IBM Rational

System Architect USRPROPS
Guide d'extensibilité

Edition 11.3.1

Avant d'utiliser le présent document, consultez la section "Remarques" à la page 5-1 de l'annexe.

LE PRESENT DOCUMENT EST LIVRE EN L'ETAT SANS AUCUNE GARANTIE EXPLICITE OU IMPLICITE. IBM DECLINE NOTAMMENT TOUTE RESPONSABILITE RELATIVE A CES INFORMATIONS EN CAS DE CONTREFACON AINSI QU'EN CAS DE DEFAUT D'APTITUDE A L'EXECUTION D'UN TRAVAIL DONNE.

Ce document est mis à jour périodiquement. Chaque nouvelle édition inclut les mises à jour. Les informations qui y sont fournies sont susceptibles d'être modifiées avant que les produits décrits ne deviennent eux-mêmes disponibles. En outre, il peut contenir des informations ou des références concernant certains produits, logiciels ou services non annoncés dans ce pays. Cela ne signifie cependant pas qu'ils y seront annoncés.

Pour plus de détails, pour toute demande d'ordre technique, ou pour obtenir des exemplaires de documents IBM, référez-vous aux documents d'annonce disponibles dans votre pays, ou adressez-vous à votre partenaire commercial.

Vous pouvez également consulter les serveurs Internet suivants :

http://www.fr.ibm.com (serveur IBM en France)
http://www.can.ibm.com (serveur IBM au Canada)
http://www.ibm.com (serveur IBM aux Etats-Unis)

Compagnie IBM France
Direction Qualité
17, avenue de l'Europe
92275 Bois-Colombes Cedex

Cette édition s'applique à IBM® Rational® System Architect®, version 11.3.1 et à toutes les éditions et modifications ultérieures jusqu'à mention contraire dans les nouvelles éditions.

Table des matières

1

Extension d'un métamodèle d'encyclopédie System Architect

Introduction

Ce chapitre présente les mécanismes d'extension d'un métamodèle d'encyclopédie IBM Rational System Architect via USRPROPS.TXT.

Extension de Rational*System Architect*

Rational System Architect peut être étendu et personnalisé de plusieurs manières. Son comportement de traçage peut être personnalisé par l'intermédiaire de diverses options de l'outil et du fichier sa2001.ini. Ses barres d'outils peuvent être personnalisées, ses éditeurs de matrice peuvent être personnalisés, ses rapports peuvent être personnalisés, et ainsi de suite. Rational System Architect propose également une prise en charge intégrée de Microsoft Visual Basic for Applications, qui permet aux utilisateurs de rédiger des macros natives pouvant s'exécuter dans Rational System Architect pour effectuer toutes sortes d'opérations, telles que l'ajout d'utilitaires ou même influer sur le comportement de l'outil.

Extension du métamodèle via USRPROPS.TXT

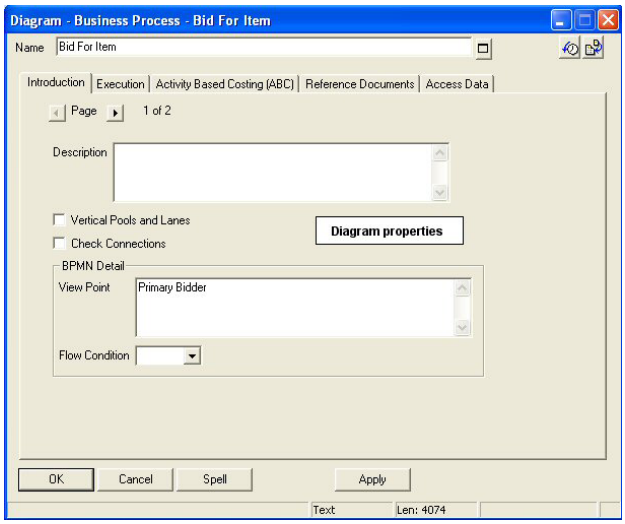
En plus de ceci, l'une des caractéristiques les plus intéressantes de Rational System Architect est la possibilité pour les utilisateurs de personnaliser et d'étendre le métamodèle sous-jacent de stockage des informations dans une encyclopédie. Le métamodèle par défaut d'une encyclopédie Rational System Architect est spécifié dans un fichier nommé SAPROPS.CFG (le principal fichier de propriétés de **S**ystem **A**rchitect), lequel contrôle des éléments tels que les symboles figurant dans un diagramme, la relation entre symboles et leurs définitions, et les propriétés des symboles, définitions et diagrammes. Les modifications apportées par l'utilisateur au métamodèle sont spécifiées dans un fichier texte nommé USRPROPS.TXT, qui, lors du chargement d'une encyclopédie, est analysé de pair avec le fichier SAPROPS.CFG afin de créer un fichier SAPROPS.BIN. Le fichier USRPROPS.TXT remplace le fichier SAPROPS.CFG. Vous pouvez modifier le fichier USRPROPS.TXT afin de personnaliser ou d'étendre le métamodèle d'une encyclopédie en utilisant un langage de script natif à Rational System Architect.

Métamodèle d'encyclopédie de Rational System Architect

Ce que fournit le métamodèle

Le métamodèle est un modèle de la manière dont Rational System Architect stocke les diagrammes, les symboles et les définitions que vous créez au cours de votre travail. Le métamodèle de Rational System Architect inclut tous les types de diagramme, de symbole et de définition, les propriétés que chacun de ces types contiennent et diverses relations entre ces éléments de modélisation.

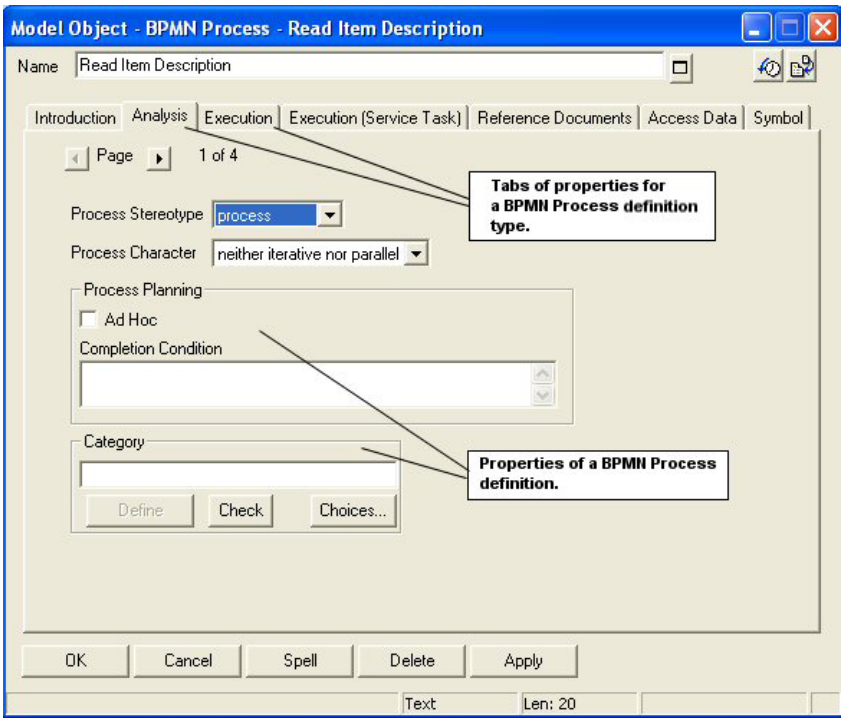
Un diagramme Processus métier constitue un exemple de *type de diagramme* ; il comporte des *propriétés* telles que des options permettant d'afficher des pools et des couloirs d'activité à la verticale ou à l'horizontale (Pools et couloirs d'activité verticaux), de vérifier automatiquement les connexions ligne-symbole sur le diagramme lors du traçage (Vérifier les connexions), etc.



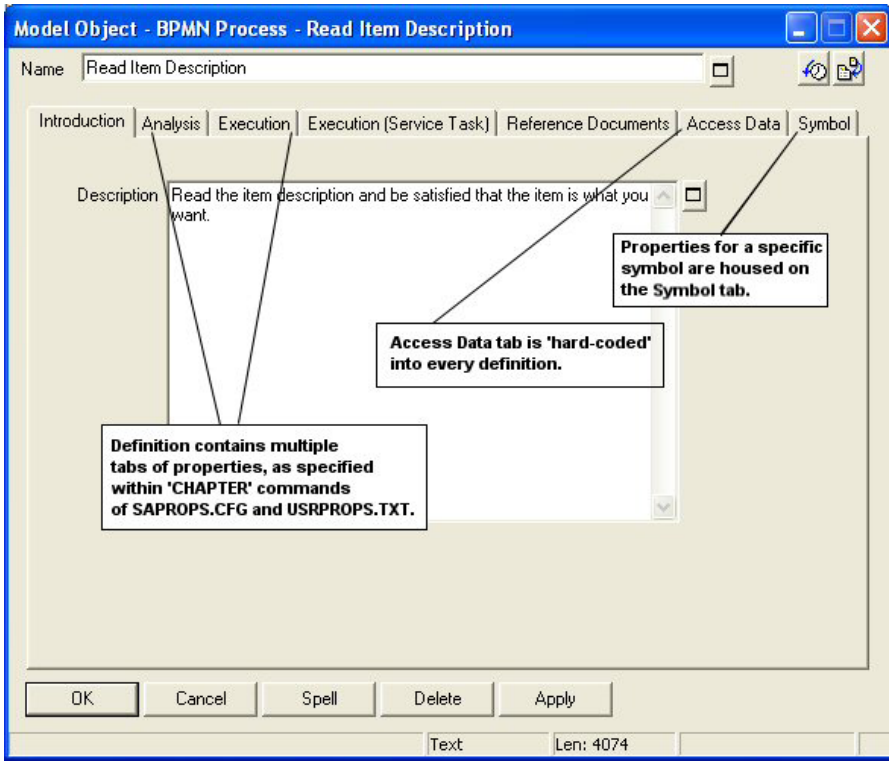
Le symbole Processus BPMN constitue un exemple de *type de symbole*. Un symbole Processus BPMN est tracé sur un diagramme Processus métier – le diagramme *contient* des symboles et les symboles sont *contenus dans* un diagramme

– ceci est un exemple de deux des nombreuses relations dans le métamodèle d'encyclopédie. Une définition Processus BPMN constitue un exemple de *type de définition*. Un symbole BPMN représente graphiquement une définition de Processus BPMN. La plupart des définitions sont représentées par un symbole ; d'autres ne le sont pas – les types de définition d'attribut ou de méthode, par exemple, ne sont pas représentés par un symbole dans un diagramme quelconque. Tous deux sont *inclus dans* (autre relation) un type de définition de classe.

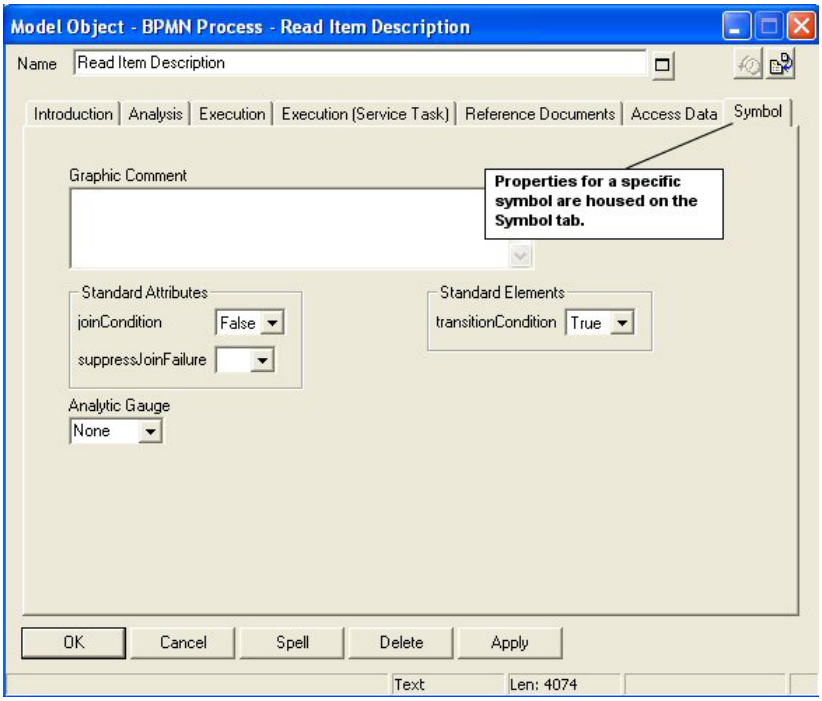
De manière similaire au type diagramme, chaque type de définition contient des propriétés. Si vous ouvrez une définition dans l'explorateur de Rational System Architect, vous pouvez voir ces propriétés dans la boîte de dialogue de la définition, classées sous les onglets et les groupes appropriés.



De manière similaire au types diagramme et définition, chaque type symbole contient des propriétés. Si vous ouvrez la boîte de dialogue sous-jacente d'un symbole (en effectuant un double clic sur le symbole dans l'espace de travail d'un diagramme ou en cliquant avec le bouton droit de la souris et en sélectionnant Editer, ou encore en le sélectionnant et en choisissant Editer, type-symbole), vous pouvez voir les propriétés de la définition sous-jacente que représente le symbole (les mêmes qui vous seraient présentées en ouvrant la définition depuis l'explorateur), **ainsi** qu'un onglet Symbole supplémentaire.



L'onglet Symbole contient des propriétés spécifiques au symbole.



Chaque symbole que vous tracez sur un diagramme constitue une instance distincte qui pointe vers la même définition. Vous pouvez donc tracer un symbole nommé Lecture de la description de l'élément sur un diagramme Processus métier, et dont la couleur sera rouge, et un autre nommé également Lecture de la description de l'élément sur un autre diagramme processus métier, et dont la couleur sera vert. Si vous apportez une modification à la définition Lecture de la description de l'élément (en ajoutant, par exemple, un mot à sa propriété Description), cette modification sera reflétée lorsque vous ouvrirez la définition du symbole en rouge, tout comme en vert, nommé Lecture de la description de l'élément. Vous avez donc deux symboles distincts pour une seule définition sous-jacente.

Les relations entre, et à l'intérieur, des types de diagrammes, de symboles et de définitions peuvent être complexes. Par exemple, dans Rational System Architect, un diagramme de classe appartient au package dans lequel il a été créé. Une relation ‘appartient à’ existe entre une classe et un package. De plus, une classe est ‘indexée au’ package auquel elle appartient. Cette relation assure l'unicité de l'espace nom d'une classe – Vous pouvez avoir une classe Personne dans un package Ressources_humaine avec un contenu totalement différent de celui d'une classe Personne dans un package Réservation_Hôtel. Ainsi, ‘indexé par’ représente une autre relation existant entre une classe et un package. De manière similaire, une méthode appartient à une classe qui elle-même appartient à un package. Une méthode est également indexée par sa classe, qui est elle-même indexée par son package. De plus, l'utilisateur peut créer des diagrammes enfant (comme un diagramme Etat) pour les symboles de classe sur ce diagramme de classe. Dans ce cas, un diagramme Etat sera ‘enfant de’ d'un symbole de classe – ce qui constitue encore une autre relation.

Comment modifier le métamodèle

Rational System Architect vous est livré avec un métamodèle prédéfini de diagrammes, symboles, définitions, propriétés et relations. Vous pouvez l'utiliser tel quel, ou bien l'étendre ou le personnaliser pour l'adapter à vos besoins de modélisation. La personnalisation peut impliquer la modification des éléments déjà fournis ou l'ajout de vos propres diagrammes, types de symboles et de définitions.

Composition physique d'un métamodèle d'encyclopédie – SAPROPS.CFG et USRPROPS.TXT

Le métamodèle de chaque encyclopédie Rational System Architect est spécifié par deux fichiers : SAPROPS.CFG (fichier de configuration des propriétés System Architect) et USRPROPS.TXT (fichier des propriétés utilisateur). Ces deux fichiers résident dans la table FILES de chaque encyclopédie.

Le fichier SAPROPS.CFG contient le métamodèle par défaut spécifié par IBM pour chaque encyclopédie utilisée avec une version donnée du produit. Le fichier USRPROPS.TXT par défaut est un fichier vide, hormis quelques commentaires (identifiés par les lettres REM). Les utilisateurs doivent ajouter le code approprié au fichier USRPROPS.TXT pour modifier le métamodèle.

Lorsque Rational System Architect ouvre une encyclopédie, il analyse le fichier SAPROPS.CFG, puis le fichier USRPROPS.TXT, afin de créer un fichier SAPROPS.BIN. Les éléments spécifiés dans USRPROPS.TXT se substituent ou sont ajoutés à la spécification SAPROPS.CFG lors de la création du fichier SAPROPS.BIN. Le fichier SAPROPS.BIN est celui utilisé pour présenter le métamodèle à l'utilisateur.

Le métamodèle comporte quelques éléments importants que vous ne pouvez pas remplacer dans SAPROPS.CFG à l'aide de USRPROPS.TXT :

- Les références de type LIST ou LISTONLY définies dans SAPROPS ne peuvent pas être supprimées. Toutefois, vous pouvez modifier le texte à afficher dans la liste ou la zone de liste.

- Les étiquettes définies dans SAPROPS ne peuvent pas être supprimées. Toutefois, vous pouvez modifier le texte à afficher dans l'étiquette.

Les fichiers 'maîtres' de SAPROPS.CFG & USRPROPS.TXT

Outre leur présence dans la table FILES de chaque encyclopédie, une copie 'maîtresse' des fichiers SAPROPS.CFG et USRPROPS.TXT est également fournie dans le répertoire d'exécution principal de Rational System Architect (généralement <C>:\Program Files\IBM\Rational\System Architect Suite\11.3.1\System Architect). Lors de la création initiale d'une encyclopédie, les copies 'maîtresses' des fichiers SAPROPS.CFG et USRPROPS.TXT résidant dans le répertoire d'exécution de Rational System Architect sont placées automatiquement dans sa table Files. Par conséquent, si vous modifiez le contenu du fichier USRPROPS.TXT dans le répertoire principal de Rational System Architect, vous modifiez alors le métamodèle de toutes les nouvelles encyclopédies que vous créez. C'est pourquoi de nombreux utilisateurs s'assurent que le fichier USRPROPS.TXT 'maître' situé dans le répertoire d'exécution principal de Rational System Architect contienne toutes les propriétés requises par les normes de leur entreprise et de leurs projets.

A l'origine, le fichier USRPROPS.TXT 'maître' est essentiellement vide – il ne contient que quelques remarques au début du fichier, précédées de la commande **REM** (rappel ou commentaire).

Fichier CONFIG.PRP

Rational System Architect fournit un troisième fichier, intitulé CONFIG.PRP, qui est la copie exacte de SAPROPS.CFG. CONFIG.PRP est situé dans le répertoire d'exécution de Rational System Architect (généralement <C>:\Program Files\IBM\Rational\System Architect Suite\11.3.1\System Architect). CONFIG.PRP vous est fourni afin que vous puissiez visualiser, copier et coller les commandes et les propriétés figurant également dans le fichier SAPROPS.CFG sans risque d'endommager ce dernier par inadvertance. Vous pouvez copier et coller des commandes depuis le fichier CONFIG.PRP et les coller dans le fichier USRPROPS.TXT, puis leur apporter des modifications.

Sélection des ensembles de diagrammes et de propriétés pour une encyclopédie

A part modifier le métamodèle via USRPROPS.TXT, vous pouvez également sélectionner à tout moment quels ensembles de diagrammes et de propriétés sont activés pour une encyclopédie via la boîte de dialogue de configuration de propriétés (à laquelle vous pouvez accéder en sélectionnant Outils, Personnaliser le support des méthodes, Configuration de l'encyclopédie).

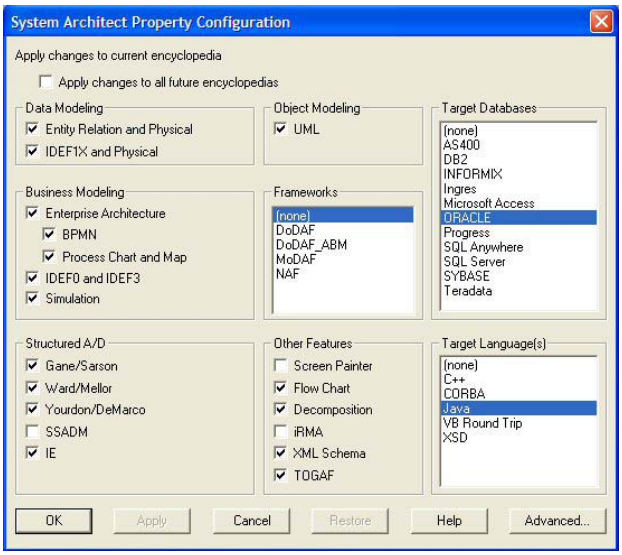


Figure 1-1. Boîte de dialogue Configuration de projet : sélectionnez les types de diagrammes et les autres diagrammes utiles pour cette encyclopédie.

Vous pouvez activer ou désactiver des ensembles de diagrammes et de propriétés, et cliquer sur le bouton Avancé dans cette boîte de dialogue pour apporter d'autres perfectionnements aux ensembles de diagrammes et de propriétés actifs dans une encyclopédie.

SADECLAR.CFG

Les sélections effectuées dans la boîte de dialogue Configuration des propriétés affectent directement le contenu du fichier SADECLAR.CFG, lequel réside dans la table Files de chaque encyclopédie. Ce fichier est à son tour référencé

par des instructions #IFDEF (remarque : '#' et 'IF' ne sont pas séparés par un espace) dans les fichiers SAPROPS.CFG et USRPROPS.TXT. Par exemple, le fichier SAPROPS.CFG contient des instructions # IFDEF pour UML – si la méthode de modélisation UML est activée (dans le dialogue ci-dessus et donc dans SADECLAR.CFG), ces instructions # IFDEF dans SAPROPS.CFG activent ou désactivent en conséquence les propriétés correspondantes pour les diagrammes UML.

Comme illustré par l'image ci-dessous, les sélections effectuées pour les ensembles de diagrammes et de propriétés dans la boîte de dialogue Configuration des propriétés (qui active ou désactive des options dans SADECLAR.CFG) filtrent de fait les propriétés du fichier SAPROPS.CFG utilisées pour l'encyclopédie. Les modifications que vous apportez à USRPROPS.TXT sont analysées avec le fichier SAPROPS.CFG filtré afin de générer un fichier SAPROPS.BIN fournissant le métamodèle d'une encyclopédie lors de l'exécution de Rational System Architect. Chaque fois que vous ouvrez une boîte de dialogue de propriétés ou de définition, ou exécutez un rapport, Rational System Architect accède à SAPROPS.BIN pour rechercher les propriétés pertinentes de l'élément de modèle que vous définissez.

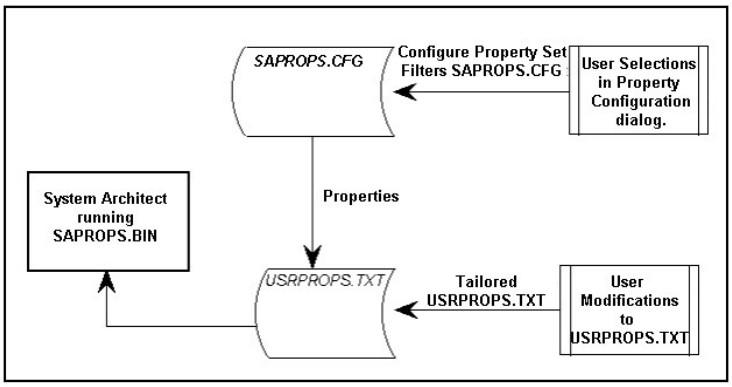


Figure 1-2. Relation entre SAPROPS.CFG, USRPROPS.TXT et les sélections de diagrammes, propriétés et technique de modélisation par l'utilisateur.

Vous pouvez exporter le fichier SADECLAR.CFG depuis la table Files d'une encyclopédie et l'ouvrir avec un éditeur de texte quelconque pour identifier les ensembles de propriétés spécifiques que vous pouvez activer ou désactiver par le biais d'instructions # IFDEF dans USRPROPS.TXT.

Certains ne correspondent pas exactement aux termes/libellés utilisés dans la boîte de dialogue Configuration des propriétés. Par exemple, l'option Enterprise Architecture est en fait dénommée Business Enterprise dans SADECLAR.CFG. Par conséquent, une instruction #IFDEF “Enterprise Architecture” dans USRPROPS.TXT n'aurait pas de sens et produirait une erreur à l'analyse ; l'instruction correcte devrait être #IFDEF “Business Enterprise”.

Figure 1-3. Le contenu de SADECLAR est utilisé pour bascule des instructions # IFDEF dans USRPROPS.TXT.

```
DECLARE "Business Enterprise" UNDEFINED
DECLARE " UML Class" DEFINED
DECLARE " UML State" DEFINED
DECLARE " UML Sequence" DEFINED
DECLARE " UML Collaboration" DEFINED
DECLARE " UML Component" DEFINED
DECLARE " UML Deployment" DEFINED
DECLARE " UML Use Case" DEFINED
DECLARE " UML Activity" DEFINED
DECLARE "UML Object-oriented" DEFINED
DECLARE " System Architecture" UNDEFINED
DECLARE " System Area Map" UNDEFINED
```

2

Modification du métamodèle à l'aide du fichier USRPROPS.TXT

Introduction

Le présent chapitre décrit la théorie et les mécanismes qui se cachent derrière le métamodèle extensible de Rational System Architect.

Accès au fichier USRPROPS.TXT et édition de ce fichier

Le fichier USRPROPS.TXT peut être édité dans tout éditeur de texte. Il doit seulement être sauvegardé au format Texte.

Accès au fichier maître USRPROPS.TXT

Comme mentionné précédemment dans ce chapitre, le fichier USRPROPS.TXT maître est automatiquement placé dans toute encyclopédie que vous créez. De nombreuses organisations modifient le fichier USRPROPS.TXT maître pour que toutes les nouvelles encyclopédies contiennent les mêmes extensions de métamodèle. Pour éditer le fichier USRPROPS.TXT maître :

- Sélectionnez Outils, Personnaliser les propriétés utilisateur, Editer USRPROPS.TXT (Maître) ou
- Accédez simplement au répertoire <C>:\Program Files\IBM\Rational\System Architect Suite\11.3.1\System Architect et ouvrez le fichier USRPROPS.TXT qui s'y trouve.

Accès au fichier USRPROPS.TXT d'une encyclopédie

Le fichier USRPROPS.TXT d'une encyclopédie se trouve dans la table Fichiers de la base de données SQL Server de l'encyclopédie. Pour l'éditer, vous devez d'abord l'exporter de la table Fichiers de la base de données. Ensuite, une fois que vous l'avez édité, vous devez le réimporter dans la table Fichiers de la base de données, puis rouvrir votre encyclopédie (pour que les fichiers SAPROPS.CFG et USRPROPS.TXT puissent être analysés).

Il existe plusieurs façons d'accéder au fichier USRPROPS.TXT d'une encyclopédie.

- Vous pouvez utiliser la fonction native d'exportation/importation du fichier USRPROPS.TXT de Rational System Architect (sélectionnez Outils, Personnaliser les propriétés utilisateur, Exporter USRPROPS.TXT (Encyclopédie),

- Vous pouvez utiliser la fonction **Gestionnaire de fichiers de l'encyclopédie** de Rational System Architect (sélectionnez Outils, Gestionnaire de fichiers de l'encyclopédie) ou
- Vous pouvez utiliser **SAEM** (en dehors de Rational System Architect, sélectionnez Démarrer, Programmes, IBM Rational, IBM Rational Lifecycle Solutions Tools, IBM Rational System Architect 11.3.1, SAEM et reportez-vous à l'aide de SAEM).

Utilisation de la fonction native d'exportation/importation du fichier USRPROPS.TXT de Rational System Architect :

Pour éditer le fichier USRPROPS.TXT à l'aide de la fonction native d'exportation/importation du fichier USRPROPS.TXT de Rational System Architect, effectuez les étapes suivantes :

1. Sélectionnez Outils, Personnaliser les propriétés utilisateur, Exporter USRPROPS.TXT (Encyclopédie).
2. Dans la boîte de dialogue **Exporter les propriétés utilisateur** qui s'ouvre, sélectionnez un répertoire vers lequel exporter le fichier USRPROPS.TXT. Cliquez sur le bouton Sauvegarder ; le fichier USRPROPS.TXT est sauvegardé dans le répertoire sélectionné et s'ouvre automatiquement dans le bloc-notes.
3. Une fois que vous avez édité le fichier, sélectionnez Outils, Personnaliser les propriétés utilisateur, Importer USRPROPS.TXT (Encyclopédie) pour réimporter le fichier USRPROPS.TXT modifié dans la table Fichiers de la base de données de l'encyclopédie.
4. Rouvrez l'encyclopédie afin qu'elle puisse analyser son fichier SAPROPS.CFG et son fichier USRPROPS.TXT modifié.

Utilisation du gestionnaire de fichiers de l'encyclopédie :

Pour éditer le fichier USRPROPS.TXT à l'aide du gestionnaire de fichiers de l'encyclopédie, procédez comme suit :

1. Sélectionnez Outils, Gestionnaire de fichiers d'encyclopédie.
2. Dans la boîte de dialogue Gestionnaire de fichiers de l'encyclopédie, vérifiez que l'option Exporter est activée, dans le coin inférieur gauche. Sélectionnez le fichier USRPROPS.TXT dans la liste Sélectionner un fichier à exporter et sélectionnez un répertoire d'exportation du fichier, à l'aide du bouton '…' de la propriété Exporter le fichier sélectionné dans.
3. Modifiez le fichier dans un éditeur de texte et utilisez le gestionnaire de fichiers de l'encyclopédie pour réimporter le fichier dans la table Fichiers de la base de données de l'encyclopédie.
4. Rouvrez l'encyclopédie afin qu'elle puisse analyser son fichier SAPROPS.CFG et son fichier USRPROPS.TXT modifié.

Utilisation de SAEM :

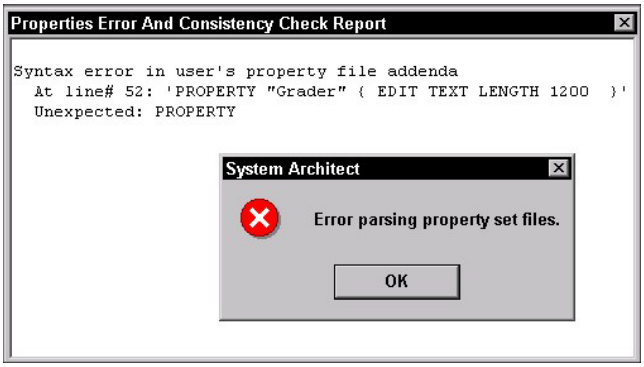
Vous pouvez également accéder au fichier USRPROPS.TXT d'une encyclopédie et l'éditer à l'aide de SAEM. Reportez-vous à l'aide de SAEM pour des instructions sur la manière de se connecter à un serveur, de sélectionner une base de données et d'exporter/importer des fichiers depuis/vers la base de données.

Rechargement des fichiers de propriétés

Comme mentionné dans les étapes précédentes, chaque fois que vous réimportez un fichier USRPROPS.TXT modifié dans une encyclopédie, vous devez rouvrir cette dernière à l'aide de Rational System Architect. Lorque vous rouvrez l'encyclopédie, ses fichiers SAPROPS.CFG et USRPROP.TXT sont analysés et un fichier SAPROPS.BIN (binaire) est créé ; ce fichier est utilisé pour présenter le métamodèle. En l'absence d'erreur, les modifications apportées au métamodèle sont appliquées immédiatement.

Si, lors de l'analyse syntaxique du fichier USRPROPS.TXT, Rational System Architect rencontre des erreurs dans le code du fichier USRPROPS.TXT, il génère un avertissement ou un message d'erreur. Rational System Architect ouvre l'encyclopédie après un avertissement, mais ne l'ouvre **pas** en cas d'erreur. Un message tel que le suivant est affiché :

Figure 2-1. Boîte de dialogue des erreurs des fichiers de propriétés



En cas d'erreur, vous ne pourrez pas accéder au fichier USRPROPS.TXT erroné à l'aide de la fonction native d'exportation/importation du fichier USRPROPS.TXT de Rational System Architect (si vous sélectionnez Outils, Personnaliser les propriétés utilisateur, l'option Exporter USRPROPS.TXT (Encyclopédie) est grisée).

Pour accéder au fichier USRPROPS.TXT et l'éditer lorsqu'une erreur s'est produite, vous devez utiliser le gestionnaire de fichiers de l'encyclopédie de Rational System Architect (sélectionnez Outils, Gestionnaire de fichiers de l'encyclopédie) ou SAEM (en dehors de Rational System Architect, sélectionnez Démarrer, Programmes, IBM Rational, IBM Rational Lifecycle Solutions Tools, IBM Rational System Architect 11.3.1, SAEM et reportez-vous à l'aide de SAEM).

Composition et syntaxe

Comme pour la plupart des langages de programmation, la syntaxe de langage du fichier USRPROPS.TXT est composée d'une série de chaînes. Au moins un *espace blanc* est requis pour séparer les chaînes les unes des autres (l'espace blanc inclut les espaces, les tabulations, les virgules, les retours chariot/retours à la ligne et certains autres séparateurs). Si plusieurs caractères d'espace blanc se suivent (par exemple, un retour chariot suivi d'une tabulation) ils sont regroupés et traités comme un seul espace blanc.

Si une chaîne inclut un ou plusieurs espaces imbriqués, placez bien cette chaîne entre guillemets. Par exemple, utilisez **"Flux de données"** et non **Flux de données**.

Mots clés

Le langage du fichier USRPROPS.TXT contient un ensemble particulier de ***mots clés***. Suivant sa position, un mot clé est considéré comme une ***commande*** ou un ***argument***. Tous les mots clés autorisés dans le fichier USRPROPS.TXT sont répertoriés dans le **Chapitre 3, Mots clés d'USRPROPS.TXT**.

Insensibilité des mots clés à la casse

Les mots clés du fichier USRPROPS.TXT ne sont ***pas*** sensibles à la casse ; vous pouvez utiliser des majuscules, des minuscules ou les deux. Dans le présent document et dans l'exemple de fichier USRPROPS.TXT, les commandes et tous les autres mots clés sont en majuscules, uniquement pour des raisons de lisibilité. Voici quelques exemples de commande :

BEGIN, Begin ou BegiN
EDIT ou Edit
LIST, List ou LiST
LISTONLY, Listonly ou ListOnly
RENAME, Rename ou ReName, etc

Commandes

Les commandes sont *toujours* des mots clés et commencent *toujours* une nouvelle phrase. Lorsque Rational System Architect analyse le fichier USRPROPS.TXT, il sait que la première chaîne du fichier doit être une commande de mot clé valide. Chaque commande doit être suivie d'un nombre

connu de chaînes d'arguments (zéro, une ou plusieurs), puis d'une autre commande.

Arguments

Les chaînes qui suivent les commandes sont des arguments. Certains arguments peuvent être des mots clés. Les autres arguments comprennent les chaînes de texte qui fournissent entre autres les noms des diagrammes, symboles, définitions, propriétés, listes de valeurs, libellés et chaînes d'aide, qui apparaissent dans les boîtes de dialogue suivantes. En voici quelques exemples :

- LIST **"Planification Processeur"**

  "Planification Processeur" n'est pas un mot clé. Il est utilisé comme argument dans l'expression ci-dessus.

- DISPLAY { FORMAT **KEY** LEGEND "Données de clé" }

  "KEY" est un mot clé. Il est utilisé comme argument dans l'expression ci-dessus.

Respect de la casse des arguments qui correspondent à des chaînes de texte

Comme mentionné précédemment, les mots clés ***ne sont pas*** sensibles à la casse. Toutefois les arguments qui correspondent à des chaînes de texte le ***sont***. Par exemple, si l'argument LIST "Planification Processeur" ci-dessus est utilisé, les références à cette liste dans le fichier SAPROPS.CFG ou USRPROPS.TXT doivent être orthographiées exactement de la même manière, avec la même sensibilité à la casse. Par exemple, si nous spécifions la liste suivante :

```
LIST "Planification Processeur"
{
VALUE"préventif"
VALUE"non préventif"
}
```

Une référence valide à cette liste doit alors être orthographiée exactement de la même manière.

```
DEFINITION "Processeur matériel"
PROPERTY "Planification"
{ EDIT text LIST "Planification Processeur" LENGTH 20
DISPLAY { LEGEND "Planif." } }
```

Toutefois, la syntaxe ci-après génère un message d'erreur indiquant 'List "PLANIFICATION PROCESSEUR" not Found.'

```
DEFINITION "Processeur matériel"
PROPERTY "Planification"
{ EDIT text LIST "PLANIFICATION PROCESSEUR" LENGTH
20
DISPLAY { LEGEND "Planif." } }
```

De la même manière, les propriétés référencées dans des rapports doivent utiliser l'orthographe et la casse de l'entrée dans les fichiers SAPROPS.CFG et/ou USRPROPS.TXT.

Regroupement de commandes pour créer des éléments de modélisation

Les accolades ouvrantes et fermantes, **{ }**, ou les commandes BEGIN...END permettent de regrouper des commandes pour constituer des éléments de modélisation.

Diagrammes, symboles et définitions

Le référentiel de Rational System Architect prend en charge trois éléments de modélisation principaux, parfois appelés *classes de dictionnaire*– ***diagrammes***, ***symboles*** (représentés sur des diagrammes) et ***définitions*** (qui peuvent ou non être représentées par des symboles). La structure BEGIN .. END ou { } permet de spécifier le contenu de ces éléments de modélisation, comme suit :

Diagram "Nom du type de diagramme**"**
{
[contenu]
}

Symbol "Nom du type de symbole**"**
{
[contenu]
}

Definition "Nom du type de définition**"**
{
[contenu]
}

ou

Diagram "Nom du type de diagramme**"**
BEGIN
[contenu]
END

Etc

Propriétés

Ces éléments de modélisation contiennent des propriétés et des commandes de présentation. La structure BEGIN .. END ou { } permet de regrouper des commandes de propriété, de la sorte :

```
Definition "Nom du type de définition"
{
PROPERTY { [spécification de la propriété]  }
PROPERTY { [spécification de la propriété]  }
…
}
```

Certains mots clés qui créent des clauses dans une propriété requièrent également des accolades ouvrantes et fermantes pour délimiter les arguments de la commande, comme c'est le cas de la commande KEYED BY.

```
Definition "Nom du type de définition"
{
PROPERTY { [spécification de la propriété] KEYED BY {
[clause] }  }
…
}
```

Présentation

La commande LAYOUT requiert également des accolades ouvrantes et fermantes ou une instruction BEGIN .. END.

```
Definition "Nom du type de définition"
{
LAYOUT { [spécification de la présentation] }
PROPERTY { [spécification de la propriété]  }
PROPERTY { [spécification de la propriété]  }
…
}
```

Chapitre

Les propriétés d'une boîte de dialogue peuvent être regroupées plus spécifiquement en pages et groupes. Les pages sont spécifiées par une commande CHAPTER ; la commande CHAPTER ne requiert pas et d'ailleurs ***ne doit pas comporter*** d'accolades ouvrantes et fermantes ou d'instructions BEGIN .. END. Elles regroupe simplement toutes les propriétés qui en dépendent dans une spécification à l'intérieur d'une page (dans la boîte de dialogue où elle est émise), jusqu'à la prochaine commande CHAPTER de la spécification.

```
Definition "Nom du type de définition"
{
LAYOUT { [spécification de la présentation] }
CHAPTER "Première page"
PROPERTY { [spécification de la propriété]  }
PROPERTY { [spécification de la propriété]  }
…
CHAPTER "Deuxième page"
PROPERTY { [spécification de la propriété]  }
PROPERTY { [spécification de la propriété]  }
…
}
```

Groupes

Contrairement à la commande CHAPTER, les commandes GROUP ***requièrent*** des accolades ouvrantes et fermantes ou une instruction BEGIN .. END.

```
Definition "Nom du type de définition"
{
LAYOUT { [spécification de la présentation] }
CHAPTER "Première page"
PROPERTY { [spécification de la propriété]  }
PROPERTY { [spécification de la propriété]  }
GROUP "Choses qui vont ensemble"
{
PROPERTY { [spécification de la propriété]  }
PROPERTY { [spécification de la propriété]  }
}
…
}
```

Listes

Vous pouvez également spécifier des listes dans une encyclopédie (des listes prédéfinies contenant des valeurs textuelles ou des listes de définitions que vous créez lors de la modélisation). Les listes de définitions que vous créez lors de la modélisation sont générées à l'aide des commandes ONE OF, LISTOF et EXPRESSIONOF dans une instruction de propriété et sont abordées plus loin. Les listes textuelles sont générées en spécifiant les valeurs de la liste dans une instruction de liste distincte et en plaçant ces valeurs entre des accolades ouvrantes/fermantes ou dans une structure BEGIN.. END. L'instruction LIST est normalement placée près du début du fichier USRPROPS.TXT et référencée dans

la spécification de propriété appropriée d'un diagramme, d'un symbole ou d'une définition.

```
LIST "Liste de choses"
{
VALUE Un
VALUE Deux
VALUE "Deux et demi"
}
```

Remarque sur la syntaxe

Les indentations et les nouvelles lignes ne sont utilisées que pour améliorer la lisibilité et n'ont aucune signification dans le processeur USRPROPS.TXT autre que celle d'agir comme séparateurs (espaces blancs) entre les chaînes. L'exemple ci-dessus peut être écrit comme suit :

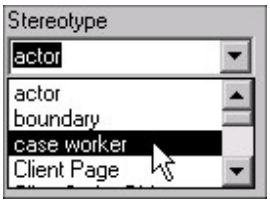
```
LIST "Liste de choses"  {   VALUE Un   VALUE
Deux   VALUE "Deux et demi"   VALUE }
```

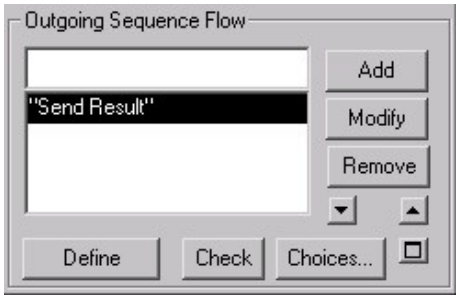
Ce format est tout à fait acceptable pour Rational System Architect, mais il rend certainement plus difficile la maintenance du fichier USRPROPS.TXT et doit donc être évité.

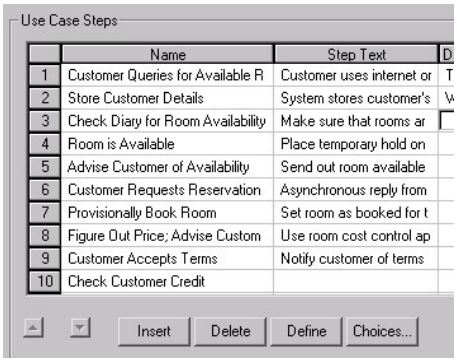
Contrôles de boîte de dialogue

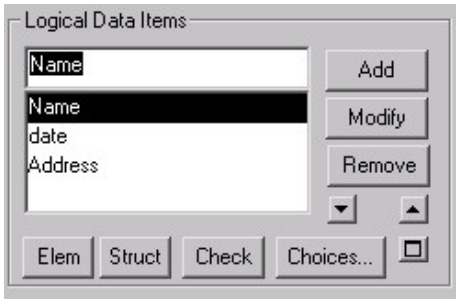
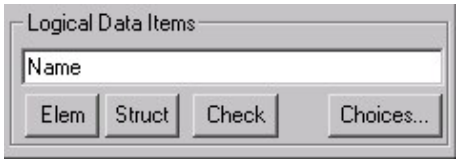
Le tableau ci-après décrit les contrôles de boîte de dialogue qui peuvent être créés via les commandes appropriées dans le fichier USRPROPS.TXT.

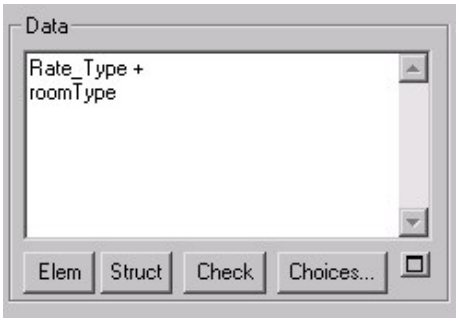
Tableau 2-2. Contrôles générés à partir des expressions de propriété

Type d'argument	Contrôle généré
Commande LIST de moins de cinq valeurs.	Zone de groupe avec un bouton d'**option** pour chaque valeur.  **Remarque importante :** Vous pouvez imposer qu'une liste de moins de cinq valeurs soit une liste déroulante si vous utilisez la commande LISTONLYCOMBO. Des informations supplémentaires sur ce mot clé sont fournies dans le Chapitre 3.
Commande LIST de plus de cinq valeurs.	Zone de liste déroulante. 

BOOLEAN	Case à cocher. La valeur True est représentée par une coche ; la valeur false, par une zone vide. Par exemple, la propriété ci-après, Virtuel, est décrite comme une valeur booléenne : PROPERTY "Virtuel" { EDIT **BOOLEAN** LENGTH 1 DEFAULT "F" } 
LISTOF "[Type de définition ou de diagramme]"	Groupe incluant une zone de liste déroulante avec les boutons **Nouveau**, **Ajouter**, **Supprimer**, **Vers le bas** et **Vers le haut** sur le côté et les trois boutons : **Définition**, **Vérifier** et **Choix** en bas. 
ONEOF "[Type de définition ou de diagramme]"	Groupe incluant une zone de texte et trois boutons : **Définition**, **Vérifier** et **Choix.** 

ASGRID	Cette commande est utilisée avec les commanes LISTOF ou ONEOF pour offrir une grille de valeurs. Par exemple : PROPERTY "Etapes de cas d'utilisation" { EDIT COMPLETE LISTOF "Etapes de cas d'utilisation" KEYED BY { "Package", "Nom du cas d'utilisation":Nom, Nom} **ASGRID** LENGTH 1200 } 
EXPRESSIONOF "[Type de définition ou de diagramme]"	Impossible.

LISTOF DATA	DATA est un mot spécial : il fournit une liste d'éléments de données et de structures de données (chaque structure de données est un groupe d'éléments de données) dans l'encyclopédie. Un contrôle LISTOF DATA est très spécial ; il s'agit d'un groupe comprenant une zone de liste déroulante avec les boutons **Nouveau**, **Ajouter**, **Supprimer**, **Vers le bas** et **Vers le haut** sur le côté et les quatre boutons **Elém**, **Struct**, **Vérifier** et **Choix** en bas. 
ONEOF DATA	Comme mentionné ci-dessus, DATA est un mot spécial : il fournit une liste d'éléments de données et de structures de données dans l'encyclopédie. La clause ONEOF DATA fournit un contrôle de groupe qui contient principalement une zone de texte dans laquelle vous spécifiez l'élément de données ou la structure de données et quatre boutons : **Elém**, **Struct**, **Vérifier** et **Choix**. 

EXPRESSIONOF DATA	Comme mentionné ci-dessus, DATA est un mot spécial : il fournit une liste d'éléments de données et de structures de données dans l'encyclopédie. La commande EXPRESSIONOF DATA fournit une zone de texte dans laquelle vous saisissez les éléments de données ou les structures de données et quatre boutons : **Elém**, **Struct**, **Vérifier** et **Choix**. 

Classement et présentation des modifications apportées au fichier USRPROPS.TXT

Le classement général des sections de SAPROPS.CFG est le suivant :

- **LIST**
- **DIAGRAM**
- **SYMBOL**
- **DEFINITION**

  LAYOUT (valeur par défaut pour l'intégralité de la boîte de dialogue de définition)

  CHAPTER

  GROUP

  LAYOUT

  PROPERTY

Toutes les entrées du fichier USRPROPS.TXT sont facultatives, mais vous devez respecter une présentation similaire à celle du fichier SAPROPS.CFG, en ajoutant une section de commande **RENAME**, si elle est utilisée, au début du fichier. Le classement général des sections du fichier USRPROPS.TXT doit être le suivant :

- **RENAME** (dans cette section, vous renommez USER DIAGRAMS, USER SYMBOLS et USER DEFINITIONS pour créer vos propres types de diagramme, symbole ou définition (voir page 2-32)
- **LIST** (voir page 2-**Error! Bookmark not defined.**)
- **DIAGRAM** (voir page 2-61)
- **SYMBOL** (voir page 2-63)
- **DEFINITION** (voir page 2-67)

CHAPTER (voir page 2-98)
GROUP (voir page 2-101)
LAYOUT (voir page 2-101)
PROPERTY (voir page 2-71)

.

Règles de modification du fichier USRPROPS.TXT

Les règles suivantes doivent être gardées à l'esprit lors de la création du fichier USRPROPS.TXT :

1. Les entrées du fichier USRPROPS.TXT correspondent à des entrées supplémentaires ou des remplacements d'entrées dans le fichier SAPROPS.CFG.
2. L'entrée du fichier USRPROPS.TXT doit commencer par l'instruction LIST, RENAME, DIAGRAM, SYMBOL ou DEFINITION appropriée.
3. Les entrées du fichier USRPROPS.TXT correspondant à des *entrées supplémentaires* dans le fichier SAPROPS.CFG sont placées à la fin de la section appropriée. Par exemple, un bloc LIST qui ne se trouve pas dans le fichier SAPROPS.CFG est principalement ajouté après tous les autres blocs LIST dans le fichier SAPROPS.CFG.
4. A moins que la commande **CHAPTER** ne soit incluse, les entrées du fichier USRPROPS.TXT sont placées à la fin de la boîte de dialogue appropriée. Par exemple, une nouvelle propriété pour une définition de classe est ajoutée après toutes les autres propriétés dans la boîte de dialogue de définition de la classe.
5. Si une commande **CHAPTER** qui se trouve déjà dans le fichier SAPROPS.CFG est incluse dans le fichier USRPROPS.TXT, les entrées du fichier USRPROPS.TXT sont placées à la fin du chapitre (ou de la page) existant.

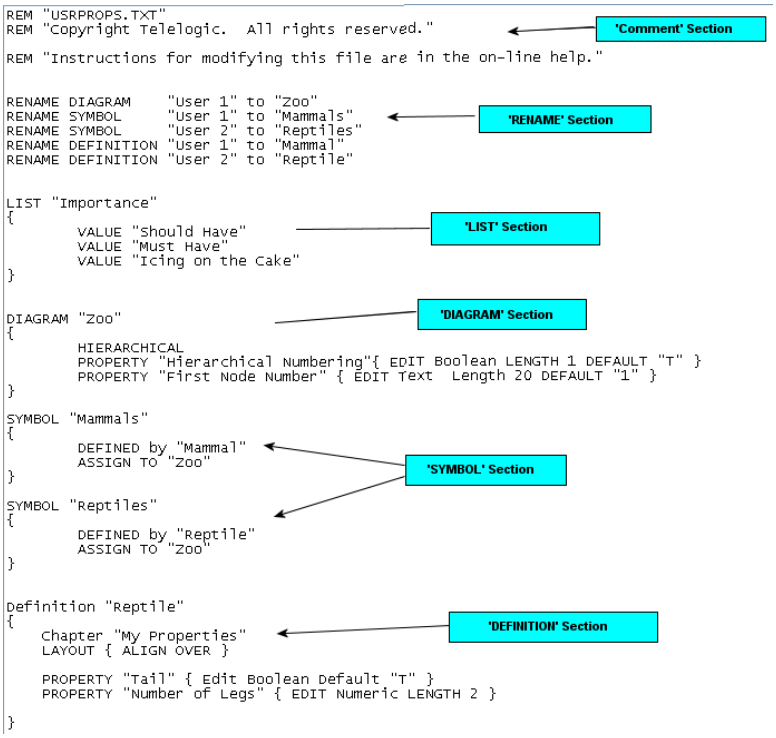
6. Si une commande **GROUP** qui se trouve déjà dans le fichier SAPROPS.CFG est incluse dans le fichier USRPROPS.TXT, les entrées du fichier USRPROPS.TXT sont placées à la fin du groupe existant.
7. La commande **GROUP** génère une zone de groupe, un contrôle Windows standard, dans lequel tous les contrôles suivants doivent être placés. Si le nombre d'entrées est trop important et que la taille du groupe est supérieure à la taille du moniteur, les propriétés en trop ne sont ni incluses, ni affichées. Un message d'avertissement correspondant est affiché à l'ouverture de l'encyclopédie.
8. Si une propriété est ajoutée à un groupe qui contient des commandes **PLACEMENT** dans ses propriétés, dans le fichier SAPROPS.CFG, les commandes **PLACEMENT** doivent également être utilisées pour les nouvelles propriétés ajoutées au fichier USRPROPS.TXT.

Présentation du code du fichier USRPROPS.TXT

Si vous n'avez pas de fichier USRPROPS.TXT ou SAPROPS.CFG, cependant, tous les diagrammes, symboles et définition possèdent tout de même un *nom* et la propriété *description*. Les valeurs par défaut de *description* sont incluses plus loin dans cette section. Comme mentionné précédemment, le texte complet du fichier SAPROPS.CFG est inclus dans le fichier CONFIG.PRP. Il s'agit d'un fichier de texte ASCII standard ; les entrées peuvent être utilisées comme modèles pour les modifications et ajouts dans le fichier USRPROPS.TXT.

Vous choisissez vous-même la manière de présenter le code dans le fichier USRPROPS.TXT. Il est recommandé de fournir une structure sous forme de pages afin de mieux discerner le début des instructions de liste, diagramme, symbole et définition. Toutefois, des éditeurs de texte différents peuvent représenter les pages différemment (par exemple, si vous utilisez Microsoft Word comme éditeur de texte, puis que vous ouvrez le fichier USRPROPS.TXT dans un autre éditeur de texte, les pages que vous avez définies dans Word peuvent être espacées différement.

Figure 2-2. Exemple de présentation de code pour le fichier USRPROPS.TXT



Exemple de modifications dans le fichier USRPROPS.TXT

Dans cette section, nous modifions une définition qui existe déjà dans le fichier SAPROPS.CFG. Le code suivant se trouve dans le fichier SAPROPS.CFG :

```
DEFINITION "Demande de modification"
{
ADDRESSABLE
LAYOUT { COLS 2 TAB ALIGN LABEL }
PROPERTY "Déclaration d'impact" { EDIT Text LENGTH 1000 }
PROPERTY "Source d'origine" { EDIT Text LIST "Unité
commerciale"
    LENGTH 80   LABEL "Dépt. source" }
PROPERTY "Nom de l'auteur" { EDIT Text LENGTH 25 }
PROPERTY "Date entrée" { EDIT date INITIAL date READONLY
    LENGTH 10 }
PROPERTY "Date de début" { EDIT date LENGTH 10 }
PROPERTY "Date d'achèvement requise"
  { EDIT date LENGTH 10  LABEL "Achèvement requis" }
```

L'illustration ci-après représente la boîte de dialogue **Objet dictionnaire** du bloc de définitions ci-dessus (notez que la plupart du temps, nous l'appelons boîte de dialogue de **définition** dans le présent document).

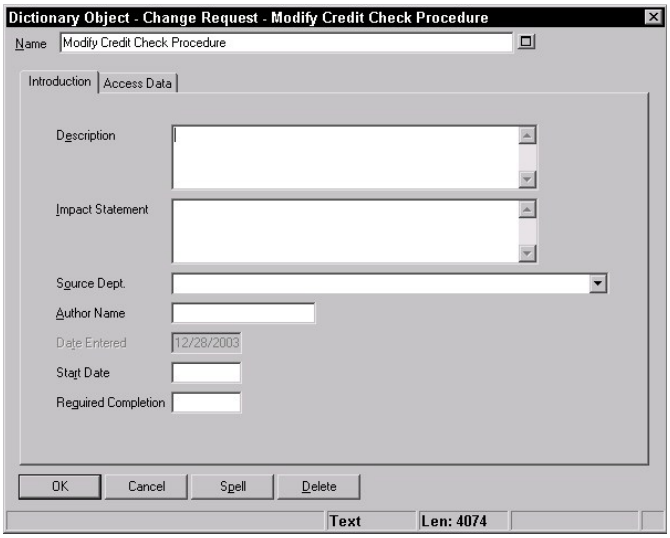


Figure 2-3. Boîte de dialogue de définition Demande de modification, telle qu'elle est définie dans le fichier maître de l'ensemble des propriétés de configuration

Notez qu'il existe une page **Introduction** alors que notre fichier SAPROPS.CFG ne l'a pas appelée spécifiquement à l'aide d'une commande CHAPTER. Si aucune commande CHAPTER n'est spécifiée, Rational System Architect fournit automatiquement une page **Introduction** par défaut. La page **Données d'accès** est codée en dur dans le logiciel ; elle n'est pas spécifiée dans le fichier SAPROPS.CFG.

Modification avec le fichier USRPROPS.TXT

Nous modifions la définition Demande de modification en ajoutant le code suivant au fichier USRPROPS.TXT et en rouvrant l'encyclopédie :

```
DEFINITION "Demande de modification"
{
LAYOUT { COLS 2 TAB ALIGN LABEL }
PROPERTY "Nom de l'auteur" { LABEL "Division client" }
PROPERTY "Gestionnaire de supervision" { EDIT text
LENGTH 45 }
PROPERTY "A l'heure" { Edit Boolean Length 1 DEFAULT "T"
}
}
```

Le tableau explique chaque ligne du code du fichier USRPROPS.TXT ci-dessus et son effet.

Tableau 2-1. Effet des entrées du fichier USRPROPS.TXT

Entrée du fichier USRPROPS.TXT	Effet
DEFINITION "Demande de modification" {	Spécifie une modification de la définition "Demande de modification"
LAYOUT { COLS 2 TAB ALIGN LABEL }	Configurez une présentation en deux colonnes pour les propriétés placées sous la commande LAYOUT (jusqu'à ce que la fin de la définition soit atteinte ou qu'une autre commande LAYOUT soit rencontrée).
PROPERTY "Nom de l'auteur" { LABEL "Division client" }	Cela modifie une propriété existante – Le libellé *Nom de l'auteur* de la zone est changé en *Division client*
PROPERTY "Gestionnaire de supervision" { EDIT TEXT LENGTH 45 }	Cela ajoute une nouvelle propriété, qui est une zone de texte, *Gestionnaire de supervision*, à la boîte de dialogue.
PROPERTY "A l'heure" { EDIT BOOLEAN LENGTH 1 DEFAULT "T" } }	Cela ajoute une nouvelle propriété (une case à cocher) à la boîte de dialogue pour indiquer si la demande de modification respecte l'échéance.

Nous importons notre fichier USRPROPS.TXT modifié dans notre encyclopédie Rational System Architect et rouvrons la définition d'une demande de modification, pour afficher les modifications apportées à sa boîte de dialogue (notez que les informations de la page **Introduction** sont maintenant présentées sur deux pages).

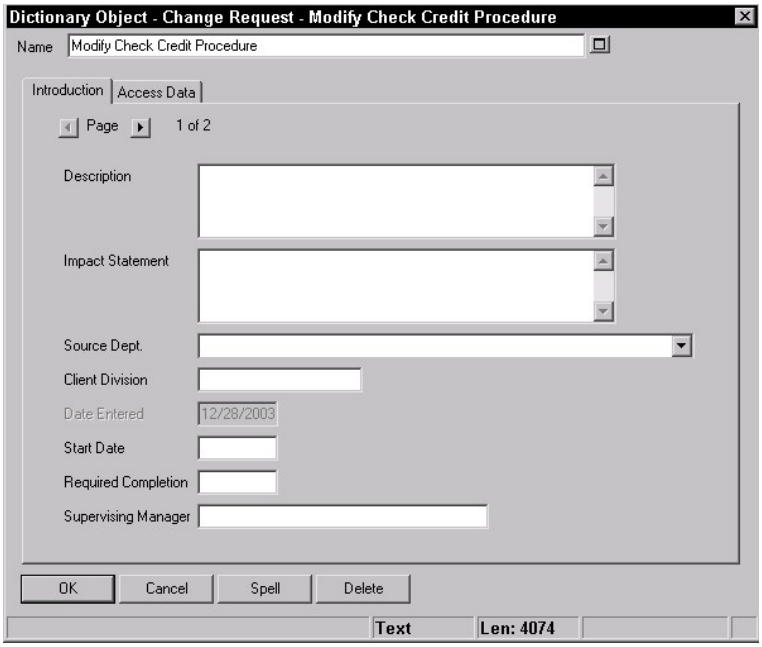
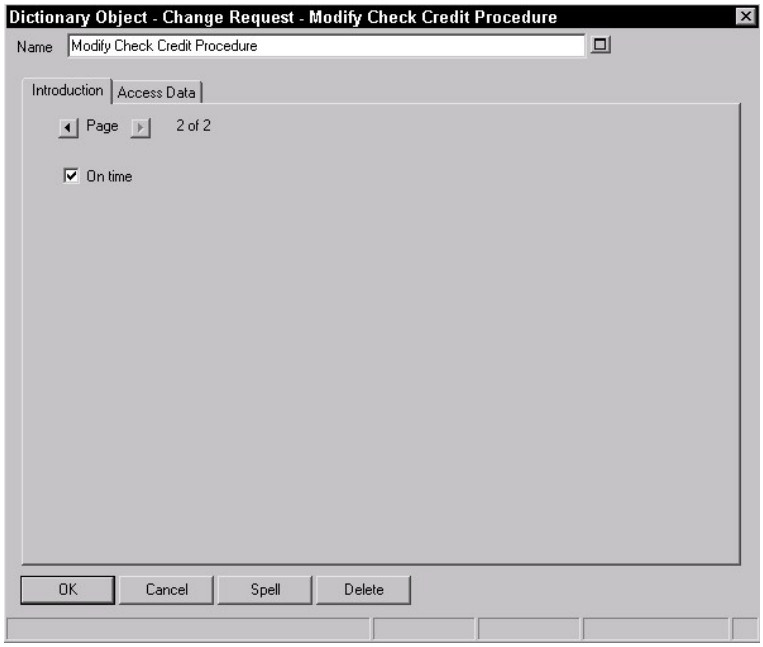


Figure 2- 4. Boîte de dialogue de définition Demande de modification, telle que modifiée par les entrées du fichier USRPROPS.TXT



Les informations sont présentées sur deux pages **Introduction** en raison des deux nouvelles propriétés ajoutées. Elles sont ajoutées à la fin de la définition (et non à la fin de la page **Données d'accès** car cette dernière ne compte pas ; elle est codée en dur et ne fait pas partie du fichier SAPROPS.CFG).

Ne modifiez que ce qui doit l'être

Notez que nous n'avons pas ressaisi l'intégralité de l'instruction PROPERTY qui existe dans SAPROPS.CFG dans notre fichier USRPROPS.TXT. Nous devons simplement entrer les instructions spécifiques qui doivent être modifiées, en regard des nouvelles instructions que nous ajoutons. Même pour les instructions que nous modifions et ressaisissons, seule la partie modifiée de l'instruction doit être ajoutée. Dans notre exemple, l'instruction du fichier SAPROPS.CFG que nous avons ressaisie et modifiée était :

```
PROPERTY "Nom de l'auteur" { EDIT TEXT LENGTH 25 }
```

Dans notre fichier USRPROPS.TXT, nous ne voulions modifier que le libellé de cette propriété ; nous avons donc simplement entré :

```
PROPERTY "Nom de l'auteur"{ LABEL "Division client" }
```

La longueur de la propriété et le fait qu'il s'agit de texte (et non d'une propriété numérique ou booléenne) restent inchangés ; seul le libellé à gauche du contrôle dans la boîte de dialogue a été modifié.

Une modification supplémentaire et un avertissement

Essayons une autre modification : nous ajoutons le texte en gras sous le code de notre fichier USRPROPS.TXT :

```
DEFINITION "Demande de modification"
{
LAYOUT { COLS 2 TAB ALIGN LABEL }
PROPERTY "Déclaration d'impact"  { EDIT text LENGTH
100 }
PROPERTY "Nom de l'auteur" { LABEL "Division client" }
PROPERTY "Gestionnaire de supervision" { EDIT text
LENGTH 45 }
PROPERTY "A l'heure" { Edit Boolean Length 1 DEFAULT "T"
}
}
```

L'explication de cette modification est données ci-dessous :

Entrée du fichier USRPROPS.TXT	Effet
PROPERTY "Déclaration d'impact" { EDIT TEXT LENGTH **100** }	Tentatives de modification d'une propriété existante, pour réduire la longueur de la propriété *Déclaration d'impact* de 1000 à 100 caractères.

Nous réimportons ce fichier USRPROPS.TXT dans notre encyclopédie, rouvrons cette dernière et recevons un message d'avertissement de Rational System Architect :

```
Avertissement : Dans l'annexe du
fichier de propriétés de l'utilisateur
entre la ligne numéro 70 et la ligne
numéro 72. Tentative non conforme de
raccourcir la longueur d'une propriété.
Longueur d'origine conservée.
```

Rational System Architect ne permet pas de réduire la longueur d'une zone ; vous ne pouvez que l'augmenter. Il se peut que des utilisateurs aient déjà entré des informations dans une zone de texte, qui seront perdues si vous réduisez la longueur de cette zone, et donc la quantité d'informations pouvant être conservées par l'encyclopédie pour cette propriété, ultérieurement.

Rational System Architect génère l'avertissement, ignore le code erroné et ouvre l'encyclopédie. Comme mentionné précédemment, s'il s'était agit d'un message d'erreur, l'encyclopédie ne se serait pas ouverte avant que vous n'ayez corrigé le fichier USRPROPS.TXT.

Si nous tentions par contre d'augmenter la longueur de la zone Déclaration d'impact, Rational System Architect accepterait volontier la modification.

PROPERTY "Déclaration d'impact" { EDIT text LENGTH **1200** }

Définition d'une liste de valeurs

Vous pouvez spécifier une liste d'éléments fournie à l'utilisateur sous forme de liste déroulante ou de liste de cases à cocher dans des boîtes de dialogue. Les valeurs de la liste doivent être spécifiées dans une définition de liste. La définition de liste est ensuite référencée dans le diagramme, le symbole ou la définition dans lequel elle est utilisée. Les listes doivent être placées dans le fichier USRPROPS.TXT avant toute entrée Diagramme, Symbole ou Définition qui y fait référence.

La gestion du fichier USRPROPS.TXT est plus facile si toutes les définitions de liste se trouvent au début du fichier, après les commandes RENAME.

Syntaxe de la définition LIST

Une définition de liste commence par le mot clé **LIST** suivi d'une chaîne (l'argument) qui correspond au nom de la liste. Les noms contenant des espaces doivent être placés entre guillemets. La définition **LIST** est délimitée par des accolades ouvrantes et fermantes **{ }** ou, par la structure **BEGIN...END**. A l'intérieur des accolades, vous spécifiez les valeurs de la liste, chacune étant appelée par le mot clé de la commande, **VALUE**. Si une valeur contient un ou plusieurs espaces, elle doit être placée entre guillemets.

```
LIST  nom_liste
{
VALUE nom_valeur_1
VALUE nom_valeur_2

…
}
```

Exemple :

```
List "Stéréotypes de méthode"
{
VALUE Get
VALUE Let
VALUE Set
VALUE "Stéréotype avec espaces imbriqués"
}
```

```
DEFINITION "Méthode"  {..PROPERTY "Stéréotype"{
EDIT Text LIST "Stéréotypes de méthode" Default ""
LENGTH 30  }  …}
```

Les indentations et les nouvelles lignes ne sont utilisées que pour améliorer la lisibilité et n'ont aucune signification dans le processeur USRPROPS.TXT autre que celle d'agir comme séparateurs (espaces blancs) entre les chaînes. L'exemple ci-dessus peut être écrit comme suit :

```
LIST "Stéréotypes de méthode"  {  VALUE get  VALUE
let  VALUE set  VALUE "Stéréotype avec espaces
imbriqués"  }
```

Ce format est tout à fait acceptable pour Rational System Architect, mais il rend plus difficile la maintenance du fichier USRPROPS.TXT et doit donc être évité.

Cases à cocher versus Liste déroulante

Rational System Architect affiche automatiquement une liste sous forme de liste de cases à cocher si le nombre de valeurs de l'instruction LIST est inférieur ou égal à quatre. Si le nombre de valeur est supérieur ou égal à cinq, la liste est automatiquement affichée sous forme de zone de liste déroulante. Les utilisateurs peuvent saisir leur propre valeur dans une zone de liste déroulante. Si vous souhaitez une zone de liste déroulante alors que votre liste contient quatre valeurs LIST ou moins, utilisez le mot-clé LISTONLYCOMBO.

Saisie de vos propres valeurs

Si la liste se présente sous forme de liste déroulante, l'utilisateur peut sélectionner l'une des valeurs de la liste ou saisir sa propre valeur (à moins que la commande LISTONLY ou LISTONLYCOMBO n'ait été utilisée ; voir la commande LISTONLY ou LISTONLYCOMBO, dans le Chapitre 3).

Modification des noms de type de diagramme, de symbole ou de définition existants

Chaque instruction DIAGRAM, SYMBOL, DEFINITION doit faire référence à un objet connu de Rational System Architect.

Toutefois, si l'un des noms fournis dans le fichier SAPROPS.CFG n'est pas approprié, vous pouvez le modifier en fonction des exigences de votre projet ou des normes de votre société. Les instructions RENAME doivent être insérées en haut du fichier USRPROPS.TXT, avant toutes les autres commandes et instructions. La syntaxe générale de la commande **RENAME** est la suivante :

```
RENAME nom_classe ancien_nom TO nouveau-nom
```

Les trois instructions suivantes permettent de renommer un diagramme, un symbole et une définition :

```
RENAME DIAGRAM "Flux de données Gane & Sarson"
TO "Flux de données Chris & Trish"

RENAME SYMBOL "Conversion des données" in
"Flux de données Ward & Mellor"
TO "Processus A"

RENAME DEFINITION "Processus"
TO "Processus A"
```

Figure 2-5. Le menu déroulant Type n'affiche pas *Flux de données Gane & Sarson*, mais *Flux de données Chris & Trish*

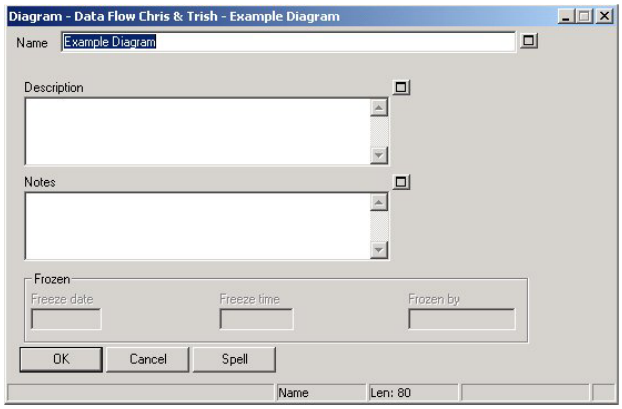


Figure 2- 6. La boîte de dialogue de modification des propriétés de diagramme est également affichée dans le titre *DFD Chris & Trish et* non *DFD Gane & Sarson.*

La commande **RENAME SYMBOL** peut être utilisée par les concepteurs utilisant des diagrammes de flux de données Ward & Mellor qui préfèrent le nom *Processus* au nom *Conversion* : Cliquez sur une *Conversion des contrôles* dans un diagramme de flux de données Ward & Mellor. Cliquez ensuite deux fois sur le symbole dans le diagramme pour afficher la boîte de dialogue **Diagramme <Type> <Nom>**. Le type du symbole est *Conversion des contrôles*.

Pour renommer le symbole, la commande suivante doit être entrée dans le fichier USRPROPS.TXT :

```
RENAME nom_classe ancien_nom IN
nom_diagramme TO nouveau-nom
```

Par exemple,

```
RENAME SYMBOL "Conversion des contrôles" IN
"Définition de flux de données Ward & Mellor" TO
"Processeu" ACCELERATOR "r"
```

Figure 2-7. Boîte de dialogue Propriétés des symboles avant RENAME

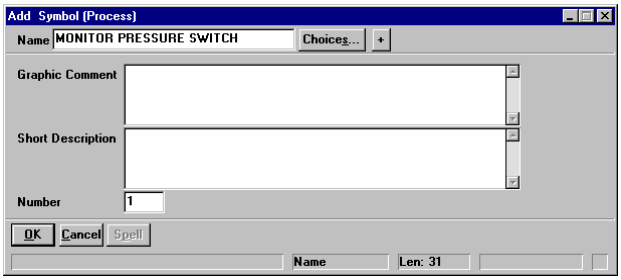
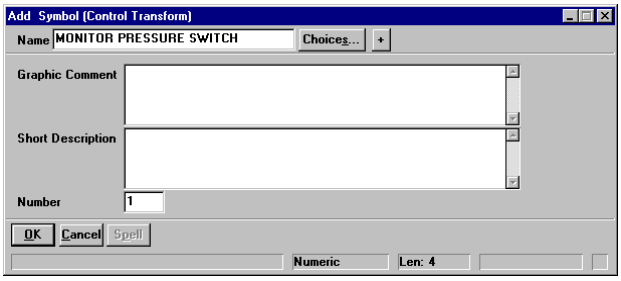


Figure 2-8. Boîte de dialogue Propriétés des symboles après RENAME



Cliquez sur le menu **Editer** et sélectionnez **Editer <Type de symbole>** de nouveau, alors que *Conversion des contrôles* est sélectionné. Notez le titre de la boîte de dialogue de définition **Modifier** : la définition est *Processus et* non *Conversion des contrôles*. La définition du symbole *Conversion des contrôles* est mappée à la définition *Processus*. Supposons que vous utilisez le diagramme de flux de données Ward & Mellor et non Gane & Sarson et que vous préférez que le nom de la définition corresponde au nom du symbole.

La syntaxe de la commande **RENAME** pour une définition est la suivante :

RENAME nom_classe ancien_nom TO nouveau-nom

RENAME DEFINITION "Processus" TO "Conversion des contrôles"

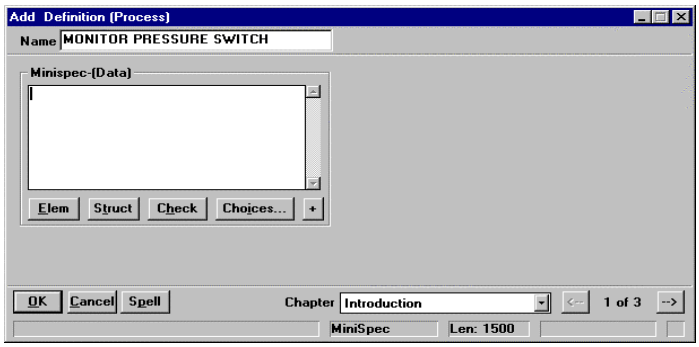


Figure 2-9. Boîte de dialogue Définition de symbole avant RENAME

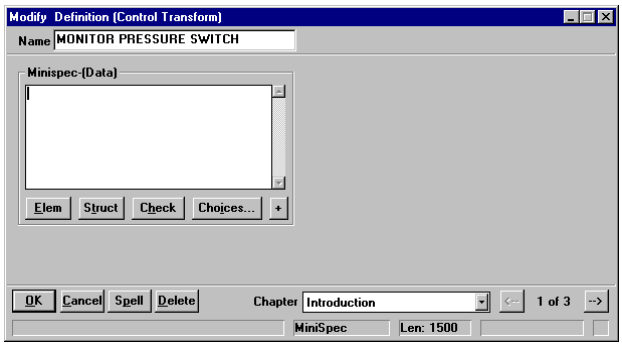


Figure 2-10. Boîte de dialogue Définition de symbole après RENAME

D'un autre côté, si vous utilisez à la fois le DFD Gane & Sarson (dont les symboles *Processus* sont mappés à la définition *Processus* et le DFD Ward & Mellor (dont les symboles *Conversion des contrôles* sont mappés à la définition *Processus*), vous avez la possibilité de ne renommer que les définitions des *conversions de contrôle* et non les définitions de tous les processus. Les entrées suivantes du fichier USRPROPS.TXT effectuent ce changement de nom :

RENAME DEFINITION "Utilisateur 2"[1] to "Conversion des contrôles"

SYMBOL "Conversion des contrôles" in <nom du diagramme> { DEFINED BY "Conversion des contrôles" }

Si la définition "Conversion des contrôles" n'inclut pas un ensemble de propriétés, elle ne contient que la propriété *Description*. Dans notre exemple, le bloc de définition ci-après, identique à celui de *Processus*, a été ajouté au fichier USRPROPS.TXT. Vous pouvez bien sûr utiliser les propriétés de votre choix ; vous n'avez pas besoin de copier celles d'une définition existante.

[1] Vous avez le choix entre 150 définitions "Utilisateur n", à commencer par Utilisateur 1.

```
DEFINITION "Conversion des contrôles"
{
PROPERTY "Description"
  { EDIT Minispec LENGTH 750 }
PROPERTY "Complexité"
  { EDIT numeric LENGTH 10 }
PROPERTY "Allocation de mémoire (Ko)"
   { EDIT numeric LENGTH 7 }
PROPERTY "Priorité"
 { EDIT numeric LENGTH 3 MINIMUM 0 MAXIMUM 999 }
PROPERTY "Classe de processus"
 { EDIT text LISTONLY LIST "Process Class" LENGTH 20 }
PROPERTY "Allocation du temps de traitement"
  { EDIT numeric LENGTH 3 MINIMUM 0 MAXIMUM 100 }
PROPERTY "Finalité"
 { EDIT text LENGTH 4095 }
PROPERTY "Débit de transactions"
  { EDIT numeric LENGTH 10 MINIMUM 1 MAXIMUM 10 }
}
```

RENAME et génération de rapports

La commande **RENAME** affecte également la manière dont vous écrivez les rapports. Vous devez utiliser le nouveau nom partout où l'ancien était utilisé. Dans le système de génération de rapports de l'interface graphique, vous devrez resélectionner les noms de propriété du diagramme, du symbole ou de la définition une fois qu'ils ont été modifiés dans le fichier SAPROPS.

Avant :

```
REPORT "Liste de processus"
{
TABULAR 1 {
SELECT Name, "Date de mise à jour", Description
WHERE Class = Definition
WHERE Type = "Processus"
ORDERBY Nom
}
}
```

Si vous consultez le rapport à l'aide de l'éditeur de texte (**Rapports**, commande **EDIT**), vous apercevez :

Après :

```
REPORT "Liste de conversions"
{
TABULAR 1 {
SELECT Name, "Date de mise à jour", Description
WHERE Class = Definition
WHERE Type = "Conversion des contrôles"
ORDERBY Name
}
}
```

Création de types de diagramme, de symbole ou de définition

Vous pouvez créer des types de diagramme, symbole ou définition dans une encyclopédie Rational System Architect. Utilisez pour cela la commande RENAME afin de renommer les types de diagramme, symbole ou définition pré-existants fournis à ces fins. Là encore, les commandes RENAME doivent être placées au début du fichier USRPROPS.TXT, immédiatement sous les instructions REM (Rappel ou Commentaire) d'ouverture.

Création de diagrammes

Vous pouvez créer jusqu'à 50 types de diagramme dans une encyclopédie Rational System Architect. Pour ajouter un nouveau type de diagramme dans le fichier USRPROPS.TXT, renommez l'un des 50 types de diagramme génériques disponibles : Utilisateur 1 à Utilisateur 50. La syntaxe est la suivante :

```
RENAME DIAGRAM "Utilisateur 1" TO
Mon_Diagramme
```

Notez que si vous souhaitez placer des espaces dans un type de diagramme, symbole ou définition que vous créez, vous devez placer le nom entre guillemets. Par exemple :

```
RENAME DIAGRAM "Utilisateur 1" TO "Mon
diagramme"
```

une fois que vous avez créé le type de diagramme, vous devez spécifier les types de symbole qui peuvent y être représentés. Vous pouvez créer des types de symbole ou affecter des symboles qui existent déjà sur d'autres diagrammes au nouveau type de diagramme. Cette opération est abordée dans la section suivante : Affectation d'un type de symbole à un type de diagramme.

Par défaut, les diagrammes utilisateur sont des réseaux (de symboles), mais vous pouvez également indiquer qu'un diagramme utilisateur est de type **Hiérarchique**. Un diagramme hiérarchique dans Rational System Architect doit respecter des règles de représentation spéciales, ce qui permet de connecter des symboles dans une hiérarchie et de représenter automatiquement les symboles de ligne. D'autres fonctionnalités hiérarchiques associées (telles que la numérotation hiérarchique) sont prises en charge. Pour spécifier qu'un diagramme est de type Hiérarchique, utilisez le mot clé HIERARCHICAL. Par exemple :

```
DIAGRAM "Zoo" {HIERARCHICAL}
```

Création de symboles

Vous pouvez créer jusqu'à 150 nouveaux types de symbole dans une encyclopédie Rational System Architect. Pour ajouter un nouveau type de symbole, renommez l'un des 150 types de symbole génériques fournis -- Utilisateur 1 à Utilisateur 150. La syntaxe est la suivante :

```
RENAME SYMBOL "Utilisateur 3" to "ce que vous
voulez"
```

Specification des symboles de ligne

Un symbole de ligne est une ligne qui peut être représentée entre deux symboles, telle qu'une ligne de relation, une ligne d'héritage, une association, une ligne de flux, etc. Vous pouvez créer un type de symbole de ligne dans une encyclopédie. Vous devez spécifier qu'elle apparaît et se comporte comme un type de symbole de ligne existant sur un autre diagramme. Vous utilisez la même commande RENAME SYMBOL que pour un symbole standard (‘noeud’), mais ensuite, dans le fichier USRPROPS.TXT, vous devez également spécifier la manière dont le symbole de ligne est représenté, à l'aide de la commande DEPICT LIKE.

```
RENAME SYMBOL "Utilisateur 4" to "Mon symbole de
ligne"

SYMBOL "Mon symbole de ligne"
{ DEPICT LIKE "Dépendance" IN "Classe UML"
ASSIGN To "Réseau sans fil" }
```

Création de définitions

Vous pouvez créer jusqu'à 150 nouveaux types de définition dans une encyclopédie Rational System Architect. Pour ajouter un nouveau type de définition, renommez l'un des 150 types de définition génériques fournis -- Utilisateur 1 à Utilisateur 150. La syntaxe est la suivante :

```
RENAME DEFINITION "Utilisateur 3" to "ce que vous voulez"
```

Un symbole représente généralement un type de définition. Pour plus d'informations, voir la section qui suit : Affectation d'un type de définition à un type de symbole.

Affectation d'un type de symbole à un type de diagramme

Vous pouvez affecter de nouveaux types de symbole ou des types de symbole existants (symboles qui existent déjà dans un autre diagramme) à des types de diagramme nouveaux ou existants. Les types de symbole peuvent être ajoutés à des types de diagramme à l'aide de la syntaxe suivante :

ASSIGN <nom_type_symbole> [IN <nom_type_diagramme1>] **TO** <nom_type_diagramme2>

Des types de symbole peuvent également être ajoutés à des types de diagramme dans la spécification SYMBOL à l'aide de la combinaison de mots clés **ASSIGN .. TO**, comme suit :

SYMBOL <nom_type_symbole> [IN <nom_type_diagramme1>] **{ASSIGN TO** <nom_type_diagramme1>**}**

Exemple

Par exemple, le fichier USRPROPS.TXT ci-après crée un type de diagramme appelé diagramme Réseau sans fil, qui fournit trois nouveaux types de symbole à représenter (un satellite, un ordinateur et un serveur) et un type de symbole existant à représenter (un symbole d'état d'un diagramme "Transition d'état Ward & Mellor" (par comparaison à un symbole d'état d'un diagramme Etat UML ou d'un diagramme Etat IDEF3, etc.) :

```
RENAME DIAGRAM "Utilisateur 1" To "Réseau sans fil"
RENAME SYMBOL "Utilisateur 1" TO "Satellite"
RENAME SYMBOL "Utilisateur 2" TO "Ordinateur"
RENAME SYMBOL "Utilisateur 3" TO "Serveur"
ASSIGN "Etat" IN "Transition d'état Ward & Mellor" TO
"Réseau sans fil"
SYMBOL "Satellite" {ASSIGN TO "Réseau sans fil"}
SYMBOL "Ordinateur" {ASSIGN TO "Réseau sans fil"}
SYMBOL "Serveur" {ASSIGN TO "Réseau sans fil"}
```

Remarque : Voir également la section *Limitations de l'affectation d'un type de symbole à un type de diagramme*.

Affectation d'un type de symbole de ligne à un type de diagramme

Là encore, comme mentionné précédemment dans cette section, un symbole de ligne est une ligne qui peut être représentée entre deux symboles, telle qu'une ligne de relation, une ligne d'héritage, une association, une ligne de flux, etc. Vous pouvez créer un type de symbole de ligne dans une encyclopédie. Vous devez spécifier qu'elle apparaît et se comporte comme un type de symbole de ligne existant sur un autre diagramme. Vous utilisez la même commande RENAME SYMBOL que pour un symbole standard (‘noeud’), mais ensuite, dans le fichier USRPROPS.TXT, vous devez également spécifier la manière dont le symbole de ligne est représenté, à l'aide de la commande DEPICT LIKE.

Des symboles définis par l'utilisateur (Utilisateur 1 à Utilisateur 150) sont fournis pour les symboles et les symboles de ligne standard (‘noeud’) ; veillez donc à ne pas utiliser le même numéro d'utilisateur pour deux symboles différents.

Exemple

Dans l'exemple ci-après, nous ajoutons un nouveau symbole de ligne à notre fichier USRPROPS.TXT, en gras :

```
RENAME DIAGRAM "Utilisateur 1" To "Réseau sans fil"

RENAME SYMBOL "Utilisateur 1" TO "Satellite"
RENAME SYMBOL "Utilisateur 2" TO "Ordinateur"
RENAME SYMBOL "Utilisateur 3" TO "Serveur"
RENAME SYMBOL "Utilisateur 4” To "Associé à"

ASSIGN "Etat" IN "Transition d'état Ward & Mellor" TO
"Réseau sans fil"
SYMBOL "Satellite" {ASSIGN TO "Réseau sans fil"}
SYMBOL "Ordinateur" {ASSIGN TO "Réseau sans fil"}
SYMBOL "Serveur" {ASSIGN TO "Réseau sans fil"}
```

```
SYMBOL "Associé à"
{ DEPICT LIKE "Dépendance" IN " Classe"
 ASSIGN To "Réseau sans fil }
```

Remarque : Voir également la section *Limitations de l'affectation d'un type de symbole à un type de diagramme*.

Limitations de l'affectation d'un type de symbole à un type de diagramme

Les limitations suivantes s'appliquent à l'affectation de types de symbole à des types de diagramme :

Aucune affectation ne peut être appliquée aux types de diagramme suivants :

- Données physiques DB2
- Relation d'entité
- Modèle de données logiques
- Vue logique
- Modèle de données physiques

Aucun des symboles suivants ne peut être affecté en raison d'un code spécial dans Rational System Architect :

- "Entité associative" dans le diagramme "Relation d'entité"
- "Entité" dans le diagramme "Relation d'entité"
- "Relation identifiante" dans le diagramme "Relation d'entité"
- "Relation incohérente" dans le diagramme "Relation d'entité"
- "Relation non identifiante" dans le diagramme "Relation d'entité"
- "Relation non spécifique" dans le diagramme "Relation d'entité"
- "Super/Sous-relation" dans le diagramme "Relation d'entité"
- "Entité faible" dans le diagramme "Relation d'entité"
- "Association" dans le diagramme "Modèle d'objet OMT"
- "Classe" dans le diagramme "Modèle d'objet OMT"
- "Contrainte d'identification" dans le diagramme "Modèle de données physiques"
- "Contrainte non identifiante" dans le diagramme "Modèle de données physiques"

- "Table" dans le diagramme "Modèle de données physiques"
- "Classe" dans le diagramme "Classe UML"
- "Interface" dans le diagramme "Classe UML"
- "Acteur" dans le diagramme "Cas d'utilisation UML"
- "Limite" dans le diagramme "Cas d'utilisation UML"
- "Travailleur social" dans le diagramme "Cas d'utilisation UML"
- "Contrôle" dans le diagramme "Cas d'utilisation UML"
- "Entité" dans le diagramme "Cas d'utilisation UML"
- "Agent" dans le diagramme "Cas d'utilisation UML"
- De plus, aucun des symboles suivants ne peut être affecté à un autre diagramme car leur définition est saisie par modèle :
- "Chemin d'accès" dans le diagramme "Relation d'entité"
- "Relation" dans le diagramme "Relation d'entité"
- "Losange des relations" dans le diagramme "Relation d'entité"
- "Individu" dans le diagramme "Modèle Conceptuel des Données"
- "Relation Ligne" dans le diagramme "Modèle Conceptuel des Données"
- "Point d'entrée saisi" dans le diagramme "Structure de données SSADM"
- "Point d'entrée non saisi" dans le diagramme "Structure de données SSADM"
- "Relation" dans le diagramme "Structure de données SSADM"
- En outre, les symboles BPMN suivants ne peuvent pas être affectés à un autre type de diagramme :
- "Pool" dans le diagramme Processus métier
- "Voie" dans le diagramme Processus métier

Remarque : Certains symboles possèdent un code spécial et sont saisis par modèle ; ils ne sont affichés que dans la première liste des symboles.

De nombreux symboles peut normalement se trouver dans plusieurs types de diagramme. Un seul type de diagramme est affiché pour tout symbole des listes ci-avant.

Affectation d'un type de définition à un type de symbole

Signification 1 : Si vous ajoutez de nouveaux symboles à une encyclopédie dans USRPROPS.TXT, vous devez spécifier le type de définition auquel ils sont associés, à l'aide de ce mot-clé. Si un nouveau symbole spécifié dans USRPROPS.TXT ne contient pas cette clause, Rational System Architect génère un avertissement d'analyse lors de l'ouverture de l'encyclopédie et utilise par défaut la définition null du symbole, qui ne contient que la propriété Description.

```
SYMBOL "Mon symbole"
{
DEFINED BY " Ma définition"
ASSIGN TO "Mon diagramme"
}
```

Exemple

Dans l'exemple ci-après, le type de symbole "Satellite" est spécifié comme étant défini par le type de définition "Satellite" (le fait qu'ils partagent le même nom n'est pas suffisant).

```
Rename Diagram "Utilisateur 1" TO "Réseau sans fil"
Rename Symbol "Utilisateur 1" TO "Satellite"
Rename Definition "Utilisateur 1" TO "Satellite"

SYMBOL "Satellite"
{ DEFINED BY "Satellite" ASSIGN To "Réseau sans fil" }
```

Représentation d'un symbole à l'aide d'un bitmap ou d'un métafichier

Vous pouvez représenter un symbole à l'aide d'un bitmap (.bmp) ou d'un métafichier Windows (.wmf) que vous fournissez. Vous pouvez indiquer comment un symbole est représenté dans l'espace de travail du diagramme, ainsi que dans la boîte à outils et le menu Tracer, en ajoutant une clause depictions à la déclaration du symbole, comme suit :

SYMBOL <nom-type-symbole>

{ ...
DEPICTIONS { DIAGRAM <fichier-représentation> }
DEPICTIONS { MENU <fichier-représentation> }
...}

La commande DIAGRAM spécifies le fichier de représentation à ***représenter*** sur l'espace de travail du diagramme. Vous devez utiliser un ***métafichier Windows (.WMF)*** pour la commande DIAGRAM car il s'agit d'une image vectorielle qui se mettra correctement à l'échelle si vous déplacez ces poignées pour en augmenter ou en réduire la taille.Vous pouvez également utiliser le format .BMP pour la commande DIAGRAM, mais la mise à l'échelle n'est alors pas correcte.

Les fichiers WMF sont des fichiers vectoriels, ce qui signifie qu'ils stockent des formules mathématiques sur la manière dont une image doit être affichée à l'écran. L'un des principaux avantages de ce format vient de ses possibilités d'évolution sans perte de qualité d'image. Les fichiers WMF ne deviennent pas désordonnés ou irréguliers lors des zooms avants ou arrières.

La commande **MENU** indique que le fichier de représentation doit apparaître dans les ***barres d'outils***, les ***menus*** et autres

zones. Il s'agit de l'image sur laquelle vous cliquez pour sélectionner un symbole à représenter. Pour la barre d'outils, l'utilisation d'images **bitmap** est recommandée car la mise à l'échelle n'est pas utile. Généralement, il est préférable de créer un bitmap de ***16x16 pixels*** pour chaque symbole à représenter dans la barre d'outils.

Les BMP sont des fichiers tramés, ce qui signifie qu'ils stockent des informations sur chaque pixel d'une image. Les fichiers bitmap peuvent afficher des images riches, de qualité photo, mais ils deviennent désordonnés lors d'un zoom avant, et irréguliers lors d'un zoom arrière.

<fichier-représentation> correspond au nom et au chemin d'accès complet d'un bitmap ou d'un métafichier. Vous pouvez spécifier un répertoire à l'extérieur du chemin d'accès de votre encyclopédie, mais il est conseillé d'ajouter les images bitmap et les métafichiers directement dans la table Files de la base de données d'une encyclopédie.

Pour ajouter vos propres fichiers de représentation à une encyclopédie, procédez comme suit :

1. **Apportez les modifications nécessaires au fichier USRPROPS.TXT.** Exemple de code :

   ```
   RENAME DIAGRAM "Utilisateur 1" TO "Communications sans fil"
   RENAME SYMBOL "Utilisateur 1" TO "Satellite"
   SYMBOL "Satellite"
   {ASSIGN To "Réseau sans fil"
   DEPICTIONS { DIAGRAM satellite.wmf }
   DEPICTIONS { MENU satellite_toolbar.bmp }
   }
   ```

2. **Importez vos fichiers .BMP et .WMF dans la table FILES de l'encyclopédie**. Vous pouvez utiliser le gestionnaire de fichiers de l'encyclopédie de Rational System Architect (Outils, Gestionnaire de fichiers de l'encyclopédie), SAEM (Démarrer, Programmes, IBM Rational, IBM Rational Lifecycle Solutions Tools, IBM Rational System Architect 11.3.1, SAEM ; voir son aide pour savoir comment l'utiliser), ou Enterprise Manager de Microsoft pour importer vos fichiers graphiques

définis par l'utilisateur dans la table FILES de la base de données de l'encyclopédie. Le gestionnaire de fichiers de l'encyclopédie ne peut importer qu'un seul fichier à la fois. Si vous avez plusieurs fichiers graphiques, il est recommandé d'utiliser SAEM pour les importer dans la table FILES.

Les noms des fichiers que vous importez doivent être cohérents avec le code de votre fichier Usrprops.txt. Dans l'exemple ci-dessus, nous avons utilisé un chemin d'accès relatif, en ne spécifiant aucun chemin ; nous avons simplement spécifié satellite.wmf et satellite.bmp. Cela signifie que les fichiers de représentations doivent être importés directement dans la table Files de la base de données.

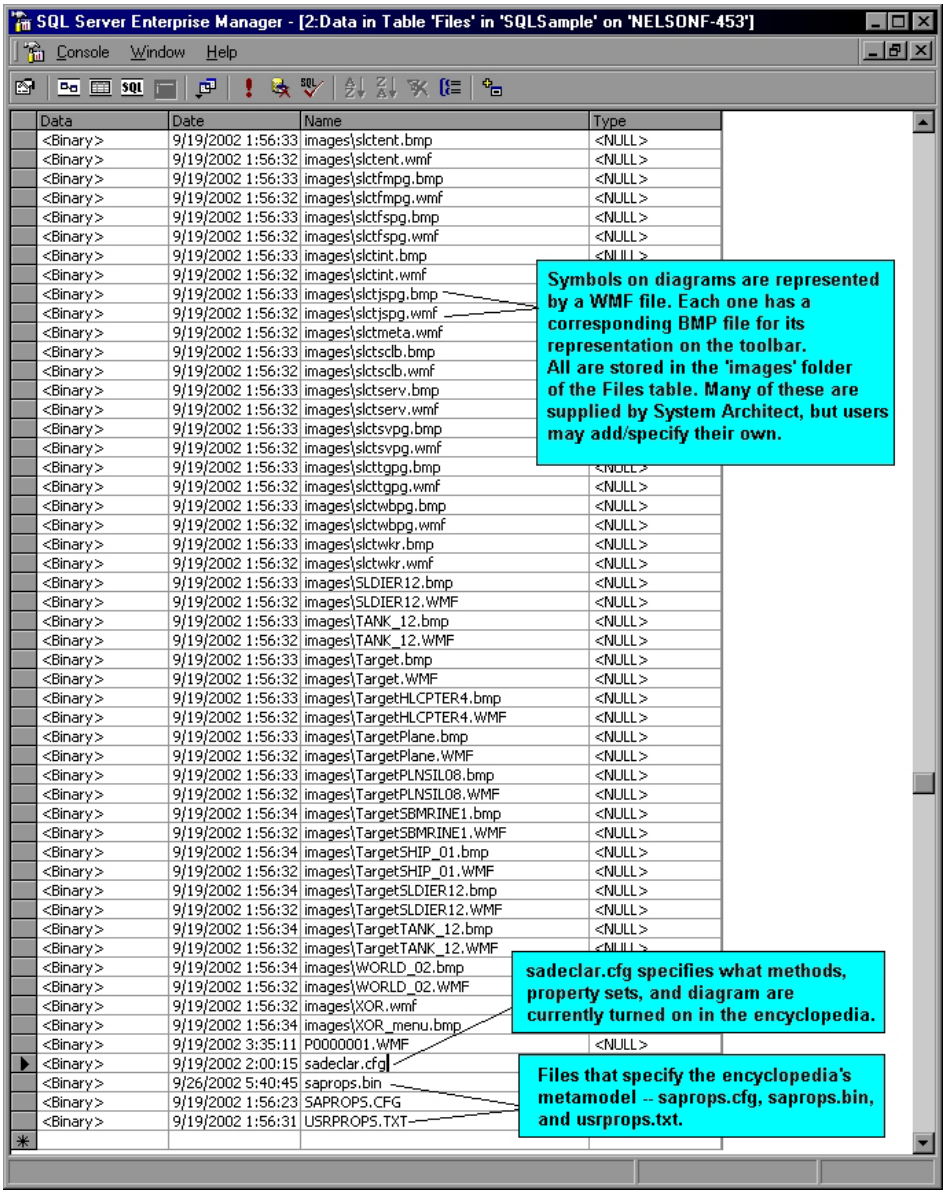
Recommandation : Il est recommandé de respecter une convention établie dans Rational System Architect et d'ajouter ‘images/’ au nom de vos fichiers de représentation pour préciser que chaque fichier de représentation se trouve dans un sous-répertoire ‘images’ de la table FILES. Si vous utilisez SAEM pour importer simultanément plusieurs fichiers, assurez-vous que ces derniers se trouvent dans un répertoire ‘images’ de votre ordinateur. SAEM ajoute automatiquement ‘images/’ aux noms de tous les fichiers importés à partir d'un répertoire intitulé images, en préfixe du nom de fichier de chaque graphique. De même, vous devez spécifier ‘images\’ avant le nom de chaque fichier de repésentation dans votre fichier USRPROPS.TXT. L'exemple ci-dessus se présente alors ainsi :

```
RENAME DIAGRAM "Utilisateur 1" TO "Communications sans fil"
RENAME SYMBOL "Utilisateur 1" TO "Satellite"
SYMBOL "Satellite"
{ASSIGN To "Réseau sans fil"
DEPICTIONS { DIAGRAM images\satellite.wmf }
DEPICTIONS { MENU images\satellite_toolbar.bmp }
}
```

Cette stratégie offre deux avantages. Tout d'abord, elle offre l'indépendance des noms et une sorte de stratégie de regroupement logique aux images spécifiées par l'utilisateur. Ensuite, elle est cohérente avec la manière dont les images sont traitées lorsque des encyclopédies sont créées : Rational System Architect place tous les graphiques du répertoire ..\System Architect\images dans la table FILES de la nouvelle encyclopédie et leur affecte un nom commençant par ‘images\’.

Pour obtenir un aperçu de la table Files de la base de données d'une encyclopédie et comprendre comment le préfixe ‘images\’ des fichiers de représentation offre un regroupement logique des images, reportez-vous à l'illustration ci-après.

Figure 2-11. Table 'Files' de la base de données de l'encyclopédie.



3. **Rouvrez l'encyclopédie pour que les modifications soient appliquées**.

Spécification des fichiers de représentation des nouvelles encyclopédies

Si vous créez une encyclopédie, plusieurs options s'offrent à vous : vous pouvez la créer d'abord, puis y importer un ou plusieurs fichiers graphiques fournis par l'utilisateur via SAEM, le gestionnaire d'encyclopédie ou l'outil Enterprise Manager de SQL Server, comme décrit ci-dessus ou vous pouvez placer vos images fournies par l'utilisateur dans le répertoire ***images*** principal de Rational System Architect (sous le répertoire principal du logiciel, <C>:\Program Files\IBM\Rational\11.3.1\System Architect Suite\System Architect\images) avant de créer l'encyclopédie. Rational System Architect place tous les graphiques de son répertoires d'images principal dans la table Files de toutes les encyclopédies créées.

Si vous souhaitez que les mêmes graphiques spécifiés par l'utilisateur soient insérés dans toutes les encyclopédies créées par vous ou les autres membres de l'équipe, effectuez les opérations suivantes :

1. **Copiez et collez vos fichiers .BMP et .WMF dans le sous-répertoire 'Images' de Rational System Architect**. Avant de créer des encyclopédies, placez vos fichiers .BMP et .WMF dans le répertoire Images du répertoire principal des programmes de Rational System Architect. Tous les utilisateurs de l'équipe qui créeront des encyclopédies par la suite doivent procéder ainsi. Ces fichiers seront automatiquement placés dans la table FILES de l'encyclopédie créée ultérieurement. Rational System Architect ajoutant 'images\' à chaque nom de fichier, une figure appelée Fred.bmp sera créée dans la table FILES de la nouvelle encyclopédie sous le nom images\Fred.bmp. Il s'agit d'un raccourci pour créer l'encyclopédie, puis importer par la suite les fichiers

graphiques fournis par l'utilisateur dans l'encyclopédie.

2. **Apportez les modifications nécessaires au fichier USRPROPS.TXT.** Utilisez la commande DEPICTIONS (et éventuellement la commande RETAIN STYLE). Des informations sur la manière d'apporter les modifications de code nécessaires sont fournies dans l'aide de Rational System Architect. Exemple de code :

```
Rename Symbol "Utilisateur 3" To "Radar
SYMBOL "Radar"
{ASSIGN To "Réseau sans fil"
DEPICTIONS { DIAGRAM RETAIN STYLE "C:\Program
Files\IBM\pictures\radar.bmp" }
DEPICTIONS { MENU "C:\Program
Files\IBM\pictures\radartoolbar.bmp" }}
```

3. **Rouvrez l'encyclopédie pour que les modifications soient appliquées**

Présentation de symbole définie par l'utilisateur en fonction d'une valeur de propriété

Vous pouvez spécifier comment un symbole est représenté *en fonction de la valeur d'une propriété* de la définition du symbole. Dans UML, cette propriété est généralement un stéréotype. Toutefois, cette fonctionnalité s'applique à l'ensemble des types de symbole et non pas seulement aux symboles UML et à la propriété Stéréotype.

Pour activer cette fonction, la clause **DEPICTIONS** est utilisée directement dans une instruction **LIST** du fichier USRPROPS.TXT.

```
LIST "Nouveau type de liste"
{
VALUE "Elément de liste 1" DEPICTIONS {DIAGRAM
imageone.wmf MENU imageone_toolbar.bmp}
…
}
```

Exemple

Dans l'exemple ci-après, une nouvelle liste est spécifiée pour Stéréotypes de noeud. Ces stéréotypes sont appliqués à un symbole Noeud sur un diagramme de déploiements UML, pour qu'un utilisateur puisse représenter un symbole de noeud à l'aide de ses propres fichiers graphiques qu'il a importés dans la table FILES de la base de données de l'encyclopédie.

```
List "Stéréotypes de noeud"
{
Value "Pare-feu" DEPICTIONS {DIAGRAM images\firewall.wmf
MENU images\firewall.bmp}
Value "Téléphone_portable"  DEPICTIONS {DIAGRAM
images\cell_phone.wmf MENU images\cell_phone.bmp}
Value "Base_de_données"  DEPICTIONS {DIAGRAM
images\data.wmf MENU images\data.bmp}
Value "Concentrateur"   DEPICTIONS {DIAGRAM
images\hub.wmf MENU images\hub.bmp}
```

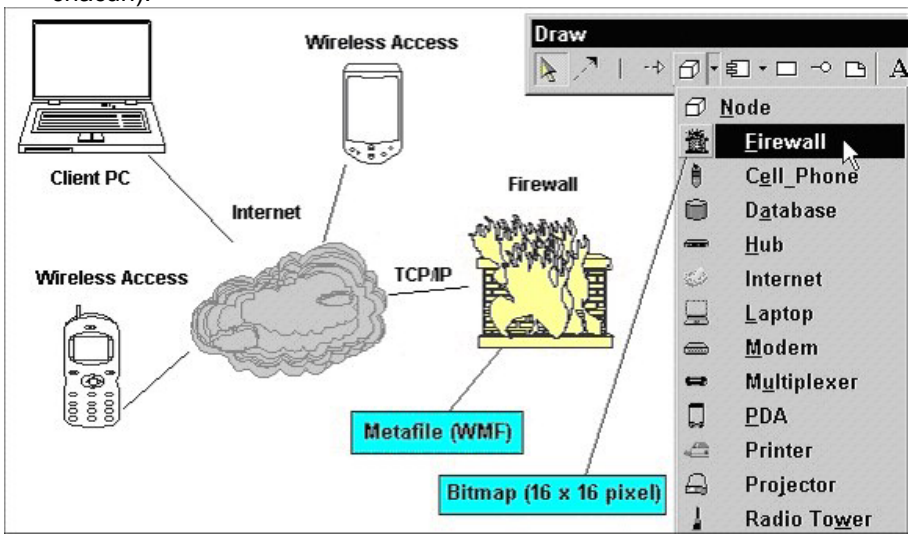
```
Value "Modem"  DEPICTIONS {DIAGRAM images\modem.wmf
MENU images\modem.bmp}
Value "Multiplexeur"  DEPICTIONS {DIAGRAM
images\multiplexer.wmf MENU images\multiplexer.bmp}
Value "Assistant_électronique"  DEPICTIONS {DIAGRAM
images\pda.wmf MENU images\pda.bmp}
Value "Imprimante"   DEPICTIONS {DIAGRAM
images\printer.wmf MENU images\printer.bmp}
Value "Projecteur" DEPICTIONS {DIAGRAM
images\projector.wmf MENU images\projector.bmp}
Value "Pylône radio" DEPICTIONS { DIAGRAM
images\radio_tower.wmf MENU images\radio_tower.bmp}
Value "Routeur"  DEPICTIONS { DIAGRAM images\router.wmf
MENU images\router.bmp}
Value "Satellite" DEPICTIONS { DIAGRAM images\satellite.wmf
MENU images\satellite.bmp}
Value "Parabolle"  DEPICTIONS { DIAGRAM images\dish.wmf
MENU images\dish.bmp}
Value "Scanner"  DEPICTIONS { DIAGRAM
images\scanner.wmf MENU images\scanner.bmp}
Value "Serveur"  DEPICTIONS { DIAGRAM images\server.wmf
MENU images\server.bmp}
Value "Commutateur"  DEPICTIONS { DIAGRAM
images\kvm_switch.wmf  MENU images\kvm_switch.bmp}
Value "Tablette_PC"  DEPICTIONS { DIAGRAM
images\tablet_pc.wmf MENU images\tablet_pc.bmp}
Value "Terminal" DEPICTIONS { DIAGRAM
images\terminal.wmf MENU images\terminal.bmp}
}

SYMBOL "Noeud" in "Déploiement"
{
PROPERTY "Stéréotype" { INVISIBLE EDIT Text ListOnly
List "Stéréotypes de noeud" DEFAULT "" LENGTH 32}
}

DEFINITION "Noeud"
{
PROPERTY "Stéréotype"
{ EDIT Text LIST "Stéréotypes de noeud" Default "" LENGTH
32  }
}
```

Dans l'exemple USRPROPS.TXT ci-dessus, notez que la liste des "Stéréotypes de noeud" est référencée dans le symbole et la définition d'un noeud. Dans le symbole, la

propriété est INVISIBLE. Le symbole conserve une référence au stéréotype spécifié pour sa définition sous-jacente.

Le code du fichier USRPROPS.TXT ci-dessus modifie la barre d'outils d'un diagramme Déploiement, en fournissant une liste déroulante des stéréotypes disponibles (et les images bitmap correspondantes de chacun).

Figure 2-12. Fichiers de représentation fournis par l'utilisateur.



L'utilisateur peut sélectionner un stéréotype et le représenter sur le diagramme par le fichier .WMF correspondant.

Après avoir représenté le symbole, vous pouvez cliquer sur chaque symbole à l'aide du bouton droit de la souris et choisir l'une des actions suivantes :

- L'afficher comme <Noeud>,
- Les décorer avec un stéréotype (dans lequel une miniature du métafichier est placée à droite du nom du symbole) ou
- L'afficher en fonction du stéréotype.

Une fois que vous l'avez représenté, vous pouvez spécifier les couleurs d'un métafichier comme pour tout autre symbole de Rational System Architect, à l'aide de la barre d'outils Style de symbole (ou en sélectionnant le symbole et en choisissant Format, Format du symbole, Style de symbole). Vous pouvez choisir la couleur des lignes, la couleur de remplissage, la couleur de police, etc.

Conserver le style

Vous pouvez spécifier que les métafichiers que vous fournissez conservent leur style graphique et leurs couleurs d'origine lorsqu'ils sont utilisés dans Rational System Architect. Pour cela, utilisez le mot clé RETAIN STYLE. Par exemple :

```
LIST "Stéréotypes de noeud"
{
VALUE "Pare-feu" DEPICTIONS {DIAGRAM RETAIN
STYLE images\firewall.wmf MENU images\firewall.bmp}
..
}
```

Lorsque le métafichier fourni par l'utilisateur, firewall.wmf, est dessiné dans le diagramme, ses couleurs sont identiques à celles qu'il avait hors de Rational System Architect. Il est impossible de les modifier avec les outils de couleur de Rational System Architect.

Propriétés affichables sur les symboles représentés

Rational System Architect permet de spécifier jusqu'à 37 propriétés à afficher sur un symbole à l'aide du mot-clé DISPLAY. Il en est de même des symboles représentés par des fichiers de représentation fournis par l'utilisateur.

Pour plus d'informations, reportez-vous à la section *Spécification de l' affichage des valeurs sur les symboles*, plus loin dans ce chapitre, ou au mot clé DISPLAY, dans le Chapitre 3.

Spécification des propriétés des diagrammes, symboles et définitions

Chaque encyclopédie **Rational System Architect** contient trois classes : *diagramme*, *symbole* et *définition*. Chacune peut être définie avec son propre ensemble de propriétés.

Le tableau ci-après inclut toutes les entrées obligatoires et facultatives qui se trouvent à l'extérieur d'une instruction de diagramme, symbole ou définition dans le fichier USRPROPS.TXT.

Tableau 2-3. Entrées obligatoires et facultatives des spécifications de diagramme, symbole ou définition

Entrée	Obligatoire Facultative	Remarque
DIAGRAM { } ou DIAGRAM BEGIN END ou SYMBOL { } ou SYMBOL BEGIN END ou DEFINTION { } ou DEFINITION BEGIN END	Obligatoire	Commence et termine la déclaration.
CHAPTER nom_chapitre	Facultative	Inclut les propriétés suivantes dans le chapitre existant ou ajoute un nouveau chapitre

GROUP nom_groupe
{
PROPERTY nom_prop
PROPERTY nom_prop
}
	Facultative	Place toutes les propriétés suivantes dans un groupe pour le contrôle de la présentation

Entrée	Obligatoire Facultative	Remarque
LAYOUT { critères_alignement critères_PACK_TAB COLS nombre_de_colonnes JUSTIFY }	Facultative	[Align Body | Align Label | Align Over] [Pack | Tab] COLS <nombre>

PROPERTY nom_propriété
{
}
	Obligatoire	Vous pouvez utiliser { ou BEGIN, et } ou END

Spécification des propriétés des types de diagramme

La propriété par défaut de tous les diagrammes est *Description*. La description est définie comme une zone de texte de 4074 caractères. Les propriétés de diagramme sont celles qu'un utilisateur souhaite définir pour tout un diagramme (par exemple, les couloirs d'activités (ou pools) sont-ils affichés verticalement ou horizontalement ?). Une boîte de dialogue **Propriétés du diagramme** type est illustrée ci-après.

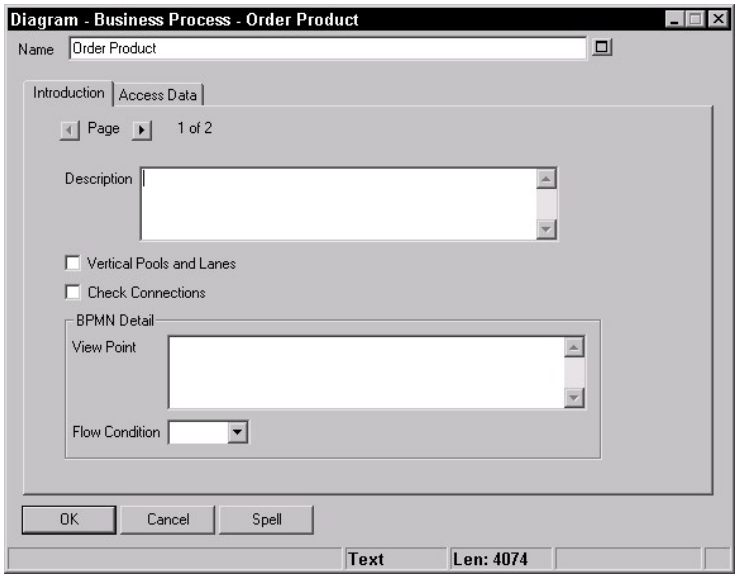


Figure 2- 13. Boîte de dialogue Propriétés du diagramme

Pour ajouter davantage de propriétés pour un diagramme, utilisez la syntaxe suivante :

```
DIAGRAM type_diagramme
{
PROPERTY-1 <nom_propriété>
  { <valeur_propriété> }
PROPERTY-2 <nom_propriété>
  { <valeur_propriété> }
PROPERTY-3 <nom_propriété>
  { <valeur_propriété> }
}
```

Par exemple, l'ajout des instructions suivantes au fichier USRPROPS.TXT modifie la boîte de dialogue **Propriétés du diagramme** du type de diagramme Processus métier, comme le montre l'illustration qui suit :

```
DIAGRAM "Processus métier"
{
CHAPTER "Mes propriétés"
PROPERTY "Ma propriété" { EDIT Text LENGTH 1200 }
}
```

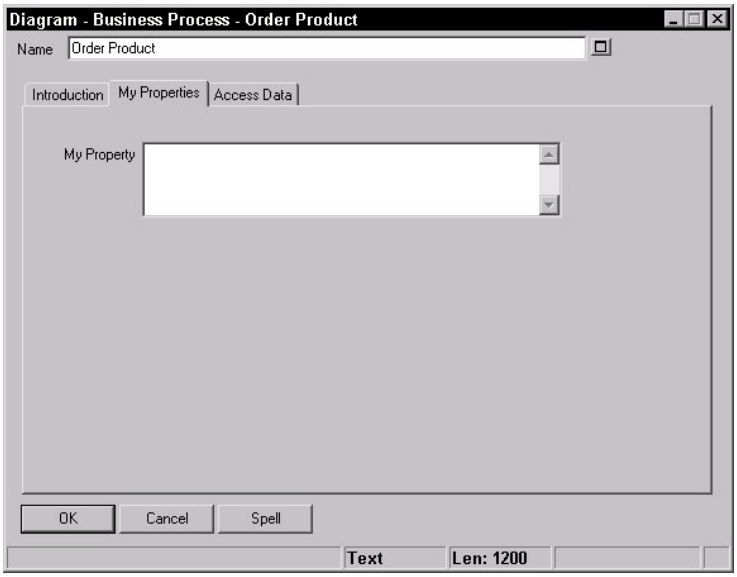


Figure 2-14. Boîte de dialogue Propriétés du diagramme révisées

Spécification des propriétés des types de symbole

propriétés de symbole sont fournies dans la page Symbole de la boîte de dialogue de définition d'un symbole.

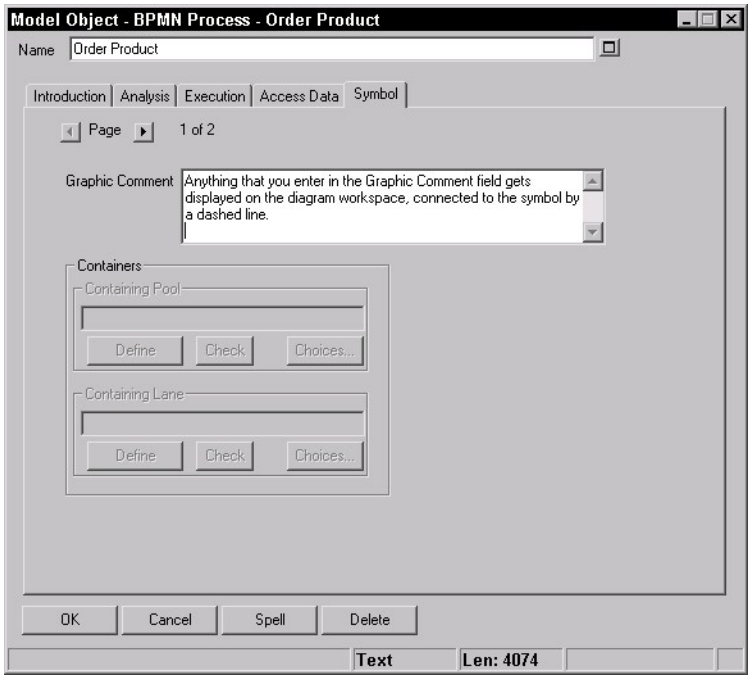


Figure 2-15. Boîte de dialogue Propriétés des symboles dans laquelle la propriété par défaut est *Commentaire graphique*

Commentaire graphique

La propriété par défaut de tous les symboles est *Commentaire graphique*. *Commentaire graphique* est défini comme une zone de texte de 4074 caractères. Tout ce que vous entrez dans la zone Commentaire graphique est affiché comme commentaire dans l'espace de travail du diagramme, connecté au symbole par une ligne. La ligne n'est représentée que si le commentaire graphique se trouve à une certaine distance du symbole. Vous pouvez ajuster cette distance en sélectionnant le symbole et en choisissant Format, Format du diagramme, Notation et en ajustant les options Ligne jusqu'au texte distant.

Vous pouvez également choisir d'afficher le commentaire graphique dans le symbole (sélectionnez le symbole, choisissez Format, Format du symbole, Position du texte, et désélectionnez l'option Placer le commentaire graphique à l'extérieur). Vous pouvez également activer/désactiver intégralement l'affichage du commentaire graphique (cliquez sur le symbole à l'aide du bouton droit de la souris, choisissez Mode Affichage, puis désélectionnez Commentaire graphique).

Ajout de propriétés supplémentaires pour un symbole

Pour ajouter davantage de propriétés pour un symbole, utilisez la syntaxe suivante :

```
SYMBOL type_symbole IN type_diagramme
{
PROPERTY-1 <nom_propriété>
  { <valeur_propriété> }
PROPERTY-2 <nom_propriété>
  { <valeur_propriété> }
PROPERTY-3 <nom_propriété>
  { <valeur_propriété> }
}
```

Il est important de spécifier le type de diagramme dans lequel se trouve le symbole auquel vous faites référence. Un symbole peut apparaître sur de nombreux types de diagramme différents.

Exemple

Par exemple, nous pouvons apporter les modifications suivantes au fichier USRPROPS.TXT :

```
SYMBOL "State" IN "Etat"
{
PROPERTY "Description courte"
  { EDIT Text LENGTH 1500 }
PROPERTY "Numéro" { EDIT Numeric LENGTH 4 }
}
```

Ces modifications ajoutent une *Description courte* et un *Numéro* aux propriétés d'un symbole d'état sur un diagramme Etat UML ; *Commentaire graphique* est toujours disponible. La boîte de dialogue modifiée est illustrée ci-dessous :

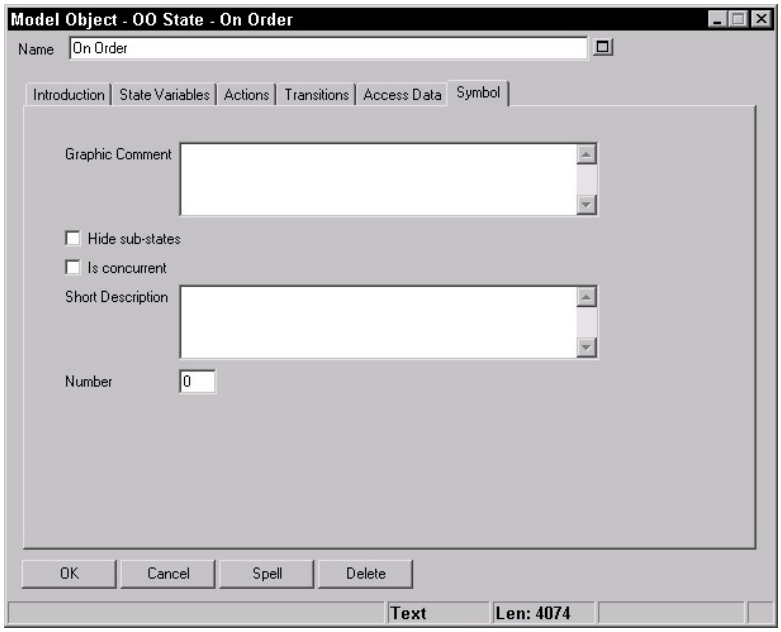


Figure 2-16. Boîte de dialogue Propriétés du diagramme révisées

Certains types de symbole peuvent apparaître sur de nombreux diagrammes différents. Dans l'exemple ci-dessus, il existe d'autres types de diagramme d'état dans Rational System Architect qui contiennent des symboles d'état, comme notamment les diagrammes Transition d'état des objets IDEF3, Transition d'état op. OV-06b et Transition d'état Ward & Mellor. Si nous souhaitons que les propriétés *Description courte* et *Numéro* apparaissent sur ces trois types, nous devons inclure trois fois le bloc de propriétés : une fois pour chaque type de diagramme.

```
SYMBOL "Etat" IN "Transition d'état des objets IDEF3"
{
PROPERTY "Description courte"
{ EDIT Text LENGTH 1500 }
PROPERTY "Numéro" { EDIT Numeric LENGTH 4 }
}

SYMBOL "Etat" IN "Transition d'état op. OV-06b"
{
PROPERTY "Description courte"
{ EDIT Text LENGTH 1500 }
PROPERTY "Numéro" { EDIT Numeric LENGTH 4 }
}

SYMBOL "Etat" IN "Transition d'état Ward & Mellor"
{
PROPERTY "Description courte"
{ EDIT Text LENGTH 1500 }
PROPERTY "Numéro" { EDIT Numeric LENGTH 4 }
}
```

Spécification des propriétés des types de définition

Toutes les définitions possèdent un *nom*. En outre, chacune d'elles possède la *description* de propriété par défaut, que nous abordons de manière plus détaillée, plus loin dans cette section. Il n'existe pas vraiment de boîte de dialogue **Définition** type car les définitions sont généralement uniques à l'intérieur d'une méthodologie et d'un type. Vous trouverez ci-après une boîte de dialogue pour une définition de classe.

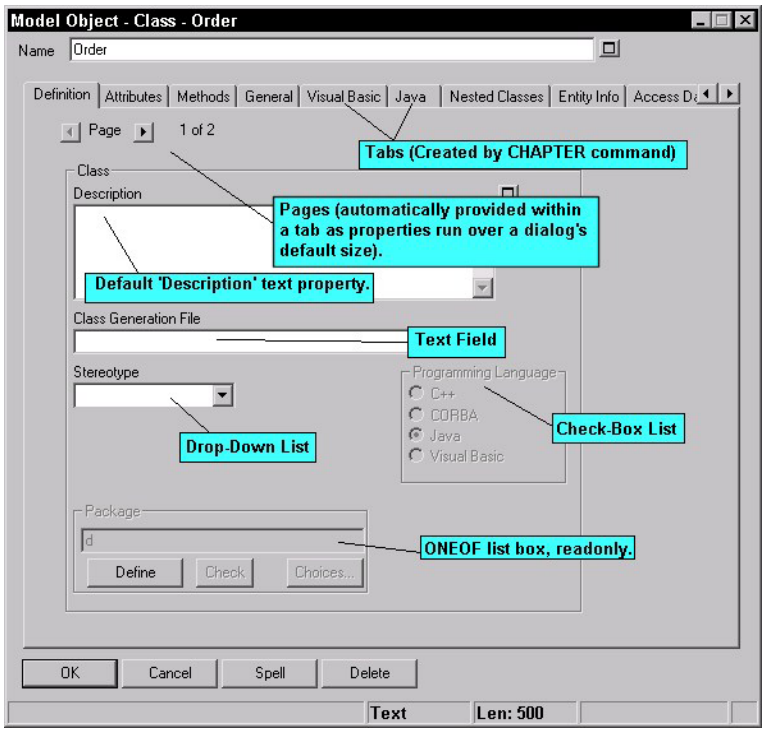


Figure 2-17. Boîte de dialogue Objet modèle (Type de définition Classe)

Syntaxe

Un bloc de définition commence par le mot-clé *Definition*, suivi d'une chaîne (l'argument) qui correspond au nom du type de définition. Son nom doit correspondre à l'un des noms connus de Rational System Architect : un nom qui se trouve dans le fichier SAPROPS.CFG, si vous modifiez une définition existante ou y ajoutez des entrées, ou un nom que vous avez créé à l'aide des commandes RENAME "Utilisateur 1" à RENAME "Utilisateur 150" (pour créer un type de définition, vous renommez l'un de ces 150 types de définition fournis par l'utilisateur). Les noms de type de définition qui contiennent des espaces doivent être placés entre guillemets (par exemple, "Utilisateur 1").

La définition de dictionnaire est délimitée par les mots clés `BEGIN...END` (ou des accolades ouvrantes et fermantes) **{ }**). Entre ces accolades se trouvent un ensemble de commandes de définition, qui sont chacune constituées du mot-clé de commande, `PROPERTY`, suivi de ses arguments. Lorsque vous appelez une boîte de dialogue **Objet dictionnaire**, ses pages sont alimentées par les propriétés désignées dans le bloc de définition.

Chaque entrée PROPERTY possède sa propre définition de sous-ensemble, elle aussi délimitée par les accolades et le mot clé **{EDIT... }**. Chaque définition est composée de phrases constituées à l'aide de mots clés, tels que *Boolean, Date, Expression, ExpressionOf, ListOf, Minispec, Numeric, OneOf, Text* et *Time*. Les détails des définitions de propriété sont abordés plus loin dans cette section.

En résumé, pour ajouter plusieurs propriétés pour une définition, utilisez la syntaxe suivante :

```
DEFINITION type_définition
{
PROPERTY-1 <nom_propriété>    { <valeur_propriété> }
PROPERTY-2 <nom_propriété>   { <valeur_propriété> }
PROPERTY-3 <nom_propriété>     { <valeur_propriété> }
}
```

Par exemple :

```
DEFINITION "Classe"
{
CHAPTER "Définition"
GROUP "Classe"
{
LAYOUT { COLS 2 ALIGN OVER TAB }
PROPERTY "Description"{ ZOOMABLE EDIT Text LENGTH
500 }
PROPERTY "Fichier d'en-tête de classe" { EDIT Text LABEL
"Fichier de génération de classe" LENGTH 80 }
PROPERTY "Stéréotype" { EDIT Text LIST "Stéréotypes de
classe" INIT_FROM_SYMBOL Default "" LENGTH 20 }
..}
```

Dans l'exemple ci-dessus, la première page par défaut de la définition qui, sauf indication contraire, s'intitule "Introduction" a été modifiée en "Définition" (par la commande CHAPTER "Définition").

Description

Comme mentionné au début de cette section, chaque définition contient la propriété par défaut *Description*. Sauf mention contraire dans le fichier SAPROPS.CFG, *Description* est défini comme une zone de texte de 4074 caractères. Vous pouvez augmenter la taille de zone d'une description dans le fichier USRPROPS.TXT en respécifiant simplement la propriété Description et en augmentant le nombre de caractères. Par exemple :

```
DEFINITION "Classe"
{
PROPERTY "Description"  { EDIT LENGTH  16000 LINES
5 }
}
```

L'exemple ci-dessus indique que la propriété Description d'une classe peut comporter 16 000 caractères, mais que seules les cinq premières lignes sont affichées dans la boîte de dialogue de la définition de classe.

Remarque importante : Quelques types de définition de Rational System Architect utilisent la propriété Description à des fins particulières. Par exemple, la définition d'une entité a été redéfinie comme commande LISTOF "Attribut" FROM "Données". La propriété *Description* des définitions de processus a été redéfinie comme *Minispec,* car le contenu des données des processus sont généralement des minispecs, du texte anglais structuré et du pseudocode. Dans chacun de ces cas, la propriété Description a été également renommée, en Attribut ou Minispec, respectivement. Voici par exemple la spécification d'une définition de processus dans le fichier SAPROPS.CFG :

```
        DEFINITION "Processus"
{
        PROPERTY "Description"
        { EDIT Minispec LENGTH 750 LABEL "Minispec" }
        ..}
```

Il est important de noter que lors de la génération de rapports, le nom de la propriété est *Description* et doit être utilisé en tant que tel dans toute référence.

Instructions PROPERTY

Vous spécifiez des instructions PROPERTY dans les spécifications de diagramme, symbole ou définition. La syntaxe d'une instruction PROPERTY est la suivante :

```
PROPERTY nom-propriété
{ EDIT type-édition
}
```

Le tableau ci-après inclut toutes les entrées obligatoires et facultatives d'une instruction PROPERTY dans le fichier USRPROPS.TXT.

Tableau 2-3. Entrées obligatoires et facultatives d'une instruction PROPERTY

Entrée	Obligatoire Facultative	Remarque

PROPERTY nom_propriété
{
}
	Obligatoire	Vous pouvez utiliser des accolades ouvrantes et fermantes, {..}, ou des instructions BEGIN .. END.

EDIT type-édition	Facultative	[Boolean | Date | ExpressionOf DATA | ListOf "dictionary-type" | Minispec | Numeric | OneOf "dictionary-type" | Text | Time]
LABEL chaîne_linellé	Facultative	Nom du contrôle dans la boîte de dialogue ; remplace le nom de la propriété, qui est la valeur par défaut

Entrée	Obligatoire Facultative	Remarque
LENGTH argument-longueur	Facultative	Longueur maximale de la zone en caractères 0 < valeur numérique < 4095
LIST nom-liste	Facultative	Indique qu'une liste de nom-liste sera affichée lorsque la propriété est sélectionnée comme entrée par l'utilisateur ; l'utilisateur peut sélectionner un nom dans la liste ou saisir une autre valeur
LISTONLY LIST nom-liste	Facultative	Indique que la seule entrée autorisée est celle d'une liste affichée en option
MINIMUM valeur numérique MAXIMUM valeur numérique	Facultative	Valeur numérique minimale/maximale de la zone
DISPLAY { FORMAT type-format LEGEND nom-légende }	Facultative	Définit l'une des 37 propriétés affichables possibles du symbole
DEFAULT chaîne_défaut	Facultative	S'il n'existe aucune entrée d'utilisateur, l'entrée de la chaîne est utilisée

Tableau 2-3. Entrées obligatoires et facultatives d'une instruction PROPERTY (suite)

Entrée	Obligatoire Facultative	Remarque
READONLY	Facultative	Arrête toute entrée du clavier ou d'une liste affichée
INVISIBLE VISIBLE	Facultative	Rend la propriété invisible ou visible. Utilisez cette entrée pour inverser la valeur qui se trouve dans le fichier SAPROPS.
CHECKOUT type-initial	Facultative	Valeur qui est automatiquement remplie lorsque l'entrée de dictionnaire est réservée. [DATE | TIME | AUDITID]
FREEZE type-initial	Facultative	Valeur qui est automatiquement remplie lorsque l'entrée de dictionnaire est gelée. [DATE | TIME | AUDITID]

Entrée	Obligatoire Facultative	Remarque
INITIAL type-initial	Facultative	Valeur qui est automatiquement remplie lors du premier accès au dictionnaire et de la première sauvegarde de ce dernier [DATE | TIME | AUDITID]
UPDATE type-mise-à-jour	Facultative	Valeur qui est automatiquement remplie lors du premier accès au dictionnaire et de la première sauvegarde de ce dernier, ainsi que lors des accès et sauvegardes ultérieures [DATE | TIME | AUDITID]
HELP	Facultative	35 à 40 caractères affichés dans la barre d'état de la boîte de dialogue lorsque le contrôle est mis en évidence.

Utilisation de ListOf, OneOf et ExpressionOf

Les mots clés ListOf, OneOf et ExpressionOf offrent un concept très puissant utilisé dans l'intégralité du métamodèle de Rational System Architect. Chacun de ces mots clés permet d'indiquer qu'une propriété d'une définition fait référence à un autre type d'objet (diagrammes, symboles ou définitions).

Vous pouvez donc dire qu'une classe contient une liste de méthodes (commande ListOf), qu'un message entre deux objets fait référence à une méthode de l'objet appelant (commande OneOf) ou qu'un processus exprime les procédures effectuées sur les données (ExpressionOf). Nous aborderons chacune de ces expressions dans les sections qui suivent.

Remarque : Comme avec tout mot clé spécifié dans le fichier USRPROPS.TXT, la casse des mots clés ListOf, OneOf et ExpressionOf n'est pas importante. Dans le présent document, nous utilisons des majuscules pour tous les mots clés, excepté dans cette section, car ListOf, OneOf et ExpressionOf décrivent mieux ces mots clés que les versions "entièrement en majuscules" (LISTOF, ONEOF et EXPRESSIONOF).

ListOf

La commande ListOf permet de spécifier qu'une propriété contient une liste d'autres objets (diagrammes, symboles ou définitions). Par exemple, une classe contient une propriété appelée Attributs, qui répertorie les attributs de classe. Un attribut de classe est un type de définition en lui-même, qui possède son propre ensemble de propriétés. Le type d'objet référencé doit déjà avoir été définie dans le fichier SAPROPS.CFG ou au début du fichier USRPROPS.TXT.

Comparez la propriété ListOf à une simple liste textuelle. Le nombre d'éléments de la liste ListOf augmente à mesure que les utilisateurs ajoutent des définitions au référentiel ; pour une simple liste, le nombre d'élément de la liste présentée à l'utilisateur est statique (en fonction de l'instruction LIST placée au début du fichier USRPROPS.TXT).

La syntaxe de la commande ListOf est la suivante :

```
PROPERTY "Votre propriété" { EDIT LISTOF <"Type de définition référencé"> LENGTH 1200}
```

Filtrage de la liste des éléments

La liste d'éléments fournie dans la liste d'une commande ListOf peut être filtrée. Des mots clés de filtrage tels que OF DEFINITION REFERENCED IN et OF DEFINITION AND SUPERS REFERENCED IN sont disponibles. Pour plus d'informations sur ces mots clés, reportez-vous au Chapitre 3.

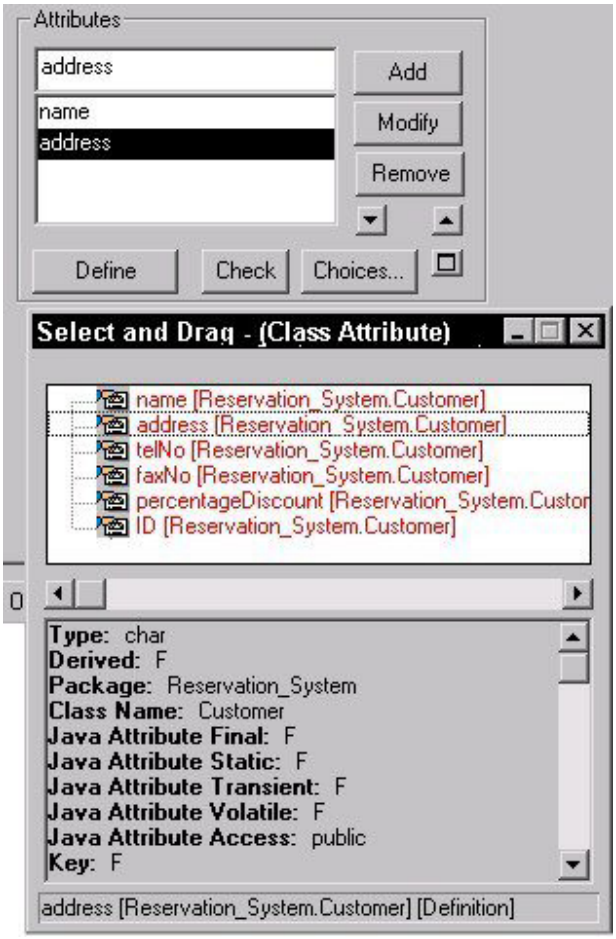
Exemple :

L'exemple ci-après montre le code d'une définition d'objet, qui inclut la propriété "Attributs", qui correspond à une liste d'attributs de classe (ListOf "Attributs de classe"). "Attributs de classe" est un autre type de définition, défini dans le fichier SAPROPS.CFG.

```
Definition "Objet"
{ ..
PROPERTY "Attributs" { ZOOMABLE EDIT LISTOF
"Attribut de classe" OF DEFINITION REFERENCED IN
"Classe"
KEYED BY {"Package", "Nom de classe":"Classe", Nom}
LENGTH 4096 DISPLAY {FORMAT COMPONENT_SCRIPT
_FmtNewUMLObjInstAttr LEGEND "$$FORCE$$"} }
..}
```

La figure ci-après illustre la boîte de dialogue ListOf par défaut créée par la commande LISTOF. Elle inclut un bouton Choix qui, lorsqu'il est activé, présente une liste de touts les définitions du type référencé ("Attribut de classe" dans cet exemple) dans l'encyclopédi. La commande OF DEFINITION REFERENCED IN, dans le code ci-dessus, spécifie que seuls les attributs de classe contenus dans la classe de l'objet sont répertoriés.

Figure 2-18. Exemple de liste LISTOF.



Création de grilles pour la commande ListOf

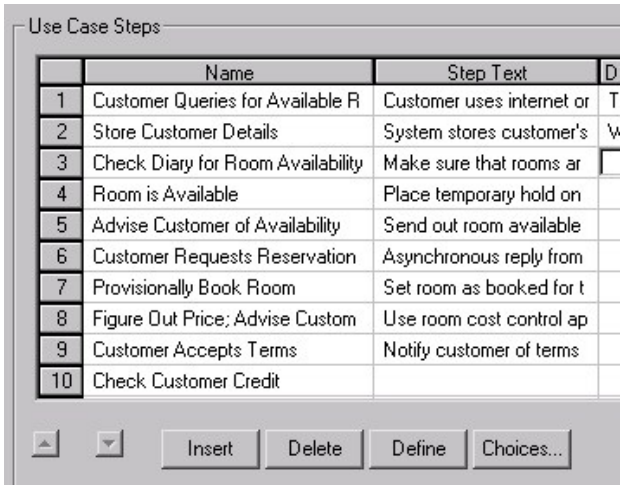
Vous pouvez présenter les élements d'une propriété ListOf sous forme de grille en ajoutant le mot clé ASGRID.

Exemple :

```
Definition "Cas d'utilisation"
{
CHAPTER "Etapes"
PROPERTY "Etapes de cas d'utilisation" { EDIT
COMPLETE LISTOF "Etapes de cas d'utilisation" KEYED
BY { "Package", "Nom du cas d'utilisation":Nom, Nom}
ASGRID LENGTH 1200 }
```

```
}
```

Figure 2-19. Exemple de liste LISTOF ASGRID.



Listes hétérogènes pour la commande ListOf

Une instruction ListOf type fournit une liste d'un type d'objet. Vous pouvez également créer une liste qui fait référence à plusieurs types d'objet, à l'aide du mot clé HETEROGENEOUSLISTOF.

Exemple :

```
Definition " Procédure"
{
PROPERTY "Procédure sous-jacente" { EDIT
HETEROGENEOUSLISTOF "Cas d'utilisation",
"Classe", "Méthode", "Etape de cas d'utilisation"
READONLY}
..}
```

Dans l'exemple ci-dessus, la propriété "Procédure sous-jacente" de la définition "Procédure" peut être alimentée avec les définitions du type Cas d'utilisation, Classe, Méthode et/ou Etape du cas d'utilisation.

Pour plus d'informations sur la commande HETEROGENEOUSLISTOF, reportez-vous au Chapitre 3.

OneOf

Une zone de liste OneOf contient une zone de liste permettant à l'utilisateur de sélectionner une (et une seule) liste d'objets (diagrammes, symboles ou définitions) d'un certain type. Le type d'objet référencé doit déjà avoir été définie dans le fichier SAPROPS.CFG ou au début du fichier USRPROPS.TXT.

Exemple :

```
DEFINITION "Incident"
{
PROPERTY "Affecté à"{EDIT ONEOF "Risque" LENGTH
100}
..}
```

Figure 2-20. Exemple de zone de liste ONEOF.

Filtrage de la liste des éléments

Tout comme pour la liste ListOf, une liste d'éléments fournie dans la liste d'une commande OneOf peut être filtrée. Des mots clés de filtrage tels que OF DEFINITION REFERENCED IN et OF DEFINITION AND SUPERS REFERENCED IN sont disponibles. Pour plus d'informations sur ces mots clés, reportez-vous au Chapitre 3.

Liste OneOf hétérogène

Une instruction OneOf type fournit une liste d'un type d'objet. Tout comme pour la liste ListOf, vous pouvez également créer une liste OneOf qui fait référence à plusieurs types d'objet, à l'aide du mot clé HETEROGENEOUS**ONE**OF.

Pour plus d'informations sur la commande HETEROGENEOUSONEOF, reportez-vous au Chapitre 3.

ExpressionOf

ExpressionOf permet d'exprimer des références aux objets à l'aide d'opérateurs et de délimiteurs complexes. Rational System Architect s'attend à ce que vous utilisiez ExpressionOf pour faire référence à des éléments de données et des structures de données (DATA), mais il n'est pas restreint à cette utilisation.

Les références définies avec la commande ExpressionOf sont entrées dans une boîte de dialogue, à l'aide de la syntaxe suivante :

```
A + B + C

ou

A +
B +
C

ou

A
B
C
```

Les éléments peuvent être écrits sur une ou plusieurs lignes ; la division entre un élément et le suivant est déterminé par un espace blanc. Par convention, un signe + permet de séparer chaque élément de données, mais cela n'est pas obligatoire.

Les opérateurs et délimiteurs spéciaux suivants peuvent être utilisés pour la spécification des expressions :

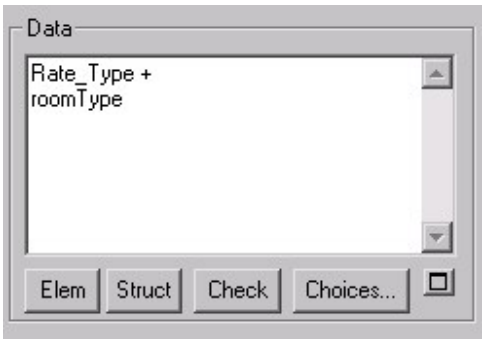
Tableau 2-5. Opérateurs et délimiteurs spéciaux utilisés lors de la spécification des expressions

+	Et (facultatif)
[...|...]	L'un ou l'autre
{...}	Itérations de
i{...}j	Permet les itérations de 'i' à 'j' de
(...)	Le composant entre paranethèses est facultatif
@	Le composant correspond à une zone de clé
@n	Le composant est le nième élément constituant une clé composée
...	Le texte entre les étoiles est un commentaire
/.../	Le texte entre les barres obliques est un commentaire, mais a une signification dans le générateur de schémas

Les sous-expressions peuvent être imbriquées dans d'autres expressions. Par exemple, ITERATIONS OF peut être inclus à l'intérieur de crochets EITHER OR.

[n1{...}n2 | n3{...}n4]

Figure 2-21. Exemple de zone de liste EXPRESSIONOF.



Commande ZOOMABLE

La commande **ZOOMABLE** permet à l'utilisateur d'agrandir temporairement la taille d'une zone de liste afin de saisir ou de consulter plus facilement de grands blocs de texte. Les emplacements les plus courants pour ajouter cette commande sont une définition de processus, dans laquelle des minispecs sont généralement entrées, ou la propriété de description d'une entité, où les informations sur les clés externes ont tendance à être trop longues.

La commande du fichier USRPROPS.TXT est écrite comme suit :

```
DEFINITION "Processus"
{
PROPERTY "Description"
{ ZOOMABLE }
}
```

La commande **ZOOMABLE** ajoute un petit bouton dans le coin droit de la zone de liste. Ce bouton apparaît avec un signe plus (+) lorsque vous effectuez un zoom avant, et avec un signe moins (-) lorsque vous effectuez un zoom arrière.

L'effet de cette commande est visible dans les deux illustrations de la Figure 2-22. L'illustration supérieure montre la zone de minispec dans son état non développé habituel ; dans l'illustration inférieure, la zone de liste a été agrandie pour couvrir l'intégralité de la page de la boîte de dialogue.

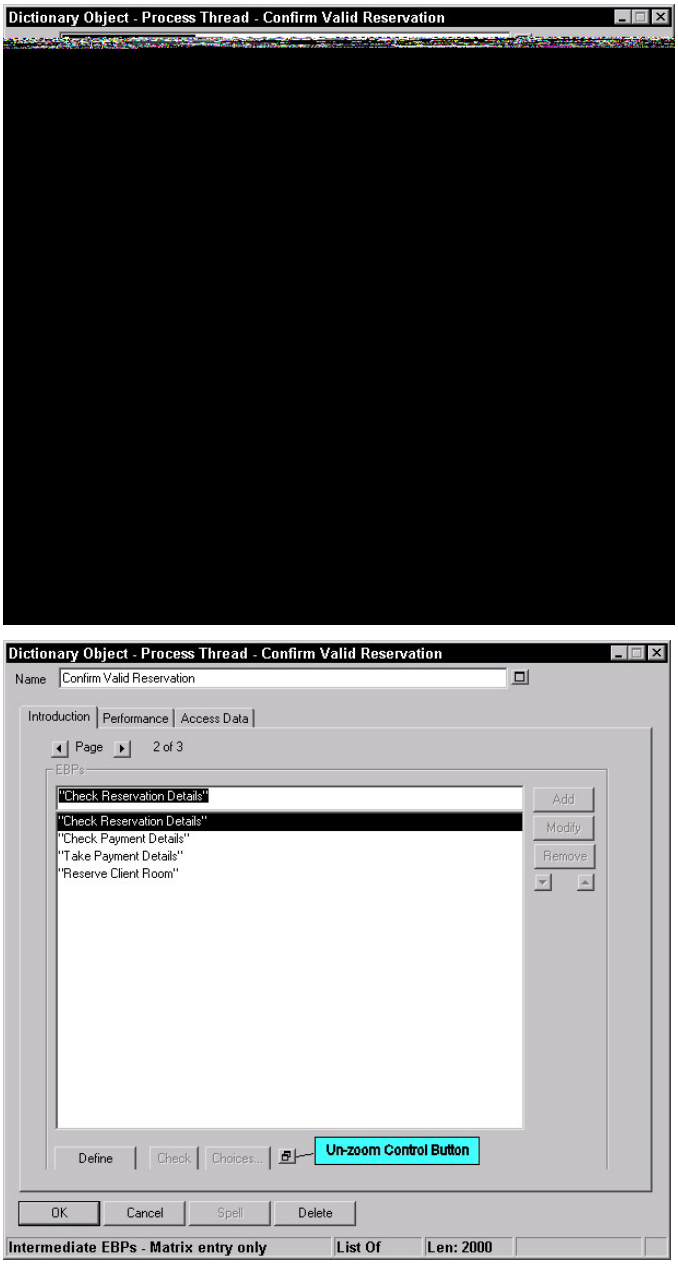


Figure 2-22. Zone de liste dans un état sans zoom et avec zoom.

Modification de l'aspect esthétique des boîtes de dialogue

Vous pouvez contrôler partiellement l'affichage des contrôles et de leur libellé dans les boîtes de dialogue. Par exemple, les libellés peuvent être affichés sur le contrôle, directement en regard du contrôle ou être séparés de ce dernier par un espace déterminé par le libellé le plus long d'un groupe.

Il n'est pas nécessaire de déterminer le nombre de contrôles pouvant apparaître sur une page de la boîte de dialogue. Rational System Architect analyse automatiquement le nombre de contrôles pouvant tenir sur une boîte de dialogue en tenant compte de l'espace disponible pour l'affichage et divise une boîte de dialogue en plusieurs pages auxquelles vous pouvez accéder via une flèche Page, en haut à gauche de la boîte de dialogue.

Vous pouvez spécifier votre propre arrangement de contrôles dans une boîte de dialogue en utilisant la commande **LAYOUT** pour spécifier des colonnes, en positionnant les libellés des contrôles et les justifications, la commande **CHAPTER** pour créer des pages et la commande **GROUP** pour créer des groupes de contrôles de propriété. De plus, vous pouvez spécifier la position exacte de chaque contrôle et libellé d'une page donnée d'une boîte de dialogue à l'aide des contrôles de positionnement (pour des instructions, voir la page 2-104, *Positionnement des contrôles et des étiquettes*).

Commande LAYOUT

La commande LAYOUT permet de spécifier le nombre de contrôles de propriété de colonne pouvant être disposés dans une boîte de dialogue et d'indiquer comment les titres (ou libellés) des contrôles sont positionnés (à gauche du contrôle ou dessus), etc.

L'utilisation de la commande LAYOUT est facultative. Si vous ne l'utilisez pas, Rational System Architect déploie le schéma de présentation par défaut. Dans ce dernier, tous les contrôles sont disposés dans une même colonne, avec le nom (ou le libellé) de chacun placé à gauche du contrôle (commande ALIGN LABEL).

Vous pouvez spécifier une commande LAYOUT dans des commandes CHAPTER (qui correspondent à des pages de la boîte de dialogue résultante) et des commande GROUP d'une spécification de diagramme, de symbole ou de définition. La commande LAYOUT a les effets suivants dans une commande CHAPTER ou GROUP :

Dans un chapitre : Vous pouvez spécifier une commande LAYOUT unique pour chaque chapitre d'une spécification de diagramme, de symbole ou de définition. Tous les contrôles de propriété, et notamnent des groupes entiers de contrôles, sont disposés en fonction de la commande LAYOUT du chapitre. Si vous spécifiez plusieurs commandes LAYOUT dans un chapitre, toutes les commandes LAYOUT de ce chapitre sont ignorées et la disposition par défaut est utilisée à la place.

Dans un groupe : Vous pouvez spécifier une commande LAYOUT dans un groupe, de sorte que les propriétés du groupe soient disposées conformément à la spécification LAYOUT du groupe. Si vous spécifiez plusieurs commandes LAYOUT dans un groupe, toutes les commandes LAYOUT de ce groupe sont ignorées et la disposition par défaut est utilisée à la place.

Exemple de classement des commandes LAYOUT et effets correspondants :

DIAGRAM (ou SYMBOL ou DEFINITION)

CHAPTER 1
LAYOUT 1
PROPERTY – disposée (dans le chapitre) conformément à LAYOUT 1
PROPERTY – disposée (dans le chapitre) conformément à LAYOUT 1
GROUP – disposée (dans le chapitre) conformément à LAYOUT 1
LAYOUT 2
PROPERTY – disposée (dans le groupe) conformément à LAYOUT 2
PROPERTY – disposée (dans le groupe) conformément à LAYOUT 2
GROUP – disposée (dans le chapitre) conformément à LAYOUT 1
LAYOUT 3
PROPERTY – disposée (dans le groupe) conformément à LAYOUT 3
PROPERTY – disposée (dans le groupe) conformément à LAYOUT 3
CHAPTER 2
LAYOUT 4
PROPERTY – disposée (dans le chapitre) conformément à LAYOUT 4
CHAPTER 3
LAYOUT 5
PROPERTY – disposée conformément au schéma par défaut car il existe deux commandes LAYOUT (5 et 6) dans ce chapitre
LAYOUT 6

Disposition de la page "Introduction"

Notez que le premier chapitre d'un diagramme, d'un symbole ou d'une définition, qui inclut la propriété Description, est toujours disposée par le schéma de présentation par défaut : une présentation en une colonne, avec le libellé "Description" à gauche de la zone de texte.

Comportement de disposition par défaut

Si un contrôle de propriété est trop large pour tenir dans la structure de colonne spécifiée d'une commande LAYOUT, ce contrôle est présenté sur une colonne pour s'adapter à la boîte de dialogue ou au groupe ; les autres contrôles dont la largeur est suffisante pour être disposé conformément à la commande LAYOUT le sont en conséquence.

Par exemple, si vous avez spécifié une présentation en quatre colonnes pour un groupe qui se trouve lui-même dans un chapitre (page) pour lequel une présentation en deux colonnes est spécifiée et que l'une des propriétés du groupe est trop large pour tenir dans l'espace disponible, mais que les autres sont asez étroites pour tenir dans une présentation en quatre colonnes, la propriété trop large est présentée seule et les autres propriétés sont disposées conformément à la présentation en quatre colonnes dans le groupe.

Exemple

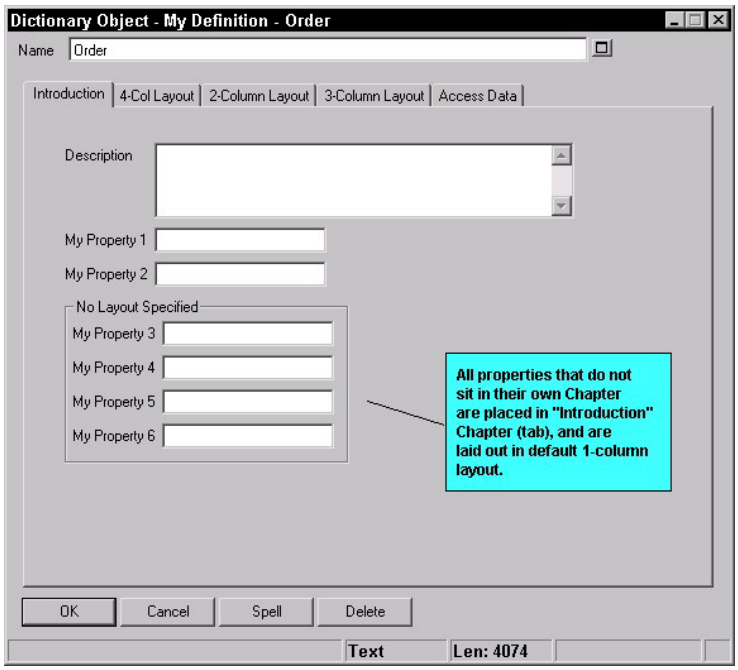
Dans l'exemple ci-après, nous examinons l'effet de la commande LAYOUT dans les instructions CHAPTER et GROUP d'une définition spécifiée par l'utilisateur, nouvellement définie.

```
RENAME DEFINITION "Utilisateur 1 TO "Ma définition"
DEFINITION "Ma définition"
{
LAYOUT { COLS 3 ALIGN OVER TAB }
   PROPERTY "Ma propriété 1"{ EDIT Text Length 10}
   PROPERTY "Ma propriété 2"{ EDIT Text Length 10}
   GROUP "Aucune disposition spécifiée" {
     PROPERTY "Ma propriété 3"{ EDIT Text Length 10}
     PROPERTY "Ma propriété 4"{ EDIT Text Length 10}
     PROPERTY "Ma propriété 5"{ EDIT Text Length 10}
     PROPERTY "Ma propriété 6"{ EDIT Text Length 10}
}
  CHAPTER "Présentation en 4 col"
   LAYOUT { COLS 4 ALIGN OVER TAB }
   GROUP "Groupe de deux colonnes" {
      LAYOUT { COLS 2 ALIGN OVER TAB }
      PROPERTY "G1"{ EDIT Boolean LENGTH 1 DEFAULT "F"}
      PROPERTY "G2"{EDIT Boolean LENGTH 1 DEFAULT "F"}
      PROPERTY "G3"{ EDIT Boolean LENGTH 1 DEFAULT "F"}
      PROPERTY "G4"{ EDIT Boolean LENGTH 1 DEFAULT "F"}
      }
   GROUP "Groupe d'une colonne" {
      LAYOUT { COLS 1 ALIGN OVER   TAB }
      PROPERTY "Propriété de groupe 5"{ EDIT Text Length 10}
      PROPERTY "Propriété de groupe 6"{ EDIT Text Length 10}
      }
    PROPERTY "Ma propriété 7"{ EDIT Text Length 5}
    PROPERTY "Ma propriété 8"{ EDIT Text Length 5}
    PROPERTY "Ma propriété 9"{ EDIT Text Length 5}
    PROPERTY "Ma propriété 10"{ EDIT Text Length 5}
    PROPERTY "Ma propriété 11"{ EDIT Text Length 1200}
    PROPERTY "Ma propriété 12"{ EDIT Text Length 1200}

  CHAPTER "Présentation en 2 colonnes"
   LAYOUT { COLS 2 ALIGN LABEL TAB }
    PROPERTY "Ma propriété 13"{ EDIT Text Length 10}
    PROPERTY "Ma propriété 14"{ EDIT Text Length 10}
    PROPERTY "Ma propriété 15"{ EDIT Text Length 10}
    PROPERTY "Ma propriété 16"{ EDIT Text Length 10}
    PROPERTY "Ma propriété 17"{ EDIT Text Length 10}
```

```
    GROUP "Groupe de trois colonnes" {
      LAYOUT { COLS 3 ALIGN OVER TAB }
      PROPERTY "G5"{ EDIT Boolean LENGTH 1 DEFAULT "T"}

    PROPERTY "G6"{EDIT Boolean LENGTH 1 DEFAULT "T"}
    PROPERTY "G7"{ EDIT Boolean LENGTH 1 DEFAULT "T"}
    PROPERTY "G8"{ EDIT Boolean LENGTH 1 DEFAULT "T"}
  }
CHAPTER "Présentation en 3 colonnes"
LAYOUT { COLS 3 ALIGN OVER TAB }
PROPERTY "Ma propriété 18"{ EDIT Text Length 10}
PROPERTY "Ma propriété 19"{ EDIT Text Length 10}
PROPERTY "Ma propriété 20"{ EDIT Text Length 10}
PROPERTY "Ma propriété 21"{ EDIT Text Length 10}
PROPERTY "Ma propriété 22"{ EDIT Text Length 10}
PROPERTY "Ma propriété 23"{ EDIT Text Length 10}
}
```

Nous examinons le code de ce fichier USRPROPS.TXT dans les figures ci-après. La première figure montre que la première commande de disposition de la définition, LAYOUT { COLS 3 ALIGN OVER TAB }, est ignorée, car elle n'est pas affectée à un chapitre et qu'elle ne peut pas remplacer la disposition de la première page "Introduction", qui correspond à la disposition sur une colonne par défaut.

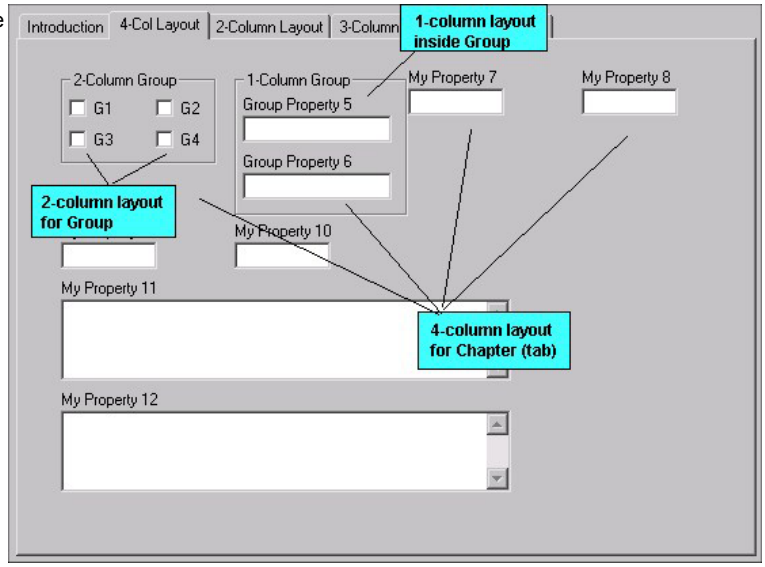
Figure 2-23. La première commande LAYOUT est ignorée car elle n'est pas affectée à un chapitre (page).



La deuxième page de la boîte de dialogue est spécifiée par la commande CHAPTER "Présentation en 4 col". Il s'agit d'une disposition sur 4 colonnes, avec le titre ou le libellé de chaque contrôle placé sur ce dernier (CHAPTER "Présentation en 4 col" LAYOUT { COLS 4 ALIGN OVER TAB }.

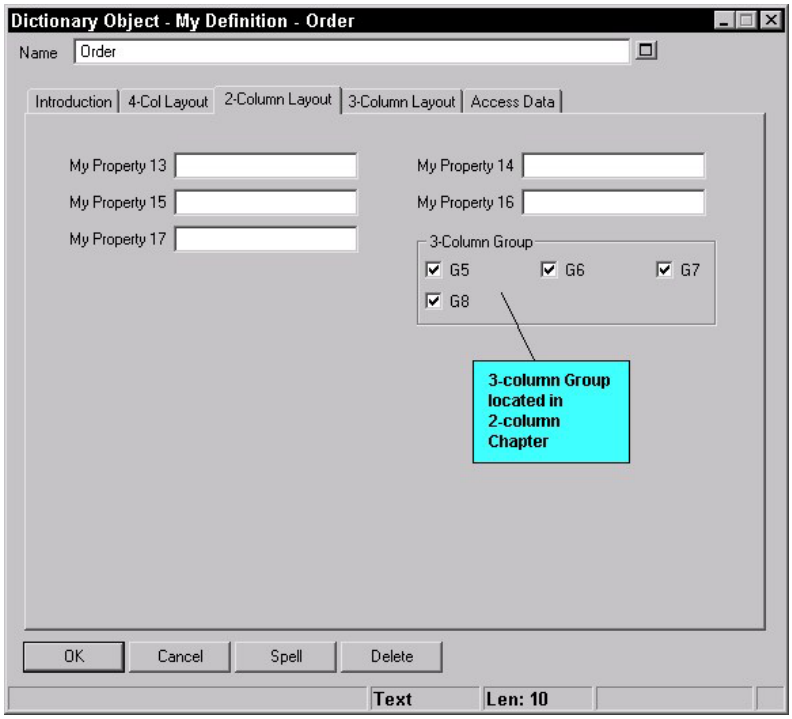
D'après la figure qui suit vous pouvez constater que même les groupes entiers (par exemple, "Groupe de deux colonnes" et "Groupe d'une colonne" sont disposés dans le chapitre selon une disposition en quatre colonnes, comme les propriétés (par exemple, "Ma propriété 7" à "Ma propriété 10"). Les propriétés trop larges pour tenir dans un schéma de présentation de quatre colonnes sont disposées sur une colonne (par exemple, "Ma propriété 11" et "Ma propriété 12", qui ont toutes deux une longueur de 1200).

Figure 2-24. Chapitre de quatre colonnes contenant des groupes de deux colonnes et d'une colonne.



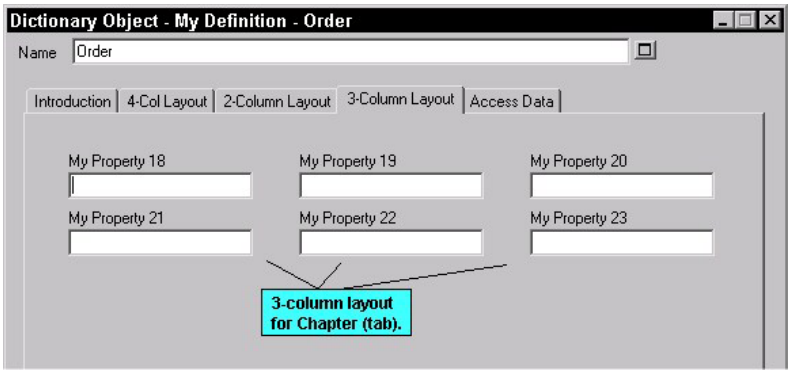
Le chapitre de deux colonnes, de la même manière, contient des propriétés disposées en deux colonnes et notamment un groupe, dans lequel les propriétés sont présentées en trois colonnes.

Figure 2-25. Chapitre de deux colonnes contenant un groupe de trois colonnes



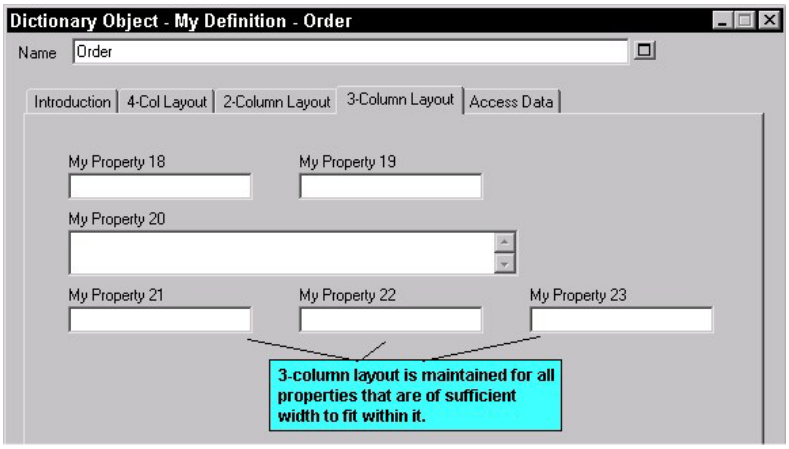
Le chapitre final contient des propriétés dans une présentation en trois colonnes. Notez que ces propriétés sont assez étroites (Longueur 10) pour tenir dans trois colonnes.

Figure 2-26. Chapitre de deux colonnes contenant un groupe de trois colonnes



Si l'une de ces propriétés est trop large pour tenir dans une présentation en trois colonnes, cette propriété est présentée indépendamment des autres, dans un format en une colonne. Si vous affectez à "Ma propriété 20" la valeur LENGTH 100 au lieu de LENGTH 10, son contrôle est affiché comme illustré dans la figure ci-après. Toutes les autres propriétés de la boîte de dialogue restent présentées en trois colonnes.

Figure 2-27. Chapitre de deux colonnes contenant un groupe de trois colonnes, avec une propriété trop large.



Arguments de la commande LAYOUT

Les valeurs valides des sous-commandes utilisées dans la commande **LAYOUT** sont les suivantes :

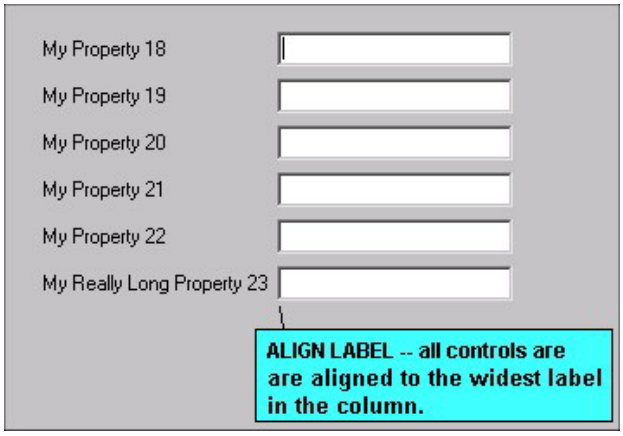
LAYOUT { [ALIGN BODY | ALIGN LABEL | ALIGN OVER] [PACK | TAB] COLS <numéro> **}**

La séquence des sous-commandes n'est pas importante.

Alignez les titres (ou libellés) des propriétés avec leur contrôle

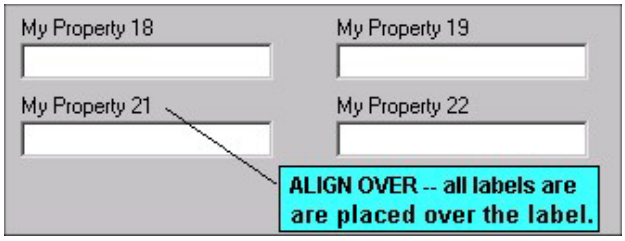
Chaque propriété possède un titre ou un nom. N'oubliez pas que vous pouvez modifier le libellé d'une propriété à l'aide de la commande LABEL. La commande ALIGN place le titre d'une propriété (ou son libellé si elle a été renommée) à une certaine position en regard du contrôle lui-même, comme suit :

1. **ALIGN BODY** et **ALIGN LABEL** : Tous les contrôles sont alignés un espace à droite du libellé le plus long de cette colonne.



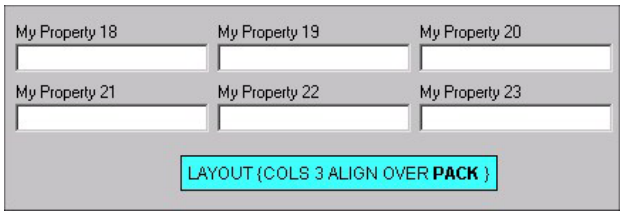
(Remarque – ALIGN BODY plaçait tous les contrôles à un espace à droite du libellé, mais il a été par la suite modifié pour se comporter comme ALIGN LABEL).

2. **ALIGN OVER** : Le libellé se trouve sur le contrôle.

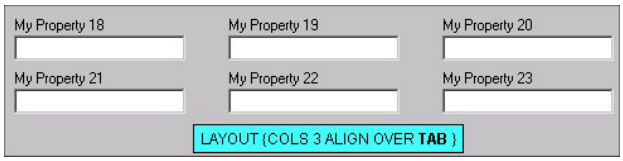


Positionnement vertical

1. **PACK** *:* Les ensembles de commandes et de libellés sur plusieurs colonnes sont séparés entre eux à droite par l'espace minimum.



2. **TAB** : Les contrôles et les libellés sur plusieurs colonnes sont séparés par des tabulations pour que les entrées de chaque ligne soient placées directement sous les entrées de la ligne supérieure.



Colonnes

COLS <nombre_de_colonnes> : Contrôle le nombre de colonnes dans lesquelles les propriétés sont divisées

Justify

JUSTIFY : Cette commande n'est plus utilisée dans les fichiers SAPROPS.CFG et USRPROPS.TXT. Elle est ignoré par l'analyseur syntaxique du fichier USRPROPS.TXT. Elle alignait tous les contrôles sur les marges de droite et de gauche de la page de la boîte de dialogue.

Création de pages à l'aide de la commande CHAPTER

La commande **CHAPTER** permet de contrôler le contenu d'une page de boîte de dialogue et de générer une *page*. Si la quantité d'informations est supérieure à celle qui peut être affichée dans une page, plusieurs pages sont automatiquement créées.

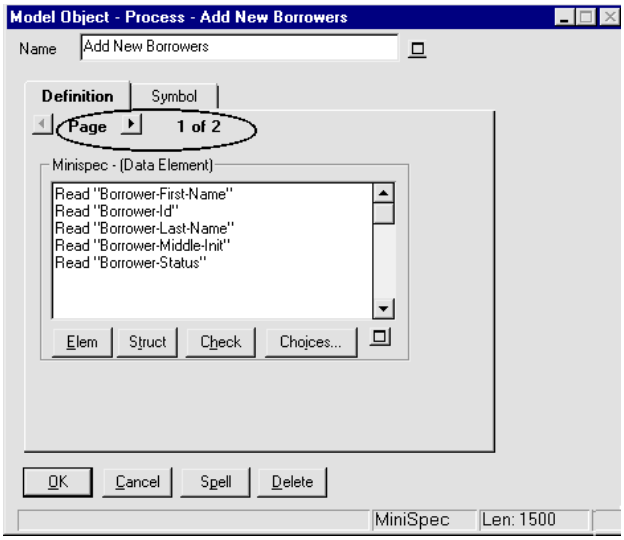


Figure 2-28. Boîte de dialogue Objet modèle illustrant deux pages pour la page Définition.

Syntaxe et positionnement de la commande CHAPTER

Pour créer une page, utilisez la commande CHAPTER et spécifiez le nom de la page comme argument (en plaçant le nom entre guillemets s'il contient des espaces). La commande CHAPTER ne nécessite pas d'accolades ouvrantes et fermantes, { }, ou un bloc d'instruction BEGIN .. END.

CHAPTER Nom_page

ou

CHAPTER "Nom de la page"

Toutes les propriétés répertoriées dans la spécification après la commande CHAPTER sont comprises dans ce chapitre (ou cette page), jusqu'à ce que la prochaine commande CHAPTER soit rencontrée. La commande **CHAPTER** peut être placée en tout point d'une spécification de diagramme, de symbole ou de définition du fichier USRPROPS.TXT, excepté dans un bloc GROUP.

Remarque : Le terme "CHAPTER" est utilisé pour cette commande à la place du terme "TAB" car "TAB" a toujours eu une signification différente dans le fichier USRPROPS.TXT (cette commande est utilisée dans la commande LAYOUT).

Les règles suivantes s'appliquent pour la commande CHAPTER :

- Une propriété ajoutée via USRPROPS.TXT sans commande **CHAPTER** est placée à la fin de toutes les instructions de propriété de définition et, par conséquent, dans la dernière page du dernier onglet 'Définition' qui constitue la boîte de dialogue de définition. (Notez qu'une définition peut contenir un ou plusieurs onglets d'informations liées au symbole, qui apparaissent à la fin de la boîte de dialogue de définition. Le premier de ces onglets 'symbole' s'intitule souvent 'Symbole', mais il peut être renommé. Les pages de symbole n'apparaissent que si vous ouvrez la boîte de dialogue de définition d'un symbole dans un diagramme ; si vous ouvrez une définition à partir de l'explorateur, ils n'apparaissent pas).
- Vous pouvez ajouter une propriété à une page existante appelée dans le fichier SAPROPS.CFG en respécifiant la page avec une commande **CHAPTER** dans le fichier USRPROPS.TXT. Les propriétés que vous ajoutez sont placées à la fin de cette page dans la boîte de dialogue.
- Les pages des nouvelles commandes **CHAPTER** que vous ajoutez aux spécifications de diagramme, symbole ou définition existantes sont placées à la fin de la boîte de dialogue, après les pages pré-existantes.

- Si vous souhaitez utiliser une commande **GROUP**, celle-ci doit être imbriquée dans une commande **CHAPTER**.

Utilisation de la commande LAYOUT dans une commande CHAPTER

La commande LAYOUT peut apparaître n'importe où sous une commande CHAPTER et s'appliquera à tous les contrôles des propriétés du chapitre. Si vous spécifiez plusieurs commandes LAYOUT dans un chapitre, l'analyseur syntaxique du fichier USRPROPS.TXT les rejette toutes et fournit la disposition par défaut (COLS 1 ALIGN LABEL).

GROUP

La commande GROUP permet de placer un ensemble de contrôles de propriété dans une zone de groupe.

Syntaxe de la commande GROUP

La syntaxe de la commande est la suivante :

GROUP Nom_du_Groupe

{ <propriétés à inclure dans le groupe>

}

ou

GROUP "Nom du groupe"

{ <propriétés à inclure dans le groupe>

}

Comme illustré ci-dessus, la commande GROUP nécessite que les propriétés du groupe soient spécifiées à l'intérieur d'accolades ouvrantes et fermantes, { }. Si le nom du groupe contient des espaces, vous devez le placer entre guillemets. Vous pouvez également ne spécifier aucun nom pour le groupe en entrant des guillemets ouvrants et fermants, sans rien à l'intérieur. Par exemple :

GROUP ""

{ <propriétés à inclure dans le groupe>

}

Remarque importante : Tous les noms de groupe d'une spécification de diagramme, symbole ou définition doivent être uniques, même s'ils se trouvent dans des chapitres différent. Ainsi, par exemple, si vous créez un groupe "x" dans un chapitre d'une définition et un deuxième groupe "x" contenant des propriétés différentes, toutes les propriétés des deux groupes "x" seront placées dans un groupe "x" qui sera placé dans le chapitre du premier groupe. Si vous souhaitez qu'une même spécification contienne plusieurs groupes de même nom, ajoutez un espace aux noms des occurrences suivantes du groupe. Par exemple, pour l'exemple ci-dessus, GROUP "x ".

Utilisation de la commande LAYOUT dans une commande GROUP

Vous pouvez spécifier une commande LAYOUT dans une commande GROUP. Dans ce cas, elle remplace la commande LAYOUT de la définition ou du chapitre (page) dans lequel le groupe se trouve, uniquement pour les propriétés du groupe.

Exemple

Le code du fichier USRPROPS.TXT suivant permet de créer les groupes représentés dans la figure ci-après.

```
RENAME DEFINTION "Utilisateur 4" To “Ma définition de groupe"

DEFINITION "Ma définition de groupe"
{
CHAPTER "Présentation en 4 col"
LAYOUT { COLS 4 ALIGN OVER TAB }
      GROUP "Groupe de deux colonnes"
      {
      LAYOUT { COLS 2 ALIGN OVER TAB }
      PROPERTY "G1"{ EDIT Boolean LENGTH 1 DEFAULT "F"}
      PROPERTY "G2"{EDIT Boolean LENGTH 1 DEFAULT "F"}
      PROPERTY "G3"{ EDIT Boolean LENGTH 1 DEFAULT "F"}
      PROPERTY "G4"{ EDIT Boolean LENGTH 1 DEFAULT "F"}
       }
      GROUP ""
       {
       LAYOUT { COLS 1 ALIGN OVER   TAB }
       PROPERTY "Propriété de groupe 5"{ EDIT Text Length 10}
       PROPERTY "Propriété de groupe 6"{ EDIT Text Length 10}
       }
}
```

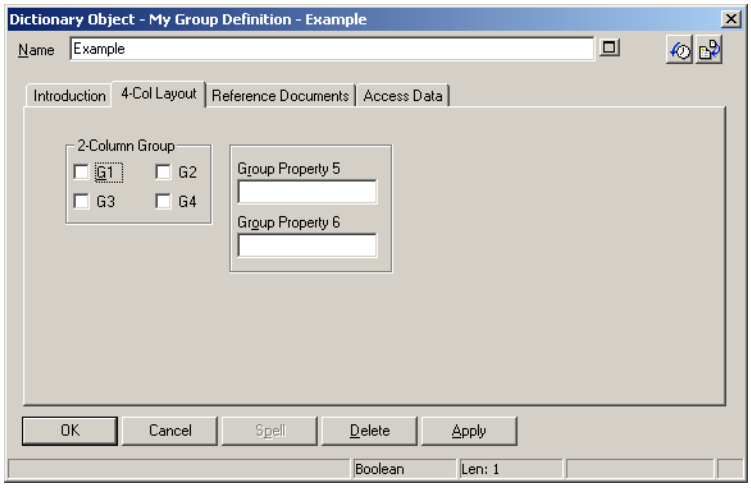


Figure 2-29.
Utilisation des commandes GROUP et LAYOUT.

Positionnement des contrôles et des libellés

La syntaxe d'un placement exact est la suivante :

```
PLACEMENT { PROPPOS(n,n)
            PROPSIZE(n,n) }
```

Exemple 1 :

```
PLACEMENT { PROPPOS(4,12)
            PROPSIZE(150,40) }
```

En outre, vous pouvez spécifier le positionnement des libellés. La syntaxe d'un placement exact de libellé est la suivante :

```
PLACEMENT { LABELPOS (n,n)
            PROPPOS(n,n)
            PROPSIZE(n,n) }
```

Exemple 2 :

```
PLACEMENT { LABELPOS (4,2)
            PROPPOS(4,12)
            PROPSIZE(150,40) }
```

Dans l'exemple 1 ci-dessus, PROPPOS (4,12) place le contrôle à 4 unités Windows dans le sens horizontal et à 12 unités Windows dans le sens vertical par rapport à l'angle supérieur gauche de la boîte de dialogue. Autrement dit, 4 unités Windows à gauche de l'angle supérieur gauche de la boîte de dialogue et 12 unités Windows au-dessous de cet angle. Avec PROPSIZE (150,40), la largeur du contrôle est égale à 150 unités Windows et sa longueur à 40 unités Windows.

Dans l'exemple 2 ci-dessus, LABELPOS place le libellé du contrôle à 4 unités Windows dans le sens horizontal et à 2 unités Windows dans le sens vertical par rapport à l'angle supérieur gauche de la boîte de dialogue. Le libellé se trouve donc à la même distance du côté de la boîte de dialogue, mais 10 unités Windows au-dessus du contrôle.

Remarque importante : Vous ne devez **pas mélanger** PLACEMENT avec les commandes de positionnement par défaut d'un CHAPITRE. Vous obtiendriez de curieux résultats de positionnement.

La syntaxe du fichier SAPROPS.CFG pour la boîte de dialogue de **définition de l'entité** est partiellement présentée ci-dessous.

```
CHAPTER "Déclencheurs et segment de table SQL Server"
GROUP "Déclencheurs d'intégrité référentielle par défaut"
        LABEL "Intégrité référentielle par défaut" {
        LAYOUT { COLS 1 ALIGN LABEL TAB }

PROPERTY "Nom du déclencheur INSERT"
        { EDIT Text LENGTH 31
        LABEL "Déclencheur INSERT"
        PLACEMENT {LABELPOS(4, 24)
            PROPPOS(50, 24) PROPSIZE(110, 12)} }

PROPERTY "Nom du déclencheur UPDATE"
        { EDIT Text LENGTH 31
        LABEL "Déclencheur UPDATE"
        PLACEMENT {LABELPOS(4, 38)
            PROPPOS(50, 38) PROPSIZE(110, 12)} }

PROPERTY "Nom du déclencheur DELETE"
        { EDIT Text LENGTH 31
        LABEL "Déclencheur DELETE"
        PLACEMENT {LABELPOS(4, 52)
            PROPPOS(50, 52) PROPSIZE(110, 12)} }
}
```

Quelques règles de dimensionnement générales

Dans la plupart des cas, vous devez modifier la longueur (coordonnée X) en fonction de vos besoins.

1. Les cases à cocher possèdent la valeur PROPSIZE (30, 12)
2. Les propriétés OneOf possèdent la valeur PROPSIZE (150, 40)
3. Les propriétés ListOf possèdent la valeur PROPSIZE (320, 98)

4. Les zones de texte courtes possèdent approximativement la valeur PROPSIZE (<LENGTH*3>, 12), en arrondissant l'abscisse de la largeur (x) pour des raisons esthétiques
5. Les zones de texte longues (par exemple, LENGTH 4074) possèdent une valeur d'environ PROPSIZE (150,115).

Quelques règles de positionnement générales

Là encore, vous devrez certainement apporter certaines modifications une fois que vous avez vu la manière dont la boîte de dialogue est disposée, mais ces nombres peuvent vous aider à commencer.

Tableau 2-4 Positionnement et taille lorsque la propriété se trouve dans un groupe et que le libellé est placé sur la propriété.

Type de PROPRIETE	PLACEMENT - Colonne de gauche
ListOf	PLACEMENT { PROPPOS (4, 24) PROPSIZE (320, 98) }
OneOf	PLACEMENT { PROPPOS (4, 24) PROPSIZE (150, 40) }
EDIT Text (moins de 75 caractères)	PLACEMENT { PROPPOS (4, 24) PROPSIZE (LENGTH * 3, 12) }
Type de PROPRIETE	PLACEMENT - Colonne de droite
ListOf	PLACEMENT { PROPPOS (165, 24) PROPSIZE (320, 98) }
OneOf	PLACEMENT { PROPPOS (165, 24) PROPSIZE (150, 40) }
EDIT Text (moins de 75 caractères)	PLACEMENT { PROPPOS (165, 24) PROPSIZE (LENGTH * 3, 12) }

Tableau 2-4. Positionnement et taille lorsque la propriété se trouve dans un groupe et que le libellé n'est pas placé sur la propriété.

Type de PROPRIETE	PLACEMENT - Colonne de gauche
ListOf	PLACEMENT { PROPPOS (4, 12) PROPSIZE (320, 98) }
OneOf	PLACEMENT { PROPPOS (4, 12) PROPSIZE (150, 40) }
EDIT Text **(moins de 75 caractères)**	PLACEMENT { PROPPOS (4, 12) PROPSIZE (LENGTH * 3, 12) }
Type de PROPRIETE	PLACEMENT - Colonne de droite
ListOf	PLACEMENT { PROPPOS (165, 12) PROPSIZE (320, 98) }
OneOf	PLACEMENT { PROPPOS (165, 12) PROPSIZE (150, 40) }
EDIT Text **(moins de 75 caractères)**	PLACEMENT { PROPPOS (165, 12) PROPSIZE (LENGTH * 3, 12) }

Tableau 2-4 Positionnement et taille lorsque la propriété ne se trouve pas dans un groupe et que le libellé n'est pas placé sur la propriété.

Type de PROPRIETE	PLACEMENT - Colonne de gauche
ListOf	PLACEMENT { PROPPOS (4, 2) PROPSIZE (320, 98) }
OneOf	PLACEMENT { PROPPOS (4, 2) PROPSIZE (150, 40) }
EDIT Text **(moins de 75 caractères)**	PLACEMENT { PROPPOS (4, 2) PROPSIZE (LENGTH * 3, 12) }
Type de PROPRIETE	PLACEMENT - Colonne de droite
ListOf	PLACEMENT { PROPPOS (165, 2) PROPSIZE (320, 98) }
OneOf	PLACEMENT { PROPPOS (165, 2) PROPSIZE (150, 40) }
EDIT Text **(moins de 75 caractères)**	PLACEMENT { PROPPOS (165, 2) PROPSIZE (LENGTH * 3, 12) }

Tableau 2-4 Positionnement et taille lorsque la propriété ne se trouve pas dans un groupe et que le libellé est placé sur la propriété.

Type de PROPRIETE	PLACEMENT - Colonne de gauche
ListOf	PLACEMENT { PROPPOS (4, 14) PROPSIZE (320, 98) }
OneOf	PLACEMENT { PROPPOS (4, 14) PROPSIZE (150, 40) }
EDIT Text **(moins de 75 caractères)**	PLACEMENT { PROPPOS (4, 14) PROPSIZE (LENGTH * 3, 12) }
Type de PROPRIETE	PLACEMENT - Colonne de droite
ListOf	PLACEMENT { PROPPOS (165, 14) PROPSIZE (320, 98) }
OneOf	PLACEMENT { PROPPOS (165, 14) PROPSIZE (150, 40) }
EDIT Text **(moins de 75 caractères)**	PLACEMENT { PROPPOS (165, 14) PROPSIZE (LENGTH * 3, 12) }

Tableau 2-4 Positionnement et taille lorsque les propriétés se trouvent sous les propriétés de la même boîte de dialogue.

PROPERTY **ajout au noeud final de la propriété précédente**	PLACEMENT - Colonne de gauche
A **+14 si le libellé est placé sur la propriété B**	PLACEMENT { PROPPOS (4, 14) PROPSIZE (150, 40) }
B **+ 4 si la propriété C est OneOf**	PROPERTY B PLACEMENT { PROPPOSE (4, 68) PROPSIZE (150, 40) }
C	PLACEMENT { PROPPOS (4, 112) PROPSIZE (150, 12) }
Type de PROPRIETE	PLACEMENT - Colonne de droite
D **+ 4 si le libellé de la propriété E se trouve sur le côté**	PLACEMENT { PROPPOS (165, 14) PROPSIZE (320, 98) }

E **+ 4 car LABELPOS est utilisé pour la propriété** **F**	PLACEMENT { PROPPOS (165, 116) PROPSIZE (150, 40) }
F	PLACEMENT { LABELPOS (165, 160) PROPPOS (165, 170) PROPSIZE (30, 12) }

Remarque : LABELPOS est facultatif. Cette commande permet de remplacer la commande LAYOUT pour le groupe. La coordonnée y est de 10 unités plus élevée que la coordonnée y de PROPPOS.

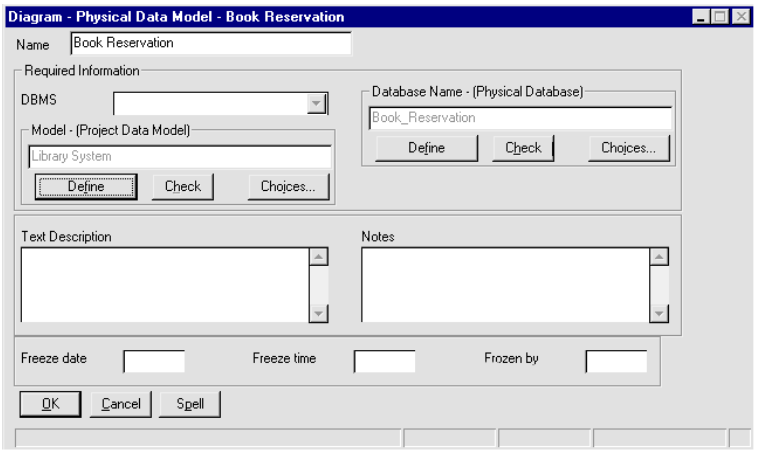


Figure 2-30. Résultat des commandes CHAPTER

Spécification de l'affichage des valeurs sur les symboles

Un symbole représente une définition dans le référentiel. Cette définition possède des propriétés. Vous pouvez spécifier que les propriétés de définition et leurs valeurs soient affichées sur un symbole. Par défaut, le nom d'un symbole (qui correspond à une propriété du symbole et sa définition) est affiché. Pour spécifier que les autres propriétés de la définition d'un symbole sont affichables, vous utilisez la commande **DISPLAY** dans la déclaration de chaque propriété. Par exemple :

```
Definition "Ma définition"
{
Property "Ma propriété 1" {EDIT TEXT LENGTH 20
DISPLAY { FORMAT String LEGEND "" } }
}
```

L'exemple de code ci-dessus rend la propriété "Ma propriété 1" affichable sur le symbole ; tout texte que vous saisissez dans la zone de texte de 20 caractères de la propriété, dans la boîte de dialogue de la définition, est affiché sur la face du symbole.

Une fois que vous avez spécifié certaines propriétés de la définition d'un symbole comme étant affichables, ces propriétés sont fournies dans la boîte de dialogue **Mode Affichage** d'un symbole, où elles peuvent être activées ou désactivées à tout moment. Vous accédez à la boîte de dialogue Mode Affichage en sélectionnant un symbole sur un diagramme, puis en choisissant **Vue, Mode Affichage** ou en cliquant sur un symbole à l'aide du bouton droit de la souris et en choisissant **Mode Affichage**. La boîte de dialogue Mode Affichage permet de sélectionner toutes les propriétés affichables que vous souhaitez afficher pour le symbole.

La figure ci-après illustre un symbole d'entité : la figure Avant le montre avec toutes les propriétés affichables désactivées, tandis que la figure Après montre les données de clé et les données autres que les données de clé activées.

Figure 2-31. Symbole rectangulaire avec les propriétés affichées et non affichées

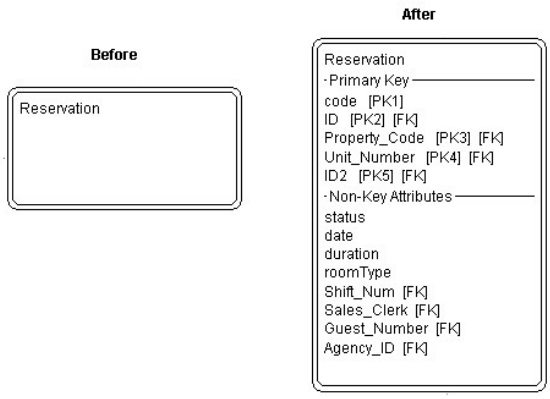
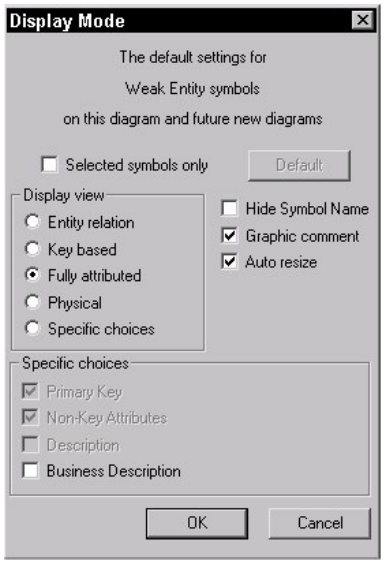


Figure 2-32. Exemple de boîte de dialogue Mode Affichage.



Vous pouvez également spécifier les propriétés affichables des symboles de ligne, comme illustré dans la figure ci-après.

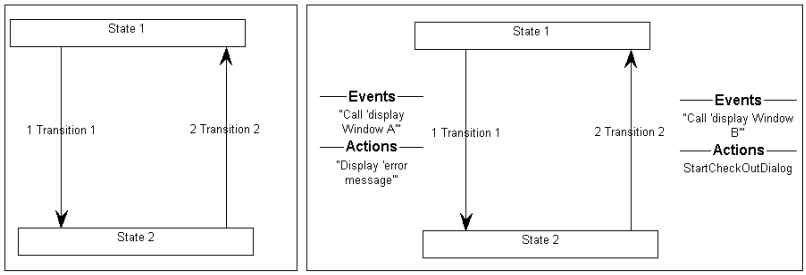


Figure 2-33. Symbole de ligne avec les propriétés affichées et non affichées

Syntaxe de la commande DISPLAY

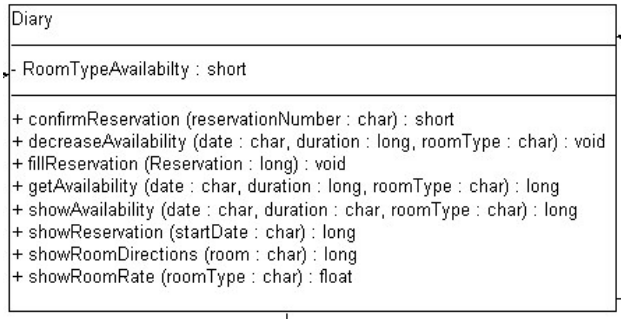
Il existe une limite de **huit** instructions d'affichage par définition. La syntaxe de la commande DISPLAY est la suivante :

DISPLAY { FORMAT [STRING | LIST | KEY | NONKEY | COMPONENT_SCRIPT | COLUMN_SCRIPT | SCRIPT]
LEGEND " (comment le bloc est libellé dans le symbole) **" }**

- STRING : Valeur par défaut. Les valeurs de la propriété apparaissent sur le symbole comme elles ont été saisies. Ce choix convient si vous souhaitez afficher les commentaires.
- LIST : Les éléments sont affichés sur le symbole dans une liste ; chaque espace génère une nouvelle ligne, sauf s'il est placé entre guillemets.
- KEY : Ce mot-clé est utilisé pour les propriétés désignées comme clés. Elles sont affichées dans une section distincte du symbole. Pour consulter un exemple, reportez-vous au mot-clé KEY du Chapitre 3.
- NONKEY : vous pouvez utiliser ce mot-clé pour des propriétés autres que des clés. Elles seront affichées dans une section distincte du symbole. Ce mot-clé était à l'origine utilisé pour les entités et les tables dans le support de modélisation des données de Rational System Architect. Pour consulter un exemple, reportez-vous au mot-clé NONKEY du Chapitre 3.

- COMPONENT_SCRIPT : appelle un script qui affiche la valeur de la propriété sur le symbole, dans un format spécial, conçu par le script. Le script lui-même est soit codé en dur dans le produit, soit écrit par l'utilisateur à l'aide du langage Rational System Architect Basic. Par convention, le script est lui-même nommé avec l'un des préfixes suivants :
- fmtxxx : la fonction est codée en dur et ne peut pas être modifiée. C'est le cas de la plupart des fonctions contenues dans SAPROPS.CFG. Le codage en dur de la fonction est effectué pour accélérer la réponse globale de Rational System Architect.
- _fmtxxx : se trouve dans le fichier fmtscript.bas dans le répertoire principal des exécutables de Rational System Architect et est codé à l'aide de SA Basic.

Les scripts de composant sont utilisés pour les propriétés ListOf et ExpressionOf. L'action effectuée par le script fonctionne sur chaque élément de la liste. Par exemple, un script de composant est utilisé pour consulter chaque attribut de classe d'une définition de classe et générer la manière dont il sera affiché sur le symbole de classe, en spécifiant la marque ‘-’ avant le nom si la propriété d'accès de l'attribut a la valeur "privé" ou la marque ‘+’ si sa valeur est "public" et en affichant le type de retour de l'attribut après le nom de l'attribut, précédé d'un point-virgule. De même, un script de composant génère la manière dont une méthode est affichée, avec une marque ‘+’ précédant le nom si son accès est public et une marque ‘-‘, si son accès est privé et affiche ses paramètres et leur type entre parenthèses après le nom de la méthode.



Pour plus d'informations, voir COMPONENT_SCRIPT et SCRIPT dans le Chapitre 3.

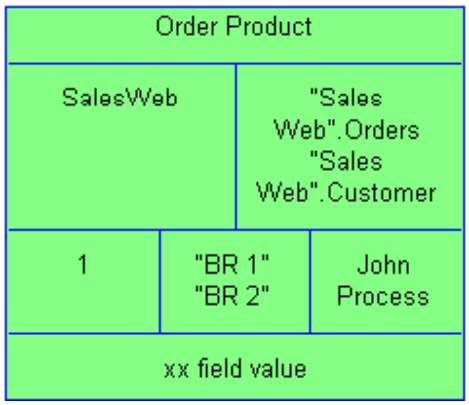
- COLUMN_SCRIPT : fonctionne comme COMPONENT_SCRIPT, en appelant un script pour appliquer un formatage spécial à une valeur de propriété affichée sur un symbole. Le script est codé en dur ou écrit par l'utilisateur à l'aide de SA Basic (et placé dans le fichier fmtscript.bas, dans le répertoire principal du programme exécutable de Rational System Architect). Les scripts de colonne sont utilisés pour l'affichage des colonnes dans les symboles de table d'un modèle de données physique. L'action effectuée par le script fonctionne sur chaque colonne de la liste. Pour plus d'informations, voir COLUMN_SCRIPT, dans le Chapitre 3.

- SCRIPT : fonctionne comme COLUMN_SCRIPT et COMPONENT_SCRIPT, en appelant un script pour appliquer un formatage spécial à une valeur de propriété affichée sur un symbole. La commande SCRIPT appelle des scripts utilisés pour les propriétés qui ne sont ni ListOf, ni ExpressionOf. Le script est lui-même codé en dur ou écrit par l'utilisateur à l'aide de SA Basic (et placé dans le fichier fmtscript.bas, dans le répertoire principal du programme exécutable de Rational System Architect). Pour plus d'informations, reportez-vous au mot clé SCRIPT, dans le Chapitre 3.

Chaque groupe de propriétés affiché est séparé des autres par une ligne. Vous pouvez spécifier un libellé ou une "légende" qui apparaîtra sur cette ligne, à l'aide de la commande LEGEND. Si aucune LEGENDE n'est fournie, le nom de la propriété sert de libellé. Les commandes LEGEND suivantes sont disponibles :

- **LEGEND "<Votre texte>" :** Le texte que vous placez entre guillemets ne sera affiché sur le symbole au dessus de l'entrée, que si cette dernière possède une valeur.
- **LEGEND "" :** N'affiche une ligne droite sans texte, que si l'entrée possède une valeur.
- **LEGEND "$$FORCE$$" :**Affiche une ligne horizontale au dessus de l'entrée sur le symbole. Cette ligne sert de diviseur. Le mot-clé "$$FORCE$$" est différent de la chaîne " " car il force l'affichage d'une ligne horizontale même si l'affichage de la propriété est supprimé via la boîte de dialogue du mode d'affichage.
- **LEGEND "$$NONE$$" :** N'affiche pas de ligne horizontale au dessus de l'entrée sur le symbole, que l'entrée possède une valeur ou non. Cette ligne sert normalement de diviseur.

- **LEGEND "$$VFORCE$$" :** Permet de disposer les propriétés de gauche à droite à l'intérieur des symboles et trace des lignes verticales entre elles. Un exemple est illustré ci-dessous :



  Pour l'exemple de fichier USRPROPS.TXT qui crée l'illustration ci-dessus, reportez-vous au mot clé VFORCE dans le Chapitre 3.

- **LEGEND "$$VNONE$$" :** Permet de disposer les propriétés de gauche à droite, mais *ne fournit pas* de ligne de division. Pour consulter un exemple, reportez-vous au mot-clé VNONE du Chapitre 3.

La famille de polices et la police des légendes affichables sont contrôlées via la commande **Format du diagramme**, **Notation** du menu **Format**.

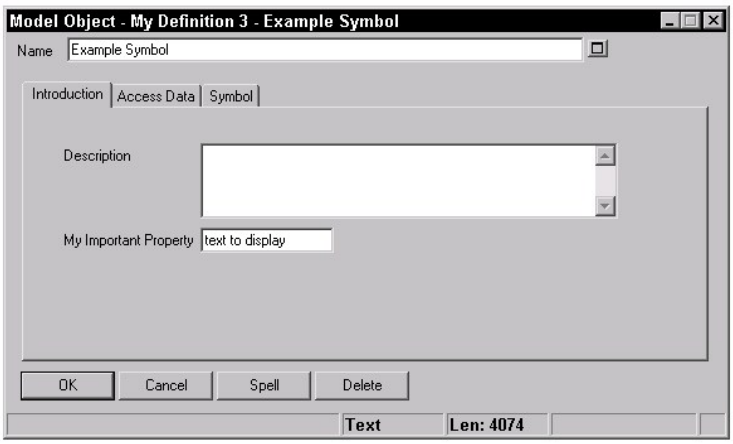
Exemple

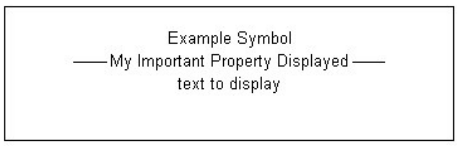
Dans l'exemple ci-après, nous spécifions un nouveau type de diagramme, un nouveau type de symbole et un nouveau type de définition. Nous spécifions que le nouveau type de symbole est défini par le nouveau type de définition et affectons le nouveau type de symbole au nouveau type de diagramme. Nous créons une propriété pour la définition, intitulée "Ma propriété importante". Nous spécifions que la légende "Ma propriété importante affichée" doit être affichée sur le symbole, sur la ligne séparatrice au dessus de la valeur affichée.

```
RENAME DIAGRAM "Utilisateur 1" TO "Mon diagramme"
RENAME SYMBOL "Utilisateur 1" TO "Mon symbole 1"

SYMBOL "Mon symbole 1"
{
DEFINED BY "Ma définition 3"
ASSIGN TO "Mon diagramme"
}
DEFINITION "Ma définition 3"
{
PROPERTY "Ma propriété importante" { EDIT Text Length
20 DISPLAY { FORMAT String LEGEND "Ma propriété
importante affichée" }}
}
```

La figure ci-après illustre un symbole suivant représenté sur un tel diagramme et la valeur affichée saisie dans la zone de propriété.

Figure 2-34. Symbole de ligne avec les propriétés affichées et non affichées





Spécification des propriétés Key et Keyed By

Etablissement des propriétés KEY

Vous pouvez spécifier qu'une définition particulière est "indexée" sur une ou plusieurs définitions. Une clé détermine l'espace de nom d'une définition dans l'encyclopédie. Par exemple, un type de définition d'attribut de classe est indexée par sa définition de classe conteneur et la définition de module conteneur de cette classe. Vous pouvez donc avoir deux attributs appelés nom, l'un appartenant à la classe Client du package Système_Réservation et l'autre à la classe Produit du package Système_commande. Les deux attributs, bien qu'ils possèdent le même nom, représentent des définitions distinctes.
Par défaut, chaque élément de modélisation d'une encyclopédie est déjà secrètement indexé à trois éléments : sa *classe* (ici *classe* est utilisé selon la définition de Rational System Architect, qui fait la différence entre un diagramme, un symbole et une définition), son *type* (diagramme Cas d'utilisation UML, diagramme Processus BPMN, etc) et son *nom* (par exemple, diagramme Cas d'utilisation Système_Réservation versus diagramme Cas d'utilisation Système_Ressources_Humaines).

Vous pouvez ajouter des propriétés de clé à une définition à l'aide de la commande KEY. Vous spécifiez la commande KEY dans la propriété qui doit être une clé d'une définition. La commande KEY peut être placée pratiquement n'importe où dans la description d'une propriété, mais en raison de son importance, elle est généralement placée comme premier élément, à l'intérieur des accolades de la propriété, juste avant le mot-clé EDIT.

Remarque : il n'est pas possible d'ajouter un mot-clé KEY EDIT ONEOF à un diagramme.

Exemple 1 :

```
Definition "Cas d'utilisation"
{
PROPERTY "Package" { KEY EDIT …}
..}
```

Exemple 2 :

Definition "Etape de cas d'utilisation"
{
PROPERTY "Nom du cas d'utilisation" { **KEY** EDIT … }
PROPERTY "Package" { **KEY** EDIT …}
…

Remarque : Les propriétés de clé d'une définition ne sont pas affichées dans une grille constituée par une commande ASGRID. Par exemple, dans une définition Cas d'utilisation, les étapes du cas d'utilisation sont représentées dans une grille constituée par une commande ASGRID, mais les propriétés de clé des étapes du cas d'utilisation (module propriétaire et cas d'utilisation) ne sont pas affichées dans la grille des étapes du cas d'utilisation.

Pour une propriété qui correspond à une clé et qui fait référence à d'autres objets (par exemple, une propriété LISTOF ou ONEOF et non une simple propriété TEXT ou NUMERIC), l'utilisateur final doit spécifier la classe et le type de classe des objets référencés lorsqu'il saisit une valeur pour la propriété, dans Rational System Architect.

Par exemple :

Définition “Processus métier”
{
PROPERTY "Cas d'utilisation du système" {EDIT **ONEOF "Cas d'utilisation"** …}

L'instruction ci-dessus indique que la propriété "Nom du cas d'utilisation" fait référence à une définition de type "Cas d'utilisation". Définition est la valeur par défaut si aucune *classe* n'est spécifiée (*classe* au sens de Rational System Architect : Diagramme, Symbole ou Définition.)

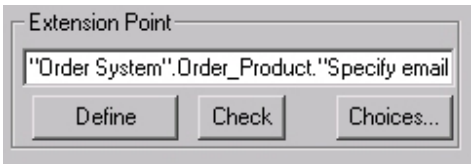
La valeur de la propriété elle-même contient souvent tous les éléments restants nécessaires pour identifier le ou les objets référencés. Si le type/la classe référencé de la propriété ne contient pas de propriétés de clé, la valeur de référence correspond simplement au nom de l'objet (car la classe et le

type sont connus), mais si le type/la classe référencé contient des propriétés de clé (telles que "Cas d'utilisation" dans l'exemple ci-dessus, qui contient la propriété de clé "module"), Rational System Architect doit connaître les valeurs de ces propriétés pour identifier correctement l'objet de référence.

Remarque : Les propriétés de référence hétérogènes sont en ce sens différentes. Voir HETEROGENEOUS dans le Chapitre 3.

Codez cela dans le fichier USRPROPS.TXT pour que Rational System Architect obtienne automatiquement les valeurs de l'utilisateur final ou forcez ce dernier à saisir le nom qualifié complet, en séparant les composants de clé par des points.

- Pour que Rational System Architect extraie automatiquement la valeur des utilisateurs, utilisez la commande KEYED BY.
- Si aucune clause KEYED BY n'est spécifiée pour la propriété, Rational System Architect s'attend à ce que ces valeurs de clé supplémentaires soient spécifiées dans la référence même. En d'autres termes, l'utilisateur doit saisir le nom qualifié complet de l'objet de référence, avec des points séparant les valeurs de clé. Pour une étape de cas d'utilisation appelée "*Spécifier e-mail"* dans le cas d'utilisation *Commander_Produit* d'un module appelé "*Système de commande"*, l'utilisateur doit saisir "*Système de commande".Commander_Produit."Spécifier e-mail"*.



Remarque : Si un composant contient un caractère important du point de vue de la syntaxe (comme un espace ou un point), il doit être placé entre guillemets pour que Rational System Architect puisse analyser correctement la référence.

Voici deux exemples de référence à une "Etape de cas d'utilisation" :

Système_commande.Commander_Produit."Spécifier e-mail"

où *Corriger_Facture* représente le nom du cas d'utilisation qui appartient au package *Comptes_Fournisseurs*.

"Système de commande".Commander_Produit."Spécifier e-mail

où *Commander_Produit* représente le nom du cas d'utilisation qui appartient au package *Système de commande*.

Utilisation de KEYED BY pour s'assurer que tous les membres d'un groupe sont de même type

La clause KEYED BY permet également de générer la liste des choses associées. Par exemple, toutes les étapes de cas d'utilisation auxquelles il est fait référence dans la propriété "Etapes de cas d'utilisation" d'une définition Cas d'utilisation appartiennent au même cas d'utilisation, à savoir, celui contenant la propriété "Etapes de cas d'utilisation". Lorsqu'une propriété à plusieurs références (telle que ListOf) fait référence à des objets appartenant tous au même objet parent, il est conseillé d'utiliser une ou plusieurs autres propriétés pour identifier l'objet parent. Dans ces cas, une clause KEYED BY est utilisée pour indiquer à Rational System Architect les autres propriétés à utiliser.

Comment utiliser la clause KEYED BY

Donc, pour résumer, la clause KEYED BY est éventuellement utilisée pour spécifier comment les composants clé d'un ou plusieurs objets référencés peuvent être trouvés. Elle offre deux principaux avantages :

1. L'utilisateur final n'a plus besoin de saisir le nom complet d'une valeur de référence (avec des points séparant les qualificateurs). Par exemple, pour une propriété qui fait référence à l'attribut de classe ***e-mail*** de la classe ***Client*** du module ***"Système de commande"***, au lieu de saisir ***"Système de commande".Client.email***, l'utilisateur final saisit simplement ***e-mail***.

2. Elle permet de s'assurer que tous les composants clé d'une valeur de référence sont identiques. Par exemple, La propriété LISTOF "Attribut de classe" d'une définition Classe contient une liste des attributs qui appartiennent à la même classe et au même module.

Une clause KEYED BY contient généralement une spécification de la manière dont chacun des composants clé des objets référencés peut être trouvé. La clause KEYED BY contient une portion pour chaque composant clé, séparée par une virgule.

Exemple :

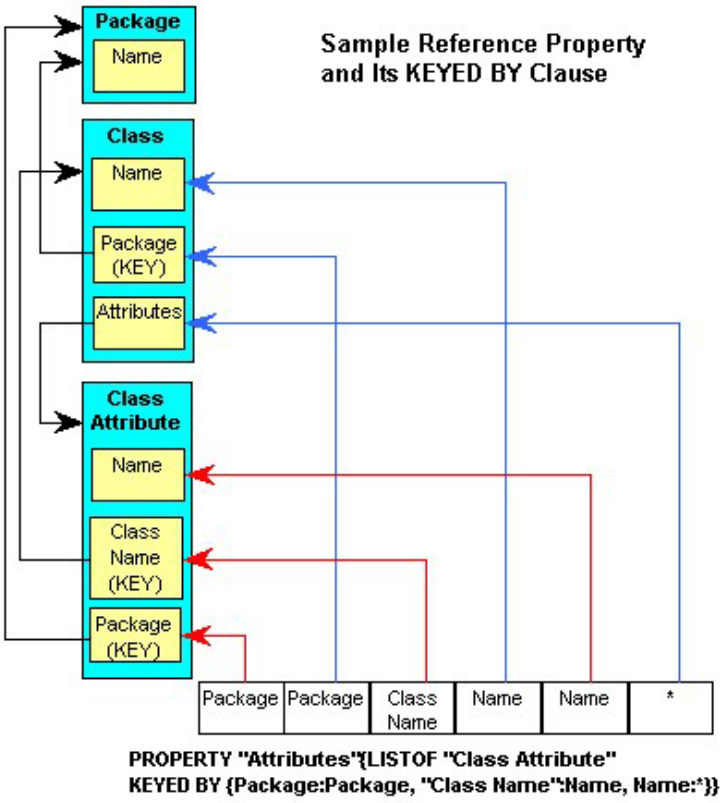
Par exemple, la clause KEYED BY de la propriété "Attributs" de la classe peut se présenter comme suit :

```
DEFINITION "Classe"
{
…
PROPERTY "Attributs" { … LISTOF "Attribut de classe"
KEYED BY {Package:Package, "Nom de la classe":Name,
Nom:* } … }
```

Dans l'exemple ci-dessus, les trois *composants de clé* (séparés par des virgules) sont Package:Package, "Nom de la classe:Name et Nom:*. Ces composants font référence aux trois parties requises pour identifier les définitions Attribut de classe référencées : le nom de module, le nom de classe et le nom de classe d'attribut. En sens inverse, cette clause stipule que :

- Le nom de l'attribut de classe se trouvera dans ***cette*** propriété (* signifie "ici"), donc :
  Nom:*
- La valeur de la propriété de clé "Nom de la classe" dans la définition Attribut de classe se trouvera dans le nom de cet objet, donc :
  "Nom de la classe":Name
- La valeur de la propriété de clé Module dans la définition Attribut de classe se trouvera dans la propriété Module de cet objet, donc :
  Package:Package

Le schéma suivant illustre comment la clause KEYED BY est utilisée dans l'exemple ci-dessus et peut permettre de comprendre la clause KEYED BY en général.



Le schéma illustre ce que nous avons abordé plus haut : dans la définition d'une classe, un attribut de classe est entré en spécifiant son module (stocké dans la propriété Module de l'attribut de classe et obtenu à partir de la valeur Module de la classe dans laquelle vous vous trouvez), son nom de classe (stocké dans la propriété "Nom de la classe" de l'attribut de classe et obtenu à partir du nom réel de la classe) et son nom (stocké dans la propriété "Nom" de l'attribut de classe et provenant de la classe).

En résumé :

1. Pour chaque composant clé de l'**objet de référence**, la clause KEYED BY possède un composant.
2. Les composants de la clause KEYED BY sont séparés par des virgules.
3. Chaque composant est constitué de deux parties :
   - La première partir identifie le composant clé de l'objet de référence.
   - La seconde partie indique où se trouve la valeur de ce composant.
   - Les deux parties sont séparées par un signe deux-points.

Toutefois, certaines valeurs par défaut peuvent être déduites pour simplifier la clause KEYED BY. Si les deux parties du composant sont identiques, la seconde peut être omise et si la seconde partie du dernier composant est omise, il est supposé que sa valeur est "ici" (astérisque). Ainsi, dans la pratique, la clause KEYED BY de la propriété "Attributs" de la classe est codée :

KEYED BY {Package**,** "Nom de la classe"**:**Name**,** Nom }

Naturellement, toutes les propriétés utilisées dans l'instruction KEYED BY doivent exister. Ainsi, Rational System Architect vérifie qu'il existe une propriété "Package" et une propriété "Nom de la classe" dans la définition "Attribut de classe" et qu'il s'agit de propriétés KEY. De la sorte, l'utilisateur n'a plus besoin de saisir les composants de clé communs dans une propriété LISTOF telle que celle-ci (par exemple, "Système de commande".Client.e-mail), mais en plus, l'utilisation d'une clause KEYED BY avec d'autres propriétés afin de fournir des valeurs communes **permet de garantir que les mêmes valeurs sont utilisées pour chaque référence**. Ainsi, dans notre exemple, tous les attributs de classe auxquels il est fait référence dans la propriété "Attributs" de la classe appartiennent obligatoirement à la même classe du même module, ce qui est souhaitable dans ce cas. Dans d'autres cas, il peut être utile que les composants de clé de l'objet référencé soient séparés à des fins de clarté et de simplicité. Dans de tels cas, une clause KEYED BY est utilisée pour désigner les propriétés qui fournissent les divers composants. En effet, pour ces raisons, lorsqu'une propriété est une propriété KEY et qu'elle fait référence à un objet avec des propriétés KEY, Rational System Architect **requiert** que les composants soient placés dans des propriétés distinctes.

Exemples de clause Key et Keyed By

Une définition indexée par une autre

Nous souhaitons classer toutes les automobiles par "Marque" et "Modèle". Nous créons une définition appelée "Marque de voiture" (une marque de voiture peut être Ford, Volkswagen, Toyota, etc) et une autre appelée "Modèle de voiture" (qui peut inclure des valeurs telles que Mustang, Passat et Corolla).

La "Marque de voiture" (ou "Gamme") d'un "Modèle de voiture" doit être spécifiée. La "Marque de voiture" est une propriété que vous pouvez utiliser pour identifier de manière unique le "Modèle de voiture" ; chaque "Modèle de voiture" ne correspond qu'à une seule "Marque de voiture".

Nous créons une propriété de clé "Gamme" d'un "Modèle de voiture", mais en réalité, le type de définition indiqué pour cette propriété est "Marque de voiture". Nous lui affectons également le mot clé REQUIRED, ce qui signifie que pour créer la définition, vous devez renseignez cette propriété dans la boîte de dialogue qui s'affiche pour créer la définition.

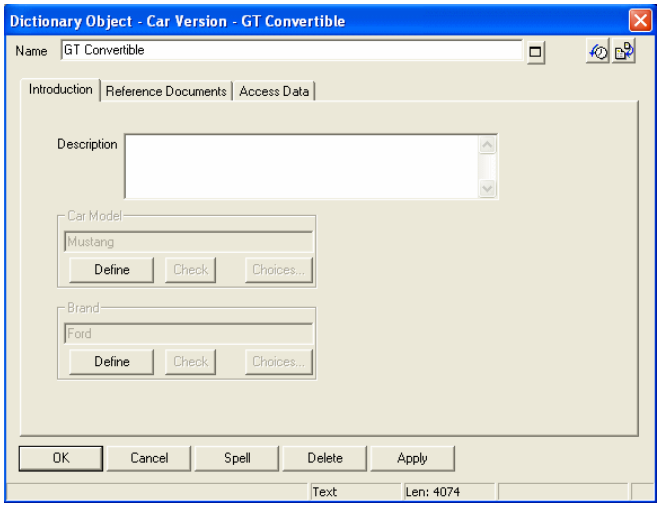
```
RENAME DEFINITION "Utilisateur 1" To "Marque de
voiture"
RENAME DEFINITION "Utilisateur 2" To "Modèle de
voiture"

DEFINITION "Marque de voiture"
{
PROPERTY "Pays d'origine"
{ EDIT  Text  LENGTH 20  }
}

DEFINITION "Modèle de voiture"
{
Property "Gamme"
{KEY EDIT ONEOF "Marque de voiture" REQUIRED}
}
```

Remarque : Nous avons introduit un problème dans le fichier USRPROPS.TXT ci-dessus pour en discuter. Nous le découvrirons plus loin dans cette section.

Une troisième définition avec deux propriétés de clé

Poussons le raisonnement un peu plus loin. Chaque "Modèle de voiture" possède une version. Par exemple, vous pouvez acheter un coupé Mustang, une décapotable, un coupé GT, une décapotable GT, une Mach 1 ou une SVT Cobra. Cette liste pouvant changer en permanence, nous créons un type de définition (au lieu d'une commande LIST statique). Nous appelons ce type de définition "Version de voiture". Lorsque l'utilisateur crée une "Version de voiture", il doit spécifier la "Marque de voiture" et le "Modèle de voiture" de la version (car il peut exister de nombreux modèles de voiture GT).

```
RENAME DEFINITION "Utilisateur 3" TO "Version de voiture"
Definition "Version de voiture"
{
Property "Modèle de voiture"
{KEY Edit ONEOF "Modèle de voiture" RELATE BY "indexé par "}
Property "Marque"
{KEY EDIT ONEOF "Marque de voiture" RELATE BY "indexé par"}
}
```



Création d'une propriété LIST OF de définition indexée

Pour chaque "Modèle de voiture", nous souhaitons créer une liste de versions de voiture qu'il offre. Nous créons une propriété LIST OF qui permet à l'utilisateur de saisir les versions de voiture dans la définition "Modèle de voiture". Notez que Versions de voiture est un type de définition avec

une clé composée ; lorsque l'utilisateur saisit la "Version de voiture", il doit spécifier la "Version de voiture", ainsi que la "Marque de voiture" et le "Modèle de voiture" de la "Version de voiture".

```
DEFINITION "Modèle de voiture"
{
Property "Gamme"
{KEY EDIT ONEOF "Marque de voiture" RELATE BY "indexé par" REQUIRED}
Property "Versions" {EDIT LISTOF "Version de voiture"}
}
```

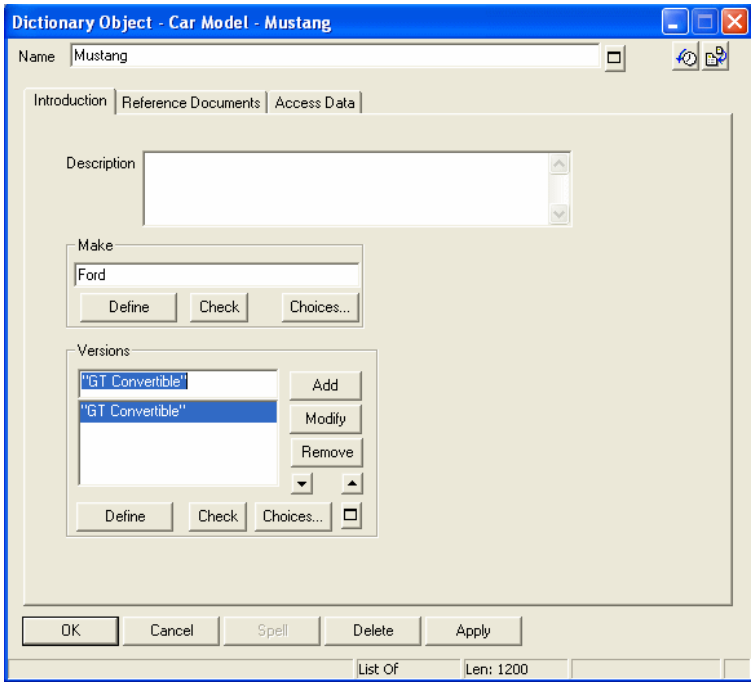
Il existe un **problème** dans le fichier USRPROPS.TXT ci-dessus. "Version de voiture" est une définition de clé composée ; cette propriété possède son propre nom comme clé et deux autres propriétés de clé, "Marque de voiture" et "Modèle de voiture". Si nous spécifions le fichier USPROPS.TXT ci-dessus, l'utilisateur devra savoir cela. Il devra saisir la "Version de voiture", ainsi que la marque et le modèle, avec un point séparant chacun. Par exemple : Ford.Mustang."Décapotable GT".

Ajout de l'instruction KEYED BY

Nous ajoutons l'instruction KEYED BY à l'instruction "Version" de la propriété pour spécifier automatiquement cela.

```
DEFINITION "Modèle de voiture"
{
Property "Gamme"
{KEY EDIT ONEOF "Marque de voiture" RELATE BY "indexé par" REQUIRED}
Property "Versions" {EDIT LISTOF "Version de voiture" KEYED BY {"Marque":"Gamme", "Modèle de voiture":Nom, Nom} }
}
```

Dans la première partie de l'instruction KEYED BY ci-dessus, **KEYED BY {"Marque":"Gamme"**, nous indiquons que la valeur "Version de voiture" que nous entrons contient une propriété de clé appelée Marque qui doit être renseignée. Cette propriété sera renseignée avec une valeur extraite de la propriété "Gamme" de la définition actuelle dans laquelle nous nous trouvons : "Modèle de voiture". Notez que la valeur réelle est une définition de type "Marque de voiture" ; l'instruction KEYED BY répertorie les noms des propriétés et non des types de définition.



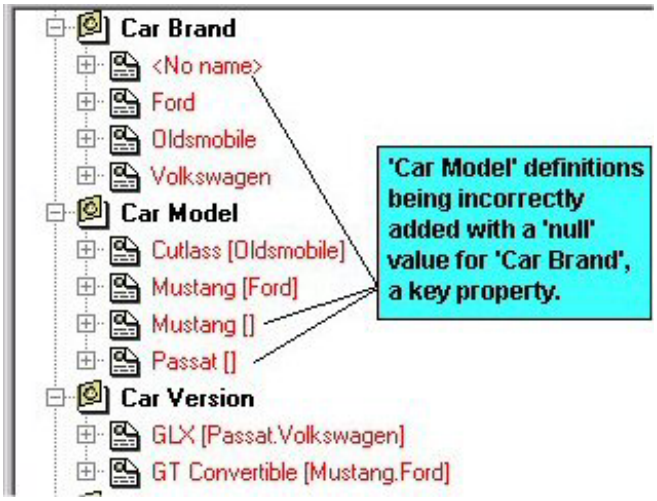
Si la propriété référencée ("Marque") et la propriété de clé référençante ("Gamme") ci-dessus étaient identiques, nous n'aurions pas besoin de spécifier les deux (si elles étaient toutes deux "Marque" nous n'aurions besoin de saisir que KEYED BY **{"Marque",** ….

Dans la deuxième partie de l'instruction KEYED BY ci-dessus, **"Modèle de voiture":Nom**, nous indiquons que la valeur "Version de voiture" que nous saisissons possède une deuxième propriété de clé appelée "Modèle de voiture", qui doit être renseignée. Cette propriété sera renseignée avec une valeur extraite du nom de la définition actuelle dans laquelle nous nous trouvons ; le nom de la définition "Modèle de voiture" que nous avons ouverte.

Dans la troisième et dernière partie de l'instruction KEYED BY ci-dessus, Nom, vous spécifions que la dernière clé de la valeur "Version de la voiture" que nous entrons est indexée par son propre nom, comme toute définition.

Par conséquent, si nous disposons d'une définition "Modèle de voiture" ouverte appelée Mustang, dont la propriété de clé "Gamme" est Ford, nous n'avons qu'à saisir GT dans la propriété LIST OF "Version de voiture" et la nouvelle définition Ford.Mustang.GT sera ajoutée à l'encyclopédie.

Il existe toujours un problème. Nous constatons lorsque nous ajoutons de nouvelles définitions "Version de voiture" dans l'encyclopédie, que des définitions "Modèle de voiture" pour lequelles "Marque de voiture" possède une propriété null sont entrées.



Cela ne se produit que si nous ajoutons une nouvelle définition "Version de voiture" directement (via la commande Nouvelle définition dans l'explorateur) et non si nous en ajoutons une dans la boîte de dialogue ListOf de la définition "Modèle de voiture".

En effet, la définition "Version de voiture" spécifie que l'une de ses propriétés de clé est "Modèle de voiture", mais ne spécifie pas que cette propriété possède sa propre propriété de clé ("Marque de voiture") à renseigner. Chaque fois que nous ajoutons une nouvelle définition "Version de voiture", le système nous demande de spécifier une propriété "Modèle de voiture". Nous spécifions cette propriété "Modèle de

voiture", mais n'indiquons pas d'où elle obtient la propriété "Marque de voiture" de sa clé. Aucune valeur n'est donc spécifiée.

pour corriger cela, nous devons spécifier dans la définition "Version de voiture" que sa propriété de clé "Modèle de voiture" est elle-même indexée par des propriétés, qui doivent être renseignées par des valeurs. Nous ajoutons la clause **KEYED BY {"Gamme":"Marque", Nom**}, qui précise que la définition "Modèle de voiture" contient la propriété "Gamme" qui doit être renseignée avec la valeur qui se trouve dans la propriété "Marque" de la définition actuelle.

```
Definition "Version de voiture"
{
Property "Modèle de voiture"
{Key Edit oneOf "Modèle de voiture" KEYED BY
{"Gamme":"Marque", Nom} RELATE BY "indexé par"}
Property "Marque"
{KEY EDIT OneOf "Marque de voiture" RELATE BY
"indexé par"}
}
```

Une fois que nous avons apporté cette modification, lorsque nous ajoutons une nouvelle "Version de voiture" à l'encyclopédie, nous n'obtenons pas de définitions "Marque de voiture" null par inadvertance.

Exemple de mot clé COMPLETE

Le fait de permettre aux utilisateurs de saisir de nouvelles définitions de version de voiture indépendamment du "Modèle de voiture" reste discutable. Par exemple, un utilisateur entrera-t-il Si comme "Version de voiture", avant de spécifier qu'il fait référence à une Honda Accord ? Probablement pas. Vous pouvez certainement aider les utilisateurs en les forçant à entrer la "Marque de voiture" et le "Modèle de voiture", puis les versions de voiture dans la propriété ListOf des versions de voiture de la définition "Modèle de voiture". Vous indiquez ainsi que ces versions de voiture appartiennent entièrement au "Modèle de voiture" ; le fait de préciser que vous possédez une Si ne signifie pas grand chose en soi. Nous utilisons pour cela la clause COMPLETE.

```
DEFINITION "Modèle de voiture"
{
```

```
Property "Gamme"
{KEY EDIT ONEOF "Marque de voiture" RELATE BY
"indexé par" REQUIRED}
REM "contient également une liste de versions"
Property "Versions"
{EDIT COMPLETE LISTOF "Version de voiture" KEYED
BY {"Marque":"Gamme", "Modèle de voiture":Nom, Nom} }
}
```

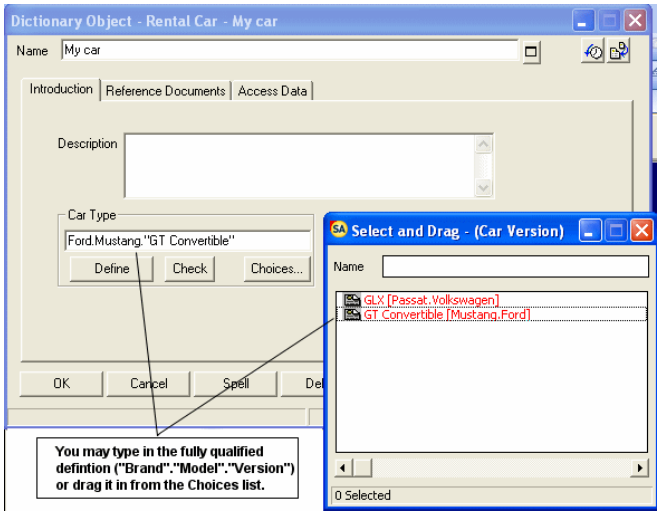
Une fois que vous avez défini la propriété COMPLETE pour une liste, la définition "Version de voiture" n'apparaît plus parmi les types de définition disponibles lorsque vous créez une définition et vous ne pouvez plus ouvrir de définition "Version de voiture" à partir de l'explorateur. Si vous essayez, Rational System Architect génère un message indiquant que vous ne pouvez ouvrir la définition qu'à partir de la définition qui l'a contient (ici, "Modèle de voiture").

Exempe qualifiable

Nous souhaitons maintenant ajouter une nouvelle définition à l'encyclopédie pour effectuer le suivi des locations de voiture. Le nouveau type de définition Location de voiture inclut une propriété permettant de rechercher le Type de voiture de chaque location.

Toutefois, nous ne souhaitons pas garder une trace de la "Marque de voiture" et du "Modèle de voiture" en plus de la "Version de voiture" pour une location de voiture. Nous souhaitons une propriété, Type de voiture, dans laquelle nous pouvons entrer un type de voiture, sans se soucier de sa gamme et de son modèle. Par conséquent, si nous spécifions un "Type de voiture" pour une voiture de location, nous ne possédons pas de propriété dans la définition Voiture de location, dans laquelle la marque ou le modèle de la voiture peut être conservé. Nous utilisons le mot clé QUALIFIABLE.

```
RENAME DEFINITION "Utilisateur 5" TO "Voiture de location"
Definition "Voiture de location"
{
Property "Type de voiture"
{EDIT ONEOF "Version de voiture" KEYED BY { "Marque"
QUALIFIABLE, "Modèle de voiture" QUALIFIABLE, Nom }
}
}
```



La phrase QUALIFIABLE oblige la propriété ONEOF "Type de voiture" à stocker les informations sur la marque et le modèle. Ces informations sont stockées dans la valeur elle-même et sont séparées par des points. Vous pouvez déplacer des valeurs de la boîte de dialogue Sélectionner et faire glisser qui s'ouvre si vous appuyez sur le bouton Choix ou entrer les valeurs avec les points appropriés.

Exemple d'utilisation de clause Where

Nous souhaitons spécifier qu'une "Version de voiture" appartient à une catégorie d'automobiles : un véhicule utilitaire, une petite voiture, une compacte, une berline intermédiaire, une grosse berline, une berline de luxe ou un camion. Cette liste étant assez stable, nous n'avons pas besoin de créer de type de définition pour elle. Nous créons une liste de "Types de véhicule".

```
LIST "Types de véhicules"
{
Value "Véhicule utilitaire"
Value "Petite voiture"
Value "Compacte"
Value "Berline intermédiaire"
Value "Grosse berline"
Value "Berline de luxe"
Value "Décapotable"
Value "Camion"
```

```
}

Definition "Version de voiture"
{
Property "Modèle de voiture"
{Key Edit oneOf "Modèle de voiture" KEYED BY
{"Gamme":"Marque", Nom}  RELATE BY "indexé par"}
Property "Marque"
{KEY EDIT OneOf "Marque de voiture" RELATE BY
"indexé par"}
Property "Type de véhicule"
{EDIT Text List "Types de véhicule" DEFAULT "Berline
intermédiaire"}
}
```

Clause Where (suite)

Nous créons une définition de type "Campagne publicitaire de véhicule utilitaire". Dans une propriété de cette définition, nous souhaitons que les utilisateurs puissent sélectionner des instances d'automobiles d'un certain type. En d'autres termes, nous souhaitons que cette propriété soit filtrée pour ne contenir que les instances des définitions de l'encyclopédie qui remplissent la condition indiquée : 'Type de véhicule' = 'SUV'. Nous utilisons la clause "Where" pour effectuer ce filtrage.

```
Definition "Campagne publicitaire de véhicule
utilitaire"
{
Property "Type de véhicule utilitaire"
{ Edit OneOf "Types de véhicule" WHERE "Types de
véhicule" = "Véhicule utilitaire" }
}
```

Masquage des entrées standard dans le fichier SAPROPS.CFG

Vous pouvez masquer ou rendre invisible les propriétés du fichier USRPROPS.TXT. La question se pose souvent de savoir ce qui se passe lorsque vous masquez une propriété pour laquelle des informations ont déjà été entrées dans l'encyclopédie. Supposons par exemple que vous ayez ajouté des éléments de données à l'encyclopédie et fourni les secteurs de marché responsable pour chaque élément de la propriété *Unité commerciale*. Toutefois, à mi-projet, il est décidé que la propriété *Unité commerciale* n'est plus nécessaire. Vous n'avez qu'à changer la propriété *Unité commerciale"* en propriété *invisible* ou *masquée* en modifiant le fichier USRPROPS.TXT ; elle n'apparaît alors plus dans la boîte de dialogue de définition des éléments de données.

```
DEFINITION "Elément de données"
{
PROPERTY "Unité commerciale"
{ INVISIBLE }
}
```

Lorsque vous entriez des valeurs *Unité commerciale*, elles étaient ajoutées à l'encyclopédie et sauvegardées dans le fichier ENTITY.DBT. La modification du métamodèle de l'encyclopédie (ci-dessus) fait disparaître la propriété Unité commerciale de la boîte de dialogue des définitions d'élément de données, mais ne supprime pas les entrées précédentes des informations sur l'unité commerciale ; ces informations existent toujours dans le fichier .DBT. Si vous supprimez le code ci-dessus du fichier USRPROPS.TXT, la propriété *Unité commerciale* apparaît de nouveau dans la boîte de dialogue des définitions d'élément de données et les valeurs précédemment entrées réapparaissent dans leur élément de données respectif.

La question suivante se pose maintenant : comment pouvez-vous supprimer les informations en trop sur les propriétés inutiles du fichier ENTITY.DBT ? Vous pouvez pour cela utiliser les commandes **Exporter des définitions** et **Importer des définitions** du menu **Dictionnaire** avec précautions.

1. Sélectionnez Dictionnaire, Exporter les définitions. Choisissez d'exporter les définitions d'élément de données au format CSV dans un fichier texte.
2. Ouvrez ce fichier texte csv dans un éditeur externe tel qu'Excel et supprimez la colonne contenant les informations non souhaitées (dans le cas ci-dessus, ce serait Unité commerciale).
3. Sélectionnez Dictionnaire, Importer les définitions. Réimportez le fichier CSV à l'aide de l'option *Supprimer toutes les zones, puis ajouter les nouvelles données.*

Cette tâche est facultative. Elle n 'est nécessaire que si vous souhaitez récupérer l'espace mémoire utilisé par les valeurs en excès.

Messages d'erreur

Messages d'erreur

Chaque fois que Rational System Architect ouvre une encyclopédie et analyse son fichier SAPROPS.CFG et son fichier USRPROPS.TXT, il vérifie la syntaxe des instructions du fichier. Les erreurs de syntaxe détectées sont affichées dans une boîte de dialogue **Erreur**. Cette boîte de dialogue **Erreur** peut contenir les messages d'erreur ci-après. Les crochets indiquent les points auxquels Rational System Architect insère des variables, telles que le nom d'une propriété ou un numéro de ligne.

< > sur la ligne < > de USRPROPS.TXT
< > défini plusieurs fois
< > déjà défini comme Liste
Impossible de charger DLL (STATBAR.DLL).
Impossible de charger DLL (STATBOX.DLL).
Chapitre < > déjà défini.
Classe de dictionnaire < > déjà définie.
Description
Dictionniare
Arg. non conforme < >
Arg. non conforme < > - doit être entre guillemets
Val. par déf. < > non conforme pour édition Boolean
Val. par déf. < > non conf. pour édition Date
Val. par déf. < > non conf. pour édition Numeric
Val. par déf. < > non conf. pour édition Time
Val. par déf. < > non conf. pour édition Numeric dans intervalle
Ress. insuf. pour charger dialogue \n%s.
Nom classe dict. non val. : < >.
Nom type princ. non val. : < >.
Nom relation non val. : < >.
Liste < > déjà définie.
Nom liste < > non défini
Nom < > déjà utilisé
Nbre éditions propriétés (OneOf, ListOf, ExpressionOf) au-dessus limite avec < > sur
Nbre propriétés au-dessus limite avec < > sur < >

Nbre propriétés DISPLAYed au-dessus limite avec < > sur < >
Nbre listes au-dessus limite avec < > sur < >
Nbre listes au-dessus limite avec < > sur < > (max=100)
Arg. numérique < > hors interv.
Arg. numérique attendu mais < > était
Arg. longueur < > hors interv. ou non valide
Fin fichier prématurée apès < >
Nom liste précéd. défini
Propriété < > déjà définie
Liste référencée < > non définie.
Erreur de syntaxe dans < > Ligne <numéro de ligne>.
Le type d'édition < > est uniquement valide pour la propriété 'Description'
Listes trop nombreuses.[2]
Propriétés < > trop nombreuses.[3]
Valeurs dans la liste < > trop nombreuses.[4]
Impossible d'ouvrir le fichier de propriétés
Blocs de début-fin non concordants ou { }
Commande < > inattendue
Type DISPLAY de propriété < > inconnu
Nom de dictionnaire < > inconnu
Type d'édition < > inconnu
Type d'initialisation < > inconnu
Type de mise à jour < > inconnu
Avertissement - RANGE trouvé mais aucun intervalle maximum défini.
Avertissement - RANGE trouvé mais aucun intervalle minimum défini.

---

2 Le nombre maximal de listes est 400. Cela inclut les fichiers SAPROPS et USRPROPS, où le nombre de listes réellement utilisées dans SAPROPS dépend de la configuration de l'encyclopédie.

3 Le nombre maximal de propriétés pour un diagramme, un symbole ou une définition est 128

4 Le nombre maximal de VALEURS dans une LISTE est 128.

En plus d'un message d'erreur, Rational System Architect place des informations supplémentaires dans la boîte de dialogue des erreurs sur l'erreur de syntaxe détectée, comme suit :

lors de la vérification d'une commande **DEFINITION**.
lors de la vérification d'une commande **DISPLAY**.
lors de la vérification d'une commande **LIST**.
lors de la vérification d'une commande **VALUE** dans une commande **LIST**.
lors de la vérification d'une commande **PROPERTY** dans une commande **DEFINITION**.
lors de la vérification d'une commande **DEFINITION** ou **LIST**.
Voulez-vous continuer ?

Par exemple, l'intégralité du message d'erreur peut se présenter ainsi :

Type DISPLAY de propriété < > inconnu lors de la vérification d'une commande **DEFINITION**.

Editions lors de la phase d'exécution

La "Phase d'exécution" correspond à la phase pendant la quelle vous représenter des diagrammes et, en particulier, vous créez des entrées d'encyclopédie. Les boîtes de dialogue qui s'affichent lorsque vous ajoutez ou modifiez le dictionnaire sont contrôlées par les fichiers SAPROPS.CFG et USRPROPS.TXT ; les commandes **EDIT** servent à empêcher l'utilisateur de créer des entrées erronées. Supposons par exemple que le fichier SAPROPS.CFG contienne l'entrée suivante :

```
PROPERTY "Ma propriété"
{ EDIT numeric LENGTH 2 MINIMUM 1 MAXIMUM 32 }
```

Dans cet exemple, un message d'erreur **Valeur incorrecte** est affiché si vous saisissez "AB" ou "0" dans la zone de texte "Ma propriété" et que vous cliquez sur OK pour fermer la boîte de dialogue. Cela se produit car la propriété a été spécifiée comme valeur numérique (elle ne peut pas être constituée de lettres) dont la propriété minimale est 1.

Rational System Architect édite l'environnement d'exécution comme suit :

BOOLEAN	doit être T, F, TRUE ou FALSE
DATE	numérique, au format MM/JJ/AA
Expression, ExpressionOf, ListOf, OneOf	voir les entrées de la section qui commence à la page 2-75.
NUMERIC	doit correspondre à une chaîne numérique
TEXT	pas d'édition
TIME	numérique, au format HH:MM:SS

3

Mots clés dans USRPROPS.TXT Keywords

Introduction

Ce chapitre contient une liste alphabétique de tous les mots clés que vous pouvez utiliser pour apporter des modifications à USRPROPS.TXT.

Certaines restrictions affectent l'utilisation des mots clés suivants : CHAPTER, GROUP, LABEL, LIST et LISTONLY. Reportez-vous à chacun de ces mots clés pour une explication des restrictions spécifiques qui s'y appliquent.

Mots clés de USRPROPS

$$FORCE$$	Voir le mot clé DISPLAY.
$$NONE$$	Voir le mot clé DISPLAY.
$$VFORCE$$	Voir le mot clé DISPLAY.
$$VNONE$$	Voir le mot clé DISPLAY.

#IFDEF

Permet d'activer des commandes dans USRPROPS.TXT selon que la clause entre guillemets après la commande IFDEF ait été activée ou non dans la boîte de dialogue de configuration des propriétés. La boîte de dialogue de configuration des propriétés (Outils, Personnaliser le support des méthodes, Configuration de l'encyclopédie) modifie le fichier sadeclar.cfg dans une encyclopédie. Il s'agit du fichier sadeclar.cfg vérifié lorsque l'instruction IFDEF est évaluée au cours de l'analyse de SAPROPS.CFG.

Cette commande doit se terminer par une instruction #endif correspondante.

Exemple :

```
#ifdef "Business Enterprise"
DEFINITION "ORGUNIT"
{
LAYOUT { COLS 2 ALIGN OVER   }
PROPERTY "RowDefinition"
{ KEY EDIT OneOf "Unité organisationnelle" RELATE BY "fait partie
de" ReadOnly LABEL "Unité organisationnelle"}

PROPERTY "ColumnDefinition"
{ KEY EDIT OneOf "Unité organisationnelle" RELATE BY "fait partie
de" ReadOnly LABEL "Unité organisationnelle"}

PROPERTY "Description"
{ EDIT Text LENGTH 255 HELP "Apparaît dans la cellule d'une
matrice" }

PROPERTY "Intersection?"
{ EDIT Boolean LENGTH 1 }
}
#endif
```

#IFDEF (suite)

Dans l'exemple ci-dessus, le type de définition “ORGUNIT” contient uniquement les propriétés spécifiées si Business Enterprise a été activé dans la boîte de dialogue **Configuration des propriétés** (Outils, Personnaliser le support des méthodes, Configuration de l'encyclopédie). Si vous spécifiez les valeurs de ces propriétés, puis que vous désactivez Business Enterprise, ces valeurs sont conservées dans la définition de “ORGUNIT” dans le référentiel mais elles ne sont pas affichées dans la boîte de dialogue de définition car cet ensemble de propriétés est désactivé.

Voir aussi le mot clé #IFNDEF.

#IFNDEF

Contrairement à la commande IFDEF, la commande IFNDEF vous permet d'activer des commandes dans USRPROPS.TXT si la propriété mentionnée entre guillemets après celle-ci n'a **pas** été activée dans la boîte de dialogue Configuration des propriétés. Cette commande doit se terminer par une instruction #endif correspondante.

Exemple :

#ifndef "Business Enterprise"

RENAME Symbol "Swim Lane" to "Org. Unit"

#endif

Dans l'exemple ci-dessus, si l'option “Enterprise Architecture” n'a pas été activée dans la boîte de dialogue Configuration des propriétés de System Architect (Outils, Personnaliser le support des méthodes, Configuration de l'encyclopédie), tous les symboles ‘Swim Lane’ seront renommés en “Org. Unit”. Vous pouvez constater cette modification lorsque vous sélectionnez un symbole de ce type dans un diagramme sur lequel il est utilisé. L'option “Enterprise Architecture” dans la boîte de dialogue de configuration s'intitulait “Business Enterprise”, mais elle été renommée dans la boîte de dialogue de la version V9.0. Toutefois, l'instruction de commutation sous-jacente qu'elle appelle dans SADECLAR.CFG s'intitule toujours “Business Enterprise”.

#INCLUDE

#include peut être utilisé dans le fichier USRPROPS.TXT afin de répartir les modifications dans des fichiers supplémentaires distincts. Chacun de ces fichiers peut comporter d'autres inclusions, elles-mêmes contenant d'autres inclusions, etc. Le niveau d'imbrication admis par le programme d'analyse est de 10. Au delà, Rational System Architect génère un avertissement et ignore les niveaux suivants.

Exemple :

Par exemple, vous pouvez créer trois fichiers USRPROPS.TXT, un pour les diagrammes (nommé arbitrairement diagrams.txt), un pour les définitions (nommé arbitrairement definitions.txt) et un pour les symboles (nommé arbitrairement symbols.txt). Le fichier USRPROPS.TXT est similaire au fichier suivant :

```
**************************************************************************
REM "USRPROPS.TXT"
REM "Copyright IBM.  All rights reserved."
REM "Les instructions de modification de ce fichier figurent dans l'aide en ligne."

#include “diagrams.txt”
#include “symbols.txt”
#include “definitions.txt”
**************************************************************************
```

Dans chacun de ces fichiers, vous pouvez placer des instructions #includes dans d'autres fichiers (par exemple, un fichier pour les listes (nommé arbitrairement lists.txt)).

Cette commande permet un niveau de réutilisation cohérent des données définies par l'utilisateur.

ADDRESSABLE

Dans Rational System Architect, les symboles peuvent être *adressés par* une ou plusieurs définitions (adressables). Treize définitions adressables sont automatiquement fournies et peuvent être destinées à n'importe quel symbole : *Objectifs métier, Processus métier, Demandes de modification, Facteur déterminant de réussite, Collecte des données actuelles, Classe de données, Livrable, Organisation fonctionnelle, Emplacement géographique, Informations requises, Objectifs de l'organisation,* , *Exigences et Plans de test.* Ces définitions adressables sont disponibles en sélectionnant un symbole dans un diagramme, puis en sélectionnant dictionnaire, Adresses. En outre, toute définition peut être déclarée adressable via la syntaxe du fichier USRPROPS.TXT. De la sorte, la définition est disponible pour s'adresser à un symbole et ce dernier est placé dans la liste déroulante Dictionnaire, Adresses.

Exemple :

```
DEFINITION "xxxxxx"
 {
   ADDRESSABLE
 }
```

Voir aussi le mot clé NONADDR.

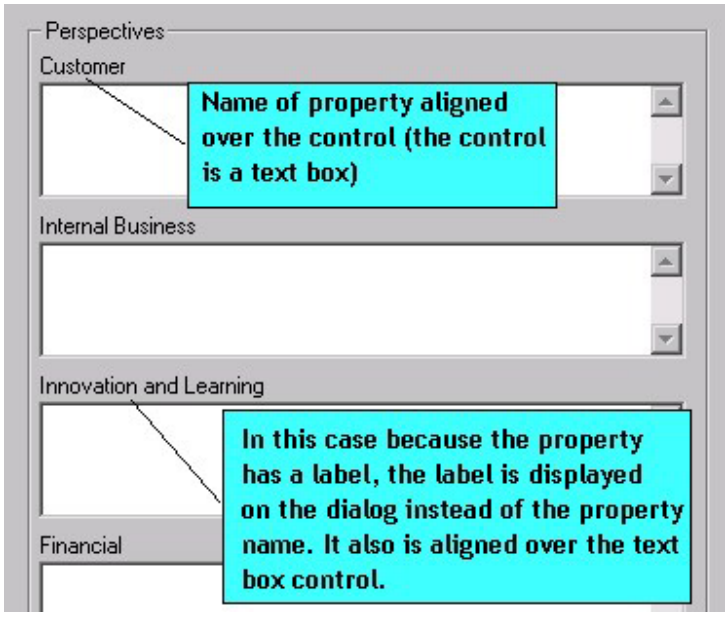
ALIGN

Permet de spécifier le positionnement du nom (ou du libellé) de la commande d'une propriété (zone de liste, zone de texte, etc.) dans une boîte de dialogue. Les options valides sont : *BODY*, *LABEL* et *OVER*.

Exemple :

```
DEFINITION "Fiche de score équilibrée"
{..
GROUP "Perspectives"
{
LAYOUT { COLS 2 TAB ALIGN OVER }
PROPERTY "Client" { EDIT Text LENGTH 300 }
PROPERTY "Activité interne" { EDIT Text LENGTH 500 }
PROPERTY "Apprentissage" { EDIT Text LENGTH 500 LABEL
"Innovation et apprentissage"}
PROPERTY "Financier" { EDIT Text LENGTH 500 }
}
```

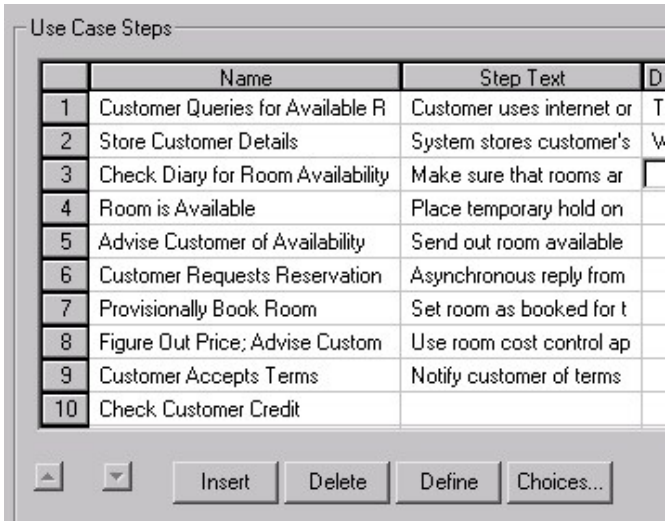
Dans l'exemple ci-dessus, la commande ALIGN OVER place les noms ou les libellés de toutes les propriétés sous l'instruction LAYOUT sur leur commande respective dans une boîte de dialogue de définition Fiche de score équilibrée.



Voir aussi les mots clés BODY, LABEL, OVER, JUSTIFY et TAB

ASGRID

La commande ASGRID spécifie de présenter une propriété ListOf dans une table ou une grille. Le type d'édition d'ASGRID doit être ListOf ou ParmListOf. Sinon, Rational System Architect génère un avertissement lorsque vous rouvrez l'encyclopédie et ignore le mot clé ASGRID .



Exemple :

Definition "Cas d'utilisation"
{CHAPTER "Etapes"
PROPERTY "Etapes de cas d'utilisation" { EDIT COMPLETE ListOf "Etape de cas d'utilisation" KEYED BY { "Package", "Nom du cas d'utilisation":Name, Name} **ASGRID** LENGTH 1200 } }

Utilisation d'ASGRID avec des définitions saisies

Les propriétés de clé d'une définition ne sont pas affichées dans une grille constituée par une commande ASGRID. Dans l'exemple ci-dessus, le nom de package de chaque étape de cas d'utilisation ou le nom du cas d'utilisation n'est pas affiché dans la grille.

ASGRID (suite)

Limitation d'ASGRID

Vous ne pouvez pas utiliser ASGRID dans une commande LISTOF qui fait référence à une définition située dans une commande COMPLETE ListOf d'une autre définition. Ainsi, par exemple, vous pouvez ajouter une commande ListOf "Attribute" à une définition mais sans spécifier ASGRID. La longueur maximale d'une propriété affichable dans la grille (GRID) est 400.

ASGRID COUNT_FIXED

Le mot-clé COUNT_FIXED est utilisé avec le mot-clé ASGRID pour indiquer que l'utilisateur ne peut pas supprimer des lignes d'une grille ou en insérer.

Voir aussi les mots clés KEY, KEYED BY et COUNT_FIXED.

ASGUID

Ce mot-clé ne peut être utilisé qu'avec les propriétés de texte. Il alimente automatiquement une propriété avec la valeur “GUID”. Cette propriété de texte peut ensuite être utilisée comme propriété de clé à la place de la propriété GUID réelle. La propriété ASGUID doit être en lecture seule. Lorsque vous rouvrez la définition, la valeur de la propriété ASGUID est renseignée.

Exemple :

```
RENAME DEFINITION “Utilisateur 1” to “Ma déf.”
DEFINITION “Ma déf.”
{
PROPERTY “Ma prop.”
{KEY EDIT Text LENGTH 100 ASUID READONLY}
Property “HIYA”
{EDIT Text Length 145}
}
```

ASPARMGRID

Le mot clé ASPARMGRID a été spécifiquement créé pour être utilisé dans le cadre de la modélisation de données dans Rational System Architect et opère à partir de code créé à cet effet. Ce mot clé se trouve dans SAPROPS.CFG et **ne doit pas** être employé par l'utilisateur dans USRPROPS.TXT.

ASSIGN

Vous pouvez affecter de nouveaux types de symbole ou des types de symbole existants (symboles qui existent déjà dans un autre diagramme) à des types de diagramme nouveaux ou existants. Des types de symbole peuvent être ajoutés à des types de diagramme à l'aide de la syntaxe suivante :

SYMBOL <nom-type-symbole> [IN <nom-type-diagramme1>]
ASSIGN [TO] <nom-type-diagramme2>

Exemple :

SYMBOL “Unité organisationnelle” IN “Organigramme”
{
ASSIGN TO “Directives de l'entreprise”
}

AUDITID

Ce mot clé représente les caractères saisis dans la boîte de dialogue **ID audit** lorsque l'utilisateur se connecte pour la première fois à Rational System Architect. AUDITID est un mot clé admis qui indique qu'une propriété contient l'ID d'audit de l'utilisateur. CHECKOUT AUDITID, FREEZE AUDITID, INITIAL AUDITID et UPDATE AUDITID ont chacun une signification spéciale. Pour plus d'informations, reportez-vous à chacun de ces mots clés, répertoriés séparément dans cette table.

Remarque : A partir de la version 9.1 de Rational System Architect, les mots clés INITIAL AUDITID (fourni dans la zone “Audit initial”) et UPDATE AUDITID (fourni dans la zone “Audit de la dernière vérification”) sont automatiquement inclus dans l'onglet Données d'accès de chaque définition.

Exemple :

```
DIAGRAM "Flux de données Gane & Sarson"
 {
   PROPERTY "Gelé par"
 { FREEZE AUDITID }
```

Toute définition peut contenir d'autres utilisations de l'ID audit.

Exemple :

```
DEFINITION "X"
 {
   PROPERTY "Nom du propriétaire actuel"
 { EDIT Text CHECKOUT AUDITID LENGTH 12 READONLY }
 }
```

AUTOCREATE

La commande AUTOCREATE crée automatiquement une définition derrière la valeur spécifiée dans une propriété, dès que vous cliquez sur le bouton OK pour fermer la boîte de dialogue qui la contient. Si vous n'utilisez pas le mot clé AUTOCREATE, la valeur entrée dans une propriété reste indéfinie une fois que vous avez cliqué sur le bouton OK pour fermer la boîte de dialogue qui la contient.

Exemple :

```
DEFINITION "Base de données physique"
{..
PROPERTY "Modèle"
{ KEY EDIT ONEOF "Modèle de données de projet" AUTOCREATE
RELATE BY "indexé par" READONLY }
..}
```

Dans l'exemple ci-dessus, la boîte de dialogue de définition de base de données physique contient une propriété intitulée “Modèle de données de projet”. Si vous renseignez cette zone (par exemple, “Réservations”), puis que vous cliquez sur OK pour fermer la boîte de dialogue Définition de base de données physique, la définition du modèle de données de projet “Réservations” est automatiquement créée dans l'encyclopédie.

Voir aussi INITIAL USER REQUIRED et OVERRIDABLE.

BEGIN

Ce mot clé indique le début de la définition d'une propriété ou d'une série de propriétés qui composent la définition d'un diagramme, d'un symbole ou d'une définition. Vous pouvez également utiliser la syntaxe suivante : **{**.

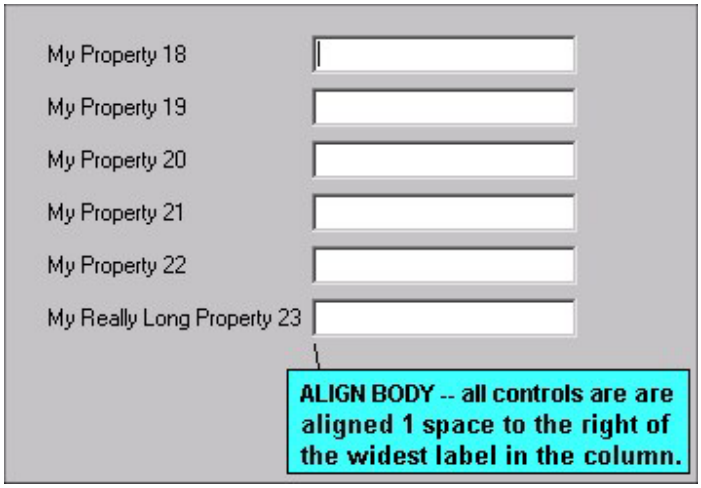
Voir aussi le mot clé PROPERTY.

BODY

Il s'agit d'un des arguments utilisés dans la commande **ALIGN**. Elle permet d'aligner tous les contrôles à un espace à droite du libellé le plus long de cette colonne. (A comparer avec la paire de mots clés ALIGN OVER, qui place le nom sur la propriété).

Exemple :

```
Definition "Ma définition"
{
CHAPTER “Mon chapitre”
LAYOUT { COLS 1 ALIGN BODY }
PROPERTY "Ma propriété 18"{ EDIT Text Length 10}
PROPERTY "Ma propriété 19"{ EDIT Text Length 10}
PROPERTY "Ma propriété 20"{ EDIT Text Length 10}
PROPERTY "Ma propriété 21"{ EDIT Text Length 10}
PROPERTY "Ma propriété 22"{ EDIT Text Length 10}
PROPERTY "Ma très longue propriété 23"{ EDIT Text Length 10}
}
```



Dans l'exemple ci-dessus, le contrôle de “Ma très longue propriété 23” est une zone de texte placée à un espace à droite du libellé. Tous les autres contrôles de zone de texte des autres propriétés de la boîte de dialogue sont alignés avec ce contrôle.

Remarque – ALIGN BODY plaçait tous les contrôles à un espace à droite du libellé, mais a été par la suite modifié pour se comporter comme ALIGN LABEL.

Voir aussi les mots clés OVER, ALIGN, TAB, LABEL et JUSTIFY.

BOOLEAN

Définition sous forme de case à cocher. Elle peut recevoir l'une des deux valeurs suivantes : True (T) ou False (F).

Exemple :

Dans l'exemple ci-après, l'utilisateur est autorisé à activer ou désactiver les fonctions de numérotation hiérarchique sur un diagramme IDEF0 en sélectionnant true ou false.

```
DIAGRAM "IDEF0"
 {
   PROPERTY "Numérotation hiérarchique"
   { EDIT Boolean LENGTH 1 DEFAULT "F" }
..
}
```

BROWSER (Explorateur)

Indique si une propriété et sa valeur sont affichées dans la zone Propriétés de l'explorateur (navigateur) de Rational System Architect lorsque le diagramme, la définition ou le symbole correspondant est sélectionné dans l'explorateur.

Les instructions de contrôle suivantes de l'explorateur sont autorisées (le terme "objet" est utilisé pour désigner un diagramme, un symbole ou une définition) :

Dans la spécification d'une propriété :

- **BROWSER {SHOW} :** Indique à l'explorateur d'afficher la valeur de cette propriété lors de l'affichage de l'objet qui la contient.

Dans la spécification d'un objet, mais pas dans la description d'une propriété :

- **BROWSER {OMITKEY} :** Indique à l'explorateur de ne pas afficher les propriétés clés de l'objet dans les cas où elles devraient l'être.
- **BROWSER {OMITTYPE} :** Indique à l'explorateur de ne pas afficher le type de l'objet dans les cas où il devrait l'être.

Pas dans la spécification d'un objet :

- **BROWSER {OMITKEY} :** Indique à l'explorateur de ne pas afficher les propriétés clés d'on objet quelconque dans les cas où elles devraient l'être.
- **BROWSER {OMITTYPE} :** Indique à l'explorateur de ne pas afficher le type d'un objet quelconque dans les cas où il devrait l'être.

L'énoncé "dans les cas où... " est utilisé ci-dessus car souvent, l'explorateur n'affiche pas tout ou partie des propriétés clés (lorsque l'affichage de l'objet est subordonné à l'un de ses objets clés) et n'affiche pas son type (lorsque son affichage est subordonné à un en-tête de type).

BROWSER (Explorateur, suite)

Exemple 1 :

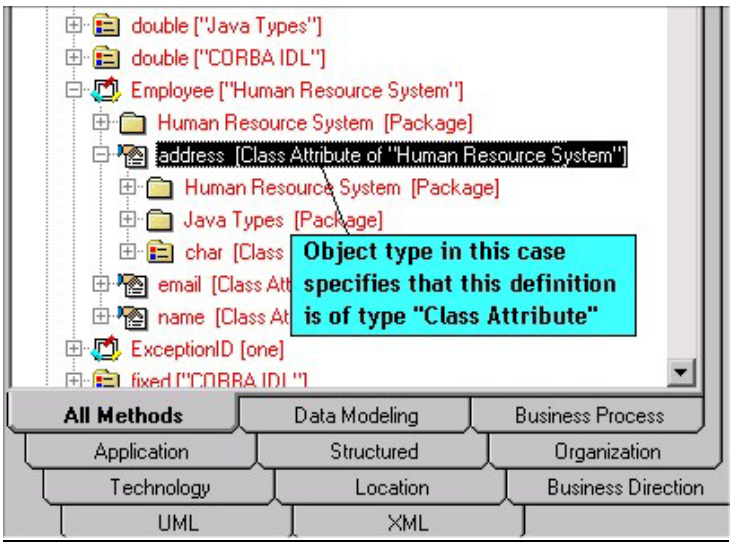
DEFINITION "Fin d'association"
{
PROPERTY "Package" { KEY EDIT OneOf "Package" RELATE BY "indexé par" READONLY **BROWSER { SHOW }}**
..}

Dans l'exemple ci-dessus, la valeur de la propriété package est affichée dans l'explorateur, bien qu'elle ne le soit généralement pas.

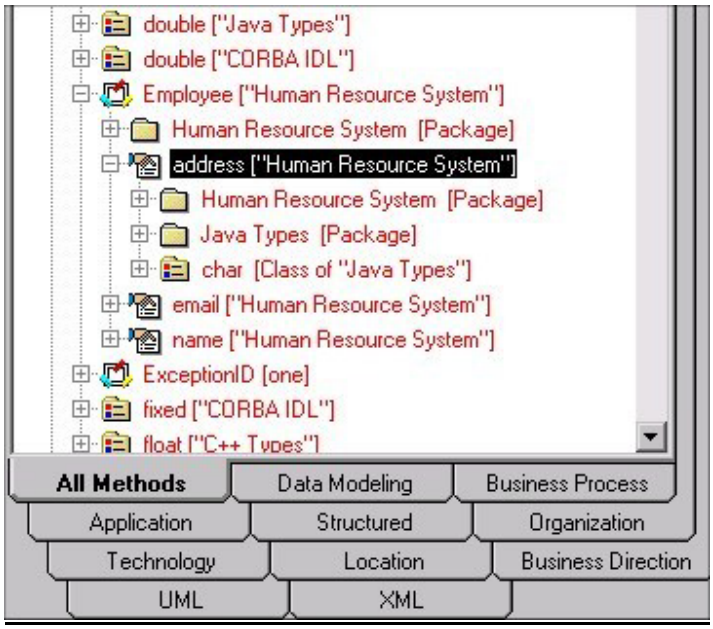
Exemple 2 :

DEFINITION "Attribut de classe"
{
BROWSER { OMITTYPE }
..}

Dans l'exemple ci-dessus, un attribut de classe est une définition de type "Attribut de classe". Par défaut, il est affiché dans l'explorateur, ce qui fait double emploi et peut être considéré comme une gêne visuelle. Si la commande BROWSER (OMITTYPE) n'était pas utilisée, l'explorateur afficherait les attributs illustrés dans le diagramme ci-dessous.



L'utilisation de la commande BROWSER {OMITTYPE} spécifie à l'explorateur d'afficher un attribut comme illustré dans le diagramme ci-après.



Exemple 3 :

```
DEFINITION "Association"
{
BROWSER { OMITKEY }
LAYOUT { COLS 1 ALIGN OVER   TAB }
CHAPTER "Rôles"
PROPERTY "GUID d'association" { KEY EDIT Text LENGTH 64
INVISIBLE READONLY}
PROPERTY "Rôles de la classe" { EDIT COMPLETE ListOf "Fin
d'association" KEYED BY { "GUID d'association":"GUID d'association",
"Association":"Nom", "Package" QUALIFIABLE, "Classe"
QUALIFIABLE, "GUID du rôle" QUALIFIABLE, "Nom" }
RELATE BY "utilise" LENGTH 4096 ASGRID COUNT_FIXED
BROWSER { SHOW } }
..
}
```

Dans l'exemple ci-dessus, la valeur de la propriété "Rôles de la classe" est affichée dans l'explorateur (car il est important de savoir à quelles classes une association est rattachée), même si elle ne l'est habituellement pas.

BY

Mot clé fréquemment utilisé comme illustré dans les expressions suivantes : *DEFINED BY, RELATED BY, RELATE BY* et *KEYED BY.* Pour plus d'informations, reportez-vous à la combinaison de mots clés spécifique.

Exemple :

```
DEFINITION "Colonne"
{..
PROPERTY "Nom de la base de données"
{ KEY EDIT OneOf "Base de données" RELATE BY "néant" }
  ..
}
```

CHAPTER

Crée ***des onglets*** dans une boîte de dialogue. Chaque instruction Chapter correspond à un onglet. La syntaxe est la suivante :

CHAPTER <nom_chapitre>

L'instruction Chapter ne requiert pas d'accolade ouvrante ou fermante pour regrouper les éléments sous l'onglet. Tous les éléments qui relèvent d'une instruction Chapter sont regroupés sous cet onglet. Le regroupement suivant est créé par la prochaine instruction Chapter.

Exemple :

CHAPTER "propriétés Screen Painter"

Modification du nom d'un onglet (instruction Chapter) :

Pour modifier le nom d'une instruction CHAPTER via USRPROPS.TXT, vous devez utiliser la commande LABEL.

Exemple :

Le fichier SAPROPS fournit un onglet Classes imbriquées pour une définition Classe :

```
DEFINITION "Classe"
 { CHAPTER "Classes imbriquées"
   ...
 }
```

Vous pouvez renommer la commande CHAPTER “Classes imbriquées” en “Fred” à l'aide de la commande LABEL dans USRPROPS.TXT :

```
DEFINITION "Classe"
 { CHAPTER "Classes imbriquées" LABEL "Fred"
    }
```

CHECKOUT

Affiche des informations sur la réservation d'un objet, telles que l'AUDIT ID de la personne qui a réservé l'objet ou les informations DATE ou TIME (heure) de la réservation. Les zones affichées sont toujours en LECTURE SEULE. Un suivi des valeurs est automatiquement effectué par Rational System Architect, mais pour les afficher dans une boîte de dialogue, vous devez ajouter des propriétés avec les caractéristiques respectives suivantes :

CHECKOUT Auditid
CHECKOUT Date
CHECKOUT Time

Exemple :

```
DIAGRAM "Flux de données Gane & Sarson"
 {
   PROPERTY "Réservé par"
 { CHECKOUT AUDITID }
   PROPERTY "Date de réservation"
 { CHECKOUT DATE }
   PROPERTY "Heure de réservation"
 { CHECKOUT TIME }
 }
```

Effectuez une recherche sur *Contrôle d'accès* dans l'aide en ligne pour plus d'informations sur la réservation et la restitution.

Voir aussi le mot clé FREEZE.

COLS, COLUMNS

Détermine le nombre de colonnes dans lesquelles un groupe de propriétés est placé dans une boîte de dialogue Diagramme, Symbole ou Définition.

Exemple :

```
DEFINITION "Référent"
{
LAYOUT { COLS 2 ALIGN OVER TAB }
...}
```

COLUMN_SCRIPT

COLUMN_SCRIPT appelle un script rédigé en SA Basic. Les scripts de colonne sont utilisés pour le comportement des colonnes dans les tables d'un modèle de données physique. L'action effectuée par le script opère sur chaque colonne de la liste.

Par convention, la fonction est elle-même nommée avec l'un des préfixes suivants :

- fmtxxx : la fonction est codée en dur et ne peut pas être modifiée. C'est le cas de la plupart des fonctions contenues dans SAPROPS.CFG. Le codage en dur de la fonction est effectué pour accélérer la réponse globale de Rational System Architect.
- _fmtxxx : existe dans le fichier fmtscript.bas sous le répertoire principal des exécutables de Rational System Architect.

Création de votre propre script

Pour plus d'informations sur la manière de créer votre propre script, reportez-vous au mot-clé SCRIPT.

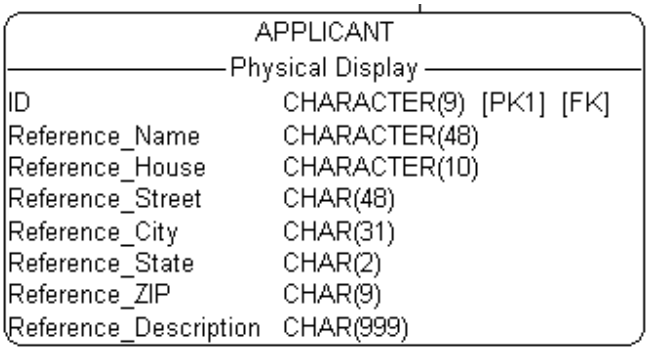
Exemple :

```
DEFINITION "Table"
{
PROPERTY "Description"
{
ZOOMABLE EDIT ListOf Definition "Colonne"  FROM "Elément de données" KEYED BY {"Nom de la base de données","Nom du propriétaire","Nom de la table":"Nom","Nom"} LENGTH 2000
DISPLAY { FORMAT Key LEGEND "Données clés" }
DISPLAY { FORMAT NonKey LEGEND "Données non clées" }
DISPLAY { FORMAT COLUMN_SCRIPT FmtERAttr LEGEND "Affichage physique" }
} ..}
```

FmtERAttr Renvoie les valeurs des diagrammes Relation d'entités entité ou Colonnes de table des diagrammes physiques.

FmtERAttr renvoie ID, NAME, ADDRESS, STREET, CITY, STATE, FIRST_5_DIGITS, ZIP CODE et LAST_4_DIGITS.

COLUMN_SCRIPT (suite)



Voir aussi les mots clés SCRIPT, COMPONENT_SCRIPT, VALUESCRIPT et FORMAT.

COMPLETE

Permet d'inclure la définition référencée dans la définition référençante de sorte que la définition référencée ne puisse pas être référencée par une autre définition et ne puisse être éditée qu'à partir de la définition référençante la contenant.

C'est notamment le cas d'un attribut dans une entité, qui appartient intégralement à une entité (et n'appartient pas à une autre définition) et qui ne peut être ouvert qu'à partir de la définition de l'entité (par exemple, vous ne pouvez pas ouvrir une définition d'attribut directement dans l'explorateur de Rational System Architect).

Exemple :

```
DEFINITION "Entité"
{
PROPERTY "Attributs"
  {ZOOMABLE EDIT COMPLETE ListOf "Attribut de classe" KEYED BY {"Nom de la classe":"Nom", Name} ASGRID LENGTH 4096 DISPLAY { FORMAT List } }
..
}
```

COMPONENT_ SCRIPT

Appelle une fonction rédigée en Basic, utilisant des appels de fonction de Rational System Architect inclus dans SA Basic. Les scripts de composant sont utilisés pour les listes ListOf et ExpressionOf. L'action effectuée par le script fonctionne sur chaque élément de la liste. Par exemple, la syntaxe **COMPONENT_SCRIPT** ***fmtomtattr*** renvoie tous les attributs et leurs types de stockage C- correspondants, séparés par un signe deux-points (:).

Par convention, la fonction est elle-même nommée avec l'un des préfixes suivants :

- fmtxxx : la fonction est codée en dur et ne peut pas être modifiée. C'est le cas de la plupart des fonctions contenues dans SAPROPS.CFG. Le codage en dur de la fonction est effectué pour accélérer la réponse globale de Rational System Architect.
- _fmtxxx : existe dans le fichier fmtscript.bas sous le répertoire principal des exécutables de Rational System Architect.

Création de votre propre script

Pour plus d'informations sur la manière de créer votre propre script, reportez-vous au mot-clé SCRIPT.

Explication des scripts existants :

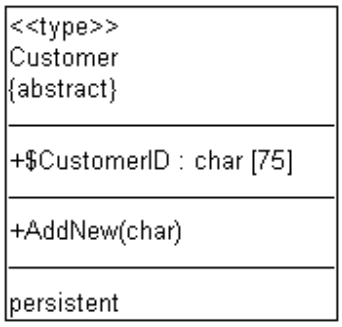
fmtUMLAttr renvoie tous les attributs et leurs types correspondants, séparés par un signe deux-points.

fmtOMTOperation renvoie toutes les opérations et leurs types de stockage C correspondants, inclus entre parenthèses (type).

FmtOMTObjInstAttr renvoie tous les attributs de la classe instanciée par un objet.

FmtOMTActivity renvoie le script **do**: et le nom de l'activité pour toutes les activités répertoriées dans une définition d'état.

FmtOMTStateActions renvoie le nom de l'action interne pour toutes les actions internes répertoriées dans une définition d'état.



Exemple (utilisant *fmtomattr*):

Definition “Classe” {
PROPERTY "Attributs"
{ PROPERTY "Attributs" {ZOOMABLE EDIT COMPLETE ListOf "Attribut de classe" KEYED BY {"Package", "Nom de la classe":"Nom", Name } LENGTH 4096 ASGRID DISPLAY { **FORMAT COMPONENT_SCRIPT _FmtNewUMLAttr** LEGEND "$$FORCE$$"} LABEL "Attributs" }
}

CONTROL

Ce mot clé est équivalent au mot clé Property (Propriété), s'il est utilisé avec TESTPROC pour configurer un commutateur dans une définition.

Il existe deux manières d'indiquer qu'une propriété apparaît dans une boîte de dialogue de définition en fonction de la valeur d'un commutateur. Vous pouvez utiliser #ifdef's, qui opère sur les valeurs définies pour l'encyclopédie dans la boîte de dialogue Configuration de l'encyclopédie (par exemple, en spécifiant comme type de langage de l'encyclopédie Java ou C++). La boîte de dialogue Configuration de l'encyclopédie définit en fait les valeurs dans le fichier sadeclar.cfg.

Vous pouvez également spécifier qu'un mot-clé PROPERTY apparaisse dans une boîte de dialogue (et préciser sa valeur initiale) en fonction d'un commutateur qui est lui-même une propriété (TESTPROPERTY) dans la boîte de dialogue de définition. Par exemple, vous pouvez spécifier dans une entité que son type de SGBD est Oracle ou SQL Server. Les

propriétés apparaîtront ou non dans la définition et posséderont certaines valeurs par défaut, en fonction de la valeur que vous avez défini pour le type de SGBD. Utilisez le mot-clé TESTPROC pour spécifier le commutateur TESTPROPERTY. Vous utilisez le mot-clé PROPERTY la première fois que vous spécifiez une propriété particulière dans la définition et le mot-clé CONTROL pour toutes les autres occurrences de cette propriété dans la définition. Le mot-clé REFPROP permet de spécifier à quelle mot-clé PROPERTY chaque mot-clé CONTROL fait référence. Pour cette raison, les mots-clés CONTROL et REFPROP sont souvent utilisés en conjonction avec TESTPROC.

Pour résumer, pour qu'un mot-clé CONTROL soit utilisé, il doit y avoir une référence initiale au mot-clé PROPERTY auquel le mot-clé CONTROL fait référence, au début de la définition. Le mot-clé REFPROP est utilisé en conjonction avec le mot-clé CONTROL.

Exemple :

```
Definition "Index"
{
CHAPTER "Propriétés de modélisation"
{ TESTPROC TestPropertyNotValue TESTPROPERTY "SGBD"
TESTSTRING { "ORACLE 8" } }
PROPERTY "Clé primaire" {EDIT Boolean LENGTH 1 DEFAULT "F"
READONLY }
PROPERTY “Unique”  {EDIT Boolean LENGTH 1 VALUESCRIPT
ProcessIndexUnique DEFAULT "F" }
PROPERTY “En cluster”  {EDIT Boolean LENGTH 1 DEFAULT "F" }
..
CHAPTER "Propriétés de modélisation"
  { TESTPROC TestPropertyValue TESTPROPERTY "SGBD"
TESTSTRING { "ORACLE 8" } }
CONTROL "Clé primaire" { REFPROP "Clé primaire" }
CONTROL “Unique” { REFPROP "Unique" }
CONTROL “En cluster” {REFPROP "En cluster"}
..
  }
```

Dans l'exemple ci-dessus, le mot clé REFPROP est utilisé conjointement avec le mot clé CONTROL afin de spécifier que

les propriétés “Clé primaire”, “Unique” et “En cluster” sont fournies à la définition d'index si Oracle 8 est sélectionné comme SGBD – ces propriétés correspondent exactement à leur propriété référencée.

COPY PROPERTIES FROM

Cette commande permet de copier des propriétés dans le type de définition actuel à partir d'autres types de définition. Elles permet de regrouper des concepts similaires dans un même type de définition. Elle ne s'applique qu'aux définitions. La syntaxe est la suivante :

DEFINITION <objet-1>
{
…
COPY PROPERTIES FROM <objet 2> {[, <objet n>]}…

Exemple :

DEFINITION "Eléphant"
{
…
CHAPTER "Propriétés copiées depuis Demande de modification"
COPY PROPERTIES FROM "Demande de modification"
CHAPTER "Propriétés copiées depuis Dépendance et Noeud"
COPY PROPERTIES FROM " Dépendance", "Noeud"
…
}

La copie est effectuée au moment où l'instruction de copie est rencontrée dans l'entrée. Si, dans l'exemple précédent, des propriétés sont ajoutées ultérieurement aux Demandes de modifications, Dépendances ou Noeuds dans les fichiers de propriétés, ou si des propriétés existantes sont modifiées par la suite dans ces fichiers, les ajouts et les modifications ne sont **pas** copiés.

COPYSCRIPT

Ce mot-clé permet de spécifier un script SABasic à appeler pour une propriété spécifique lors d'une copie de la définition.

Création de votre propre script

Pour plus d'informations sur la manière de créer votre propre script, reportez-vous au mot-clé SCRIPT.

Exemple :

```
DEFINITION "Entité"
{
CHAPTER "Attributs"
PROPERTY "Description"
{ EDIT COMPLETELISTOF "Attribute" FROM "Données" KEYED BY
{Model, "Nom de l'entité":"Nom", "Nom"} RELATE BY "utilise" ASGRID
COPYSCRIPT OnCopyEntityDesc EDITCLASS
SACPropertyAttributeGrid
..}
```

COUNT_FIXED

Le mot-clé COUNT_FIXED est utilisé avec le mot-clé ASGRID pour indiquer que l'utilisateur ne peut pas supprimer des lignes d'une grille ou en insérer. Le nombre de lignes est fixe.

Exemple :

```
DEFINITION "Association"
{
BROWSER { OMITKEY }
LAYOUT { COLS 1 ALIGN OVER   TAB }
CHAPTER "Rôles"
PROPERTY "GUID d'association" { KEY EDIT Text LENGTH 64
INVISIBLE READONLY}
PROPERTY "Rôles de la classe" { EDIT COMPLETE ListOf "Fin
d'association" KEYED BY { "GUID d'association":"GUID d'association",
"Association":"Nom", "Package" QUALIFIABLE, "Classe"
QUALIFIABLE, "GUID du rôle" QUALIFIABLE, "Nom" }
RELATE BY "utilise" LENGTH 4096 ASGRID COUNT_FIXED
BROWSER { SHOW } }
```

Dans l'exemple ci-dessus, une association entre des classes occupe une ligne dans la grille pour chaque classe à laquelle elle se rapporte (il s'agit habituellement de deux classes, mais elles peuvent être au nombre de trois, voire plus, si des classes supplémentaires sont rattachées à la ligne d'association – ce comportement est codé en dur dans le logiciel). En raison du mot-clé COUNT_FIXED, les utilisateurs ne peuvent pas ajouter ou supprimer de lignes dans la grille.

A comparer avec les autres grilles (par exemple, la grille Etape de cas d'utilisation, dans laquelle les utilisateurs peuvent ajouter des étapes à la grille ou en supprimer).

DATA

Il ne s'agit ***pas*** d'un mot clé mais d'un terme spécial utilisé comme argument des commandes ONEOF, LISTOF et EXPRESSIONOF, en fournissant une référence aux éléments de données et aux structures de données qui constituent les dictionnaire de données de Rational System Architect.

DATE

Ce mot-clé correspond à un type d'édition dont la longueur doit être 10. L'affichage graphique est basé sur le format de date défini dans Windows. DATE correspond également à un type de zone autorisable qui indique qu'une propriété contient un horodatage dans la notation correspondant au format horaire défini dans Windows.

CHECKOUT DATE, FREEZE DATE, INITIAL DATE et UPDATE DATE ont tous des significations spéciales.

Exemple 1 :

```
DIAGRAM "Flux de données Gane & Sarson"
 {
   PROPERTY "Date de gel"
 { FREEZE DATE }
```

D'autres utilisations du mot clé DATE peuvent être trouvées dans n'importe quelle définition.

Exemple 2 :

```
DEFINITION "X"
 { PROPERTY "Date de création"
   { EDIT Text INITIAL DATE LENGTH 12 READONLY }
 }
```

DEASSIGN

Le mot-clé DEASSIGN permet de supprimer des symboles d'un type de diagramme.

Exemple :

```
SYMBOL "Flux de messages" in "Processus métier"
{
DEASSIGN from "Processus métier"
}
```

DEFAULT

Valeur affectée par Rational System Architect à une propriété et pouvant être remplacée par l'utilisateur. Dans l'affichage graphique, la valeur par défaut est initialement affichée dans une zone de texte ou détermine si une case à cocher est initialement sélectionnée.

Exemple :

```
PROPERTY "N'est pas une table"
  { EDIT Boolean LENGTH 1 DEFAULT F }
```

DEFINED BY

Ce mot clé associe une définition à un symbole. Il permet également de réassocier un symbole à une autre définition.

Signification 1 : Si vous ajoutez de nouveaux symboles à une encyclopédie dans USRPROPS.TXT, vous devez spécifier à l'aide de ce mot clé le type de définition auquel ils sont associés. Si un nouveau symbole spécifié dans USRPROPS.TXT ne contient pas cette clause, Rational System Architect génère un avertissement d'analyse lors de l'ouverture de l'encyclopédie et utilise par défaut la définition null du symbole.

Exemple 1 :

```
RENAME DIAGRAM "Utilisateur 1" to "Mon diagramme"
RENAME SYMBOL "Utilisateur 1" to "Direction"
RENAME DEFINITION "Utilisateur 1" to "Direction"

SYMBOL "Direction"
{
DEFINED BY " Direction"
ASSIGN TO "Mon diagramme"
}
```

Dans l'exemple ci-dessus, de nouveaux type de diagramme, type de symbole et type de définition ont été spécifiés dans USRPROPS.TXT. Le mot clé DEFINED BY permet d'indiquer que le symbole “Direction” est défini par la définition “Direction”. De plus, le symbole est affecté au diagramme intitulé “Mon diagramme”. (Remarque : Vous pouvez également spécifier une instruction de définition pour la nouvelle définition, “Direction”, mais ceci n'est pas obligatoire. Si elle n'a pas été spécifiée, la nouvelle définition comportera simplement les propriétés par défaut “Nom” et “Description”.)

Signification 2 : Le mot clé DEFINED BY permet également de définir un symbole à l'aide d'une autre définition que celle spécifiée dans SAPROPS.CFG. Si vous utilisez ce mot-clé pour réassocier un symbole à une autre définition, prenez soin de spécifier dans quel diagramme le symbole auquel vous faites référence est représenté (par exemple, Symbol “Classe” in “Classe” versus Symbol “Classe” in “Composant” – dans le premier cas, vous spécifiez que vous redéfinissez la définition du symbole de classe dans un diagramme Classe et dans le second cas, dans un diagramme Composant).

Exemple 2 :

```
SYMBOL Process IN "Flux de données Gane & Sarson"
{
DEFINED BY "Conversion des contrôles"
}
```

Dans l'exemple ci-dessus, le symbole Processus dans un diagramme “Flux de données Gane & Sarson” est maintenant défini par “Conversion des contrôles” alors qu'il est normalement défini par “Processus”.

DEFINITION

Ce mot-clé est le premier mot-clé d'un bloc dans lequel les propriétés d'une DEFINITION, par opposition à un DIAGRAMME ou un SYMBOLE, sont répertoriées.

Exemple :

```
DEFINITION "Elément de données"
 {
   PROPERTY "Longueur"
 { EDIT number LENGTH 2 }
 .
 .
 .
 }
```

Voir aussi les mots clés DIAGRAM et SYMBOL.

DEFINITION REFERENCED IN

Voir ‘OF DEFINITION REFERENCED IN’.

DEPICT LIKE

La combinaison de mots-clés DEPICT LIKE permet de spécifier comment un symbole est représenté dans un diagramme. Vous pouvez utiliser cette combinaison de mots-clés lorsque vous créez un symbole et que vous spécifiez à quoi il doit ressembler dans un diagramme. Vous pouvez spécifier qu'il doit ressembler à un symbole d'un autre diagramme.

Vous pouvez utiliser la combinaison de mots clés DEPICT LIKE avec les symboles de ‘noeud’ et de ‘ligne’.

Exemple (Symbole de noeud) :

```
SYMBOL "Connexion des communications"
{
ASSIGN TO "Concept op. de niveau supérieur OV-01"
..
DEPICT LIKE "Flux d'événements" IN "Flux d'événements Ward &
Mellor"
```

Exemple (Symbole de ligne) :

```
SYMBOL "Ligne de besoin"
{
PROPERTY "Depuis Noeud opérationnel"
{EDIT ONEOF "Noeud opérationnel" READONLY INVISIBLE}
PROPERTY "Vers Noeud opérationnel"
{EDIT ONEOF "Noeud opérationnel" READONLY INVISIBLE}
DEPICT LIKE "Transition" IN "Etat OMT"
DEFINED BY "Ligne de besoin"
ASSIGN TO "Connectivité de noeud op. OV-02"
}
```

DEPICTIONS

Identifie comment un symbole peut être représenté par un fichier image que vous fournissez. Vous pouvez représenter un symbole à l'aide d'un bitmap ou d'un métafichier. Vous pouvez indiquer comment ce symbole est représenté dans l'espace de travail du diagramme en combinant les mots-clés DEPICTIONS et DIAGRAM. Vous pouvez également spécifier comment le symbole est représenté dans la boîte à outils et le menu Tracer en combinant les mots clés DEPICTIONS et MENU. La syntaxe est la suivante :

```
SYMBOL <nom-type-symbole>
        { ...
        DEPICTIONS { DIAGRAM <fichier-
        représentation> }
        DEPICTIONS { MENU <fichier-représentation>
        }
        ...}
```

où <fichier-représentation> correspond au nom et au chemin d'accès complet d'un bitmap ou d'un métafichier.

Exemple :

```
Rename Symbol "Utilisateur 3" To "Radar
SYMBOL "Radar"
{ASSIGN To "Réseau sans fil"
DEPICTIONS { DIAGRAM "C:\Program Files\IBM\pictures\radar.bmp" }
DEPICTIONS { MENU "C:\Program
Files\IBM\pictures\radartoolbar.bmp" }}
```

Vous pouvez également utiliser le mot-clé DEPICTIONS dans une liste, pour que le symbole soit représenté différemment en fonction de la valeur sélectionnée dans la liste.

Exemple :

```
List "Stéréotypes de classe"
{
Value "acteur"DEPICTIONS {DIAGRAM images\slctact.wmf MENU
images\slctact.bmp}
Value "limite"DEPICTIONS { DIAGRAM images\slctbndy.wmf MENU
images\slctbndy.bmp}
..}
```

DEFINITION “Classe” {
PROPERTY "Stéréotype" { EDIT Text LIST "Stéréotypes de classe" INIT_FROM_SYMBOL Default "" LENGTH 20 } ..}

DIAGRAM

La commande DIAGRAM est utilisée de deux manières différentes.

Spécification des propriétés d'un diagramme :

La commande DIAGRAM est utilisée comme premier mot d'un bloc dans lequel les propriétés d'un diagramme, contrairement à DEFINITION ou SYMBOL, sont répertoriées.

Exemple :

```
DIAGRAM "Classe Booch"
 { PROPERTY "Nom fichier DGX"
   { EDIT Text LENGTH 255 }
 PROPERTY "Notes"
   { EDIT Text LENGTH 4000 }
   }
```

Voir aussi les mots-clés DEFINITION et SYMBOL.

Utilisée avec la commande DEPICTIONS :

Référence le graphique utilisé pour représenter un symbole dans l'espace de travail du diagramme, par opposition à la barre d'outils ou au menu Tracer.

Exemple :

```
SYMBOL "Satellite"
{ASSIGN To "Réseau sans fil"
DEPICTIONS { DIAGRAM "C:\Program
Files\IBM\pictures\satellite.bmp" }
DEPICTIONS { MENU "C:\Program
Files\IBM\pictures\satellitetoolbar.bmp" }}
```

DISPLAY

Permet de rendre affichable une propriété et sa valeur sur un symbole de diagramme. La limite est de 37 instructions d'affichage par définition.

La syntaxe est la suivante :

DISPLAY { FORMAT [STRING | LIST | KEY | NONKEY | COMPONENT_SCRIPT | COLUMN_SCRIPT | SCRIPT] **LEGEND "** (libellé du bloc dans le symbole) **" }**

Vous pouvez spécifier l'un des mots-clés FORMAT suivants :

STRING : Affiche les valeurs de la propriété sur le symbole exactement telles que saisies. Voir le mot clé STRING pour un exemple.

LIST : Affiche les éléments sous forme de liste sur le symbole – chaque espace génère une nouvelle ligne, sauf s'il figure entre guillemets. Voir le mot clé LIST pour plus d'informations.

KEY : Ce mot clé est utilisé pour les propriétés désignées comme clés. Elles sont affichées dans une section distincte du symbole. Voir le mot clé KEY pour un exemple et pour de plus amples informations.

NONKEY : Vous pouvez utiliser ce mot-clé pour des propriétés autres que des clés. Elles seront affichées dans une section distincte du symbole. Ce mot clé était utilisé à l'origine pour les entités et les tables dans le support de modélisation de Rational System Architect. Voir le mot clé NONKEY pour un exemple.

COLUMN_SCRIPT : Voir le mot clé COLUMN_SCRIPT.

COMPONENT_SCRIPT : Voir le mot clé COMPONENT_SCRIPT.

SCRIPT :. Voir le mot clé SCRIPT.

DISPLAY (suite)

A l'intérieur des guillemets qui suivent le mot-clé LEGEND, vous indiquez comment le bloc est libellé dans le symbole. Les options possibles sont les suivantes :

LEGEND “<Votre texte>” : Le texte que vous placez entre guillemets ne sera affiché sur le symbole au dessus de l'entrée que si cette dernière possède une valeur.

LEGEND "" : Affiche une ligne droite sans texte et ce, uniquement si l'entrée possède une valeur.

LEGEND "$$FORCE$$" : Affiche une ligne horizontale au dessus de l'entrée sur le symbole. Cette ligne sert de diviseur. Le mot clé “$$FORCE$$” est différent d'une simple utilisation de “ “ en ce qu'il force l'affichage d'une ligne horizontale même si l'affichage de la propriété est supprimé via la boîte de dialogue du mode d'affichage.

LEGEND “$$NONE$$” : N'affiche pas de ligne horizontale au dessus de l'entrée sur le symbole, que l'entrée possède des valeurs ou non. Cette ligne sert normalement de diviseur.

LEGEND “$$VFORCE$$” : Permet de disposer les propriétés de gauche à droite à l'intérieur des symboles et trace des lignes verticales entre-elles. Voir le mot clé VFORCE.

LEGEND “$$VNONE$$” : Permet de disposer les propriétés de gauche à droite, mais ne fournit *pas* de ligne de division. Voir le mot clé VNONE.

Exemple :

```
DEFINITION "Entité organisationnelle"
  { PROPERTY "Nom titulaire"
    { EDIT Text LENGTH 100  HELP "Nom de la personne
          occupant actuellement le poste"
  DISPLAY { FORMAT String LEGEND "" } }
```

EDIT

Utilisé avec le mot-clé BEGIN (ou {), indique le début de la définition d'une propriété. Le mot clé EDIT signifie "Il s'agit de l'argument de début".

Exemple :

```
SYMBOL "Processus" IN "Flux de données Gane & Sarson"
  { PROPERTY "Description abrégée"
    { EDIT Text LENGTH 1500 }
  PROPERTY "Nombre" { EDIT Numeric LENGTH 4 }
```

EDIT COMPLETE

Voir le mot clé COMPLETE.

EDITCLASS

N'utilisez pas ce mot clé. Il s'agit d'un mot-clé spécial développé spécifiquement pour une certaine situation dans Rational System Architect, héritée de propriétés Elément de données par un attribut dans une entité. Cette commande est utilisée dans SAPROPS.CFG pour ce cas précis. C'est le seul cas dans lequel ce mot-clé peut être appliqué. Son utilisation dans d'autres cas peut générer des erreurs.

EDIT URLS

Vous pouvez spécifier qu'une propriété listof est désignée comme pouvant référencer des documents externes. Cette commande provoque l'affichage de divers boutons au bas de la propriété listof
les boutons Ouvrir, Parcourir en externe et Parcourir en interne. Vous pouvez utiliser ces boutons pour parcourir et sélectionner des documents externes, ou saisir des hyperliens externes, ou parcourir la table Files interne de la base de données de l'encyclopédie, ou ouvrir un document externe ou interne référencé.

Syntaxe :

PROPERTY property name { **EDIT URLS** }

Restrictions :

SA applique les restrictions ci-après sur la propriété URLS : il ne peut pas s'agir d'une clé.

Exemple :

```
Definition "Cas d'utilisation"
{
PROPERTY "Documents de référence" { EDIT URLS }
}
```

END

Indique la fin de la spécification d'une propriété ou du groupe de propriétés qui composent la définition d'un diagramme, d'un symbole ou d'une définition. L'instruction END est associée à l'instruction BEGIN pour délimiter la spécification. A la place des instructions BEGIN et END, vous pouvez utiliser des accolades ouvrantes et fermantes, **{ }**

Exemple :

PROPERTY “<nom_propriété>”
BEGIN EDIT <type_édition> <paramètre_propriété>
END

EXPRESSION

Indique que la valeur de la définition doit être entrée comme une série de chaînes séparées par un signe + ou un espace.

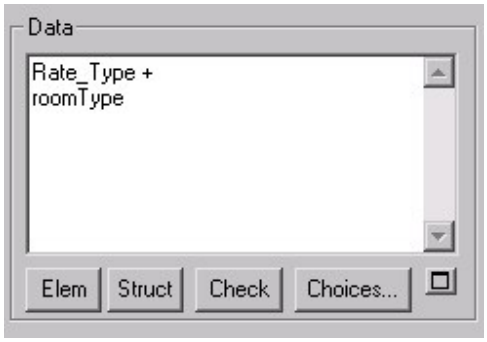
Exemple :

VendorName +
VendorCity +
VendorState

Voir aussi le mot clé EXPRESSIONOF qui a remplacé celui-ci.

EXPRESSIONOF

EXPRESSIONOF permet d'exprimer des références aux objets à l'aide d'opérateurs et de délimiteurs complexes. EXPRESSIONOF est généralement utilisé avec l'argument spécial DATA, lequel fait référence aux éléments de données et aux structures de données – En d'autres termes, EXPRESSIONOF DATA. Cette combinaison de mots clés produit une zone de texte dans laquelle les valeurs de définition sont saisies comme une série de chaînes. La séparation entre une valeur de définition et la suivante est déterminée par un espace. Par convention, un signe + est utilisé pour séparer chaque valeur de définition, mais cette pratique n'est pas obligatoire.



Exemple :

```
DEFINITION "Indicateur de contrôle"
{
PROPERTY "Description"
{ EDIT EXPRESSIONOF DATA LENGTH 4074 LABEL "Données" }
}
```

Voir aussi les mots clés ONEOF et LISTOF, et reportez-vous dans le chapitre 2 à la section traitant de ExpressionOf pour plus d'informations et pour consulter la liste des opérateurs et des délimiteurs pouvant être utilisés.

fmtxxx ou_fmtxxx

Ces deux préfixes de nom sont utilisés, par convention, au début du nom d'une fonction appelée par les mots clés SCRIPT, COLUMN_SCRIPT ou COMPONENT_SCRIPT. La fonction elle-même (par exemple, _fmtUMLAttr) fournit généralement l'affichage d'une valeur de propriété (par exemple, un attribut et toutes ses propriétés) d'après un format spécial sur le symbole. La convention de dénomination est la suivante :

- _fmt (par exemple, _fmtUMLAttr) : la fonction elle-même est codée en dur et ne peut pas être modifiée. C'est le cas de la plupart des fonctions contenues dans SAPROPS.CFG. Le codage en dur de la fonction est effectué pour accélérer la réponse globale de Rational System Architect.
- fmt (par exemple, fmtUMLAttr) : existe dans le fichier fmtscript.bas du répertoire principal des exécutables de Rational System Architect.

Exemple :

DEFINITION “Classe”
PROPERTY "Attributs" {ZOOMABLE EDIT COMPLETE ListOf "Attribut de classe" KEYED BY {"Package", "Nom de la classe":"Nom", Name } LENGTH 4096 ASGRID DISPLAY { FORMAT COMPONENT_SCRIPT **_FmtNewUMLAttr** LEGEND "$$FORCE$$"} LABEL "Attributs" }

Dans l'exemple ci-dessus, un script est appelé pour afficher les attributs d'un diagramme de classe d'une certaine manière (par exemple, si la propriété d'accès de l'attribut est définie à ‘public’, un signe ‘+’ est placé avant l'attribut sur le symbole de classe, etc.

Création de vos propres fonctions

Pour créer vos propres fonctions, reportez-vous au mot-clé SCRIPT.

Voir aussi les mots clés DISPLAY, FORMAT, SCRIPT, COLUMN_SCRIPT, COMPONENT_SCRIPT et VALUESCRIPT.

FORCE

Il s'agit en réalité du mot clé $$FORCE$$, utilisé avec le mot clé DISPLAY.

Pour plus d'informations, voir le mot clé DISPLAY.

FORMAT

Indique comment les données doivent être présentées pour une propriété affichable spécifique.

Exemple :

```
PROPERTY "Description"
  { EDIT ExpressionOf "Données"
  Display { FORMAT List LEGEND "Données" } }
```

Reportez-vous au mot clé DISPLAY pour plus d'informations.

FREEZE

Affiche des informations sur le gel d'un objet, telles que l'AUDIT ID de la personne l'ayant gelé ou les informations DATE ou TIME (heure) de l'opération. Les zones affichées sont toujours en LECTURE SEULE. Un suivi des valeurs est automatiquement effectué par Rational System Architect, mais pour les afficher dans une boîte de dialogue, vous devez ajouter des propriétés avec les caractéristiques respectives suivantes :

```
FREEZE Auditid
FREEZE Date
FREEZE Time
```

Exemple :

```
DIAGRAM "Flux de données Gane & Sarson"
  {
    PROPERTY "Gelé par"
  { FREEZE Auditid }
    PROPERTY "Date du gel"
  { FREEZE Date }
    PROPERTY "Heure du gel"
  { FREEZE Time }
  }
```

Pour plus d'informations sur le gel d'objets, effectuez une recherche sur *Contrôle d'accès* dans l'aide en ligne.

Voir aussi le mot clé CHECKOUT.

FROM_CHOICES_ONLY

Restreint l'utilisateur à la sélection d'une définition dans une liste d'options sans qu'il puisse saisir une nouvelle définition. Une zone de message invite l'utilisateur à sélectionner uniquement une option dans la liste "Choix". Utilisé avec ListOf et OneOf.

Par exemple :

```
DEFINITION "Produit"
{
CHAPTER "Modèle de référence technique"
PROPERTY "Statut" {Zoomable EDIT Oneof "Statut du produit"}
Group "Implications"{
LAYOUT { COLS 2 ALIGN OVER   }
PROPERTY "Proposition dominante" {Zoomable EDIT Oneof "Unité organisationnelle"
FROM_CHOICES_ONLY}
PROPERTY "Autres impliqués" {Zoomable EDIT Listof "Unité organisationnelle" FROM_CHOICES_ONLY}
}
}
```

GROUP

Permet de générer une zone de groupe avec des paramètres de présentation spécifiques, tels qu'une série de boutons d'option, dans laquelle se trouvent plusieurs propriétés.

Exemple 1 :

```
GROUP "Intégrité référentielle"
 {
   LAYOUT { ALIGN OVER TAB COLS 3 }
     PROPERTY "Suppression parent"
 { EDIT Text LISTONLY LIST RDC
     LENGTH 15 }
 …
 }  REM "Fin de l'intégrité référentielle du groupe"
```

Vous ne pouvez pas modifier un nom GROUP qui est déjà prédéfini par Rational System Architect dans le fichier SAPROPS.

Exemple 2 :

```
DEFINITION "Attribut de classe"
 { CHAPTER "Classe, Données source, Desc."
    GROUP "Données source" {
 LAYOUT { COLS 2 ALIGN OVER TAB }PROPERTY
       "Description"
 { EDIT Text LENGTH 1500 }
```

A la troisième ligne de l'exemple ci-dessus, si vous essayez de modifier 'GROUP "Données source"' en 'GROUP "Données d'origine" dans le fichier USRPROPS.TXT, votre modification sera sans effet. Le texte contenu dans l'entrée SAPROPS GROUP, "Données source" ne sera pas remplacé. Il demeurera le texte d'affichage du groupe Attribut de classe.

HELP

Cette chaîne est affichée sur la ligne de statut dans le ***coin inférieur gauche*** d'une boîte de dialogue **Diagramme** ou **Définition** lorsqu'une propriété donnée est sélectionnée.

Syntaxe :

HELP "<chaîne_texte>"

Exemple :

```
PROPERTY Length
  { EDIT Numeric LENGTH 2 MIN 1 MAX 99
    HELP "Longueur de cette zone"
  }
```

HETEROGENEOUS (ONEOF, LISTOF)

Permet à une propriété de faire référence à des définitions de types différents. (Une liste normale fait référence à des définitions d'un seul type.) Le mot-clé HETEROGENEOUS permet de modifier le mot-clé ONEOF ou LISTOF.

Par exemple, lorsque vous cliquez sur le bouton Choix d'une liste de classes, seules des définitions de “classe” peuvent être sélectionnées. Si vous cliquez sur le bouton Choix d'une liste hétérogène, les divers types de définition que vous avez spécifiés dans la clause de liste Heterogeneous, tels que "classe", "processus", “entité”, etc., y figurent.

Syntaxe du mot-clé ONEOF :

PROPERTY *property name* **{ EDIT HeterogeneousOneOf [** *class* **]** *type-1* **{ [,** *type-n* **] } … .etc. }**

Syntaxe du mot-clé LISTOF :

PROPERTY *property name* **{ EDIT HeterogeneousListOf [** *class* **]** *type-1* **{ [,** *type-n* **] } … .etc. }**

Restrictions

Certaines restrictions s'appliquent à une propriété de liste Heterogeneous. Il ne peut pas prendre les valeurs suivantes (en d'autres termes, vous ne pouvez pas utiliser l'un des mots-clés suivants avec le mot-clé HETEROGENEOUSLISTOF ou HETEROGENEOUSONEOF dans une même propriété) :

- La propriété ne peut pas être KEY.
- Elle ne peut pas contenir une clause KEYED BY.
- Elle ne peut pas être COMPLETE.
- Elle ne peut pas contenir une clause FROM.
- Elle ne peut pas être ASGRID.
- Elle ne peut pas avoir de valeur DEFAULT.
- Elle ne peut pas être INITIAL USER REQUIRED.
- Elle ne peut pas contenir de clause de restriction (REFERENCED IN ou WHERE).
- Elle ne peut pas avoir l'attribution INIT_FROM_SYMBOL.
- Aucun nom de type ne peut être répertorié plusieurs fois

Ajout de nouvelles valeurs à la liste

La plupart du temps, les utilisateurs doivent faire glisser des valeurs dans une liste Heterogeneous à l'aide du navigateur Sélectionner et faire glisser fourni en cliquant sur le bouton Choix, mais les utilisateurs peuvent ajouter de nouvelles valeurs à la liste hétérogène. Toutefois, pour ajouter de nouvelle valeur dans une liste Heterogeneous, les utilisateurs doivent l'entrer avec son nom qualifié complet, au format suivant :

ClassName:TypeName:FullyQualifiedName

Où :

- ClassName correspond aux types de classe des encyclopédies System Architect – Diagramme, Symbole ou Définition.
- TypeName correspond au nom spécifique du type Diagramme, Symbole ou Définition, tel que Classe (définition) ou Etape de cas d'utilisation (définition).
- Chaque partie du nom FullyQualifiedName est séparée par un point. Ainsi, par exemple, une étape de cas d'utilisation, indexée par son cas d'utilisation, lui-même indexé par son module, doit être entrée comme suit :

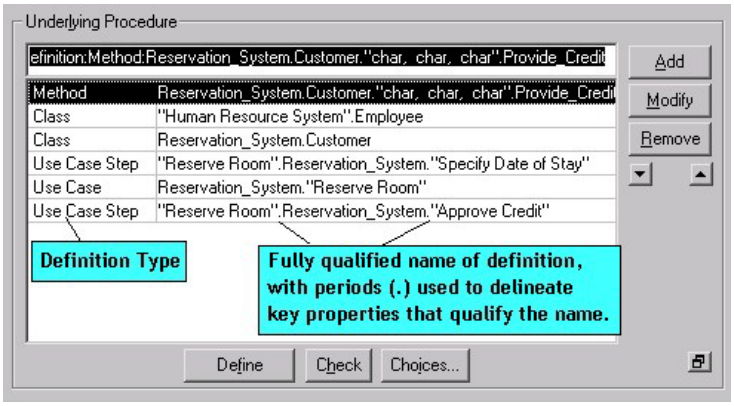
  Definition:”Etape de cas d'utilisation”:”Nom du package”.”Nom du cas d'utilisation”.”Nom de l'étape du cas d'utilisation”

Exemple :

```
Definition " Procédure"
{
PROPERTY "Procédure sous-jacente" { EDIT
HETEROGENEOUSLISTOF " Cas d'utilisation","Classe", "Méthode",
"Etape de cas d'utilisation" READONLY}
```

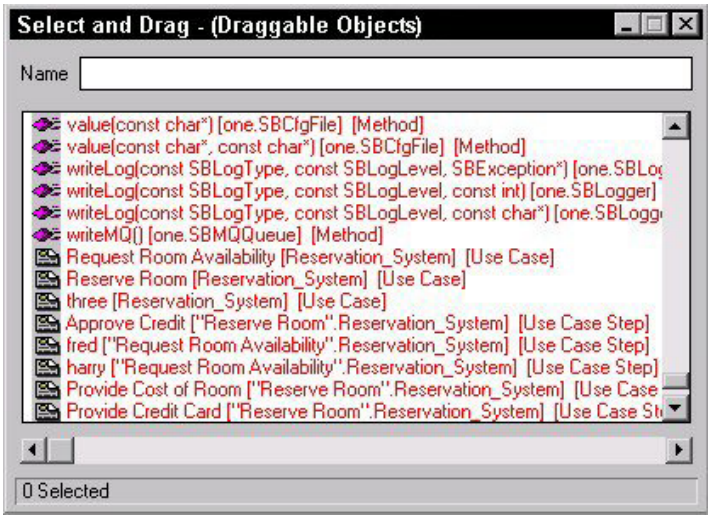
Dans l'exemple ci-dessus, la propriété “Procédure sous-jacente” de la définition “Procédure” peut être alimentée avec les définitions du type Cas d'utilisation, Classe, Méthode et Etape du cas d'utilisation.

L'interface graphique fournie par le mot-clé HETEROGENEOUSONEOF ou HETEROGENEOUSLISTOF affiche une colonne qui contient le nom de chaque type de définition et le nom qualifié complet de la définition particulière que vous avez fait glisser dans la liste.

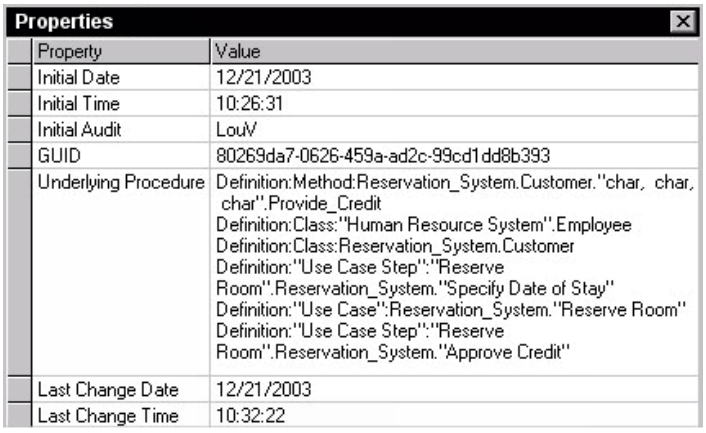


Les propriétés de clé qui qualifient le nom d'une définition sont fournies dans l'interface graphique et sont séparées les unes des autres par des points (.). Par exemple, une étape de cas d'utilisation est indexée au cas d'utilisation qui la contient, qui est lui-même indexé par le module qui le contient. Dans la zone HETEROGENEOUSONEOF ou HETEROGENEOSLISTOF, une étape de cas d'utilisation est représentée par “Nom du package”.”Nom du cas d'utilisation”.”Nom de l'étape du cas d'utilisation”. Si le nom d'un élément contient un ou plusieurs espaces, cet élément est placé entre guillemets. Par exemple, dans l'image ci-dessus, l'étape de cas d'utilisation “Approuver le crédit” se trouve dans le cas d'utilisation Système_réservations, lequel appartient au module “Réserver une chambre”.

Lorsque vous cliquez sur le bouton Choix d'une liste Heterogeneous, tous les types de diagramme, de symbole ou de définition sont représentés et leur type est répertorié entre crochets après leur nom.



La fenêtre Propriétés présente également les valeurs d'une liste hétérogène. Vous pouvez faire glisser les bordures des lignes ou des colonnes des propriétés pour afficher toutes les valeurs. Chaque valeur est précédée de son type de classe (diagramme, symbole ou définition), et du nom de type (par exemple, définition Etape du cas d'utilisation), suivis de la valeur elle-même.



Property	Value
Initial Date	12/21/2003
Initial Time	10:26:31
Initial Audit	LouV
GUID	80269da7-0626-459a-ad2c-99cd1dd8b393
Underlying Procedure	Definition:Method:Reservation_System.Customer."char, char, char".Provide_Credit Definition:Class:"Human Resource System".Employee Definition:Class:Reservation_System.Customer Definition:"Use Case Step":"Reserve Room".Reservation_System."Specify Date of Stay" Definition:"Use Case":Reservation_System."Reserve Room" Definition:"Use Case Step":"Reserve Room".Reservation_System."Approve Credit"
Last Change Date	12/21/2003
Last Change Time	10:32:22

HIDE DEFINITION

Supprime le type de définition référencé des boîtes de dialogue **Nouvelle définition** et **Ouvrir définition**.

Syntaxe :

HIDE DEFINITION <nom de la définition>

Exemple :

HIDE DEFINITION "Table SQL Server"

AVERTISSEMENT : Soyez prudent lorsque vous masquer des définitions, surtout si elles sont utilisées par des symboles que vous avez activés à l'aide de la boîte de dialogue de configuration des propriétés (Outils, Personnaliser le support des méthodes). Vous pouvez vous trouver dans une situation où vous tracez des symboles sans définitions sous-jacentes.

HIDE DIAGRAM

Supprime le type de diagramme référencé des boîte de dialogue **Nouveau diagramme** et **Ouvrir diagramme**.

Syntaxe :

HIDE DIAGRAM <nom du diagramme>

Exemple :

HIDE DIAGRAM "Processus Booch"

Remarque : Au lieu d'utiliser ce mot clé, une modification moins radicale consiste simplement à désélectionner le type de diagramme de la boîte de dialogue **Configuration des propriétés** (sélectionnez Outils, Personnaliser le support des méthodes, Configuration de l'encyclopédie et désactivez la méthode employant le type de diagramme ou cliquez sur le bouton Avancé de la boîte de dialogue Configuration des propriétés et transférez le type de diagramme de la liste “Diagrammes sélectionnés” à la liste “Diagrammes disponibles”).

HIERARCHICAL

Par défaut, les diagrammes utilisateur sont des réseaux (de symboles), mais si ce mot clé est inclus dans la description d'un type de diagramme, ce dernier est traité comme un diagramme hiérarchique. Cela signifie que tous les symboles de noeud qui lui sont affectés pourront être organisés dans une hiérarchie et que les autres fonctionnalités hiérarchiques associées (telles que la numérotation hiérarchique) sont prises en charge.

Le mot clé HIERARCHICAL ne peut être utilisé qu'avec des types de diagramme définis par l'utilisateur ; il ne peut pas être appliqué aux types de diagramme existants. Dans tout autre contexte, il est ignoré après un avertissement envoyé à l'utilisateur. (Pour des informations sur la manière de créer un type de diagramme défini par l'utilisateur, voir le mot-clé RENAME DIAGRAM.)

Exemple :

```
RENAME DIAGRAM “Utilisateur 1” to “Zoo”
RENAME SYMBOL "Utilisateur 1" to "Mammifères"
RENAME SYMBOL "Utilisateur 2" to "Reptiles"
RENAME DEFINITION “Utilisateur 1” to “Mammifères”
RENAME DEFINITION “Utilisateur 2” to “Reptile”

SYMBOL "Mammifères"
{DEFINED by "Mammifères"
ASSIGN TO "Zoo"}

SYMBOL "Reptiles"
{DEFINED by "Reptile"
ASSIGN TO "Zoo"}

DIAGRAM "Zoo"
{HIERARCHICAL
PROPERTY "Numérotation hiérarchique"
{ EDIT Boolean LENGTH 1 DEFAULT "T" }
PROPERTY "Numéro du premier noeud"
{ EDIT Text  Length 20 DEFAULT "1" }
}
```

Le diagramme créé par le fichier USRPROPS.TXT ci-dessus sera de nature hiérarchique – il sera similaire à un organigramme, etc.

IFDEF

Voir la commande #IFDEF.

IFNDEF

Voir la commande #IFNDEF.

IN

Etablit le contexte pour la commande **RENAME** lorsqu'il est appliqué à un symbole. Peut également être utilisé pour **DEPICT LIKE**.

Exemples :

Par exemple, pour renommer un symbole Application :

```
#ifdef "Business Enterprise"
RENAME SYMBOL "Application" IN "Architecture du système"
TO "Mon symbole"
#endif
```

Pour donner à un diagramme l'apparence d'un autre dans un diagramme différent :

```
#ifdef "Business Enterprise"
SYMBOL "Système" IN "Contexte système"
{
DEPICT LIKE "Processus" IN "Flux de données Gane &
Sarson"
}
#endif
```

INCLUDE

Voir #INCLUDE.

INITIAL

Permet d'horodater un diagramme, un symbole ou une définition avec les données AUDIT ID, DATE et TIME de sa création. La valeur de cette zone n'est jamais modifiée par Rational System Architect.

Variantes :

INITIAL DATE

INITIAL TIME

INTIAL AUDITID

A partir de Rational System Architect V9, INITIAL DATE, INITIAL TIME et INITIAL AUDITID figurent par défaut sous l'onglet Données d'accès de chaque boîte de dialogue Diagramme ou Définition. Ils sont codés en dur dans le produit. En d'autres termes, vous ne trouverez pas le mot clé INITIAL dans chaque définition dans SAPROPS.CFG et vous n'avez pas besoin de l'ajouter dans USRPROPS.TXT pour les nouveaux types de diagramme ou de définition que vous créez.

Exemple :

```
DEFINITION "X"
{ PROPERTY "Auditid de création"
{ EDIT Text INITIAL AUDITID LENGTH 12 READONLY }
}
```

Voir aussi le mot-clé UPDATE.

INIT_FROM_SYMBOL

Le mot-clé INIT_FROM_SYMBOL est utilisé dans une définition qui définit un symbole. Il spécifie qu'une propriété de la définition hérite initialement de sa valeur d'une propriété de nom similaire dans le symbole. Cela est utilisé lorsqu'une propriété doit exister à la fois dans le symbole et la définition et qu'elle doit posséder la même valeur. Cela est par exemple nécessaire lors de la spécification de métafichiers fournis par l'utilisateur pour un symbole, en fonction d'une propriété telle que Stéréotype. Le stéréotype doit être spécifié pour le symbole (car c'est ce qui indique comment le symbole est représenté dans le diagramme) et dans la définition correspondante.

Exemple 1 :

```
LIST "Stéréotypes de classe"
{
VALUE "acteur" DEPICTIONS {diagram images\slctact.wmfmenu
images\slctact.bmp}
VALUE "limite" DEPICTIONS {diagram images\slctbndy.wmfmenu
images\slctbndy.bmp}
VALUE "Travailleur social" DEPICTIONS {diagram
images\slctcwkr.wmfmenu images\slctcwkr.bmp}
}

SYMBOL "Classe" in "Classe"
{
PROPERTY "Stéréotype" { INVISIBLE EDIT Text ListOnly List
"Stéréotypes de classe" DEFAULT "" LENGTH 20}..}

DEFINITION “Classe”
{
PROPERTY "Stéréotype" { EDIT Text LIST "Stéréotypes de classe"
INIT_FROM_SYMBOL Default "" LENGTH 20 } ..}
```

Dans l'exemple ci-dessus, la propriété Stéréotype est déclarée à la fois dans la spécification du symbole de classe et dans la définition de classe. Elle doit posséder la même valeur. La propriété stéréotype de SYMBOL permet d'afficher sous forme de menu déroulant les valeurs de stéréotype possibles dans le menu Tracer de Rational System Architect pour leur sélection (ces valeurs sont elles-mêmes des fichiers bitmap spécifiés par la clause DEPICTIONS dans l'instruction LIST). Une fois que vous avez sélectionné une classe stéréotypée dans cette liste depuis la barre d'outils Tracer et que vous l'avez placée dans le diagramme, la définition de la classe est créée et sa propriété stéréotype est automatiquement renseignée par le stéréotype que vous avez choisi pour le symbole. Notez que si vous modifiez cette valeur dans la définition, elle l'est également dans le symbole.

INIT_FROM_SYMBOL (suite)

Notez également que cette valeur de stéréotype n'est pas visible dans l'onglet de symbole de la classe car elle a été rendue INVISIBLE.

Exemple 2 :

```
DIAGRAM “Classe”
{
PROPERTY "Langage de programmation"{ EDIT Text Listonly LIST
"Langages de programmation" Default "CORBA" LENGTH 30 INITIAL
USER REQUIRED }
}
SYMBOL "Classe" in "Classe"
{
PROPERTY "Package" { EDIT OneOf "Package" READONLY }
PROPERTY "Stéréotype" { INVISIBLE EDIT Text ListOnly List
"Stéréotypes de classe" DEFAULT "" LENGTH 20}
PROPERTY "Langage de programmation" { INVISIBLE EDIT Text
ListOnly List "Langages de programmation" DEFAULT "" LENGTH 30}
}
DEFINITION “Classe”
{
PROPERTY "Package" { KEY EDIT OneOf "Package" RELATE BY
"indexé par" READONLY}
PROPERTY "Stéréotype" { EDIT Text LIST "Stéréotypes de classe"
INIT_FROM_SYMBOL Default "" LENGTH 20 }
PROPERTY "Langage de programmation"{ EDIT Text ListOnly LIST
"Langages de programmation" INIT_FROM_SYMBOL Default
"CORBA" LENGTH 30 INITIAL USER REQUIRED READONLY }
}
```

Dans l'exemple ci-dessus, la propriété Langage de programmation existe dans le diagramme et le symbole Classe hérite de la valeur de cette propriété du diagramme. La définition du symbole Classe hérite également de la valeur de cette propriété par l'intermédiaire du symbole, en raison du mot clé INIT_FROM_SYMBOL.

Si une définition Classe est créée via l'explorateur, la propriété requise DOIT être fournie lors de sa création en raison du mot clé INITIAL USER REQUIRED dans la définition Classe.

INITIAL USER REQUIRED

Ce mot clé indique que lors de la création de l'élément de modélisation (diagramme, symbole ou définition), une valeur **doit** être fournie pour la propriété. Si vous l'omettez et tentez de fermer la boîte de dialogue en cliquant sur OK, Rational System Architect affiche le message suivant : “La propriété xxx doit être fournie”. Vous ne pourrez pas cliquer sur OK pour fermer la boîte de dialogue et créer le diagramme, le symbole ou la définition. Vous devrez fournir une valeur pour la propriété ou annuler la boîte de dialogue.

Exemple :

```
DIAGRAM "Activité"
{
PROPERTY "Package" { EDIT OneOf "Package" RELATE BY "fait partie de" INITIAL USER REQUIRED OVERRIDABLE }
PROPERTY "Modèle d'activité" { EDIT OneOf "Modèle d'activité" ReadOnly INITIAL USER REQUIRED }  ..}
```

Dans l'exemple ci-dessus, vous devez renseigner les propriétés “Package” et “Modèle d'activité” avant de pouvoir cliquer sur le bouton OK dans la boîte de dialogue du diagramme lorsque vous créez un diagramme Activité.

Notez que dans l'exemple ci-dessus, la propriété “Package” est également définie comme OVERRIDABLE (remplaçable) alors que la propriété “Modèle d'activité” ne l'est pas. Le mot-clé OVERRIDABLE n'a une signification que pour les symboles tracés sur ce diagramme qui héritent des valeurs de la propriété à partir du diagramme.

Exemple 2 :

DIAGRAM “XML”
{..
PROPERTY "Schéma XML" { Edit OneOf "XML Schema" **AUTOCREATE** Relate By "fait partie de" **INITIAL USER REQUIRED OVERRIDABLE READONLY** } ..}

Dans l'exemple ci-dessus, le mot-clé READONLY utilisé avec le mot clé INITIAL USER REQUIRED spécifie que la boîte de dialogue ne peut pas être fermée à moins qu'une valeur ne soit entrée par l'utilisateur pour cette propriété et, qu'une fois que la valeur initiale a été fournie, la propriété passe en lecture seule et ne peut pas être modifiée par l'utilisateur. Le mot-clé OVERRIDABLE n'a une signification que pour les définitions dont cette valeur de propriété provient du diagramme.

INITIAL USER REQUIRED (suite)

Le mot clé INITIAL USER REQUIRED impose donc qu'une valeur soit fournie pour la propriété Schéma XML lors de la création du diagramme. Le mot-clé AUTOCREATE crée automatiquement une définition pour toute valeur entrée dans cette propriété. Par conséquent, lorsque l'utilisateur clique sur OK pour fermer la boîte de dialogue Diagramme, une définition Schéma XML définie est créée.

Voir aussi les mots clés OVERRIDABLE, READONLY et AUTOCREATE.

INVISIBLE

Rend une propriété non visible dans la boîte de dialogue graphique, sans la supprimer. Les propriétés non visibles sont utilisées lorsqu'une propriété est requise pour une définition, mais qu'elle n'a aucune signification pour l'utilisateur.

Exemple :

```
SYMBOL "Classe" in "Classe"
{
PROPERTY "Package" { EDIT OneOf "Package" READONLY }
PROPERTY "Stéréotype" { INVISIBLE EDIT Text ListOnly List
"Stéréotypes de classe" DEFAULT "" LENGTH 20}
PROPERTY "Langage de programmation" { INVISIBLE EDIT Text
ListOnly List "Langages de programmation" DEFAULT "" LENGTH 30}
}

DEFINITION “Classe”
{
PROPERTY "Package" { KEY EDIT OneOf "Package" RELATE BY
"indexé par" READONLY}
PROPERTY "Stéréotype" { EDIT Text LIST "Stéréotypes de classe"
INIT_FROM_SYMBOL Default "" LENGTH 20 }
PROPERTY "Langage de programmation"{ EDIT Text ListOnly LIST
"Langages de programmation" INIT_FROM_SYMBOL Default
"CORBA" LENGTH 30 INITIAL USER REQUIRED READONLY }
}
```

Dans l'exemple ci-dessus, la propriété Stéréotype est utilisée avec le symbole et la définition d'une classe. Ils doivent correspondre. Les utilisateurs peuvent choisir la propriété de stéréotype dans la classe et cette valeur est automatiquement affectée au symbole, où elle permet de déterminer comment le symbole est affiché. Toutefois, l'utilisateur n'a pas besoin d'afficher la propriété Stéréotype dans la page Symbole de la définition Classe car elle se trouve déjà dans la boîte de dialogue de la définition Classe. Si elle se trouvait aux deux emplacements, cela prêterait à confusion. Elle est rendue invisible.

Voir aussi le mot clé VISIBLE.

JUSTIFY

Cette commande **n'est plus utilisée** dans SAPROPS.CFG ou USRPROPS.TXT. Elle ne génère pas d'erreur si elle est spécifiée dans USRPROPS.TXT ; elle sera simplement **ignorée** par l'analyseur syntaxique du fichier. Il s'agissait auparavant de l'un des arguments de la commande LAYOUT. Lorsqu'elle était utilisée, elle alignait tous les contrôles sur les bords des marges droite et gauche de la page de la boîte de dialogue.

Voir aussi les mots clés LAYOUT et ALIGN.

KEY

Le mot clé KEY permet d'établir une propriété comme clé. Les clés permettent de déterminer l'espace de nom des éléments de modélisation dans l'encyclopédie. Le mot clé KEY a également une autre utilisation : c'est l'un des arguments autorisés après le mot clé FORMAT dans la commande DISPLAY. Pour cette utilisation, voir KEY (utilisé pour l'affichage).

Par défaut, chaque élément de modélisation d'une encyclopédie se distingue par sa classe (qu'il s'agisse d'un diagramme, d'un symbole ou d'une définition), son type (diagramme Cas d'utilisation UML, diagramme Processus BPMN, etc) et son nom (par exemple, diagramme Cas d'utilisation Reservation_System versus diagramme Cas d'utilisation Human_Resource_System). En plus de ces valeurs par défaut intégrées, vous pouvez spécifier des clés supplémentaires pour un élément de modélisation de définition (par exemple, une définition d'attribut de classe est indexée par sa définition de classe conteneur et la définition de package conteneur de la classe).

Pour utiliser la commande KEY, spécifiez-le dans la propriété qui doit servir de clé de définition. La commande KEY peut être positionnée pratiquement n'importe où dans la description d'une propriété, mais en raison de son importance, elle est généralement placée comme premier élément, à l'intérieur des accolades de la propriété, juste avant le mot clé EDIT.

Exemple :

```
Definition “Etape de cas d'utilisation”
{
PROPERTY "Nom du cas d'utilisation" { KEY EDIT …  }
PROPERTY "Package" { KEY EDIT …}
…
```

KEY (suite)

Pour une propriété qui correspond à une clé et qui fait référence à d'autres objets (par exemple, une propriété LISTOF ou ONEOF et non une simple propriété TEXT ou NUMERIC), l'utilisateur final doit spécifier la classe et le type de classe des objets référencés lorsqu'il saisit une valeur pour la propriété, dans Rational System Architect.

Par exemple :

Definition “Processus métier”
{
PROPERTY “Cas d'utilisation du système” {EDIT **ONEOF “Cas d'utilisation”** …}

L'instruction ci-dessus indique que la propriété “Nom du cas d'utilisation” se réfère à une définition de type “Cas d'utilisation”. Définition est la valeur par défaut si aucune *classe* n'est spécifiée (*classe* dans le sens Rational System Architect : Diagramme, Symbole ou Définition).
La valeur de la propriété elle-même contient souvent tous les éléments restants nécessaires pour identifier le ou les objets référencés. Si le type/la classe référencé de la propriété ne contient pas de propriétés de clé, la valeur de référence correspond simplement au nom de l'objet (car la classe et le type sont connus), mais si le type/la classe référencé contient des propriétés de clé (telles que “Cas d'utilisation” dans l'exemple ci-dessus, qui contient la propriété clé “package”), Rational System Architect doit connaître les valeurs de ces propriétés clés afin d'identifier correctement l'objet de référence.

Codez cela dans le fichier USRPROPS.TXT pour que Rational System Architect obtienne automatiquement les valeurs de l'utilisateur final ou forcez ce dernier à saisir le nom qualifié complet, en séparant les composants de clé par des points.

- Pour que Rational System Architect extraie automatiquement la valeur des utilisateurs, utilisez la commande KEYED BY.
- Si aucune clause KEYED BY n'est spécifiée pour la propriété, Rational System Architect s'attend à ce que ces valeurs de clé supplémentaires soient spécifiées dans la référence elle-même. En d'autres termes, l'utilisateur doit saisir le nom qualifié complet de l'objet de référence, avec des points séparant les valeurs (dans le cas d'une étape de cas d'utilisation nommée "*Spécifier e-mail"* relatif à un cas d'utilisation intitulé *Commander_produit* dans un package appelé "*Système de commande"*, l'utilisateur devrait saisir "*Système de commande".Commander_produit."Spécifier e-mail"*.

Remarque : Les propriétés de référence hétérogènes diffèrent sur ce plan. Voir HETEROGENEOUS.

KEY (suite)

La clause KEYED BY permet également de générer la liste des choses associées. Par exemple, toutes les étapes de cas d'utilisation auxquelles il est fait référence dans la propriété "Etapes de cas d'utilisation" d'une définition Cas d'utilisation appartiennent au même cas d'utilisation, à savoir, celui contenant la propriété "Etapes de cas d'utilisation". Lorsqu'une propriété à plusieurs références (telle que ListOf) fait référence à des objets appartenant tous au même objet parent, il est conseillé d'utiliser une ou plusieurs autres propriétés pour identifier l'objet parent. Dans ces situations, une clause KEYED BY est utilisée pour indiquer à Rational System Architect les autres propriétés à utiliser.

Remarque : Les propriétés de clé d'une définition ne sont pas affichées dans une grille constituée par une commande ASGRID. Par exemple, dans une définition Cas d'utilisation, les étapes du cas d'utilisation sont représentées dans une grille assemblée par une commande ASGRID, mais les propriétés de clé des étapes du cas d'utilisation (package propriétaire et cas d'utilisation) ne sont pas affichées dans la grille des étapes du cas d'utilisation.

Remarque : il n'est pas possible d'ajouter un mot clé KEY EDIT ONEOF à un diagramme.

Voir aussi le mot clé KEYED BY.

KEYED BY

Une clause KEYED BY est éventuellement utilisée pour spécifier comment les composants clé d'un ou plusieurs objets référencés peuvent être trouvés. La clause KEYED BY contient une portion pour chaque composant clé, séparée par une virgule.

La clause KEYED BY offre deux avantages :

1. L'utilisateur n'a plus besoin de saisir le nom complet d'une valeur de référence (avec des points séparant les qualificateurs). Par exemple, pour une propriété qui référence un attribut de classe nommé ***e-mail*** de la classe ***Client*** du package ***"Système de commande"***, au lieu de saisir ***"Système de commande".Client.e-mail***, l'utilisateur saisit simplement ***e-mail***.
2. Elle permet de s'assurer que tous les composants clé d'une valeur de référence sont identiques. Par exemple, la propriété LISTOF "Attribut de classe" d'une définition Classe contient une liste de tous les attributs qui appartiennent à la même classe et au même package.

Exemple :

Par exemple, la clause KEYED BY de la propriété "Attribut de classe" de la classe pourrait se présenter comme suit :

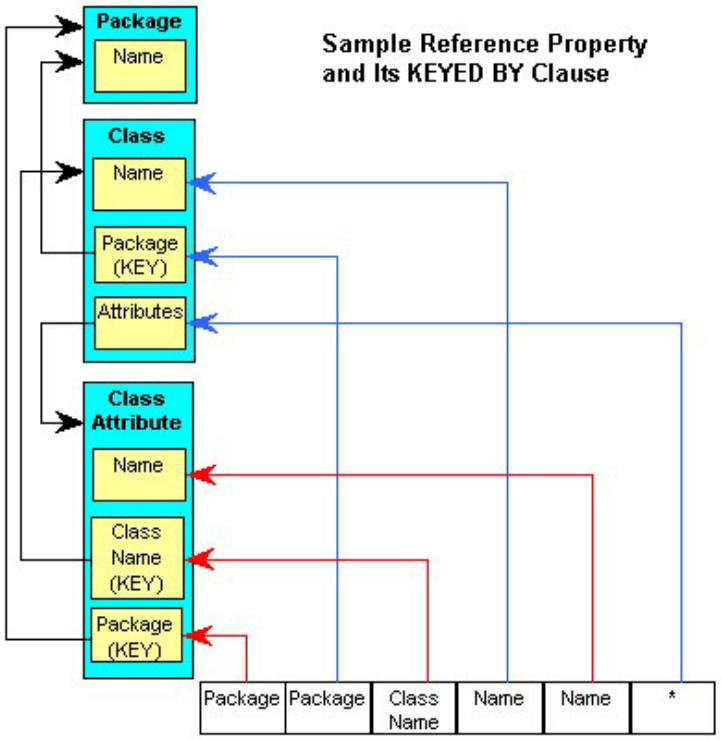
```
DEFINITION "Classe"
{
…
PROPERTY "Attributs" { … LISTOF "Attributs de classe"
KEYED BY {Package:Package, "Nom de la classe":Name, Name:* } …
}
```

Dans l'exemple ci-dessus, les trois *composants clés* (séparés par des virgules) sont Package:Package, "Nom de la classe":Name et Name:*. Ces composants font référence aux trois parties requises pour identifier les définitions Attribut de classe référencées : Nom du package, Nom de la classe et Nom de classe d'attribut. En ordre inverse, il indique que :

- Le nom de l'attribut de classe est situé dans ***cette*** propriété (* signifie "ici"), et donc : **Name:***
- La valeur de la propriété clé "Nom de la classe" dans la définition Attribut de classe est située dans le nom de cet objet, et donc : **"Nom de la classe":Name**
- La valeur de la propriété clé Package dans la définition Attribut de classe est situé dans la propriété Package de cet objet, et donc : **Package:Package**

KEYED BY (suite)

Le schéma suivant illustre comment la clause KEYED BY est utilisée dans l'exemple ci-dessus et peut permettre de comprendre la clause KEYED BY en général.



PROPERTY "Attributes"{LISTOF "Class Attribute" KEYED BY {Package:Package, "Class Name":Name, Name:*}}

Le schéma illustre ce que nous avons abordé plus haut : dans la définition d'une classe, un attribut de classe est entré en spécifiant son package (stocké dans la propriété Package de l'attribut de classe et obtenu à partir de la valeur Package de la classe en cours), son nom de classe (stocké dans la propriété “Nom de la classe” et obtenu à partir du nom réel de la classe), et son nom (stocké dans la propriété “Nom” de l'attribut de classe et provenant de lui-même).

KEYED BY (suite)

En résumé :

1. Pour chaque composant clé de l'**objet de référence**, la clause KEYED BY comporte un composant.
2. Les composants de la clause KEYED BY sont séparés par des virgules.
3. Chaque composant est constitué de deux parties :
   - La première partir identifie le composant clé de l'objet de référence.
   - La seconde partie indique où se trouve la valeur de ce composant.
   - Les deux parties sont séparées par un signe deux-points.

Toutefois, certaines valeurs par défaut peuvent être déduites pour simplifier la clause KEYED BY. Si les deux parties du composant sont identiques, la seconde peut être omise et dans ce cas, sa valeur est supposée être “ici” (représenté par un astérisque). Ainsi, dans la pratique, la clause KEYED BY de la propriété “Attributs” de la classe est codée ainsi :

KEYED BY {Package**,** "Nom de la classe"**:**Name**,** Name }

Naturellement, toutes les propriétés utilisées dans l'instruction KEYED BY doivent exister. Ainsi, Rational System Architect vérifie qu'il existe une propriété “Package” et une propriété “Nom de la classe” dans la définition “Attribut de classe” et qu'il s'agit de propriétés KEY.

De la sorte, non seulement il n'est plus nécessaire de coder les composants clé communs dans une propriété LISTOF telle que celle ci, mais encore l'utilisation d'une clause KEYED BY avec d'autres propriétés pour fournir des valeurs communes **permet de garantir que les mêmes valeurs sont utilisées pour chaque référence.** Ainsi, dans notre exemple, les attributs de classe auxquels il est fait référence dans la propriété “Attributs” de la classe appartiennent obligatoirement à la même classe du même package, ce qui est souhaitable dans ce cas.

KEYED BY (suite)

Dans d'autres cas, il peut être utile que les composants clé de l'objet référencé soient séparés à des fins de clarté et de simplicité. Dans de tels cas, une clause KEYED BY est utilisée pour désigner les propriétés qui fournissent les divers composants. En effet, pour ces raisons, lorsqu'il s'agit d'une propriété KEY et qu'elle fait référence à un objet avec des propriétés KEY, Rational System Architect **requiert** que les composants figurent dans des propriétés distinctes.

Il est souvent souhaitable que certaines valeurs de composant de clé en regard des noms soient fournies dans la référence plutôt qu'extraites d'une autre propriété. Cela peut se produire lorsqu'aucune propriété convenable ne peut fournir de valeur ou lorsqu'il n'est pas souhaitable que le composant de clé possède la même valeur pour toutes les références de la propriété. Dans ce cas, le mot-clé QUALIFIABLE est utilisé. Par exemple, la définition de classe contient la propriété suivante :

PROPERTY "Opérations" {Edit … ParmListOf "Méthode"
KEYED BY {"Package","Nom de la classe":Name,"Paramètres formels"
QUALIFIABLE, Name } … }

Cela indique que bien que les valeurs des propriétés “Package” et “Nom de la classe” des méthodes référencées doivent provenir respectivement des propriétés “Package” et Name, celles de la propriété “Paramètres formels" des méthodes et leurs noms doivent provenir de la propriété “ Opérations” de la classe elle-même. Par conséquent, chaque référence contiendra deux composants : la valeur de la propriété “Paramètres formel” et la valeur du nom, séparés par un point.

Notez que Rational System Architect requiert que les propriétés KEY qui contiennent une clause KEYED BY n'utilisent **pas** le mot clé QUALIFIABLE, et ce aux fins de clarté et de simplicité mentionnées plus haut.

KEY (utilisé pour l'affichage)

KEY est également l'un des arguments admis à la suite de FORMAT dans la commande **DISPLAY**. Les propriétés désignées comme clés sont affichées dans une section distincte du symbole.

Exemple :

```
DEFINITION "Entité"
{
PROPERTY "Description"
{ EDIT COMPLETE LISTOF "Attribut" FROM "Elément de données"
KEYED BY {Model, "Nom de l'entité":"Nom", "Nom"} RELATE BY
"utilise" ASGRID COPYSCRIPT OnCopyEntityDesc EDITCLASS
SACPropertyAttributeGrid Label "Liste d'attributs" LENGTH 4096
ZOOMABLE DISPLAY { FORMAT KEY LEGEND "Clé primaire" }
DISPLAY { FORMAT NONKEY LEGEND "Attributs non clés" }
...}
```

Dans l'exemple ci-dessus, la commande FORMAT KEY place tous les attributs désignés par l'utilisateur comme clés primaires sous la légende “Clé primaire” du symbole de l'entité. La commande FORMAT NONKEY place tous les attributs de clé non primaire sous la légende “Attributs non clés” du symbole de l'entité.

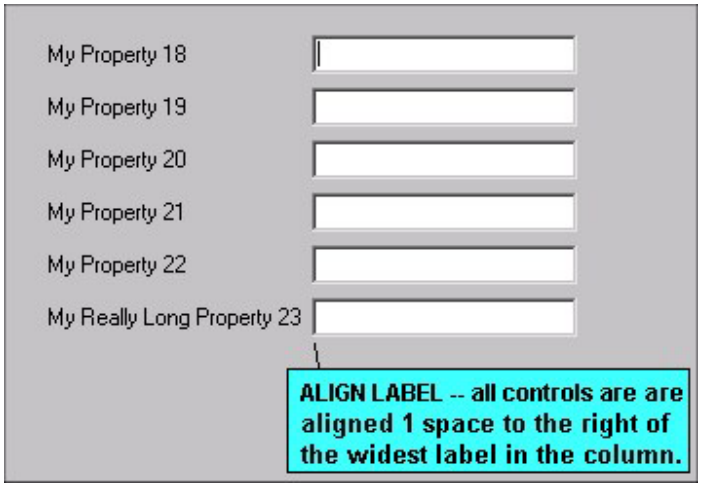
LABEL

La commande **LABEL** remplit deux rôles.

Rôle 1: LABEL constitue l'un des arguments de la commande **ALIGN**. Elle permet d'aligner tous les contrôles à un espace à droite du libellé le plus long de cette colonne. (Par opposition à la paire de mots clés ALIGN OVER qui place le nom sur la propriété).

Exemple :

```
Definition "Ma définition"
{
CHAPTER “Mon chapitre”
LAYOUT { COLS 1 ALIGN LABEL }
PROPERTY "Ma propriété 18"{ EDIT Text Length 10}
PROPERTY "Ma propriété 19"{ EDIT Text Length 10}
PROPERTY "Ma propriété 20"{ EDIT Text Length 10}
PROPERTY "Ma propriété 21"{ EDIT Text Length 10}
PROPERTY "Ma propriété 22"{ EDIT Text Length 10}
PROPERTY "Ma très longue propriété 23"{ EDIT Text Length 10}
}
```



Dans l'exemple ci-dessus, le contrôle de “Ma très longue propriété 23” est une zone de texte placée à un espace à droite du libellé. Tous les autres contrôles de zone de texte des autres propriétés de la boîte de dialogue sont alignés avec ce contrôle.

Voir aussi les mots-clés ALIGN, BODY et OVER.

LABEL

(suite)

Rôle 2 : LABEL permet d'attribuer un **nouveau libellé** aux onglets (chapitres), groupes ou propriétés d'une boîte de dialogue. Vous ne pouvez pas supprimer un nom de propriété qui a été défini dans SAPROPS. Toutefois, vous pouvez modifier le texte affiché pour la propriété en utilisant la commande LABEL dans le fichier USRPROPS.TXT.

Exemple 2 :

```
DIAGRAM "Diagramme de flux de données" {
  PROPERTY "Préfixe de libellé d'événement"
    { EDIT Text LENGTH 10 }
  PROPERTY "Lettres clés"
    { EDIT text LENGTH 10 LABEL "Préfixe de processus" } ..}
```

Si vous ajoutez le code ci-dessus au fichier USRPROPS.TXT (puis que vous rouvrez votre encyclopédie), les mots "Préfixe du processus" sont affichés comme libellé “Lettres clés”.

Modification du nom d'un groupe dans une définition
La commande LABEL permet de renommer un groupe. Si vous spécifiez une chaîne de texte vide (“ “), aucun mot n'apparaît pour la zone Groupe.

Exemple 2 :

```
DEFINITION “Attribut” {
GROUP "autres éléments" LABEL "" }
```

Pour renommer un chapitre, voir la commande CHAPTER. Voir aussi les mots clés ALIGN et BODY.

LABELPOS

Paramètre de la commande PLACEMENT qui permet de spécifier la position exacte du nom (ou du libellé) d'une propriété dans une boîte de dialogue DIAGRAM, SYMBOL ou DEFINITION. La commande LABELPOS reçoit deux arguments : la position horizontale (à partir du haut de la boîte de dialogue) et la position verticale (à partir de la gauche de la boîte de dialogue) dans les unités Windows.

Syntaxe :

PLACEMENT { **LABELPOS(4, 52)** PROPPOS (positionnement-horizontal, positionnement-vertical) PROPSIZE (largeur, hauteur) }

Exemple :

```
DEFINITION “Ma définition”
{
PROPERTY "Nom de la table" { EDIT Text LENGTH 31 PLACEMENT {
LABELPOS (4, 24) PROPPOS (20, 24) PROPSIZE(150, 12)} }
}
```

Notez que dans l'exemple ci-dessus LABELPOS et PROPPOS ont la même coordonnée y (24), ce qui signifie que le haut de leurs lettres se trouvent à 24 unités du sommet de la boîte de dialogue. Le libellé se trouvera donc à gauche du contrôle (et non dessus). Notez également que la différence entre l'abscisse de PROPPOS et de LABELPOS (20 - 4 = 16) laisse largement de la place (16 -10 = 6 unités) pour le nom de libellé sur 12 caractères, à savoir “Nom de table”, car ce dernier doit se placer à gauche de la position de départ du contrôle de la propriété.

Important : reportez-vous au chapitre 2 du présent document pour des conseils généraux sur le positionnement et le dimensionnement.

Voir aussi les mots clés PLACEMENT, PROPPOS, PROPSIZE et FORMAT.

LAYOUT

Ce mot-clé spécifie la présentation des propriétés d'une boîte de dialogue Diagramme, Symbole ou Définition. (Notez que la boîte de dialogue Symbole est incluse comme dernier onglet d'une boîte de dialogue Définition.)

A l'intérieur des accolades ouvrantes et fermantes de la commande LAYOUT, utilisez des arguments pour spécifier la présentation de toutes les propriétés appelées sous cette commande. Vous pouvez spécifier le nombre de colonnes de présentation des propriétés et la manière dont les propriétés doivent être alignées.

Plusieurs commandes LAYOUT peuvent être spécifiées pour une boîte de dialogue Diagramme, Symbole ou Définition. Vous pouvez spécifier une commande LAYOUT pour une boîte de dialogue intégrale et/ou la remplacer dans chaque GROUPE d'une boîte de dialogue ou dans chaque page (CHAPITRE) d'une boîte de dialogue.

Syntaxe :

LAYOUT { [critère d'alignement] [critère_PACK_TAB] [Nombre de colonnes] [JUSTIFY] }

Ou, plus spécifiquement :

LAYOUT {[ALIGN BODY | ALIGN LABEL | ALIGN OVER] [PACK | TAB] [COLS <number>] [JUSTIFY] }

Exemple :

```
SYMBOL "Objet" IN "Séquence"
{
LAYOUT { COLS 2 ALIGN OVER }
PROPERTY "Package" { EDIT OneOf "Package" READONLY }
PROPERTY "Classe" { EDIT OneOf "Class" KEYED BY { "Package",
Name } REQUIRED READONLY }
..}
```

Dans l'exemple ci-dessus, toutes les propriétés de la boîte de dialogue Symbole de l'objet sont présentées en deux colonnes.

Voir aussi les mots clés ALIGN, BODY, LABEL, OVER, PACK, TAB, COLS et JUSTIFY.

LEGEND

Chaîne de la propriété affichable dans un symbole rectangulaire qui remplace le nom de la propriété.

Exemple :

```
PROPERTY "Description"
  { EDIT ListOf Data
     DISPLAY { FORMAT Key LEGEND "Données clés" }
  }
```

Syntaxe :

LEGEND “<Votre texte>” : Le texte que vous placez entre guillemets ne sera affiché sur le symbole au dessus de l'entrée que si celle-ci comporte une valeur.

LEGEND "" : Affiche une ligne droite sans texte et ce, uniquement si l'entrée possède une valeur.

LEGEND "$$FORCE$$" : Affiche une ligne horizontale au dessus de l'entrée sur le symbole. Cette ligne sert de diviseur. Le mot clé “$$FORCE$$” est différent d'une simple utilisation de “ “ en ce qu'il force l'affichage d'une ligne horizontale même si l'affichage de la propriété est supprimé via la boîte de dialogue du mode d'affichage.

LEGEND “$$NONE$$” : N'affiche pas de ligne horizontale au dessus de l'entrée sur le symbole, que l'entrée possède des valeurs ou non. Cette ligne sert normalement de diviseur.

LEGEND “$$VFORCE$$” : Permet de disposer les propriétés de gauche à droite à l'intérieur des symboles et trace des lignes verticales entre-elles. Voir le mot clé VFORCE.

LEGEND “$$VNONE$$” : Permet de disposer les propriétés de gauche à droite, mais ne fournit *pas* de ligne de division. Voir le mot clé VNONE.

Voir aussi le mot clé DISPLAY.

LENGTH

Indique le nombre de caractères que l'utilisateur peut entrer dans la zone de la propriété.

Exemple :

```
PROPERTY "Depuis entité"
  { EDIT TEXT LENGTH 80 }
```

Dans l'exemple ci-dessus, *Depuis entité* peut comporter jusqu'à 80 caractères.

LINES

Définit le nombre de lignes, en profondeur, pour une zone de propriété.

Exemple :

```
DEFINITION "Constructeur"
 { CHAPTER "Desc., Formal Parm"
 GROUP "" {
 LAYOUT { COLS 2 TAB ALIGN OVER }
 PROPERTY "Paramètres formels"   { KEY EDIT Text LENGTH 1020 }
 PROPERTY "Liste d'initialiseurs"    { EDIT Text LENGTH 1000 LINES
4 }
```

Ce mot-clé n'est utile que si vous souhaitez que l'espace automatiquement fourni soit bien plus grand ou bien plus petit que la valeur par défaut.

Voir aussi le mot clé ZOOMABLE.

LIST

Le mot clé LIST remplit deux rôles dans USRPPROPS.TXT. La longueur par défaut est 1200.

Rôle 1 : Le mot clé LIST établit une liste des valeurs texte possibles. Il doit être défini à deux endroits : au début du fichier USRPROPS.TXT, pour spécifier la liste des valeurs possibles, et dans la propriété qui utilise la liste. Toutes les instructions de spécification LIST doivent figurer au début du fichier USRPROPS.TXT, avant toute instruction de spécification DIAGRAM, DEFINITION ou SYMBOL.

Exemple :

List "Stéréotypes des méthodes"
{
VALUE "Acquérir"
VALUE "Permettre"
VALUE "Définir"
}
DEFINITION “Méthode” {..
PROPERTY "Stéréotype"{ EDIT Text **LIST "Stéréotypes des méthodes"** Default "" LENGTH 30 } …}

Boutons d'option versus Liste déroulante

Rational System Architect affiche automatiquement une liste sous forme de liste de boutons d'option si le nombre de valeurs de l'instruction LIST est inférieur ou égal à quatre. Si le nombre de valeur est supérieur ou égal à cinq, la liste est automatiquement affichée sous forme de zone de liste déroulante. Les utilisateurs peuvent saisir leur propre valeur dans une zone de liste déroulante. Si vous souhaitez une zone de liste déroulante alors que votre liste ne contient que quatre valeurs, ou moins, utilisez le mot clé LISTONLYCOMBO.

Rôle 2 : Le mot clé LIST est également l'un des arguments admis après le mot clé FORMAT dans la commande **DISPLAY**. Le mot clé LIST entraîne l'affichage des éléments sur le symbole dans une liste, chaque espace génère une nouvelle ligne, sauf s'il est placé entre guillemets.

Exemple :

DEFINITION "Noeud opérationnel"
{PROPERTY "Activités opérationnelles" {EDIT LISTOF "Activité opérationnelle" LENGTH 2000 DISPLAY {**FORMAT LIST** Legend "Activités"} ..}

Voir aussi les mots clés LISTONLY et LISTONLYCOMBO.

LISTOF

L'un des types autorisés pour une propriété. Utilisé avec le mot clé EDIT pour spécifier que la propriété référence une **liste d'autres définitions**. Par exemple, une classe contient une propriété appelée Attributs, qui répertorie les attributs de classe. Un attribut de classe est un type de définition en lui-même, qui possède son propre ensemble de propriétés. Il s'oppose à la propriété d'une classe appelée Type d'accès, qui correspond à une liste de simples options textuelles, telle que Public, Privé, Protégé, etc. (La commande LIST permet de définir cette liste textuelle simple. Voir LIST.) Il diffère également de ONEOF, qui spécifie qu'une propriété référence une seule autre définition. Par exemple, une classe contient une propriété nommée Package, qui spécifie le package dans lequel la classe se trouve ; Package est une définition en soi.

LISTOF est utilisé avec le mot clé EDIT. La syntaxe est la suivante :

PROPERTY "Votre propriété" { EDIT LISTOF "Type de définition référencé" } LENGTH 1200}

Le mot clé ASGRID est souvent utilisé avec LISTOF. ASGRID présente la liste des définitions dans une grille. S'il n'est pas spécifié, les définitions sont répertoriées dans une structure de liste par défaut. LISTOF est parfois utilisé avec le mot-clé ZOOMABLE et également COMPLETE (décrit dans ce chapitre). Pour une propriété LISTOF, le mot-clé LENGTH a la valeur 1200 par défaut. LENGTH indique le nombre de caractères que l'utilisateur peut entrer dans la zone de la propriété. Dans ce cas, le nombre total de caractères des noms des définitions qui peuvent être hébergés dans la liste.

Exemple :

DEFINITION "Cas d'utilisation"
{
PROPERTY "Préconditions" { ZOOMABLE EDIT ListOf "Pré/Post condition" LENGTH 1200 }...}

Voir aussi les mots clés OF, EXPRESSIONOF, COMPLETE et ZOOMABLE.

LISTONLY

Indique que les valeurs d'une propriété doivent provenir de la liste affichée (créée via le mot clé LIST au sommet du fichier USRPROPS.TXT). L'utilisateur n'est ***pas autorisé*** à entrer ses propres valeurs dans la liste.

Exemple :

```
List "Importance"
{
Value "Obligatoire"
Value "Fortement souhaité"
Value "Souhaitable"
Value "Appréciable"
Value “Négligeable”
}

Definition "Etape de cas d'utilisation"
{
PROPERTY "Importance" {Edit Text ListOnly List "Importance"
Length 20 Default "Souhaitable" }
..}
```

Dans l'exemple ci-dessus, la liste est fournie dans la boîte de dialogue de définition Etape de cas d'utilisation sous forme de liste déroulante que vous pouvez sélectionner dans votre propre entrée. Notez que l'instruction List comporte cinq valeurs. Si elle en comportait quatre ou moins, la liste dans la boîte de dialogue Etape de cas d'utilisation serait présentée sous forme de boutons d'option. Si vous souhaitez une zone de liste déroulante alors que votre liste ne comporte que quatre valeurs ou moins, utilisez le mot clé LISTONLYCOMBO.

Voir aussi les mots clés LIST et LISTONLYCOMBO.

LISTONLYCOMBO

Fournit une liste déroulante quelque soit le nombre de valeurs LIST. En outre, l'utilisateur ne peut pas saisir ses propres valeurs dans la liste.

Le mot clé LISTONLYCOMBO offre une fonctionnalité absente de la commande LIST. Lorsque la commande LIST est utilisée, Rational System Architect affiche automatiquement une liste sous forme de cases à cocher si le nombre de valeurs de l'instruction LIST est inférieur ou égal à quatre. Si le nombre de valeur est supérieur ou égal à cinq, la liste est automatiquement affichée sous forme de zone de liste déroulante. Les utilisateurs peuvent saisir leur propre valeur dans une zone de liste déroulante. Si vous souhaitez une zone de liste déroulante alors que votre liste ne contient que quatre valeurs, ou moins, utilisez le mot clé LISTONLYCOMBO.

Exemple :

```
List "Importance"
{
Value "Obligatoire"
Value "Fortement souhaité"
Value "Souhaitable"
}

Definition "Etape de cas d'utilisation"
{
PROPERTY "Importance" {EDIT TEXT LISTONLYCOMBO LIST
"Importance" LENGTH 20 DEFAULT "Souhaitable" }
..}
```

Dans l'exemple ci-dessus, la liste est fournie dans la boîte de dialogue de définition Etape de cas d'utilisation sous forme de liste déroulante, bien que l'instruction LIST ne contienne que trois valeurs. Si vous aviez utilisé une simple instruction LIST, les valeurs auraient été affichées sous forme de boutons d'option car le nombre de valeurs est inférieur à cinq.

Voir aussi les mots clés LIST et LISTONLY.

MAX; MAXIMUM

Indique la valeur maximale autorisée pour une propriété définie par une valeur numérique. Une zone numérique est une zone dans laquelle seuls les caractères numériques sont autorisés.

Exemple :

```
PROPERTY Length
  { EDIT numeric LENGTH 2 MINIMUM 1 MAXIMUM 99  }
```

MENU

Référence le graphique utilisé pour représenter un symbole sur le menu et la barre d'outils Tracer, par comparaison avec l'espace de travail du diagramme.

Exemple :

```
SYMBOL "Satellite"
{ASSIGN To "Réseau sans fil"
DEPICTIONS { DIAGRAM "C:\Program
Files\IBM\pictures\satellite.bmp" }
DEPICTIONS { MENU "C:\Program Files\IBM
\pictures\satellitetoolbar.bmp" }}
```

Dans l'exemple ci-dessus, l'image satellitetoolbar.bmp est placée sur le menu Tracer du diagramme “Réseau sans fil”.

Voir aussi le mot clé DEPICTIONS.

MIN; MINIMUM

Indique la valeur maximale autorisée pour une propriété définie par une valeur numérique. Une zone numérique est une zone dans laquelle seuls les caractères numériques sont autorisés.

Exemple :

```
PROPERTY Length
  { EDIT numeric LENGTH 2 MINIMUM 1 MAXIMUM 99  }
```

MINISPEC

Dans Rational System Architect, une minispec est l'instruction exprimant la logique de traitement d'un symbole de type processus. Les minispecs sont rédigées dans une syntaxe formelle souvent appelée "Structured English" (langage structuré en anglais). Le mot clé MINISPEC est utilisé avec le mot clé EDIT.

Exemple :

```
DEFINITION "Processus"
{
PROPERTY "Description"
{ ZOOMABLE EDIT MINISPEC LENGTH 1500 LABEL "Minispec" }
...}
```

Une minispec est une instruction exprimant la logique de traitement d'un symbole de type processus, c'est-à-dire comment le traitement transforme des données d'entrée en données de sortie.

Voici un exemple d'instruction minispec :

```
If ISBN number brand new,
Create "LISTE ISBN PRINCIPALE"
Else
Update "Requête emprunteur"
```

Rational System Architect peut équilibrer les flux d'entrée et de sortie d'un processus à l'aide des mots contenus dans la minispec en fonction des éléments de données et de la structure des données de ces flux. Le système doit analyser le texte mot à mot et rechercher les mots importants. Ces derniers sont marqués en les délimitant avec des apostrophes ou des guillemets. Vous pouvez demander au système de prendre en compte tous les mots ou uniquement les mots marqués à cet effet.

Par défaut, le système prend en compte uniquement les mots importants et placés entre guillemets, par exemple :

```
Compute "coût_global" = "coût_unitaire" times "quantité"
```

Si vous souhaitez que le système prenne en compte tous les mots contenus dans les minispecs et non pas seulement ceux délimités par des guillemets, vous devez définir MinispecUsesQuotes à "N" dans le fichier SA2001.INI. La minispec ci-dessus peut alors être rédigée ainsi :

Compute coût_global = coût-unitaire times quantité

NAME

Permet d'indiquer que la clé d'une définition est le nom de l'objet lui-même et peut aussi être le nom de l'objet parent.

Exemple :

DEFINITION "Déclencheur SQL Server"
{
PROPERTY "Nom de la table"
{ EDIT OneOf Definition "Table" RELATE BY "indexé par" KEYED BY {"Nom de la base de données", "Nom du propriétaire","Nom de la table":**Name**, **Name**} }
...}

Dans l'exemple ci-dessus, la clé de la propriété "Nom de la table" dans la définition d'un déclencheur est le nom de la table dans laquelle ce déclencheur est défini. Le nom du propriétaire du déclencheur fait également partie de la clé.

NODESC;
NODESCRIPTION

Ce mot clé indique que la définition n'est associée à aucune propriété DESCRIPTION.

Syntaxe :

DEFINITION <nom-déf>
{ **NODESC**
}

Exemple :

DEFINITION "Type d'attribut XML"
{
NODESC
PROPERTY "Type de données" { Edit Text LIST "Type de données XML" Length 100 }
PROPERTY "Requis" { Edit Text List "oui/non XML" Length 100 }
PROPERTY "Par défaut" { Edit Text Length 1000 }
..}

Il existe un certain nombre de types de définitions dans Rational System Architect pour lesquels la zone Description a été supprimée à l'aide du mot clé NODESC. Il s'agit de types de définitions pour lesquels une description n'est pas nécessaire et ne serait d'aucune utilité à l'utilisateur. Par exemple, Modèle de déclencheur, Synonyme de table, Table, Procédures mémorisées et Vues.

NONADDR, NONADDRESSABLE

Permet de supprimer des définitions de la liste des adresses. 13 types de définition ont été prédéfinis en tant que ‘Adressable’, c'est-à-dire que vous pouvez ‘adresser’ un symbole présent dans un diagramme avec ces types de définition (sélectionnez un symbole et choisissez Dictionnaire, Adresse, puis le type de définition). Les types de définition ‘Adressables’ sont généralement des éléments tels que des conditions requises, des règles, des plans de test, etc., autrement dit des éléments que le symbole du diagramme "adresse" ou satisfait. Pour supprimer l'une de ces définitions du menu déroulant Dictionnaire, Adresses , modifiez l'instruction dans USRPROPS.TXT.

Exemple :

```
DEFINITION "Demande de modification"
  {
    NONADDR
  }
```

Voir aussi le mot clé ADDRESSABLE.

NONE

Il s'agit en réalité du mot clé $$NONE$$, utilisé avec le mot clé DISPLAY. Pour plus d'informations, voir le mot clé DISPLAY.

NONKEY

Désigne l'un des arguments admis après FORMAT dans la commande **Display**. Les éléments qui ne sont pas désignés comme clés peuvent être affichés dans une section distincte du symbole.

Exemple :

```
DEFINITION "Entité"
{
PROPERTY "Description"
{ EDIT COMPLETE LISTOF "Attribut" FROM "Elément de données"
KEYED BY {Model, "Nom de l'entité":"Nom", "Nom"} RELATE BY
"utilise" ASGRID COPYSCRIPT OnCopyEntityDesc EDITCLASS
SACPropertyAttributeGrid Label "Liste d'attributs" LENGTH 4096
ZOOMABLE DISPLAY { FORMAT KEY LEGEND "Clé primaire" }
DISPLAY { FORMAT NONKEY LEGEND "Attributs non clés" }
...}
```

Dans l'exemple ci-dessus, la commande FORMAT NONKEY place tous les attributs non désignés comme clés primaires sous la légende “Attributs non clés” d'un symbole d'entité.

NOTHING

Utilisé dans la commande RELATE BY NOTHING.

Voir RELATE BY.

NUMERIC

L'un des types autorisés pour une propriété. Il indique que la propriété est un nombre : seuls des chiffres peuvent être saisis dans la zone (ainsi que les signes plus ou moins). L'instruction LENGTH détermine le nombre de chiffres pouvant être entrés dans la zone. L'utilisateur ne pourra pas entrer de séparateur décimal ou de caractères dans cette zone, il ne pourra y entrer que des chiffres et les signes plus ou moins.

Exemple :

```
SYMBOL "Processus" IN "Flux de données Gane & Sarson"
  { PROPERTY "Description abrégée"
    { EDIT Text LENGTH 1500 }
     PROPERTY "Nombre" { EDIT Numeric LENGTH 4 }
```

OF DEFINITION REFERENCED IN

Vous permet de spécifier une liste restreinte de définitions parmi lesquelles vous pouvez choisir lorsque vous cliquez sur le bouton Choix d'une propriété. Vous permet d'affiner une instruction EDIT LISTOF ou EDIT ONEOF. Vous pouvez spécifier que seules sont répertoriées les définitions appartenant à une définition de référencement précise.

Exemple 1 :

```
DEFINITION "Objet"
{
PROPERTY "Package" { KEY EDIT OneOf "Package" RELATE BY
"indexé
   par" READONLY}
PROPERTY "Classe" { KEY EDIT OneOf "Classe" KEYED BY
   { "Package", Name } RELATE BY "indexé par" READONLY }
PROPERTY "Attributs" { ZOOMABLE EDIT LISTOF  "Attribut de
   classe" OF DEFINITION REFERENCED IN "Classe"
  KEYED BY {"Package", "Nom de la classe":"Classe", Name}
LENGTH
   4096 DISPLAY {FORMAT COMPONENT_SCRIPT
   _FmtNewUMLObjInstAttr LEGEND "$$FORCE$$"}  }
..}
```

Dans l'exemple ci-dessus, lorsque vous cliquez sur le bouton Choix de la grille de l'attribut d'un objet, seuls les attributs de la classe conteneur de l'objet sont indiqués. La propriété “Classe” référencée par OF DEFINITION REFERENCED IN doit également être spécifiée dans la définition de l'objet, comme illustré ci-dessus. (Par ailleurs, toujours dans l'exemple ci-dessus, l'attribut lui-même est spécifié par son package, son nom de classe et son propre nom d'attribut via l'instruction KEYED BY {"Package", "Nom de la classe":"Classe", Name.)

Exemple 2 :

```
DEFINITION "Message"
{
PROPERTY "Vers Classe"  { KEY EDIT OneOf "Classe" }
...
PROPERTY "Opération"  { EDIT ParmOneOf "Méthode" OF
DEFINITION REFERENCED IN "Vers Classe"  KEYED BY { "Nom de
la classe" : "Vers Classe", Name, "Paramètres formels"} LENGTH
1000}
..}
```

Dans l'exemple ci-dessus, la définition d'une ligne de message est épurée de telle manière que seules les méthodes de la classe de l'objet vers lequel la ligne de message est tracée sont répertoriées. Il s'agit des méthodes “Vers classe”. Ceci est possible car le symbole de l'objet (ligne de vie d'objet) vers lequel la ligne de message est tracée contient des propriétés pour sa classe de référencement et la ligne de message contient la propriété, “Vers classe”. Notez qu'il s'agit d'une définition de diagramme de séquence OMT. Le diagramme de séquence UML est indexé à des éléments supplémentaires (par package), par rapport à cet exemple. Pour une définition de ligne de message UML, voir le mot clé OF DEFINITION AND SUPERS REFERENCED IN.

Voir aussi le mot clé DEFINITION AND SUPERS REFERENCED IN.

OF DEFINITION AND SUPERS REFERENCED IN

Vous permet de spécifier une liste restreinte de définitions parmi lesquelles vous pouvez choisir lorsque vous cliquez sur le bouton Choix d'une propriété - cette liste restreinte référence les éléments d'une définition particulière **et** les éléments de toutes les autres définitions dont elle hérite (auxquelles elle est connectée via une ligne d'héritage). Elle permet d'affiner davantage une instruction EDIT LISTOF ou EDIT ONEOF et est utilisée pour la modélisation UML.

Exemple :

```
DEFINITION "Message/Stimulus"
{
PROPERTY "Vers Package" { KEY EDIT OneOf "Package" RELATE
  BY "indexé par" READONLY}
PROPERTY "Vers classe" { KEY EDIT OneOf "Classe" KEYED BY
  { "Package":"Vers Package", Name } }
PROPERTY "Vers objet"{ KEY EDIT OneOf "Objet" KEYED BY
  { "Package":"Vers Package", "Classe":"Vers classe",Name} }
PROPERTY "Opération" { EDIT ParmOneOf "Méthode"
  OF DEFINITION AND SUPERS REFERENCED IN "Vers classe"
  KEYED BY { "Package" QUALIFIABLE,"Nom de la classe"
  QUALIFIABLE,"Paramètres formels" QUALIFIABLE ,Name}
  LENGTH 1000 DISPLAY {FORMAT COMPONENT_SCRIPT
  _FmtNewUMLEventOperation LEGEND "$$NONE$$" } LABEL
  "Méthode" HELP "Sélectionnez une méthode dans la classe
appropriée" }
```

Dans l'exemple ci-dessus, lorsque vous cliquez sur Choix dans la définition du message, vous obtenez les méthodes de la classe à laquelle la ligne de message est associée (“Vers classe” et non “Depuis classe”) et les méthodes de toutes les classes qui sont des superclasses de celle-ci (connectées à cette classe via une ligne d'héritage).

Voir aussi le mot clé OF DEFINITION REFERENCED IN.

ONEOF

L'un des types autorisés pour une propriété. Utilisé avec le mot clé EDIT pour indiquer que la propriété référence **l'une** des définitions d'un autre type de définition.

Exemple :

SYMBOL "Relation" IN "Relation d'entité"
{
PROPERTY "Depuis entité" **{ EDIT ONEOF Entity** READONLY **}**
PROPERTY "Vers entité" { **EDIT ONEOF Entity** READONLY }
}

Dans l'exemple ci-dessus, la ligne de relation reliant deux entités contient, dans l'onglet Symbole, les entités qu'elle connecte : l'entité vers laquelle la ligne est tracée et l'entité à partir de laquelle la ligne est tracée. Dans les deux cas, une entité et une seule est répertoriée. Ces informations sont fournies automatiquement (Rational System Architect garde une trace des informations de provenance et de destination), par conséquent la propriété est de type READONLY.

Exemple 2 :

Definition “Etend”
{
PROPERTY "Etapes de cas d'utilisation" { ZOOMABLE **EDIT ONEOF "Etapes de cas d'utilisation"** KEYED BY {"Nom du modèle":"Nom du modèle", "Nom du cas d'utilisation":"Depuis cas d'utilisation", Name}
}

Dans l'exemple ci-dessus, la définition placée derrière la ligne des relations extend contient une référence à l'étape de cas d'utilisation (dans le cas d'utilisation de référence) dans laquelle l'extension (vers l'autre cas d'utilisation auquel la ligne est reliée) a lieu. Lorsque vous cliquez sur Choix pour cette propriété dans la définition Etend, vous obtenez une liste de toutes les étapes de cas d'utilisation. La zone de propriété ne peut toutefois recevoir qu'une seule étape de cas d'utilisation (à la différence de LISTOF qui permet de faire glisser plusieurs définitions).

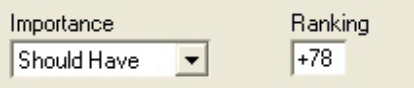
Voir aussi les mots clés LISTOF et EXPRESSIONOF.

OVER

Argument de la commande **ALIGN** qui place le nom de la propriété (ou son libellé) au-dessus de la commande correspondante (zone de texte, liste déroulante, etc.).

Exemple :

```
Definition "Etape de cas d'utilisation"
{
Chapter "Mes propriétés"
LAYOUT { COLS 2 ALIGN OVER TAB }
PROPERTY "Importance" {Edit Text ListOnlycombo List "Importance"
Length 20 Default "Souhaitable" }
PROPERTY "Numéro" { EDIT Numeric LENGTH 4  LABEL
“Classement”}
}
```



Dans l'exemple ci-dessus, toutes les propriétés de l'onglet (mot clé Chapitre) sont présentées de telle façon que le nom ou le libellé de chaque propriété est placé au-dessus de la commande correspondante. La propriété Numéro est renommée en “Classement”. Son libellé est placé au-dessus de sa commande, qui est une zone numérique simple. Le nom de la propriété Importance est placé au-dessus de la commande correspondante, qui est une liste déroulante.

A comparer au mot clé BODY, qui place le nom ou le libellé de la propriété à la gauche de la commande.

Voir aussi les mots clés BODY, TAB, ALIGN, LABEL et JUSTIFY.

OVERRIDABLE

Ce mot clé permet de modifier une propriété en lecture seule d'une définition, héritée du diagramme, ou d'écrire dans celle-ci, lors de la création de la définition bien qu'elle soit en lecture seule. OVERRIDABLE est utilisé **uniquement** au niveau du diagramme pour spécifier qu'il est possible de modifier une propriété appartenant à une définition représentant un symbole tracé dans ce diagramme, lors de la création du symbole, bien qu'elle soit en lecture seule.

Exemple :

```
Diagram "XML"
{
CHAPTER "Diagramme"
PROPERTY "Schéma XML" { EDIT ONEOF "Schéma XML"
AUTOCREATE RELATE BY "fait partie de” INITIAL USER REQUIRED
OVERRIDABLE READONLY }
..}

DEFINITION "Elément XML"
{
..
PROPERTY "Schéma XML" { Key Edit ONEOF "Schéma XML" Relate
By "indexé par" Initial User Required Readonly }
..}
```

Dans l'exemple ci-dessus, la définition Elément XML hérite pour sa valeur de propriété “Schéma XML” du symbole Elément XML qu'il définit, qui hérite à son tour la sienne du diagramme dans lequel il est placé. Lorsque vous placez initialement le symbole de l'élément XML dans l'espace de travail du diagramme, vous avez la possibilité de modifier la valeur propriété “Schéma XML” de la définition Elément XML. Si vous cliquez sur OK pour fermer la définition, puis rouvrez celle-ci, vous constaterez alors que la propriété “Schéma XML” est en lecture seule et ne peut plus être modifiée.

Notez que ce mot clé est utilisé dans la spécification Diagramme mais non pas dans la spécification de définition Elément XML.

PACK

Contrôle le positionnement vertical dans la commande **LAYOUT**. Cette commande sépare les ensembles de commandes et de libellés situés directement à droite en plusieurs colonnes en utilisant pour ce faire l'espace minimum possible.

Exemple :

```
GROUP "En-têtes/libellés Power Builder" {
  LAYOUT { COLS 2 ALIGN OVER PACK }
```

Voir aussi les mots clés LAYOUT et TAB.

PARENT

Dans la terminologie orientée objet et méthodologique, désigne l'objet duquel l'objet en cours hérite de toutes les propriétés de clés. Doit inclure RELATE BY "indexé par" dans l'instruction.

Exemple :

```
DEFINITION "Etape de cas d'utilisation"
{
PROPERTY "Nom du cas d'utilisation" { KEY EDIT OneOf "Cas
d'utilisation"
  PARENT RELATE BY "indexé par" READONLY  HELP "Nom du
  cas d'utilisation propriétaire" }
...}
```

PARMONEOF

Ce mot clé est utilisé uniquement dans certains cas dans SAPROPS.CFG et ne **doit pas** être utilisé dans USRPROPS.TXT. Il spécifie qu'une propriété de référence dans la syntaxe Rational System Architect s'affiche comme une opération UML standard. En d'autres termes, la partie qualifiée de la clé d'une propriété de référence qualifiée s'affiche entre parenthèses et non entre guillemets doubles, comme d'habitude dans Rational System Architect et l'ordre des clés est modifié de telle façon que le nom de l'objet référencé apparaît en tête de liste. Les méthodes se présentent par exemple ainsi : **meth(int, char)** au lieu de **"int, char".meth**.

Exemple :

```
DEFINITION "Modèle d'activité"
{ ..
PROPERTY "Opération" { EDIT PARMONEOF"Méthode" OF
DEFINITION REFERENCED IN "Classe active" KEYED BY
{ "Package" QUALIFIABLE, "Nom de la classe":"Classe active",
"Paramètres formels" QUALIFIABLE, Name}  LENGTH 1000 DISPLAY
{ FORMAT  STRING LEGEND "$$NONE$$" } LABEL "Méthode" HELP
"Spécifiez la classe, puis cliquez sur le bouton d'options" }
}
```

PARMLISTOF

Ce mot clé est utilisé uniquement dans certains cas dans SAPROPS.CFG et ne **doit pas** être utilisé dans USRPROPS.TXT. Il équivaut à PARMONEOF mais il s'applique à une liste de propriétés de référence, telle qu'une grille de méthodes. Il spécifie qu'une propriété de référence dans la syntaxe Rational System Architect s'affiche comme une opération UML standard. En d'autres termes, la partie qualifiée de la clé d'une propriété de référence qualifiée s'affiche entre parenthèses et non entre guillemets doubles, comme d'habitude dans Rational System Architect et l'ordre des clés est modifié de telle façon que le nom de l'objet référencé apparaît en tête de liste. Les méthodes se présentent par exemple ainsi : **meth(int, char)** au lieu de **"int, char".meth**.

Exemple :

```
Definition “Classe” {
CHAPTER "Méthodes"
PROPERTY "Opérations" { ZOOMABLE EDIT
COMPLETE_ALLOW_NEW PARMLISTOF "Méthode" KEYED BY {
"Package", "Nom de la classe":Name, "Paramètres formels"
QUALIFIABLE, Name }
LENGTH 1200 ASGRID DISPLAY { FORMAT COMPONENT_SCRIPT
_FmtNewUMLOperation LEGEND "$$FORCE$$"} LABEL "Méthodes" }
```

PLACEMENT

Cette commande permet de spécifier l'emplacement exact des propriétés dans une boîte de dialogue DIAGRAM, SYMBOL ou DEFINITION. La commande PLACEMENT reçoit les paramètres suivants :

LABELPOS (x, y) – spécifie le point de départ du coin supérieur gauche du nom (ou du libellé) d'une propriété. x correspond à la position horizontale (à partir du côté gauche de la boîte de dialogue) et y correspond à la position verticale (à partir du côté supérieur de la boîte de dialogue). x et y sont exprimés en unités Windows.

PROPPOS (x, y) – spécifie le point de départ du coin supérieur gauche du contrôle d'une propriété dans une boîte de dialogue. x correspond à la position horizontale (à partir du côté gauche de la boîte de dialogue) et y correspond à la position verticale (à partir du côté supérieur de la boîte de dialogue). x et y sont exprimés en unités Windows.

PROPSIZE (x, y) – spécifie la taille rectangulaire du contrôle. x correspond à la largeur et y correspond à la hauteur du contrôle, en unités Windows.

Exemple :

```
DEFINITION “Classe”
{
CHAPTER "Informations sur l'entité”
LAYOUT { COLS 2 TAB ALIGN OVER }
PROPERTY "Nom de la table" { EDIT Text LENGTH 31
   PLACEMENT {PROPPOS (4,   24) PROPSIZE(150, 12)} }
PROPERTY "Préfixe de désignation" { EDIT Text LENGTH 8 LABEL
   "Préfixe de colonne" HELP "Préfixe de nom de colonne"
   PLACEMENT {PROPPOS (175,   24) PROPSIZE(40, 12)} }
..
}
```

Dans l'exemple ci-dessus, la commande PLACEMENT remplace la commande LAYOUT pour l'onglet (CHAPTER). Le contrôle de la zone de texte de la propriété Nom de la table est positionné dans la boîte de dialogue de définition Classe (onglet Informations sur l'entité) à l'emplacement suivant : 4 unités Windows à partir du côté gauche de la boîte de dialogue et 24 unités Windows à partir du bord supérieur de la boîte de dialogue. La largeur de la zone de texte est égale à 150 unités et sa profondeur à 12 unités.

Important : reportez-vous au chapitre 2 du présent document pour des conseils généraux sur le positionnement et le dimensionnement.

Voir aussi les mots clés PROPPOS, PROPSIZE et LABELPOS.

PROPERTY

Introduit l'argument qui définit une caractéristique d'un diagramme, d'un symbole ou d'une définition. Le mot clé PROPERTY doit être suivi du nom de la propriété entre guillemets. Vous devez ensuite spécifier les caractéristiques de la propriété, soit entre deux accolades (ouvrante et fermante), soit entre les instructions BEGIN et END.

Syntaxe :

PROPERTY "<nom_propriété>"
{ EDIT <type_édition> <paramètre_propriété> **}**

Ou

PROPERTY "<nom_propriété>"
BEGIN EDIT <type_édition> <paramètre_propriété>
END

PROPPOS, PROPSIZE

Paire de paramètres de la commande PLACEMENT permettant de spécifier l'emplacement exact des propriétés dans une boîte de dialogue DIAGRAM, SYMBOL ou DEFINITION. La commande PROPPOS reçoit deux arguments : la position horizontale (à partir du haut de la boîte de dialogue) en unités Windows et la position verticale (à partir du côté gauche de la boîte de dialogue) en unités Windows. La commande PROPSIZE reçoit également deux arguments, x et y, qui indiquent respectivement la largeur et la hauteur du contrôle de la propriété en unités Windows.

Syntaxe :

PLACEMENT { **PROPPOS (positionnement-horizontal, positionnement-vertical) PROPSIZE (largeur, hauteur)** }

Exemple :

```
DEFINITION “Ma définition”
{
PROPERTY "Nom de la table" { EDIT Text LENGTH 31
PLACEMENT {PROPPOS (4, 24) PROPSIZE(150, 12)} }
}
```

Dans l'exemple ci-dessus, le début (le côté supérieur gauche de sa zone de texte) de la propriété Nom de la table est placée 4 unités Windows à partir du côté gauche de la boîte de dialogue de définition et 24 unités Windows à partir de son côté supérieur. La largeur de la zone de texte est égale à 150 unités Windows et sa longueur à 12 unités Windows. Cette instruction ne spécifie rien sur le nom (ou le libellé, “Nom de la table”) qui accompagne cette zone de texte. Ce libellé est donc placé à gauche de la zone de texte, son emplacement par défaut. Vous pouvez modifier cet emplacement à l'aide de la commande FORMAT ou de la commande PLACEMENT {LABELPOS}.

Important : reportez-vous au chapitre 2 du présent document pour des conseils généraux sur le positionnement et le dimensionnement.

Voir aussi les mots clés PLACEMENT, LABELPOS et FORMAT.

PUBLISHER

Ce mot clé vous permet de spécifier si les valeurs d'une propriété sont publiées dans la sortie de **SA Information Publisher**. Il reçoit deux arguments : **PUBLISHER SHOW** et **PUBLISHER ORDER**.

Syntaxe :

PROPERTY "Propriété utilisateur" {
PUBLISHER
{
SHOW (YES|NO) ' la valeur par défaut est YES
ORDER nnnn ' la valeur par défaut est zéro (pas de tri)}
}

PUBLISHER ORDER

Cet argument de la commande **PUBLISHER** vous permet de spécifier l'ordre dans lequel les valeurs de la propriété apparaissent dans la sortie publiée de **SA/Publisher**. Il est utilisé avec l'argument **PUBLISHER SHOW**.

Syntaxe :

PROPERTY "Propriété utilisateur" { ... **PUBLISHER** {SHOW YES|NO **ORDER** nnnn } ...}

La valeur par défaut est zéro (pas de tri).

Exemple :

DEFINITION "Exigence métier"
{
PROPERTY "Avantage" { EDIT Text LENGTH 50 **PUBLISHER {ORDER 2 }** }
PROPERTY "Statut" { EDIT Text LENGTH 50 **PUBLISHER {ORDER 1 }** }
PROPERTY "Difficulté" { EDIT Text LENGTH 50 **PUBLISHER {ORDER 3 }** }
PROPERTY "Affecté à" { EDIT Text LENGTH 50 **PUBLISHER {ORDER 4}** }
}

PUBLISHER SHOW

Cet argument de la commande **PUBLISHER** vous permet de spécifier si les valeurs d'une propriété doivent être publiées dans la sortie de **SA/Publisher**. Vous pouvez également utiliser la commande **PUBLISHER ORDER** avec ce mot clé.

Syntaxe :

PROPERTY "Propriété utilisateur" {.....PUBLISHER {SHOW YES|NO} ... }

La valeur par défaut est YES.

Exemple :

DEFINITION "Exigence métier"
{
PROPERTY "Avantage" { EDIT Text LENGTH 50 **PUBLISHER** {**SHOW NO**} }
}

Dans l'exemple ci-dessus, la propriété Avantage n'apparaîtra pas dans un site Web généré par SA/Publisher, même si cette propriété a été spécifiée comme sortie du rapport.

QUALIFIABLE

QUALIFIABLE est utilisé dans une propriété de référence où un ou plusieurs composants clé des objets référencés ne doivent pas nécessairement provenir d'autres propriétés de l'objet de référence, mais peuvent être fournis dans la propriété elle-même. Il est utilisé lorsqu'il est impossible de stocker toutes les données clé dans les propriétés d'une définition référençante mais que le nom de la définition référencée doit être qualifié par la propriété clé.

Par exemple, la clause KEYED BY suivante :

KEYED BY {composant_clé-1: nom_propriété_1, name}

indique que la valeur de composant_clé_1 doit provenir de nom_propriété_1 de sorte que la propriété de référence ne contienne que le(s) nom(s) des objets de référence. En revanche, la clause KEYED BY suivante :

KEYED BY {key_component-1 QUALIFIABLE, name}

indique que la valeur de composant_clé_1 doit provenir de cette propriété, c'est à dire de celle avec la clause KEYED BY, et que la propriété de référence contienne alors le(s) nom(s) tout comme le(s) composant(s) composant_clé_1 des objets de référence. Dans ces conditions, les valeurs des noms sont séparés des valeurs des composants key-component_1 par des points.

Exemple :

```
PROPERTY "Opérations”
{ ZOOMABLE EDIT ParmListOf "Méthode”
KEYED BY {"Nom de la classe":Name, "Paramètres formels"
QUALIFIABLE, Name}
LENGTH 1200
ASGRID
DISPLAY { FORMAT COMPONENT_SCRIPT FmtUMLOperation
LEGEND "$$FORCE$$"}
}
```

READONLY

Indique qu'une propriété peut être lue mais ne peut pas être modifiée. READONLY est utilisé dans le fichier SAPROPS pour les propriétés dont la valeur est insérée par le logiciel, tout en devant être visible de l'utilisateur. Il est toujours utilisé pour Initial AuditId, Date et Time, et Update AuditId, Date et Time. Les lignes de relation, les contraintes et les autres lignes connectant des symboles, dans lesquelles les symboles Depuis et Vers sont importants, sont toujours de type READONLY.

Exemple :

```
SYMBOL "Lien" IN "Booch (94) Object"
REM "défini par un objet Lien"
{
PROPERTY "Depuis classe"
{ EDIT OneOf "Classe" READONLY }
PROPERTY "Depuis objet"
{ EDIT OneOf "Objet" KEYED BY {"Classe":"Depuis
classe", Name}  READONLY }
PROPERTY "Vers classe"
{ EDIT OneOf "Classe"  READONLY }
PROPERTY "Vers objet"
{ EDIT OneOf "Objet" KEYED BY {"Classe":"Vers
classe", Name} READONLY  }
...}
```

REFPROP

Une propriété ne peut être utilisée qu'une seule fois dans une définition (à moins qu'elle ne figure entre deux instructions #ifdef). Si vous souhaitez faire figurer la même propriété plusieurs fois dans une définition, vous devez utiliser les mots clés Control et RefProp. C'est pourquoi ces deux mots clés sont souvent utilisés avec TESTPROC.
Pour pouvoir utiliser le mot clé Control, une référence initiale à la propriété qu'il référence doit figurer au début de la définition.
Le mot clé REFPROP est utilisé conjointement avec le mot clé Control.

Exemple :

```
Definition "Index"
{
CHAPTER "Propriétés de modélisation"
  { TESTPROC TestPropertyNotValue TESTPROPERTY
"DBMS" TESTSTRING { "ORACLE 8" } }
  PROPERTY "Clé primaire"
  {EDIT Boolean LENGTH 1 DEFAULT "F" READONLY }
  PROPERTY Unique
  {EDIT Boolean LENGTH 1 VALUESCRIPT
ProcessIndexUnique DEFAULT "F" }
  PROPERTY Clustered
  {EDIT Boolean LENGTH 1 DEFAULT "F" }
  ...

CHAPTER "Propriétés de modélisation"
  { TESTPROC TestPropertyValue TESTPROPERTY "DBMS"
TESTSTRING { "ORACLE 8" } }
  Control "Clé primaire"
  { REFPROP "Clé primaire" }
  Control Unique
  { REFPROP "Unique" }
  Control Clustered
  {REFPROP "En cluster"}
  ...
  }
```

Voir aussi le mot clé CONTROL.

RELATE (BY), RELATED (BY)

Le type de relation par défaut d'une propriété de référence est “utilise”. Le mot clé RELATE BY permet de la remplacer par une autre relation (par exemple, “indexé par”) ou de n'utiliser aucune relation (“RELATE BY nothing” devant être codé).

Vous pouvez utiliser les relations suivantes avec le mot clé RELATE BY :

Aucune : pas de relation.

Utilise : valeur par défaut. Signifie que la définition contient une définition.

Expliqué par : signifie qu'un symbole est expliqué par une définition.

Défini par : signifie qu'un symbole est défini par une définition.

Est une : signifie qu'une définition "est une instance d'une définition" (par exemple, une colonne est un élément de données)

Identifie : signifie qu'un objet identifie un autre objet.

Comprend : signifie qu'un objet comprend des objets (par exemple, un modèle comprend des entités, des relations, etc. Une catégorisation comprend des relations de catégorie).

Issu de : indique qu'un objet provient d'une définition.

Basé sur : signifie qu'un objet est basé sur une définition (généralement un élément de données).

Fait partie de : signifie qu'une définition fait partie d'une définition. Utilisé avec OneOf ou ListOf.

Indexé par – Signifie qu'une définition est identifiée par une définition.

Relations définies par l'utilisateur : il existe également 20 types de relation définies par l'utilisateur (USER 1 à USER 20) que l'utilisateur peut créer à l'aide d'une commande RENAME, par exemple RENAME RELATION “USER 1” to “XXXX”.

Exemple :

Definition “Etape de cas d'utilisation”
{
PROPERTY "Nom du cas d'utilisation" { KEY EDIT ONEOF "Cas d'utilisation" KEYED BY {"Package", Name} **RELATE BY** "indexé par" READONLY HELP "Nom du cas d'utilisation propriétaire" }
PROPERTY "Package" { KEY EDIT ONEOF "Package" **RELATE BY** "indexé par" READONLY}}

Dans l'exemple ci-dessus, le premier KEY EDIT indique que la propriété “Nom du cas d'utilisation” est une propriété clé de la définition Etape de cas d'utilisation. Cette propriété “Nom du cas d'utilisation” fait référence à une définition Cas d'utilisation, ce que spécifie ONEOF “Cas d'utilisation”. Vous devez spécifier la clé complète de ce cas d'utilisation (vers lequel l'étape est codée). Dans ce cas, vous utilisez la commande KEYED BY pour spécifier les clés (c'est-à-dire le package contenant le cas d'utilisation) et le nom du cas d'utilisation. Enfin, vous devez indiquer que l'étape de cas d'utilisation est indexée par ce cas d'utilisation, ce qui est réalisé par la commande RELATE BY “indexé par”. La clause RELATE BY est ajoutée vu que le type de relation par défaut d'une propriété de référence est “utilise”. Si vous souhaitez utiliser une autre relation (par exemple, “indexé par”), vous devez remplacer la valeur par défaut.

Le second KEY EDIT de l'exemple ci-dessus spécifie que la propriété “Package” est également une clé de la définition Etape de cas d'utilisation. Plus spécifiquement, l'étape du cas d'utilisation est indexée par une définition de package. Toutefois, remarquez qu'une définition de package n'est pas codée par d'autres propriétés que son propre nom, et que la commande KEYED BY n'est par conséquent pas utilisée. Le package est associé à une étape de cas d'utilisation par la relation “indexé par”.

RELATION

La relation existant entre une propriété et ses référents.

Voir aussi les mots clés RELATE [BY], RELATED [BY].

REM, REMARK

Le texte suivant cette commande ou le texte placé entre guillemets simples ou doubles est ignoré.

Exemple :

```
GROUP "Connexions"
  { LAYOUT { COLS 2 TAB ALIGN OVER }
    PROPERTY "Depuis entité"
  { KEY EDIT OneOf "Entité" RELATE BY "indexé par"
        READONLY}
    PROPERTY "Vers entité"
  { KEY EDIT OneOf "Entité"  RELATE BY "indexé par"
        READONLY}
  } REM "Fin du groupe Connexions"
```

RENAME

Permet d'établir des références à un objet par un nom autre que celui qui est normalement utilisé par Rational System Architect.

Exemple :

```
RENAME SYMBOL "Conversion de contrôle"
  IN "Flux de données & Mellor" TO "Processus"
RENAME DIAGRAM "Flux de données Ward & Mellor"
  TO "Ward Mellor"
```

RENAME DEFINITION

Permet d'utiliser 150 définitions fournies par l'utilisateur. Ces 150 définitions sont appelées Utilisateur 1 à Utilisateur 150. Utilisez la commande RENAME DEFINITION pour renommer un certain nombre de ces définitions et créer ainsi de nouveaux types de définition. Les instructions RENAME DEFINITION doivent être insérées en haut du fichier USRPROPS.TXT.

Exemple :

```
Rename Definition "Utilisateur 10" to "Configuration requise"
```

RENAME DIAGRAM

Permet d'utiliser 20 diagrammes fournis par l'utilisateur. Ces 20 diagrammes sont appelées Utilisateur 1 à Utilisateur 20. Utilisez la commande RENAME DIAGRAM pour renommer un certain nombre de ces diagrammes et créer ainsi de nouveaux types de diagramme. Les instructions RENAME DIAGRAM doivent être insérées en haut du fichier USRPROPS.TXT.

Exemple :

Rename Diagram "Utilisateur 10" to "Hiérarchie des besoins"

RENAME SYMBOL

Permet d'utiliser 150 symboles fournis par l'utilisateur. . Ces 150 symboles sont appelés Utilisateur 1 à Utilisateur 150. Utilisez la commande RENAME SYMBOL pour renommer un certain nombre de ces symboles et créer ainsi de nouveaux types de symbole. Les instructions RENAME SYMBOL doivent être insérées en haut du fichier USRPROPS.TXT.

Exemple :

Rename Symbol "Utilisateur 10" to "Configuration requise"

REQUIRED

Indique que l'utilisateur doit obligatoirement renseigner une propriété pour pouvoir créer un diagramme ou une définition. La propriété apparaît automatiquement dans la boîte de dialogue **Nom** initiale du diagramme ou de la définition.

Exemple :

Dans cet exemple, pour pouvoir créer la définition de l'entité d'élément XML, l'utilisateur doit indiquer une valeur pour la propriété Schéma XML :

```
DEFINITION "Entité d'élément XML"
{
PROPERTY "Schéma XML" { Key Edit ONEOF "Schéma XML"
Relate By "indexé par" Required
Readonly }
}
```

RETAIN STYLE

Ce mot clé spécifie que, lors de leur utilisation dans l'outil, les métafichiers fournis par l'utilisateur conservent leur style graphique et leurs couleurs d'origine. Il est utilisé avec la clause DEPICTIONS.

Lorsque vous utilisez des images externes fournies par l'utilisateur pour représenter des symboles dans un diagramme, vous pouvez par défaut spécifier les caractéristiques de ces symboles (par exemple la couleur de remplissage ou la couleur du contour) comme pour n'importe quel autre symbole dans Rational System Architect. Si vous spécifiez le mot clé RETAIN STYLE dans la clause DEPICTIONS, les couleurs du symbole défini par l'utilisateur ne sont plus modifiables.

Exemple :

```
LiST "Stéréotypes de noeud"
{
Value "Client" DEPICTIONS {diagram images\client.wmf menu
images\client.bmp}
Value "Base de données" DEPICTIONS {diagram
images\database.wmf menu images\database.bmp}
Value "Pare-feu" DEPICTIONS {diagramme RETAIN STYLE
images\firewall.wmf menu images\firewall.bmp}

SYMBOL "Noeud" in "Déploiement"
{
PROPERTY "Stéréotype" { INVISIBLE EDIT Text ListOnly List
"Stéréotypes de noeud" DEFAULT "" LENGTH 32}
}

DEFINITION "Noeud"
{
PROPERTY "Stéréotype"
{ EDIT Text LIST "Stéréotypes de noeud" Default "" LENGTH 32 }
```

Dans l'exemple ci-dessus, firewall.wmf peut être utilisé pour représenter un symbole de noeud dans un diagramme Déploiement si le stéréotype de noeud est "Pare-feu". Lorsque le métafichier firewall.wmf fourni par l'utilisateur (ajouté par celui-ci à la table FILES de la base de données de l'encyclopédie), est dessiné dans le diagramme, ses couleurs sont identiques à celles qu'il avait hors de Rational System Architect et ne peuvent pas être modifiées avec les outils de couleur de Rational System Architect.

SACPropertyOnOfBase	Utilisé dans la commande EDITCLASS SACPropertyOneOfBase. **N'utilisez pas cette combinaison de mots clés.** Elle a été conçue pour une situation spécifique dans Rational System Architect, à savoir l'héritage des propriétés d'éléments de données par un attribut dans une entité. Cette combinaison ne pourra être utilisée que dans SAPROPS.CFG. Il s'agit de la seule situation pour laquelle cette combinaison de mots clés peut s'appliquer. Son utilisation dans d'autres cas peut générer des erreurs.

Script

Cette commande appelle un script écrit en SA Basic. Ce script est utilisé pour les propriétés autres que ListOf et ExpressionOf. Une script reçoit la valeur d'une propriété et exécute une action, généralement pour afficher un type d'annotation spécifique sur le symbole d'un diagramme. La convention de dénomination associée au script est la suivante :

- _fmt (par exemple, _fmtUMLAttr) : la fonction elle-même est codée en dur et ne peut pas être modifiée. C'est le cas de la plupart des fonctions contenues dans SAPROPS.CFG. Le codage en dur de la fonction est effectué pour accélérer la réponse globale de Rational System Architect.
- fmt (par exemple, fmtUMLAttr) : existe dans le fichier fmtscript.bas du répertoire principal des exécutables de Rational System Architect.

Création de vos propres fonctions

Vous avez la possibilité de créer vos propres fonctions pour disposer des éléments de manière particulière sur un symbole, ou calculer une valeur spécifique. Ces fonctions ***ne doivent pas*** être placées dans le fichier fmtscript.bas, car ce dernier est écrasé à chaque nouvelle installation ou mise à jour de System Architect. Si vous créez vos propres fonctions, vous devez les insérer dans un fichier **usr_fn.bas**, que vous devez créer (ce fichier n'étant pas fourni par défaut) et placer dans le répertoire principal de Rational System Architect (<C>:\Program Files\IBM\Rational\11.3.1\System Architect Suite\System Architect). (Le fichier sarules.bas comporte une instruction #include vers le fichier usr_fn.bas).

La plupart des fonctions appelées dans SAPROPS.CFG, qui sont codées en dur et dont le nom commence par un trait de soulignement, tel que _fmtUMLAttr, ont une fonction équivalente dans le fichier fmtscript.bas (sans trait de soulignement). Vous pouvez vous aider des scripts contenus dans le fichier fmtscript.bas pour mener à bien la création de vos fonctions.

Explication des fonctions existantes :

FmtOMTAbstractClass renvoie le script **{abstrait}** si cette propriété est définie ; sinon ne renvoie rien.

SCRIPT (suite)

FmtBOOClassConstraint renvoie le nom de la contrainte entouré d'accolades, par exemple, **{*nom contrainte*}**, si celle-ci a été définie ; sinon ne renvoie rien.

FmtOMTObjInstClass renvoie le nom de la classe entourée de parenthèses, par exemple, **(*nom de classe*)**, si celle -ci a été définie ; sinon ne renvoie rien.

FmtEntryAction renvoie le script **entrée /** et le nom de l'action d'entrée lorsque cette dernière est définie ; sinon ne renvoie rien.

FmtExitAction renvoie le script **sortie** et le nom de l'action de sortie lorsque celle-ci est définie ; sinon ne renvoie rien.

FmtOMTTransition renvoie les valeurs suivantes, en respectant la ponctuation, lorsqu'elles sont définies dans une définition de transition d'état : **(*nom d'attribut*) [*condition*] /*action***

Exemples :

```
CHAPTER "Technique OMT orientée objet"
GROUP "Technique OMT orientée objet"  {
..
PROPERTY "Type abstrait"
{ EDIT Boolean LENGTH 1 DEFAULT "F" DISPLAY { FORMAT
SCRIPT FmtOMTAbstractClass  LEGEND "$$FORCE$$"} }
...
PROPERTY "Contraintes" { EDIT Text LENGTH 500 DISPLAY {
FORMAT SCRIPT FmtBOOClassConstraint  LEGEND "$$NONE$$"}
}
} REM "Fin de groupe OMT orienté objet"
```

Voir aussi les mots clés FORMAT, COLUMN_SCRIPT, COMPONENT_SCRIPT et fmtxxx.

STRING

Argument par défaut utilisé après le mot clé FORMAT dans une commande FORMAT **Display**. L'argument *string* permet d'afficher à l'écran le contenu de l'entrée du dictionnaire exactement tel qu'il a été saisi.

Exemple :

```
SYMBOL "Relation" IN "Modèle d'informations Shlaer"
{
PROPERTY "Description"  { EDIT Text LENGTH 100 }
PROPERTY "Phrase inversée"  { EDIT Text LENGTH 65 DISPLAY {
FORMAT STRING LEGEND "$$NONE$$" } }
..}
```

Voir aussi les mots clés FORMAT et DISPLAY.

SUPERS

Voir le mot clé OF DEFINITION AND SUPERS REFERENCED IN.

SYMBOL

Il s'agit du premier mot à l'intérieur du bloc dans lequel sont répertoriées les propriétés d'un symbole, par opposition à DEFINITION ou DIAGRAM.

Exemple :

```
SYMBOL "Entité" IN "Relation d'entité"
{
PROPERTY "Description"  { EDIT Text LENGTH 500 }
…
}
```

Voir aussi les mots clés DIAGRAM et DEFINITION.

TAB

Ce mot clé permet de contrôler la position verticale dans la commande **LAYOUT** en séparant des groupes de contrôles et de libellés de plusieurs colonnes par des tabulations de sorte que les entrées de chaque ligne soit alignées sur celle de la ligne précédente.

Exemple :

```
GROUP "Contraintes de vérification de schéma SQL Server"
{
LAYOUT { TAB ALIGN Over COLS 2 }
PROPERTY "Nom de contrainte de vérification SQL Server"
{ EDIT Text LENGTH 30 Label "Nom de la contrainte"}
PROPERTY "Contrainte de vérification SQL Server"
{ EDIT TEXT LENGTH 256 LINES 10 LABEL "Vérification de
contrainte"}
} REM "Fin de groupe Contrainte de vérification de schéma"
```

Voir aussi les mots clés LAYOUT et PACK.

Groupe d'instructions TESTPROC, TESTPROPERTY, TESTSTRING

Le groupe d'instructions TESTPROC, TESTPROPERTY et TESTSTRING fournit une fonction conditionnelle, au niveau d'un diagramme, pour les propriétés. Ce groupe fournit les mêmes fonctions que #ifdef si ce n'est que #ifdef fournit une fonction conditionnelle basée sur toute une encyclopédie. Le groupe d'instructions TESTPROC opère à partir d'une **propriété de diagramme** : une définition contiendra un certain ensemble de propriétés si une valeur est sélectionnée pour une "propriété de test" au sein des propriétés du diagramme.

Le groupe d'instructions TESTPROC est particulièrement utile pour spécifier des ensembles de propriété pour la modélisation des données d'après le système de gestion de base de données relationnelle choisi.

TESTPROC désigne une procédure test. Les deux valeurs suivantes peuvent être indiquées après le mot clé TESTPROC : **TestPropertyValue** et **TestPropertyNotValue**. Si TESTPROC est suivi de TestPropertyValue, ceci indique de "tester la propriété et, si sa valeur est identique à celle d'une des valeurs spécifiées dans TESTSTRING, appliquer les propriétés de la section TESTPROC à la définition en question". Si TESTPROC est suivi de TestPropertyNotValue, ceci indique de "tester la propriété et, si sa valeur n'est **pas** identique à l'une des valeurs spécifiées dans TESTSTRING, appliquer les propriétés de la section TESTPROC à la définition en question". **TestPropertyValue** et **TestPropertyNotValue** sont sensibles à la casse et doivent être spécifiées exactement comme indiqué. Elles ne fonctionneront pas si vous n'utilisez que des minuscules ou que des majuscules.

TESTPROPERTY désigne la propriété de diagramme qui sera interrogée.

TESTSTRING désigne les valeurs qui seront interrogées. Vous pouvez répertorier une ou plusieurs valeurs dans la chaîne.

Controls et RefProps : chaque propriété ne peut être utilisée qu'une fois dans une même définition (à moins d'être entourée de #ifdef). Pour utiliser plusieurs fois une propriété dans une définition, vous pouvez vous servir des mots clés Control et RefProp. C'est pourquoi ces deux mots clés sont souvent utilisés avec TESTPROC. Pour pouvoir utiliser le mot clé Control, une référence initiale à la propriété qu'il référence doit figurer au début de la définition.

Groupe d'instructions TESTPROC, TESTPROPERTY, TESTSTRING (suite)

Exemple :

```
Definition "Index"
{
CHAPTER "Propriétés de modélisation"
  { TESTPROC TestPropertyNotValue TESTPROPERTY "DBMS"
TESTSTRING { "ORACLE 8" } }
  PROPERTY "Clé primaire"
  {EDIT Boolean LENGTH 1 DEFAULT "F" READONLY }
  PROPERTY Unique
  {EDIT Boolean LENGTH 1 VALUESCRIPT ProcessIndexUnique
DEFAULT "F" }
  PROPERTY Clustered
  {EDIT Boolean LENGTH 1 DEFAULT "F" }
  ...

CHAPTER "Propriétés de modélisation"
  { TESTPROC TestPropertyValue TESTPROPERTY "DBMS"
TESTSTRING { "ORACLE 8" } }
  Control "Clé primaire"
  { REFPROP "Clé primaire" }
  Control Unique
  { REFPROP "Unique" }
  Control Clustered
  {REFPROP "En cluster"}
  ...
  }
```

TestPropertyValue Voir Groupe d'instructions TESTPROC, TESTPROPERTY, TESTSTRING.

TestPropertyNotValue Voir Groupe d'instructions TESTPROC, TESTPROPERTY, TESTSTRING.

TEXT

Il s'agit d'un type de zone autorisé. La définition spécifiée comme texte peut provenir d'une liste ou peut être constituée de caractères alphanumériques saisis par l'utilisateur.

Exemple :

```
DEFINITION "Relation"
{
CHAPTER "Relations et connexions"
GROUP "Relation"
{ LAYOUT { COLS 2 TAB ALIGN OVER }
PROPERTY "Role" { EDIT Text LENGTH 31 }
PROPERTY "Préfixe de rôle"  { EDIT Text LENGTH 31 }
}
```

TIME

Zone indiquant que la propriété comporte un horodatage dans la notation, qui est conforme au format horaire définit sous Windows. Les propriétés CHECKOUT TIME, FREEZE TIME, INITIAL TIME et UPDATE TIME ont toutes une signification particulière.

Exemple :

```
DIAGRAM "Flux de données Gane & Sarson"
{
PROPERTY "Heure de gel"
{ FREEZE  TIME }
..
}
```

D'autres utilisations possibles de la propriété TIME sont disponibles dans les définitions.

Exemple :

```
DEFINITION "X" {
PROPERTY "Heure de création"
{ EDIT Text INITIAL TIME LENGTH 12 READONLY }
}
```

TO

Utilisé dans la commande **Rename** pour séparer le nom initial et le nouveau nom d'un objet.

Exemple :

```
RENAME SYMBOL Class IN "Classe Booch"
  TO "Classe Booch"
```

METTRE A JOUR

Zone autorisée qui indique que le système met automatiquement à jour la zone lorsque la propriété est modifiée. Utilisée par défaut pour *Audit ID, Update Date* et *Update Time*.

Le mot clé UPDATE fournit les mêmes informations que le mot clé LAST CHANGED. Tous deux indiquent l'heure de la dernière modification d'une définition, à savoir que quelqu'un a ouvert une boîte de dialogue Définition, a apporté des modifications (comme l'ajout d'un espace ou la suppression d'une lettre dans l'une des propriétés, ou la suppression, puis le rajout de la même lettre), puis a cliqué sur le bouton SAUVEGARDER pour mémoriser les modifications. Si un utilisateur ouvre la boîte de dialogue d'une définition et clique sur Sauvegarder sans avoir apporté la moindre modification, la définition est inchangée et elle n'est pas considérée comme ayant été modifiée. (Remarque : Lorsqu'un utilisateur ouvre une définition, cette dernière est temporairement ‘verrouillée’ par cet utilisateur. Si aucune modification n'est effectuée, et que l'utilisateur referme la définition, les propriétés LAST CHANGED ou UPDATE sont inchangées. Toutefois, System Architect conserve une trace interne des utilisateurs ayant ‘verrouillé’ en dernier une définition. Ces informations sur le dernier verrouillage ne sont pas disponibles via les mots clés de USRPROPS.)

A partir de Rational System Architect V9, LAST CHANGED AUDITID, LAST CHANGED DATE et LAST CHANGED TIME sont fournis par défaut sous l'onglet Données d'accès de chaque boîte de dialogue de diagramme ou de définition. Ces propriétés sont codées en dur dans le produit, ce qui signifie que vous ne trouverez pas le mot clé LAST CHANGED dans les définitions de SAPROPS.CFG et que vous n'aurez pas besoin de les ajouter à USRPROPS.TXT pour les nouveaux types de diagramme ou de définition que vous créez.

Exemple :

```
DEFINITION "X"
  { PROPERTY "Heure de modification"
    { EDIT Text UPDATE TIME LENGTH 12 READONLY }
  }
```

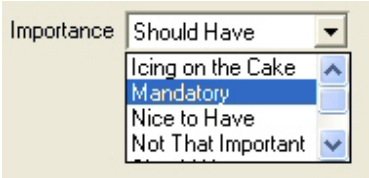
Voir aussi le mot clé INITIAL.

VALUE

Ce mot clé annonce une chaîne de valeur dans une liste.

Exemple :

```
List "Importance"
{
VALUE "Obligatoire"
VALUE "Fortement souhaité"
VALUE "Souhaitable"
VALUE "Appréciable"
VALUE "Négligeable"
}

Definition "Etape de cas d'utilisation"
{
PROPERTY "Importance" {EDIT TEXT LIST "Importance" LENGTH 20
DEFAULT "Souhaitable" }
}
```



Dans l'exemple ci-dessus, une liste est créée dans le fichier USRPROPS.TXT (en début de fichier). Cinq valeurs sont attribuées à la liste. Cette liste est ensuite utilisée dans le fichier USRPROPS.TXT avec la propriété “Importance” dans la définition d'une étape de cas d'utilisation. Notez que dans ce type de liste, l'utilisateur peut saisir les valeurs de son choix dans la zone.

Voir les mots clés LIST, LISTONLY et LISTONLYCOMBO.

VALUESCRIPT

Le mot clé VALUESCRIPT appelle une fonction écrite en SA Basic. Cette valeur nécessite de tester la cohérence des propriétés dans une boîte de dialogue ouverte. En règle générale, les fonctions analysent les ensembles de valeurs des propriétés d'une boîte de dialogue et apportent, en temps réel, les modifications nécessaires aux autres propriétés de cette boîte de dialogue, de manière à appliquer des règles de cohérence.

Création de fonctions personnalisées

Vous pouvez créer vos propres fonctions pour tester la cohérence des propriétés dans une boîte de dialogue ouverte. Pour plus d'informations sur la manière de créer votre propre fonction, reportez-vous au mot clé SCRIPT.

Exemple :

```
DEFINITION "Index"
{
PROPERTY Unique
{ PLACEMENT {PROPPOS(84, 0) PROPSIZE(100, 12)} EDIT Boolean
LENGTH 1 VALUESCRIPT ProcessIndexUnique DEFAULT "F" }
.. }
```

Dans l'exemple ci-dessus, la fonction ProcessIndexUnique, qui se trouve dans le fichier fmtscript.bas, est appelée. Elle est appelée uniquement lorsque le système de gestion de base de données (SGBD) choisi est Oracle. La fonction vérifie si la propriété Bitmap associée à l'index est activée. Si tel est le cas, la fonction ProcessIndexUnique désactive la propriété Index si celle-ci était activée. En effet, un index ne peut pas avoir la valeur Unique lorsqu'il est défini comme index Bitmap dans Oracle.

Exemple :

```
Definition “Elément de données”
{
PROPERTY "Type de données SQL"
{ EDIT text LIST "Types de données standard" LENGTH 30
VALUESCRIPT ProcessSQLDataType LABEL "Type de données"
"Type" "DT" PLACEMENT {PROPPOS(4, 26) PROPSIZE(80, 12)} }
..}
```

Dans l'exemple ci-dessus, la valeur ProcessSQLDataType vérifie si le type de l'élément de données est obtenu à partir d'un domaine de données sous-jacent, et le cas échéant, l'insère automatiquement dans la zone. Si le type n'est pas hérité et que l'utilisateur laisse la zone vide, la fonction stipule automatiquement comme valeur par défaut Caractère 10 lorsque l'utilisateur appuie sur la touche Entrée ou qu'il modifie une zone dans la grille des attributs.

VFORCE

Permet de tracer des lignes verticales à l'intérieur de symboles, et non des lignes horizontales (valeur par défaut). VFORCE présente les propriétés de gauche à droite et les sépare par une ligne verticale. Remarque : La commande VNONE est identique à VFORCE, sauf qu'elle n'affiche pas la ligne verticale).

Syntaxe :

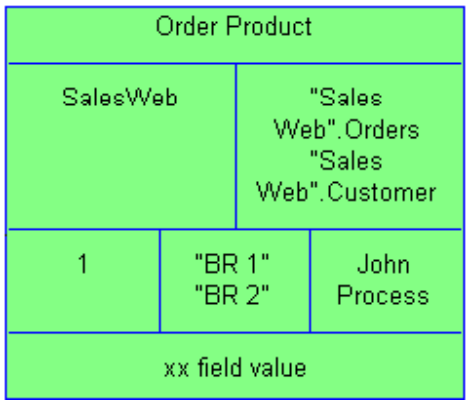
{FORMAT String LEGEND **"$$VFORCE$$"**}

Exemple :

```
DEFINITION "Processus métier élémentaire"
{
PROPERTY "Applications prises en charge"
{  Edit Listof "Application" Label "Applications" LENGTH 2000 HELP
"Doit être saisi via une matrice"  READONLY DISPLAY { FORMAT
String LEGEND "$$FORCE$$"} }
PROPERTY "Données référencées"
{ EDIT ListOf "Entité" KEYED BY {Model QUALIFIABLE,
Name} LENGTH 5000 READONLY DISPLAY { FORMAT String
LEGEND "$$VFORCE$$"}}
..}
```

Notez que la première propriété énumérée est de type FORCE, et non VFORCE. Les propriétés suivantes que vous souhaitez aligner à droite de la première propriété comportent la spécification VFORCE. Dans l'image ci-dessous, les valeurs “Ventes Web”.Commandes et “Ventes Web”.Client sont répertoriées dans une zone “Données référencées”. La commande VFORCE a été utilisée pour faire figurer cette zone à droite de la zone de propriété “Applications prises en charge”, qui dans l'exemple comporte une valeur Ventes Web.

VFORCE (suite)



Notez également que la commande VFORCE a également été utilisée pour faire apparaître les zones contenant “BR 1, “BR 2” et Processus de Jean à droite de celle contenant la valeur 1, mais que ceci n'est pas indiqué dans l'exemple USRPROPS.TXT fourni.

VISIBLE

Si une propriété est définie comme INVISIBLE dans SAPROPS.CFG, l'utilisation du mot clé VISIBLE permet de la faire apparaître dans la boîte de dialogue de la définition.

Exemple (SAPROPS) :

```
DEFINITION "Procédure mémorisée Watcom"
      { CHAPTER "Clés et paramètres"
   PROPERTY "Nom du propriétaire"
      { EDIT Text KEY  LENGTH 31 }
   PROPERTY "Numéro de procédure"
      { INVISIBLE EDIT Numeric LENGTH 9 }
   PROPERTY "Description"
      { EDIT Text  LENGTH 400 }
```

Exemple (USRPROPS):

```
DEFINITION "Procédure mémorisée Watcom"
   PROPERTY "Numéro de procédure"
      { VISIBLE }
```

Voir aussi le mot clé INVISIBLE.

VNONE

Il s'agit en réalité du mot clé $$VNONE$$, utilisé avec le mot clé DISPLAY. Pour plus d'informations, voir le mot clé DISPLAY.

Permet de tracer des lignes verticales à l'intérieur de symboles, et non des lignes horizontales (valeur par défaut). VNONE présente les propriétés de gauche à droite, mais sans afficher de ligne verticale entre les deux. (Remarque : La commande VFORCE remplit la même fonction mais en affichant la ligne verticale.)

Syntaxe :

{FORMAT String LEGEND **"$$VNONE$$"**}

Exemple :

Notez que dans l'exemple USRPROPS.TXT ci-dessous, la première propriété répertoriée indique FORCE. Les propriétés suivantes que vous souhaitez aligner à droite de la première propriété (sans ligne verticale de séparation) comportent la spécification VNONE.

Exemple :

```
DEFINITION "Processus métier élémentaire"
{
PROPERTY "Applications prises en charge"
{  Edit Listof "Application" Label "Applications" LENGTH 2000 HELP
"Doit être saisi via une matrice"  READONLY DISPLAY { FORMAT
String LEGEND "$$FORCE$$"} }
PROPERTY "Données référencées"
{ EDIT ListOf "Entity" KEYED BY {Model QUALIFIABLE,
Name} LENGTH 5000 READONLY DISPLAY { FORMAT String
LEGEND "$$VNONE$$"}}

..}
```

WHERE

Affiche dans la boîte de dialogue Choix uniquement les définitions contenant une valeur fixe dans une propriété nommée de la définition.

Exemple :

```
Rename Definition "Utilisateur 1" To "Type d'avion"
Rename Definition "Utilisateur 2" To "Avions filtrés"


List "Moteur"
{
  Value "Hélice"
  Value "Réacteur"
  Value "Planeur"
}

Definition "Type d'avion"
{
  Property "Type de moteur"
   { EDIT Text List "Moteur" Length 48 }
}

Definition "Avions filtrés"
{
   Property "Type d'avion sélectionné"
        { edit listof "Type d'avion" WHERE "Type de moteur" =
"Réacteur"}
}
```

Si le fichier USRPROPS.TXT était appliqué à une encyclopédie et que les définitions Type d'avion suivantes étaient créées dans l'encyclopédie :

Mustang (moteur = Hélice)
Spitfire (moteur = Hélice)
F-16 Fighting Falcon (moteur = Réacteur)
F-86 Sabre (moteur = Réacteur)

après la création d'une nouvelle définition de type Avions filtrés (nommée, par exemple, Chasseurs à réaction actuels), un clic sur le bouton **Choix** pour cette définition ne ferait apparaître que deux options : F-16 Fighting Falcon et F-86 Sabre ; toutes les autres définitions pour lesquelles Type de moteur est défini à Hélice n'apparaîtraient pas dans la liste **Choix**.

ZOOMABLE

Insère dans une zone de liste un bouton permettant à celle-ci de couvrir toute la page.

Exemple :

PROPERTY "Rôles utilisateur"
{ZOOMABLE EDIT ListOf "Rôle utilisateur avec droits d'accès"
LENGTH 1500 LABEL "Rôle(s) utilisateur"}

Voir aussi le mot clé LINES.

4

Support IBM

Introduction

Un certain nombre de ressources et d'outils d'aide vous permettant d'identifier et de résoudre les incidents sont à votre disposition. En cas d'incident avec le produit, vous pouvez :

Vous reporter aux informations sur l'édition du produit pour en savoir plus sur les incidents connus, les solutions existantes et les informations d'identification et de résolution des incidents.

Vérifier si un module téléchargeable ou un correctif est disponible pour résoudre l'incident.

Effectuer une recherche dans les bases de connaissances disponible pour déterminer si une résolution du problème est déjà documentée. Si vous avez toujours besoin d'aide, contactez le service de support logiciel IBM® et signalez l'incident.

Rubriques contenues dans ce chapitre	Page

Pour contacter le service de support logiciel IBM Rational

Si vous ne parvenez pas à résoudre un incident à l'aide des ressources d'aide, contactez le service de support logiciel IBM® Rational®.

Remarque : si vous êtes un ancien client de Telelogic, consultez le site unique de référence pour toutes les ressources de support à l'adresse http://www.ibm.com/software/rational/support/telelogic/

Configuration requise :

Pour soumettre un incident au service de support logiciel IBM Rational, vous devez être détenteur d'un contrat de maintenance logicielle Passport Advantage® actif. Passport Advantage constitue l'offre globale de licence et de maintenance logicielle d'IBM (mises à niveau des produits et support technique). Vous pouvez vous inscrire à Passport Advantage en ligne à l'adresse http://www.ibm.com/software/lotus/passportadvantage/howtoenroll.html

- Pour en savoir plus, consultez les foires aux questions relatives à Passport Advantage à l'adresse http://www.ibm.com/software/lotus/passportadvantage/brochures_faqs_quickguides.html.
- Pour obtenir de l'aide, prenez contact avec votre interlocuteur IBM habituel.

Pour soumettre un incident en ligne (à partir du site Web IBM) au service de support logiciel IBM Rational, procédez comme suit :

- Enregistrez-vous en tant qu'utilisateur sur le site Web du service de support logiciel IBM Rational. Pour plus d'informations sur cet enregistrement, consultez la page http://www.ibm.com/software/support/.
- Inscrivez-vous comme appelant autorisé dans l'outil de demande de service.

Autres informations

Pour en savoir plus sur les actualités, les événements et autres informations relatives au produit, consultez le site Web IBM Rational : http://www.ibm.com/software/rational/.

Soumission d'incidents

Pour soumettre un incident au service de support logiciel IBM Rational, procédez comme suit :

1. Déterminez l'impact de l'incident. Lorsque vous soumettez un incident à IBM, vous devez indiquer un niveau de gravité. Vous devez donc comprendre et évaluer l'impact de cet incident sur les opérations.

Pour déterminer le niveau de gravité, utilisez le tableau suivant.

Gravité	Description
1	L'impact de l'incident est critique : vous ne pouvez plus utiliser le programme, ce qui a des conséquences critiques pour vos opérations. La résolution de l'incident doit être immédiate.
2	L'incident a un impact important sur les opérations : le programme peut être utilisé, mais de façon très limitée.
3	L'incident a un certain impact sur les opérations : le programme peut être utilisé, mais certaines fonctions moins importantes ne sont pas disponibles.
4	L'impact de l'incident est minime : l'impact sur les opérations est minime ou une solution palliative acceptable a été mise en place.

2. Décrivez l'incident et rassemblez les informations nécessaires. Lorsque vous décrivez l'incident à IBM, soyez aussi précis que possible. Donnez toutes les informations utiles afin que les spécialistes du service de support logiciel IBM Rational puissent vous aider efficacement à résoudre l'incident. Pour

gagner du temps, préparez les réponses aux questions suivantes :

- Quelles versions du logiciel étaient en cours d'exécution lorsque l'incident s'est produit ?
- Pour déterminer le nom et la version exacts du produit, utilisez l'option qui vous concerne :
  - Démarrez le gestionnaire d'installation IBM et cliquez sur **File** > **View Installed Packages**. Développez un groupe de package et sélectionnez un package pour en voir le nom et le numéro de version.
  - Démarrez le produit et cliquez sur **Aide** > **A propos de** pour voir le nom et le numéro de version de l'offre.
- Quel système d'exploitation utilisez-vous ? Quelle est sa version (et celle des service packs ou des correctifs installés) ?
- Avez-vous des fichiers journaux, traces et messages relatifs à l'incident ?
- Pouvez-vous recréer l'incident ? Si oui, quelles étapes avez-vous exécuté pour recréer l'incident ?
- Avez-vous apporté des modifications au système ? Avez-vous par exemple modifié le matériel, le système d'exploitation, le logiciel réseau ou d'autres composants du système ?

3. Utilisez-vous actuellement une solution palliative ? Si oui, préparez une description de cette solution.
4. Soumettez l'incident au service de support logiciel IBM Rational de l'une des façons suivantes :
   - En ligne : accédez au site Web du service de support logiciel IBM Rational à la page https://www.ibm.com/software/rational/support/.

Dans le navigateur de tâches du support Rational, cliquez sur **Open Service Request**. Sélectionnez l'outil électronique de rapport d'incident et ouvrez un PMR (Problem Management Record) pour décrire l'incident.

- Pour plus d'informations sur l'ouverture d'une demande de service, consultez la page http://www.ibm.com/software/support/help.html.

- Vous pouvez également ouvrir une demande de service en ligne avec IBM Support Assistant. Pour plus d'informations, consultez la page http://www.ibm.com/software/support/isa/faq.html.
- Par téléphone : pour identifier le numéro de téléphone à appeler dans votre pays ou dans votre région, consultez le répertoire IBM des contacts dans le monde à l'adresse http://www.ibm.com/planetwide/ et cliquez sur le nom de votre pays ou de votre région géographique.

- Via votre interlocuteur IBM habituel : si vous ne parvenez pas à contacter le service de support logiciel IBM Rational en ligne ou par téléphone, contactez votre interlocuteur IBM habituel. Il pourra si nécessaire ouvrir une demande de service pour vous. Pour plus d'informations sur les contacts de chaque pays, consultez la page http://www.ibm.com/planetwide/.

5

Annexe :

Introduction

Ce chapitre contient des informations sur les utilisations autorisées et sur les marques de IBM® Rational® System Architect.

Remarques

Le présent document peut contenir des informations ou des références concernant certains produits, logiciels ou services IBM non annoncés dans ce pays. Pour plus de détails, référez-vous aux documents d'annonce disponibles dans votre pays, ou adressez-vous à votre partenaire commercial IBM. Toute référence à un produit, logiciel ou service IBM n'implique pas que seul ce produit, logiciel ou service puisse être utilisé. Tout autre élément fonctionnellement équivalent peut être utilisé, s'il n'enfreint aucun droit d'IBM. Il est de la responsabilité de l'utilisateur d'évaluer et de vérifier lui-même les installations et applications réalisées avec des produits, logiciels ou services non expressément référencés par IBM.

IBM peut détenir des brevets ou des demandes de brevet couvrant les produits mentionnés dans le présent document. La remise de ce document ne vous donne aucun droit de licence sur ces brevets ou demandes de brevet. Si vous désirez recevoir des informations concernant l'acquisition de licences, veuillez en faire la demande par écrit à l'adresse suivante :

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785

Les informations sur les licences concernant les produits utilisant un jeu de caractères double octet peuvent être obtenues par écrit à l'adresse suivante :

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

Le paragraphe suivant ne s'applique ni au Royaume-Uni, ni dans aucun pays dans lequel il serait contraire aux lois locales.LE PRESENT DOCUMENT EST LIVRE EN L'ETAT SANS AUCUNE GARANTIE EXPLICITE OU IMPLICITE. IBM DECLINE NOTAMMENT TOUTE RESPONSABILITE RELATIVE A CES INFORMATIONS EN CAS DE CONTREFACON AINSI QU'EN CAS DE DEFAUT

D'APTITUDE A L'EXECUTION D'UN TRAVAIL DONNE. Certaines juridictions n'autorisent pas l'exclusion des garanties implicites. Dans ce cas, l'exclusion ci-dessus ne vous sera pas applicable.

Le présent document peut contenir des inexactitudes ou des coquilles. Ce document est mis à jour périodiquement. Chaque nouvelle édition inclut les mises à jour. IBM peut, à tout moment et sans préavis, modifier les produits et logiciels décrits dans ce document.

Les références à des sites Web non IBM sont fournies à titre d'information uniquement et n'impliquent en aucun cas une adhésion aux données qu'ils contiennent. Les éléments figurant sur ces sites Web ne font pas partie des éléments du présent produit IBM et l'utilisation de ces sites relève de votre seule responsabilité.

IBM pourra utiliser ou diffuser, de toute manière qu'elle jugera appropriée et sans aucune obligation de sa part, tout ou partie des informations qui lui seront fournies.

Les licenciés souhaitant obtenir des informations permettant : (i) l'échange des données entre des logiciels créés de façon indépendante et d'autres logiciels (dont celui-ci), et (ii) l'utilisation mutuelle des données ainsi échangées, doivent adresser leur demande à :

Intellectual Property Dept. for Rational Software
IBM Corporation
1 Rogers Street
Cambridge, MA 02142
U.S.A

Ces informations peuvent être soumises à des conditions particulières, prévoyant notamment le paiement d'une redevance.

Le logiciel sous licence décrit dans ce document et tous les éléments sous licence disponibles s'y rapportant sont fournis par IBM conformément aux dispositions de l'ICA, des Conditions internationales d'utilisation des logiciels IBM ou de tout autre accord équivalent.

Les données de performance indiquées dans ce document ont été déterminées dans un environnement contrôlé. Par conséquent, les résultats peuvent varier de manière significative selon l'environnement d'exploitation utilisé. Certaines mesures évaluées sur des systèmes en cours de développement ne sont pas garanties sur tous les systèmes disponibles. En outre, elles peuvent résulter d'extrapolations. Les résultats peuvent donc varier. Il incombe aux utilisateurs de ce document de vérifier si ces données sont applicables à leur environnement d'exploitation.

Les informations concernant des produits non IBM ont été obtenues auprès des fournisseurs de ces produits, par l'intermédiaire d'annonces publiques ou via d'autres sources disponibles. IBM n'a pas testé ces produits et ne peut confirmer l'exactitude de leurs performances ni leur compatibilité. Elle ne peut recevoir aucune réclamation concernant des produits non IBM. Toute question concernant les performances de produits non IBM doit être adressée aux fournisseurs de ces produits.

Le présent document peut contenir des exemples de données et de rapports utilisés couramment dans l'environnement professionnel. Ces exemples mentionnent des noms fictifs de personnes, de sociétés, de marques ou de produits à des fins illustratives ou explicatives uniquement. Toute ressemblance avec des noms de personnes, de sociétés ou des données réelles serait purement fortuite.

Licence de copyright

Le présent logiciel contient des exemples de programmes d'application en langage source destinés à illustrer les techniques de programmation sur différentes plateformes d'exploitation. Vous avez le droit de copier, de modifier et de distribuer ces exemples de programmes sous quelque forme que ce soit et sans paiement d'aucune redevance à IBM, à des fins de développement, d'utilisation, de vente ou de distribution de programmes d'application conformes aux interfaces de programmation des plateformes pour lesquels ils ont été écrits ou aux interfaces de programmation IBM. Ces exemples de programmes n'ont pas été rigoureusement testés dans toutes les conditions. Par conséquent, IBM ne peut garantir expressément ou implicitement la fiabilité, la maintenabilité ou le fonctionnement de ces programmes.

Toute copie totale ou partielle de ces programmes exemples et des oeuvres qui en sont dérivées doit comprendre une notice de copyright, libellée comme suit :

Si vous visualisez ces informations en ligne, il se peut que les photographies et illustrations en couleur n'apparaissent pas à l'écran.

Marques

IBM, le logo IBM et ibm.com sont des marques d'International Business Machines Corp., dans de nombreux pays. Les autres noms de produits et de services peuvent appartenir à IBM ou à des tiers. La liste actualisée de toutes les marques d'IBM est disponible sur la page Web "Copyright and trademark information" à l'adresse : www.ibm.com/legal/copytrade.html

Microsoft et Windows sont des marques de Microsoft Corporation aux Etats-Unis et/ou dans certains autres pays.

Les autres noms de sociétés, de produits et de services peuvent appartenir à des tiers.

Index

B

C

D

E

G

H

I

J

K

L

M

N

P

R

T

U

Z