrait des listes d'objets enfant (telles que les colonnes d'un index ou les clés ou jointures d'une référence) à récupérer.
- `SqlSysIndexQuery` - Extrait les index système créés par la base de données.
- `SqlChckQuery` - Extrait les contraintes de vérification d'objet.
- `SqlPermQuery` - Extrait les permissions sur les objets.

Remarque : Vous pouvez également créer vos propres requêtes (voir *Création de requêtes pour récupérer des attributs supplémentaires* à la page 138).

Chaque type de requête a la même structure de base constituée d'une liste de variables PowerAMC séparées par des virgules et encadrées d'accolades { } et suivie d'une instruction Select qui permet d'extraire les valeurs destinées à renseigner ces variables. Les valeurs des enregistrements renvoyés sont stockées dans ces variables, qui sont ensuite validées comme valeurs d'attributs d'objet.

Par exemple, `SqlListQuery` dans la catégorie View d'Oracle 11g R1 extrait des valeurs pour huit variables :

```
{OWNER, VIEW, VIEWSTYLE, ExtObjViewType,
    ExtObjOIDList, ExtObjSuperView, XMLSCHEMA EX, XMLELEMENT EX}

select
```

```
   v.owner,
   v.view_name,
   decode (v.view_type, 'XMLTYPE', 'XML', 'View'),
   v.view_type,
   v.oid_text,
   v.superview_name,
   decode (v.view_type, 'XMLTYPE', '%SqlXMLView.'||v.owner||
v.view_name||'1%', ''),
   decode (v.view_type, 'XMLTYPE', '%SqlXMLView.'||v.owner||
v.view_name||'2%', '')
from sys.all_views v
[where v.owner = %.q:SCHEMA%]
```

Chaque partie de l'en-tête placé entre virgules est associée aux informations suivantes :

- Nom de la variable - [obligatoire] peut être n'importe quelle variable de MPD (voir *Variables et macros de MPD* à la page 229), nom public du métamodèle (voir *Navigation dans le métamodèle* à la page 372) ou bien le nom d'un attribut étendu défini sous la métaclasse dans la catégorie Profile (voir *Catégorie Profile (SGBD)* à la page 217).
- `ID` - [facultatif] la variable fait partie de l'identifiant.
- `...` - [facultatif] la variable doit être concaténée pour toutes les lignes renvoyées par la requête SQL et ayant les mêmes valeurs pour les colonnes d'ID. Les mots clé `ID` et `...` (points de suspensions) sont mutuellement exclusifs.
- Paires de valeurs - [facultatif] répertorie les conversions entre des valeurs extraites et des valeurs dans PowerAMC au format suivant (où `*` signifie toutes les autres valeurs) :

  ```
  (value1 = PDvalue1, value2 = PDvalue2, * = PDvalue3)
  ```

Exemple : Utilisation de ID pour définir l'identifiant

Dans ce script, l'identifiant est défini comme `TABLE + ISKEY+ CONSTNAME` via l'utilisation du mot clé `ID` :

```
{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 les lignes de résultats renvoyées par le script SQL, les valeurs du quatrième champ sont concaténées dans le champ COLUMNS tant que leurs valeurs d'ID sont identiques.

```
SQL Result set
Table1,1,null,' ,'
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 séparée par des virgules. PowerAMC va traiter le contenu du champ COLUMNS pour supprimer la dernière virgule.

Exemple : Conversions de paires de valeurs

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

```
{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
```

Création de requêtes pour récupérer des attributs supplémentaires

Vous pouvez créer des requêtes afin de récupérer des attributs supplémentaires. Ces attributs peuvent être ajoutés dans `SqlAttrQuery`, mais le fait de les récupérer dans une requête distincte vous permet d'éviter de surcharger cet élément. Les requêtes utilisateur ne sont appelées que lors du reverse engineering si leur nom est ajouté dans l'élément `ReversedQueries`.

Pour créer une nouvelle requête dans une catégorie, pointez sur cette catégorie, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau > Elément Texte**. Saisissez le nom approprié, puis ajoutez ce nom dans l'élément `ReversedQueries` item.

Par exemple, dans la famille de SGBD Oracle, `SqlColnListQuery` est défini dans la catégorie View :

```
{OWNER ID, VIEW ID, VIEWCOLN ...}

select
   c.owner,
   c.table_name,
   c.column_name||', '
from
   sys.all_tab_columns c
where 1 = 1
   [and c.owner=%.q:OWNER%]
   [and c.table_name=%.q:VIEW%]
order by
   1, 2, c.column_id
```

Cette requête extrait des colonnes de vue, et est activée en l'ajoutant à to `ReversedQueries` dans la catégorie View.

Remarque : Les sous-requêtes qui sont appelés à l'aide du mot clé `EX` depuis `SqlAttrQuery` ou d'autres requêtes (voir *Appel de sous-requêtes à l'aide du mot clé EX* à la page 139) n'ont pas besoin d'être ajoutées dans `ReversedQueries`.

Appel de sous-requêtes à l'aide du mot clé EX

Les tables système de SGBD peuvent stocker des informations que vous pouvez souhaiter pouvoir récupérer par le reverse engineering dans des colonnes de type LONG, BLOB, TEXT et autres types de données non compatibles, et PowerAMC ne peut pas les concaténer directement dans des chaînes.

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 sous-requêtes que vous rédigez.

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||'%)',
    ...
```

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

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. Vous pouvez créer des requêtes utilisateur dans n'importer quelle requête de reverse engineering de base de données directe. Chaque requête doit avoir un nom unique.
2. L'exécution de la requête définie par l'utilisateur doit générer un jeu de résultats contenant pour chaque variable à évaluer des paires de variable définie par l'utilisateur (sans %) et de valeur de variable. 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

3. Ces valeurs remplacent les variables définies par l'utilisateur dans la requête d'origine.

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`.

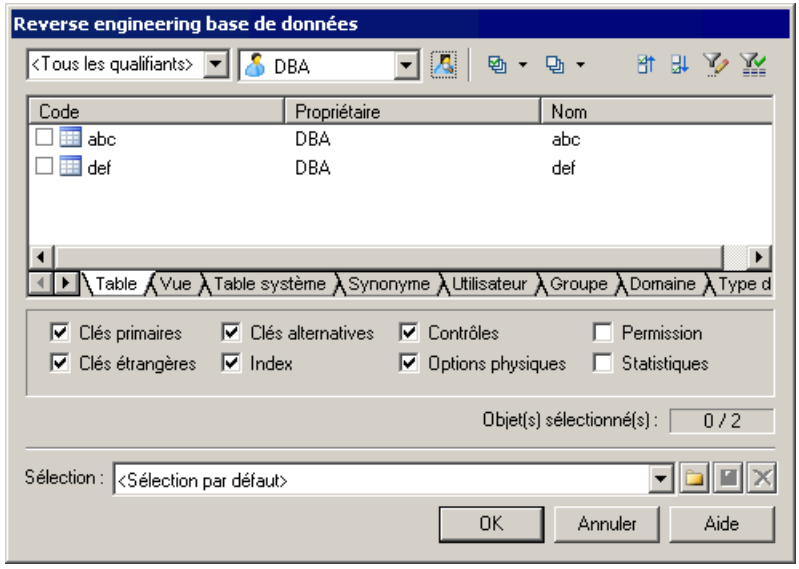
`SqlExpression` est suivi d'une variable définie par l'utilisateur comme suit :

```
{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
```

Définition de la génération et du reverse engineering des nouvelles métaclasses

Vous pouvez étendre votre SGBD pour y inclure de nouvelles métaclasses qui ne sont pas présentes dans le métamodèle PowerAMC standard. De nombreux SGBD contiennent de telles métaclasses, qui sont définies en créant un stéréotype sur une métaclasses existante, et vous pouvez créer votre propre métaclasse. Pour inclure ces objets dans la génération et le reverse engineering, vous devez les ajouter dans la catégorie `Script/Objects`, et définir les instructions et requêtes SQL appropriées.

1. Créez une nouvelle métaclasse dans votre fichier de définition de SGBD en définissant un nouveau stéréotype sur une métaclasse existante, et en sélectionnant l'option **Utiliser comme métaclasse** (voir *Création de nouvelles métaclasses à l'aide de stéréotypes* à la page 42).
2. Définissez les attributs étendus appropriés (voir *Attributs étendus (Profile)* à la page 45) ainsi que les éventuelles extensions nécessaires pour définir avec précision la nature de votre objet.
3. Pointez sur la catégorie Script/Objects, cliquez le bouton droit de la souris, puis sélectionnez **Ajouter des éléments**, sélectionnez votre nouvel objet dans la liste, puis cliquez sur **OK** pour l'ajouter dans la catégorie.
4. Pointez sur l'entrée du nouvel objet, cliquez le bouton droit de la souris, puis sélectionnez **Ajouter des éléments** pour lui ajouter les éléments de script nécessaires. Vous devez ajouter au minimum les éléments suivants pour permettre la génération et le reverse engineering de l'objet :
   - `Create`
   - `Drop`
   - `AlterStatementList`
   - `SqlAttrQuery`
   - `SqlListQuery`
5. Cliquez sur **OK** pour ajouter ces éléments de script à votre objet, et saisissez les instructions et requêtes SQL appropriées. Vous devrez saisir des valeurs pour chacun de ces éléments. Pour obtenir de l'aide sur la syntaxe, voir *Eléments communs aux différents objets* à la page 155.

6. [facultatif] Pour contrôler l'ordre dans lequel cet objet et les autres objets seront générés, utilisez l'élément `Generation Order` (voir *Catégorie Script/Objects (SGBD)* à la page 153).

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**.
2. Sur le sous-onglet PdPDM, sélectionnez Model, puis cliquez sur **OK** pour ajouter l'élément Model dans le dossier Profile.
3. Pointez sur l'élément Model, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau > Gestionnaire d'événement** (voir *Gestionnaires d'événement (Profile)* à la page 84).
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 ajouter les gestionnaires d'événement sélectionnés dans 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 (SGBD)

La catégorie General est située directement sous la racine, et contient des éléments de haut niveau qui définissent le comportement de base du SGBD.

Elément	Description
EnableCheck	Spécifie si la génération des paramètres de contrôle est autorisée ou non. Les valeurs possibles sont les suivantes. Si cet élément est défini à `No`, aucune variable liée aux paramètre de contrôle ne sera évaluée lors de la génération ou du reverse engineering.

Elément	Description
EnableConstName	Spécifie si des noms de contraintes sont pris en charge par le SGBD. Si cet élément est défini à `Yes`, les noms de contrainte de table et de colonne sont générés en plus des contraintes elles-même.
EnableIntegrity	Spécifie si les contraintes d'intégrité sont prises en charge par le SGBD. Si cet élément est défini à `Yes`, les cases à cocher de clé primaire, alternative et étrangère sont disponibles pour la génération et la modification de base de données
EnableMultiCheck	Spécifie si la génération de plusieurs paramètres de contrôle pour les tables et les colonnes est prise en charge par le SGBD. Si cet élément est défini à `Yes`, plusieurs paramètres de contrôle sont générés, et 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. Si cet élément est défini à `No`, toutes les règles de gestion (validation et contrainte) sont concaténées dans une même expression de contrainte.
SchemaStereotype	Spécifie le stéréotype utilisateur à utiliser pour indiquer un schéma (propriétaire de l'objet).
SqlSupport	Spécifie si la syntaxe SQL est prise en charge par le SGBD. Si cet élément est défini à `Yes`, la syntaxe SQL est prise en charge et l'Aperçu SQL est disponible.
UniqConstName	Spécifie si les noms de contrainte uniques pour les objets sont requis par le SGBD. Si cet élément est défini à `Yes`, tous les noms de contrainte (y compris les noms d'index) doivent être uniques dans la base de données. Dans le cas contraire, les noms de contrainte ne doivent être uniques qu'au niveau de l'objet.
UserStereotype	Spécifie que le stéréotype utilisateur doit être utilisé pour un utilisateur (détenteur de permissions).

Catégorie Script/Sql (SGBD)

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

Syntax (catégorie de SGBD)

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 : `/* */`
BlockTerminator	Spécifie le caractère de fin de bloc, utilisé pour terminer les instructions telles que les instructions portant sur les triggers et procédures stockées.
Delimiter	Spécifie le caractère de séparation de champs.
IdentifierDelimiter	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 vide, c'est `BlockTerminator` qui est utilisé.
UseBlockTerm	Spécifie l'utilisation de `BlockTerminator`. Les valeurs possibles sont les suivantes : • Yes - `BlockTerminator` est toujours utilisé • No - `BlockTerminator` est utilisé pour les triggers et les procédures stockées uniquement

Format (catégorie de SGBD)

La catégorie Format est située dans la catégorie **Racine > Script > SQL**, et contient les éléments suivants qui définissent le format 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
CaseSensitivityUsingQuote	Spécifie si la sensibilité à la casse est gérée à l'aide de guillemets. Activez cette option si le SGBD que vous utilisez nécessite des guillemets pour préserver la casse des codes d'objet.
DateTimeFormat / OdbcDateTimeFormat / DateFormat / OdbcDateFormat / TimeFormat / OdbcTimeFormat	Spécifie le format utilisé pour générer des données de test relatives à la date et à l'heure dans un script ou via une connexion directe à une base de données. • `yyyy`/`yy`, `mm`, `dd` - Années, mois et jours. • `HH`, `MM`, `SS` - Heures, minutes et secondes. Par exemple, vous pouvez définir la valeur suivante pour l'entrée `DateTimeFormat` pour SQL : `yy-mm-dd HH:MM`. Reportez-vous à l'élément `Script\DataType\PhysDataType` (voir *Catégorie Script/Data Type Category (SGBD)* à la page 214) pour voir comment PowerAMC convertit les types de données de date et d'heure dans votre SGBD en types de données conceptuels internes.
EnableOwnerPrefix / EnableDtbsPrefix	Spécifie que les codes d'objet peuvent être préfixés par le nom du propriétaire de l'objet (`%OWNER%`), le nom de la base de données (`%DBPREFIX%`), ou les deux (`%QUALIFIER%`). Les valeurs possibles sont les suivantes : • Yes – active les options Préfixe de base de données e/out Préfixe de propriétaire dans la boîte de dialogue de génération de base de données afin de requérir l'un de ces types de préfixe, ou les deux, pour les objets. • No - Les options Préfixe de base de données et Préfixe de propriétaire sont indisponibles **Remarque :** `EnableOwnerPrefix` active l'option de modèle **Ignorer le propriétaire** pour les tables et vues.

Elément	Description

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.

LowerCaseOnly / UpperCaseOnly	Lorsque vous générez un script, tous les objets sont générés en minuscules ou 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 minuscules ou majuscules. • No - Génère tout le script sans changer la façon dont les objets sont écrits dans le modèle. **Remarque :** Ces éléments sont mutuellement exclusifs. Si vous les activez tous les deux, le script est généré en *minuscules*.
MaxScriptLen	Spécifie la longueur maximale d'une ligne de script.

File (catégorie de SGBD)

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.

Elément	Description
EnableMultiFile	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.
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 et macros de MPD* à la page 229).
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).

Elément	Description
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.

Keywords (catégorie de SGBD)

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.
ConvertAnyToString	Spécifie une fonction permettant de convertir tout type de données en chaîne.
ConvertDateToMonth, ConvertDateToQuarter, ConvertDateToYear	Spécifie une fonction permettant d'extraire la partie pertinente d'une date.

Elément	Description

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`

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`

Elément	Description

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 (SAP® Sybase® 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**).
StringConcatena-tionOperator	Spécifie l'opérateur utilisé pour concaténer deux chaînes.

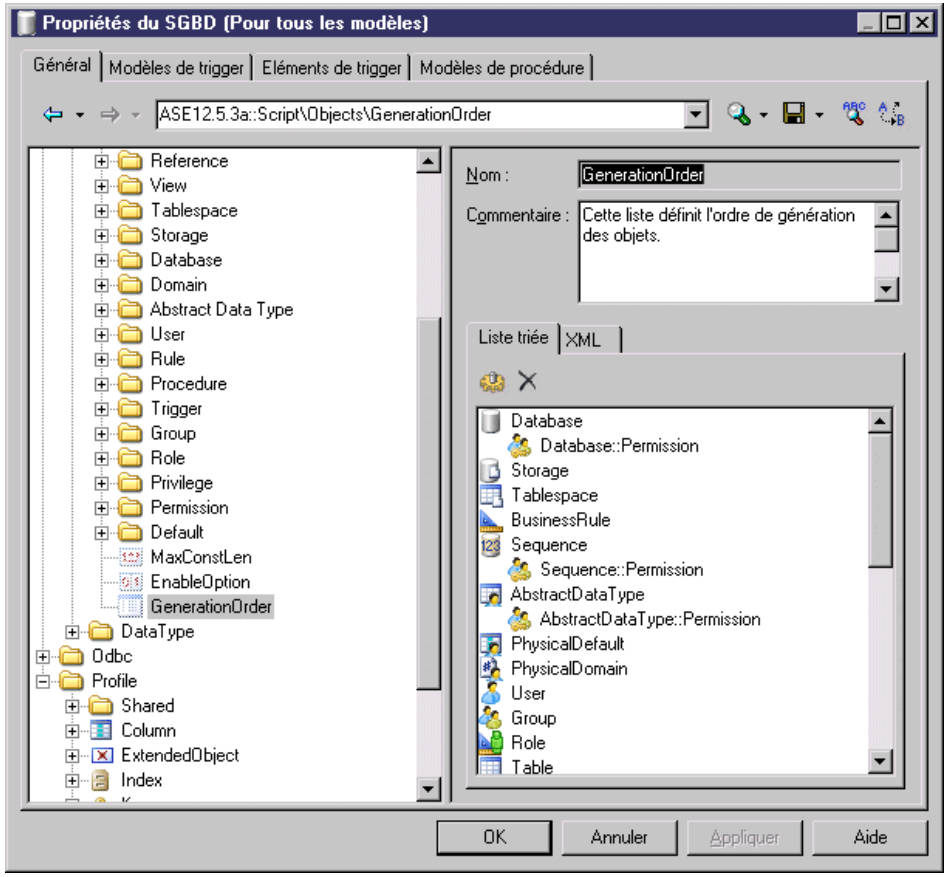
Catégorie Script/Objects (SGBD)

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

Les éléments suivants sont situés dans les catégories **Racine > Script > Objects** et **Racine > ODBC > Objects**, et s'appliquent à tous les objets :

- MaxConstLen - Spécifie la longueur maximale de nom de contrainte prise en charge pour l'objet dans la base de données cible pour les tables, colonnes, clés primaires et clés étrangères. 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 - Spécifie que les options physiques sont prises en charge par le SGBD cible pour le modèle, les tables, les index, les clés alternatives et d'autres objets, et active l'affichage de l'onglet **Options** dans les feuilles de propriétés d'objet. Pour plus d'informations, voir *Options physiques (SGBD)* à la page 224.
- GenerationOrder - Spécifie l'ordre de génération des objets de base de données. 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.



Remarque : Si un objet ne figure pas dans la liste, il sera généré malgré tout, mais uniquement après les objets présents dans la liste. Vous pouvez ajouter et supprimer des éléments en utilisant les outils de l'onglet. 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. Les objets étendus (voir *Définition de la génération et du reverse engineering des nouvelles métaclasses* à la page 144) ne peuvent pas être ajoutés dans cette liste, et son générés après tous les autres objets.

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%)]`
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/ AfterCreate / BeforeDrop / AfterDrop / BeforeModify / AfterModify	Spécifie les instructions étendues exécutées avant ou après les instructions principales Create, Drop ou Modify (voir *Génération de script* à la page 131).
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 (voir *Options physiques (SGBD)* à la page 224) qui seront appliquées à tous les objets. Ces valeurs doivent respecter la syntaxe SQL. Exemple : `in default_tablespace`

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')[%QUALI-FIER%?`
	`  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_%TA-BLE%:[%TABLQUALIFIER%]%TABLE% (`
	` %CIDXLIST%`
	`)]`
	`[%OPTIONS%]`
	%QUALIFIER% fait référence à l'objet courant (index) et %TABLQUA-LIFIER% fait référence à la table parent de l'index.

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

Elément	Description
MaxConstLen	Spécifie la longueur maximale pour la longueur de nom de contraine prise en charge dans la base de données cible, avec une valeur différente de la valeur par défaut spécifiée dans `MaxConstLen` (voir *Catégorie Script/ Objects (SGBD)* à la page 153).
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	Spécifier une liste d'attributs étendus qui seront pris en compte dans la boîte de dialogue de fusion lors d'une synchronisation de base de données (voir *Génération de script* à la page 131). Exemple (ASE 12.5) : `ExtTablePartition`
Options	Spécifie les options physiques (voir *Options physiques (SGBD)* à la page 224) disponibles pour application lors de la création d'un objet. Exemple (ASA 6) : `in %s : category=tablespace`

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`

ReversedQueries	Spécifie une liste de requêtes d'attributs supplémentaires à appeler lors du reverse engineering direct d'une base de données (voir *Reverse engineering direct de base de données* à la page 136).
ReversedStatements	Spécifie une liste d'instructions supplémentaires qui seront récupérées par reverse engineering (voir *Reverse engineering de script* à la page 134).

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_table_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 (catégorie de SGBD)

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 communs, voir *Eléments communs aux différents objets* à la page 155.

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:CONS-TRAINT%)`

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 à placer après les instructions `alter table` (et avant le caractère de fin). Exemple : `AlterTableFooter = /* End of alter statement */`
AlterTable Header	Spécifie une instruction à placer 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.constraint_id)`
	`join sys.systab t on (t.object_id = k.table_object_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 (catégorie de SGBD)* à la page 165.
SqlXMLTable	Spécifie une sous-requête utilisée pour améliorer les performances de SqlAttrQuery (voir *Eléments communs aux différents objets* à la page 155).

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 (catégorie de SGBD)

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 modélisés 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 155.

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 • La colonne Identity a un type de données approprié. 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 type de données non pris en charge, une erreur s'affiche. 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 172.

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 (catégorie de SGBD)

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 155. **Remarque :** Pour plus d'informations sur l'utilisation de variables dans SqlListQuery pour le reverse engineering des index basés sur des fonctions, voir *Reverse engineering direct d'index basés sur une fonction* à la page 141
AddColIndex	Spécifie une instruction pour ajouter une colonne dans l'instruction `Create Index`. paramètre définir chaque colonne dans la liste de colonnes de l'instruction `Create Index` statement. 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.
AlterIgnoreOrder	Spécifie que les changements dans l'ordre de la collection ne doivent pas donner lieu à une commande de modification de base de données.
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.

Elément	Description
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.
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 (catégorie de SGBD)

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 modélisés 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 155.
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 (catégorie de SGBD)

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 155.
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.
AlterIgnoreOrder	Spécifie que les changements dans l'ordre de la collection ne doivent pas donner lieu à une commande de modification de base de données.
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 suivantes sont disponibles : • 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 (catégorie de SGBD)

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 155.
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. See also `UseSpPrimKey` (*Pkey (catégorie de SGBD)* à la page 177).

View (catégorie de SGBD)

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 155.
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 (catégorie de SGBD)

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 155.
Tablespace Comment	Spécifie une instruction permettant d'ajouter commentaire au tablespace.

Storage (catégorie de SGBD)

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 155.

Elément	Description
Storage Comment	Spécifie une instruction permettant d'ajouter commentaire à un storage.

Database (catégorie de SGBD)

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 modélisés 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 155.
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 (catégorie de SGBD)

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 155.
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 (catégorie de SGBD)

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 155.
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 (catégorie de SGBD)

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 155.

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 (catégorie de SGBD)

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 155.
UserComment	Spécifie une instruction permettant d'ajouter commentaire à une utilisateur.

Rule (catégorie de SGBD)

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 modélisés 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 155.

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 EMPFUNC de la table EMPLOYEE a une valeur par défaut `Technical Engineer.` Le nom de colonne par défaut `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 `Domain_speciality` doit appartenir à un jeu de valeurs. Cette vérification de domaine a été définie dans une règle de validation. Le script SQL va générer le nom de règle en suivant le template défini dans l'élément `UddtRuleName` :
	`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 (catégorie de SGBD)

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 155.
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 (catégorie de SGBD)

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 155.
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 (catégorie de SGBD)

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 155.
EventDelimiter	Spécifie un caractère qui sépare les différents événements de trigger.

Item	Description
Events_*portée*	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*.

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 (catégorie de SGBD)

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 155.
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 (catégorie de SGBD)

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 155.
Label	Spécifie un libellé pour <Tout> dans la liste de sélection de qualifiant.

Sequence (catégorie de SGBD)

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 155.
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 (catégorie de SGBD)

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 155.
EnableAlias	Spécifie si les synonymes peuvent avoir un type d'alias.

Group (catégorie de SGBD)

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 155.
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 (catégorie de SGBD)

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 155.

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 (catégorie de SGBD)

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 155.

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 (catégorie de SGBD)

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 155.

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 (catégorie de SGBD)

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 155.

Privilege (catégorie de SGBD)

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 155.

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 (catégorie de SGBD)

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 155.
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 (catégorie de SGBD)

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 155.

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 (catégorie de SGBD)

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 155.
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 à une opération de service Web.

Web Parameter (catégorie de SGBD)

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 155.
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 (catégorie de SGBD)

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 (catégorie de SGBD)

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 modélisés 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 155.
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 (catégorie de SGBD)

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 155.
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 Category (SGBD)

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
OdbcPhysDataType	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`.
PhysOdbcDataType	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.
PhysLogADTType	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 11.

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.

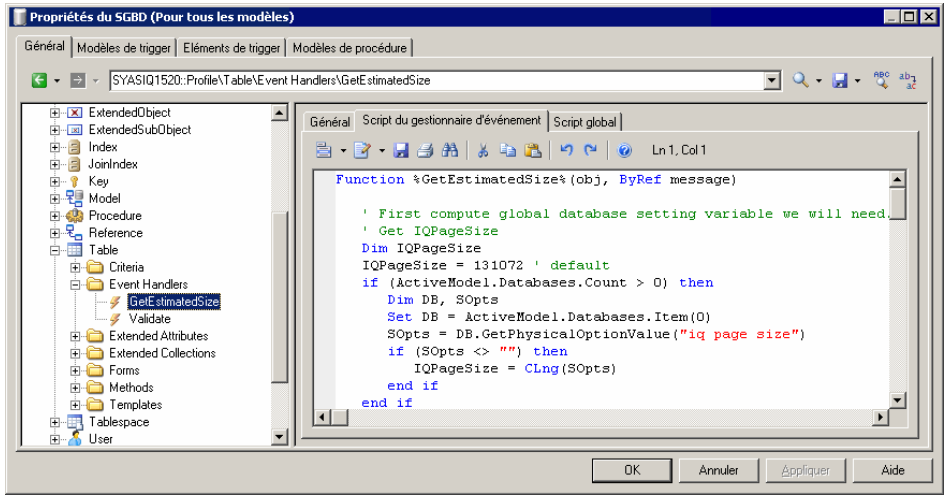
Dans l'élément `SqlListQuery` :

```
{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 222):

```
Function %GetEstimatedSize%(obj, ByRef message)

' Commencer par calculer la variable globale de base de données
qui sera nécessaire.

' Lire la taille de la table et conserver la taille de colonne pour
la suite
   Dim ColSizes, TblSize, ColSize, C
   Set ColSizes = CreateObject("Scripting.Dictionary")

   TblSize = 0 ' Peut être changé pour prendre en compte la taille
initiale de définition de table.

  for each C in obj.Columns
```

```
    ' Commencer à parcourir les colonnes de table et utiliser le
gestionnaire d''événement défini sur la métaclasse de colonne
(s''il existe).
      ColSize = C.GetEstimatedSize(message, false)

       ' Stocker la taille de colonne pour une future utilisation
dans les index.
      ColSizes.Add C, ColSize

        ' Augmenter la taille globale de la table.
      TblSize = TblSize + ColSize
   next
   Dim RawDataSize
   RawDataSize = BlockSize * int(obj.Number * TblSize / BlockSize)
    ' A ce stade, RawDataSize est la taille de la table dans la
base de données.
```

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 :

```
' Claculer la taille des index. Définir des variables pour stocker
ces tailles.
   Dim X, XMsg, XDataSize
   XMsg = ""
   for each X in obj.Indexes
      XDataSize = 0
     ' Parcourir les colonnes d'index et lire leur taille ajoutée
dans XDataSize
      For each C in X.IndexColumns
         XDataSize = XDataSize + ColSizes.Item(C.Column)
      next
      XDataSize = BlockSize * int(obj.Number * XDataSize /
BlockSize)

       ' Mettre en forme le message pour obtenir l'info de taiil
dans la fenêtre résultats et la liste de résultats.
      XMsg = XMsg & CStr(XDataSize) & "|" & X.ObjectID & vbCrLf

      ' Ajouter la taille d'index à la taille de table.
      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 222). 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 :

```
   ' Définir le message global avec la taille de table et tous les
index (séparés par un retour à la ligne).
   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.

The syntax for the output is as follows:

```
[taille-objet][:compartiment]|[ObjectID][|libellé]
```

where:

- *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 totale de toutes les colonnes de type LONG de 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...]
```

Catégorie ODBC (SGBD)

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.

Options physiques (SGBD)

Dans certains SGBD, des options supplémentaires sont utilisées afin de spécifier comment un objet est optimisé ou stocké dans une base de données. Dans PowerAMC, ces options sont appelées *options physiques* et sont affichées dans les onglets **Options physiques** et **Physical Options (Common)** des feuilles de propriétés d'objet.

Pour être affichée sur l'onglet **Options physiques** une option doit être définie dans l'élément `Script\Objects\objet\Options` (voir *Eléments communs aux différents objets* à la page 155). Les valeurs par défaut peuvent être stockées dans `Options` ou dans `DefOptions`. Pour apparaître dans l'onglet **Physical Options (Common)** (ou dans tout autre onglet de feuille de propriétés), l'option physique doit en outre être associée à un attribut étendu (voir *Ajout d'options physiques de SGBD dans vos formulaires* à la page 228).

Lors de la génération, les options sélectionnées dans le modèle pour chaque objet sont stockées dans une chaîne SQL dans la variable %OPTIONS%, qui doit se trouver à la fin de l'instruction Create de l'objet, et ne peut pas être suivie de quoique ce soit d'autre. L'exemple suivant montre la syntaxe appropriée :

```
create table
[%OPTIONS%]
```

Lors du reverse engineering par script, la section de la requête SQL identifiée comme étant l'option physique est stockée dans %OPTIONS%, et sera ensuite analysée lorsque requis par une feuille de propriétés d'objet.

Lors du reverse engineering direct de base de données, l'instruction SQL `SqlOptsQuery` est exécutée afin de récupérer les options physiques stockées dans %OPTIONS% qui seront ensuite analysées lorsque requis par une feuille de propriétés d'objet.

Vous pouvez utiliser des variables PowerAMC (voir *Variables et macros de MPD* à la page 229) pour définir des options physiques pour un objet. Par exemple, dans Oracle, vous pouvez

définir les variable suivantes pour un cluster afin de faire en sorte que le cluster porte le même nom que la table.

```
Cluster %TABLE%
```

Pour plus d'informations sur la définition des options physiques, voir *Modélisation des données > Construction de modèles de données > Mise en oeuvre physique > Options physiques*.

Options physiques simples

Les options physiques simples doivent contenir un nom, et peuvent contenir un %d, un %s, ou une autre variable permettant à l'utilisateur de spécifier une valeur, ainsi que des mots clés pour spécifier les valeurs permises et valeurs par défaut.

Les options physiques simples sont spécifiées sur une seule ligne en utilisant la syntaxe suivante :

nom [=] %s|%d|%*variable*% [: *mots clés*]

Tout ce qui est saisi avant le signe deux points est généré dans les scripts. Le *nom* est requis par PowerAMC, mais vous pouvez le placer entre signes supérieur et inférieur (<*nom*>) si vous devez l'exclure du script final. Les variables %d ou %s requièrent une valeur numérique ou chaîne, et vous pouvez également utiliser une variable PowerAMC ou un snippet du langage de génération par templates.

Option physique	Générée sous la forme
max_rows_per_page=%d	max_rows_per_page=*valeur*
for instance %s	for instance *chaîne*
<Partition-name> %s	*nom*

Vous pouvez insérer un signe deux points suivi par des mots clés séparés par des virgules afin de contrôler vos option :

Mot clé	Valeur et résultat

`category=`*`métaclasse`*	Permet à l'utilisateur d'associer l'objet à un objet à un objet du type spécifié. Les paramètres suivants sont disponibles :
	• tablespace
	• storage
	Remarque : Dans Oracle, l'option physique composite `storage` est utilisée comme modèle pour définir toutes les valeurs de storage dans une entrée de storage afin de ne pas avoir à définir les valeurs une par une chaque fois que vous devez réutiliser les mêmes valeurs dans une clause de storage. Pour cette raison, l'option physique Oracle ne contient pas le nom de storage (%s).
	• *collection de métaclasses qualifiée* - Par exemple : `Model.Tables` ou `Table.Columns`
	`on %s : category=storage`
	`{`

`list=`*`valeur`*`|`*`valeur`*	Spécifie une liste de valeurs séparées par un trait vertical (|) permises pour l'option.
`default=`*`valeur`*	Spécifie une valeur par défaut pour l'option.
`dquoted=yes` et `squoted=yes`	Spécifie que la valeur est placée entre guillemets ou entre apostrophes.
`multiple=yes`	Spécifie que l'option est affichée avec un suffixe <*> dans le volet de gauche de l'onglet Options physiques et peut être ajoutée dans le volet de droite autant de fois que nécessaire. Si l'option est sélectionnée dans le volet de droite et que vous cliquez sur la même option dans le volet de gauche pour l'ajouter, un message vous demande si vous souhaitez réutiliser l'option sélectionnée. Si vous cliquez sur **Non**, une seconde instance de l'option est ajoutée dans le volet de droite.
`enabledbprefix=yes`	Spécifie que le nom de la base de données est inséré comme préfixe (voir les options de tablespace dans DB2 OS/390).
`prevmand=yes` et `nextmand=yes`	Spécifie que l'option physique précédente ou suivante est requise par l'option courante et que si l'option courante est ajoutée dans le volet droit, l'option suivante ou précédentes est également ajoutée.

Exemples

Option physique	Générée sous la forme
`ccsid %s : list=ascii|ebcdic|unicode, default=ascii`	`ccsid ascii`
`table=%s : category=Model.Tables, dquoted=yes`	`table="`*`table`*`"`
`<flashback_archive> %s`	*`string`*

Options physiques composites

Les options physiques composites sont spécifiées sur plusieurs lignes, et contiennent une ou plusieurs options dépendantes. Si vous ajoutez l'option composite dans la partie droite de l'onglet **Options physiques**, toutes les options dépendantes sont ajoutées en même temps. Si vous ajoutez une option dépendante, l'option composite est ajoutée également pour la contenir.

Les options physiques composites sont définies avec la syntaxe suivante :

```
nom [=] [%s|%d|%variable%] : composite=yes[, mots clés]
{
sous-option
[sous-option...]
}
```

Tout ce qui est saisi avant le signe deux points est généré dans les scripts. Le *nom* est requis par PowerAMC, mais vous pouvez le placer entre signes supérieur et inférieur (`<`*`nom`*`>`) si vous devez l'exclure du script final. Les variables `%d` ou `%s` requièrent une valeur numérique ou chaîne, et vous pouvez également utiliser une variable PowerAMC ou un snippet du langage de génération par templates.

Le mot clé `composite=yes` est requis pour les options composites, et peut être utilisé en conjonction avec n'importe quels mots clés d'options physiques ou n'importe quel des mots clés suivants :

Mot clé	Valeur et résultat
`composite=yes`	Spécifie que l'option est une option composite contenant des options dépendantes placées entre accolades.
`separator=yes`	Spécifie que les options dépendantes sont séparées par des virgules.
`parenthesis=yes`	Spécifie que tous les objets dépendants sont placés entre parenthèses.

Mot clé	Valeur et résultat
`chldmand=yes`	Spécifie qu'au moins une des options dépendantes doit être définie.

Exemples

Option physique	Générée sous la forme

`<list> : composite=yes, multiple=yes`
`{`
`<frag-expression> %s`
`in %s : category=storage`
`}`
	expression-fragmentation
	`in` *storage*
	
	expression-fragmentation2
	`in` *storage2*
	`etc`

`<using_block> : composite=yes,parenthesis=yes`
`{`
`using vcat %s`
`using stogroup %s : category=storage, composite=yes`
`{`
`priqty %d : default=12`
`secqty %d`
`erase %s : default=no, list=yes | no`
`}`
	`using vcat` *chaîne*
	`using stogroup` *storage*
	`priqty` *valeur*
	`secqty` *valeur*
	`erase no)`

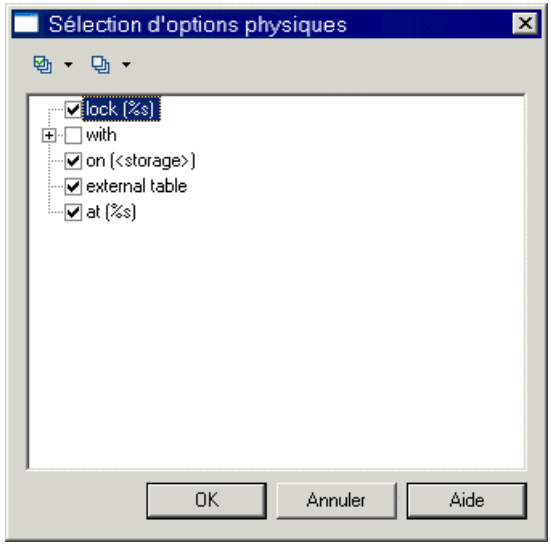
Ajout d'options physiques de SGBD dans vos formulaires

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 (SGBD)* à la page 224).

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 64).

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.

Variables et macros de MPD

Les requêtes SQL enregistrées dans les éléments du fichier de définition de SGBD utilisent diverses variables de MPD, qui sont écrites entre signes pourcent. 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.

Par exemple, dans la requête suivante, la variable `%TABLE%` sera remplacée par le code de la table créée :

```
CreateTable = create table %TABLE%
```

Remarque : Vous pouvez utiliser librement ces variables dans vos propres requêtes, mais vous ne pouvez pas changer la méthode de leur évaluation (c'est-à-dire que l'évaluation de

`%TABLE%` ne pourra produire que le code d'une table). Vous pouvez également accéder à n'importe quelle propriété d'objet en utilisant le GTL (voir *Chapitre 5, Personnalisation de la génération à l'aide du langage de génération par template* à la page 265) ainsi que les noms publics disponibles via le métamodèle public PowerAMC (voir *Chapitre 8, Métamodèle public PowerAMC* à la page 371).

L'évaluation des variables dépend des paramères et du contexte. Par exemple, la variable `%COLUMN%` ne peut pas être utilisées dans une requête `Create Tablespace`, car elle n'est valide que dans le contexte d'une colonne.

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. Exemple : (A <= 0) AND (A >= 10)
%CONSTDEFN%	Définition de contrainte de colonne. Exemple : constraint 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

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%=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.

Variable	Génération
`[.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 :

```
%[[?][-][x][.[-]y][options]:]variable%
```

Les options de format des variables sont les suivantes :

Option	Description
?	Champ obligatoire, si une valeur nulle est renvoyée, l'appel de conversion échoue

`[-][x].[-]y[M]`	Extrait les *y* premiers caractères ou, dans le cas de *-y*, les derniers *y* caractères.
	Si *x* est spécifié, et que *y* est inférieur à *x*, des blancs ou des zéros sont ajoutés à droite des caractères extraits pour atteindre *x* caractères. Dans le cas de *-x*, des blancs ou des zéros sont ajoutés à gauche et le résultat est justifié à droite.
	Si l'option `M` est ajoutée, les *x* premiers caractères de la variable sont supprimés et les *y* caractères suivants sont produits.
	Ainsi, pour un objet nommé `abcdefghijklmnopqrstuvwxyz` (les parenthèses sont présentes uniquement pour illustrer le remplissage) :
	`Template Résultat`
	`(%.3:Name%) gives (abc)`
	`(%.-3:Name%) gives (xyz)`
	`(%10.3:Name%) gives (abc )`
	`(%10.-3:Name%) gives (xyz )`
	`(%-10.3:Name%) gives ( abc)`
	`(%-10.-3:Name%) gives ( xyz)`
	`(%10.3M:Name%) gives (jkl)`

`L[F]`, `U[F]`, et `C`	Convertit le résultat en minuscules ou majuscules. Si `F` est spécifié, seul le premier caractère est converti. `C` équivaut à `UF`.
`q` et `Q`	Encadre la variable d'apostrophes ou de guillemets.
`T`	Supprime les espaces de début et de fin de la variable.
`H`	Convertit le nombre en valeur hexadécimale.

Vous pouvez combiner les codes de format. Par exemple, le template `(%12.3QMFU:Name %)` appliqué à l'objet `abcdefghijklmnopqrstuvwxyz` génère `("Lmn")`.

Variables pour les tables et les vues

PowerAMC peut utiliser des variables lors de la génération et du reverse-engineering des 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

Variable	Commentaire
%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 des colonnes, domaines et contraintes. Les variables pour les 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
%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

Variable	Commentaire
%NULLNOTNULL%	Statut obligatoire d'une colonne. Cette variable est systématiquement utilisée avec NullRequired, voir *Gestion des valeurs Null* à la page 172
%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
%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

Variable	Commentaire
%DEFAULT%	Valeur par défaut
%MINVAL%	Valeur minimum
%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
%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"

Nom de la variable	Commentaire
%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
%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
%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

Variable	Commentaire
%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 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

Variable	Commentaire
%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
%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 de 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
%SUPERADT%	Supertype du type de données abstrait
%ADTNOTFINAL%	Type de données abstrait final
%ADTABSTRACT%	Type de données abstrait instanciable

Variable	Commentaire
%ADTHEADER%	Corps du type de données abstrait avec ODBC
%ADTTEXT%	Spéc du type de données abstrait avec ODBC
%SUPERQUALI-FIER%	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
%ATTRJAVANA-ME%	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)
%PROCDEFN%	Corps de procédure (begin... end)
%PROCRETURN%	Type de résultat de procédure
%PARAM%	Paramètres de procédure

Variable	Commentaire
%PROCNOTFI-NAL%	Procédure finale
%PROCSTATIC%	Membre de procédure
%PROCABS-TRACT%	Procédure instanciable
%SUPERPROC%	Superprocédure de procédure
%ISCONSTRUC-TOR%	True si la procédure est un constructeur
%PROCJAVANA-ME%	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
%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 pour 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

Variable	Commentaire
%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

Variable	Commentaire
%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 services Web.

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%	Autorisation 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 dépendant de la dimension
%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 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

Nom de la variable	Commentaire
%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
%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

Nom de la variable	Commentaire
%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)

Macros .AKCOLN, .FKCOLN et .PKCOLN

Répète une instruction pour chaque colonne de clé alternative, étrangère ou primaire dans une table.

Syntax

```
.AKCOLN("instruction","préfixe","suffixe","dernier_suffixe",
"condition")
```

```
.FKCOLN("instruction","préfixe","suffixe","dernier_suffixe")
```

```
.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
last suffix	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` :

-
```
message .AKCOLN("'%COLUMN% est une colonne de clé
alternative'","", "", "", "AKEY1")
```

  génère le script de trigger suivant :

```
message 'COMMANDE_ECRIT est une colonne de clé alternative',
```

-
```
message .FKCOLN("'%COLUMN% est une colonne de clé
étrangère'","","","","";")
```

  génère le script de trigger suivant :

```
message 'ID_AUTEUR est une colonne de clé étrangère,
ISBN_TITRE est une colonne de clé étrangère;'
```

- `message .PKCOLN("'%COLUMN% est une colonne de clé primaire'","","","","";")`

  génère le script de trigger suivant :

```
message 'ID_AU est une colonne de clé primaire',
    'ISBN_TITRE est une colonne de clé primaire';
```

Remarque : Pour les colonnes, ces macros n'acceptent que la variable `%COLUMN%`.

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 :

```
ID_AUTEUR char(12),
NOM_AUTEUR varchar(40),
PRENOM_AUTEUR varchar(40),
BIO_AUTEUR long varchar,
AVANCE_AUTEUR numeric(8,2),
ADRESSE_AUTEUR varchar(80),
VILLE varchar(20),
ETAT char(2),
CODEPOSTAL char(5),
TEL_AUTEUR 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 'Erreur : Trigger(%TRIGGER%) de la table %TABLE%'
```

Génère le script de trigger qui suit :

```
message 'Erreur : Trigger(T_AUTEUR) de la table AUTEUR';
```

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, "Le parent n'existe pas, impossible d'insérer
l'enfant")
```

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")
--  Impossible de supprimer le parent "%PARENT%" si "%CHILD%"
contient encore un enfant
.ENDFOR
```

Génère le script de trigger qui suit :

```
--  Impossible de supprimer le parent "TITRE" si "DROITS" contient
encore un enfant
```

```
--  Impossible de supprimer le parent "TITRE" si "VENTE" contient
encore un enfant
--  Impossible de supprimer le parent "TITRE" si "ECRIT" contient
encore un enfant
```

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%" ne peut pas être modifié
.ENDFOR
```

Génère le script de trigger qui suit :

```
-- "ISBN_TITRE" ne peut pas être modifié
-- "ID_EDITEUR" ne peut pas être modifié
```

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
FKNOTNULL	Clés étrangères obligatoires
FKCANTCHG	Clés étrangères non-modifiables

Exemple

Dans un trigger pour la table VENTE, la macro suivante :

```
.FOREACH_PARENT("FKCANTCHG")
--  Impossible de changer le code parent de "%PARENT%" dans l'enfant
"%CHILD%"
.ENDFOR
```

Génère le script de trigger qui suit :

```
--  Impossible de changer le code parent de "MAGASIN" dans l'enfant
"VENTE"
--  Impossible de changer le code parent de "TITRE" dans l'enfant
"VENTE"
```

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 'La référence %REFR% lie la table %PARENT% à %CHILD%'
 .ENDFOR
```

Génère le script de trigger qui suit :

```
message 'La référence EDITEUR_TITRE lie la table EDITEUR à TITRE
```

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);
```

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 :

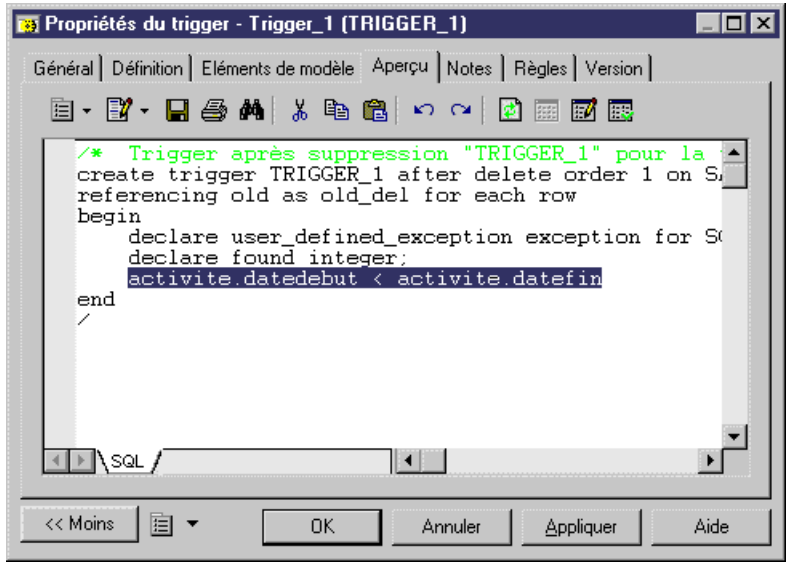
```
activite.datedebut < activite.datefin
```

Dans un trigger basé sur le modèle de trigger AfterDeleteTrigger, vous saisissez la macro suivante dans l'onglet Définition du trigger :

```
.SERVEREXPRESSION(CONTROLE_DATE_ACTIVITE)
```

Vous générez ainsi le script de trigger suivant :

```
activite.datedebut < activite.datefin
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.

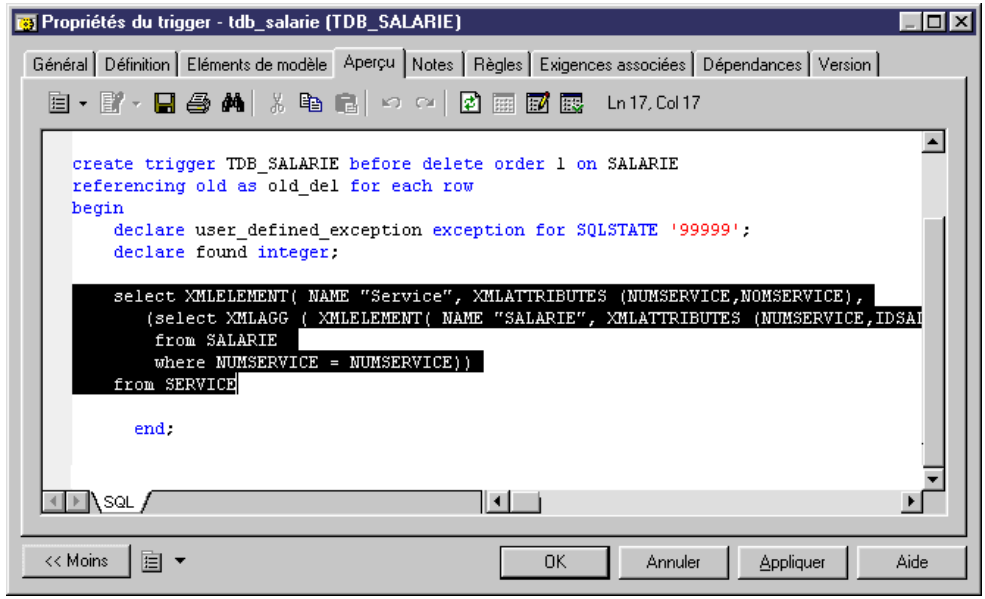
Exemple

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
```



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) est utilisé pour extraire des propriétés d'objet de modèle sous forme de texte. Le GTL est écrit sous forme de *templates* et de *fichiers générés* définis sous des métataclasses dans des fichiers de définition de langage ou d'extension. Le GTL alimente la génération de code pour les langages de processus métiers, orientés objet et XML, et peut être utilisé afin de définir de nouvelles génération pour tout type de modèle.

Lorsque vous lancez une génération à partir d'un modèle, PowerAMC génère un fichier pour chaque instance de la métaclasse pour laquelle vous avez défini un fichier généré (voir *Fichiers générés (Profile)* à la page 95) en évaluant le templates qu'il appelle et en résolvant les éventuelles 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, il fournit des macros permettant de tester les variables et de procéder à l'itération des 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

Remarque : Bien que le GTL puisse être utilisé pour étendre la génération dans un MPD, la génération standard est principalement définie à l'aide d'un mécanisme différent (voir *Génération et reverse engineering de base de données* à la page 131).

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. Ouvrez votre fichier de définition de langage ou d'extension dans l'Editeur de ressources (voir *Ouverture de fichiers de ressources dans l'Editeur de ressources* à la page 2).

2. Si nécessaire, ajoutez une métaclasse dans la catégorie Profile (voir *Métaclasses (Profile)* à la page 35), puis pointez sur cette dernière, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau > Template** (voir *Templates (Profile)* à la page 94).
3. Saisissez `salutMonde` comme nom du template et saisissez le code suivant dans la zone de texte :

```
Salut le monde !
Ce template est généré pour l'objet %Name%.
```

Remarque : 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 propriétés et de collections qui, par convention, utilisent la présentation full CamelCase.

4. Pointez à nouveau sur la métaclasse, cliquez le bouton droit de la souris, puis sélectionnez **Nouveau > Fichier généré** (voir *Fichiers générés (Profile)* à la page 95).
5. Saisissez `monFichier` comme nom pour le fichier généré, puis saisissez le code suivant dans la zone de texte pour appeler votre template :

```
%salutMonde%
```

6. Cliquez sur **OK** pour enregistrer les modifications dans le fichier de ressources et revenir à votre modèle.
7. Créez une instance de la métaclasse sur laquelle vous avez défini le template et le fichier généré, ouvrez sa feuille de propriétés, puis cliquez sur l'onglet **Aperçu**.
8. Sélectionnez le sous-onglet **monFichier** pour afficher un aperçu qui serait généré pour cet objet.

Extraction des propriétés d'objet

Les propriétés d'objet sont référencées sous la forme de variables et encadrées de signe pourcent : `%variable%`. Les noms de variable sont sensibles à la casse, et les noms de propriétés sont, par convention, définis en casse CamelCase (mots collés, avec une majuscule au début de chaque mot).

Les propriétés sont extraites sous la forme des types suivants :

- String - renvoie du texte .
- Boolean - renvoie `true` ou `false`.
- Object - renvoie l'D de l'objet ou `null`.

Exemple

```
Ce fichier est généré pour %Name% sous forme de %Shape% %Color%
```

Résultat :

```
Ce fichier est généré pour MonObjet sous forme de Triangle Rouge.
```

Les propriétés standard définies dans le métamodèle public PowerAMC (voir *Chapitre 8, Métamodèle public PowerAMC* à la page 371) sont référencées en utilisant leur nom public, qui est écrit au format CamelCase. Vous pouvez parfois déduire les noms publics des propriétés à partir de leur libellé dans les feuilles de propriétés, mais en cas de doute, cliquez sur le bouton **Menu de la feuille de propriétés** en bas de la feuille de propriétés et sélectionnez **Rechercher dans l'aide sur les objets du métamodèle** afin d'afficher toutes les propriétés disponibles pour l'objet.

Les attributs étendus (voir *Attributs étendus (Profile)* à la page 45) sont référencés par leur **Nom** défini dans l'Editeur de ressources.

Remarque : Pour accéder à un attribut étendu défini dans un autre fichier d'extension attaché au modèle, préfixez le nom à l'aide de l'option de format `.D`. Par exemple :

```
%.D:MonAttribEtendu%
```

Accès aux collections de sous-objets ou d'objets associés

Un MOO contient une collection de classe et des classes contiennent des collections d'attributs et d'opérations. Pour procéder à l'itération d'une collection, utilisez la macro `.foreach_item`.

Exemple

```
%Name% contient les widgets suivants :
.foreach_item(Widgets)
    \n\t%Name% (%Color% %Shape%)
.next
```

Résultat :

```
MonObjet contient les widgets suivants :
    Widget1 (triangle rouge)
    Widget2 (carré jaune)
    Widget3 (cercle vert)
```

Les collections standard définies dans le métamodèle public PowerAMC (voir *Chapitre 8, Métamodèle public PowerAMC* à la page 371) référencées en utilisant leur nom public, qui est écrit au format CamelCase. Vous pouvez parfois déduire les noms publics des collections à partir de leur libellé dans les feuilles de propriétés, mais en cas de doute, cliquez sur le bouton **Menu de la feuille de propriétés** en bas de la feuille de propriétés et sélectionnez **Rechercher dans l'aide sur les objets du métamodèle** afin d'afficher toutes les collections disponibles pour l'objet.

Les collections étendues (voir *Collections et compositions étendues (Profile)* à la page 54 et *Collections calculées (Profile)* à la page 57) sont référencées par leur propriété **Nom**.

Vous pouvez utiliser les mots clés suivants pour accéder aux informations sur une collection :

Nom	Description
`First`	(object) Renvoie le premier élément de la collection.
`IsEmpty`	(boolean) Renvoie `True` si la collection est vide, `false` si elle contient au moins un membre.
`Count`	(integer) Renvoie le nombre d'éléments de la collection. Vous pouvez utiliser ce mot clé pour définir un critère basé sur la taille de la collection, par exemple `Attributes.Count>=10`.

Exemple

```
%Name% est associée à %AttachedRules.Count% règles de gestion,
 dont la première est %AttachedRules.First.Name%.
```

Résultat :

```
maClasse est associée à 3 règles de gestion,
 dont la première est maRègle.
```

Mise en forme de votre résultat

Vous pouvez changer le format des variables en incorporant des options de mise en forme dans la syntaxe des variables. Les nouvelles lignes et les tabulations sont spécifiées respectivement à l'aide des séquence d'échappement `\n` et `\t`.

```
%[[-][x][.[-]y][options]:]variable%
```

Les options de format pour les variables sont les suivantes :

Option	Description

`[-][x].[-]y[M]`	Extrait les *y* premiers caractères ou, dans le cas de *-y*, les *y* derniers caractères.
	Si vous avez spécifié *x*, et si *y* est inférieur à *x*, des espaces ou des zéros sont ajoutés à la droite des caractères extraits afin d'atteindre les *x* caractères. Dans le cas de *-x*, des espaces ou des zéros sont ajoutés à la gauche et le résultat est justifié à droite.
	Si l'option M est ajoutée, les premiers *x* caractères de la variable sont supprimés et les *y* caractères suivants sont produits.
	Par conséquent, pour un objet nommé `abcdefghijklmnopqrstuvwxyz` (les parenthèses montrent simplement le remplissage) :
	`Template Résultat`
	`(%.3:Name%) gives (abc)`
	`(%.-3:Name%) gives (xyz)`
	`(%10.3:Name%) gives (abc )`
	`(%10.-3:Name%) gives (xyz )`
	`(%-10.3:Name%) gives ( abc)`
	`(%-10.-3:Name%) gives ( xyz)`
	`(%10.3M:Name%) gives (jkl)`

`L[F]`, `U[F]` et `C`	Convertit le résultat en minuscules ou majuscules. Si `F` est spécifié, seul le premier caractère est converti. `C` équivaut à `UF`.
`q` et `Q`	Encadre la variable d'apostrophes ou des guillemets.
`A`	Supprimer le retrait et aligne le texte sur le bord gauche.
`T`	Supprime les espaces de début et de fin de la variable.
`H`	Convertit le nombre en valeur hexadécimale.
`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. Le template `%Visibility%` est généré sous la forme `+`, mais `%.D:Visibility%` est généré sous la forme `public`. **Remarque :** Vous pouvez accéder aux attributs étendus définis dans un autre fichier d'extension en les préfixant avec l'option `.D` (voir *Extraction des propriétés d'objet* à la page 266).
`X`	Ignore les caractères interdits pour XML.

Option	Description
E	[abandonné – utilisez plutôt l'opérateur d'évaluation !, voir *Opérateurs du langage de génération par template* à la page 273].

Exemples

```
Ce fichier est généré pour %.UQ:Name%. Il a une forme de %.L:Shape%
    %.L:Color%.
```

```
Ce fichier est généré pour "MONGADGET". Il a une forme de triangle
rouge.
```

Le template suivant est appliqué à l'objet `abcdefghijklmnopqrstuvwxyz`

```
%12.3QMFU:Name%
```

Résultat :

```
"Lmn"
```

Contrôle des passages à la ligne dans les chaînes d'en-tête et de fin

Les chaînes d'en-tête et de fin dans un bloc de macro ne sont générés que lorsque nécessaire. Si le bloc ne renvoie rien, les chaînes d'en-tête et de fin n'apparaissent pas, ce qui peut aider à contrôler la création de nouvelles lignes.

Exemple

Le texte et les nouvelles ligne dans l'en-tête et la fin de chaque boucle `.foreach_item` ne sont imprimés que si la collection n'est pas vide. Lorsque ce template est appliqué à une classe avec des attributs mais sans opération, le texte `// Opérations` et les nouvelles lignes spécifiées avant et après la liste des opérations ne sont pas imprimés :

```
class "%Code%" {
 .foreach_item(Attributes, // Attributes\n,\n\n)
 %DataType% %Code%
  .if (%InitialValue%)
 = %InitialValue%
  .endif
 .next(\n)
 .foreach_item(Operations, // Operations\n,\n\n)
 %ReturnType% %Code%(...)
 .next(\n)
<Source>
}
```

Résultat :

```
class "C1" {// Attributes
 int a1 = 10
 int a2
 int a3 = 5
 int a4

<Source>
}
```

Remarque : Pour imprimer un espace vide entre l'accolade et la chaîne `// Attributs`, vous devez encadrer la chaîne d'en-tête de guillemets :

```
.foreach_item(Attributes," // Attributes\n",\n)
```

Blocs conditionnels

Placez entre crochets du texte contenant une variable afin de faire en sorte qu'il apparaisse uniquement si la variable est résolue en une valeur non-nulle.

Vous pouvez utiliser une forme similaire à C et aux expressions ternaires Java afin de faire apparaître une chaîne uniquement si la variable est True ou non- nulle :

```
[variable ? ifNotNull]
```

Vous pouvez également inclure une chaîne à afficher si la variable est évaluée à False, à Null ou à la chaîne vide :

```
[variable ? ifNotNull :ifNull]
```

Exemples

```
Attribute %Code%[ = %InitialValue%];
```

Résultat :

```
Attribute A1 =0;
Attribute A2 =100;
Attribute A3;
Attribute A4 =10;
```

```
La classe %Name% est [%Abstract%?Abstract:Concrète].
```

Résultat si la propriété **Abstraite** est sélectionnée :

```
The class myClass is Abstract.
```

Résultat si la propriété **Abstraite** n'est pas sélectionnée :

```
La classe myClass est Concrète.
```

Accès aux variables globales

Vous pouvez insérer des informations telles que votre nom d'utilisateur et la date courante à l'aide de variables globales.

Nom	Description
`%ActiveModel%`	(object) Renvoie l'UID du modèle. Utilisez `%ActiveModel.Name%` pour obtenir le nom du modèle.
`%GenOptions%`	(struct) Renvoie les options de génération de modèle.
`%PreviewMode%`	(boolean) Renvoie `true` dans l'onglet **Aperçu**, `false` lorsque généré dans un fichier.
`%CurrentDate%`	(string) Renvoie la date et l'heure système courantes mises en forme en fonction des paramètres locaux.
`%CurrentUser%`	(string) Renvoie l'ID de l'utilisateur courant.
`%NewUUID%`	(string) Renvoie un nouvel identifiant utilisateur universellement unique.

Exemple

```
Ce fichier a été généré depuis %ActiveModel.Name% par %CurrentUser%
le
    %CurrentDate%.
```

Result:

```
Ce fichier a été généré depuis Mon modèle par jsmith le mardi 6
novembre
    2012 14:06:41.
```

Opérateurs du langage de génération par template

Le langage de génération par template prend en charge des opérateurs arithmétiques et logiques 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
=	Opérateur d'affectation.
== et !=	Opérateurs égal à et différent de.
> et <	Opérateurs supérieur à et inférieur à.
>= et <=	Opérateurs supérieur ou égal à et inférieur ou égal à.
&& et ||	Opérateurs logiques AND et OR.
%+(*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 - Correspond à une double évaluation, qui renvoie un template au lieu d'un texte, en utilisant la syntaxe suivante :
	`%*template [(P1,P2...)]%`
	Pour plus d'informations sur les paramètres des templates, voir *Passage de paramètres à un template* à la page 281.
	Dans l'exemple suivant, une variable locale est renvoyée normalement et sous une forme déréférencée :
	`.set_value(C, Code)`
	`%C%`
	`%*C%`
	Résultat :
	`Code`
	`%Code%`

!	Opérateur d'évaluation - Evalue le résultat de l'évaluation de la variable comme un template.
	Dans l'exemple suivant, une variable locale est renvoyée normalement et sous une forme évaluée :
	`.set_value(C, %%MonAttribut%%)`
	`%C%`
	`%!C%`
	Résultat :
	`%MonAttribut%`
	`Red`
	L'opérateur ! peut être utilisé plusieurs fois. Par exemple :
	`%!!t%`
	Vous obtenez ainsi les résultats de l'évaluation de l'évaluation du template `t`.

Opérateur	Description

?	Opérateur d'existence - Teste si un template, une variable locale ou une propriété est ou non présent, et renvoie `false` si tel n'est pas le cas.
	
	Par exemple :
	`.set_value (maVariable, 20, new)`
	`%myVariable?%`
	`.unset (maVariable)`
	`%myVariable?%`
	
	Résultat :
	`true`
	`false`

+	Opérateur de visibilité - Teste si une propriété d'objet est visible dans l'interface, et renvoie `false` si telle n'est pas le cas. Par exemple, pour tester si le champ **Type** est affiché dans l'onglet **Général** d'une feuille de propriétés de base de données dans un MFI (ce qui signifie qu'un fichier d'extension Replication Server® est attaché au modèle), saisissez le code suivant : `%Database.Type+%`

Portée de la conversion

La portée initiale d'un template est toujours la métaclasse sur laquelle le template est défini. Tous les attributs standard et étendus, toutes le collections et les templates définis sur la métaclasse d'objet active et ses parents sont visibles, mais un seul objet est actif à la fois.

Exemples

Le template suivant est appliqué au package `P1`, qui contient une classe `C1`, qui contient les opérations `O1` et `O2`, qui chacune contiennent les paramètres `P1` et `P2`. La portée change, affectant la valeur de la variable `%Name%`, à mesure que chaque collection est parcourue. Le mot clé `Outer` est utilisé pour revenir temporairement aux portées précédentes :

```
%Name%
.foreach_item(Classes)
    \n\t*%Name% in %Outer.Name%
   .foreach_item(Operations)
       \n\t*%Name% in %Outer.Name% in %Outer.Outer.Name%
      .foreach_item(Parameters)
          \n\t\t*%Name% in %Outer.Name% in %Outer.Outer.Name% in
%Outer.Outer.Outer.Name%
      .next
   .next
.next
```

Résultat :

```
P1
    *C1 in P1
    *O1 in C1 in P1
        *P1 in O1 in C1 in P1
        *P2 in O1 in C1 in P1
    *O2 in C1 in P1
        *P1 in O2 in C1 in P1
        *P2 in O2 in C1 in P1
```

La portée `Outer` est restaurée lorsque vous quittez un bloc `.foreach_item`. Les portées imbriquées forment une hiérarchie qui peut être visualisée sous la forme d'une arborescence, le plus haut niveau étant la racine. Utilisez `Parent` plutôt que `Outer` pour revenir au niveau de la portée de l'objet d'origine. Par exemple, rien ne sera produit si le template suivant est appliqué au paramètre `P1`:

```
%Name% in %Outer.Name% in %Outer.Outer.Name%
```

Toutefois, ce template produira un résultat :

```
%Name% in %Parent.Name% in %Parent.Parent.Name%
```

Résultat :

```
P1 in O1 in C1
```

Conversion d'un raccourci

Les raccourcis sont déréférencés lors de la conversion : la portée de l'objet cible remplace la portée du raccourci.. Ce comportement est différent de celui de VB Script, dans lequel la conversion du raccourci récupère le raccourci lui-même. Vous pouvez utiliser la variable `%IsShortcut%` pour tester si un objet est un raccourci, puis le mot clé `Shortcut` pour accéder aux propriétés du raccourci lui-même.

Template

Dans cet exemple, le template est appliqué à un package `P1` de MOO qui contient deux classes et deux raccourcis vers des classes situées dans `P2` :

```
.foreach_item(Classes)
\n*Classe %Code% [%IsShortcut% ? située dans le package %Packa-
ge.Name% : objet local]
.next
```

Résultat :

```
*Classe C1  objet local
*Classe C2  objet local
*Classe C3  située dans le package P2
*Classe C4  située dans le package P2
```

Remarque : Si votre modèle contient des raccourcis vers des objets contenus dans un autre modèle qui n'est pas ouvert, une boîte de dialogue vous invite à ouvrir le modèle cible. Vous pouvez utiliser la macro `.set_interactive_mode` pour changer ce comportement (voir *Macro .set_interactive_mode* à la page 300).

Séquences d'échappement

Le langage de génération par template prend en charge différentes séquences d'échappement permettant de simplifier la présentation de vos templates et fichiers générés, et de rendre accessibles les caractères réservés.

Les séquences d'échappement suivantes peuvent être utilisées dans des templates :

Séquence d'échappement	Description
\n	Passage à la ligne. Pour des exemples de passage à la ligne dans des blocs de macros, voir *Contrôle des passages à la ligne dans les chaînes d'en-tête et de fin* à la page 271.
\t	Tabulation

Séquence d'échappement	Description
\\	Barre oblique inverse
\ au début d'une ligne	Caractère de suite (ignore la nouvelle ligne)
. au début d'une ligne	Commentaire. Ignore la ligne.
.. au début d'une ligne	Caractère point (pour générer une macro).
%%	Caractère pourcent.

Appel de templates

Vous pouvez appeler un template depuis un fichier généré ou depuis un autre template en saisissant son nom entre signes pourcent. Les propriétés et collections d'objet, et les variables locales et globales sont appelées de la même manière. Au moment de la génération, un appel de template est remplacé pour le contenu du template, qui est ensuite résolu dans sa valeur textuelle finale.

Exemples :

- `%Name%` - Appelle la propriété **Nom** de l'objet
- `%monTemplate%` - Appelle le template `%monTemplate%`
- `%CurrentDate%` - Appelle la variable globale `%CurrentDate%` (voir *Accès aux variables globales* à la page 272)

En décomposant les templates en unités concises et en les appelant au moment de la génération, vous améliorez leur lisibilité et les possibilités de réutilisation. Par exemple, vous pouvez définir une condition commune dans un template et y faire référence dans plusieurs autres templates :

Exemple

Le template `%isInner%` est défini comme :

```
.bool (%ContainerClassifier%!=null)
```

Le template `%QualifiedCode%` appelle le template `%isInner%` pour tester si la classe est une classe interne :

```
.if (%isInner%)
    %ContainerClassifier.QualifiedCode%::%Code%
.else
    %Code%
.endif
```

Résultat :

```
C2::C1
```

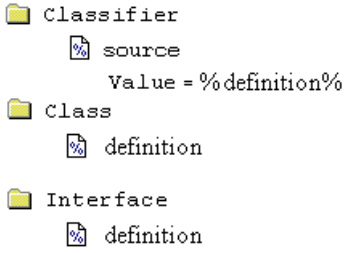
Le template `%QualifiedCode%` est appliqué à la classe `C1` qui est une classe interne pour `C2`.

Héritage et polymorphisme

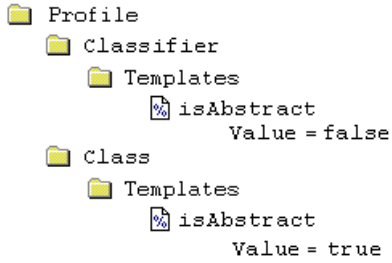
Les templates sont définis sur une métaclasse particulière dans un fichier de définition de langage ou dans une extension et sont hérités par et disponibles pour les enfants de cette métaclasse. Par exemple, un template défini sur la métaclasse Classifier est disponible pour les templates ou fichiers générés définis sur les métaclasses Class et Interface.

GTL prend en charge les concepts orientés objet suivants comme faisant partie de l'héritage :

- *Polymorphisme* - Le choix du template à évaluer est effectué au moment de la conversion. Un template défini sur un classificateur peut accéder aux templates définis sur ses enfants (classe, interface). Dans l'exemple suivante, le contenu de `%definition%` dépend de l'objet (classe ou interface) en cours de traitement :

- *Redéfinition de template* - Un template défini sur une métaclasse particulière peut être redéfini par un template de même nom défini sur une classe enfant. Dans l'exemple suivant, le template défini sur la métaclasse Classifier est redéfini par celui défini sur la métaclasse Class :



Vous pouvez afficher le parent redéfini en pointant sur le template enfant, cliquant le bouton droit de la souris, puis sélectionnant **Afficher la super-définition**. Vous pouvez spécifier l'utilisation du template parent en préfixant l'appel de template de l'opérateur qualifiant `::`. Par exemple : `%Classifier::isAbstract%`.

- *Surcharge de template* - Vous pouvez surcharger vos définitions de templates et tester pour différentes conditions. Les templates peuvent également être définis sous des critères (voir *Critères (Profile)* à la page 43) ou des stéréotypes (voir *Stéréotypes (Profile)* à la page 40), et les conditions correspondantes sont combinées. Au moment 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>
```

Vous pouvez définir le même template plusieurs fois dans la hiérarchie de fichiers de définition de langage ou d'extension, et PowerAMC le résoudra en utilisant des règles d'héritage. Par exemple, le fichier de définitions de langage de MOO `monLang` et le fichier d'extension `monExtension` contiennent chacun un template `%t%` défini sur chacune des métaclasses `Classifier` et `Class` :

Fichier de définition de langage monLang	Fichier d'extension monExtension

• Classifier :
  • fichier généré `monFichier`
  • template `%t%` template
• Class :
  • `%t%` template
	• Classifier :
	  • fichier généré `monAutreFichier`
	  • template `%t%`
	• Class :
	  • `%t%` template

Les métaclasses `Class` et `Interface` héritent toutes deux de la métaclasse `Classifier`, et génèrent chacune un `monFichier` et un `monAutreFichier`.

Les appels de template suivants sont possibles dans `monLang/Classifier/monFichier` (qui ne peut pas accéder aux templates contenus dans `monExtension`) :

Appel de template dans monFichier	Template appelé
`%t%` or `%monLang::t%`	`monLang/Class/t`
`%Classifier::t%` or `%monLang::Classifier::t%`	`monLang/Classifier/t`

Les appels de template suivants sont possibles dans `myExtension/Classifier/monAutreFichier` (qui peut accéder à la fois à ses propres templates et à ceux contenus dans `monLang`) :

Appel de template dans monAutreFichier	Template appelé
`%t%` or `%myExtension::t%`	`myExtension/Class/t`
`%Classifier::t%` or `%myExtension::Classifier::t%`	`myExtension/Classifier/t`
`%monLang::t%` or `%monLang::Class::t%`	`monLang/Class/t`
`%monLang::Classifier::t%`	`monLang/Classifier/t`

Remarque : Pour qu'un fichier d'extension puisse atteindre des templates définis dans un fichier de définition de langage, la propriété **Compléter la génération de langage** doit être sélectionnée dans l'extension (voir *Propriétés d'un fichier d'extension* à la page 15).

Passage de paramètres à un template

Vous pouvez passer des paramètres à un template, en utilisant la syntaxe suivante : `%t(p1,p2...)%`.

Les valeurs de paramètre ne peuvent pas contenir de caractère `%` (vous ne pouvez pas passer un template), et sont séparés par des virgules. Ils sont récupérés dans le template par le biais de variables locales avec les noms `@1, @2, ...`.

Exemples

L'appel de template suivant :

```
%monTemplate(beau,ensoleillé,24,12)%
```

appelle `%monTemplate%`:

```
Le temps aujourd'hui est %@1% et %@2%, avec des températures entre
%@3% et %@4%.
```

Résultat :

```
Le temps aujourd'hui est beau et ensoleillé, avec des températures
entre 12 et 24.
```

Le template `%Attributes%` est défini comme suit :

```
.foreach_item(Attributes)
 .if (%Visibility% == %@1%)
 %DataType% %Code%
 .endif
.next(\n)
```

Le template `%AttributeList%` appelle `%Attributes%` trois fois, en passant une valeur de visibilité différente chaque fois pour boucler uniquement sur les attributs qui ont cette visibilité :

```
Attributes de la classe "%Code%" :
// Public
%attributes(+)%

// Protected
%attributes(#)%

// Private
%attributes(-)%
```

Result:

```
Attributs de la classe "C1" :
// Public
 int height
 int width

// Protected
 int shape

// Private
 int cost
 int price
```

Templates récursifs

Un template peut s'appeler lui-même, mais un tel template doit contenir des critères ou un changement de portée pour éviter de créer une boucle infinie.

Exemple

La classe `C1` est interne à la classe `C2`, qui à son tour est interne à `C3`. Le template `%topContainerCode%` teste si le classificateur courant est interne à un autre, et si tel est le cas, il s'appelle lui-même sur le classificateur conteneur pour effectuer le même test, et ainsi de suite jusqu'à ce qu'il atteigne un classificateur qui n'est pas interne, et récupère alors le code de ce conteneur racine :

```
.if (%isInner%)
    %ContainerClassifier.topContainerCode%
.else
    %Code%
.endif
```

Résultat :

```
C3
```

Extensions du métamodèle spécifiques au langage de génération par template

Un grand nombre d'attributs et de collections calculés sont fournis sous la forme d'extensions du métamodèle spécifiques au langage de génération par template.

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

Métaclasse	Attributs spécifiques au GTL
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)

Métaclasse	Attributs spécifiques au GTL
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
PdOOM.Interface	• @<tag> [spécifique Java] (string) - Attribut étendu Javadoc@<tag> avec formatage supplémentaire
PdOOM.Operation	• DeclaringInterface (object) • GetSetAttribute (object) • Overridden (boolean) • ReturnTypeModifierPrefix (string) • ReturnTypeModifierSuffix (string) • @<tag> [spécifique Java] (string) - Attribut étendu Javadoc@<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 GTL :

Nom de métaclasse	Nom de collection
PdCommon.BaseModel	Generated <nom-métaclasse> List - Collection de tous les objets du type <nom-métaclasse> qui font partie de la sélection dans la boîte de dialogue de génération

Nom de métaclasse	Nom de collection
PdCommon. BaseClassifier-Mapping	SourceLinks
PdCommon. BaseAssociation-Mapping	SourceLinks

Guide de référence des macros du langage de génération par template

Le langage de génération par template (GTL) prend en charge des macros pour exprimer la logique de template, et pour boucler sur les 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.

Remarque : Les paramètres de macro peuvent être délimités par des guillemets, ce qui est obligatoire lorsque la valeur d'un 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 \". Lorsque les paramètres de macro spécifient qu'un paramètre est de type *template simple*, ce dernier peut contenir du texte, des variables et des bloc conditionnels, mais pas des macros. Les paramètres de type *template complexe* peuvent également contenir des macros.

Les macros suivantes sont disponibles :

- *Macros conditionnelles et macro de boucle/itératives* :
  - *Macro .if* à la page 297 - évalue des conditions.
  - *Macro .foreach_item* à la page 292 – permet l'itération sur les collections d'objets.
  - *Macro .foreach_line* à la page 294 – permet l'itération sur les lignes d'un bloc de texte multiligne.
  - *Macro .foreach_part* à la page 295 – permet l'itération sur les parties d'une chaîne.
  - *Macro .break* à la page 287 – interrompt une boucle.
- *Macro de mise en forme et de manipulation de chaîne* :
  - *Macros .lowercase et .uppercase* à la page 299 - change la casse d'un bloc de texte.
  - *Macros .convert_name et .convert_code* à la page 288 - convertit des codes en noms ou des noms en codes.
  - *Macros .delete et .replace* à la page 289 - effectue des opérations sur des sous-chaînes.
  - *Macro .unique* à la page 303 - filtre les lignes redondantes dans un bloc de texte.
  - *Macro .block* à la page 286 - ajoute un en-tête et une fin à un bloc de texte.
- *Macros de commande de génération* - à utiliser lorsque vous rédigez du GTL dans le contexte de l'exécution d'une commande de génération :
  - *Macro .vbscript* à la page 303 - incorpore du code de script VB dans un template.
  - *Macro .execute_vbscript* à la page 291 - lance des scripts VB.

- *Macro .execute_command* à la page 291 - lance des exécutables.
- *Macro .abort_command* à la page 286 - interrompt l'exécution d'une commande.
- *Macros .change_dir et .create_path* à la page 288 - change de répertoire ou créer un chemin.
- *Macro .log* à la page 298 - écrit des messages de consignation.

- *Macros diverses* :
  - *Macros .set_object, .set_value et .unset* à la page 300 - créer des objets ou variables locaux.
  - *Macros .comment et .//* à la page 288 - insère un commentaire dans un template.
  - *Macros .object et .collection* à la page 299 - renvoie une collection d'objets en fonction de la portée et de la condition spécifiées.
  - *Macros .object et .collection* à la page 299 - renvoie une objet ou une collection en fonction de la portée et de la condition spécifiées.
  - *Macro .bool* à la page 287 - évalue une condition.
  - *Macro .set_interactive_mode* à la page 300 – spécifie si l'exécution du GTL doit interagir avec l'utilisateur.
  - *Macros .error et .warning* à la page 290

Macro .abort_command

Cette commande stoppe l'exécution d'une commande de génération.

Exemple

```
.if %_JAVAC%
  .execute_command (%_JAVAC%,%FileName%)
.else
  .abort_command
.endif
```

Pour plus d'informations sur les commandes de génération, voir *Catégorie Generation* à la page 122.

Macro .block

Cette macro ajoute un en-tête et/ou une fin à un bloc de résultats, si le résultat n'est pas vide.

```
.block [(en-tête)]
     bloc-entrée
.endblock[(fin)]
```

Les paramètres suivants sont disponibles :

Paramètre	Description
en-tête	[facutlatif] Généré uniquement si **bloc-entrée** n'est pas vide. Type : Template simple

Paramètre	Description
bloc-entrée	Spécifie le texte à produire entre l'en-tête et la fin. Type : Template complexe
fin	[facutlatif] Généré uniquement si **bloc-entrée** n'est pas vide. Type: Template simple

Exemple	Résultat

`.block (<b>)`
`%Comment%`
`.endblock (</b>)`
	`<b>Mon commentaire est en gras !<b>`
	Remarque : Les balises <b> ne seront pas générées si aucun commentaire n'est saisi pour un objet particulier.

Macro .bool

Cette macro renvoie `true` ou `false` en fonction de la valeur de la condition spécifiée.

`.bool (`**condition**`)`

Les paramètres suivants sont disponibles :

Paramètre	Description
condition	Spécifie la condition à évaluer. Type : Condition

Exemple	Résultat
`.bool(%.3:Code%= =ejb)`	`true`

Macro .break

Cette macro peut être utilisée pour quitter des boucles `.foreach`.

Exemple

```
.set_value(_hasMain, false, new)
.foreach_item(Operations)
 .if (%Code% == main)
  .set_value(_hasMain, true)
  .break
 .endif
.next
%_hasMain%
```

Macros .change_dir et .create_path

Ces macros changent le répertoire courant ou créent le chemin spécifié dans le cadre d'une commande de génération.

```
.change_dir (chemin)
```

```
.create_path (chemin)
```

Les paramètres suivants sont disponibles :

Paramètre	Description
chemin	Spécifie le répertoire à activer ou à créer. Type : Template simple (séquences d'échappement ignorées)

Exemple	Résultat
`.change_dir(C:\temp)`	Change le chemin pour écrire dans `C:\temp`.
`.create_path(C:\temp\monrep)`	Crée le nouveau répertoire `C:\temp\monrep`.

Pour plus d'informations sur les commandes de génération, voir *Catégorie Generation* à la page 122.

Macros .comment et .//

Ces macros sont utilisées pour insérer des commentaires dans un template. Les lignes qui commencent par `.//` ou par `.comment` sont ignorées lors de la génération.

Exemple
`.// Ceci est un commentaire` `.comment Ceci aussi est un commentaire`

Macros .convert_name et .convert_code

Ces macros convertissent le nom d'un objet en son code (et vice versa).

Utilisez la syntaxe suivante pour convertir un nom en code :

```
.convert_name (expression[,"séparateur"[,"délimiteurs"],case])
```

Utilisez la syntaxe suivante pour convertir un code en nom :

```
.convert_code (expression[,"séparateur"[,"délimiteurs"]])
```

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 il peut inclure un suffixe ou un préfixe. Type : Template simple
séparateur	[facultatif] Caractère généré chaque fois qu'un séparateur déclaré dans **délimiteurs** est trouvé dans le code. Par exemple, "_" (tiret bas). Type : Texte
délimiteurs	[facultatif] Spécifie les différents délimiteurs 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

Macros .delete et .replace

Ces macros suppriment ou remplacent toutes les instances de la chaîne donnée dans le texte en entrée.

```
.delete (chaîne)
    bloc-entrée
.enddelete
```

```
.replace (chaîne,nouvelle-chaîne)
    bloc-entrée
.endreplace
```

Les paramètres suivants sont disponibles :

Paramètre	Description
chaîne	Spécifie la chaîne à supprimer. Type : Texte
nouvelle-chaîne	[.replace uniquement] Spécifie la chaîne à remplacer **chaîne**. Type: Text

Paramètre	Description
bloc-entrée	Spécifie la chaîne à parcourir pour y chercher des instances de la **chaîne** à supprimer ou remplacer. Type : Template complexe

Exemples	Résultat

`.delete(Get)`
`    GetCustomerName`
`.enddelete`
	`CustomerName`

`.replace(Get,Set)`
`GetCustomerName`
`.endreplace`
	`SetCustomerName`

`.replace(" ", _)`
`Customer Name`
`.endreplace`
	`Customer_Name`

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** de la feuille de propriétés d'objet et dans la fenêtre **Résultats**.

```
.error message
```

```
.warning message
```

Les paramètres suivants sont disponibles :

Paramètre	Description
message	Spécifie le texte du message. Type : Template simple

Exemple
`.error aucune valeur initiale fournie pour l'attribut %Code% de la classe %Parent.Code%`

Macro .execute_command

Cette macro lance des exécutables dans le cadre d'une commande de génération. Si elle échoue pour une raison quelconque (exécutables non trouvés, ou bien résultat envoyé vers `stderr`), l'exécution de la commande est interrompue.

```
.execute_command (cmd [,args [,mode]])
```

Les paramètres suivants sont disponibles :

Paramètre	Description
cmd	Spécifie le chemin d'accès de l'exécutable. Type : Template simple (séquences d'échappement ignorées)
args	[facultatif] Spécifie les arguments de l'exécutable. Type : Template simple (séquences d'échappement ignorées)
mode	[facultatif] Spécifie le mode d'exécution. 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

Exemple
`.execute_command(notepad, fichier1.txt, cmd_ShellExecute)`

Pour plus d'informations sur les commandes de génération, voir *Catégorie Generation* à la page 122.

Macro .execute_vbscript

Cette macro est utilisée pour exécuter un script VB spécifié dans un fichier séparé dans le cadre d'une commande de génération.

```
.execute_vbscript (fichier-vbs [,paramètre-script])
```

Les paramètres suivants sont disponibles :

Paramètre	Description
fichier-vbs	Spécifie le chemin du fichier VB script. Type : Template simple (séquences d'échappement ignorées)

Paramètre	Description
paramètre-script	[facultatif] Paramètre passé au script via la propriété globale `ScriptInputParameters`. Type : Template simple

Exemple

```
.execute_vbscript(C:\exemples\vbs\login.vbs, %username%)
```

Le résultat du script est disponible dans la propriété globale `ScriptResult` (voir *Manipulation des modèles, des collections et des objets (Scripting)* à la page 338). 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 les commandes de génération, voir *Catégorie Generation* à la page 122.

Macro .foreach_item

Cette macro procède à l'itération sur une collection de sous-objets ou d'objets associés.

```
.foreach_item (collection [,en-tête [,fin [,filtre [,ordre]]]])
    résultat
.next [(séparateur)]
```

Les paramètres suivants sont disponibles :

Paramètre	Description
collection	Spécifie la collection sur laquelle effectuer l'itération. Type : Template simple
en-tête	[facultatif] Spécifie le texte à générer avant le résultat, sauf si la collection est vide. Type : Texte
fin	[facultatif] Spécifie le texte à générer après le résultat, sauf si la collection est vide. Type : Texte
filtre	[facultatif] Spécifie un filtre à appliquer à la collection avant itération. Type : Condition simple

Paramètre	Description
ordre	[facultatif] Spécifie l'ordre dans lequel l'itération sera effectuée sur la collection au format : `%Item1.propriété% <= %Item2.propriété%` Lorsque la comparaison est évaluée à true, `%Item1%` est placé après `%Item2%`. Par défaut, la collection est classée par ordre alphabétique de nom. Type : Condition simple
résultat	Spécifie le texte à produire pour chaque élément de la collection. Type : Template complexe
séparateur	[facultatif] Spécifie le texte à générer entre chaque instance de **résultat**. Type : Texte

Remarque : Si des valeurs de paramètre contiennent des virgules, des accolades et des espaces de début ou de fin, elles doivent être délimitées par des guillemets. La séquence d'échappement pour les guillemets au sein d'un paramètre est `\"`.

Exemples

Liste simple :

```
.foreach_item(Attributes)
    *%Code% (%DataType%)[ = %InitialValue%];
.next(\n)
```

Résultat :

```
    *available (boolean) = true;
    *actualCost (int);
    *baseCost (int);
    *color (String);
    *height (int) = 10;
    *width (int) = 5;
    *name (int);
```

Exemples

Avec en-tête et fin :

```
.foreach_item(Attributes,Attributes:\n,\n\nEnd of Attribute List)
    *%Code% (%DataType%)[ = %InitialValue%];
.next(\n)
```

Résultat :

```
Attributes:
*available (boolean) = true;
*actualCost (int);
*baseCost (int);
*color (String);
*height (int) = 10;
*width (int) = 5;
*name (int);

End of Attribute List
```

Avec filtre :

```
.foreach_item(Attributes,,,%.1:Code%==a)
    *%Code% (%DataType%)[ = %InitialValue%];
.next(\n)
```

Résultat :

```
    *available (boolean) = true;
    *actualCost (int);
```

Avec classement alphabétique inverse :

```
.foreach_item(Attributes,,,, %Item1.Code% <= %Item2.Code% )
    *%Code% (%DataType%)[ = %InitialValue%];
.next(\n)
```

Résultat :

```
    *width (int) = 5;
    *name (int);
    *height (int) = 10;
    *color (String);
    *baseCost (int);
    *available (boolean) = true;
    *actualCost (int);
```

Macro .foreach_line

Cette macro procède à l'itération sur les lignes d'un bloc de texte multiligne en utilisant la variable locale `%CurrentLine%`.

```
.foreach_line (entrée [,en-tête [,fin]])
    résultat
.next [(séparateur)]
```

Les paramètres suivants sont disponibles :

Paramètre	Description
entrée	Spécifie le texte sur lequel l'itération doit être effectuée. Type : Template simple
en-tête	[facultatif] Spécifie le texte à générer avant le résultat, sauf s'il n'y a aucun résultat. Type : Texte
fin	[facultatif] Spécifie le texte à générer après le résultat, sauf s'il n'y a aucun résultat. Type : Texte
résultat	Spécifie le texte à produire pour chaque ligne de l'entrée. Type : Template complexe
séparateur	[facultatif] Spécifie le texte à générer entre chaque ligne de **résultat**. Type : Texte

Exemple

```
.foreach_line(%Comment%,"/**\n","\n*/")
* %CurrentLine%
.next("\n")
```

Résultat :

```
/**
* Ceci est mon commentaire.
* Il s'agit d'un commentaire de style documentation Java.
* Il est réparti sur plusieurs lignes.
*/
```

Macro .foreach_part

Cette macro permet une itération des parties d'une chaîne divisée par un délimiteur utilisant la variable locale `%CurrentPart%` spéciale.

```
.foreach_part (entrée [,"délimiteur" [,en-tête [,fin]]])
    résultat
.next[(séparateur)]
```

Les paramètres suivants sont disponibles :

Paramètre	Description
entrée	Spécifie le texte sur lequel effectuer l'itération. Type : Template simple

Paramètre	Description
délimiteur	Spécifie la sous-chaîne qui divise l'entrée en parties. Vous pouvez spécifier plusieurs caractères, et mêmes des plages de caractères. Par exemple `[A-Z]` spécifie que n'importe quelle lettre majuscule peut agir comme délimiteur. Par défaut le délimiteur est défini à `' -_,\t'` (espace, tiret, trait de soulignement, virgule, ou tabulation). **Remarque :** Le délimiteur doit être encadré d'apostrophes s'il contient un espace. Type : Texte
en-tête	[facultatif] Spécifie le texte à générer avant le résultat, sauf s'il n'y a pas de résultat. Type : Texte
fin	[facultatif] Spécifie le texte à générer après le résultat, sauf s'il n'y a pas de résultat. Type : Texte
résultat	Spécifie le texte à produire pour chaque partie de l'entrée. Type : Template complexe
séparateur	[facultatif] Spécifie le texte à générer entre chaque partie du **résultat**. Type : Texte

Par exemple :

Exemples

Ce template est appliqué à `My class` :

```
.foreach_part (%Name%)
%.FU:CurrentPart%
.next
```

Résultat :

```
MyClass
```

Ce template est appliqué à `My class` :

```
.foreach_part (%Name%,' -_',tbl_)
%.L:CurrentPart%
.next(_)
```

Résultat :

```
tbl_my_class
```

Exemples

Ce template est appliqué à `MyClass` :

```
.foreach_part (%Name%,[A-Z])
%.L:CurrentPart%
.next(-)
```

Résultat :

```
my-class
```

Macro .if

Cette macro est utilisée pour la génération conditionnelle.

```
.if[not] condition
        résultat
    [(.elsif[not] condition
        résultat)*]
    [.else
        résultat]
.endif [(fin)]
```

Les paramètres suivants sont disponibles :

Paramètre	Description
condition	Spécifie la condition à évaluer, sous la forme : **variable** [**opérateur** **comparaison**] Dans laquelle *comparaison* peut être : • Texte, ou template simple • `true` ou `false` • `null` ou `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 vide. Si **variable** et **comparaison** ne sont pas des entiers, les opérateurs effectuent une comparaison de chaîne qui prend en compte les numéros imbriqués. Par exemple : `Class_10 > Class_2` Vous pouvez chaîner les conditions en utilisant les opérateurs logiques `and` et `or`. Type : Template simple
résultat	Spécifie le résultat si la condition est remplie (true). Type : Template complexe

Paramètre	Description
fin	[facultatif] Spécifie le texte à générer après le résultat, sauf si le résultat est vide. Type : Texte

Exemples

Bloc `.if` simple :

```
.if %Abstract%
        Cette classe est abstraite.
.endif
```

Résultat (si la propriété **Abstrait** est sélectionnée) :

```
Cette classe est abstraite.
```

Avec deux conditions et une clause `.else` :

```
.if (%Abstract%==false) && (%Visibility%=="+")
        Cette classe est publique et concrète.
    .else
Cette classe n'est ni publique, ni concrète.
.endif
```

Résultat (si la propriété **Abstrait** n'est pas sélectionnée et que la propriété **Visibilité** est définie à `Public`):

```
Cette classe est publique et concrète.
```

Avec une clausse `.elseif` :

```
.if (%Abstract%==false) && (%Visibility%=="+")
        Cette classe est publique et concrète.
.elsif (%Visibility%=="+")
        Cette classe est publique.
    .else
Cette classe n'est ni publique, ni concrète.
.endif
```

Macro .log

Cette macro consigne un message dans l'onglet **Génération** de la fenêtre **Résultats** dans le cadre d'une commande de génération.

```
.log message
```

Exemple

```
.log undefined environment variable: JAVAC
```

Pour plus d'informations sur les commandes de génération, voir *Catégorie Generation* à la page 122.

Macros .lowercase et .uppercase

Ces macros convertissent des blocs de texte dans la casse spécifiée.

```
.lowercase
    bloc-entrée
.endlowercase
```

```
.uppercase
    bloc-entréeinput
.enduppercase
```

Les paramètres suivants sont disponibles :

Paramètre	Description
bloc-entrée	Spécifie le texte à convertir. Type : Template complexe

Exemple	Résultat

`.lowercase`
`  %Comment%`
`.endlowercase`
	Appliqué à :
	`Ceci est un commentaire.`
	Produit :
	`ceci est un commentaire.`

Macros .object et .collection

Ces macro renvoient un OID d'objet unique ou une collection d'objets sous la forme d'une concaténation des OID terminées par des points-virgules, et sont généralement utilisées pour créer des templates qui renvoient des objets à utiliser par les autres templates.

```
.collection (portée [,filtre])
```

```
.object (portée [,filtre])
```

Les paramètres suivants sont disponibles :

Paramètre	Description
portée	Spécifie la collection sur laquelle procéder à l'itération. Type: Template simple qui renvoie une portée de collection
filtre	[facultatif] Spécifie une condition de filtre pour filtrer la collection. Type: Template simple

Exemples

`.object(Attributes, (%.1:Code%>= a) and (%.1:Code% <= e))`
Résultat :
`C73C03B7-CD73-466A-B323-0B90B67E82FC`

`.collection(Attributes, (%.1:Code%>= a) and (%.1:Code% <= e))`
Résultat :
`C73C03B7-CD73-466A-B323-0B90B67E82FC;77E3F55C-CF24-440F-84E7-5AA7B3399C00;F369CD8C-0C16-4896-9C2D-0CD2F80D6980;00ADD959-0705-4061-BF77-BB1914EDC018;`

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 paramètres suivants sont disponibles :

Paramètre	Description
mode	Spécifie le niveau d'interaction requis. Vous pouvez choisir l'une des valeurs suivantes : • `im_Batch` - Supprime l'affichage des boîte de dialogue et utilise systématiquement les valeurs par défaut. Par exemple, si votre modèle contient des raccourcis externes et que le modèle cible pour les raccourcis est fermé, ce mode va automatiquement ouvrir le modèle sans interaction de l'utilisateur. • `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.

Macros .set_object, .set_value et .unset

Ces macros sont utilisées afin de définir une variable locale de type objet (objet local) ou un type de valeur, ou d'annuler leur définition.

Utilisez la syntaxe suivante pour créer un objet local :

```
.set_object ( [portée.] nom [,ref-objet [,mode]])
```

Utilisez la syntaxe suivante pour créer une variable locale :

```
.set_value ( [portée.] nom, valeur [,mode] [,unescape])
```

Utilisez la syntaxe suivante pour supprimer un objet local ou une variable locale :

```
.unset ( [portée.] nom)
```

Les paramètres suivants sont disponibles :

Paramètre	Description
portée	[facultatif] Spécifie la portée qualifiante. Si vous ne définissez pas de portée, la portée de l'objet est la portée courante. Utilisez le mot clé `this` pour attributer de façon explicite la portée de l'objet courant, ou `Parent` pour attriibuer celle de l'objet parent . Type : Template simple qui renvoie une portée d'objet ou de collection
nom	Spécifie le nom de l'objet ou de la variable, que vous pouvez référencer ailleurs dans le template sous la forme `%name%`. Type : Template simple
ref-objet	[.set_object uniquement - facultatif] Spécifie une référence d'objet. Si aucune référence n'est spécifiée ou si une chaîne vide est fournie, la variable est une référence à l'objet actif dans la portée de conversion courante. Type : [*portée.*]*portée-objet*]
valeur	[.set_value uniquement] Spécifie la valeur à donner à la variable. Type : Template simple (les séquences d'échappement sont ignorées)
mode	[facultatif] Spécifie le mode de création. Vous pouvez choisir l'une des valeurs suivantes : • new - Force la (re)-définition de la variable à la portée courante. Recommandé lorsqu'une variable portant le même nom peut être déjà définie dans une portée précédente. • 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.
unescape	[.set_value uniquement - facultatif] Spécifie que les caractères d'échapement tels que `\n` dans la valeur fournie doivent être interprétés. Par défaut, ces caractères ne sont pas interprétés.

Examples:

Exemples

```
.set_object(Attribute1, Attributes.First)
.set_value(FirstAttributeCode, %Attributes.First.Code%)
%FirstAttributeCode% (OID: %Attribute1%)
```

Résultat :

```
a1 (OID: 63442F85-48DF-42C8-92C1-0591F5D34525)
```

```
.set_value(this.key, %Code%-%ObjectID%)
```

Résultat :

```
C1-40D8F396-EE29-4B7B-8C78-E5A0C5A23325
```

```
.set_value(i, 1, new)
%i?%
.unset(i)
%i?%
```

Résultat :

```
true
false
```

Le premier appel à `%i?%` produit `true` car la variable `i` est définie, et le second produit `false`, car elle a été annulée.

```
.set_value(oneline, "line1\nline2")
.set_value(twolines,"line3\nline4",, unescape)
%oneline%
%twolines%
```

Result:

```
line1\nline2
line3
line4
```

Remarque : Vous pouvez utiliser l'opérateur déréférençant, `*` (voir *Opérateurs du langage de génération par template* à la page 273), afin de convertir la valeur d'une variable avec la macro `.set_value` en un nom de template. Par exemple, le code suivant équivaut à `%Code%.`:

```
.set_value(i, Code)
%*i%
```

Macro .unique

Cette macro produit un bloc dans lequel chaque ligne de texte généré est unique, et elle est souvent utilisée pour calculer les importations, inclusions, typedefs ou des déclarations anticipées dans des langages tels que Java, C++ ou C#.

```
.unique
    bloc-entrée
.endunique[(fin)]
```

Les paramètres suivants sont disponibles :

Paramètre	Description
bloc-entrée	Spécifie le bloc de texte à traiter. Type : Template complexe
fin	[facultatif] Spécifie le texte à générer après le résultat, sauf si la collection est vide. Type : Texte

Exemple

```
.unique
    import java.util.*;
    import java.lang.String;
    %imports%
.endunique
```

Macro .vbscript

Cette macro est utilisée pour incorporer du code VBScript dans un template dans le cadre d'une commande de génération. Le résultat du script est disponible sous la forme du tableau `ScriptResult`.

```
.vbscript [(liste-param-script)]
    bloc-entrée
.endvbscript [(fin)]
```

Les paramètres suivants sont disponibles :

Paramètre	Description
liste-param-script	Spécifie les paramètres à passer au script via le table `ScriptInputArray`. Type : Liste d'arguments de template-simple séparés par des virgules
bloc-entrée	Spécifie le code VBscript à exécuter. Type : Texte

Paramètre	Description
fin	Ajouté au résultat, s'il y en a un Type : Texte

Exemples

Ce simple script accepte les deux mots `hello` et `world` comme paramètres d'entrée, et les renvoie comme une seule et unique chaîne séparée par un espace :

```
.vbscript(hello, world)
ScriptResult = ScriptInputArray(0) + " " + ScriptInputArray(1)
.endvbscript
```

Résultat :

```
hello world
```

Exemples

Ce script accepte un code d'attribut, le compare à tous les codes d'attribut du modèle courant et lui ajoute un `1` s'il en trouve un identique :

```
.set_value(_code,%@1%,new)
.vbscript(%_code%)
   Dim attrCode
   attrCode = ScriptInputArray(0)

   While (attrFound(attrCode))
      attrCode = attrCode + "1"
   Wend

   Function attrFound(attrCode)
      Dim found, attr
      found = False
      For Each attr in ActiveSelection.Item(0).Attributes
         If attr.Code = attrCode Then
            found = True
            Exit For
         End If
      Next

      For Each attr in ActiveSelection.Item(0).InheritedAttribu-
tes
         If attr.Code = attrCode Then
            found = True
            Exit For
         End If
      Next
      attrFound = found
   End Function

   ScriptResult = attrCode
.endvbscript
```

Remarque : L'objet actif de la portée de traduction courante est accédé comme `ActiveSelection.Item(0)` (voir *Manipulation des modèles, des collections et des objets (Scripting)* à la page 338).

Pour plus d'informations sur les commandes de génération, voir *Catégorie Generation* à la page 122.

Syntaxe du langage de génération par templates et erreurs de conversion

Les messages d'erreur interrompent la génération des fichiers dans lequel les erreurs ont été détectées, et ces erreurs sont affichées dans l'onglet **Aperçu** de la feuille de propriétés d'objet correspondante.

Les messages d'erreur ont le format suivant :

```
cible::catég-chemin nom-template-complet(numéro-ligne)
métaclasse-objet-actif code-objet-actif):
    type-erreur message-erreur
```

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 .else sans .if correspondant .endif sans .if correspondant	Ajoutez un `.endif` ou `.if` (voir *Macro .if* à la page 297).
.next attendu .next sans .foreach correspondant	Ajoutez un `.next` ou `.foreach` approprié pour le bloc de collection (par exemple, voir *Macro .foreach_item* à la page 292).
.end%s attendu	Ajoutez un `.end` approprié pour le bloc de macro (par exemple, voir *Macro .unique* à la page 303).
.end%s sans .%s correspondant	Ajoutez un *.macro* approprié à `.end`*macro* (par exemple, voir *Macro .vbscript* à la page 303).
Parenthèses manquantes ou non appariées	Corrigez les éventuelles parenthèses non appariées.
Paramètre inattendus : *params-supplémentaires*	Supprimez les éventuels paramètres inutiles
Macro inconnue	Remplacez par une macro valide (voir *Guide de référence des macros du langage de génération par template* à la page 285).
.execute_command incorrect syntax	La syntaxe correcte est affichée dans l'onglet Aperçu, ou dans la fenêtre Résultats (voir *Macro .execute_command* à la page 291).
Syntaxe incorrecte pour Change_dir	Voir *Macros .change_dir et .create_path* à la page 288.

Message d'erreur	Description et correction
Syntaxe incorrecte pour convert_name Syntaxe incorrecte pour convert_code	Voir *Macros .convert_name et .convert_code* à la page 288.
Syntaxe incorrecte pour set_object Syntaxe incorrecte pour set_value	Voir *Macros .set_object, .set_value et .unset* à la page 300.
Syntaxe incorrecte pour execute_vbscript	Voir *Macro .execute_vbscript* à la page 291.

Les erreurs de conversion sont des erreurs d'évaluation sur une variable lors de l'évaluation d'un template :

Message d'erreur de conversion	Description et correction
collection non résolue : *collection*	Collection inconnue (voir *Accès aux collections de sous-objets ou d'objets associés* à la page 267).
membre non résolu : *membre* objet null variable objet attendue : *objet*	Membre inconnu, membre d'objet null, ou chaîne attendue au lieu d'un objet (voir *Extraction des propriétés d'objet* à la page 266).
aucune portée externe	Utilisation incorrecte du mot clé `Outer` (voir *Portée de la conversion* à la page 276).
Erreur d'exécution VBScript	Erreur VB script (voir *Macro .vbscript* à la page 303).
Point de blocage potentiel détecté	Blocage provoqué par une boucle infinie.

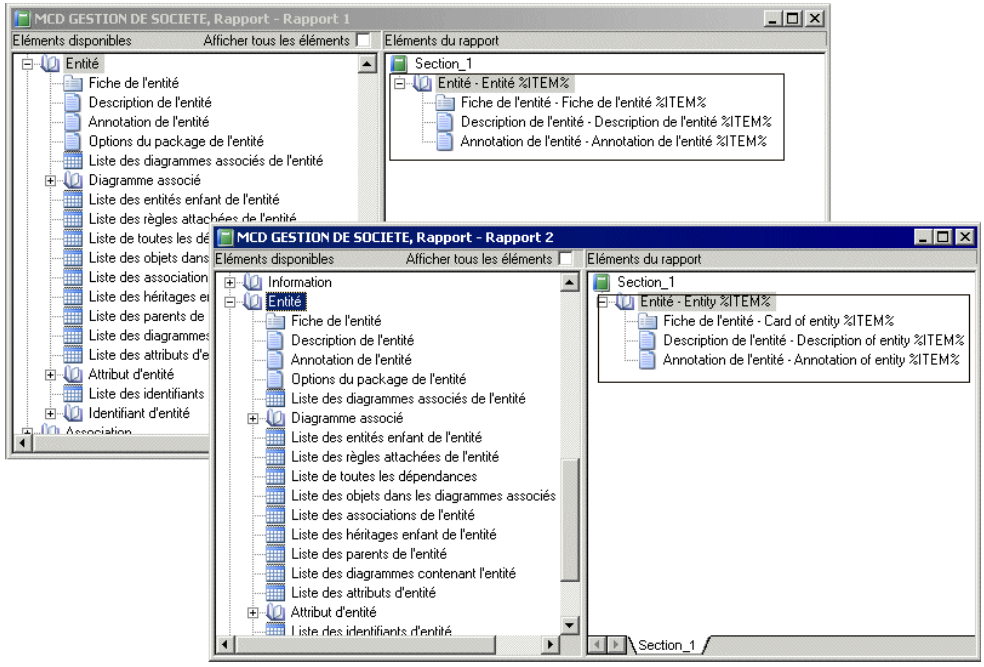
Chapitre 6

Traduction de rapports à l'aide des fichiers de langue de rapport

Lorsque vous créez un rapport, vous sélectionnez une langue de rapport, qui contient tous les textes utilisés pour générer un rapport dans la langue sélectionnée, comme par exemple des titres de section de rapport, des types d'objet de modèle, et leurs propriétés. PowerAMC prend en charge le Français (valeur par défaut) l'Anglais, le Chinois simplifié et Chinois traditionnel. Vous pouvez éditer ces fichiers, ou les utiliser comme base pour créer vos propres fichiers de traductions dans d'autres langues.

Les fichiers de langue de rapport ont un suffixe .xrl et sont stockés dans *répertoire_installation*`/Fichiers de ressources`. Pour afficher la liste des langues de rapport, sélectionnez **Outils > Ressources > Langues de rapport**. Pour plus d'informations sur les outils disponibles dans les listes de fichiers de ressources, voir *Chapitre 1, Fichiers de ressources PowerAMC* à la page 1.

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 (voir *Chapitre 5, Personnalisation de la génération à l'aide du langage de génération par template* à la page 265) 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 et réduire 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 données de l'entité MonEntité` est automatiquement généré comme suit :

- le template d'élément de rapport List - object collections (voir *Catégorie Profile/Report Item Templates* à la page 328) est traduit comme suit :

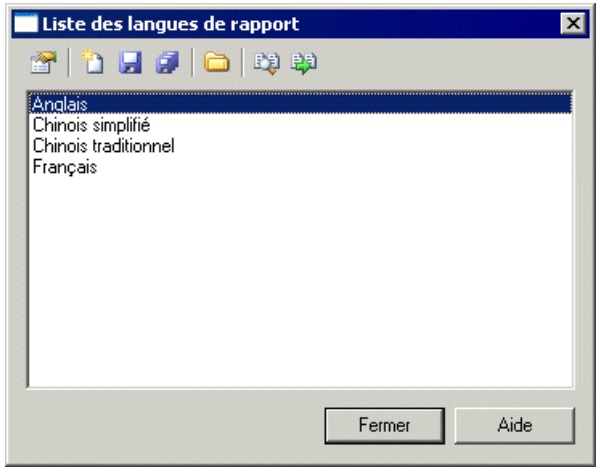
  ```
  Liste des %@Value% %ParentMetaClass.OFTHECLSSNAME% %%PARENT%%
  ```

  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 322), `données`.
  - `%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 326), `l'entité`.
  - `%%PARENT%%` - est remplacée par le nom de l'objet spécifique (voir *Catégorie Object Attributes* à la page 322), `MonEntité`.

Ouverture d'un fichier de langue de rapport

Vous pouvez consulter et éditer les fichiers 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'ouvrir 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 **Langue**. 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, Fichiers de ressources PowerAMC* à la page 1.

Création d'un fichier de langue de rapport pour une nouvelle langue

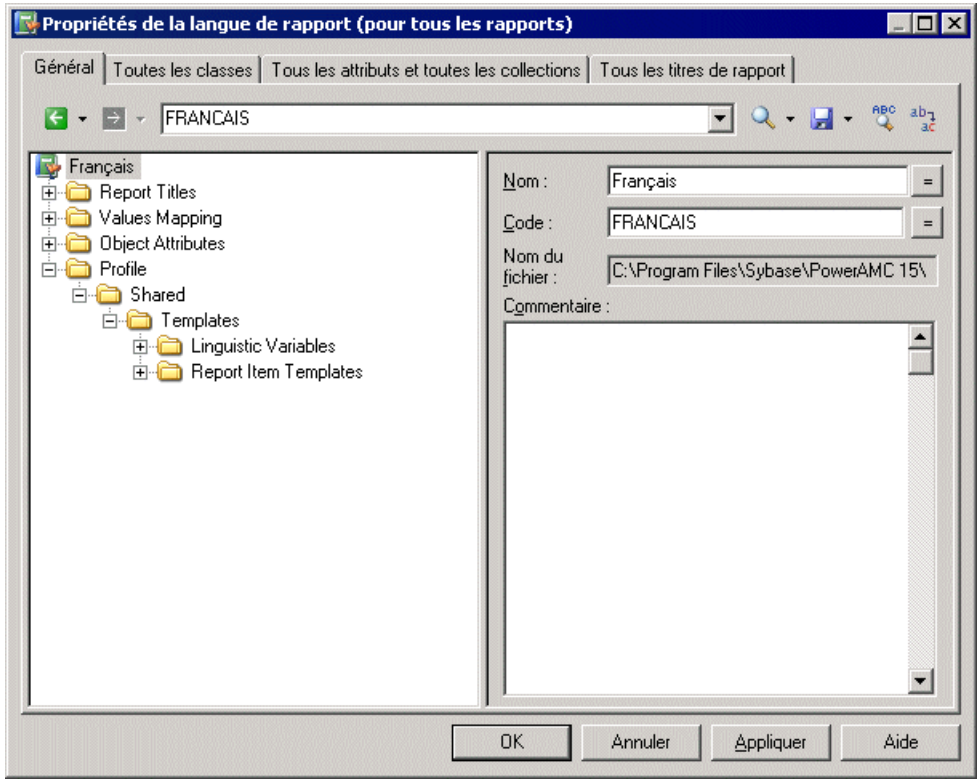
Vous pouvez traduire les titres de rapport et autres blocs de texte utilisés pour générer des rapports PowerAMC dans un nouveau langage.

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**, 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 (voir *Catégorie Values Mapping* à la page 314).

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 (voir *Catégorie Profile/Linguistic Variables* à la page 326).
7. Ouvrez la catégorie **Profile > Report Items Templates**, et traduisez les différents templates (voir *Catégorie Profile/Report Item Templates* à la page 328). Lors de la traduction, vous pouvez découvrir d'autres variables linguistiques à créer.
8. Cliquez sur l'onglet **Toutes les classes** afin d'afficher une liste triable de toutes les métaclasses disponibles dans le métamodèle PowerAMC (voir *Onglet Toutes les classes* à la page 324). Traduisez chaque nom de métaclasse.
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 (voir *Onglet Tous les attributs et toutes les collections* à la page 325). Traduisez chaque nom d'attribut et de collection.
10. Cliquez sur l'onglet **Tous les titres de rapport**, puis passez en revue les titres de rapport générés (voir *Onglet Tous les titres de rapport* à la page 321). 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 langue de rapport peut maintenant être associé à un rapport.

Propriétés d'un fichier de langue de rapport

Tous les fichiers de langue de rapport peuvent être ouverts dans l'Editeur de ressources, et ils ont la même structure de base.

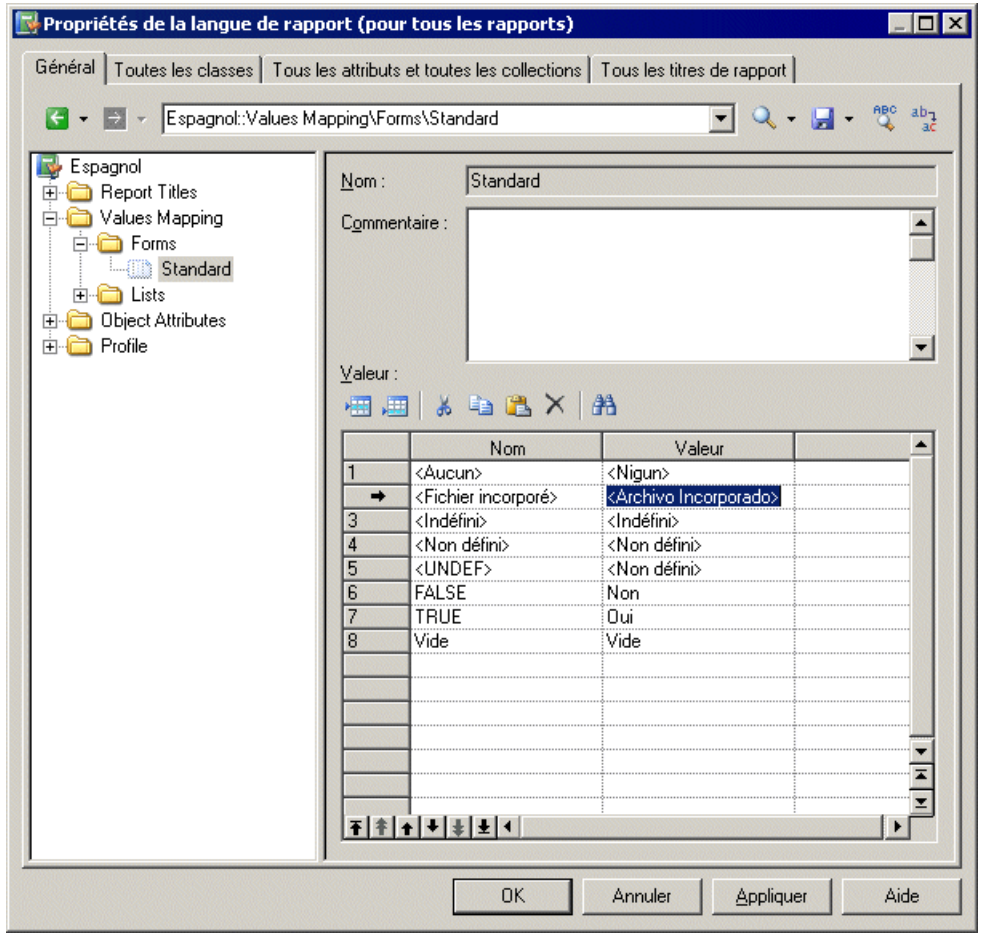


Le noeud racine de chaque fichier contient les propriétés suivantes :

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.

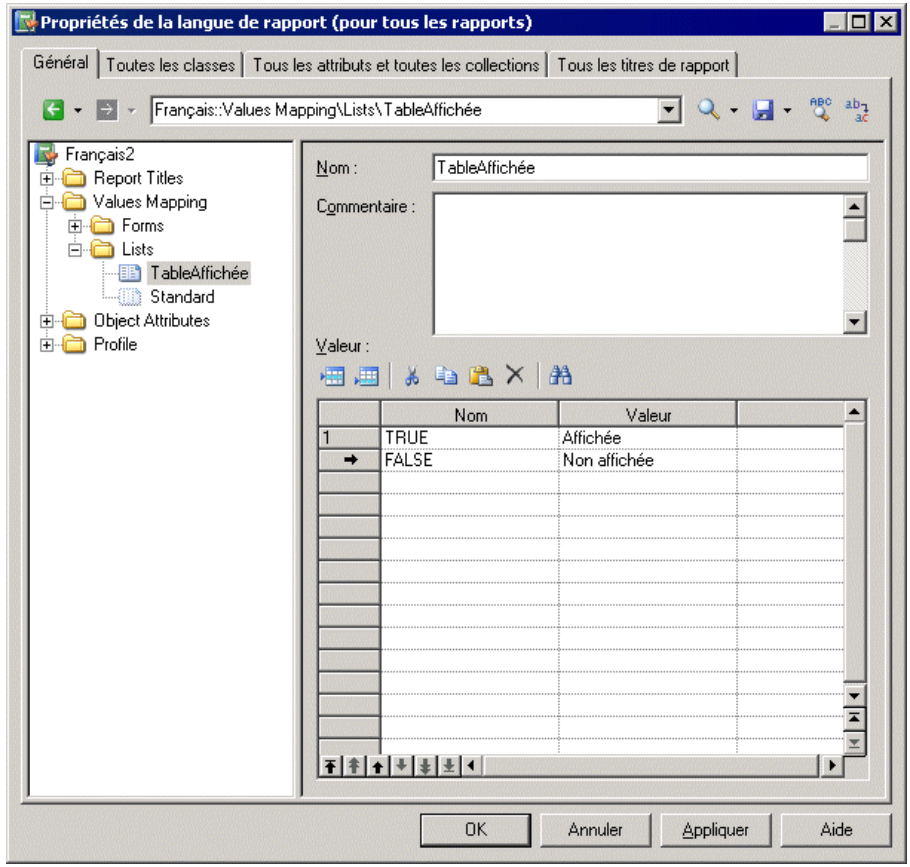
Exemple : Création d'une table de correspondances, et association de cette table à un objet de modèle particulier

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.

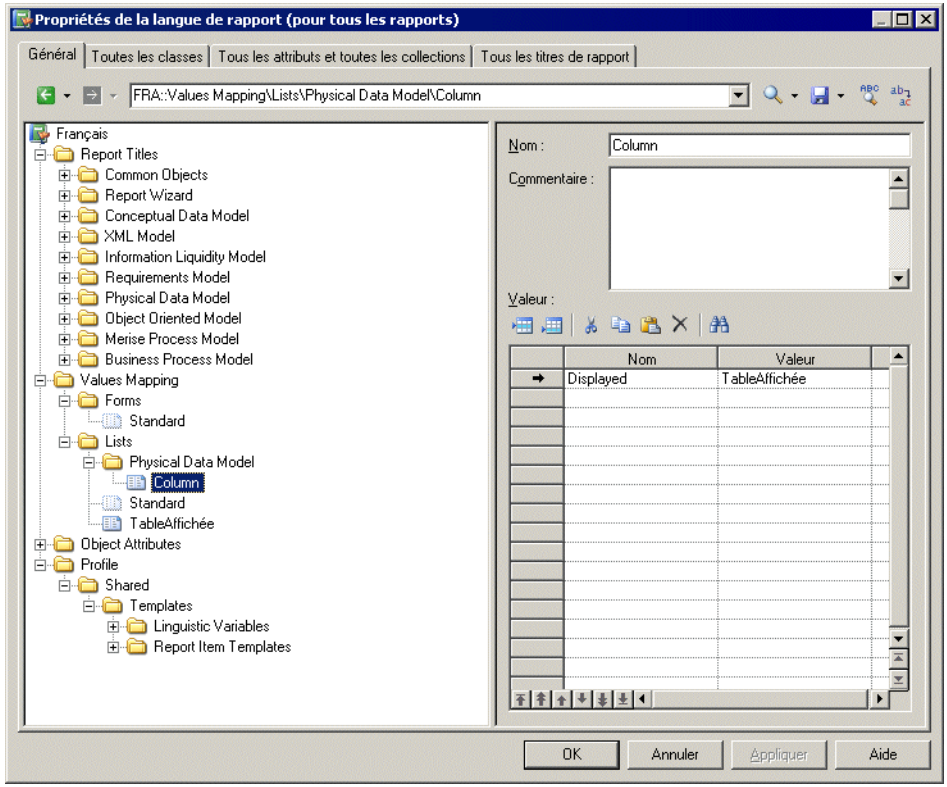
Dans l'exemple suivant, la table de correspondances TableAffichée est utilisée pour remplacer la table de correspondances Standard pour les colonnes de MPD afin de fournir des valeurs personnalisées pour la propriété Affichée, qui contrôle l'affichage de la colonne sélectionnée dans le symbole de table. 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.



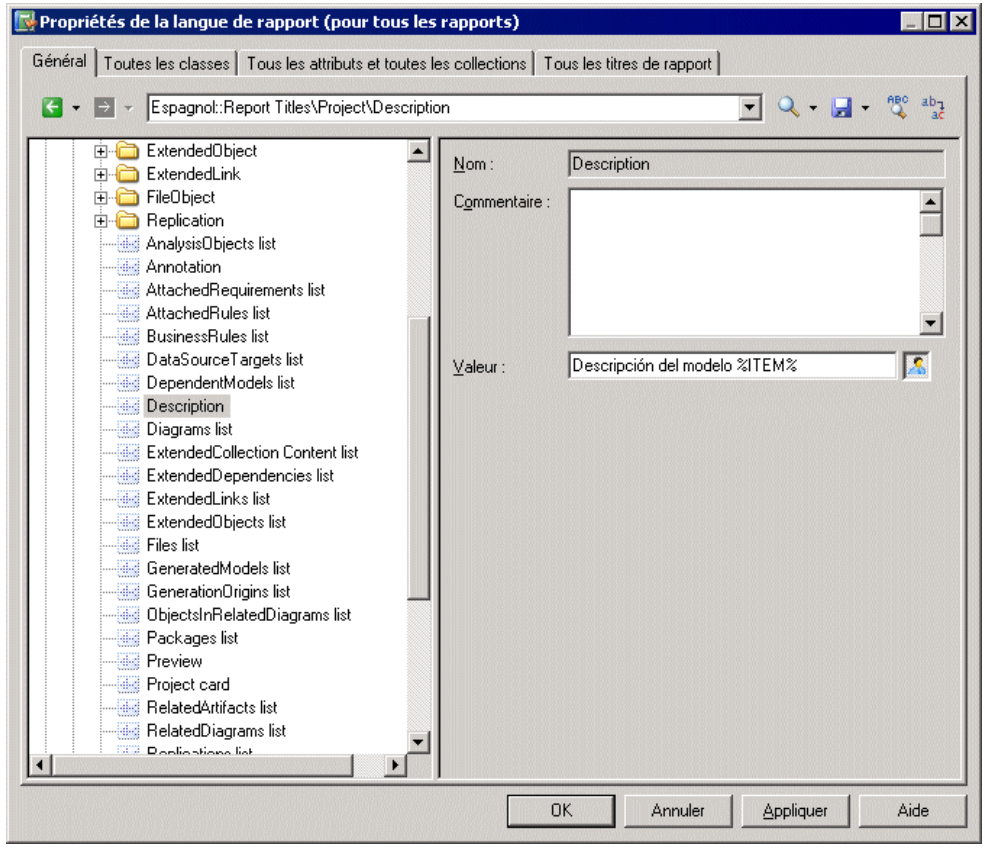
8. Cliquez sur Appliquer pour enregistrer vos modifications. Lorsque vous générez un rapport, la propriété Affichée sera affichée avec les valeurs spécifiées :

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 model 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.

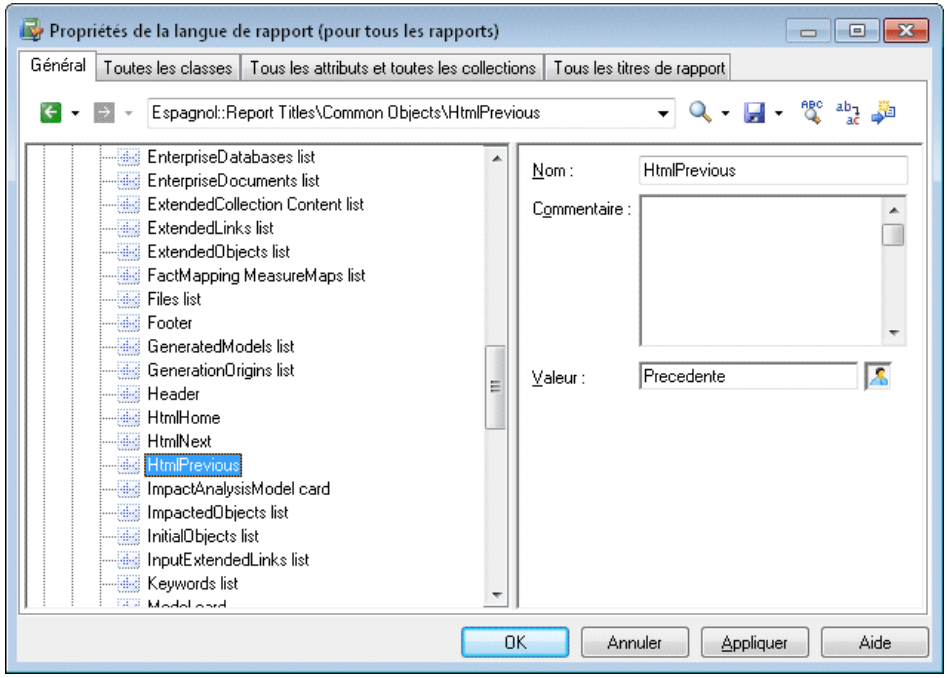
Par défaut (à l'exception des sous-catégories Common Objects et Report Wizard) ces traductions sont automatiquement générées dans les templates de la catégorie Profile (voir *Catégorie Profile/Report Item Templates* à la page 328). Vous pouvez passer outre les valeurs générées automatiquement en saisissant votre propre texte dans la zone **Nom localisé**, ce qui va enfoncer le bouton **Défini par l'utilisateur** pour indiquer que la valeur n'est plus une valeur générée.

Remarque : L'onglet **Tous les titres de rapport** (voir *Onglet Tous les titres de rapport* à la page 321) 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.

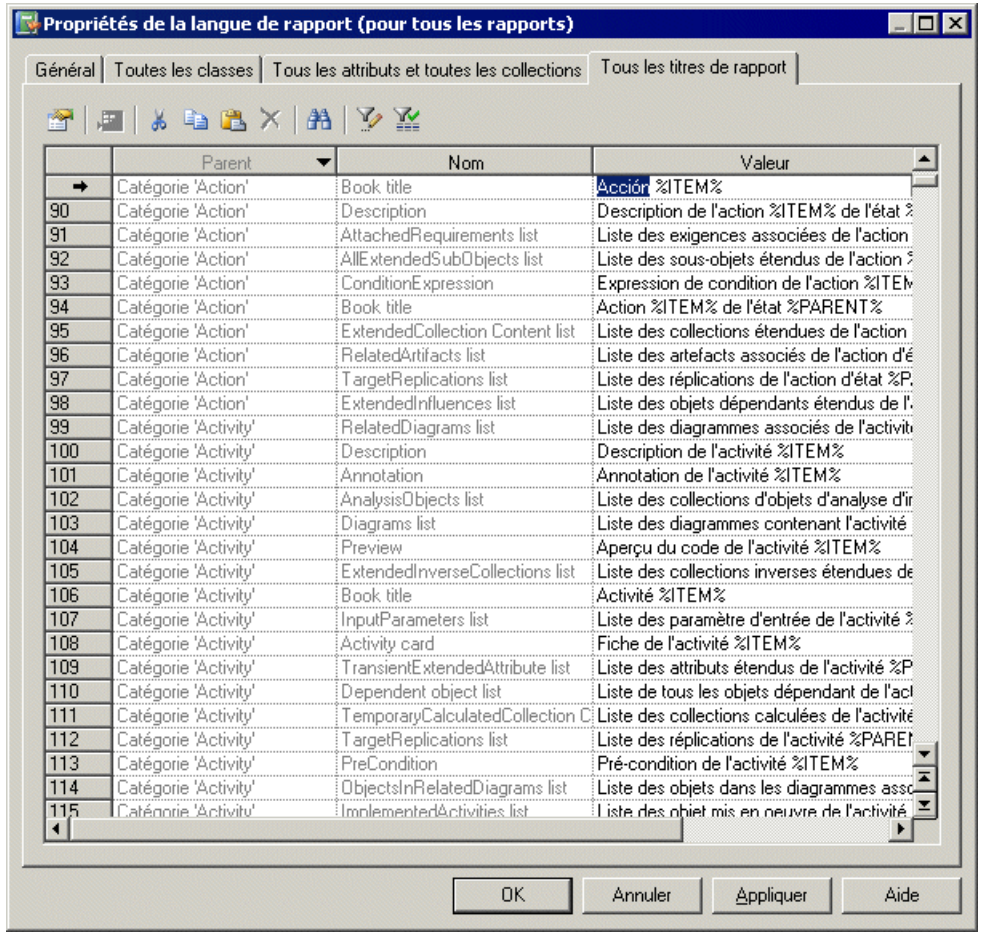
1. Ouvrez la catégorie **Report Titles** > **Common Objects**.
2. Cliquez sur l'entrée `HtmlPrevious` pour afficher ses propriétés, et saisissez une traduction dans la zone **Valeur**. Le bouton **Défini par l'utilisateur** est enfoncé pour indiquer que la valeur n'est plus une valeur générée.



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

Onglet Tous les titres de rapport

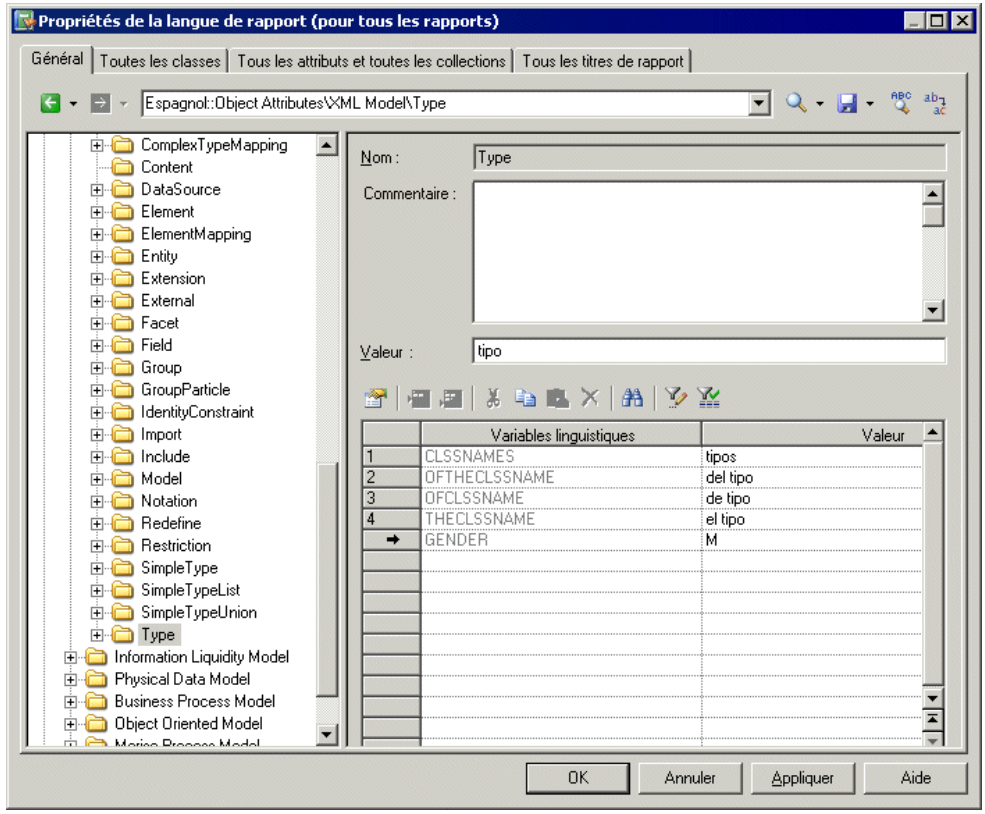
L'onglet Tous les titres de rapport répertorie tous les titres de rapport et autres textes disponibles dans la catégorie Report Titles de l'onglet Général, mais sa présentation sous forme de tableau rend son utilisation plus simple



Pour chaque rapport répertorié dans la colonne **Nom**, vous pouvez consulter ou redéfinir une traduction dans la colonne **Nom localisé**. Vous pouvez trier la liste pour regrouper les objets dont les noms sont similaires, et traduire simultanément plusieurs éléments identiques en sélectionnant plus lignes.

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).

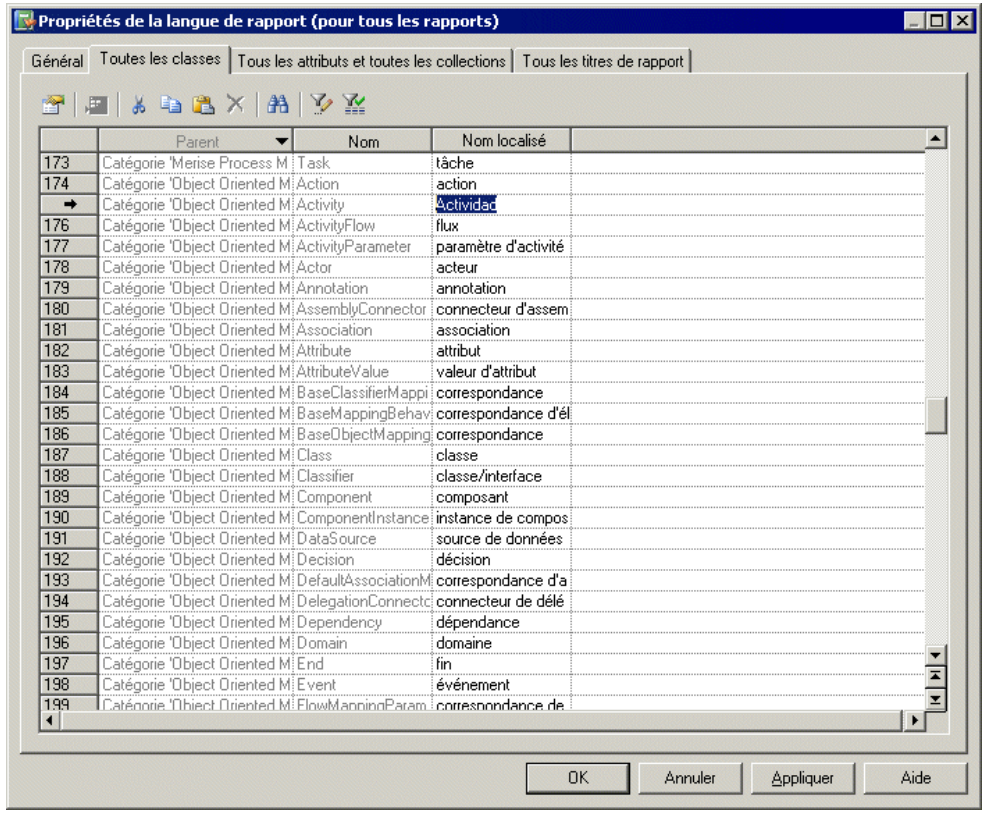
Sous-catégorie	Description
Common Objects	Contient les textes pour les métaclasses, collections et attributs disponibles pour tous les types de modèles, pour lesquels vous devez fournir des traductions. Exemple : Diagram fournit le texte pour un diagramme dans n'importe quel type de modèle.

Pour chaque élément dont le nom est donné, vous devez fournir une traduction dans la zone **Nom localisé**. 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 318).

En ce qui concerne les métaclasses uniquement, les variables linguistiques que vous avez spécifiées (voir *Catégorie Profile/Linguistic Variables* à la page 326) sont répertoriées avec le résultat de leur application dans les traductions spécifiées dans la zone **Nom localisé**. Vous pouvez passer outre les valeurs générées automatiquement en saisissant votre propre texte dans la zone **Nom localisé**, ce qui va enfoncer le bouton **Défini par l'utilisateur** pour indiquer que la valeur n'est plus une valeur générée.

Onglet Toutes les classes

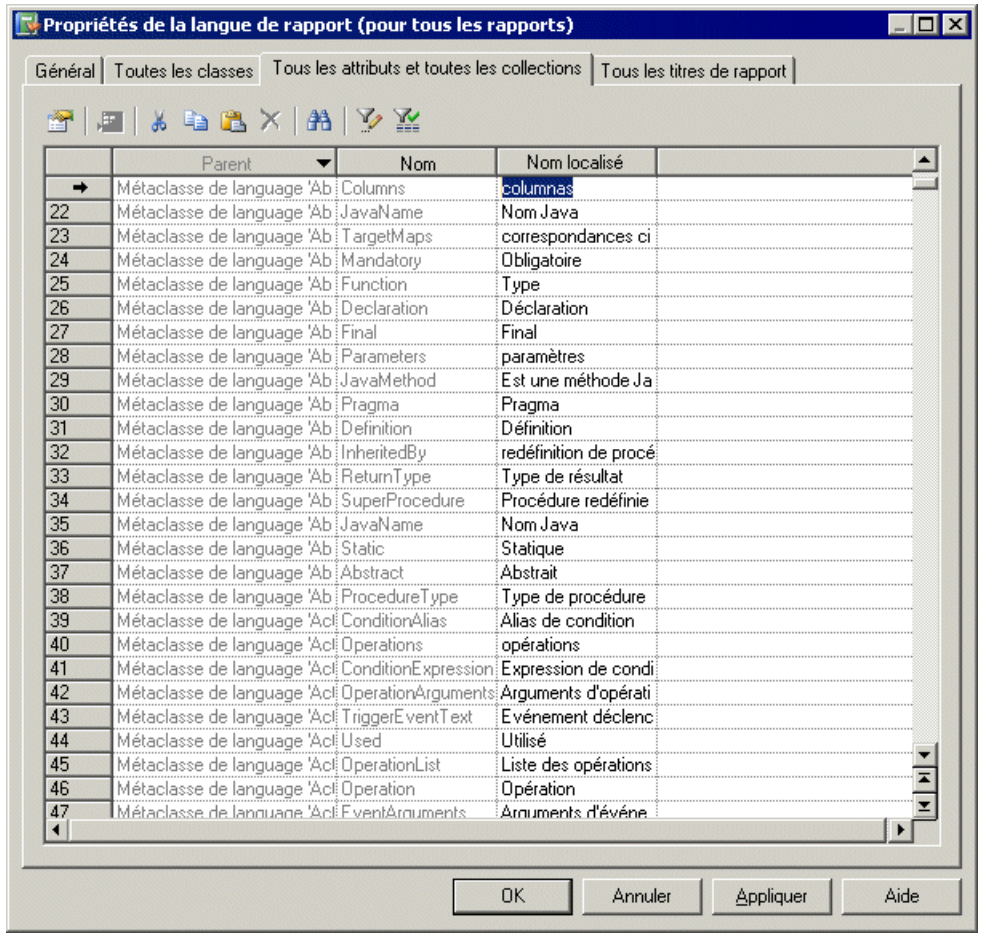
L'onglet Toutes les classes répertorie toutes les métaclasses disponibles dans la catégorie Object Attributes disponibles dans la catégorie Object Attributes de l'onglet Général, mais sa présentation sous forme de tableau rend son utilisation plus simple.



Pour chaque métaclass répertoriée dans la colonne **Nom**, vous devez saisir une traduction dans la colonne **Nom localisé**. Vous pouvez trier la liste pour regrouper les objets dont les noms sont similaires, et traduire simultanément plusieurs éléments identiques en sélectionnant plus 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.

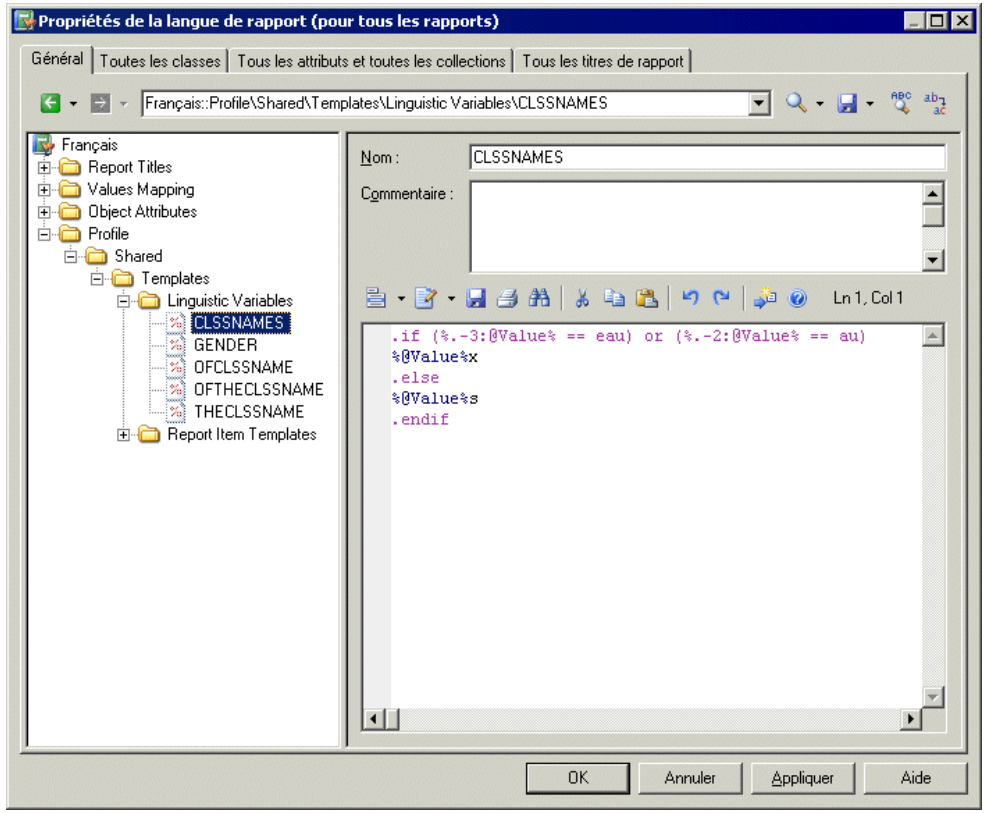


Pour chaque attribut ou collection répertorié dans la colonne **Nom**, vous devez saisir une traduction dans la colonne **Nom localisé**. Vous pouvez trier la liste pour regrouper les objets dont les noms sont similaires, et traduire simultanément plusieurs éléments identiques en sélectionnant plus lignes.

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.

Les règles de grammaire peuvent notamment inclure la forme plurielle d'un nom, ainsi que l'article défini censé le précéder (voir *Catégorie Profile/Report Item Templates* à la page 328).



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 322).

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.

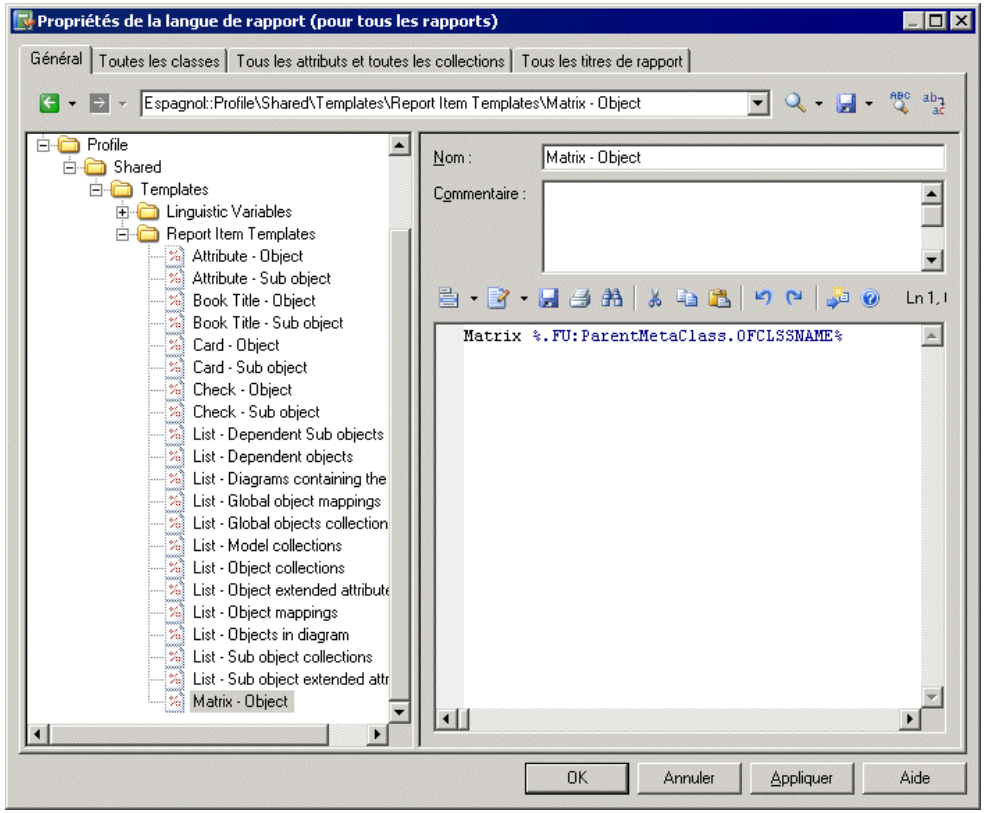
```
.if (%.1U:@Value% == A) or (%.1U:@Value% == E) or (%.1U:@Value% == I)
or (%.1U:@Value% == O) or (%.1U:@Value% == U)
d'%@Value%
```

```
.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.).



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 du nom localisé pour le parent de l'objet, et`%%PARENT%%` est remplacée 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 326).

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.

Remarque : Vous ne pouvez ni créer ni supprimer des templates.

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. De telles opérations peuvent être automatisées à l'aide de scripts.

Vous pouvez lire et modifier n'importe quel objet PowerAMC en utilisant un langage de script tel que Java, VBScript, C#, ou de nombreux autres langages. Dans ce chapitre, nous nous focalisons principalement sur la rédaction de code VBScript destiné à être exécuté dans la boîte de dialogue Edition/Exécution d'un script, mais vous pouvez également appeler des compléments à partir des menus de PowerAMC (voir *Lancement des scripts et de compléments depuis les menus* à la page 365) ou scripter l'application PowerAMC via OLE automation (voir *OLE Automation et compléments* à la page 359).

Le script suivant illustre la syntaxe de base de VBScript appliquée à la manipulation des modèles et objets PowerAMC, ce qui inclut :

- Déclaration d'une variable locale
- Affectation d'une valeur à une variable locale (avec le cas particulier d'un objet)
- Opérateur de condition : `If Then/Else/End If`
- Itération sur une liste : `For Each/Next`
- Définition et appel d'une procédure : `Sub`
- Définition et appel d'une fonction : `Function`
- Gestion d'erreur à l'aide d'instructions `On Error`

```
' Ceci est un commentaire VBScript.
Dim var ' Déclaration d'une variable locale
var = 1 ' Affectation d'une valeur pour un type simple
Set var = ActiveModel ' Affectation d'une valeur pour un objet.
ActiveModel est une propriété globale PowerAMC
If not var is Nothing Then ' Condition sur un objet, testant pour
savoir s'il est 'null'
   Dim objt ' Déclaration d'une autre variable locale
   For Each objt In ActiveModel.Children ' Boucle sur la collection
d'objets enfant
      DescribeObject objt ' Appel de procédure avec objt comme
paramètre (sans parenthèses). La procédure est définie ci-
dessous.
   Next
Else
   output "Il n'y a pas de modèle actif" ' Output est une procédure
PowerAMC qui écrit du texte dans la fenêtre Résultats
End If
```

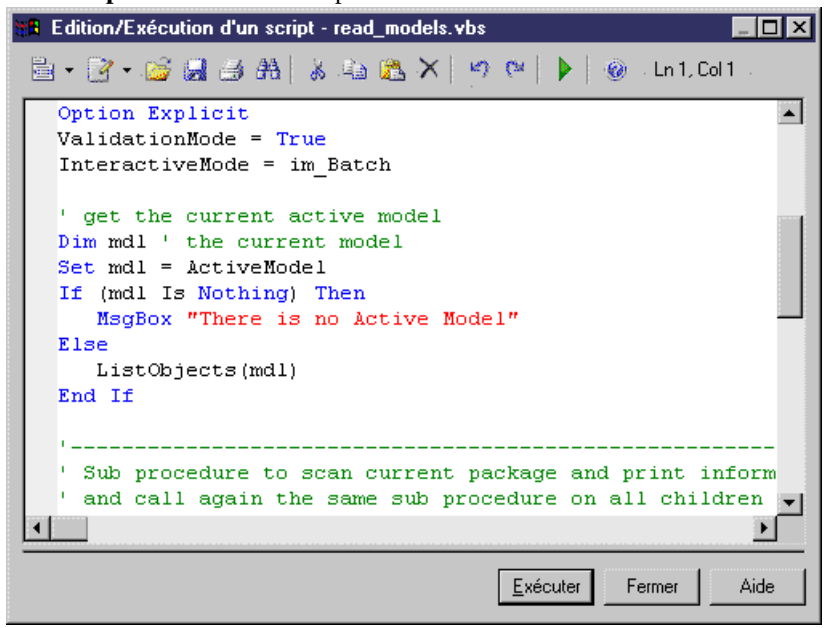
```
' Ceci est une procédure - un méthode qui ne renvoie pas de valeur
Sub DescribeObject(objt)
   Dim desc ' Déclaration de variable dans la procédure
   desc = ComputeObjectLabel(objt) ' Appel de fonction avec objt
comme paramètre (avec parenthèses). La fonction est définie ci-
dessous.
                              ' On récupère la valeur renvoyée par la
fonction dans la variable desc
   output desc ' Affiche la description de l'objet dans le résultat
End Sub

' Ceci est une fonction - une méthode qui renvoie une valeur
Function ComputeObjectLabel(objt)
   Dim label ' Déclaration d'une variable locale pour stocker le
libellé de l'objet
   label = "" ' Initialisation de la variable de libellé avec une
valeur par défaut
   If objt is nothing then
      label = "Aucun objet"
   ElseIf objt.IsShortcut() then ' IsShortcut est une fonction
PowerAMC disponible sur les objets
      label = objt.Name & " (shortcut)" ' Concaténation de deux
chaînes
   Else
     On Error Goto 0 ' Désactive l'abandon d'exécution de script sur
erreur
      label = objt.Name ' Assigne la propriété Nom de l'objet à la
variable locale
      On Error Resume Next ' Réactive l'exécution de script en cas
d'erreur
   End If
   ComputeObjectLabel = label ' La valeur est renvoyée en affectant
une variable implicite avec le même nom que la fonction
End Function
```

Remarque : VBScript peut également être utilisé pour créer des vérifications personnalisées, des gestionnaires d'événements, des transformations et des méthodes dans un fichier d'extension (voir *Chapitre 2, Fichiers d'extension* à la page 11) et être incorporé dans ou appelé depuis de templates de GTL (voir *Macro .execute_vbscript* à la page 291 et *Macro .vbscript* à la page 303).

Les exemples dans ce chapitre sont destinés à introduire les concepts et techniques de base permettant de contrôler PowerAMC à l'aide de script. Pour obtenir une documentation complète sur le métamodèle PowerAMC, sélectionnez **Aide > Aide sur les objets du métamodèle**. Pour une documentation complète sur VBScript, voir le site *Microsoft MSDN*.

Exécution de scripts dans PowerAMC

Vous pouvez exécuter des scripts VBScript dans votre client PowerAMC en sélectionnant **Outils > Exécuter des commandes** pour afficher la boîte de dialogue **Edition/Exécution d'un script**. Le résultat du script s'affiche dans la fenêtre **Résultats**.



Les outils suivants sont disponibles sur la barre d'outils de la boîte de dialogue **Edition/ Exécution d'un script** :

Outils	Description
	**Menu de l'éditeur [Maj+F11]** - Contient les commandes suivantes : • **Enregistrer sous...** - Enregistre le contenu de la zone dans un nouveau fichier. • **Sélectionner tout [Ctrl+A]** - Sélectionne tout le contenu de la zone. • **Suivant... [F3]** - Trouve l'occurrence suivante du texte recherché. • **Précédent... [Maj+F3]** - Trouve l'occurrence précédente du texte recherché. • **Aller à la ligne... [Ctrl+G]** - Ouvre une boîte de dialogue permettant d'aller à la ligne spécifiée. • **Activer/Désactiver le signet [Ctrl+F2]** Insère et supprime un signet (marque bleue) à l'emplacement du curseur. Notez que les signets ne sont pas imprimables et sont perdus si vous réactualisez l'affichage de l'onglet, ou si vous utilisez l'outil • **Nouveau signet [F2]** - Passe au signet suivant. • **Signet précédent [Maj+F2]** - Revient au signet précédent.

Outils	Description
	**Editer avec [Ctrl+E]** - Affiche un aperçu du code dans un éditeur externe. Cliquez sur la flèche vers le bas pour sélectionner un éditeur particulier ou sur **Choisir un programme** pour spécifier un nouvel éditeur. Les éditeurs spécifiés ici sont ajoutés dans la liste des éditeurs disponibles en sélectionnant **Outils > Options générales > Editeurs**.
	**Enregistrer [Ctrl+S]** - Enregistre le contenu de la zone dans le fichier spécifié.
	**Imprimer [Ctrl+P]** - Imprime le contenu de la zone.
	**Rechercher [Ctrl+F]** - Ouvre une boîte de dialogue afin de rechercher un texte.
	
	**Effacer** - Supprime le script dans la boîte de dialogue.
	**Annuler [Ctrl+Z]** et **Répéter [Ctrl+Y]** - Annule ou revalide les modifications. Plusieurs niveaux d'annulation et de répétition sont pris en charge mais, si vous exécutez un script qui modifie les objets dans plusieurs modèles, vous devez utiliser les commandes Annuler et Répéter dans chacun des modèles appelé par le script.
	**Exécuter [F5]** - Exécute le script. Le résultat est affiché dans la fenêtre **Résultats**. Si une erreur de compilation se produit, une boîte de message s'affiche et une brève description de l'erreur apparaît dans le volet **Liste de résultats** de la boîte de dialogue, avec le curseur placé à l'endroit de l'erreur. Vous pouvez intercepter les erreurs en utilisant l'instruction `On Error Resume Next`, sauf si le script est appelé en mode interactif `im_Abort` (voir *Macro .set_interactive_mode* à la page 300).
	**Rechercher dans l'aide sur les objets du métamodèle [Ctrl+F1]** - Affiche le fichier d'aide sur les objets du métamodèle de PowerAMC, qui fournit des informations détaillées sur tous les attributs, collections, et méthodes disponibles pour chaque métaclasse.

Exemples de fichiers VBScript

PowerAMC est fourni avec un jeu d'exemples de script, que vous pouvez utiliser comme base pour créer vos propres scripts, et qui sont situés dans le dossier `VB Scripts` du répertoire d'installation de PowerAMC. Ces scripts sont destinés à vous montrer la variété des tâches que vous pouvez réaliser sur des modèles PowerAMC en utilisant VBScript.

Avertissement ! Vous devez toujours réaliser une copie de sauvegarde des exemples de script avant de les modifier.

Exemple de balayage d'un modèle

Le script suivant parcourt n'importe quel modèle, en bouclant sur les packages et répertoriant les objets qu'ils contiennent :

```
Option Explicit ' Force la déclaration de chaque variable
'avant affectation
InteractiveMode = im_Batch ' Supprime l'affichage de boîtes de
dialogue
' Identifie le modèle actif courant
Dim diag
Set diag = ActiveDiagram ' le diagramme courant
If (diag Is Nothing) Then
 MsgBox "Il n'y a pas de diagramme actif"
Else
 Dim fldr
 Set Fldr = diag.Parent
 ListObjects(fldr)
End If
' Sous-procédures pour parcourir le package courant et imprimer
' des infos sur les objets du package courant et rappeler
' la même sous-procédure sur tous les packages enfant
Private Sub ListObjects(fldr)
 output "Balayage de " & fldr.code
 Dim obj ' objet concerné
 For Each obj In fldr.children
  ' Appel de sous-procédure pour imprimer des infos sur l'objet
  DescribeObject obj
 Next
 ' parcourir les sous-packages
 Dim f ' dossier concerné
 For Each f In fldr.Packages
  'Appel de sous-procédure pour parcourir le package enfant
  ListObjects f
 Next
End Sub
' Sous-procédure pour imprimer des infos sur l'objet courant
Private Sub DescribeObject(CurrentObject)
 if CurrentObject.ClassName ="Association-Class link" then exit sub
 'output "Trouvé "+CurrentObject.ClassName
 output "Trouvé "+CurrentObject.ClassName+" """+CurrentObject.Name
+""", Créé par "+CurrentObject.Creator+" le
"+Cstr(CurrentObject.CreationDate)
End Sub
```

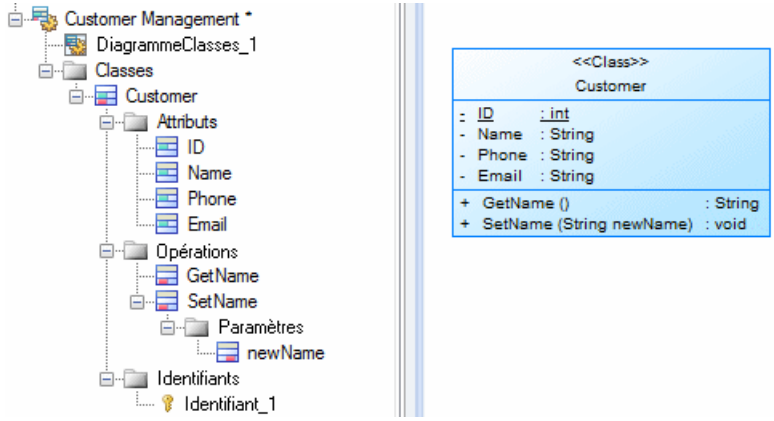
Exemple de création d'un modèle

Le script suivant crée un nouveau MOO, puis crée une classe avec des attributs et des opérations :

```
ValidationMode = True 'Force PowerAMC à valider des
' actions et renvoyer des erreurs en cas d'action interdite
InteractiveMode = im_Batch ' Supprime des boîtes de dialogue PowerAMC
' Fonction princpale
' Crée un MOO avec un diagramme de classes
```

```
Dim Model
Set model = CreateModel(PdOOM.cls_Model, "|Diagram=ClassDiagram")
model.Name = "Gestion clients"
model.Code = "GestionClients"
' Récupère de la diagramme de classes
Dim diagram
Set diagram = model.ClassDiagrams.Item(0)
' Crée les classes
CreateClasses model, diagram
' Fonction de création des classes
Function CreateClasses(model, diagram)
 ' Crée une classe
 Dim cls
 Set cls = model.CreateObject(PdOOM.cls_Class)
 cls.Name = "Client"
 cls.Code = "Client"
 cls.Comment = "Classe client"
 cls.Stereotype = "Class"
 cls.Description = "La classe client définit les attributs et
comportements d'un client."
 ' Création des attributs
 CreateAttributes cls
 ' Création des méthodes
 CreateOperations cls
 ' Création d'un symbole pour la classe
 Dim sym
 Set sym = diagram.AttachObject(cls)
 CreateClasses = True
End Function
' Fonction de création d'attributs
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 = "Nom"
 attr.Code = "Nom"
 attr.DataType = "String"
 attr.Persistent = True
 attr.PersistentCode = "NOM"
 attr.PersistentDataType = "A30"
 Set attr = cls.CreateObject(PdOOM.cls_Attribute)
 attr.Name = "Téléphone"
 attr.Code = "Telephone"
 attr.DataType = "String"
 attr.Persistent = True
 attr.PersistentCode = "TELEPHONE"
 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
' Fonction de création des opérations
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 = "nouvNom"
 param.Code = "nouvNom"
 param.DataType = "String"
 body = "{" + vbCrLf
 body = body + " Name = nouvNom;" + vbCrLf
 body = body + "}"
 oper.Body = body
 CreateOperations = True
End Function
```



Manipulation des modèles, des collections et des objets (Scripting)

Vous pouvez manipuler le contenu d'un modèle en le créant et en l'ouvrant, puis en descendant depuis la racine du modèle dans les collections d'objets. Des propriétés, constantes et fonctions globales sont disponibles dans n'importe quel contexte afin de servir de point d'entrée pour vos scripts.

Les propriétés globales suivantes permettent d'accéder à l'espace de travail et aux modèles qu'il contient :

- `ActiveWorkspace` - Récupère l'espace de travail courant.
- `ActiveModel`, `ActivePackage` et `ActiveDiagram` - Récupère le modèle, le package ou le diagramme correspondant à la vue active.
- `ActiveSelection` - Collection en lecture seule des objets sélectionnés dans le diagramme actif.
- `Models` - Collection en lecture seule des modèles ouverts dans l'espace de travail courant.
- `RepositoryConnection` - Récupère la connexion au référentiel courante (voir *Manipulation du référentiel (Scripting)* à la page 352).

Les fonctions globales suivantes sont couramment utilisées pour créer ou ouvrir des modèles et effectuer des actions sur ces derniers :

- `CreateModel()` et `OpenModel()` - Créent et ouvrent un modèle (voir *Création et ouverture de modèles (Scripting)* à la page 339).
- `Output()` - Imprime du texte dans l'onglet **Script** de la fenêtre **Résultats** de PowerAMC.
- `IsKindOf()` - Teste la métaclasse de l'objet.
- `ExecuteCommand()` - Lance une application externe.
- `EvaluateNamedPath()` et `MapToNamedPath()` - Gèrent les chemins nommés dans les fichiers de modèle.
- `BeginTransaction()`, `CancelTransaction()` et `EndTransaction()` - Démarrent, annulent et valident des transactions.

Les constantes globales suivantes fournissent des informations sur l'instance de PowerAMC :

- `UserName` - Récupère le nom de connexion de l'utilisateur.
- `Version` - Récupère la version de PowerAMC.
- `HomeDirectory` - Récupère le répertoire racine de l'application.
- `RegistryHome` - Récupère le chemin de la racine de l'application dans le registre.

- `Viewer` - Renvoie True si l'application est une version Visionneuse dotée de fonctionnalités limitées.
- `ValidationMode` - Par défaut, PowerAMC effectue différentes vérifications pour valider vos actions et émet une erreur en cas d'action interdite. Vous pouvez définir `ValidationMode = False` (qui débranche les règles de validation telles que celles portant sur l'unicité du nom ou celles s'assurant que les liens ont leurs extrémités définies) pour améliorer les performances ou si votre algorithme requiert temporairement un état invalide.
- `InteractiveMode` - Spécifie le niveau d'interaction requis. Vous pouvez choisir l'une des valeurs suivantes :
  - `im_Batch` [défaut] - Supprime l'affichage des boîte de dialogue et utilise systématiquement les valeurs par défaut. Par exemple, si votre modèle contient des raccourcis externes et que le modèle cible pour les raccourcis est fermé, ce mode va automatiquement ouvrir le modèle sans interaction de l'utilisateur.
  - `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` - Supprime les boîtes de dialogue et abandonne l'exécution si une boîte de dialogue est rencontrée.
- `ShowMode` [spécifique-OLE] - Vérifie ou modifie la visibilité de la fenêtre principale de l'application. Renvoie `True` si la fenêtre principale de l'application est visible et n'est pas réduite.
- `Locked` [spécifique-OLE] - Lorsque définie à `True`, s'assure que l'application continue à être exécutée après déconnexion du client OLE.

Pour obtenir des informations détaillées sur toutes les propriétés, fonctions et constantes globales, sélectionnez **Aide > Aide sur les objets du métamodèle** puis sélectionnez `Basic Elements`.

Création et ouverture de modèles (Scripting)

Vous créez des modèles et ouvrez des modèles existants en utilisant les fonctions globales `CreateModel()` et `OpenModel()`. Le modèle actif est accessible via les propriété globale `ActiveModel`et ceux ouverts dans l'espace de travail sont disponibles via la collection globale `Models`.

Ce script crée un nouveau MOO ayant comme cible le langage Analysis, y crée des classes, les affiche dans le diagramme, puis enregistre le modèle et le referme :

```
Dim NouveauModele
set NouveauModele = CreateModel(PdOOM.Cls_Model, "Language=Analysis|
Diagram=ClassDiagram|Copy")
If NouveauModele is Nothing then
 msgbox "Impossible de créer le modèle", vbOkOnly, "Error" ' Affiche
un message d'erreur
Else
 output "Le modèle UML a été créé" ' Affichage d'un message dans la
fenêtre Résultats
```

```
 NouveauModele.SetNameAndCode "MonMOO", "MonMOO" 'Initialisation des
nom et code du modèle
 For idx = 1 to 12 'Création et affichage des classes
  Set obj=NouveauModele.Classes.CreateNew()
  obj.SetNameAndCode "C" & idx, "C" & idx
  Set sym=ActiveDiagram.AttachObject (obj)
 Next
 ActiveDiagram.AutoLayoutWithOptions(2)
 NouveauModele.Save "c:\temp\MonMOO.mom" ' Enregistrement du modèle
 NouveauModele.Close ' Fermeture du modèle
 Set NouveauModele = Nothing ' Libération de la dernière référence à
l'objet pour libérer de la mémoire
End If
```

Ce script vérifie que le modèle qui vient d'être créé existe bien, et l'ouvert dans l'espace de travail :

```
Dim MonModele, FileName
FileName = "c:\temp\MonMOO.moo"
On Error Resume Next ' Omission du message d'erreur de scripting
générique
Set MonModele = OpenModel(FileName)
If MonModele is nothing then ' Affichage d'un message d'erreur
 msgbox "Impossible d'ouvrir le modèle :" + vbCrLf + FileName,
vbOkOnly, "Error"
Else ' Affichage d'un message dans la fenêtre Résultats
 output "Le MOO a été ouvert."
End If
```

Consultation et modification des collections (Scripting)

La plupart de la navigation dans le métamodèle est effectuée en descendant depuis la racine du modèle via les collections d'objets vers les collections de sous-objets ou les objets associés. Un MOO contient une collection de classes et les classes contiennent des collections d'attributs et d'opérations. Vous pouvez obtenir des informations sur les membres d'une collection et naviguer sur eux par le biais de script, mais aussi ajouter, retirer et déplacer des objets dans la collection.

Pour parcourir les membres d'une collection, naviguez jusqu'à l'objet parent, puis utilisez une boucle `For each`. Ce script imprime les noms de toutes les tables contenues dans un MPD ouvert.

```
Dim MonModele
Set MonModele=ActiveModel
For each t in MonModele.Tables
    Output "* " & t.Name
Next
```

Lorsque vous parcourez une collection, les objets de la collection et les éventuels raccourcis vers des objets sont renvoyés indifféremment.

Remarque : Pour plus d'informations sur l'accès aux collections définies dans des extensions, voir *Création et utilisation d'extensions (Scripting)* à la page 355.

Les types de collection suivants sont affichés dans le métamodèle :

- Compositions - contiennent des objets qui seront supprimés si le parent est supprimé. Par exemple, les collections `PdPDM/Tables` et `PdPDM/Table/Columns` sont des compositions.
- Agrégations - référence des objets qui vont continuer à exister si leur parent est supprimé. Par exemple, la collection `PdCommon/NamedObject/AttachedRules` (héritée par la plupart des objets) est une agrégation.
- Collections non ordonnées - contient des objets sans ordre significatif. Par exemple, la collection `PdCDM/Entity/Relationships` n'est pas ordonnée.
- Collections ordonnées - contient des objets dans un ordre spécifié par l'utilisateur. Par exemple, la collection `PdPDM/Table/Columns` est ordonnée.
- Collections en lecture seule - peuvent uniquement être parcourues. Par exemple, la collection globale `Models` (tous les modèles ouverts) est en lecture seule.

Les propriétés suivantes sont disponibles pour toutes les collections :

- `Count` - Récupère le nombre d'objets contenus dans une collection.
- `Item[(`**index**`)]` - Récupère l'élément spécifié dans la collection comme objet. `Item(0)` est le premier objet (et l'objet par défaut) et `Item(-1)` est le dernier objet.
- `MetaCollection` - Récupère la métadéfinition de la collection comme un objet.
- `Kind` - Récupère le type des objets que la collection peut contenir.
- `Source` - Récupère l'objet sur lequel la collection est définie.

Les méthodes suivantes sont disponibles pour modifier les collections éditables :

- `CreateNew([`**typeobjet**`]` et `CreateNewAt(`**index**`[,`**typeobjet**`])` - [compositions uniquement] Crée un nouvel objet à la fin de la collection ou à l'**index** spécifié (par défaut, -1). Le paramètre **typeobjet** (par exemple, `PdPDM.cls_Table`) est pertinent uniquement si la collection prend en charge plusieurs sortes d'objets.
- `Add(`**objet**`)` - Insère l'**objet** spécifié à la fin de la collection.
- `Insert([`**index**`][, `**objet**`])` - Insère l'**objet** spécifié dans la collection à l'**index** de position spécifié (par défaut, -1).
- `Move(`**index2, index1**`)` - Déplace l'objet de la position **index1** à la position **index2** dans la collection.
- `Remove(`**objet**`[, delete = y|n])` et `RemoveAt([`**index**`][, delete = y|n])` - Retire l'**objet** spécifié ou l'objet situé à l'**index** (par défaut, -1) spécifié de la collection. Dans le cas des agrégations, vous pouvez également spécifier la suppression de l'objet (les objets retirés d'une composition sont toujours supprimés).
- `Clear([delete = y|n])` - Retire tous objets de la collection et peut également les supprimer.

Le script suivant :

- Crée un MPD,
- Crée des objets dans les collections de composition non ordonnées `Tables` et `BusinessRules`, et
- Ajoute certains objets à la collection d'agrégation ordonnée de `T1` nommée `AttachedRules`, puis manipule cette collection :

```
Dim MonModele, t, r, sym
set MonModele = CreateModel(PdPDM.Cls_Model,"DBMS=SYASA12")
MonModele.SetNameAndCode "MonMPD" , "MonMPD"
'Créer des tables des règles
For idx = 1 to 12
   Set t=MonModele.Tables.CreateNew()
   t.SetNameAndCode "T" & idx, "T" & idx
   Set sym=ActiveDiagram.AttachObject (t)
   Set r=MonModele.BusinessRules.CreateNew()
   r.SetNameAndCode "RG" & idx, "RG" & idx
Next
ActiveDiagram.AutoLayoutWithOptions(2)
'Attacher des règles à Table 1
   Dim MaTable
   Set MaTable=MonModele.FindChildByName("T1",cls_table)
   For idx = 1 to 10
      MaTable.AttachedRules.Add(MonModele.FindChildByName("RG" &
(idx),cls_businessrule))
   Next
'Imprimer la liste de règles attachées à Table 1
Output "Règles attachées à T1 (" & MaTable.AttachedRules.Count & ")"
For each r in MaTable.AttachedRules
    Output "* " & r.Name
Next
'Modifie les règles attachées par insertion, déplacement et
suppression
MaTable.AttachedRules.Insert 3,
MonModele.FindChildByName("RG12",cls_businessrule)
MaTable.AttachedRules.Move 5,0
MaTable.AttachedRules.Remove(MonModele.FindChildByName("RG6",cls_bu
sinessrule))
'Imprimer la liste de règles modifiée
Output "Règles modifiées attachées à T1 (" &
MaTable.AttachedRules.Count & ")"
For each r in MaTable.AttachedRules
    Output "* " & r.Name
Next
```

Accès et modification des objets et propriétés (Scripting)

Vous pouvez utiliser le script pour accéder à n'importe quel objet ou propriété d'objet et les modifier. Les objets n'incluent pas uniquement des objets de modélisation courants (tels que les tables, les classes, les processus et les colonnes), mais également les diagrammes et symboles, ainsi que les objets fonctionnels (tels qu'un rapport ou référentiel). Un objet appartient à une métaclasse du métamodèle de PowerAMC et hérite des propriétés, collections et méthode de cette métaclasse.

Les objets racine, tels que les modèles, sont accessibles via des propriétés et fonctions (voir *Manipulation des modèles, des collections et des objets (Scripting)* à la page 338), tandis que les objets standard sont accessibles en parcourant les collections (voir *Consultation et modification des collections (Scripting)* à la page 340) ou individuellement via les méthodes suivantes :

- `FindChildByName(`**"Nom", Type**`[, ParamètresFacultatifs]`
- `FindChildByCode(`**"Code", Type**`[, ParamètresFacultatifs]`
- `FindChildByPath(`**"Chemin", Type**`[, ParamètresFacultatifs]`

Les paramètres suivants sont disponibles :

Paramètre	Description
Nom / **Code** / **Chemin**	Spécifie le nom ou le code de l'objet à trouver, ou son chemin d'accès. Par exemple, pour trouver la colonne `Adresse` dans la table `Client` du package `Ventes` à partir du contexte du noeud de modèle, vous pouvez chercher par le nom `Adresse` ou par le chemin `Ventes/Client/Adresse`.
Type	Spécifie la métaclasse de l'objet à trouver sous la forme `cls_`*`NomPublic`*. Par exemple, pour trouver une colonne, sélectionnez `cls_Column`. Ces ID de métaclasse sont uniques dans leur bibliothèque de modèle, mais, dans des cas comme les packages, qui peuvent apparaître dans plusieurs types de modèles, vous devez préfixer l'ID à l'aide du nom du module (`PdOOM.cls_Package`). Lorsque vous créez un modèle, vous devez utiliser le préfixe de module (par exemple `PdPDM.cls_Model`).
ParamètresFacultatifs	Les paramètres suivants sont facultatifs : • **"Stereotype"** - Spécifie que l'objet à trouver doit porter le stéréotype spécifié. • **"LastFound"** - Spécifie que la recherche doit commencer après cet objet. Ce paramètre peut s'avérer particulièrement utile dans le cas où plusieurs objets ont le même chemin. Dans ce cas, la recherche peut être lancée dans une boucle while qui utilise la correspondance précédente comme paramètre LastFound (dernier trouvé). • **CaseSensitive=y|n** - [défaut : y] Spécifie que la recherche tient compte de la casse. • **IncludeShortcuts** - [défaut : n] Spécifie que les raccourcis peuvent être trouvés. • **UseCodeInPlaceOfName** - [ByPath, défaut : n] Spécifie que l'objet peut être trouvé par son code (Default=n). • **PathSeparator** - [ByPath, défaut= /, \ ou::)] Spécifie le caractère pour séparer les noeuds dans le chemin.

Vous pouvez obtenir les valeurs d'attribut standard en utilisant la notation avec point (**objet . attribut**) ou à l'aide d'une des méthodes suivante :

- `GetAttribute(`**"attribut"**`)` - récupère la valeur stockée pour l'attribut

- `GetAttributeText("attribut")` - récupère la valeur affichée pour l'attribut

Vous pouvez définir des valeurs d'attribut en utilisant la notation avec point (**objet . attribut**=**valeur**) ou à l'aide d'une des méthodes suivante :

- `SetAttribute "attribut", valeur`
- `SetAttributeText "attribut", "valeur"`

Remarque : Pour plus d'informations sur l'obtention et la définition de valeurs d'attributs étendus, voir *Création et utilisation d'extensions (Scripting)* à la page 355

Le script suivant ouvre un MOO exemple, trouve une classe par nom et un paramètre par chemin, puis imprime et modifie certaines de leurs propriétés :

```
Dim MonModele, C, P
'Ouverture du fichier de modèle
Set MonModele=OpenModel(EvaluateNamedPath("%_EXEMPLES%\" & "UML2
Sample.moo"))
'Obtention de la classe et du paramètre
Set C=MonModele.FindChildByName("OrderManager",cls_Class)
Set P=MonModele.FindChildByPath("SecurityManager/CheckPassword/
login",PdOOM.cls_Parameter)

'Impression des valeurs initiales
Output "Valeurs initiales :"
PrintProperties C, P
'Modification des valeurs
C.Comment="Cette classe contrôle les commandes."
C.SetAttributeText "Visibility", "private"
P.Name="LoginName"
'Impression des valeurs modifiées
Output "Valeurs modifiées :"
PrintProperties C, P

'Procédure d'impression des valeurs
Sub PrintProperties(MyClass, MyParam)
 output "Classe : " & MyClass.Name
 output vbTab & "Commentaire : " & MyClass.Comment
 output vbTab & "Visibilité : " &
MyClass.GetAttributeText("Visibility")
 output vbTab & "Persistance : " &
MyClass.GetAttributeText("PersistentGenerationMode")
 output "Paramètre : " & MyParam.Parent & "." & MyParam.Name
 output vbTab & "Type de données : " & MyParam.DataType
 output vbTab & "Type de paramètre : " &
MyParam.GetAttributeText("ParameterType")
End Sub
```

Création d'objets (Scripting)

Il est recommandé de créer un objet directement à partir de la collection sous l'objet parent en utilisant la méthode `CreateNew()`. La méthode CreateObject(*type*) est également disponible sur les objets de modèle.

Ce script crée une classe dans un MOO, définit certaines des ses propriétés, puis crée un attribut sous la classe, dans chaque cas la création des objets s'effectue dans des collections :

```
Dim MonModele
Set MonModele = ActiveModel
Dim MaClasse
' Création d'une classe
Set MaClasse = MonModele.Classes.CreateNew()
If MaClasse is nothing Then
  ' Affichage d'un message d'erreur
   msgbox "Impossible de créer une classe", vbOkOnly, "Error"
Else
  output "La classe a été créée."
  ' Définition des attributs Nom, Code, Sétéréotype et Finale
  MaClasse.SetNameAndCode "Client", "cli"
  MaClasse.Comment = "Créée par script"
  MaClasse.Stereotype = "MonStereotype"
  MaClasse.Final = true
  ' Création d'un attribut dans la classe
  Dim MonAttr
  Set MonAttr = MaClasse.Attributes.CreateNew()
  If not MonAttr is nothing Then
   output "L'attribut a été créé."
   MonAttr.SetNameAndCode "Name", "custName"
   MonAttr.DataType = "String"
  ' Redéfinition de la variable pour éviter les fuites de mémorie
  End If
 End If
```

Vous pouvez également créer des objets en utilisant la méthode `CreateObject(`*`type`*`)`. Ce script crée une classe dans un MOO et définit certaines de ses propriétés :

```
Dim MonModele
Set MonModele = ActiveModel
Dim MaClasse
' Création d'une classe
Set MaClasse = MonModele.CreateObject(cls_Class)
MaClasse.SetNameAndCode "Autre classe", "Classe2"
MaClasse.Comment = "Créée par CreateObject"
```

Lorsque vous créez un objet lien, vous devez définir ses extrémités. Ce script crée deux classes et les joint au moyen d'un lien d'association :

```
Dim MonModele
Set MonModele = ActiveModel
Dim MaPremiereClasse, MaSecondeClasse, MyAssociation
' Create classes
Set MaPremiereClasse = MonModele.Classes.CreateNew()
MaPremiereClasse.SetNameAndCode "Classe1", "C1"
```

```
Set MaSecondeClasse = MonModele.Classes.CreateNew()
  MaSecondeClasse.SetNameAndCode "Classe2", "C2"
' Création d'une association
Set MonAssociation = MonModele.Associations.CreateNew()
MonAssociation.Name = "A1"
' Définition de ses extrémités
Set MonAssociation.Object1 = MaPremiereClasse
Set MonAssociation.Object2 = MaSecondeClasse
```

Affichage, mise en forme et positionnement des symboles (Scripting)

Lorsque vous créez un objet, celui-ci ne s'affiche pas dans un diagramme, sauf si vous utilisez la méthode `AttachObject()` ou `AttachLinkObject()`. Les symboles sont des objets de plein droit qui ne sont accessibles que via des collections sur l'objet ou diagramme parent. Vous pouvez positionner un symbole en utilisant la méthode `Position()` et changer son format en utilisant `LineWidth` et d'autres attributs de format.

Le script suivant crée deux classes, les joint par un lien, et affiche les trois symboles dans le diagramme actif :

```
Dim MonModele, MonDiagramme, C1, C2, A1 Set MonModele = ActiveModel
Set MonDiagramme =
    ActiveDiagram ' Création des classes Set C1 =
MonModele.Classes.CreateNew() C1.SetNameAndCode
    "C1", "C1" Set C2 = MonModele.Classes.CreateNew()
C2.SetNameAndCode "C2", "C2" '
    Affichage de symboles de classe MonDiagramme.AttachObject(C1)
MonDiagramme.AttachObject(C2) '
    Création de l'association Set A1 =
MonModele.Associations.CreateNew() A1.SetNameAndCode =
   "A1", "A1" ' Définition de ses extrémités Set A1.Object1 = C1 Set
A1.Object2 = C2 '
    Affichage du symbole d'association
MonDiagramme.AttachLinkObject(A1)
```

Le script suivant crée un MAE et quatre zones d'architecture, les aligne dans un carré, et met en forme la partie supérieure gauche :

```
Dim MouveauModele, idx, obj, sym
set MouveauModele = CreateModel(PdEAM.Cls_Model,
"Diagram=CityPlanningDiagram")
MouveauModele.SetNameAndCode "MonMAE" , "MonMAE"
For idx = 1 to 4
   Set obj=MouveauModele.ArchitectureAreas.CreateNew()
   obj.SetNameAndCode "A" & idx, "A" & idx
   Set sym=ActiveDiagram.AttachObject (obj)
   sym.width=30000
   sym.height=20000
Next
dim A1, A2, A3, A4, X1, Y1
set A1 =
MouveauModele.FindChildByName("A1",cls_architecturearea).Symbols.It
em(0)
set A2 =
MouveauModele.FindChildByName("A2",cls_architecturearea).Symbols.It
```

```
em(0)
set A3 =
MouveauModele.FindChildByName("A3",cls_architecturearea).Symbols.It
em(0)
set A4 =
MouveauModele.FindChildByName("A4",cls_architecturearea).Symbols.It
em(0)
X1 = A1.Position.X
Y1 = A1.Position.Y
' Déplacement des symboles pour les rendre adjacents
A2.Position = NewPoint(X1 + A1.Width, Y1)
A3.Position = NewPoint(X1, Y1 - A1.Height)
A4.Position = NewPoint(X1 + A1.Width, Y1 - A1.Height)
A1.DashStyle = 2
A1.LineWidth = 3
```

Suppression d'objets (Scripting)

Vous pouvez supprimer des objets en utilisant la méthode `Delete`.

Le script suivant crée un nouveau MCD, le remplit avec des entités et des relations, puis supprime l'entité `E5` et la relation `R8`:

```
Dim MonModele, obj, sym, idx
set MonModele = CreateModel(PdCDM.Cls_Model)
MonModele.SetNameAndCode "MonMCD" , "MonMCD"
'Création des entités
For idx = 1 to 12
   Set obj=MonModele.Entities.CreateNew()
   obj.SetNameAndCode "E" & idx, "E" & idx
   Set sym=ActiveDiagram.AttachObject (obj)
Next
'Création des relations relations
For idx = 2 to 11
   Set obj=MonModele.Relationships.CreateNew()
   obj.SetNameAndCode "R" & idx-1, "R" & idx-1
   Set obj.Object1 = MonModele.FindChildByName("E" &
(idx-1),cls_entity)
   Set obj.Object2 = MonModele.FindChildByName("E" & (idx
+1),cls_entity)
   Set sym=ActiveDiagram.AttachLinkObject (obj)
Next
ActiveDiagram.AutoLayoutWithOptions(2)
'Suppression des objets
MonModele.FindChildByName("E5",cls_entity).Delete
MonModele.FindChildByName("R8",cls_relationship).Delete
```

Création d'une sélection d'objets (Scripting)

Vous pouvez créer une sélection d'objets en utilisant la méthode `CreateSelection()`. Vous pouvez effectuer des actions sur la sélection, comme par exemple changer leurs propriétés ou leur format, ou les déplacer dans un autre package.

Le script suivant crée un MPD, le remplit à l'aide de table et effectue une sélection de tables qu'il déplace dans un package :

```
Dim MonModele, obj, sym
set MonModele = CreateModel(PdPDM.Cls_Model,"DBMS=SYASA12")
MonModele.SetNameAndCode "MonMPD" , "MonMPD"
'Création des tables
For idx = 1 to 12
   Set obj=MonModele.Tables.CreateNew()
   obj.SetNameAndCode "T" & idx, "T" & idx
   Set sym=ActiveDiagram.AttachObject (obj)
Next
ActiveDiagram.AutoLayoutWithOptions(2)
'Création de package
Dim MonPackage
Set MonPackage=MonModele.Packages.CreateNew()
MonPackage.SetNameAndCode "P1", "P1"
ActiveDiagram.AttachObject (MonPackage)
'Création d'une sélection
Dim MaSelection
Set MaSelection = ActiveModel.CreateSelection
For idx = 1 to 5
   MaSelection.Objects.Add(MonModele.FindChildByName("T" &
(idx*2),cls_table))
Next
'Déplacement de la sélection dans le package
MaSelection.MoveToPackage(MonPackage)
```

Pour ajouter toutes les tables dans la sélection, utilisez la méthode `AddObjects` :

```
MaSelection.AddObjects MonModele,cls_table
```

Pour retirer un objet de la sélection, utilisez la méthode `Remove` :

```
MaSelection.Objects.Remove(MonModele.FindChildByName("T6",cls_table
))
```

Contrôle de l'espace de travail (Scripting)

Vous pouvez accéder à l'espace de travail courant en utilisant la propriété globale `ActiveWorkspace`, ouvrir, enregistrer et fermer des espaces de travail, et y ajouter des dossiers et documents.

Le script suivant construit une structure de dossiers simple dans un espace de travail et ajoute et crée plusieurs modèles dans cet espace de travail :

```
Option Explicit
' Fermeture de l'espace de travail existant et enregistrement dans
Temp
Dim workspace, curentFolder
Set workspace = ActiveWorkspace
workspace.Load "%_EXEMPLES%\monwsp.sws"
Output "Enregistrement de l'espace de travail existant dans le
répertoire Exemple : "+EvaluateNamedPath("%_EXEMPLES%\temp.sws")"
workspace.Save "%_EXEMPLES%\Temp.SWS"
workspace.Close
workspace.Name = "VBS WSP"
workspace.FileName = "VBSWSP.SWS"
workspace.Load "%_EXEMPLES%\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
'insertion du dossier à la racine
 Set subfolder =
workspace.Children.CreateNew(PdWsp.cls_WorkspaceFolder)
 subfolder.name = "Nouvdossier(VBS)"
 'insertion du dossier à la racine en pos 6
 Set subfolder = workspace.Children.CreateNewAt(5,
PdWsp.cls_WorkspaceFolder)
 subfolder.name = "Nouvdossier(VBS)insertedAtPos5"'
   ' ajout d'un dossier dans ce dossier
 Set subfolder =
subfolder.Children.CreateNew(PdWsp.cls_WorkspaceFolder)
 subfolder.name = "NouvSousDossier(VBS)"
 subfolder.AddDocument EvaluateNamedPath("%_EXEMPLES%\rapmpd.rtf")
 subfolder.AddDocument EvaluateNamedPath("%_EXEMPLES%\rapmcd.rtf")
 subfolder.AddDocument EvaluateNamedPath("%_EXEMPLES%\gestsoc.mpd")
 subfolder.AddDocument EvaluateNamedPath("%_EXEMPLES%\demo.moo")
 dim lastmodel
 set lastmodel = subfolder.AddDocument
(EvaluateNamedPath("%_EXEMPLES%\Ordinateurs.mlb"))
 lastmodel.open
 lastmodel.name = "Ordinateurs"
 lastmodel.close
 'détachement du modèle de l'espace de travail
 lastmodel.delete
workspace.Save "%_EXEMPLES%\Final.SWS"
```

Pour plus d'informations sur les propriétés et méthodes disponibles sur l'espace de travail, sélectionnez **Aide > Aide sur les objets du métamodèle**, puis naviguez vers `Libraries/PdWSP/Workspace`.

Création de raccourcis (Scripting)

Vous pouvez créer un raccourci dans un modèle en utilisant la méthode `CreateShortcut()`.

Le script suivant agit sur un MOO et crée un raccouci vers la classe `C1` du package `P1` dans le package `P2` :

```
Dim obj, shortcut, recipient
' Récupération de la classe cible du raccourci
Set obj = ActiveModel.FindChildByPath("P1/C1",cls_Class)
```

```
' Récupération du package dans lequel créer le raccourci
Set recipient = ActiveModel.FindChildByPath("P2",PdOOM.cls_Package)
' Création du raccourci
Set shortcut = obj.CreateShortcut(recipient)
If not shortcut is nothing then
  output "Le raccourci de classe a été créé"
End If
```

Le script suivant crée un raccourci vers la classe `C1` du modèle `O1` package `P1` directement sous le modèle `O2` :

```
Dim targetmodel, usingmodel, obj, shortcut
For each m in Models
   Output m.Name
   If m.Name="O1" then 'Récupération du modèle contenant l'objet
cible du raccourci à créer
      Set targetmodel=m
   End If
   If m.Name="O2" then 'Récupération du modèle dans lequel créer le
raccourci
      Set usingmodel=m
   End If
Next
' Récupération de l'objet vers lequel établir un raccourci
Set obj = targetmodel.FindChildByPath("P1/C1",cls_Class)
' Création du raccourci
Set shortcut = obj.CreateShortcut(receivingmodel)
If not shortcut is nothing then
  output "Le raccourci de classe a été créé"
End If
```

Création de correspondances entre objets (Scripting)

Vous pouvez créer des sources de données dans un modèle et à partir de là créer des correspondances entre des objets source contenus dans d'autres modèles et des objets contenus dans le premier modèle à l'aide de scripts.

Le script suivant crée un MOO et un MPD, les remplit à l'aide de classes et de tables, puis crée une source de données dans le MOO, y associe le MPD et crée des correspondances :

```
Dim MonMOO, MonMPD
'Création d'un MOO et d'un MPD
set MonMOO = CreateModel(PdOOM.Cls_Model, "|Language=Analysis|
Diagram=ClassDiagram|Copy")
MonMOO.SetNameAndCode "MonMOO", "MOO"
set MonMPD = CreateModel(PdPDM.Cls_Model, "|DBMS=Sybase SQL Anywhere
12|Copy")
MonMPD.SetNameAndCode "MonMPD", "MPD"
 'Création des classes et des tables
 For idx = 1 to 6
   Set c=MonMOO.Classes.CreateNew()
   c.SetNameAndCode "Classe" & idx, "C" & idx
   Set t=MonMPD.Tables.CreateNew()
   t.SetNameAndCode "Table" & idx, "T" & idx
```

```
Next
'Création d'une source de données dans le MOO et ajout du MPD comme
source
Dim ds, m1
Set ds = MonMOO.DataSources.CreateNew()
ds.SetNameAndCode "MonMPD", "MPD"
ds.AddSource MonMPD

'Création d'une correspondance entre C1 et T6
set m1 =
ds.CreateMapping(MonMOO.FindChildByName("Classe1",cls_class))
m1.AddSource MonMPD.FindChildByName("Table6",cls_table)
' Identification des correspondances pour chaque classe du MOO
 For each c in MonMOO.Classes
   Dim m, sc
   set m = ds.GetMapping(c)
   If not m is nothing then
      Output c.Name & vbtab & "Correspond à : "
      for each sc in m.SourceClassifiers
         output vbtab & vbtab & "- " & sc.Name
      next
   Else
      Output c.Name & vbtab & "Aucune correspondance définie."
   End if
Next
```

Pour plus d'informations sur les correspondances d'objets, voir *Guide des fonctionnalités générales > Liaison et synchronisation de modèles > Mise en correspondance d'objets*.

Création et génération de rapports (Scripting)

Vous pouvez créer un rapport, parcourir son contenu et le générer au format HTML ou RTF en utilisant le script.

Pour créer un rapport, utilisez la méthode `CreateReport()` sur un modèle. Par exemple :

```
Dim model
Set model = ActiveModel
model.CreateReport("MonRapport")
```

Pour parcourir les rapports dans un modèle, utilisez la collection `Reports`. Par exemple :

```
Dim model
Set model = ActiveModel
For each m in model.Reports
    Output m.Name
Next
```

Pour générer un rapport sous forme RTF ou HTML, utilisez la méthode `GenerateRTF()` ou `GenerateHTML()` :

```
set m = ActiveModel
For each r in m.Reports
 filename = "C:\temp\" & r.Name & ".htm"
```

```
 r.GenerateHTML (filename)
Next
```

Manipulation du référentiel (Scripting)

Vous pouvez établir une connexion au référentiel et y consolider ou en extraire des documents, et procéder à l'itération des dernières versions des documents de référentiel via l'objet `RepositoryConnection`. Vous pouvez gérer les dossiers et branches de référentiel ainsi que les serveurs LDAP et SMTP de même que la politique de mot de passe, mais vous ne pouvez pas manipuler les utilisateurs et groupes de référentiel.

Le script suivant ouvre une connexion au référentiel, crée un nouveau MPD et le consolide, puis boucle sur la création de tables, et des consolidations supplémentaires, avant de refermer la connexion :

```
Dim rc
Set rc = RepositoryConnection
rc.Open "NOMREFERENTIEL", "UTILISATEUR", "MOTPASSE",
"UTILBASEDONNES", "MOTPASSEBASEDONNEES"
Output "Avant consolidation"
ListChildren rc
Dim NouveauModele
Set NouveauModele = CreateModel(PdPDM.Cls_Model, "|
Language=SYASIQ1540")
NouveauModele.Name = "Mon MPD"
NouveauModele.ConsolidateNew rc
For i = 1 to 5
   For j = 1 to 5
      NouveauModele.Tables.CreateNew()
   Next
   NouveauModele.Consolidate
Next
Output "Après consolidation"
ListChildren rc
rc.Close

Sub ListChildren(rc)
For each c in rc.ChildObjects
  Output c.Name & "(Modifié le : " & c.ModificationDateInRepository
& ")"
Next
End Sub
```

Pour extraire un modèle, utilisez la méthode `CheckOut`.

Pour obtenir des informations détaillées sur les membres, collections et méthodes disponibles pour le scripting du référentiel, sélectionnez **Aide > Aide sur les objets du métamodèle**, puis naviguez vers `Libraries/PdRMG`.

Génération d'une base de données (Scripting)

Vous pouvez générer un MPD sous la forme d'un script SQL ou directement dans la base via une connexion directe en utilisant la méthode `GenerateDatabase()`. Vous pouvez générer des données de test à l'aide de la méthode `GenerateTestData()`.

Le fragment de script suivant ouvre un MPD, puis appelle des procédures pour générer différents scripts :

```
Dim GenDir, MonModele
    GenDir = "C:\temp\"
    Set MonModele=OpenModel(EvaluateNamedPath("%_EXEMPLES%\" &
"gestsoc.mpd"))

    GenerateDatabaseScripts MonModele 'Génère un script SQL pour
créer la base de données
    ModifyModel MonModele 'Modifie chaque table dans le modèle
    GenerateAlterScripts MonModele - Génère des scripts alter pour
modifier la base de données
    GenerateTestDataScript MonModele - Génère des données de test à
charger dans la base de données
```

Cette procédure génère un script SQL pour créer la base de données :

```
Sub GenerateDatabaseScripts(m)
   Dim opts
   Set opts = m.GetPackageOptions()
   InteractiveMode = im_Batch ' Evite l'affichage de la fenêtre de
génération
   opts.GenerateODBC = False ' Force la génération de script SQL
plutôt qu'ODBC
   opts.GenerationPathName = GenDir
   opts.GenerationScriptName = "MonScript.sql"
  m.GenerateDatabase ' Lance la fonctionnalité de génération de base
de données
End Sub
```

Pour générer via une connexion directe à une base de données, vous vous connectez à la base de données (en utilisant la méthode `ConnectToDatabase()`) puis définissez la propriété `GenerateODBC` à `true`.

Remarque : Pour plus d'informations sur les options de génération, sélectionnez **Aide > Aide sur les objets du métamodèle**, puis naviguez vers `Libraries/PdPDM/BasePhysicalPackageOptions`.

Cette procédure modifie le modèle en ajoutant une nouvelle colonne à chaque table :

```
Sub ModifyModel(m)
 dim pTable, pCol
 For each pTable in m.Tables
  Set pCol = pTable.Columns.CreateNew()
  pCol.SetNameAndCode "az" & pTable.Name, "AZ" & pTable.Code
```

```
  pCol.Mandatory = False
 Next
End Sub
```

Cette procédure génère un script alter pour modifier la base de données :

```
Sub GenerateAlterScripts(m)
 Dim pOpts
 Set pOpts = m.GetPackageOptions()
 InteractiveMode = im_Batch ' Evite l'affichage de la fenêtre de
génération
' Définit les options de génération en utilisant les options de
modèle
 pOpts.GenerateODBC = False ' Force la génération via script sql
plutôt que via ODBC
 pOpts.GenerationPathName = GenDir
 pOpts.DatabaseSynchronizationChoice = 0 'Force un fichier apm
existant comme source
 pOpts.DatabaseSynchronizationArchive = GenDir & "modele.apm"
 pOpts.GenerationScriptName = "MonScriptAlter.sql"
 m.ModifyDatabase ' Lance la fonctionnalité de modification de base
de données
End Sub
```

Cette procédure génère des données de test à charger dans la base :

```
Sub GenerateTestDataScript(m)
 Dim pOpts
 Set pOpts = m.GetPackageOptions()
 InteractiveMode = im_Batch ' Evite l'affichage de la fenêtre de
génération
' Définit les options de génération en utilisant les options de
modèle
 pOpts.TestDataGenerationByODBC = False ' Force la génération de
script sql plutôt qu'ODBC
 pOpts.TestDataGenerationDeleteOldData = False
pOpts.TestDataGenerationPathName = GenDir
 pOpts.TestDataGenerationScriptName = "MesDonneesTest.sql"
m.GenerateTestData ' Lance la fonctionnalité de génération de données
de teste
End Sub
```

Reverse engineering d'une base de données (Scripting)

Vous pouvez vous connecter à une base de données en utilisant la méthode `ConnectToDatabase()`, et procéder au reverse engineering du schéma vers un MPD en utilisant `ReverseDatabase()`.

Pour connecter une base de données via une source de données utilisateur ou système, définissez une constante sous la forme `"ODBC:`**`nomsourcedonnées`**`"`. Par exemple :

```
Const cnxDSN = "ODBC:ASA 9.0 sample"
```

Pour utiliser un fichier de source de données, définissez une constante avec le chemin complet vers le fichier DSN. Par exemple :

```
Const cnxDSN = "\\romeo\public\DATABASES\_filedsn
\sybase_asa9_sample.dsn"
```

Ce script crée un nouveau MPD, se connecte à une base de données via une source de données système, définit des options de reverse engineering et récupère tous les objets dans le MPD :

```
' Définition d'une source de données ODBC et d'un fichier de MPD
Const cnxDSN = "ODBC:MaBaseDonnees"
Const cnxUSR = "MonUtilisateur"
Const cnxPWD = "MonMotPasse"
Const filename = "C:\temp\MaBaseReverse.mpd"

Dim pModel, pOpt
' Création du modèle avec le SGBD approprié
Set pModel=CreateModel(PdPDM.cls_Model, "|DBMS=Sybase SQL Anywhere
12")
' Masquage des boîtes de dialogue
InteractiveMode = im_Batch

' Connexion à la base de données
 pModel.ConnectToDatabase cnxDSN, cnxUSR, cnxPWD
' Définition d'options de reverse engineering pour récupérer tous les
objets listés via ODBC
 Set pOpt = pModel.GetPackageOptions()
 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

' Récupère la base dans le modèle puis enregistre le modèle
 pModel.ReverseDatabase
 pModel.save(filename)
```

Création et utilisation d'extensions (Scripting)

Vous pouvez créer des extensions à l'aide de script afin de définir des propriétés supplémentaires, de nouvelles métaclasses, des formulaires, et tout autre type d'extension du métamodèle standard.

L'exemple suivant crée un MAE, puis y crée une extension, définit un nouveau type d'objet appelé `tablette` dérivé de la métaclasse `MobileDevice` et crée un nouvel attribut étendu et un nouveau formulaire personnalisé pour lui :

```
Dim MonModele, MonExt, MonStereotype, MonAttribEtendu,
MonFormulaire, FormDef
set MonModele =
CreateModel(PdEAM.Cls_Model,"Diagram=TechnologyInfrastructureDiagra
m")
MonModele.SetNameAndCode "MonMAE" , "MonMAE"
'Création de l'extension
Set MyExt = MonModele.ExtendedModelDefinitions.CreateNew()
MyExt.Name = "MonExtension"
MyExt.Code = "MonExtension"
'Création du stéréotype
Set MonStereotype = MonExt.AddMetaExtension(PdEAM.Cls_MobileDevice,
Cls_StereotypeTargetItem)
MonStereotype.Name = "Tablette"
MonStereotype.UseAsMetaClass = true
'Création d'attribut étendu
Set MyExAtt =
MonStereotype.AddMetaExtension(Cls_ExtendedAttributeTargetItem)
MyExAtt.Name = "TypeTablette"
MyExAtt.Label = "Type"
MyExAtt.DataType = "12" ' (String) Pour obtenir la liste des
valeurs,
' voir ExtendedAttributeTargetItem dans l'Aide sur les objets du
métamodèle
MyExAtt.ListOfValues = "iPad;Android;Playbook;Windows8"
MyExAtt.Value = "iPad"
'Création d'un formulaire pour remplacer l'onglet Général
Set MonFormulaire = MyStype.AddMetaExtension(Cls_FormTargetItem)
MonFormulaire.Name = "RemplaceGeneral"
MonFormulaire.FormType = "GENERAL"
'Assemblage de la définition du formulaire
FormDef = "<Form><StandardNameAndCode Attribute=""NameAndCode"" />"
& vbcrlf
FormDef = FormDef + "<StandardAttribute Attribute=""Comment"" />" &
vbcrlf
FormDef = FormDef + "<ExtendedAttribute Attribute=""TabletType"" />"
& vbcrlf
FormDef = FormDef + "<StandardAttribute Attribute=""KeywordList""  /
></Form>"
MyForm.Value = FormDef
```

Vous pouvez lire et définir des valeurs d'attributs étendus en utilisant les méthodes suivantes :

- `GetExtendedAttribute(`**"ressource.attribut"**`)`
- `GetExtendedAttributeText(`**"ressource.attribut"**`)`
- `SetExtendedAttribute` **"ressource.attribut" "valeur"**
- `SetExtendedAttributeText` **"ressource.attribut" "valeur"**

Vous pouvez accéder aux collection définies dans une extension à l'aide des méthodes suivantes :

- `GetCollectionByStereotype(`**`"stéréotype"`** - pour de nouveaux types d'objet définis dans une cible ou un fichier d'extension (voir *Création de nouvelles métaclasses à l'aide de stéréotypes* à la page 42).
- `GetExtendedCollection(`**`"ressource.collection"`**`)` - pour les collections et compositions étendues (voir *Collections et compositions étendues (Profile)* à la page 54).
- `GetCalculatedCollection(`**`"resource.collection"`**`)` - pour les collections calculées (voir *Collections calculées (Profile)* à la page 57).
- `GetCollectionByName(`**`"resource.collection"`**`)` - pour toute sorte de collection.

Le script suivant utilise la méthode `GetCollectionByStereotype()` afin d'accéder à la collection de tablettes et la méthode `SetExtendedAttribute` afin de définir le type de tablette :

```
Dim col, obj
'La collection de tablettes n'est pas directement accessible
 set col = ActiveModel.GetCollectionByStereotype("Tablette")
 'Création d'un tableau pour contenir les valeurs à affecter aux
propriétés de tablette
Dim myArray(3)
myArray(0) = "Tablette1, T1, PlayBook"
myArray(1) = "Tablette2, T2, Android"
myArray(2) = "Tablette3, T3, iPad"
myArray(3) = "Tablette4, T4, iPad"
CreateObjects col, myArray

'Procédure pour affecter des valeurs aux propriétés
Sub CreateObjects(compColl, dataArray)
   For Each line In dataArray
      Dim myProps
      myProps = split(line, ",")
      set obj = compColl.CreateNew()
      obj.Name = myProps(0)
      obj.Code = myProps(1)
        'Syntaxe spéciale pour définir l'attribut étendu
      obj.SetExtendedAttribute "MONEXT.TypeTablette", myProps(2)
   Next
End Sub
```

Accès aux métadonnées (Scripting)

Vous pouvez explorer la structure du métamodèle PowerAMC sous la forme d'un modèle autonome ou en partant des instances d'objet contenues dans votre modèle.

Pour obtenir des informations générales sur l'accès au métamodèle et la navigation dans le métamodèle, voir *Chapitre 8, Métamodèle public PowerAMC* à la page 371. Les métaclasses (telles que `CheckModelInternalMessage` et `FileReportItem`) qui ne sont pas accessibles par script sont visibles dans Metamodel.moo, mais portent le stéréotype `<<notScriptable>>` et ne sont pas répertoriées dans le fichier d'Aide sur les objets du métamodèle.

Vous pouvez accéder aux métaclasses, méta-attributs et métacollections en procédant à l'itération sur les collections en partant de la racine de `MetaModel` ou individuellement via les méthodes suivantes :

- GetMetaClassByPublicName (**nom**) - pour accéder à une métaclasse par son nom public.
- GetMetaMemberByPublicName (**nom**) - pour accéder à un méta-attribut ou à une métacollection par son nom public.

Le script suivant traverse le métamodèle par bibliothèques et répertorie chaque classe concrète :

```
for each l in MetaModel.Libraries
   for each c in l.Classes
      if c.Abstract = false then
         Output l.PublicName + "." + c.PublicName
      end if
   next
next
```

Le script suivant localise la racine `BaseClass` et montre les deux premiers niveaux d'héritage située au-dessous :

```
set root = MetaModel.GetMetaClassByPublicName("PdCommon.BaseObject")
for each c in root.Children
   output c.PublicName
   for each cc in c.Children
      output "    " + cc.PublicName
   next
next
```

Le script suivant obtient une table dans un MPD, puis montre la métaclasse dont l'objet est une instance, la métaclasss parent et la métabibliothèque de la métaclasse, ainsi que toutes les attributs et toutes les collections qui sont disponibles pour cette métaclasse :

```
Dim object
Set object = ActiveModel.FindChildByName("myTable",cls_Table)
Output "Objet: " + object.Name

Dim metaclass
Set metaclass = object.MetaClass
Output "Métaclasse : " + metaclass.PublicName
Output "Parent : " + metaclass.Parent.PublicName
Output "Métabibliothèque : " + metaclass.Library.PublicName
Output "Attributs :"
For each attr in metaclass.attributes
   Output " - " + attr.PublicName
Next
Output "Collections :"
For each coll in metaclass.collections
   Output " - " + coll.PublicName
Next
```

Les propriétés et les collections sont en lecture seule pour tous les objets du métamodèle.

OLE Automation et compléments

OLE Automation est un moyen de communiquer avec PowerAMC à partir d'une autre application grâce à l'architecture COM. 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.Vous pouvez créer des exécutables qui appellent PowerAMC ou des compléments qui sont appelés par PowerAMC.

Les programmes VBScript qui sont exécutés depuis PowerAMC et les programmes OLE Automation sont très similaires, mais OLE requiert de travailler via un objet application PowerAMC, et d'utiliser un typage plus fort. Vous devez :

- Créer une instance de l'objet application PowerAMC et la libérer lorsque votre script se termine :

```
 Dim PAMC As PdCommon.Application
Set PAMC = CreateObject("PowerAMC.Application")
'Saisir le script ici
'A la fois du script, libérer l'objet PAMC
Set PAMC = Nothing
```

  Si PowerAMC est en cours d'exécution, cette instance sera utilisée ; faute de quoi une nouvelle instance sera lancée. Si vous ne spécifiez pas de numéro de version, c'est la version la plus récente qui est utilisée. Pour spécifier une version particulière, utilisez la syntaxe suivante :

```
Set PAMC = CreateObject("PowerAMC.Application.version")
```

- Préfixer toutes les propriétés et fonctions globales (voir *Manipulation des modèles, des collections et des objets (Scripting)* à la page 338) à l'aide de l'objet application PowerAMC. Par exemple, pour accéder au modèle actif en utilisant un objet application PowerAMC appelé `PAMC`, utilisez la syntaxe suivante :

```
PAMC.ActiveModel
```

- Spécifier des types d'objet chaque fois que possible. Par exemple, plutôt que de simplement utiliser `Dim cls`, vous devez utiliser :

```
Dim cls as PdOOM.Class
```

  Si votre modèle contient des raccourcis, nous vous recommandons d'utiliser la syntaxe suivante pour éviter les erreurs au moment de l'exécution lorsque le modèle cible est fermé :

```
Dim obj as PdCommon.IdentifiedObject
```

- Adapter la syntaxe de l'ID de classe de l'objet au langage dans lequel vous créez cet objet. Pour VBScript, VBA, VB et les autres langages qui prennent en charge l'énumération définie hors d'une classe, vous pouvez utiliser la syntaxe suivante :

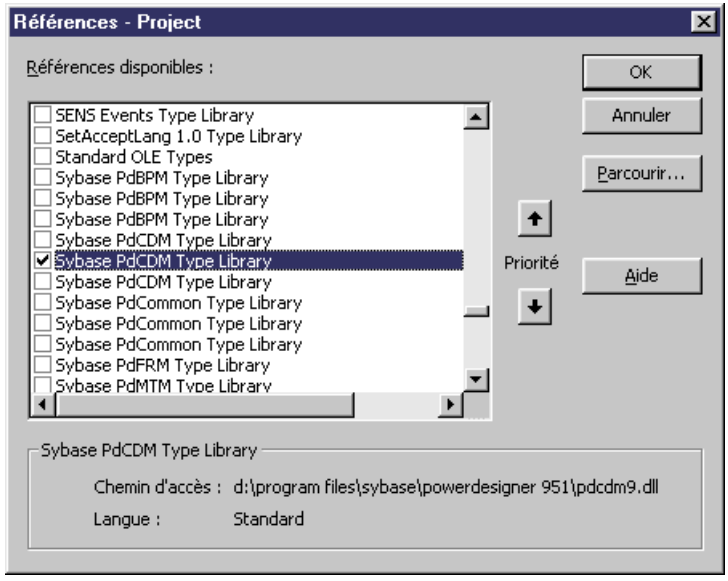
```
Dim cls  as PdOOM.Class
Set cls = model.CreateObject(PdOOM.cls_Class)
```

Pour C# et VB.NET, vous pouvez utiliser la syntaxe suivante (dans laquelle `PdOOM_Classes` est le nom de l'énumération) :

```
Dim cls As PdOOM.Class
Set cls = model.CreateObject(PdOOM.PdOOM_Classes.cls_Class)
```

Pour d'autres langages tels que le JavaScript ou PowerBuilder, vous devez définir des constantes qui représentent les objets que vous souhaitez créer. Pour obtenir la liste complète des constantes d'ID de classe, reportez-vous au fichier VBScriptConstants.vbs situé dans le répertoire OLE Automation de PowerAMC.

- Ajouter des références aux bibliothèques de type d'objet que vous devez utiliser. Par exemple, dans un éditeur VBA, sélectionnez **Tools** > **References**:



Ce script est lancé depuis l'extérieur de PowerAMC, crée une instance de l'objet application de PowerAMC, et la réutilise pour créer deux MOO via OLE Automation :

```
'* Purpose:  Ce script affiche le nbre de classes définies dans un
MOO dans la fenêtre Résultats.
Option Explicit
' Fonction principale
Sub VBTest()
 ' Définir l'objet application PowerAMC
 Dim PAMC As PdCommon.Application
 ' Lire l'objet application PowerAMC
 Set PAMC = CreateObject("PowerAMC.Application")
' Obtenir le modèle actif courant
 Dim model As PdCommon.BaseModel
 Set model = PAMC.ActiveModel
 If model Is Nothing Then
  MsgBox "Il n'y pas de modèle courant."
 ElsIf Not model.IsKindOf(PdOOM.cls_Model) Then
  MsgBox Le modèle courant n'est pas un MOO."
 Else
```

```
  ' Afficher le nombre de classes
  Dim nbClass
  nbClass = Model.Classes.Count
  PAMC.Output "Le modèle '" + model.Name + "' contient " +
CStr(nbClass) + " classes."
' Créer un nouveau MOO
  Dim model2 As PdOOM.Class
  Set model2 = PAMC.CreateModel(PdOOM.cls_Model)
  If Not model2 Is Nothing Then
   ' Copier le nom de l'auteur
   model2.Author = Model.Author
   ' Afficher un message dans la fenêtre Résultats
   PAMC.Output "Le modèle '" + model2.Name + "' a été correctement
créé + "'."
  Else
   MsgBox "Impossible de créer un MOO."
  End If
 End If
' Libérer l'objet application PowerAMC
 Set PAMC = Nothing
End Sub
```

Des exemples OLE Automation pour différents langages sont fournis dans le répertoire OLE Automation situé dans votre répertoire d'installation de PowerAMC.

Création d'un complément ActiveX

Vous pouvez créer des compléments ActiveX pour ajouter des fonctionnalités supplémentaires dans PowerAMC, et les appeler via des commandes de menu.

Pour pouvoir fonctionner comme un complément PowerAMC, le complément ActiveX doit mettre en oeuvre l'interface `IPDAddIn`, qui définit les méthodes suivantes, appelées par PowerAMC pour dialoguer avec les menus et exécuter les commandes définies par le complément :

- `HRESULT Initialize([in] IDispatch * pApplication)` et `HRESULT Uninitialize()` - La méthode `Initialize()` initialise les communications entre PowerAMC et le complément. PowerAMC fournit un pointeur vers son objet application, défini dans la bibliothèque de type PdCommon, qui permet d'accéder à l'environnement PowerAMC (fenêtre de résultats, modèle actif, etc.). La méthode `Uninitialize()` est appelée lorsque PowerAMC est fermé afin de libérer toutes les variables globales et références aux objet PowerAMC.
- `BSTR ProvideMenuItems([in] BSTR sMenu, [in] IDispatch *pObj)` - est appelé chaque fois que PowerAMC doit afficher un menu, et renvoie un texte XML qui décrit les éléments de menu à afficher. Il est appelé une fois sans paramètre d'objet à l'initialisation de PowerAMC pour remplir les menus **Importer** et **Reverse engineering**. Lorsque vous faites un clic droit sur un symbole dans un diagramme, cette méthode est appelée deux fois : une fois pour l'objet et l'autre pour le symbole. Par conséquent, vous pouvez créer une méthode qui n'est appelée que dans les menus contextuels graphiques.
  Le DTD pour la définition de menu se présente comme suit :

```
<!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>
```

Par exemple :

```
ProvideMenuItems ("Object", pModel)
```

renvoie le texte suivant :

```
<Menu>
<Popup Caption="&Perforce">
    <Command Name="CheckIn" Caption="Check &In"/>
    <Separator/>
    <Command Name="CheckOut" Caption="Check &Out"/>
</POPUP>
</MENU>
```

- `BOOL IsCommandSupported([in] BSTR sMenu, [in] IDispatch * pObject, [in] BSTR sCommandName)` - permet de désactiver de façon dynamique les commandes définies dans un menu. La méthode doit renvoyer true pour activer une commande et false pour la désactiver.
- `HRESULT DoCommand(in BSTR sMenu, in IDispatch *pObj, in BSTR sCommandName)` - met en oeuvre l'exécution d'une commande designée par son nom. Par exemple :

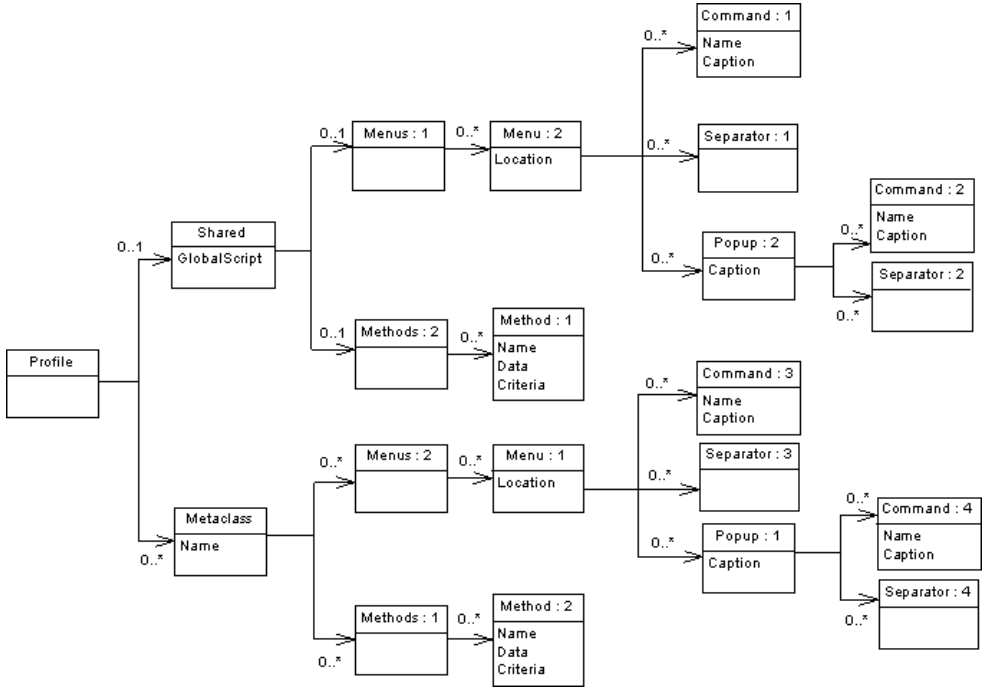
```
DoCommand ("Object", pModel, "CheckIn")
```

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 > Modélisation avec PowerAMC > Personnalisation de votre environnement de modélisation > Options générales > Compléments*).

Création d'un complément fichier XML

Vous pouvez créer des compléments XML pour grouper plusieurs commandes afin d'appeler des programmes exécutables ou des scripts VB et les ajouter dans des menus PowerAMC.

L'illustration suivante vous permet de comprendre la structure du fichier XML :



Remarque : Le DTD est disponible dans `répinstall_PowerAMC\Add-ins\XMLAddins.dtd`.

`Profile` est l'élément racine dans le descripteur du complement fichier XML, et peut contenir :

- Un élément `Shared` - définit les menus qui sont toujours disponibles et leurs méthodes associées, avec un attribut `GlobalScript`, qui peut contenir un script global pour les fonctions partagées.
- Un ou plusieurs éléments Metaclass - qui définissent les commandes et menus pour une métaclasse spécifique, identifiée par son nom public préfixé par le nom public de sa bibliothèque de type.

Ces deux éléments peuvent contenir des sous-éléments comme suit :

- `Menus` contient des éléments `Menu` qui spécifient un emplacement, il peut s'agir de :
  - FileImport - Shared uniquement
  - FileExport - Metaclass uniquement
  - FileReverse - Shared uniquement
  - Tools
  - Help
  - Object - Metaclass uniquement (défaut)

  Chaque élément `Menu` peut contenir :

    - Un élément `Command` - dont le `Name` doit être identique au nom d'une Method, et dont le `Caption` définit le nom de la commande qui s'affiche dans le menu.
    - Un élément `Separator` - qui indique que vous souhaitez insérer une ligne dans le menu.
    - Un élément `Popup` - qui définit un sous-menu qui peut à son tour contenir des commandes, séparateurs et sous-menus.
- `Methods` contient des éléments `Method`, qui définissent les méthodes utilisées dans les menus, et qui sont définis par un nom et un VBScript. Une méthode définie sous une métaclasse a l'objet courant comme paramètre. L'héritage est pris en compte, de sorte qu'un menu défini sur la métaclasse `PdCommon.NamedObject` sera disponible sur `PdOOM.Class`.

L'exemple suivant définit deux commandes de menu pour le référentiel Perforce et les méthodes qu'elles appellent :

```
<?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>
```

L'exemple suivant définir un script global qui est référencé par une méthode définie sous une métaclasse :

```
<?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="Convertir en 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>
```

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 *Guide des fonctionnalités générales > Modélisation avec PowerAMC > Personnalisation de votre environnement de modélisation > Options générales > Compléments*).

Lancement des scripts et de compléments depuis les menus

Vous pouvez étendre les menus PowerAMC afin d'ajouter des commandes qui appellent des scripts définis dans les fichiers de ressources ou de façon externe et lancer des exécutables et des compléments ActiveX. Les compléments XML peuvent être utilisés pour regrouper et organiser plusieurs commandes. Vous pouvez étendre les menus **Fichier**, **Outils** et **Aide**, ainsi que les menus contextuels disponibles dans l'Explorateur d'objets et dans les diagrammes.

Vous pouvez modifier les menus PowerAMC de l'une des façons suivantes :

- Commandes personnalisées - elles sont définies directement dans PowerAMC et peuvent appeler des programmes exécutables ou des scripts VB (voir *Ajout de commandes dans le menu Outils* à la page 366).
- Extensions de menus et de méthodes – elles sont spécifiées dans un fichier de définition de SGBD ou de langage et définissent les commandes pour une cible pour un type de modèle particuliers (voir *Menus (Profile)* à la page 90).
- Compléments ActiveX – sont rédigés dans un langage tel que VB, C#, C++ ou tout autre langage qui prend en charge COM, et ils permettent des interactions plus complexes avec PowerAMC, comme par exemple d'activer et de désactiver des menus en fonction d'une

sélection d'objets, et des interaction avec l'environnement d'affichage des fenêtres (voir *Création d'un complément ActiveX* à la page 361).

Remarque : La syntaxe XML utilisée pour définir les menus dans un complément ActiveX ou XML est la même que celle utilisée dans la création d'une extension de menu, et vous pouvez utiliser l'Editeur de ressources pour afficher la page XML d'un menu XML (voir*Menus (Profile)* à la page 90) pour vous aider à construire la syntaxe de vos compléments.

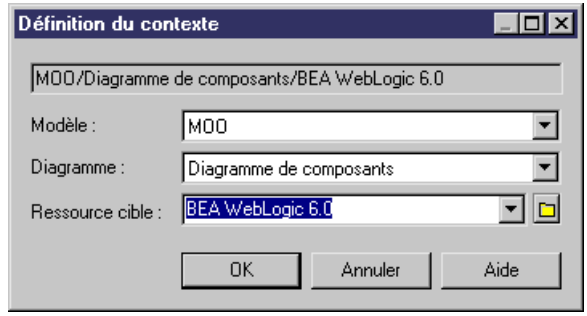
- Compléments XML – définit plusieurs commandes pour appeler des programmes exécutables ou des scripts VB. Les commandes liées aux mêmes applications (par exemple, ASE, IQ, etc.) doivent être rassemblée dans le même fichier XML (voir *Création d'un complément fichier XML* à la page 362).

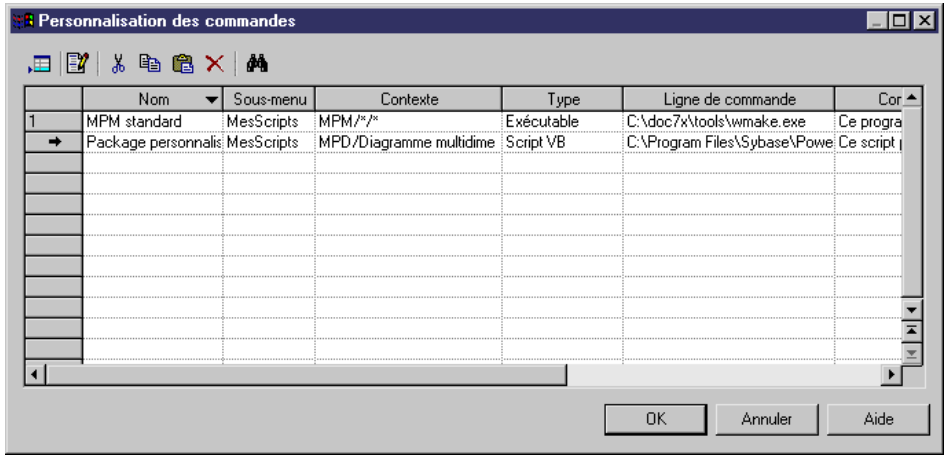
Ajout de commandes dans le menu Outils

Vous pouvez créer vos propres éléments de menu dans le menu Outils de PowerAMC afin d'accéder à des objets PowerAMC à l'aide de vos propres scripts ou programmes exécutables. Vous pouvez définir jusqu'à 256 commandes dans la boîte de dialogue **Personnalisations des commandes**, et contrôler les contextes (modèle, diagramme et type de cible) dans lesquels elles apparaissent.

1. Sélectionnez **Outils > Exécuter des commandes > Personnaliser des commandes** puis cliquez sur l'outil **Ajouter une ligne**.
2. Saisissez les propriétés suivantes :

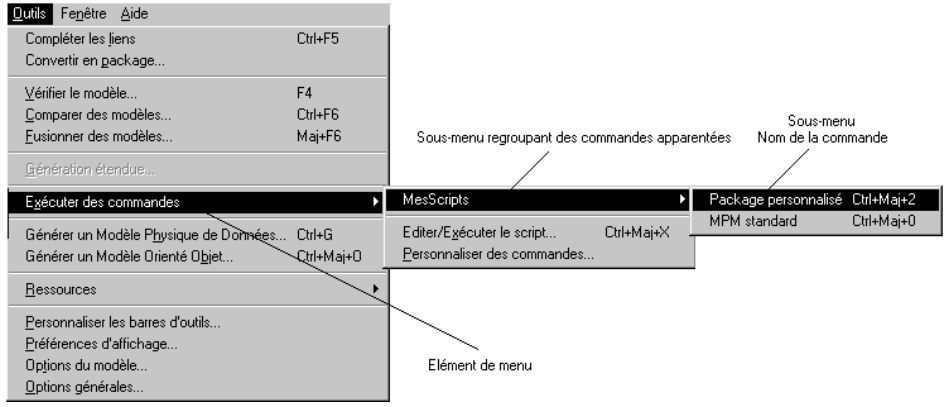
Propriété	Description
Nom	Spécifie le nom de la commande qui va s'afficher dans le menu. Les noms doivent être uniques et peuvent contenir une touche d'accès rapide (&Génération Java s'affichera sous la forme Génération Java)
Sous-menu	Spécifie le sous-menu dans lequel placer la commande. Vous pouvez spécifier une sous-menu ou sélectionner : • <Aucun> - directement sous **Outils > Exécuter des commandes** • Vérification du modèle • Exporter • Génération • Importer - apparaît aussi sous **Fichier > Importer** • Reverse engineering - apparaît aussi sous **Fichier > Reverse engineering**

Propriété	Description
Contexte	Spécifie quand la commandes est disponible. Par défaut, la commande est toujours disponible (`*/*/*`). Cliquez sur le bouton Points de suspension pour limiter l'affichage de la commande à un contexte particulier : • Type de modèle - par exemple `MOO/*/*` • Type de modèle et de diagramme - par exemple `MOO/Diagramme de classes/*` • Type de modèle, de diagramme et de cible - par exemple `MOO/Diagramme de classes/Java`. Par défaut, la liste contient des extensions disponibles pour le type de modèle choisi. Cliquez sur l'outil **Chemin** pour naviguer vers un autre dossier contenant des extensions ou les fichiers de définition de SGBD ou de langage objet. 
Type	Spécifie si la commande va lancer un exécutable ou exécuter du code VBScript.
Ligne de commande	Spécifie le chemin vers l'exécutable ou le fichier de script à exécuter. Cliquez sur le bouton Points de suspension pour sélectionner un fichier. Si votre fichier est un fichier VBScript, vous pouvez consulter ou éditer le script en cliquant sur l'outil **Editer avec** dans la barre d'outils.
Commentaire	Spécifie le texte qui s'affiche dans la barre d'état lorsque vous sélectionner la commande.
[A]fficher dans le menu	Spécifie que la commande doit être affichée. Désélectionnez cette zone pour masque la commande en conservant sa définition.
Touche de raccourci	Associe à l'a commande l'un des dix raccourcis clavier réservés (de Ctrl-Maj-0 à Ctrl-Maj-9).



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

   Votre commande est maintenant disponible sous **Outils > Exécuter des commandes**.

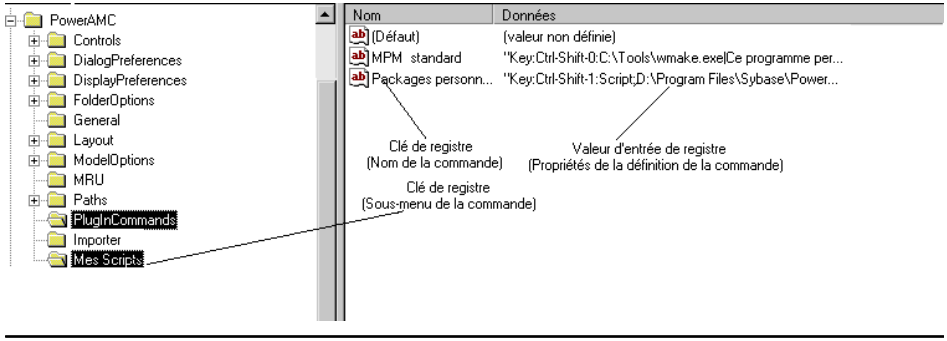


Remarque : Les commandes personnalisées sont enregistrées par défaut dans le Registre sous `HKEY_CURRENT_USER\Software\Sybase\PowerAMC` **v** `\PlugInCommands\`**sous-menu** et sont disponibles uniquement pour l'utilisateur qui les a définies. Pour les rendre disponibles pour les autres utilisateurs, créez une entrée au même endroit, sous `HKEY_LOCAL_MACHINE`.

Le nom de l'entrée est celui de la commande, et sa valeur prend la syntaxe suivante, dans laquelle seul le paramètre **lignecommande** est obligatoire et doit être terminé par un caractère | (pipe)

`[Hide:][Key:`**raccourci**`:][Script:]`**lignecommande**`[ |`**commentaire**`]`

Si vous souhaitez insérer un caractère pipe dans une commande, vous devez doubler ce caractère.


PowerAMC
Controls
DialogPreferences
DisplayPreferences
FolderOptions
General
Layout
ModelOptions
MRU
Paths
PlugInCommands
Importer
Mes Scripts
Nom
Données
(Défaut)
(valeur non définie)
MPM standard
"Key:Ctrl-Shift-0:C:\Tools\wmake.exe|Ce programme per...
Packages personn...
"Key:Ctrl-Shift-1:Script;D:\Program Files\Sybase\Power...
Clé de registre
(Nom de la commande)
Valeur d'entrée de registre
(Propriétés de la définition de la commande)
Clé de registre
(Sous-menu de la commande)

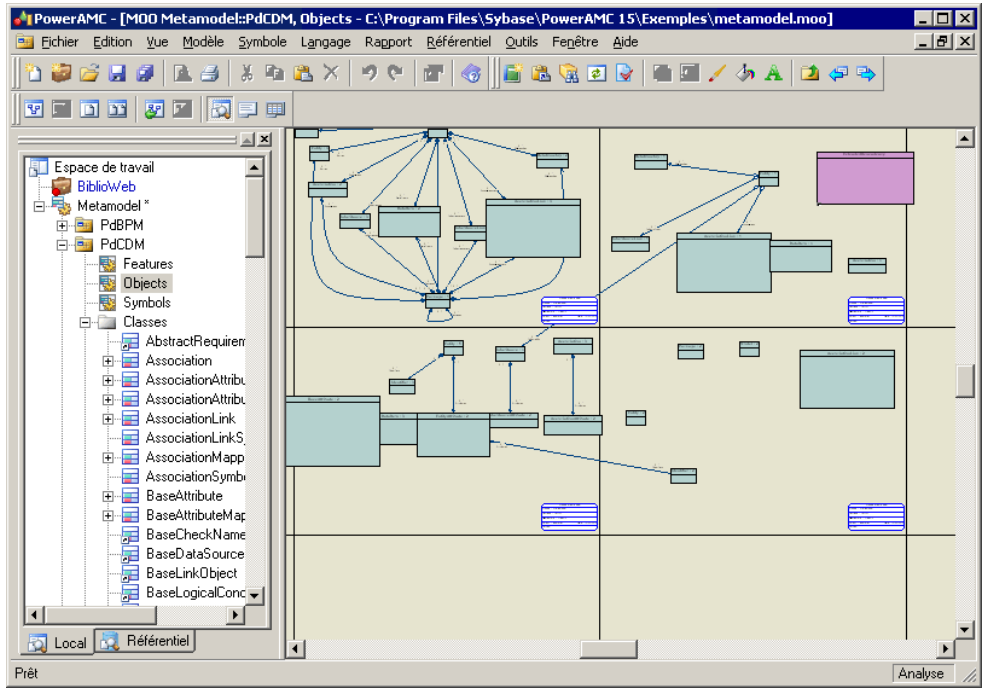
Chapitre 8 Métamodèle public PowerAMC

Le métamodèle public PowerAMC est une abstraction des métadonnées de tous les modèles de PowerAMC, et décrit les éléments d'un modèle, ainsi que la syntaxe et la sémantique de leur manipulation.

Vous pouvez consulter le métamodèle public dans PowerAMC en ouvrant *répertoire d'installation*`\Exemples\MetaModel.moo`, et consulter la documentation exhaustive sur tous les objets, collections et méthodes du métamodèle disponible via scripting en sélectionnant **Aide > Aide sur les objets du métamodèle** (voir *Utilisation du fichier d'aide sur les objets du métamodèle* à la page 375).

Ce MOO et ce fichier d'aide sont destinés à vous aider à comprendre la structure générale de vos modèles PowerAMC lorsque vous utilisez des :

- Templates du GTL (Generation Template Language, langage de génération par template) (voir *Chapitre 5, Personnalisation de la génération à l'aide du langage de génération par template* à la page 265).
- Scripts VB (voir *Chapitre 7, Pilotage de PowerAMC à l'aide de scripts* à la page 331).
- Fichiers de modèle XML PowerAMC (voir *Format du fichier de modèle PowerAMC* à la page 376).



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)
- PdGLM - Modèle de glossaire
- 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

Navigation dans le métamodèle

Vous pouvez développer et réduire les packages dans l'Explorateur d'objets afin d'en explorer le contenu. Double-cliquez sur un diagramme pour l'afficher dans la fenêtre de modèle.

Chaque métaclasse est dotée d'un nom, contient zéro ou plus attributs et assure zéro ou plus rôles dans des associations avec d'autres classes, qui permettent d'identifier les collections. Le métamodèle public PowerAMC utilise des concepts standard UML :

- *Noms publics* - Chaque objet du métamodèle PowerAMC a un nom et un code qui correspondent au nom public de l'objet, qui est un identificateur unique de cet objet dans la

bibliothèque du modèle ou dans un package. Les noms publics sont référencés dans les fichiers de modèle XML de PowerAMC et lorsque vous utilisez le langage de génération par template (GTL) et le scripting. Le nom public correspond souvent au nom de l'objet dans l'interface de PowerAMC, mais si les deux divergent, le nom public doit être utilisé dans des scripts et dans les templates de GTL.

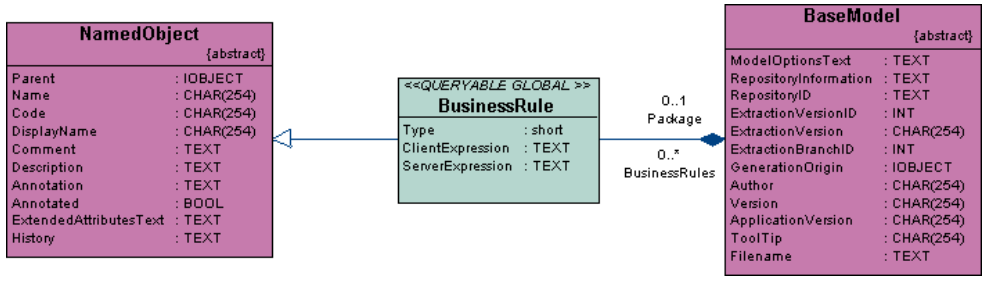
- *Classes* - sont utilisées pour représenter les métadonnées des façons suivantes :
  - *Classes abstraites* - elles sont utilisées pour partager des attributs et comportements, et ne sont pas visibles dans l'interface PowerAMC.
  - *Classes instanciables/concrètes* - elles correspondent aux objets affichés dans l'interface. Elles ont leurs propres attributs et comportements en plus de ceux dont elles héritent via les 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.
- *Attributs de classes* - ce sont des propriétés d'objet. 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. Ni les attributs dérivés, ni les attributs migrés depuis des associations navigables ne sont stockés dans le fichier de modèle. Les attributs non-dérivés sont propres à la classe, et sont stockés dans le modèle et enregistrés dans le fichier de modèle.
- *Associations* - expriment les connexions sémantiques entre des classes. Dans la feuille de propriétés de l'association, les rôles transportent l'information relative à l'objet d'extrémité de l'association. Les objets PowerAMC sont liés à d'autres objets en utilisant des collections, et le rôle situé à l'autre extrémité de l'association donne le nom de la collection pour un objet. Par exemple, `NamedObject` a une collection des règles de gestion appelée `AttachedRules`, et `BusinessRule` a une collection d'objets appelés `Objects` :

  Lorsque les associations ont deux rôles, seule la collection ayant le rôle *navigable* sera enregistré dans le fichier XML. Dans le cas, seule la collection `AttachedRules` est enregistrée.
- *Compositions* – expriment une association dans laquelle les enfant vivent et meurent avec les parents, et si le parent est copié, l'enfant est également copié. Par exemple, `Table` a une association de composition avec la classe `Column` :

- *Généralisations* - montrent les liens d'*héritage* qui existent 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. Par exemple, `Class` hérite de `Classifier`

Chaque diagramme montre les connexions entre les métaclasses via des associations et des généralisations. Les classes en *vert* sont définies dans le diagramme courant, tandis que celles en *mauve* ne sont présents que pour fournir du contexte. Pour en savoir plus sur une classe mauve, pointez sur elle, cliquez le bouton droit de la souris, puis sélectionnez **Diagrammes associés** > ***diagramme*** pour ouvrir le diagramme dans lequel elle est définie.

Dans l'exemple suivant, `BusinessRule` est définie, tandis que `NamedObject` et `BaseModel` ne sont présents que pour montrer les liens d'héritage et de composition :



Double-cliquez sur n'importe quelle classe pour afficher sa feuille de propriétés et consulter les onglets suivants :

- **Général** - fournit le nom public dans les zones **Nom** et **Code**, ainsi qu'un **Commentaire** donnant une brève description de la classe, et indique si elle est **Abstraite**.

  Remarque : Certains objets, tels que `RepositoryGroup` ne prennent pas en charge le scripting et portent le `<<notScriptable>>` stéréotype.

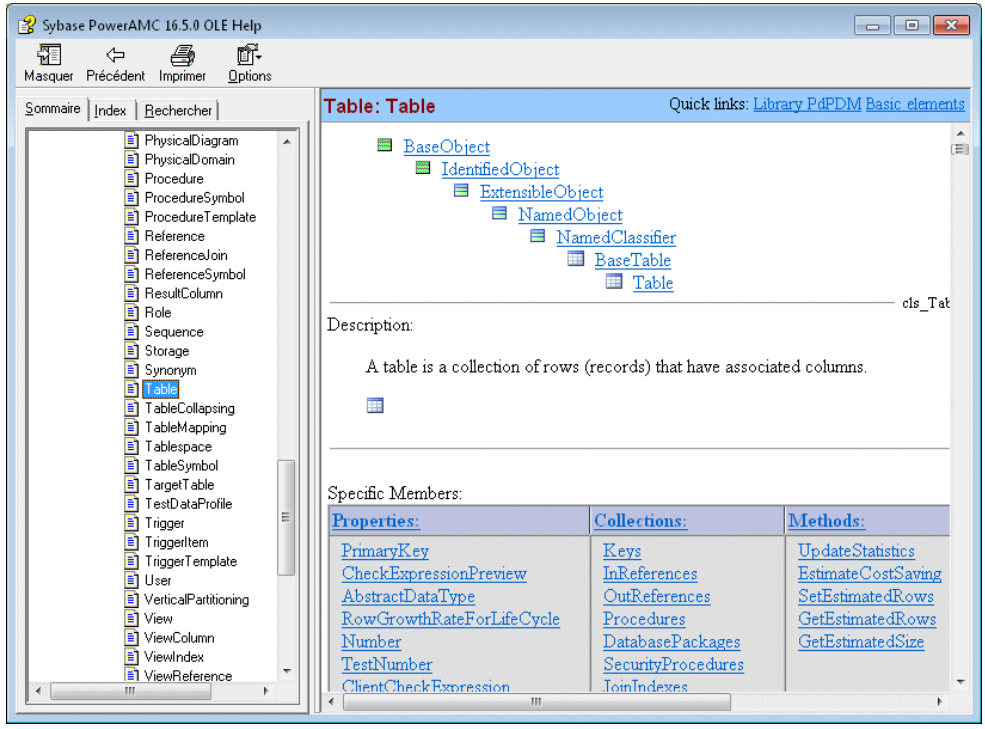
- **Attributs** - répertorie les propriétés définies directement sur la classe, mais pas celles dont cette classe hérite de ses classes parent.
- **Associations** - répertorie les associations mitrées pour la classe, qui représentent les collections. La colonne **Rôle B** répertorie les collections pour la classe, tandis que la colonne **Rôle A** répertorie les collections dans laquelle figure la classe.
- **Opérations** - répertorie les méthodes disponibles pour le scripting.
- **Dépendances** - contient (notamment) les sous-onglets suivants :
  - **Associations**
  - **Généralisations** - répertorie les liens de génération dans lesquels la classe courante est l'enfant et hérite d'attributs d'une classe parent.
  - **Spécialisations** - répertorie les liens de génération pour lesquels la classe courante est le parent et ses enfants héritent de ses attributs.
  - **Raccourcis** - répertorie les raccourcis créés depuis l'objet courant.

- **Notes** - peut inclure des informations supplémentaires sur les sous-onglets **Description** ou **Annotation**.

Utilisation du fichier d'aide sur les objets du métamodèle

PowerAMC fournit une documentation sur le métamodèle disponible en sélectionnant **Aide** > **Aide sur les objets du métamodèle**.

Le fichier peut être ouvert à partir de la boîte de dialogue **Edition/Exécution d'un script** (voir *Exécution de scripts dans PowerAMC* à la page 333) ou depuis une métaclasse dans un fichier de ressource (voir *Métaclasses (Profile)* à la page 35) en cliquant sur le bouton **Rechercher dans l'aide sur les objets du métamodèle** ou en appuyant sur **Ctrl+F1**. Vous pouvez également l'ouvrir à partir de n'importe quelle feuille de propriétés d'objet en appuyant sur **Ctrl+F1**, ou bien en appuyant sur le bouton **Menu de la feuille de propriétés** et en sélectionnant **Rechercher dans l'aide sur les objets du métamodèle**.



Les trois noeuds situés à la racine contiennent la documentation suivante :

Noeuds	Ce que vous trouverez...
Basic Elements	Fournit des informations générales sur : • Collections d'objets - principale méthode de navigation dans le métamodèle (voir *Consultation et modification des collections (Scripting)* à la page 340). • Types structurés - permettent de positionner des symboles dans les diagrammes (voir *Affichage, mise en forme et positionnement des symboles (Scripting)* à la page 346). • Propriétés globales, constantes, et fonctions - fournissent des points d'entrée pour le scripting (voir *Manipulation des modèles, des collections et des objets (Scripting)* à la page 338).
Libraries	Fournissent une documentation exhaustive de toutes les propriétés, collections et méthodes pour les objets du métamodèle utilisables dans le script, et regroupées par module.
Appendix	Inclut une hiérarchie développable affichant toutes les métaclasses du modèle PowerAMC, un exemple de code VBScript, et une liste des constantes d'ID de classe permettant d'identifier les objets dans certains contextes (voir *Accès et modification des objets et propriétés (Scripting)* à la page 342).

Pour obtenir des informations sur les propriétés, collections et méthodes disponibles pour une métaclasse particulière, affichez cette dernière sous la catégorie Libraries, ou localisez-la dans l'index. Toutes les propriétés, collections et méthodes sont répertoriées dans l'index.

Chaque métaclasse montre la hiérarchie de ses ancêtres dont elle descend et hérite. Après une brève description et un symbole, elle répertorie :

- Specific Members (Membres spécifiques) - une table qui répertorie les propriétés, collections et méthodes définies directement sur cette métaclasse
- Full Definition (Définition complète) - répertorie, dans des tables distinctes, les propriétés, collections et méthodes héritées de chacun de ses ancêtres. Par exemple, la métaclasse Table (située à l'adresse `Libraries\PdPDM\Table`) hérite des membres de :
  - PdCommon.BaseObject
  - PdCommon.IdentifiedObject
  - PdCommon.ExtensibleObject
  - PdCommon.NamedObject
  - PdCommon.NamedClassifier
  - PdPDM.BaseTable
  - PdPDM.View

Format du fichier de modèle 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:`*`objet`*`>` - 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:`*`objet`* `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:`*`attribut`*`>` - 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.

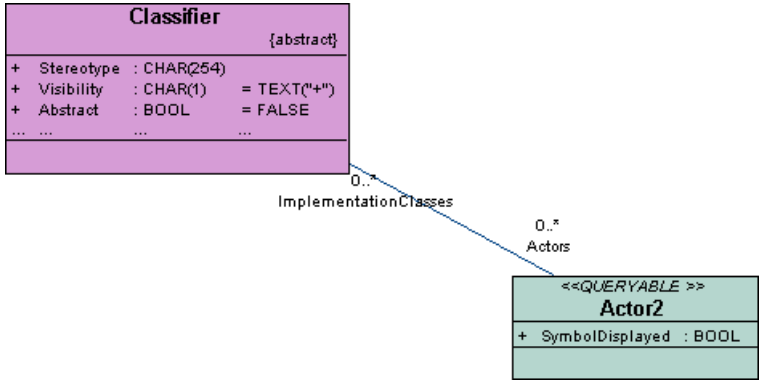
Vous pouvez chercher un objet dans le métamodèle en utilisant son nom d'objet dans le fichier XML afin de mieux comprendre sa définition. Une fois vous que vous avez trouvé l'objet dans le métamodèle, vous pouvez lire les informations suivantes :

- 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.

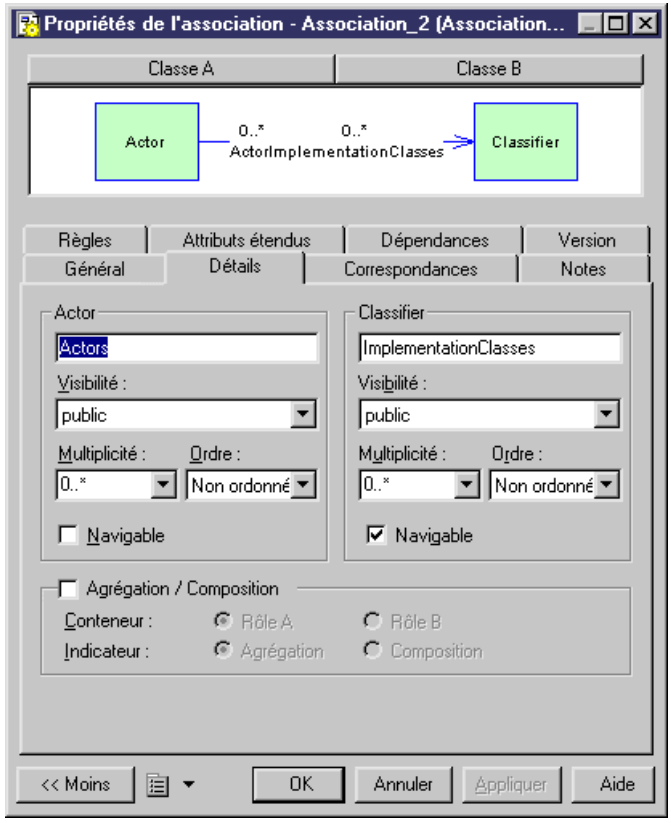
  En règle générale, les associations n'ont qu'un seul rôle, le rôle s'affiche à l'opposé de la classe pour laquelle il représente une collection. Toutefois, le métamodèle contient également des associations ayant deux rôles, auquel cas, les deux collections ne peuvent pas être enregistrées dans le fichier 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 :



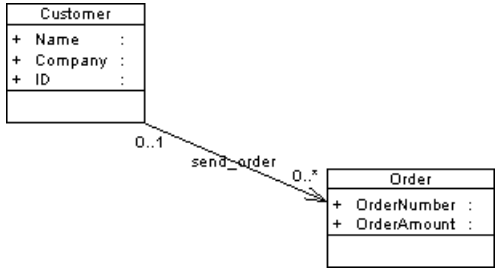
Si vous affichez la feuille de propriétés de l'association, vous pouvez voir que la case Navigable est cochée pour le rôle ImplementationClass, ce qui signifie que seule la collection ImplementationClass sera enregistrée dans le fichier.



- 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

Dans cet exemple, nous allons explorer la structure d'un simple fichier de MOO contenant deux classes et une association.



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>
```

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X