IBM Rational

System Architect USRPROPS 拡張ガイド

リリース 11.3.1

本書をご使用になる前に、「付録」の『特記事項』(5-1 ページ) を参照してください。

本書は、IBM® Rational® System Architect® バージョン 11.3.1 および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

目次

1

Rational System Architect エンサイクロペディアのメタモデルの拡張

はじめに

この章では、IBM® Rational® System Architect® エンサイクロペディアのメタモデルを USRPROPS.TXT を介して拡張するメカニズムについて説明します。

Rational*System Architect*

Rational System Architect は、さまざまな方法で拡張およびカスタマイズすることができます。描画動作については、ツールでさまざまな選択を行ったり、sa2001.ini ファイルを使用したりすることでカスタマイズできます。ツールバー、マトリックス・エディター、レポートなどをカスタマイズすることができます。さらに、Rational System Architect には Microsoft Visual Basic for Applications のサポートが組み込まれています。これによって、ユーザーは、Rational System Architect 内で実行できるネイティブ・マクロを作成して、便利なユーティリティーの追加やツール動作の実行など、さまざまな操作を行うことができます。

USRPROPS.TXT を使用したメタモデルの拡張

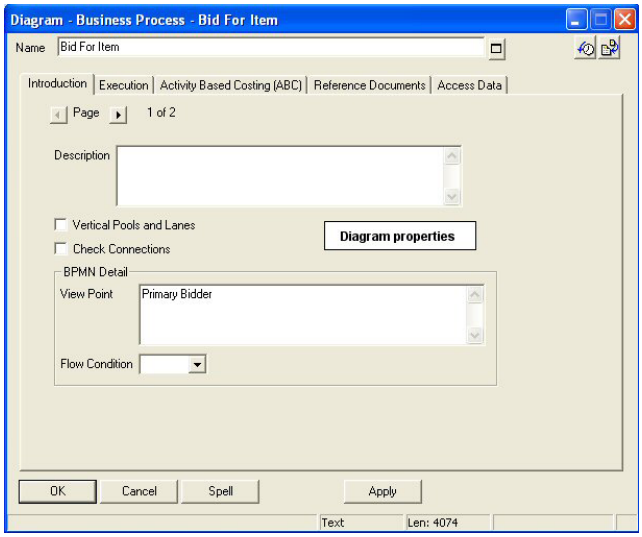
これらの機能に加えて、Rational System Architect の最も優れた特徴の 1 つとして、エンサイクロペディアに情報を保管する方法の基礎となるメタモデルを、ユーザーが調整および拡張することもできる点が挙げられます。Rational System Architect エンサイクロペディアのデフォルト・メタモデルは、SAPROPS.CFG (メイン **S**ystem **A**rchitect **プロパティー**・ファイル) というファイルに指定されています。このファイルによって、ダイアグラム上のシンボル、シンボルとその定義の関係、およびシンボル、定義、ダイアグラムのプロパティーなどが制御されます。ユーザーによるメタモデルへの変更は、USRPROPS.TXT という名前のテキスト・ファイルに指定されます。このファイルは、エンサイクロペディアがロードされるときに、SAPROPS.CFG とともに構文解析されて、SAPROPS.BIN ファイルが作成されます。USRPROPS.TXT は SAPROPS.CFG よりも優先されます。USRPROPS.TXT ファイルを編集して、Rational System Architect 固有のスクリプト言語を使用するエンサイクロペディアのメタモデルをカスタマイズまたは拡張することができます。

Rational System Architect のエンサイクロペディアのメタモデル

メタモデルが提供するもの

メタモデルとは、ユーザーが作業中に作成するダイアグラム、シンボル、および定義を、Rational System Architect が保管する方法のモデルです。Rational System Architect のメタモデルには、すべてのダイアグラム・タイプ、シンボル・タイプ、および定義タイプが含まれています。また、これらの各タイプに含まれるプロパティー、および各モデル化要素間のさまざまな関係も含まれています。

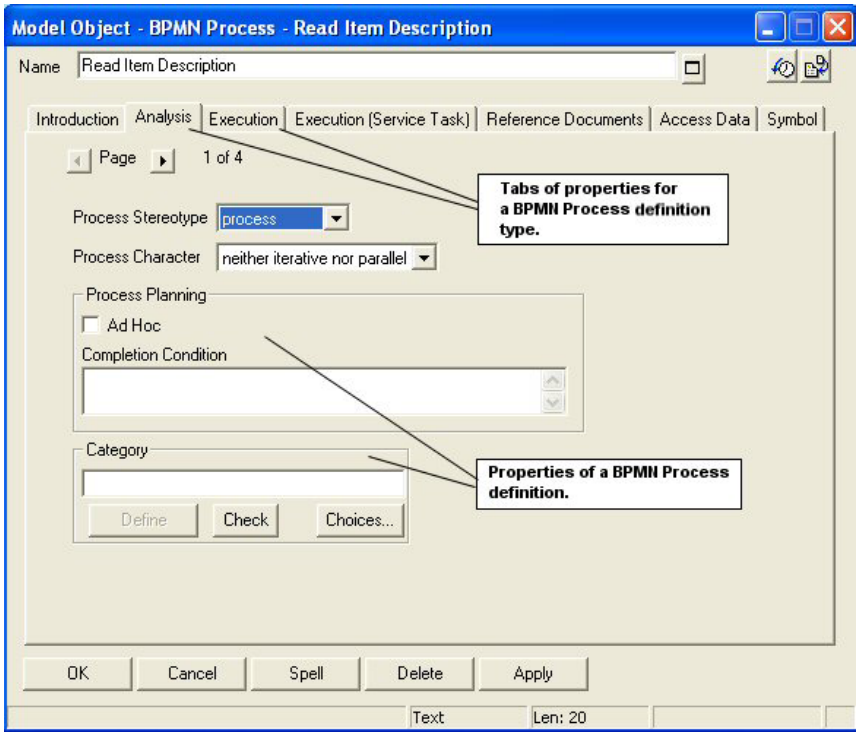
*ダイアグラム・タイプ*の一例として、ビジネス・プロセス図が挙げられます。このダイアグラムには、プールとレーンを水平に表示するか垂直に表示するかについてのオプション (「プールとレーンを垂直化」) や、ユーザーの描画に合わせてダイアグラム上でラインとシンボルの接続を自動的にチェックするかどうかについてのオプション (「接続チェック」) などの、*プロパティー*があります。



*シンボル・タイプ*の一例として、BPMN プロセス・シンボルが挙げられます。BPMN プロセス・シンボルは、ビジネス・プロセス図に描画されます。ダイアグラムはシンボルを*含み (contains)*、シンボルはダイアグラム*に含まれる (contained in)* ことになります。これは、エンサイクロペデ

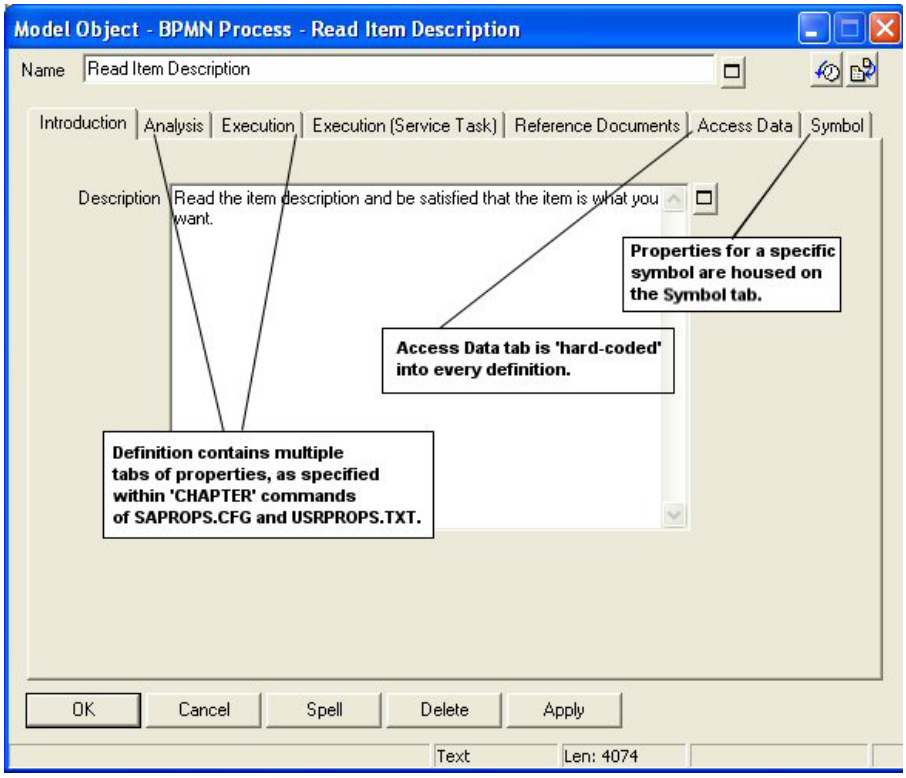
ィアのメタモデル内にある多数の関係のうちの 2 つを示す例です。*定義タイプ*の一例として、BPMN プロセス定義が挙げられます。BPMN プロセス・シンボルは、BPMN プロセス定義をグラフィカルに表現します。大部分の定義はシンボルによって表現されますが、例えば、属性定義タイプまたはメソッド定義タイプなど、一部の定義はダイアグラム上のシンボルで表現されません。これらは両方とも、クラス定義タイプ*に含まれる (included in)* ことになります (別の関係)。

ダイアグラム・タイプと同様に、各定義タイプにはプロパティーが含まれます。Rational System Architect のエクスプローラーから定義を開くと、定義のダイアログに、これらのプロパティーが該当するタブとグループに分類されて表示されます。

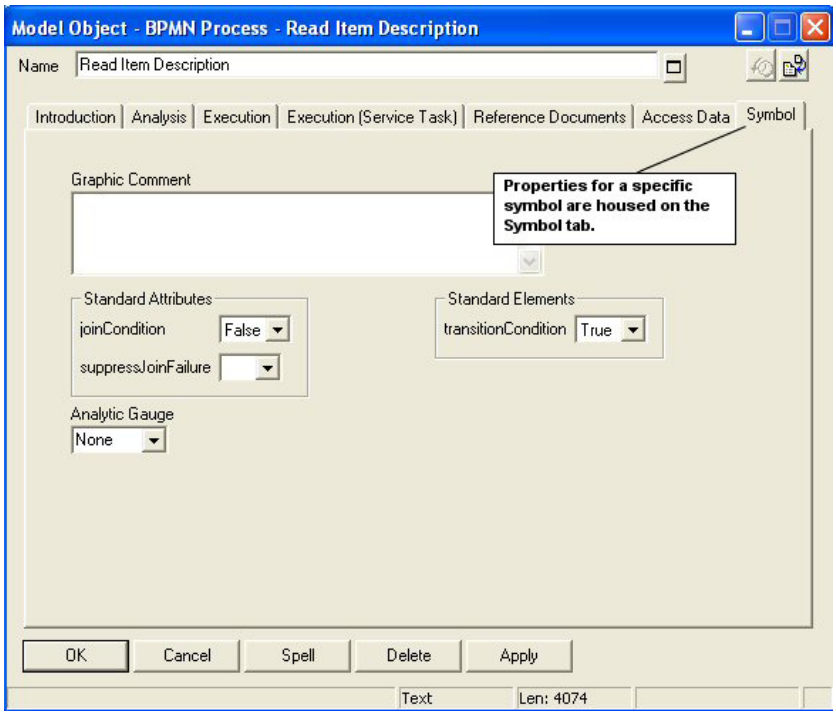


ダイアグラム・タイプおよび定義タイプと同様に、各シンボル・タイプにはプロパティーが含まれます。シンボルの基礎となるダイアログを開くと (ダイアグラム・ワークスペースでシンボルをダブルクリックするか、シンボルを右クリックして「編集」を選択するか、シンボルを選択して「編集」、「シンボル・タイプ」を選択します)、シンボルが表現する

基礎となる定義のプロパティーが表示され (エクスプローラーから定義を開いたときの表示と同様)、**さらに**追加の「シンボル」タブも表示されます。



「シンボル」タブには、そのシンボルに固有のプロパティーが表示されます。



ダイアグラムに描画する各シンボルは、同じ定義を指している別個のインスタンスです。このため、あるビジネス・プロセス図上に「Read Item Description」という名前のプロセス・シンボルを描画してそれを赤色にし、別のビジネス・プロセス図上に「Read Item Description」という名前のもう 1 つのプロセス・シンボルを描画してそれを緑色にすることができます。「Read Item Description」の定義に変更を加えると (例えば、「説明」プロパティーに単語を加えると)、赤色または緑色のいずれかの「Read Item Description」シンボルの定義を開いたときに、その変更が反映されます。2 つの別個のシンボルでも、基礎となる定義は 1 つです。

ダイアグラム・タイプ、シンボル・タイプ、および定義タイプ間の関係は複雑になる場合があります。例えば、Rational System Architect では、クラス図は、その図が作成されるパッケージに属します。クラスとパッケージの間には「belongs to」という関係があります。さらに、クラスは、そのクラスが属するパッケージ「に対してキー設定される (keyed to)」ことになります。この「keyed to」という関係によって、クラスの名前空間に固有性がもたらされます。例えば、Human_Resources パッケージ内の「人」というクラ

スが、Hotel_Reservation パッケージ内の「人」というクラスとは完全に異なる内容を持つようにすることができます。したがって、クラスとパッケージの間には「keyed by」という別の関係が存在します。同様に、メソッドは、パッケージに属するクラスに属します。また、メソッドは、パッケージによってキー設定されるクラスによってキー設定されます。さらに、ユーザーは、そのクラス図にクラス・シンボル用の子ダイアグラム (状態図など) を作成することができます。この場合、状態図は、クラス・シンボル「の子である (is child of)」ことになります (さらに別の関係)。

メタモデルを変更する方法

Rational System Architect は、ダイアグラム、シンボル、定義、プロパティー、および関係に関する事前設定されたメタモデルとともに提供されます。このメタモデルをそのまま使用することも、自分のモデリング・ニーズに合わせて拡張または調整することもできます。調整には、既に提供されているものへの変更や、ユーザー独自の新規ダイアグラム・タイプ、シンボル・タイプ、および定義タイプの追加といった作業が含まれます。

エンサイクロペディアのメタモデルの物理的構成 – SAPROPS.CFG および USRPROPS.TXT

Rational System Architect の各エンサイクロペディアには、SAPROPS.CFG (System Architect プロパティー構成ファイル) および USRPROPS.TXT (ユーザー・プロパティー・ファイル) という 2 つのファイルで指定される独自のメタモデルが含まれています。これらの 2 つのファイルは、各エンサイクロペディアの FILES テーブルにあります。

SAPROPS.CFG ファイルには、製品の特定のバージョンで使用される各エンサイクロペディア用に IBM によって指定されたデフォルト・メタモデルが含まれています。USRPROPS.TXT ファイルは、いくつかのコメント (REM、または覚書) のステートメントを除き、デフォルトで空のファイルです。ユーザーが、USRPROPS.TXT ファイルにコードを追加して、メタモデルを変更します。

Rational System Architect によってエンサイクロペディアが開かれると、SAPROPS.CFG ファイルの構文解析が行われた後、USRPROPS.TXT ファイルの構文解析が行われて SAPROPS.BIN ファイルが作成されます。SAPROPS.BIN ファイルが作成されるときに、USRPROPS.TXT に指定されたものはすべて、SAPROPS.CFG での指定をオーバーライドするか、その指定に追加されます。ユーザーにメタモデルを示すのに使用されるのは、SAPROPS.BIN ファイルです。

メタモデルには以下のような重要な項目がいくつかあり、これらは USRPROPS.TXT によって SAPROPS.CFG でオーバーライドすることはできません。

- SAPROPS に定義されている LIST または LISTONLY 参照は除去できません。ただし、リストまたはリスト・ボックスに表示されるテキストは変更できます。

- SAPROPS に定義されているラベルは除去できません。ただし、ラベルに表示されるテキストは変更できます。

「マスター」SAPROPS.CFG および USRPROPS.TXT ファイル

各エンサイクロペディアの FILES テーブルの他に、Rational System Architect のメイン実行可能ディレクトリー (通常、<C>:\Program Files\IBM\Rational\System Architect Suite\11.3.1\System Architect) にも、SAPROPS.CFG ファイルと USRPROPS.TXT ファイルの「マスター」コピーがあります。エンサイクロペディアが最初に作成されるときに、Rational System Architect の実行可能ディレクトリーにある「マスター」SAPROPS.CFG ファイルと USRPROPS.TXT ファイルが、その Files テーブルに自動的に配置されます。したがって、メイン Rational System Architect ディレクトリーにある USRPROPS.TXT ファイルの内容を変更すると、作成するすべての新規エンサイクロペディアのメタモデルに変更を加えることになります。このため、メイン Rational System Architect 実行可能ディレクトリーにある「マスター」USRPROPS.TXT に、会社およびプロジェクトの標準に必要なすべてのプロパティーが含まれるようにするのが一般的です。

最初は、「マスター」USRPROPS.TXT ファイルは基本的に空のファイルです。このファイルには、**REM** (覚書、またはコメント) コマンドが先頭に付けられたいくつかの注釈がファイルの最初の部分に含まれているだけです。

CONFIG.PRP ファイル

Rational System Architect には、CONFIG.PRP という名前の 3 番目のファイルがあります。このファイルは SAPROPS.CFG を完全にコピーしたものです。CONFIG.PRP は、Rational System Architect の実行可能ディレクトリー (通常、<C>:\Program Files\IBM\Rational\System Architect Suite\11.3.1\System Architect) にあります。CONFIG.PRP を使用すれば、SAPROPS.CFG 自体に誤って不要な変更を加える心配をせずに、SAPROPS.CFG にもあるコマンドおよびプロパティーの表示、切り取り、およびコピーを行うことができます。CONFIG.PRP ファイルからコマンドの切り取りまたはコピーを行って、それを USRPROPS.TXT ファイルに貼り付けてから、変更を行うことができます。

エンサイクロペディアのダイアグラム・セットおよびプロパティー・セットの選択

USRPROPS.TXT を使用してメタモデルを変更するほかに、Rational System Architect の「プロパティー構成」ダイアログ (「ツール」、「メソッド・サポートのカスタマイズ」、「エンサイクロペディアの構成」を選択してアクセスします) を使用して、エンサイクロペディアに対してどのダイアグラム・セットおよびプロパティー・セットをオンにするかをいつでも選択することができます。

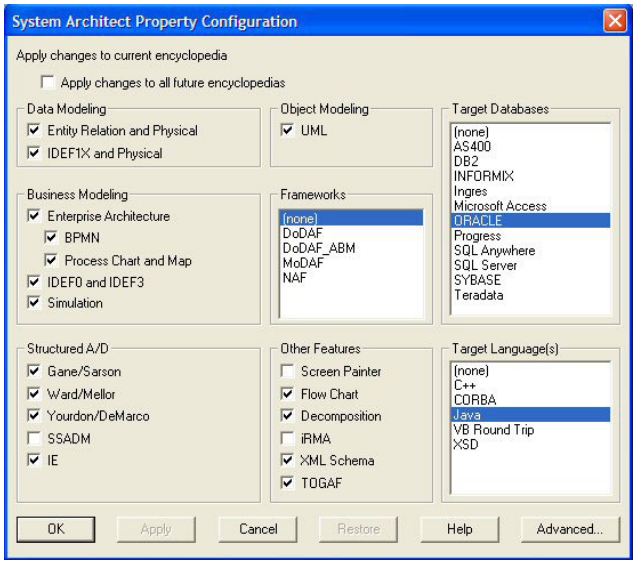


図 1-1。「プロジェクト構成」ダイアログ: ダイアグラム・タイプと、このエンサイクロペディアで有用なその他のダイアグラムを選択します。

ダイアグラム・セットおよびプロパティー・セットのオン/オフを切り替えたり、このダイアログの「詳細」ボタンをクリックしたりすることで、エンサイクロペディアでどのダイアグラム・セットとプロパティー・セットをアクティブにするかについてさらに改良を加えることができます。

SADECLAR.CFG

「プロパティー構成」ダイアログで行った選択は、SADECLAR.CFG ファイルの内容に直接影響します。このファイルは、すべてのエンサイクロペディアの Files テーブルにあります。次にこのファイルは、SAPROPS.CFG ファイルおよび USRPROPS.TXT ファイルの #IFDEF (注: 「#」と「IF」の間にスペースはありません) ステートメントによっ

て参照されます。例えば、SAPROPS.CFG ファイルには UML 用の #IFDEF が含まれており、UML モデリング手法がオンに切り替えられると (上記のダイアログにおいて、したがって SADECLAR.CFG において)、SAPROPS.CFG の #IFDEF は、UML ダイアグラム用の該当するプロパティーをオンまたはオフにします。

下記の図に示されているように、「プロパティー構成」ダイアログで行うダイアグラム・セットおよびプロパティー・セットに対する選択によって (これによって SADECLAR.CFG での選択のオン/オフが切り替えられます)、実際には、エンサイクロペディアで使用される SAPROPS.CFG ファイル・プロパティーにフィルターが掛けられます。USRPROPS.TXT に行うユーザー変更は、フィルターに掛けられた SAPROPS.CFG ファイルとともに構文解析され、SAPROPS.BIN ファイルが生成されます。この SAPROPS.BIN ファイルによって、Rational System Architect の実行中にエンサイクロペディアのメタモデルが提供されます。プロパティー・ダイアログまたは定義ダイアログを立ち上げるか、レポートを実行すると、Rational System Architect は常に SAPROPS.BIN に移動し、ユーザーが定義しているモデル要素の関連プロパティーを検索します。

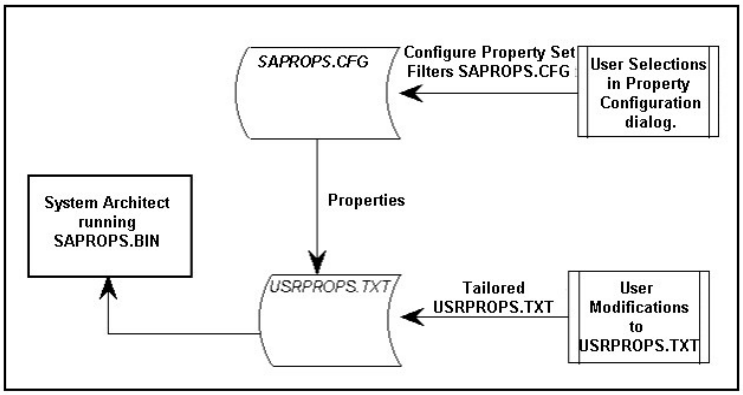


図 1-2。SAPROPS.CFG、USRPROPS.TXT、およびユーザーが選択したダイアグラム、プロパティー、モデリング手法の間の関係。

エンサイクロペディアの Files テーブルか SADECLAR.CFG ファイルをエクスポートし、任意のテキスト・エディターを使用してそのファイルを開くことにより、USRPROPS.TXT の #IFDEF ステートメントのスイッチとして使用できる特定のプロパティー・セットを確認することができます。

「プロパティー構成」ダイアログで使用されている語/ラベルと完全に一致しないものもあります。例えば、Enterprise Architecture という選択項目は、SADECLAR.CFG では実際には Business Enterprise と呼ばれます。このため、USRPROPS.TXT で #IFDEF "Enterprise Architecture" というステートメントを使用しても意味がなく、構文解析エラーを生じます。正しいステートメントは、#IFDEF "Business Enterprise" です。

図 1-3。SADECLAR の内容は、USRPROPS.TXT の #IFDEF スイッチに使用されます。

```
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

USRPROPS.TXT を使用したメタモデルの変更

はじめに

この章では、Rational System Architect の拡張可能メタモデルの背景にある理論とメカニズムについて説明します。

USRPROPS.TXT ファイルへのアクセスと編集

USRPROPS.TXT ファイルは、任意のテキスト・エディターで編集できます。要件としては、TEXT ファイルとして保存しなければならないということがあります。

マスター USRPROPS.TXT ファイルへのアクセス

前にこの章で述べたように、マスター USRPROPS.TXT ファイルはユーザーが作成したいずれかの新しいエンサイクロペディアの中に自動的に置かれます。多くの組織では、すべての新しいエンサイクロペディアに同じメタモデル拡張が含まれるよう、マスター USRPROPS.TXT ファイルを変更します。マスター USRPROPS.TXT ファイルを編集するには、以下のようにします。

- 「ツール」、「ユーザー設定のカスタマイズ」、「USRPROPS.TXT の編集 (マスター)」を選択します。または、
- 単に <C>:\Program Files\IBM\Rational\System Architect Suite\11.3.1\System Architect ディレクトリーまでナビゲートし、そこにある USRPROPS.TXT ファイルを開きます。

エンサイクロペディアの USRPROPS.TXT ファイルへのアクセス

エンサイクロペディアの USRPROPS.TXT ファイルは、エンサイクロペディアの SQL Server データベース内にある Files テーブルに置かれています。これを編集するには、最初にこのファイルをデータベースの Files テーブルからエクスポートする必要があります。次に、編集を行った後、そのファイルを元のデータベースの Files テーブル内にインポートして戻し、エンサイクロペディアを再オープンする必要があります (これにより、SAPROPS.CFG および USRPROPS.TXT ファイルを構文解析できます)。

エンサイクロペディアの USRPROPS.TXT ファイルにアクセスする方法は、複数あります。

- Rational System Architect のネイティブ USRPROPS.TXT エクスポート/インポート機能を使用できます (「ツール」、「ユーザー設定のカスタマイズ」、「USRPROPS.TXT ファイルをエクスポート (エンサイクロペディア)」を選択します)。または、

- Rational System Architect の「**エンサイクロペディア・ファイル・マネージャー**」ユーティリティーを使用できます (「ツール」、「エンサイクロペディア・ファイル・マネージャー」を選択します)。または、
- **SAEM** を使用できます (Rational System Architect の外部から、「スタート」、「プログラム」、「IBM Rational」、「IBM Rational Lifecycle Solutions Tools」、「IBM Rational System Architect 11.3.1」、「SAEM」を選択し、SAEM のヘルプを参照してください)。

Rational System Architect のネイティブ USRPROPS.TXT エクスポート/インポート機能の使用:

Rational System Architect のネイティブ USRPROPS.TXT エクスポート/インポート機能を使用して USRPROPS.TXT ファイルを編集するには、以下のステップを実行します。

1. 「ツール」、「ユーザー設定のカスタマイズ」、「USRPROPS.TXT ファイルをエクスポート (エンサイクロペディア)」を選択します。
2. 表示される「**ユーザー・プロパティーのエクスポート**」ダイアログで、USRPROPS.TXT ファイルのエクスポート先にするディレクトリーを選択します。「保存」ボタンをクリックします。USRPROPS.TXT ファイルは選択したディレクトリーに保存され、「メモ帳」で自動的に開かれます。
3. ファイルを編集後、「ツール」、「ユーザー設定のカスタマイズ」、「USRPROPS.TXT のインポート (エンサイクロペディア)」を選択し、変更した USRPROPS.TXT ファイルをエンサイクロペディアのデータベースの Files テーブル内に再インポートします。
4. SAPROPS.CFG ファイルと変更した USRPROPS.TXT ファイルを構文解析するために、エンサイクロペディアを再オープンします。

エンサイクロペディア・ファイル・マネージャーの使用:

エンサイクロペディア・ファイル・マネージャーを使用して USRPROPS.TXT ファイルを編集するには、以下のステップを実行します。

1. 「ツール」、「エンサイクロペディア・ファイル・マネージャー」を選択します。
2. 「エンサイクロペディア・ファイル・マネージャー」ダイアログで、左下隅の「エクスポート」選択項目がオンに切り替えられていることを確認します。「ファイルの選択」リストで USRPROPS.TXT ファイルを選択し、「エクスポート先」プロパティーの「…」ボタンを使用して、ファイルのエクスポート先にするディレクトリーを選択します。
3. テキスト・エディターでファイルを変更し、「エンサイクロペディア・ファイル・マネージャー」を使用してファイルを元のエンサイクロペディアのデータベースの Files テーブルにインポートして戻します。
4. SAPROPS.CFG ファイルと変更した USRPROPS.TXT ファイルを構文解析するために、エンサイクロペディアを再オープンします。

SAEM の使用:

SAEM を使用して、エンサイクロペディアの USRPROPS.TXT ファイルにアクセスし、編集することもできます。サーバーに接続し、データベースを選択し、データベースとの間でファイルをエクスポート/インポートする方法については、SAEM のヘルプを参照してください。

プロパティー・ファイルの再ロード

上記の手順で述べたように、変更した USRPROPS.TXT をエンサイクロペディアに再インポートするたびに、Rational System Architect を使用してエンサイクロペディアを再オープンする必要があります。エンサイクロペディアを再オープンすると、エンサイクロペディアの SAPROPS.CFG ファイルと USRPROP.TXT ファイルが構文解析され、SAPROPS.BIN (バイナリー) ファイルが作成されます。このファイルは、メタモデルを提示するために使用されるファイルです。エラーがなければ、メタモデルへの変更が即時に有効になります。

USRPROPS.TXT ファイルの構文解析で Rational System Architect が USRPROPS.TXT コード内の誤りを検出した場合は、警告またはエラーのメッセージが発行されます。Rational System Architect は警告の発行後にエンサイクロペディアを開きますが、エラーを検出した場合はエンサイクロ

ペディアを開き**ません**。以下に示すようなメッセージが表示されます。

図 2-1。プロパティー・ファイルのエラー・ダイアログ

エラーが検出された場合は、Rational System Architect のネイティブ USRPROPS.TXT エクスポート/インポート機能を使用して問題のある USRPROPS.TXT ファイルにアクセスすることができなくなります (「ツール」、「ユーザー設定のカスタマイズ」を選択した場合、「USRPROPS.TXT のエクスポート (エンサイクロペディア)」選択項目は淡色表示になります)。

エラーの発生後に USRPROPS.TXT ファイルにアクセスし、編集するには、Rational System Architect のエンサイクロペディア・ファイル・マネージャー (「ツール」、「エンサイクロペディア・ファイル・マネージャー」を選択) または SAEM (Rational System Architect の外部から「スタート」、「プログラム」、「IBM Rational」、「IBM Rational Lifecycle Solutions Tools」、「IBM Rational System Architect 11.3.1」、「SAEM」を選択して SAEM のヘルプを参照) を使用する必要があります。

構成と構文

ほとんどのプログラミング言語と同様に、USRPROPS.TXT の言語構文も一連のストリングから構成されています。ストリング同士を区切るために、少なくとも 1 つの*空白*文字が必要です (空白文字には、スペース、タブ、コンマ、復帰/改行文字などが含まれます)。復帰文字の直後にタブがあるなど、複数の空白文字が連続している場合、それらは一まとまりにされ、1 つの空白文字として扱われます。

ストリングに 1 つ以上のスペースが含まれている場合は、必ずそのストリングを二重引用符で囲みます。例えば、**Data Flow** とせずに、**"Data Flow"** のようにします。

キーワード

USRPROPS.TXT 言語には、一連の***キーワード***があります。キーワードは、その配置によって、***コマンド***または***引数***と見なされます。USRPROPS.TXT 内で使用できるすべてのキーワードは、『**第 3 章 USRPROPS.TXT キーワード**』にリストされています。

キーワードの大/小文字の区別

USRPROPS.TXT のキーワードには大/小文字の区別が***ありません***。このため、大文字、小文字、またはそれらを混用してもかまいません。このマニュアルとサンプルの USRPROPS.TXT ファイルでは、コマンドとそれ以外のすべてのキーワードがすべて大文字になっていますが、これは単に、読みやすくするためです。コマンドの例を以下に示します。

BEGIN、または Begin または BegiN
EDIT または Edit
LIST または List または LiST
LISTONLY または Listonly または ListOnly
RENAME または Rename または ReName など

コマンド

コマンドは、*常に*キーワードであり、*常に*新しい句の始まりを示します。Rational System Architect は、USRPROPS.TXT を構文解析する場合、ファイル内の最初のストリングが有効なキーワード・コマンドでなければならないことを認識しています。各コマンドの直後には、既知の数の引数ストリング (ゼロか 1 つ以上) が続いている必要があり、その後に別のコマンドが検出されなければなりません。

引数

コマンドの直後に続くストリングは、引数です。一部の引数はキーワードである場合があります。それ以外の引数はテキ

スト・ストリングからなり、それらのストリングは、後続のダイアログ内にある「ダイアグラム」、「シンボル」、「定義」、「プロパティー」、「リスト値」、「ラベル」、「ヘルプ・ストリング」などの名前を提供します。以下に、いくつかの例を示します。

- LIST **"Processor Scheduling"**

  "Processor Scheduling" はキーワードではありません。これは上記の式の中で引数として使用されています。

- DISPLAY { FORMAT **KEY** LEGEND "Key data" }

  "KEY" はキーワードです。これは上記の式の中で引数として使用されています。

テキスト・ストリングである引数の大/小文字の区別

前に述べたように、キーワードに大/小文字の区別は***ありません***。しかし、テキスト・ストリングである引数には、大/小文字の区別が***あります***。 例えば、上記の LIST "Processor Scheduling" 引数を使用すると、SAPROPS.CFG と USRPROPS.TXT のどちらの中でも、このリストへの参照には、大/小文字の区別を同じにして、まったく同じスペルを使用する必要があります。例えば、以下のリストを指定したとします。

```
LIST "Processor Scheduling"
{
VALUE"preemptive"
VALUE"nonpreemptive"
}
```

この場合、このリストへの有効な参照には、まったく同じスペルが必要です。

```
DEFINITION "Hardware Processor"
PROPERTY "Scheduling"
{ EDIT text LIST "Processor Scheduling" LENGTH 20
DISPLAY { LEGEND "Sched" } }
```

しかし、次の構文は、「リスト "PROCESSOR SCHEDULING" がありません。」というエラー・メッセージを返します。

```
DEFINITION "Hardware Processor"
PROPERTY "Scheduling"
{ EDIT text LIST "PROCESSOR SCHEDULING" LENGTH 20
DISPLAY { LEGEND "Sched" } }
```

同様に、レポートの中で参照されるすべてのプロパティーには、SAPROPS.CFG ファイルおよび USRPROPS.TXT ファイル、あるいはそのいずれかの項目のスペルと大/小文字を使用する必要があります。

モデル化要素を作成するための コマンドのグループ化

左右の中括弧、**{ }**、またはそれに代わる BEGIN...END コマンドは、モデル化要素を形成するために、複数のコマンドをグループ化するのに使用されます。

ダイアグラム、シンボル、および定義

Rational System Architect のリポジトリーは、3 つの主要なモデル化要素をサポートしています。すなわち、*辞書クラス*とも呼ばれる、***ダイアグラム***、***シンボル*** (ダイアグラム上に描画されます)、および***定義*** (シンボルによって表される場合もそうでない場合もあります) です。BEGIN .. END または { } 構造は、それらのモデル化要素の内容を指定するために、以下のように使用されます。

Diagram "Name of Diagram Type**"**
{
[contents]
}

Symbol "Name of Symbol Type**"**
{
[contents]
}

Definition "Name of Definition Type**"**
{
[contents]
}

または

Diagram "Name of Diagram Type**"**
BEGIN
[contents]
END

など

プロパティー

これらのモデル化要素の内容は、プロパティー・コマンドとレイアウト・コマンドによって構成されます。BEGIN .. END または { } 構造は、プロパティー・コマンドをグループ化するために、以下のように使用されます。

Definition **"**Name of Definition Type**"**
{
PROPERTY { [specification of property] **}**
PROPERTY { [specification of property] **}**

…
}

プロパティー内で節を作成する特定のキーワードも、例えば KEYED BY コマンドなどのコマンドの引数を線引きするために、左右の中括弧を必要とします。

```
Definition “Name of Definition Type”
{
PROPERTY { [specification of property] KEYED BY
{ [clause] } }
…
}
```

レイアウト

LAYOUT コマンドも、 左右の中括弧または BEGIN .. END ステートメントを必要とします。

```
Definition “Name of Definition Type”
{
LAYOUT { [specification of layout] }
PROPERTY { [specification of property] }
PROPERTY { [specification of property] }
…
}
```

章

ダイアログ内のプロパティーを、さらにタブとグループにグループ化できます。タブは CHAPTER コマンドによって指定されます。CHAPTER コマンドは左右の中括弧も BEGIN .. END ステートメントも必要とせず、実際には、それらを***持っていてはなりません***。CHAPTER コマンドは単に、指定の中でそのコマンドより下にあるすべてのプロパティーを 1 つの (後続のダイアログ内の) タブにグループ化します。グループ化は、その指定の中で次の CHAPTER コマンドが検出されるまで行われます。

```
Definition “Name of Definition Type”
{
LAYOUT { [specification of layout] }
CHAPTER “First Tab”
PROPERTY { [specification of property] }
PROPERTY { [specification of property] }
…
CHAPTER “Second Tab”
PROPERTY { [specification of property] }
PROPERTY { [specification of property] }
…
}
```

グループ

CHAPTER コマンドとは異なり、GROUPS は左右の中括弧または BEGIN .. END ステートメントを***必要***とします。

```
Definition “Name of Definition Type”
{
LAYOUT { [specification of layout] }
CHAPTER “First Tab”
PROPERTY { [specification of property]  }
PROPERTY { [specification of property]  }
GROUP “Things That Go Together”
{
PROPERTY { [specification of property]  }
PROPERTY { [specification of property]  }
}
…
}
```

リスト

エンサイクロペディアの中に、リスト (テキスト値を含んでいる事前設定リストや、ユーザーがモデル化のときに作成した定義のリスト) を指定することもできます。ユーザーがモデル化のときに作成する定義のリストは、プロパティー・ステートメント内の ONE OF、LISTOF、および EXPRESSIONOF コマンドを使用して構築されます。これについては、あとで説明します。テキスト・リストは、別個のリスト・ステートメント内でリストの値を指定することによって構築されます。その場合、値を左右の中括弧または BEGIN.. END 構造で囲みます。通常、LIST ステートメントは USRPROPS.TXT ファイルの先頭の近くに置かれ、「ダイアグラム」、「シンボル」、または「定義」の適切なプロパティー指定の中で参照されます。

```
LIST “List of Things”
{
VALUE One
VALUE Two
VALUE “Two and a Half”
}
```

構文に対する注

インデントと改行は単に読みやすくするために使用され、USRPROPS.TXT プロセッサーに対しては、ストリング同士を区切る空白区切り記号として機能する以外、何も意味がありません。上記の例は、以下のように書くこともできます。

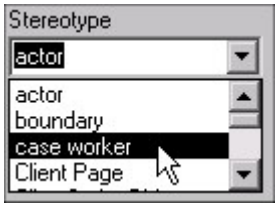
```
LIST "List of Things"  {   VALUE One   VALUE
Two   VALUE “Two and a Half”   VALUE }
```

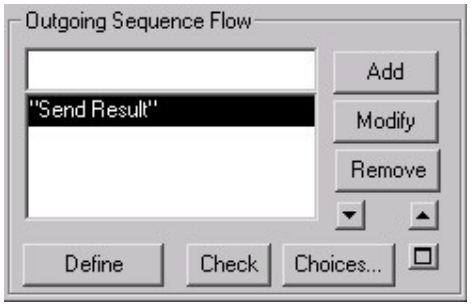
この形式でも Rational System Architect に完全に受け入れられますが、USRPROPS.TXT ファイルの保守を難しくする可能性があるので、避けるようにしてください。

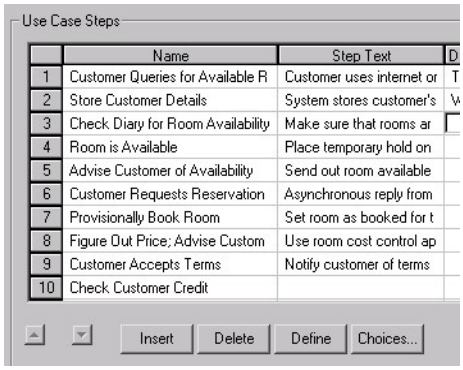
ダイアログ・コントロール

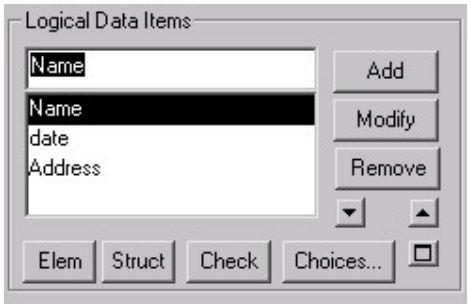
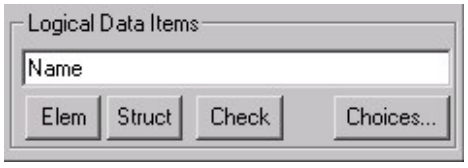
次の表は、USRPROPS.TXT 内の該当するコマンドによって作成できるダイアログ・コントロールを説明したものです。

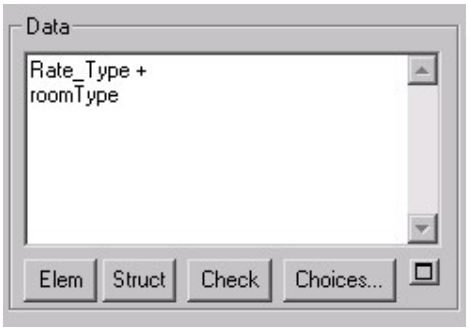
表 2-2。プロパティー式によって生成されるコントロール

引数タイプ	生成されるコントロール
値が 5 個未満の LIST コマンド。	1 つの値に 1 つの**ラジオ**・ボタンがあるグループ・ボックス。  **重要な注:** LISTONLYCOMBO コマンドを使用すると、値が 5 個未満のリストでも強制的にドロップダウン・リストにすることができます。このキーワードの詳細については、第 3 章に説明があります。
値が 5 個以上の LIST コマンド	ドロップダウン・リスト・ボックス。 
BOOLEAN	チェック・ボックス。TRUE 値はチェック・マークによって表され、FALSE 値は空のボックスです。例えば、下記のプロパティー「Virtual」はブール値として記述されています。 PROPERTY "Virtual" { EDIT **BOOLEAN** LENGTH 1 DEFAULT "F" } 

LISTOF “[Definition or Diagram Type]”	ドロップダウン・リスト・ボックスと、側面に「**新規** 」、「**追加**」、「**除去 (Remove)**」、「**下矢印**」、「**上矢印**」の各ボタン、および下部に「**定義**」、「**チェック**」、および「**選択**」の 3 つのボタンがあるグループ。 
ONEOF “[Definition or Diagram Type]”	テキスト・ボックスと、「**定義**」、「**チェック**」、「**選択** 

ASGRID	このコマンドは、LISTOF または ONEOF コマンドと一緒に使用され、値のグリッドを提供します。以下に例を示します。 PROPERTY "Use Case Steps" { EDIT COMPLETE LISTOF "Use Case Step" KEYED BY { "Package", "Use Case Name":Name, Name} **ASGRID** LENGTH 1200 } 
EXPRESSIONOF “[Definition or Diagram Type]”	不可能。

LISTOF DATA	DATA は特殊語で、エンサイクロペディア内のデータ要素とデータ構造 (それぞれのデータ構造はデータ要素のグループです) のリストを提供します。LISTOF DATA コントロールは非常に特殊なコントロールで、ドロップダウン・リスト・ボックスと、側面に「**新規**」、「 **追加** 」、「**除去 (Remove)**」、「**下矢印**」、「**上矢印**」の各ボタン、および下部に「**要素**」、「**構造**」、「**チェック**」、「**選択**」の 4 つのボタンを含んでいるグループです。 
ONEOF DATA	上で述べたように、DATA は特殊語で、エンサイクロペディア内のデータ要素とデータ構造のリストを提供します。ONEOF DATA 節は、ユーザーがデータ要素またはデータ構造を指定できるテキスト・ボックスを中心として、「**要素**」、「**構造**」、「**チェック**」、「**選択**」の 4 つのボタンを持つグループ・コントロールを提供します。 

EXPRESSIONOF DATA	上で述べたように、DATA は特殊語で、エンサイクロペディア内のデータ要素とデータ構造のリストを提供します。EXPRESSIONOF DATA は、ユーザーがデータ要素またはデータ構造を入力できるテキスト域と、「**要素**」、「**構造**」、「**チェック**」、「**選択**」の 4 つのボタンを提供します。 

USRPROPS.TXT の変更の順序とレイアウト

SAPROPS.CFG の各セクションの一般的な順序は、以下のとおりです。

- **LIST** コマンド・セクション
- **DIAGRAM** コマンド・セクション
- **SYMBOL** コマンド・セクション
- **DEFINITION** コマンド・セクション

  LAYOUT コマンド・サブセクション (定義ダイアログ全体のデフォルト)

  CHAPTER コマンド・サブセクション

  GROUP コマンド・サブセクション

  LAYOUT コマンド・サブセクション

  PROPERTY コマンド・サブセクション

USRPROPS.TXT 内のすべての項目はオプションですが、SAPROPS.CFG と同様なレイアウトに従う必要があり、**RENAME** コマンド・セクションを使用する場合は、それをファイルの先頭に追加します。USRPROPS.TXT の各セクションの一般的な順序は、以下のようにしてください。

- **RENAME** コマンド・セクション (このセクション内で USER DIAGRAMS、USER SYMBOLS、および USER DEFINITIONS を名前変更して、ユーザー独自のダイアグラム・タイプ、シンボル・タイプ、または定義タイプを作成します) (2-31 ページを参照)
- **LIST** コマンド・セクション (2-29 ページを参照)
- **DIAGRAM** コマンド・セクション (2-59 ページを参照)
- **SYMBOL** コマンド・セクション (2-61 ページを参照)
- **DEFINITION** コマンド・セクション (2-65 ページを参照)

CHAPTER コマンド・セクション (2-93 ページを参照)

GROUP コマンド・セクション (2-96 ページを参照)

LAYOUT コマンド・セクション (2-96 ページを参照)

PROPERTY コマンド・セクション (2-68 ページを参照)

.

USRPROPS.TXT の変更規則

USRPROPS.TXT を作成するときは、以下の規則に留意してください。

1. USRPROPS.TXT 項目は、SAPROPS.CFG 内の項目に追加されるか、それらを置き換えます。
2. USRPROPS.TXT 項目は、関連する LIST、RENAME、DIAGRAM、SYMBOL、または DEFINITION ステートメントで始まる必要があります。
3. SAPROPS.CFG に*追加* される USRPROPS.TXT 項目は、関連するセクションの末尾に置かれます。例えば、SAPROPS.CFG 内にない LIST ブロックは、原則的に SAPROPS.CFG 内にある他のすべての LIST ブロックの後に追加されます。
4. **CHAPTER** コマンドが含まれている場合を除いて、USRPROPS.TXT 項目は関連するダイアログの末尾に置かれます。例えば、クラス定義の新しいプロパティーは、そのクラスの定義ダイアログ内にある他のすべてのプロパティーの後に追加されます。
5. 既に SAPROPS.CFG 内にある **CHAPTER** コマンドが USRPROPS.TXT に含まれている場合、USRPROPS.TXT 項目は既存の章 (またはタブ) の末尾に置かれます。
6. 既に SAPROPS.CFG 内にある **GROUP** コマンドが USRPROPS.TXT に含まれている場合、USRPROPS.TXT 項目は既存のグループの末尾に置かれます。

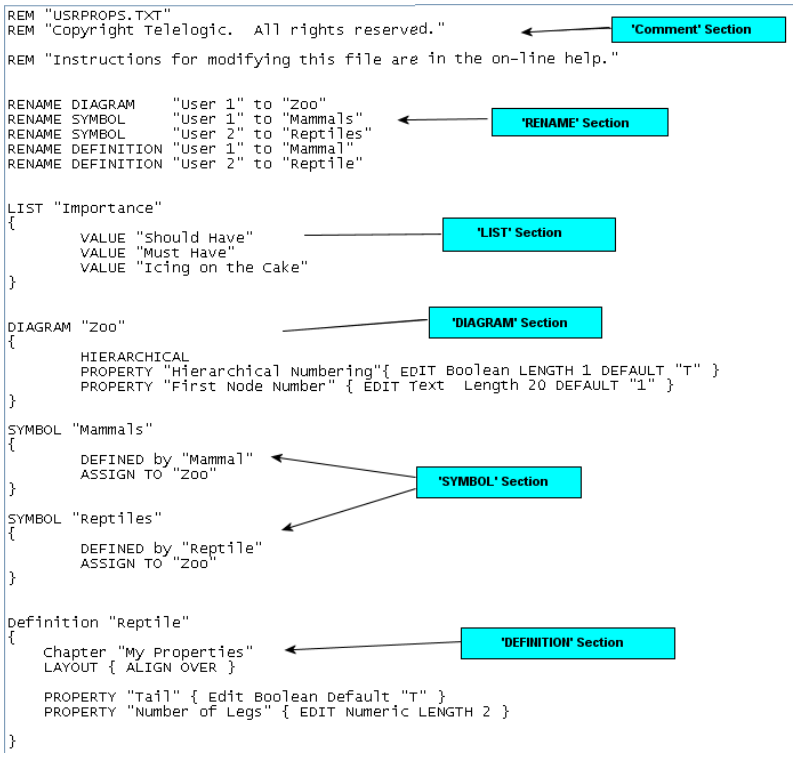
7. **GROUP** コマンドは、標準 Windows コントロールであるグループ・ボックスを生成します。後続のすべてのコントロールは、その中に配置する必要があります。項目の数が多すぎて、グループのサイズがモニターのサイズより大きい場合、余分のプロパティーは組み込まれず、表示されません。それについての警告メッセージは、エンサイクロペディアを開いたときに表示されます。
8. あるグループにプロパティーを追加する場合、SAPROPS.CFG 内にそのプロパティーに対する **PLACEMENT** コマンドが存在するときは、USRPROPS.TXT 内で追加する新しいプロパティーにも **PLACEMENT** コマンドを使用する必要があります。

USRPROPS.TXT コードのレイアウト

USRPROPS.TXT ファイルと SAPROPS.CFG ファイルが両方ともない場合でも、すべてのダイアグラム、シンボル、および定義に*名前*とプロパティーの*説明*があります。*説明*のデフォルト値は、このセクションのあとの部分に記載されています。前に述べたように、SAPROPS.CFG の完全なテキストは、CONFIG.PRP というファイルに含まれています。これは標準 ASCII テキスト・ファイルです。項目は USRPROPS.TXT に対する変更と追加のモデルとして使用できます。

実際のコードを USRPROPS.TXT ファイル自体にどのようにレイアウトするかは、ユーザーの自由です。推奨される方法は、「リスト」、「ダイアグラム」、「シンボル」、および「定義」ステートメントの開始が分かりやすくなるように、タブ構造を使用することです。ただし、タブの表現方法は、テキスト・エディターによって異なる場合があります。例えば、Microsoft Word をテキスト・エディターとして使用している場合、あとで別のテキスト・エディターで USRPROPS.TXT を開くと、Word で設定したタブがまったく異なる間隔になることがあります。

図 2-2。
USRPROPS.TXT の
コード・レイアウト
の例



USRPROPS.TXT の変更の例

このセクションでは、既に SAPROPS.CFG に存在する定義に変更を加えます。以下のコードが SAPROPS.CFG 内にあります。

```
DEFINITION "Change Request"
{
ADDRESSABLE
LAYOUT { COLS 2 TAB ALIGN LABEL }
PROPERTY "Impact Statement" { EDIT Text LENGTH 1000 }
PROPERTY "Original Source" { EDIT Text LIST "Business Unit"
    LENGTH 80   LABEL "Source Dept." }
PROPERTY "Author Name" { EDIT Text LENGTH 25 }
PROPERTY "Date Entered" { EDIT date INITIAL date READONLY
    LENGTH 10 }
PROPERTY "Start Date" { EDIT date LENGTH 10 }
PROPERTY "Required Completion Date"
  { EDIT date LENGTH 10  LABEL "Required Completion" }
```

下の図は、上記の定義ブロックの「**辞書オブジェクト**」ダイアログを示しています (このマニュアルでは、多くの場合、このダイアログを非公式に**定義** ダイアログと呼んでいるので、注意してください)。

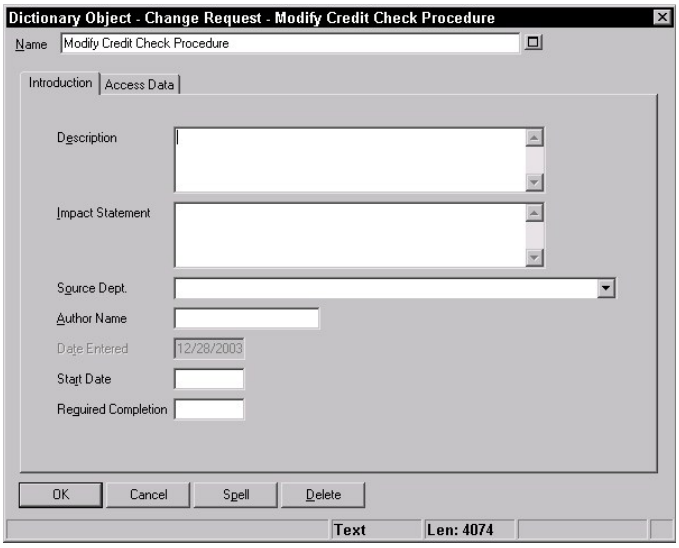


図 2-3。マスター構成プロパティー・セット・ファイル内で定義されている「変更依頼」定義ダイアログ

SAPROPS.CFG で CHAPTER コマンドを使用して特に呼び出さなくても、「**はじめに**」タブが存在することに注意してください。CHAPTER コマンドが指定されていない場合、

Rational System Architect は自動的にデフォルトの「**はじめに**」タブを提供します。「**アクセス・データ**」タブはソフトウェア内にハードコーディングされており、SAPROPS.CFG 内では指定されません。

USRPROPS.TXT での変更

ここでは、以下のコードを USRPROPS.TXT に追加してエンサイクロペディアを再オープンすることによって、「変更依頼」定義を変更します。

```
DEFINITION "Change Request"
{
LAYOUT { COLS 2 TAB ALIGN LABEL }
PROPERTY "Author Name" { LABEL "Client Division" }
PROPERTY "Supervising Manager" { EDIT text LENGTH 45 }
PROPERTY "On time" { Edit Boolean Length 1 DEFAULT "T" }
}
```

次の表は、上記の USRPROPS.TXT コードの各行と、その効果を説明したものです。

表 2-1。 USRPROPS.TXT 項目の効果

USRPROPS.TXT 項目	**効果**
DEFINITION "Change Request" {	「定義」"変更依頼 (Change Request)" に対する変更を指定します。
LAYOUT { COLS 2 TAB ALIGN LABEL }	LAYOUT コマンドの下にあるプロパティーの 2 列のレイアウトをセットアップします (定義の終わりに到達するまで、または別の LAYOUT コマンドが検出されるまで。
PROPERTY "Author Name" { LABEL "Client Division" }	これは既存のプロパティーを変更します。つまり、フィールドのラベルを「*Author Name*」から「*Client Division*
PROPERTY "Supervising Manager" { EDIT TEXT LENGTH 45 }	これは、新しいプロパティーであるテキスト・フィールド「*Supervising Manager*」をダイアログ・ボックスに追加します。

PROPERTY "On time" { EDIT BOOLEAN LENGTH 1 DEFAULT "T" } }	これは、変更依頼が期限に間に合っているかどうかを示すために、新しいプロパティーであるチェック・ボックスをダイアログ・ボックスに追加します。

ここでは、変更した USRPROPS.TXT ファイルを Rational System Architect エンサイクロペディアにインポートし、変更依頼の定義を再オープンしてダイアログの変更を確認します。「**はじめに**」タブの情報が 2 ページになったことに注意してください。

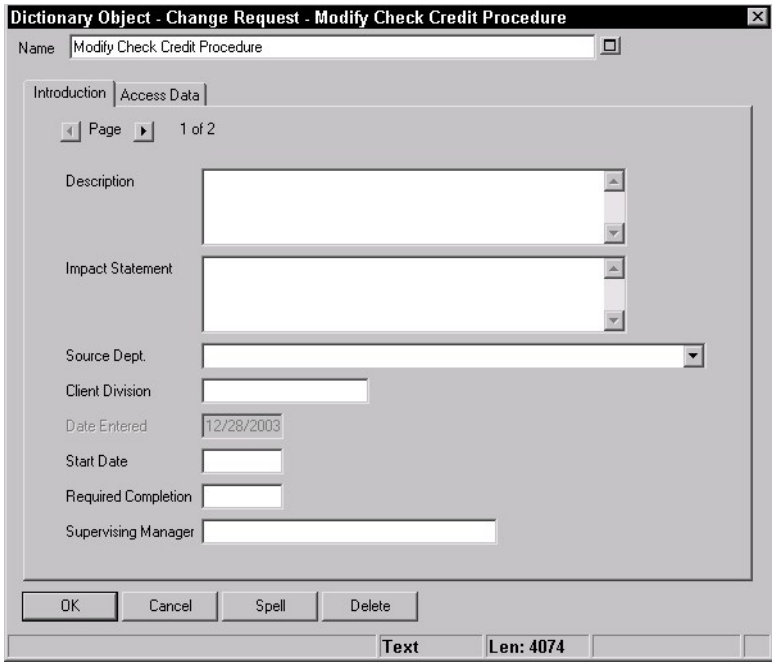
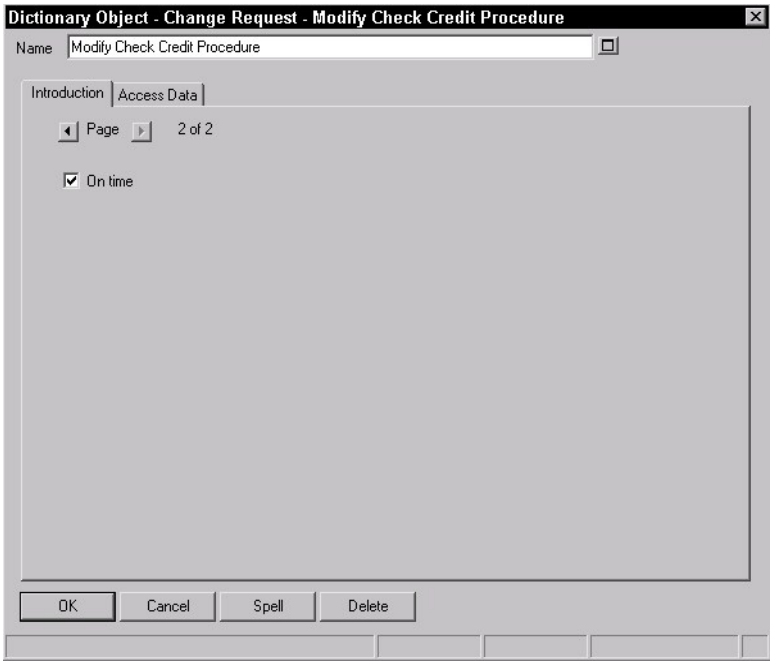


図 2- 4。
USRPROPS.TXT 内の項目によって変更された変更依頼定義ダイアログ



「**はじめに**」タブの情報が 2 ページになった理由は、2 つの新しいプロパティーが追加されたことにあります。それらは定義の末尾に追加されました (「**アクセス・データ**」タブの末尾に追加されなかったのは、このタブがカウントされないためです。このタブはハードコーディングされており、SAPROPS.CFG の一部ではありません)。

変更する必要があるものだけを変更する

ここでは、SAPROPS.CFG 内に存在する PROPERTY ステートメントの全体を USRPROPS.TXT ファイルに再入力していないことに注意してください。追加するすべての新しいステートメントと、変更を要する特定のステートメントを入力するだけで済みます。また、再入力しようとしている変更するステートメントでも、ステートメントの変更する部分を追加するだけで済みます。この例では、再入力して変更したのは、SAPROPS.CFG からの次の 1 つのステートメントです。

```
PROPERTY "Author Name” { EDIT TEXT LENGTH 25 }
```

USRPROPS.TXT ファイルでは、このプロパティーのラベルの変更のみが必要だったため、単に次のように入力しました。

```
PROPERTY "Author Name"{ LABEL "Client Division" }
```

プロパティーの長さと、それが (数値やブール値でなく) テキストであるという事実は変更されずに残っています。ダイアログ内でコントロールの左側にあるラベルだけが変更されています。

さらに 1 つの変更と警告

別の変更を試してみましょう。USRPROPS.TXT コードの下に、太字のテキストを追加します。

```
DEFINITION "Change Request"
{
LAYOUT { COLS 2 TAB ALIGN LABEL }
PROPERTY "Impact Statement”  { EDIT text LENGTH 100 }
PROPERTY "Author Name" { LABEL "Client Division" }
PROPERTY "Supervising Manager" { EDIT text LENGTH 45 }
PROPERTY "On time" { Edit Boolean Length 1 DEFAULT "T" }
}
```

この変更の説明を以下に示します。

USRPROPS.TXT 項目	効果
PROPERTY "Impact Statement" { EDIT TEXT LENGTH **100** }	既存のプロパティーを変更し、「*影響評価*」のスペースを 1000 文字から 100 文字に減らしてみます。

この USRPROPS.TXT をエンサイクロペディアにインポートして戻し、エンサイクロペディアを再オープンすると、次のような Rational System Architect からの警告メッセージを受け取ります。

> 警告：ユーザーによるプロパティー・ファイルへの追加の行番号 70 と 72 の間。(Warning: In user's property file addenda between line number 70 and line number 72.) プロパティーの長さを不正に短くしようとしています。元の長さのままになります。

Rational System Architect ではフィールドの長さを短くすることは許されません。フィールドの長さは長くすることだけができます。その理由は、既にいずれかのテキスト・フィールドに情報が入力してある場合、あとからそのフィールドの長さを短くし、それによってこのプロパティーについてエンサイクロペディアが保持できる情報の量を減らすと、情報が失われる可能性があるからです。

Rational System Architect は警告を発行し、障害のあるコードを無視してエンサイクロペディアを開きます。前に述べたように、それがエラー・メッセージだった場合には、エンサイクロペディアはユーザーが USRPROPS.TXT を修正するまで開かれません。

逆に、「影響評価」フィールドの長さを増やそうとしていたなら、Rational System Architect は変更を問題なく受け入れていたはずです。

PROPERTY "Impact Statement”{ EDIT text LENGTH **1200** }

値の LIST の定義

ダイアログ内でドロップダウン・リストまたはチェック・ボックス・リストとしてユーザーに提供される項目のリストを指定できます。リストの値は、「リスト」定義の中で指定する必要があります。その後、「リスト」定義は、それが使用される「ダイアグラム」、「シンボル」、または「定義」の中で参照されます。リストは、USRPROPS.TXT の中で、そのリストを参照するすべての「ダイアグラム」、「シンボル」、または「定義」項目の前に置く必要があります。

USRPROPS.TXT ファイルの管理を容易にするには、すべてのリスト定義をファイルの先頭に置きます。ただし、名前変更コマンドがある場合は、その後に置きます。

LIST 定義の構文

リスト定義は、キーワード **LIST** で始め、その直後にリスト名を表すストリング (引数) を続けます。空白文字を含む名前は、二重引用符で囲む必要があります。 **LIST** 定義は左右の中括弧 **{ }** で囲むか、それに代わる **BEGIN...END** 構造で囲みます。囲みの中では、リストの値を指定し、それぞれの値をコマンド・キーワード **VALUE** で呼び出します。値に 1 つ以上のスペースが埋め込まれている場合は、値を二重引用符で囲む必要があります。

```
LIST  list_name
{
VALUE value_name_1
VALUE value_name_2

…
}
```

例:

```
List "Method Stereotypes"
{
VALUE Get
VALUE Let
VALUE Set
VALUE “Stereotype with embedded spaces”
}

DEFINITION “Method”  {..PROPERTY "Stereotype"
{ EDIT Text LIST "Method Stereotypes" Default ""
LENGTH 30  }  …}
```

インデントと改行は単に読みやすくするために使用され、USRPROPS.TXT プロセッサーに対しては、ストリング同士を区切る空白区切り記号として機能する以外、何も意味がありません。上記の例は、以下のように書くこともできます。

```
LIST "Method Stereotypes" { VALUE get VALUE let
VALUE set VALUE “Stereotype with embedded
spaces” }
```

この形式でも Rational System Architect に完全に受け入れられますが、USRPROPS.TXT ファイルの保守を難しくするので、避けるようにしてください。

チェック・ボックスとドロップダウン・リスト

Rational System Architect は、LIST ステートメント内の値の数が 4 以下の場合には、リストを自動的にチェック・ボックス選択項目のリストとして表示します。値の数が 5 以上の場合には、リストは自動的にドロップダウン・リスト・ボックスとして表示されます。 ユーザーは、ドロップダウン・リスト・ボックスに独自の値を入力できます。リストされる値が 4 以下のときにドロップダウン・リスト・ボックスを表示したい場合には、LISTONLYCOMBO キーワードを使用してください。

独自の値の入力

リストがドロップダウン・リストとして提供される場合、ユーザーはリストから値の 1 つを選択するか、独自の値を入力できます (LISTONLY コマンドまたは LISTONLYCOMBO コマンドを使用した場合 (第 3 章の LISTONLY コマンドまたは LISTONLYCOMBO コマンドを参照) を除きます)。

既存のダイアグラム・タイプ、シンボル・タイプ、または定義タイプの名前変更

DIAGRAM、SYMBOL、および DEFINITION、および RELATIONSHIP の各ステートメントは、Rational System Architect で認識されているオブジェクトを参照する必要があります。

ただし、提供された SAPROPS.CFG ファイル内にあるいずれかの名前が適切でない場合は、個々のプロジェクトまたは企業の標準の要件に合わせて、名前を変更できます。RENAME ステートメントは、USRPROPS.TXT の先頭 (他のすべてのコマンドやステートメントより前の位置) に指定する必要があります。**RENAME** コマンドの一般的な構文は次のとおりです。

```
RENAME class_name from_type_name TO to_type_name
```

次の 3 つのステートメントでは、それぞれダイアグラム、シンボル、および定義の名前を変更します。

```
RENAME DIAGRAM "Data Flow Gane & Sarson"
TO "Data Flow Chris & Trish"

RENAME SYMBOL "Data Transform" in
"Data Flow Ward & Mellor"
TO "Process A" ACCELERATOR "r"

RENAME DEFINITION "Process"
TO "Process A"
```

図 2- 5。「タイプ」プルダウン・メニューは「*Data Flow Gane & Sarson*」を表示せずに「*Data Flow Chris & Trish*

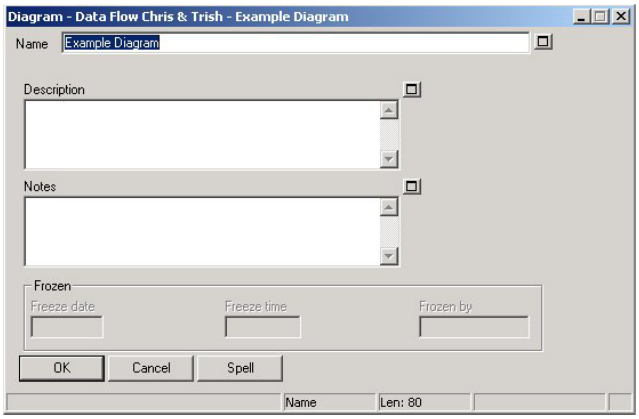


図 2- 6。ダイアグラム・プロパティー変更ダイアログもタイトル「*DFD Gane & Sarson*」でなくタイトル「*DFD Chris & Trish*」内に表示される

RENAME SYMBOL コマンドは、Ward & Mellor DFD を使用して作業する設計者が「*Transform*」よりも 「*Process*」という名前を好む場合に使用できます。DFD Ward & Mellor で、「*制御変換*」をクリックしてください。その後、ダイアグラム内のシンボルをダブルクリックして、「**ダイアグラム <Type> <Name>**」ダイアログを表示します。シンボルのタイプは、「*制御変換*」です。

シンボルを名前変更するには、USRPROPS.TXT 内に次のコマンドを入力する必要があります。

```
RENAME class_name from_type_name IN
from_diagram_name TO to_type_name
```

以下に例を示します。

```
RENAME SYMBOL "Control Transform" IN "DFD Ward &
Mellor" TO "Process" ACCELERATOR "r"
```

図 2-7。RENAME 前のシンボル・プロパティー・ダイアログ

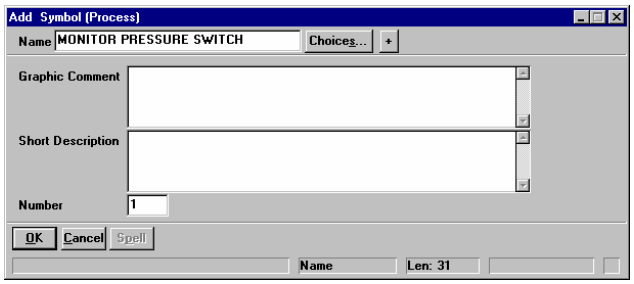
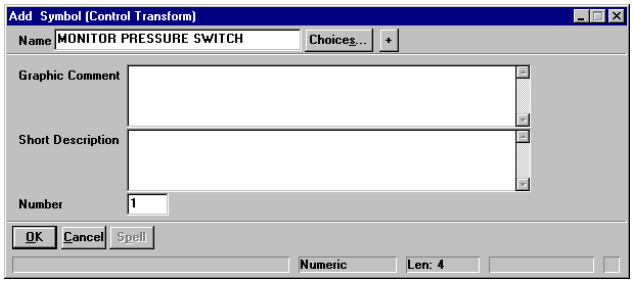


図 2-8。RENAME 後のシンボル・プロパティー・ダイアログ



「**編集**」メニューをクリックし、「**編集 <シンボル・タイプ>**」を、「*制御変換*」が選択されている間に再び選択します。「**定義変更**」ダイアログのタイトルに注意してください。定義は「*プロセス*」であり、「*制御変換*」ではありません。つまり、シンボル「*制御変換*」の定義は、定義「*プロセス*」にマップされます。DFD Gane & Sarson でなく DFD Ward & Mellor を使用しており、定義名をシンボル名に一致させたいとします。

定義の **RENAME** コマンドの構文は、以下のとおりです。

RENAME class_name from_type_name TO to_type_name

RENAME DEFINITION "Process" TO "Control Transform"

図 2-9。 RENAME 前のシンボル定義ダイアログ

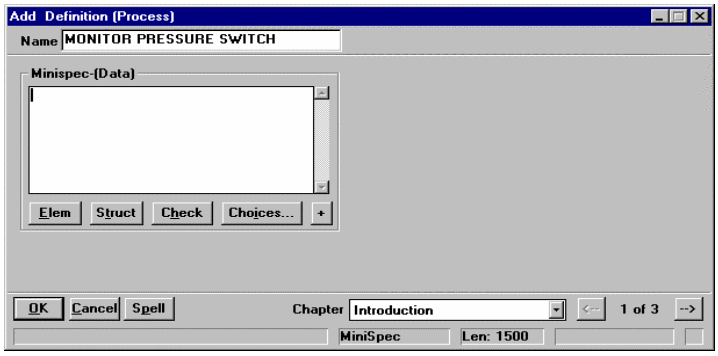
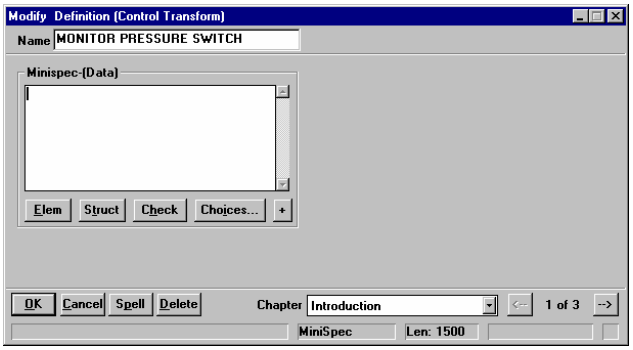


図 2-10。 RENAME 後のシンボル定義ダイアログ



一方、DFD Gane & Sarson (この「*プロセス*」シンボルは定義「*プロセス*」にマップされます) と DFD Ward & Mellor (この「*制御変換*」シンボルは定義「*プロセス*」にマップされます) の両方を使用している場合、すべてのプロセスの定義ではなく、「*制御変換*」の定義だけを名前変更したいことがあります。以下の USRPROPS.TXT 内の項目はそのような名前変更を行います。

RENAME DEFINITION "User 2"[1] to "Control Transform"

SYMBOL "Control Transform" in <diagram name>
{ DEFINED BY "Control Transform" }

「定義」"制御変換 (Control Transform)" は、一連のプロパティーを含んでいない場合、プロパティー「*説明*」だけを持ちます。ここで使用している例では、以下の定義ブロック (「*プロセス*」のそれと同じもの) が USRPROPS.TXT に追加されました。もちろん、適切と思われる任意のプロパティーを使用できます。既存の定義のプロパティーをコピーする必要はありません。

[1] User 1 から始めて、150 個の "User n" 定義を使用できます。

```
DEFINITION "Control Transform"
{
PROPERTY "Description"
  { EDIT Minispec LENGTH 750 }
PROPERTY "Complexity"
  { EDIT numeric LENGTH 10 }
PROPERTY "Memory Allocation (KB)"
   { EDIT numeric LENGTH 7 }
PROPERTY "Priority"
  { EDIT numeric LENGTH 3 MINIMUM 0 MAXIMUM 999 }
PROPERTY "Process Class"
  { EDIT text LISTONLY LIST "Process Class" LENGTH
20 }
PROPERTY "Processing Time Allocation"
  { EDIT numeric LENGTH 3 MINIMUM 0 MAXIMUM 100 }
PROPERTY "Purpose"
  { EDIT text LENGTH 4095 }
PROPERTY "Transaction Rate"
  { EDIT numeric LENGTH 10 MINIMUM 1 MAXIMUM 10 }
}
```

RENAME とレポート作成

RENAME コマンドは、レポートを作成する方法にも影響します。古い名前を使用しているすべての場所で、代わりに新しい名前を使用する必要があります。GUI レポート作成システムでは、ダイアグラム、シンボル、または定義プロパティーの名前を SAPROPS 内で変更した後に、それらの名前を再選択する必要があります。

変更前:

```
REPORT "List of Processes"
{
TABULAR 1 {
SELECT Name, "Update Date", Description
WHERE Class = Definition
WHERE Type = "Process"
ORDERBY Name
}
}
```

「テキスト・エディター」(「**レポート**」ダイアログ、**EDIT** コマンド) を使用してレポートを表示した場合は、以下のような表示になります。

変更後:

```
REPORT "List Of Transforms"
{
TABULAR 1 {
SELECT Name, "Update Date", Description
WHERE Class = Definition
WHERE Type = "Control Transform"
ORDERBY Name
}
}
```

新規ダイアグラム・タイプ、シンボル・タイプ、または定義タイプの作成

Rational System Architect エンサイクロペディア内に、新しいダイアグラム・タイプ、シンボル・タイプ、または定義タイプを作成できます。これを行うには、RENAME コマンドを使用して、作成の目的のために提供された既存のダイアグラム・タイプ、シンボル・タイプ、または定義タイプを名前変更します。この場合も、RENAME コマンドを USRPROPS.TXT ファイルの先頭で、冒頭の REM (覚書またはコメント) ステートメントの直下に配置してください。

新しいダイアグラムの作成

Rational System Architect エンサイクロペディアには、最大 50 の新規ダイアグラム・タイプを作成することができます。新しいダイアグラム・タイプを追加するには、USRPROPS.TXT の中で、用意された 50 個の汎用ダイアグラム・タイプ (User 1 から User 50) のうちの 1 つを名前変更します。構文は以下のとおりです。

```
RENAME DIAGRAM “User 1” TO My_Diagram
```

作成する新しいダイアグラム・タイプ、シンボル・タイプ、または定義タイプにスペースを埋め込みたい場合は、名前を引用符で囲む必要があることに注意してください。以下に例を示します。

```
RENAME DIAGRAM “User 1” TO “My Diagram”
```

新しいダイアグラム・タイプを作成後、その上にどのタイプのシンボルを描画できるかを指定する必要があります。新しいシンボル・タイプを作成することも、既に他のダイアグラム上に存在するシンボルを新しいダイアグラム・タイプに割り当てることもできます。これについては、次のセクションの『ダイアグラム・タイプへのシンボル・タイプの割り当て』で説明します。

デフォルトでは、ユーザー・ダイアグラムは (シンボルの) ネットワークですが、ユーザー・ダイアグラムのタイプを「**階層**」に指定することもできます。Rational System Architect の階層ダイアグラムには、特殊な描画規則が課されており、これによってユーザーは階層内のシンボルを接続

したり、ライン・シンボルを自動的に描画させたりすることができます。それ以外にも、関連する階層機能 (階層番号付けなど) がサポートされています。ダイアグラムのタイプを「階層」に指定するには、HIERARCHICAL キーワードを以下のように使用します。

DIAGRAM "Zoo" {**HIERARCHICAL**}

新規シンボルの作成

Rational System Architect エンサイクロペディアには、最大 150 の新規シンボル・タイプを作成することができます。新規シンボル・タイプを追加するには、用意された 150 個の汎用シンボル・タイプ (User 1 から User 150) の 1 つを名前変更します。構文は以下のとおりです。

RENAME SYMBOL "User 3" to "whatever"

新しいライン・シンボルの指定

ライン・シンボルとは、2 つのシンボルの間に描画できる線のことで、リレーションシップ・ライン、継承ライン、関連、流れ線などがあります。エンサイクロペディアの中に新しいライン・シンボル・タイプを作成できます。その場合、別のダイアグラム内にある既存のライン・シンボル・タイプのような外観と動作を持つことを指定する必要があります。通常の (「ノード」) シンボルの場合と同じ RENAME SYMBOL コマンドを使用しますが、あとで USRPROPS.TXT の中で DEPICT LIKE コマンドを使用して、ライン・シンボルの描画方法も指定する必要があります。

```
RENAME SYMBOL "User 4" to "My Line Symbol"

SYMBOL "My Line Symbol"
{ DEPICT LIKE "Dependency" IN "UML Class"
ASSIGN To "Wireless Network" }
```

新しい定義の作成

Rational System Architect エンサイクロペディアには、最大 150 個の新しい定義タイプを作成できます。新規定義タイプを追加するには、用意された 150 個の汎用定義タイプ (User 1 から User 150) の 1 つを名前変更します。構文は以下のとおりです。

RENAME DEFINITION "User 3" to "whatever"

シンボルは通常、定義タイプを表します。これについては、この後のセクション『シンボル・タイプへの定義タイプの割り当て』を参照してください。

ダイアグラム・タイプへのシンボル・タイプの割り当て

新規または既存のダイアグラム・タイプに、新規のシンボル・タイプまたは既存のシンボル・タイプ (他のダイアグラムに既に存在するシンボル) を割り当てることができます。シンボル・タイプは、以下の構文を使用してダイアグラム・タイプに追加することができます。

ASSIGN <symbol-type-name> [IN <diagram-type-name1>] **TO** <diagram-type-name2>

シンボル・タイプは、以下のように **ASSIGN .. TO** のキーワードの組み合わせを使用して、SYMBOL 指定の中でダイアグラム・タイプに追加することもできます。

SYMBOL <symbol-type-name> [IN <diagram-type-name1>] **{ASSIGN TO** <diagram-type-name1>**}**

例

例えば、下記の USRPROPS.TXT は「無線ネットワーク (Wireless Network)」図という新しいダイアグラム・タイプを作成します。このダイアグラムは、このダイアグラム上に描画される 3 つの新しいシンボル・タイプ (「衛星」、「コンピューター」、および「サーバー」) と、このダイアグラム上に描画される 1 つの既存のシンボル・タイプを提供します。その既存のシンボル・タイプは、「状態遷移 Ward & Mellor」図からの状態シンボルです (「UML 状態」図または「IDEF3 状態 (IDEF3 State)」図などからの「状態」シンボルと対比されます)。

```
RENAME DIAGRAM "User 1" To "Wireless Network"

RENAME SYMBOL "User 1" TO "Satellite"
RENAME SYMBOL "User 2" TO "Computer"
RENAME SYMBOL "User 3" TO "Server"


ASSIGN "State" IN "State Transition Ward & Mellor" TO
“Wireless Network”

SYMBOL "Satellite" {ASSIGN TO "Wireless Network"}
SYMBOL "Computer" {ASSIGN TO "Wireless Network"}
SYMBOL "Server" {ASSIGN TO "Wireless Network"}
```

注: 『*ダイアグラム・タイプへのシンボル・タイプの割り当て*』のセクションも参照してください。

ダイアグラム・タイプへのライン・シンボル・タイプの割り当て

この場合も、このセクションで前に述べたように、ライン・シンボルとは、2 つのシンボルの間に描画できる線のことで、リレーションシップ・ライン、継承ライン、関連、流れ線などがあります。エンサイクロペディアの中に新しいライン・シンボル・タイプを作成できます。その場合、別のダイアグラム内にある既存のライン・シンボル・タイプのような外観と動作を持つことを指定する必要があります。通常の (「ノード」) シンボルの場合と同じ RENAME SYMBOL コマンドを使用しますが、あとで USRPROPS.TXT の中で DEPICT LIKE コマンドを使用して、ライン・シンボルの描画方法も指定する必要があります。

ユーザー定義シンボル (User 1 から User 150 まで) は、通常の (「ノード」) シンボルとライン・シンボルの両方に用意されているので、同じユーザー番号を 2 つの異なるシンボルに使用しないように注意してください。

例

下記の例では、太字で示す新しいライン描画シンボルを USRPROPS.TXT に追加します。

```
RENAME DIAGRAM "User 1" To "Wireless Network"

RENAME SYMBOL "User 1" TO "Satellite"
RENAME SYMBOL "User 2" TO "Computer"
RENAME SYMBOL "User 3" TO "Server"
RENAME SYMBOL "User 4” To “Relates To”

ASSIGN "State" IN "State Transition Ward & Mellor" TO
“Wireless Network”
SYMBOL "Satellite" {ASSIGN TO "Wireless Network"}
SYMBOL "Computer" {ASSIGN TO "Wireless Network"}
SYMBOL "Server" {ASSIGN TO "Wireless Network"}

SYMBOL "Relates To"
{ DEPICT LIKE "Dependency" IN "UML Class"
ASSIGN To "Wireless Network" }
```

注: 『*ダイアグラム・タイプへのシンボル・タイプの割り当て*』のセクションも参照してください。

ダイアグラム・タイプへのシンボル・タイプの割り当てに関する制限事項

ダイアグラム・タイプへのシンボル・タイプの割り当てには、以下の制限があります。

以下のダイアグラム・タイプについては、割り当てを行えません。

- DB2 物理
- エンティティー・リレーション
- 論理データ・モデル
- 論理ビュー
- 物理データ・モデル

Rational System Architect に特別なコードがあるため、以下のシンボルを割り当てることはできません。

- 図「エンティティー・リレーション」内の「結合型エンティティー」
- 図「エンティティー・リレーション」内の「エンティティー」
- 図「エンティティー・リレーション」内の「識別関係」
- 図「エンティティー・リレーション」内の「矛盾した関係」
- 図「エンティティー・リレーション」内の「非識別関係」
- 図「エンティティー・リレーション」内の「非固有関係」
- 図「エンティティー・リレーション」内の「スーパー/サブ関係」
- 図「エンティティー・リレーション」内の「脆弱エンティティー」
- 図「OMT オブジェクト・モデル」内の「関連」
- 図「OMT オブジェクト・モデル」内の「クラス」
- 図「物理データ・モデル」内の「識別制約」
- 図「物理データ・モデル」内の「非識別制約」
- 図「物理データ・モデル」内の「テーブル」
- 図「UML クラス」内の「クラス」
- 図「UML クラス」内の「インターフェース」

- 図「UML ユースケース」内の「アクター」
- 図「UML ユースケース」内の「境界」
- 図「UML ユースケース」内の「ケースワーカー」
- 図「UML ユースケース」内の「制御」
- 図「UML ユースケース」内の「エンティティー」
- 図「UML ユースケース」内の「ワーカー」
- また、以下のシンボルの定義にはモデルによるキーが付けられているため、これらのシンボルを他のダイアグラムに割り当てることはできません。
- 図「エンティティー・リレーション」内の「アクセス・パス」
- 図「エンティティー・リレーション」内の「リレーション」
- 図「エンティティー・リレーション」内の「関係ダイヤモンド」
- 図「データの概念モデル」内の「Individu」
- 図「データの概念モデル」内の「Relation Ligne」
- 図「SSADM データ構造」内の「キーであるエントリー・ポイント」
- 図「SSADM データ構造」内の「キーではないエントリー・ポイント」
- 図「SSADM データ構造」内の「リレーション」
- さらに、以下の BPMN シンボルを別のダイアグラム・タイプに割り当てることはできません。
- 図「ビジネス・プロセス」内の「プール」
- 図「ビジネス・プロセス」内の「レーン」

注: 特殊コードを持つと同時にモデルによるキーが付いているシンボルもあり、それらは最初のシンボル・リスト内にのみ示されます。

通常、多くのシンボルは複数のダイアグラム・タイプに存在できます。 上のリストにあるシンボルについては、1 つのダイアグラム・タイプのみが表示されます。

シンボル・タイプへの定義タイプの割り当て

USRPROPS.TXT でエンサイクロペディアに新しいシンボルを追加する場合、このキーワードを使用して、関連付ける定義タイプを指定する必要があります。USRPROPS.TXT に指定した新しいシンボルにこの節がない場合、 Rational System Architect はエンサイクロペディアを開くときに構文解析の警告を出し、 そのシンボルの定義を既定値の NULL にします。この定義は、「説明」プロパティーだけで構成されています。

```
SYMBOL "My Symbol"
{
DEFINED BY " My Definition"
ASSIGN TO "My Diagram"
}
```

例

下記の例では、シンボル・タイプ「Satellite」が定義タイプ「Satellite」によって定義されるように指定されています (たまたま同じ名前を共有するだけでは不十分です)。

```
Rename Diagram "User 1" TO "Wireless Network"
Rename Symbol "User 1" TO "Satellite"
Rename Definition "User 1" TO "Satellite"

SYMBOL "Satellite"
{ DEFINED BY “Satellite” ASSIGN To "Wireless Network" }
```

ビットマップまたはメタファイルでシンボルを描写

シンボルは、ユーザーが提供するビットマップ (.bmp) または Windows メタファイル (.wmf) によって描画できます。以下のように、シンボルの宣言に depictions 節を追加することで、 シンボルをダイアグラムのワークスペース上に描写する方法と、 ツールボックスおよび「描画」メニューに描写する方法も指定できます。

SYMBOL <symbol-type-name>

{ ...
DEPICTIONS { DIAGRAM <depiction-file> }
DEPICTIONS { MENU <depiction-file> }
...}

DIAGRAM コマンドは、描写ファイルをダイアグラム・ワークスペースに***描画***するよう指定します。DIAGRAM コマンドには ***Windows メタファイル (.WMF)*** を使用してください。これはベクトル・イメージであり、そのサイズを増減するためにハンドルバー上でドラッグすると、正しく拡大縮小されるからです。.BMP を DIAGRAM コマンドに使用することもできますが、その場合は正しく拡大縮小されません。

WMF はベクトル・ファイルです。つまり、 イメージを画面上に表示する方法に関する数式が保管されています。この形式の主な利点は、 画質を低下させずにイメージを拡大縮小できることです。 WMF ファイルでは、ズームインまたはズームアウトしても、変形したりギザギザになったりすることはありません。

MENU コマンドは、***ツールバー***、***メニュー***、およびその他の領域に表示される描写ファイルを指定します。これは、描画するシンボルを選択するためにクリックするグラフィックです。ツールバーの場合は、**ビットマップ**・イメージを使用するのが最良の方法です。ビットマップ・イメージは拡大縮小する必要がないからです。通常、ツールバー内に表示する各シンボルには、***16x16 ピクセル*** のビットマップを作成するのが最適です。

BMP はラスター・ファイルです。つまり、 イメージ上の各ピクセルについての情報が保管されています。ビットマップでは画質の良い、写真品質のイメージをレンダリングできますが、ズームインすると変形し、ズームアウトするとギザギザになります。

<depiction-file> は、ビットマップまたはメタファイルの名前とフルパスです。エンサイクロペディアのパスの外部にあるディレクトリーを指定してもかまいませんが、ビットマップとメタファイルを、エンサイクロペディアのデータベースの Files テーブルに直接追加することをお勧めします。

独自の描写ファイルをエンサイクロペディアに追加するには、以下のステップを実行します。

1. **USRPROPS.TXT に必要な変更を加えます。** そのようなコードの例を以下に示します。

   ```
   RENAME DIAGRAM “User 1” TO “Wireless Communications”
   RENAME SYMBOL “User 1” TO “Satellite”
   SYMBOL "Satellite"
   {ASSIGN To "Wireless Network"
   DEPICTIONS { DIAGRAM satellite.wmf }
   DEPICTIONS { MENU satellite_toolbar.bmp }
   }
   ```

2. **使用する .BMP および .WMF ファイルをエンサイクロペディアの FILES テーブルにインポートします。** Rational System Architect のエンサイクロペディア・ファイル・マネージャー (「ツール」、「エンサイクロペディア・ファイル・マネージャー」)、または SAEM (「スタート」、「プログラム」、「IBM Rational」、「IBM Rational Lifecycle Solutions Tools」、「IBM Rational System Architect 11.3.1」、「SAEM」(使用方法については SAEM のヘルプを参照))、または Microsoft の Enterprise Manager を使用して、ユーザー定義グラフィック・ファイルをエンサイクロペディア・データベースの FILES テーブルにインポートできます。エンサイクロペディア・ファイル・マネージャーでは、一度に 1 つのファイルだけをインポートできます。複数のグラフィック・ファイルがある場合は、SAEM を使用して FILES テーブルにファイルをインポートすることをお勧めします。

インポートするファイルの名前は、Usrprops.txt コードと整合する必要があります。上記の例ではパスをまったく指定せず、単に satellite.wmf と satellite.bmp をリストすることによって、相対パスを使用しました。これは、描写ファイルをデータベースの Files テーブルに直接インポートすることを意味します。

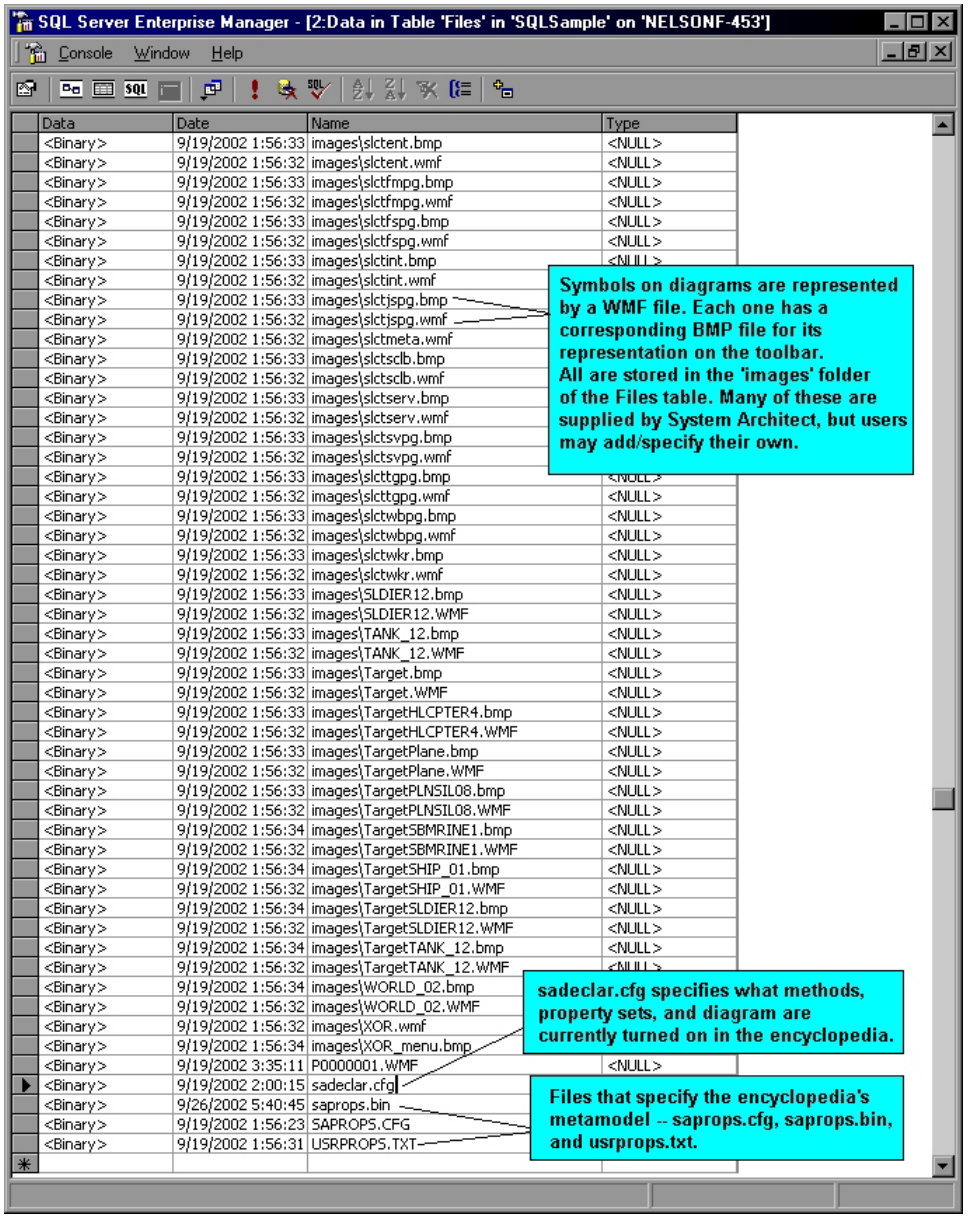
推奨事項: Rational System Architect での既定の規則に従い、描写ファイルの名前に「images/」を付加して、それぞれの描写ファイルが FILES テーブルの「images」サブディレクトリーに入っていることをシミュレートするようお勧めします。SAEM を使用して複数のファイルを一度にインポートする場合は、それらのファイルが、コンピューター上の任意の場所にある「images」という名前のディレクトリーに入っていることを確認してください。SAEM は images という名前のディレクトリーからインポートされるすべてのファイルの名前に、自動的に「images/」を (各グラフィックのファイル名の前に) 付加します。同じように、USRPROPS.TXT 内の各描写ファイルの名前の前にも「images\」を指定してください。これにより上記の例は以下のようになります。

```
RENAME DIAGRAM “User 1” TO “Wireless Communications”
RENAME SYMBOL “User 1” TO “Satellite”
SYMBOL "Satellite"
{ASSIGN To "Wireless Network"
DEPICTIONS { DIAGRAM images\satellite.wmf }
DEPICTIONS { MENU images\satellite_toolbar.bmp }
}
```

この方法を使用する利点は 2 つあります。第一に、ユーザー指定イメージに対して、ある種の名前独立性および論理グループ化方法が提供されます。第 2 に、新しいエンサイクロペディアの作成時にイメージが扱われる方法と整合性があります。つまり、Rational System Architect はすべてのグラフィックを ..\System Architect\images ディレクトリーに取り込み、新しいエンサイクロペディアの FILES テーブル内に置き、「images\」が付加された名前を付けます。

エンサイクロペディアのデータベースの Files テーブル内部の様子と、描写ファイルの「images\」接頭部によるイメージの論理グループ化については、下の図を参照してください。

図 2-11。
エンサイクロペディア・データベースの「Files」テーブル。



Data	Date	Name	Type
<Binary>	9/19/2002 1:56:33	images\slctent.bmp	<NULL>
<Binary>	9/19/2002 1:56:32	images\slctent.wmf	<NULL>
<Binary>	9/19/2002 1:56:33	images\slctfmpg.bmp	<NULL>
<Binary>	9/19/2002 1:56:32	images\slctfmpg.wmf	<NULL>
<Binary>	9/19/2002 1:56:33	images\slctfspg.bmp	<NULL>
<Binary>	9/19/2002 1:56:32	images\slctfspg.wmf	<NULL>
<Binary>	9/19/2002 1:56:33	images\slctint.bmp	<NULL>
<Binary>	9/19/2002 1:56:32	images\slctint.wmf	<NULL>
<Binary>	9/19/2002 1:56:33	images\slctjspg.bmp	<NULL>
<Binary>	9/19/2002 1:56:32	images\slctjspg.wmf	<NULL>
<Binary>	9/19/2002 1:56:32	images\slctmeta.wmf	<NULL>
<Binary>	9/19/2002 1:56:33	images\slctsclb.bmp	<NULL>
<Binary>	9/19/2002 1:56:32	images\slctsclb.wmf	<NULL>
<Binary>	9/19/2002 1:56:33	images\slctserv.bmp	<NULL>
<Binary>	9/19/2002 1:56:32	images\slctserv.wmf	<NULL>
<Binary>	9/19/2002 1:56:33	images\slctsvpg.bmp	<NULL>
<Binary>	9/19/2002 1:56:32	images\slctsvpg.wmf	<NULL>
<Binary>	9/19/2002 1:56:33	images\slcttgpg.bmp	<NULL>
<Binary>	9/19/2002 1:56:32	images\slcttgpg.wmf	<NULL>
<Binary>	9/19/2002 1:56:33	images\slctwbpg.bmp	<NULL>
<Binary>	9/19/2002 1:56:32	images\slctwbpg.wmf	<NULL>
<Binary>	9/19/2002 1:56:33	images\slctwkr.bmp	<NULL>
<Binary>	9/19/2002 1:56:32	images\slctwkr.wmf	<NULL>
<Binary>	9/19/2002 1:56:33	images\SLDIER12.bmp	<NULL>
<Binary>	9/19/2002 1:56:32	images\SLDIER12.WMF	<NULL>
<Binary>	9/19/2002 1:56:33	images\TANK_12.bmp	<NULL>
<Binary>	9/19/2002 1:56:32	images\TANK_12.WMF	<NULL>
<Binary>	9/19/2002 1:56:33	images\Target.bmp	<NULL>
<Binary>	9/19/2002 1:56:32	images\Target.WMF	<NULL>
<Binary>	9/19/2002 1:56:33	images\TargetHLCPTER4.bmp	<NULL>
<Binary>	9/19/2002 1:56:32	images\TargetHLCPTER4.WMF	<NULL>
<Binary>	9/19/2002 1:56:33	images\TargetPlane.bmp	<NULL>
<Binary>	9/19/2002 1:56:32	images\TargetPlane.WMF	<NULL>
<Binary>	9/19/2002 1:56:33	images\TargetPLNSIL08.bmp	<NULL>
<Binary>	9/19/2002 1:56:32	images\TargetPLNSIL08.WMF	<NULL>
<Binary>	9/19/2002 1:56:34	images\TargetSBMRINE1.bmp	<NULL>
<Binary>	9/19/2002 1:56:32	images\TargetSBMRINE1.WMF	<NULL>
<Binary>	9/19/2002 1:56:34	images\TargetSHIP_01.bmp	<NULL>
<Binary>	9/19/2002 1:56:32	images\TargetSHIP_01.WMF	<NULL>
<Binary>	9/19/2002 1:56:34	images\TargetSLDIER12.bmp	<NULL>
<Binary>	9/19/2002 1:56:32	images\TargetSLDIER12.WMF	<NULL>
<Binary>	9/19/2002 1:56:34	images\TargetTANK_12.bmp	<NULL>
<Binary>	9/19/2002 1:56:32	images\TargetTANK_12.WMF	<NULL>
<Binary>	9/19/2002 1:56:34	images\WORLD_02.bmp	<NULL>
<Binary>	9/19/2002 1:56:32	images\WORLD_02.WMF	<NULL>
<Binary>	9/19/2002 1:56:32	images\XOR.wmf	<NULL>
<Binary>	9/19/2002 1:56:34	images\XOR_menu.bmp	<NULL>
<Binary>	9/19/2002 3:35:11	P0000001.WMF	<NULL>
<Binary>	9/19/2002 2:00:15	sadeclar.cfg	<NULL>
<Binary>	9/26/2002 5:40:45	saprops.bin	<NULL>
<Binary>	9/19/2002 1:56:23	SAPROPS.CFG	<NULL>
<Binary>	9/19/2002 1:56:31	USRPROPS.TXT	<NULL>

3. **「エンサイクロペディア」を再オープンして、変更を有効にします。**

新しいエンサイクロペディア用の描写ファイルの指定

新しいエンサイクロペディアを作成する場合、操作を選択できます。つまり、最初にエンサイクロペディアを作成してから、1 つ以上のユーザー提供のグラフィック・ファイルを、前述のように SAEM、エンサイクロペディア・マネージャー、または SQL Server の Enterprise Manager によってそのエンサイクロペディアにインポートするか、エンサイクロペディアを作成する前に、ユーザー提供のイメージを Rational System Architect のメイン・***イメージ***・ディレクトリー (メイン・ソフトウェア・ディレクトリーの下の <C>:\Program Files\IBM\Rational\11.3.1\System Architect Suite\System Architect\images) に配置することができます。Rational System Architect は、すべてのグラフィックをそのメイン・イメージ・ディレクトリーに取り込み、作成されるすべての新規エンサイクロペディアの Files テーブル内に置きます。

自分または他のチーム・メンバーが作成するすべての新規エンサイクロペディアに同じユーザー指定グラフィック・ファイルを入れたい場合は、以下のステップを実行します。

1. **使用する .BMP および .WMF ファイルをコピーして Rational System Architect の「Images」サブディレクトリーに貼り付けます**。新しいエンサイクロペディアを作成する前に、使用する .BMP および .WMF ファイルを Rational System Architect メインプログラム・ディレクトリー内の Images ディレクトリーに入れます。将来の任意の時点で新しいエンサイクロペディアを作成するすべてのチーム・メンバーは、これを実行する必要があります。これらのファイルは、あとで作成するエンサイクロペディアの FILES テーブル内に自動的に配置されます。Rational System Architect は、それぞれのファイル名に「images\」を付加します。このため、Fred.bmp という図は、新しいエンサイクロペディアの FILES テーブル内に images\Fred.bmp という名前で作成されます。これは、エンサイクロペディアを作成し、そのあとにユーザー提供のグラフィック・ファイルをエン

サイクロペディアにインポートすることをより簡単に行うための方法です。

2. **USRPROPS.TXT に必要な変更を加えます。**
   DEPICTIONS コマンド (および、オプションとして RETAIN STYLE コマンド) を使用します。必要なコード変更を行う方法については、Rational System Architect のヘルプに説明があります。そのようなコードの例を以下に示します。

```
Rename Symbol "User 3" To "Radar
SYMBOL "Radar"
{ASSIGN To "Wireless Network"
DEPICTIONS { DIAGRAM RETAIN STYLE "C:\Program
Files\IBM\pictures\radar.bmp" }
DEPICTIONS { MENU "C:\Program
Files\IBM\pictures\radartoolbar.bmp" }}
```

3. **「エンサイクロペディア」を再オープンして、変更を有効にします。**

プロパティー値に基づく ユーザー定義シンボル表示

シンボルの定義の *プロパティー値に基づいて*、シンボルの描画方法を指定できます。UML では、通常、このプロパティーはステレオタイプです。 ただし、この機能は、UML シンボルだけでなく、またステレオタイプ・プロパティーだけでもなく、すべてのシンボル・タイプに例外なく適用されます。

この機能を有効にするには、**DEPICTIONS** 節を直接、USRPROPS.TXT 内の **LIST** ステートメントの中に使用します。

```
LIST “New List Type”
{
VALUE “List Item One” DEPICTIONS {DIAGRAM
imageone.wmf MENU imageone_toolbar.bmp}
…
}
```

例

以下の例では、「Node Stereotypes」の新しいリストを指定します。これらのステレオタイプは、「UML 配置」図上の「ノード」シンボルに適用されるため、ユーザーは自らがエンサイクロペディア・データベースの FILES テーブルにインポートした独自のグラフィック・ファイルを使用して、ノード・シンボルを描画できます。

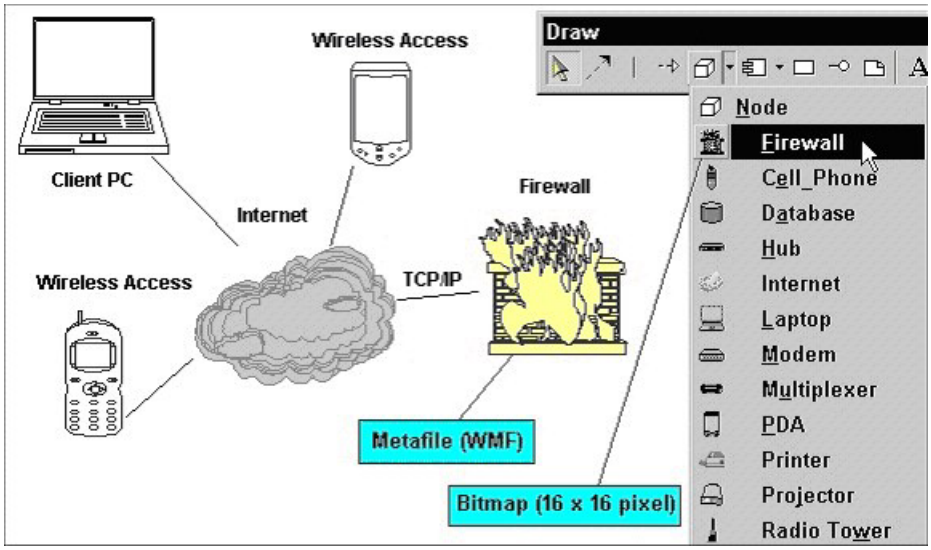
```
List "Node Stereotypes"
{
Value "Firewall" DEPICTIONS {DIAGRAM images\firewall.wmf
MENU images\firewall.bmp}
Value "Cell_Phone"  DEPICTIONS {DIAGRAM
images\cell_phone.wmf MENU images\cell_phone.bmp}
Value "Database"  DEPICTIONS {DIAGRAM images\data.wmf
MENU images\data.bmp}
Value "Hub"   DEPICTIONS {DIAGRAM images\hub.wmf
MENU images\hub.bmp}
Value "Modem"  DEPICTIONS {DIAGRAM images\modem.wmf
MENU images\modem.bmp}
Value "Multiplexer"  DEPICTIONS {DIAGRAM
images\multiplexer.wmf MENU images\multiplexer.bmp}
Value "PDA"  DEPICTIONS {DIAGRAM images\pda.wmf
MENU images\pda.bmp}
Value "Printer"   DEPICTIONS {DIAGRAM images\printer.wmf
MENU images\printer.bmp}
```

```
Value "Projector" DEPICTIONS {DIAGRAM
images\projector.wmf MENU images\projector.bmp}
Value "Radio Tower" DEPICTIONS { DIAGRAM
images\radio_tower.wmf MENU images\radio_tower.bmp}
Value "Router"  DEPICTIONS { DIAGRAM images\router.wmf
MENU images\router.bmp}
Value "Satellite" DEPICTIONS { DIAGRAM images\satellite.wmf
MENU images\satellite.bmp}
Value "Satellite Dish"  DEPICTIONS { DIAGRAM
images\dish.wmf MENU images\dish.bmp}
Value "Scanner"  DEPICTIONS { DIAGRAM
images\scanner.wmf MENU images\scanner.bmp}
Value "Server"  DEPICTIONS { DIAGRAM images\server.wmf
MENU images\server.bmp}
Value "Switch"  DEPICTIONS { DIAGRAM
images\kvm_switch.wmf  MENU images\kvm_switch.bmp}
Value "Tablet_PC"  DEPICTIONS { DIAGRAM
images\tablet_pc.wmf MENU images\tablet_pc.bmp}
Value "Terminal" DEPICTIONS { DIAGRAM
images\terminal.wmf MENU images\terminal.bmp}
}

SYMBOL "Node" in "Deployment"
{
PROPERTY "Stereotype" { INVISIBLE EDIT Text ListOnly
List "Node Stereotypes" DEFAULT "" LENGTH 32}
}

DEFINITION "Node"
{
PROPERTY "Stereotype"
{ EDIT Text LIST "Node Stereotypes" Default "" LENGTH 32  }
}
```

上記の USRPROPS.TXT の例で、「Node Stereotypes」の LIST がノードの SYMBOL と DEFINITION の両方で参照されていることに注意してください。SYMBOL 内では、プロパティーは INVISIBLE にされます。SYMBOL は、基礎となる定義に対して指定されたステレオタイプへの参照を保守します。

上記の USRPROPS.TXT コードは、「配置」図のツールバーを変更し、使用可能なステレオタイプ (および対応するそれぞれのビットマップ) のドロップダウン・リストを提供します。

図 2-12。ユーザー提供の描写ファイル。



ユーザーはステレオタイプを選択して、ダイアグラム上に描画することができます。ステレオタイプは対応する .WMF ファイルによってダイアグラム上に表されます。

シンボルを描画した後、各シンボルをマウスで右クリックして、以下の操作を選択できます。

- <Node> として表示。または、
- ステレオタイプで修飾 (この場合、メタファイルのサムネールがシンボルの名前の右側に配置される)。または、
- Stereotype に従って表示

描画されたら、「シンボル・スタイル」ツールバーを使用して (またはシンボルを選択し、「書式 (Format)」、「シンボル書式」、「シンボル・スタイル」を選択して) Rational System Architect 内の他のシンボルと同様に、メタファイルの色を指定できます。これには、ライン色の指定、塗りつぶし色の指定、フォント色の指定などが含まれます。

スタイルの保持

提供するメタファイルが、Rational System Architect で使用されたときに、元のグラフィカル・スタイルと色を保持するよう、指定できます。これを指定するには、RETAIN STYLE キーワードを使用します。以下に例を示します。

```
LIST "Node Stereotypes"
{
VALUE "Firewall" DEPICTIONS {DIAGRAM RETAIN
STYLE images\firewall.wmf MENU images\firewall.bmp}
..
}
```

ユーザーが提供するメタファイル firewall.wmf は、ダイアグラム上に描画される際に、Rational System Architect の外部で使用されていたのとまったく同じ色を使用して描画され、Rational System Architect のカラー・ツールを使用してこれらの色を変更することはできません。

描写されたシンボル上に表示できるプロパティー

System Architect では、DISPLAY キーワードを使用して、シンボルに表示するプロパティーを 37 種類まで指定できます。これは、ユーザー提供の描写ファイルによって描写されるシンボルでも同様です。

詳細については、この章で後に述べる『*シンボル上の値の表示の指定*』というタイトルの章を参照するか、第 3 章の DISPLAY キーワードを参照してください。

ダイアグラム、シンボル、および定義のプロパティーの指定

すべての **Rational System Architect** エンサイクロペディアに、*ダイアグラム*、*シンボル*、および*定義*という 3 つのクラスがあります。それぞれのクラスは、独自のプロパティー・セットによって定義できます。

次の表には、USRPROPS.TXT 内の「ダイアグラム」、「シンボル」、または「定義」ステートメントの外部にある、すべての必須項目とオプションの項目が含まれています。

表 2-3。「ダイアグラム」、「シンボル」または「定義」指定の必須項目とオプションの項目

入力	必須かオプションか	注釈
DIAGRAM { } or DIAGRAM BEGIN END または SYMBOL { } or SYMBOL BEGIN END または DEFINTION { } or DEFINITION BEGIN END	Mandatory	宣言を開始し、終了します。
CHAPTER chapter_name	オプション	後続のプロパティーを既存の章に組み込むか、新しい章を追加します。

GROUP group_name
{
PROPERTY prop_name
PROPERTY prop_name
}
	オプション	後続のすべてのプロパティーを、レイアウト制御用の 1 つのグループ内に配置します。

入力	必須か オプションか	注釈
LAYOUT { alignment_criteria PACK_TAB_criteria COLS no_of_columns JUSTIFY }	オプション	[Align Body | Align Label | Align Over] [Pack | Tab] COLS <number>

PROPERTY property_name
{
}
	Mandatory	{ か BEGIN、および } か END を使用できます。

ダイアグラム・タイプのプロパティーの指定

すべてのダイアグラムのデフォルト・プロパティーは*「説明」*です。「説明」は、4074 文字のテキスト・フィールドとして定義されます。ダイアグラムのプロパティーは、ユーザーがダイアグラム全体に設定できるプロパティーで、例えば、スイムレーン (またはプール) を垂直方向または水平方向のどちらに表示するかなどです。一般的な「**ダイアグラム・プロパティー**」ダイアログを以下に示します。

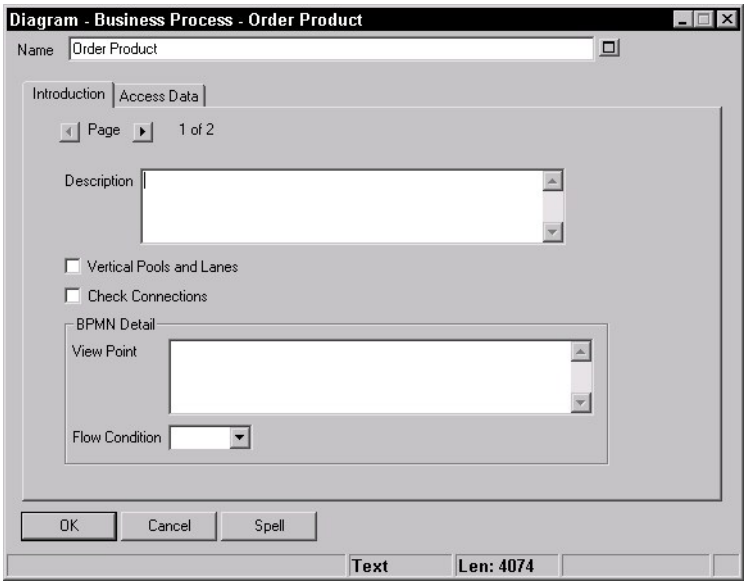


図 2- 13。「ダイアグラム・プロパティー」ダイアログ

ダイアグラムにさらにプロパティーを追加するには、以下の構文を使用します。

```
DIAGRAM diagram_type
{
PROPERTY-1 <property_name>
  { <property_value> }
PROPERTY-2 <property_name>
  { <property_value> }
PROPERTY-3 <property_name>
  { <property_value> }
}
```

例えば、以下のステートメントを USRPROPS.TXT に追加すると、次の図に示すように、「ビジネス・プロセス」ダイアグラム・タイプの「**ダイアグラム・プロパティー**」ダイアログ・ボックスが変更されます。

```
DIAGRAM "Business Process"
{
CHAPTER “My Properties”
PROPERTY "My Property" { EDIT Text LENGTH 1200 }
}
```

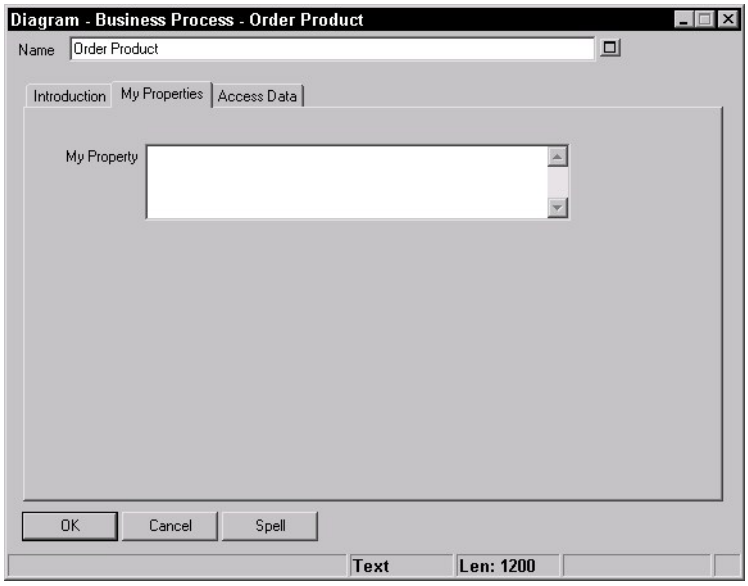


図 2-14。改訂された「ダイアグラム・プロパティー」ダイアログ

シンボル・タイプの
プロパティーの指定

「シンボル・プロパティー」は、シンボルの定義ダイアログの「シンボル」タブ内で提供されます。

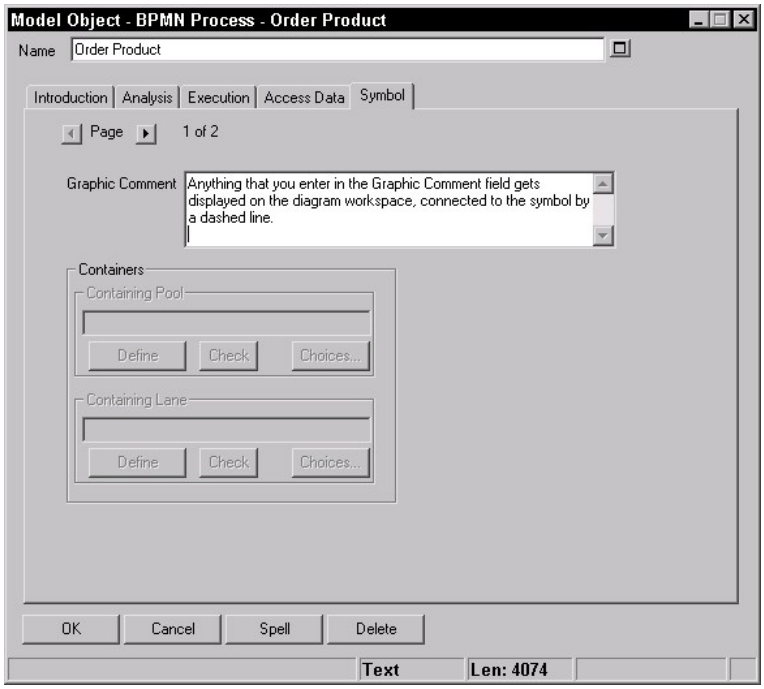


図 2-15。デフォルト・プロパティーが「*グラフィック・コメント*

グラフィック・コメント

すべてのシンボルのデフォルト・プロパティーは*「グラフィック・コメント」*です。*「グラフィック・コメント」*は、4074 文字のテキスト・フィールドとして定義されます。「グラフィック・コメント」フィールドに入力したすべてのものは、ラインによってシンボルに接続され、ダイアグラム・ワークスペースにコメントとして表示されます。ラインは、グラフィック・コメントがシンボルから一定の距離にある場合にのみ描画されます。この距離を調整するには、シンボルを選択し、「書式 (Format)」、「ダイアグラム書式」、「表記」を選択して「離れたテキストまでのライン」オプションを調整します。

グラフィック・コメントをシンボルの内部に表示させることもできます (シンボルを選択し、「書式 (Format)」、「シン

ボル書式」、「テキスト位置」を選択し、「グラフィック・コメントを外へ」選択項目をオフに切り替えます)。「グラフィック・コメント」の表示を完全にオン/オフにすることもできます (シンボルをマウスで右クリックし、「表示モード」を選択してから「グラフィック・コメント」をオフに切り替えます)。

シンボルのプロパティーの追加

シンボルにさらにプロパティーを追加するには、以下の構文を使用します。

```
SYMBOL symbol_type IN diagram_type
{
PROPERTY-1 <property_name>
  { <property_value> }
PROPERTY-2 <property_name>
  { <property_value> }
PROPERTY-3 <property_name>
  { <property_value> }
}
```

重要な点は、参照するシンボルが入っているダイアグラム・タイプを指定することです。ある 1 つのシンボルが、多数の異なるダイアグラム・タイプ上に表示される場合があります。

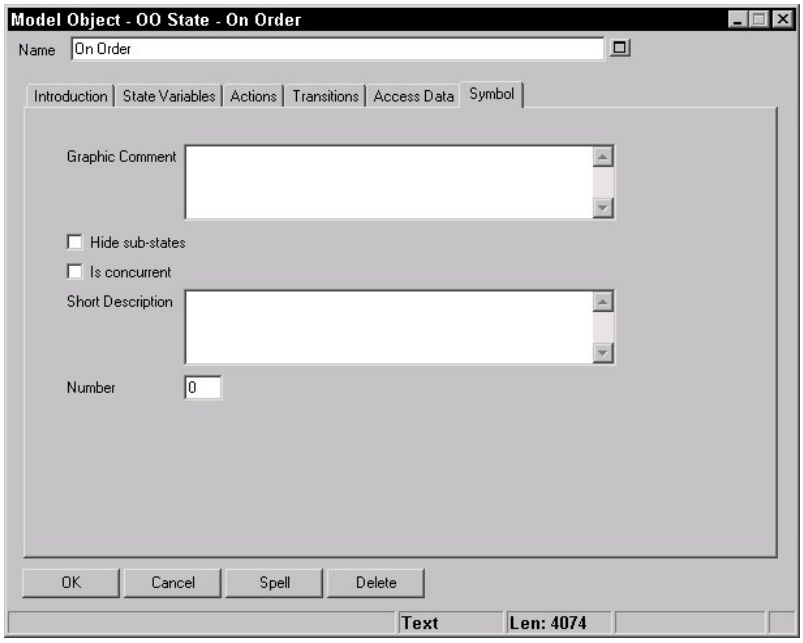
例

例えば、USRPROPS.TXT に対して以下の変更を加えることができます。

```
SYMBOL "State" IN "State"
{
PROPERTY "Short Description"
  { EDIT Text LENGTH 1500 }
PROPERTY "Number" { EDIT Numeric LENGTH 4 }
}
```

これらの変更は、「*概要説明*」と「*番号*」を「UML 状態」図上の状態シンボルのプロパティーに追加します。「*グラフィック・コメント*」は、常に使用可能です。変更後のダイアログ・ボックスを以下に示します。

図 2-16。改訂された「ダイアグラム・プロパティー」ダイアログ



一部のシンボル・タイプは、多数の異なるダイアグラムに現れます。上記の例の説明を続けると、Rational System Architect 内には、状態シンボルを持つその他のタイプの状態図があります。例えば、「IDEF3 オブジェクト状態遷移」図、「OV-06b 運用状態遷移」図、および「状態遷移 Ward & Mellor」図などです。「*概要説明*」および「*番号*」プロパティーをそれら 3 つのタイプで現れるようにしたい場合は、プロパティー・ブロックを 3 回 (ダイアグラム・タイプごとに 1 回ずつ) 組み込む必要があります。

```
SYMBOL "State" IN "IDEF3 Object State Transition"
{
PROPERTY "Short Description"
{ EDIT Text LENGTH 1500 }
PROPERTY "Number" { EDIT Numeric LENGTH 4 }
}

SYMBOL "State" IN "OV-06b Op State Transition"
{
PROPERTY "Short Description"
{ EDIT Text LENGTH 1500 }
PROPERTY "Number" { EDIT Numeric LENGTH 4 }
}
```

```
SYMBOL "State" IN "State Transition Ward & Mellor"
{
PROPERTY "Short Description"
{ EDIT Text LENGTH 1500 }
PROPERTY "Number" { EDIT Numeric LENGTH 4 }
}
```

定義タイプのプロパティーの指定

すべての定義に「*名前*」があります。さらに、どの定義にもデフォルトのプロパティー、「*説明*」があり、この詳細については、このセクションのあとの部分で説明します。実際には、典型的な「**定義**」ダイアログというものは存在しません。定義は、方法論とタイプが固有なものになりがちだからです。以下に「クラス」定義のダイアログを示します。

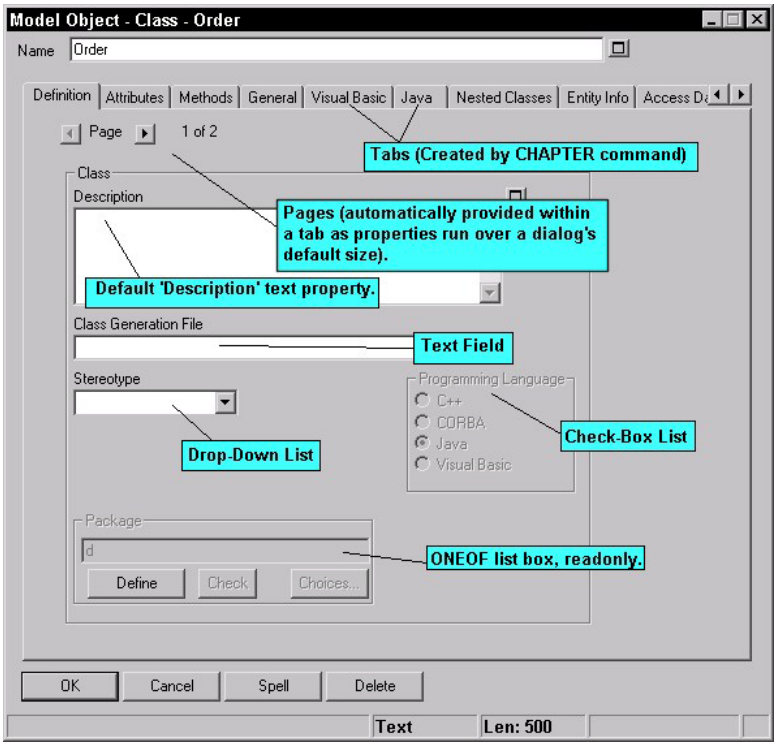


図 2-17。「モデル・オブジェクト」ダイアログ (「定義タイプ」
クラス)

構文

定義ブロックは、キーワード *Definition* で始まり、 定義タイプの名前となる文字列 (引数) が続きます。名前は Rational System Architect で認識されている名前でなければなりません。つまり、SAPROPS.CFG 内にあるか (既存の定義に対する変更または追加を行う場合)、RENAME "User 1" から RENAME "User 150" までのコマンドを使用して作成した名前です (新しい定義タイプを作成するには、これらの 150 個のユーザー提供定義タイプのいずれかを名前変更します)。

スペースが埋め込まれている定義タイプ名は、二重引用符で囲む必要があります (例えば、"User 1")。

辞書定義は、BEGIN...END (または左右の中括弧 **{ }**) キーワードで囲みます。これらの囲みの内部には、一連の定義コマンドを置き、各コマンドは、コマンド・キーワード PROPERTY とそれに続くコマンド引数によって構成します。「**辞書オブジェクト**」ダイアログを呼び出すと、各ページに、定義ブロック内で名前が指定されたプロパティーが入力されます。

それぞれの PROPERTY 項目には独自のサブセット定義があり、これも {**EDIT...** } の中括弧とキーワードで囲まれています。それぞれの定義は、*「ブール」*、*「日付」*、*「式」*、*「ExpressionOf」*、*「ListOf」*、*「ミニスペック」*、*「数値」*、*「OneOf」*、*「テキスト」*、および *「時刻」* などのキーワードで構成された句から成ります。プロパティー定義の詳細は、このセクションのあとの部分で示します。

要約すると、定義にさらにプロパティーを追加するには、以下の構文を使用します。

```
DEFINITION definition_type
{
PROPERTY-1 <property_name>  { <property_value> }
PROPERTY-2 <property_name>  { <property_value> }
PROPERTY-3 <property_name>  { <property_value> }
}
```

以下に例を示します。

```
DEFINITION "Class"
{
CHAPTER "Definition"
GROUP "Class"
{
LAYOUT { COLS 2 ALIGN OVER TAB }
PROPERTY "Description"{ ZOOMABLE EDIT Text LENGTH
500 }
PROPERTY "Class Header File" { EDIT Text LABEL "Class
Generation File" LENGTH 80 }
PROPERTY "Stereotype" { EDIT Text LIST "Class
Stereotypes" INIT_FROM_SYMBOL Default "" LENGTH 20 }
..}
```

上記の例では、定義の最初のタブは (特に指定されなければデフォルトで「はじめに」になりますが)、「Definition」に変更されています。これは、CHAPTER “Definition”コマンドが行っていることです。

説明

このセクションの始めに述べたように、すべての定義にはデフォルトのプロパティー、「*説明*」があります。SAPROPS.CFG 内に別の指定がなければ、「*説明*」は 4074 文字のテキスト・フィールドとして定義されます。「説明」のフィールド・サイズは、USRPROPS.TXT の中で「説明」プロパティーを再指定して文字数を増やすだけで、大きくすることができます。以下に例を示します。

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

上記の例では、クラスの「説明」プロパティーに 16,000 文字を保持できるよう指定していますが、クラス定義のダイアログには最初の 5 行だけが表示されます。

重要な注: Rational System Architect には、「説明」プロパティーを特殊な目的に使用する少数の定義タイプがあります。例えば、「エンティティー」の定義は LISTOF "Attribute" FROM "Data" として再定義されています。「プロセス」定義の「*説明*」プロパティーは、「*ミニスペック*」として再定義されています。これは、プロセスのデータの内容が一般的にミニスペックや、構造化英語や、疑似コードや、それに似たものであるためです。これらの場合のそれぞれにおいて、「説明」プロパティーも、それぞれ「属性」または「ミニスペック」としてラベルが付け直されています。例えば、以下は SAPROPS.CFG 内の「プロセス」定義の指定です。

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

レポートを作成する場合、プロパティーの名前は「*説明*」で、そのようなものとして参照しなければならない点に注意することが重要です。

「プロパティー」ステートメント

「プロパティー」ステートメントは、「ダイアグラム」、「シンボル」、または「定義」指定の中で指定します。「プロパティー」ステートメントの構文は、以下のとおりです。

```
PROPERTY property-name
{ EDIT edit-type
}
```

次の表には、USRPROPS.TXT 内にあるプロパティー・ステートメントの、すべての必須項目とオプションの項目が含まれています。

表 2-3。「プロパティー」ステートメントの必須項目とオプションの項目

入力	必須か オプションか	注釈

PROPERTY property_name
{
}
	Mandatory	左右の中括弧、{..}、または BEGIN .. END ステートメントを使用できます。

EDIT edit-type	オプション	[Boolean | Date | ExpressionOf DATA | ListOf "dictionary-type" | Minispec | Numeric | OneOf "dictionary-type" | Text | Time]
LABEL label_string	オプション	ダイアログ内のコントロールの名前。デフォルトであるプロパティー名を置き換えます。

入力	必須か オプションか	注釈
LENGTH length-argument	オプション	フィールドの最大長 (文字単位) 0 < 数値 < 4095
LIST list-name	オプション	このプロパティーがユーザーによって入力用に選択された場合に、list-name のリストを表示するよう指示します。ユーザーはリストから選択するか、別の値を入力することができます。
LISTONLY LIST list-name	オプション	オプションとして表示されたリストからの入力のみが許可されることを示します。
MINIMUM numeric MAXIMUM numeric	オプション	そのフィールドに指定できる最小/最大の数値。
DISPLAY { FORMAT format-type LEGEND legend-name }	オプション	シンボルについて表示可能な 37 個のプロパティーのうちの 1 つを定義します。
DEFAULT default_string	オプション	ユーザー入力がない場合、ストリング内の項目が使用されます。

表 2-3。「プロパティー」ステートメントの必須項目とオプションの項目 (続き)

入力	必須かオプションか	注釈
READONLY	オプション	キーボードまたは表示されたリストからのすべての入力を停止します。
INVISIBLE VISIBLE	オプション	プロパティーを非表示または可視にします。この項目は SAPROPS 内の値を反転させるために使用します。
CHECKOUT initial-type	オプション	辞書項目がチェックアウトされている場合に自動的に入力される値。 [DATE | TIME | AUDITID]
FREEZE initial-type	オプション	辞書項目が凍結されている場合に自動的に入力される値。 [DATE | TIME | AUDITID]

入力	必須か オプションか	注釈
INITIAL initial-type	オプション	辞書項目が初めてアクセスされて保存された場合に自動的に入力される値。 [DATE | TIME | AUDITID]
UPDATE update-type	オプション	辞書項目が初めておよびそれ以後にアクセスされて保存されるたびに自動的に入力される値。 [DATE | TIME | AUDITID]
ヘルプ	オプション	コントロールにフォーカスがあるときに、ダイアログのステータス・バーに表示される 35 から 40 文字。

ListOf、OneOf、および ExpressionOf の使用

ListOf、OneOf、および ExpressionOf キーワードは、Rational System Architect のメタモデル全体で使用される非常に強力な概念を提供します。それぞれ、ある定義のプロパティーが別のオブジェクト・タイプ (「ダイアグラム」、「シンボル」、または「定義」) を参照していることを表す機能を備えています。

したがって、これらを使用すると、あるクラスにメソッドのリストが含まれていること (ListOf コマンド)、または 2 つのオブジェクト間で、あるメッセージが呼び出し側オブジェクト内のメソッドを参照していること (OneOf コマンド)、または、あるプロセスがデータに対して実行されたプロシージャーを表していること (ExpressionOf) を表現できます。以下の各セクションでは、これら 3 つの式を順に説明します。

注: USRPROPS.TXT 内で指定されるすべてのキーワードと同様に、ListOf、OneOf、または ExpressionOf キーワードの大/小文字の区別は重要ではありません。このマニュアルでは、すべてのキーワードをすべて大文字で表記していますが、このセクションだけは例外です。これらのキーワードを「すべて大文字」で LISTOF、 ONEOF、および EXPRESSIONOF とするよりも、ListOf、OneOf、または ExpressionOf とした方が説明的であるからです。

ListOf

ListOf コマンドを使用すると、プロパティーに他のオブジェクト (ダイアグラム、シンボル、または定義) のリストが含まれることを指定できます。例えば、あるクラスに、クラス属性のリストである Attributes というプロパティーが含まれているとします。クラス属性は、それ自体が定義タイプであり、独自のプロパティーのセットが割り当てられています。参照先のオブジェクト・タイプは、SAPROPS.CFG 内で既に定義されているか、USRPROPS.TXT ファイルの先頭で定義されている必要があります。

ListOf プロパティーを単純なテキスト・リストと比べてください。ListOf リスト内の要素は、ユーザーがリポジトリーに定義を追加するにつれて増加します。単純リストの場合、ユーザーに提示されるリスト内の要素の数は、(USRPROPS.TXT ファイルの冒頭にある LIST ステートメントに基づいており) 変化しません。

ListOf コマンドの構文は以下のとおりです。

```
PROPERTY "Your Property" { EDIT LISTOF <"Referenced Definition Type"> LENGTH 1200}
```

項目のリストのフィルタリング

ListOf コマンド用のリスト内で提供される項目のリストを、フィルターに掛けることができます。フィルター・キーワードには、OF DEFINITION REFERENCED IN および OF DEFINITION AND SUPERS REFERENCED IN などを使用できます。これらのキーワードの詳細については、第 3 章を参照してください。

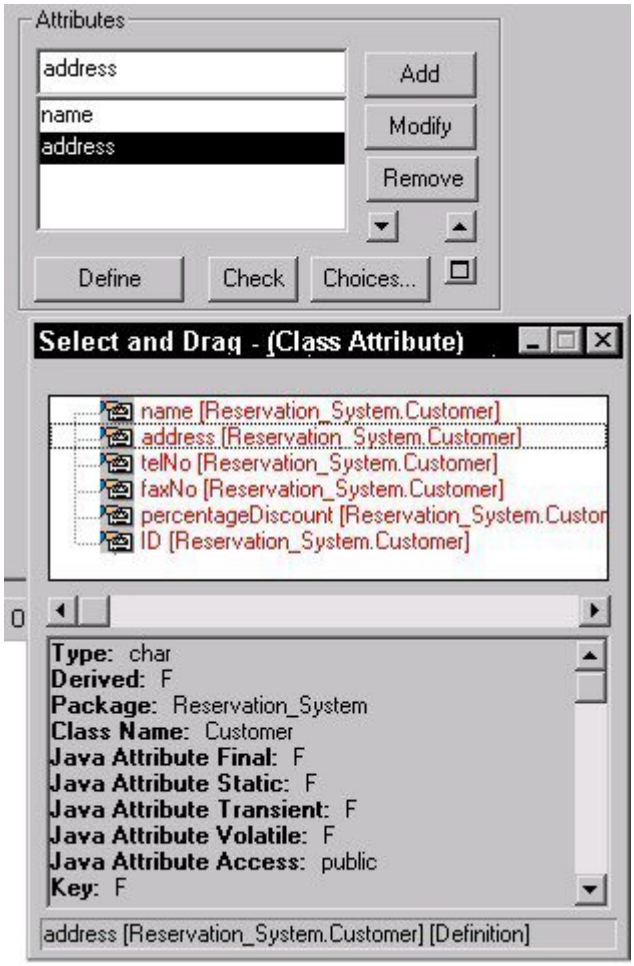
例:

下記の例は、プロパティー "属性 (Attributes)" (これは ListOf "クラス属性 (Class Attributes)" です) を含んでいるオブジェクト定義のコードを示しています。"クラス属性 (Class Attributes)" は別の定義タイプであり、SAPROPS.CFG 内で定義されています。

Definition “Object”
{ ..
PROPERTY "Attributes" { ZOOMABLE EDIT **LISTOF "Class Attribute"** OF DEFINITION REFERENCED IN "Class"
KEYED BY {"Package", "Class Name":"Class", Name}
LENGTH 4096 DISPLAY {FORMAT COMPONENT_SCRIPT
_FmtNewUMLObjInstAttr LEGEND "$$FORCE$$"} }
..}

下記の図は、LISTOF コマンドによって作成されたデフォルトの ListOf ダイアログを示しています。これには「選択」ボタンが含まれており、このボタンを押すと、エンサイクロペディア内の参照タイプ (この例では "クラス属性 (Class Attribute)") のすべての定義のリストが提示されます。上記のコード内で、OF DEFINITION REFERENCED IN コマンドは、そのオブジェクトのクラスに含まれているクラス属性のみをリストすることを指定します。

図2-18。LISTOF リストの例。



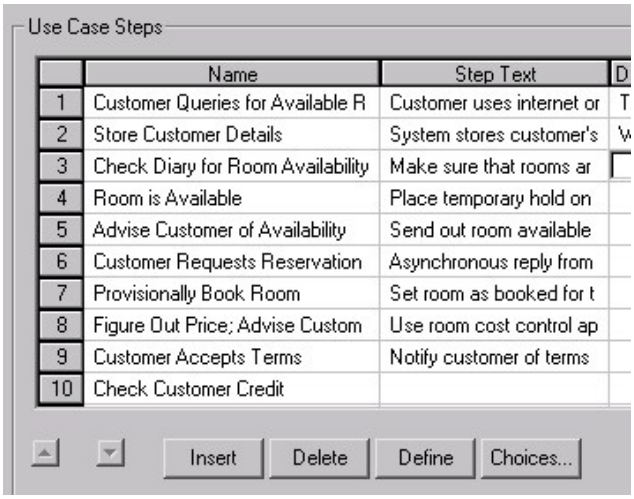
ListOf のグリッドの作成

ASGRID キーワードを追加することにより、ListOf プロパティー内の項目をグリッドとして表すことができます。

例:

```
Definition "Use Case"
{
CHAPTER "Steps"
PROPERTY "Use Case Steps" { EDIT COMPLETE
LISTOF "Use Case Step" KEYED BY { "Package", "Use
Case Name":Name, Name} ASGRID LENGTH 1200 }
}
```

図 2-19。 LISTOF ASGRID リストの例



ListOf の混成リスト

典型的な ListOf ステートメントは、1 つのオブジェクト・タイプだけからなるリストを提供します。HETEROGENEOUSLISTOF キーワードを使用すると、複数のオブジェクト・タイプを参照するリストも作成できます。

例:

```
Definition " Procedure"
{
PROPERTY "Underlying Procedure" { EDIT
HETEROGENEOUSLISTOF " Use Case",
"Class", "Method", "Use Case Step" READONLY}
..}
```

上記の例で、"プロシージャー (Procedure)" 定義の "Underlying Procedure" プロパティーには、「ユースケース」、「クラス」、「メソッド」および「ユースケース・ステップ」の各タイプ、あるいはそのいずれかの定義を取り込むことができます。

HETEROGENEOUSLISTOF コマンドの詳細については、第 3 章を参照してください。

OneOf

OneOf リスト・ボックスは、ある特定のタイプのオブジェクト (「ダイアグラム」、「シンボル」、または「定義」) のリストを 1 つだけ選択できるリスト・ボックスを提供します。参照先のオブジェクト・タイプは、SAPROPS.CFG 内で既に定義されているか、USRPROPS.TXT ファイルの先頭で定義されている必要があります。

例:

```
DEFINITION "Issue"
{
PROPERTY "Assigned To"{EDIT ONEOF "Risk" LENGTH
100}
..}
```

図 2-20。ONEOF リスト・ボックスの例

項目のリストのフィルタリング

ListOf リストと同様に、OneOf コマンド用のリスト内で提供される項目のリストを、フィルターに掛けることができます。フィルター・キーワードには、OF DEFINITION REFERENCED IN および OF DEFINITION AND SUPERS REFERENCED IN などを使用できます。これらのキーワードの詳細については、第 3 章を参照してください。

混成 OneOf リスト

典型的な OneOf ステートメントは、1 つのオブジェクト・タイプだけからなるリストを提供します。ListOf リストと同様に、HETEROGENEOUS**ONE**OF キーワードを使用すると、複数のオブジェクト・タイプを参照する OneOf リストも作成できます。

HETEROGENEOUSONEOF コマンドの詳細については、第 3 章を参照してください。

ExpressionOf

ExpressionOf を使用すると、オブジェクトへの参照を、複雑な演算子および区切り文字を使用して表現することができます。Rational System Architect は、ExpressionOf がデータ要素とデータ構造 (DATA) の参照に使用されることを想定していますが、そのような用途に限定されているわけではありません。

ExpressionOf を使用して定義した参照は、以下の構文を使用してダイアログ・ボックス内に入力します。

A + B + C

または

A +
B +
C

または

A
B
C

要素は、1 つの行または複数の行に書くことができます。1 つの要素と次の要素の区切りは、空白によって定められます。規則では、個々のデータ項目を区切るために + 符号が使用されることになっていますが、+ 符号は必須ではありません。

式を指定する際に、以下の特殊演算子と区切り文字を使用できます。

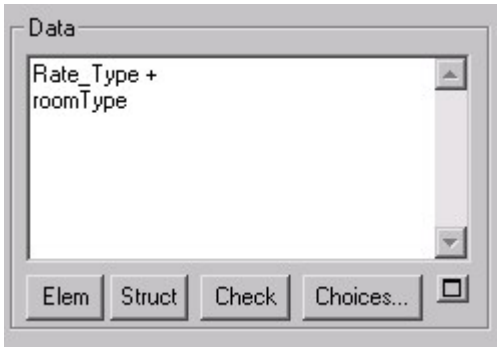
表 2-5。式の指定で使用される特殊演算子と区切り文字

+	AND (オプション)
[...|...]	どちらか一方
{...}	繰り返し
i{...}j	「i」から「j」までの繰り返しを許可する
(...)	囲まれているコンポーネントはオプションである
@	コンポーネントはキー・フィールドである {xe "Key field in a record"}
@n	コンポーネントは複合キーを構成している要素のうちの n 番目の要素である
...	囲まれているテキストはコメントである
/.../	囲まれているテキストはコメントだが、スキーマ・ジェネレーターにとって重要である

副次式を他の式の中にネストさせることができます。例えば、ITERATIONS OF を EITHER OR の大括弧内に組み込むことができます。

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

図 2-21。EXPRESSIONOF リスト・ボックスの例



ZOOMABLE コマンド

ZOOMABLE コマンドを使用すると、大きなテキスト・ブロックを入力しやすくしたり、見やすくしたりするために、一時的にリスト・ボックスのサイズを拡大することができます。このコマンドを追加する最も一般的な場所は、通常ではミニスペックが入力されるプロセス定義や、外部キー情報がかなり長くなりがちなエンティティーの説明プロパティーの中です。

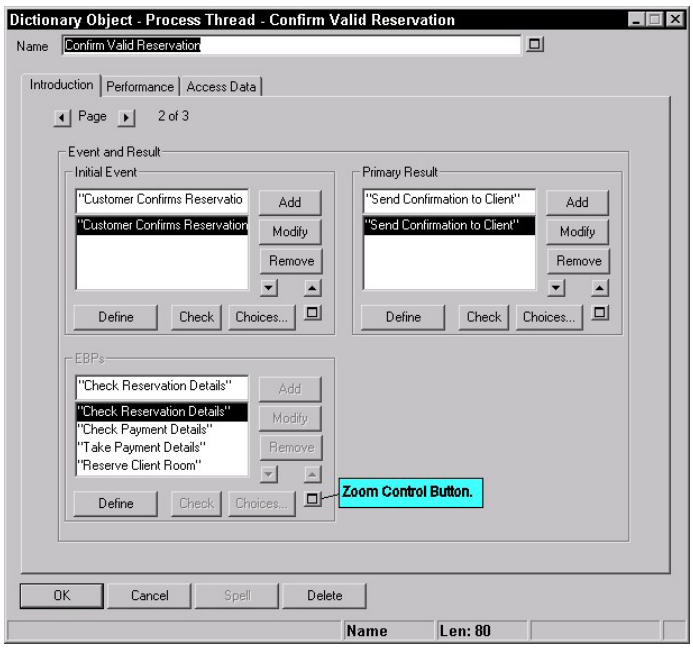
このコマンドは、USRPROPS.TXT の中に次のように書きます。

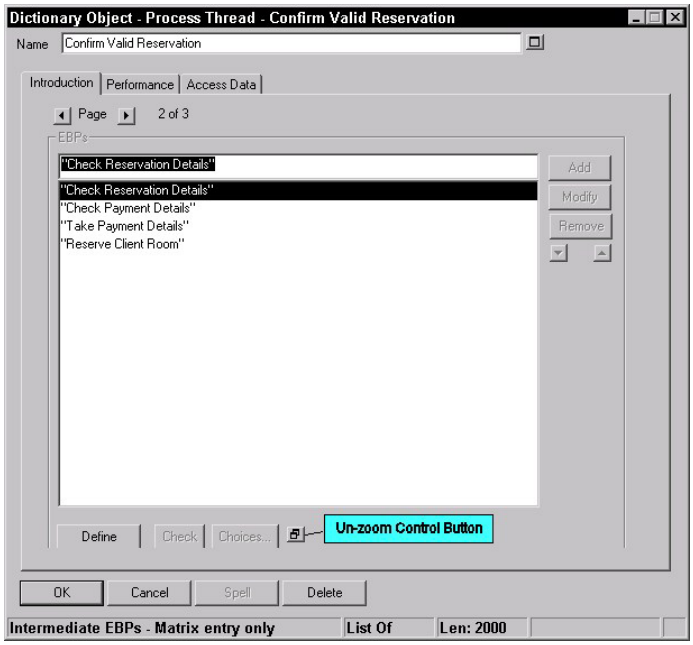
```
DEFINITION "Process"
{
PROPERTY "Description"
{ ZOOMABLE }
}
```

ZOOMABLE コマンドは、リスト・ボックスの右隅に小さなボタンを追加します。このボタンは、ボックスがズームされていない場合は中に正符号が入っており、ズームされている場合は負符号が入っています。

このコマンドの効果を示す実例を、図 2-22 の 2 つの図に示します。上の図では、ミニスペック領域が通常の拡大されていない状態で示されており、下の図では、リスト・ボックスがダイアログ・ページ全体を覆うように拡大されています。

図 2-22。リスト・ボックスの「ズームされていない」状態と「ズームされている」状態





ダイアログの外観調整の変更

ダイアログ内のコントロールとそのラベルの表示を、一定の範囲内で制御することができます。例えば、ラベルをコントロールの上に表示したり、コントロールのすぐ隣に表示したり、グループ内の最も長いラベルによって決められるスペースの量だけ離して表示したりすることができます。

1 つのダイアログ・ページに収めることができるコントロールの数を計算する必要はありません。Rational System Architect は表示に使用できるスペースの量に基づいて、ダイアログ・ページに収まるコントロールの数を自動的に計算し、1 つのダイアログを複数のページに分割します。ユーザーはダイアログの左上の「ページ」矢印によって、ページをめくることができます。

ダイアログ内にあるコントロールのユーザー独自の配置を指定するには、**LAYOUT** コマンドを使用して列、コントロール・ラベルの位置決め、および位置調整を指定し、**CHAPTER** コマンドを使用してタブを作成し、**GROUP** コマンドを使用してプロパティー・コントロールのグループを作成します。さらに、位置決めコントロールを使用して、ダイアログの任意のページについて、各コントロールとラベルの正確な配置を指定できます (方法については、2-98 ページの『*コントロールとラベルの位置決め*』を参照してください)。

LAYOUT コマンド

LAYOUT コマンドを使用すると、1 つのダイアログ内にレイアウトされる列プロパティー・コントロールの数や、コントロールのタイトル (またはラベル) の配置方法 (コントロールの左側か上か) などを指定できます。

LAYOUT コマンドの使用はオプションです。これを使用しなかった場合、Rational System Architect はデフォルトのレイアウト・スキームを実装します。デフォルトのレイアウト・スキームでは、すべてのコントロールは 1 列でレイアウトされ、各コントロールの名前 (またはラベル) はコントロールの左側に配置されます (ALIGN LABEL コマンド)。

LAYOUT コマンドは、CHAPTERS (これは、後続のダイアログ内のタブに対応します)、および「ダイアグラム」、「シンボル」、または「定義」指定の GROUPS の中で指定できます。LAYOUT コマンドは、CHAPTER および GROUP の中で以下のような効果があります。

「章」の中: 「ダイアグラム」、「シンボル」、または「定義」指定の各「章」に、固有の LAYOUT コマンドを指定できます。すべてのプロパティー・コントロールは、グループ全体も含めて、「章」の LAYOUT コマンドに従ってレイアウトされます。1 つの「章」の中で複数の LAYOUT コマンドを指定した場合、その「章」内のすべての LAYOUT コマンドは無視され、デフォルトのレイアウトが代わりに使用されます。

GROUP の中: 「グループ」の中に LAYOUT コマンドを指定して、その「グループ」内のプロパティーがそのグループの LAYOUT 指定に従ってレイアウトされるようにすることができます。1 つの「グループ」の中で複数の LAYOUT コマンドを指定した場合、その「グループ」内のすべての LAYOUT コマンドは無視され、デフォルトのレイアウトが代わりに使用されます。

LAYOUT コマンドの指定順序とその効果の例を以下に示します。

```
DIAGRAM (or SYMBOL or DEFINITION)
   CHAPTER 1
      LAYOUT 1
```

```
      PROPERTY – LAYOUT 1 に従って (章の中に) レイアウトされる
      PROPERTY – LAYOUT 1 に従って (章の中に) レイアウトされる
      GROUP – LAYOUT 1 に従って (章の中に) レイアウトされる
       LAYOUT 2
         PROPERTY – LAYOUT 2 に従って (グループの中に) レイアウトされる
         PROPERTY – LAYOUT 2 に従って (グループの中に) レイアウトされる
      GROUP – LAYOUT 1 に従って (章の中に) レイアウトされる
       LAYOUT 3
         PROPERTY – LAYOUT 3 に従って (グループの中に) レイアウトされる
         PROPERTY – LAYOUT 3 に従って (グループの中に) レイアウトされる
   CHAPTER 2
    LAYOUT 4
      PROPERTY – LAYOUT 4 に従って (章の中に) レイアウトされる
   CHAPTER 3
    LAYOUT 5
      PROPERTY – この「章」に 2 つの LAYOUT コマンド (5 と 6) があるため、デフォルト・スキームによってレイアウトされる
    LAYOUT 6
```

「はじめに」タブのレイアウト

「ダイアグラム」、「シンボル」、または「定義」ダイアログの最初の「章」は「説明」プロパティーを含んでおり、デフォルトのレイアウト・スキーム (1 列のレイアウトで「説明」ラベルがテキスト・ボックスの左側にあります) によって常にレイアウトされることに注意してください。

デフォルトのレイアウト動作

プロパティー・コントロールの幅が広すぎて、LAYOUT コマンドで指定された列構造に収まらない場合、そのコントロールはダイアログまたは「グループ」内に収まるように、1 列でレイアウトされます。それ以外の、LAYOUT コマンドに従ってレイアウトするのに差し支えない幅のコントロールは、そのとおりにレイアウトされます。

例えば、「グループ」に 4 列のレイアウトを指定し、そのグループ自体は指定された 2 列のレイアウトを持つ「章」(タブ) に置かれており、そのグループ内のプロパティーの 1 つが、使用可能なスペースに収まらないほど幅が広い一方、それ以外のプロパティーは 4 列のレイアウト内に収まるほど小さい場合、幅が広すぎるプロパティーは単独でレイアウトされ、それ以外のプロパティーは「グループ」内の 4 列のレイアウトに適合するようにレイアウトされます。

例

以下の例では、新規に定義したユーザー指定の定義の中で、CHAPTER および GROUP ステートメントの内部における LAYOUT コマンドの効果を調べてみます。

```
RENAME DEFINITION “User 1” TO “My Definition”
DEFINITION "My Definition"
{
LAYOUT { COLS 3 ALIGN OVER TAB }
   PROPERTY "My Property 1"{ EDIT Text Length 10}
   PROPERTY "My Property 2"{ EDIT Text Length 10}
   GROUP "No Layout Specified" {
     PROPERTY "My Property 3"{ EDIT Text Length 10}
     PROPERTY "My Property 4"{ EDIT Text Length 10}
     PROPERTY "My Property 5"{ EDIT Text Length 10}
     PROPERTY "My Property 6"{ EDIT Text Length 10}
}
  CHAPTER "4-Col Layout"
   LAYOUT { COLS 4 ALIGN OVER TAB }
   GROUP "2-Column Group" {
      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 "1-Column Group" {
      LAYOUT { COLS 1 ALIGN OVER   TAB }
      PROPERTY "Group Property 5"{ EDIT Text Length 10}
      PROPERTY "Group Property 6"{ EDIT Text Length 10}
      }
    PROPERTY "My Property 7"{ EDIT Text Length 5}
    PROPERTY "My Property 8"{ EDIT Text Length 5}
    PROPERTY "My Property 9"{ EDIT Text Length 5}
    PROPERTY "My Property 10"{ EDIT Text Length 5}
    PROPERTY "My Property 11"{ EDIT Text Length 1200}
    PROPERTY "My Property 12"{ EDIT Text Length 1200}

  CHAPTER "2-Column Layout"
   LAYOUT { COLS 2 ALIGN LABEL TAB }
    PROPERTY "My Property 13"{ EDIT Text Length 10}
    PROPERTY "My Property 14"{ EDIT Text Length 10}
    PROPERTY "My Property 15"{ EDIT Text Length 10}
    PROPERTY "My Property 16"{ EDIT Text Length 10}
    PROPERTY "My Property 17"{ EDIT Text Length 10}
    GROUP "3-Column Group" {
      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 "3-Column Layout"
LAYOUT { COLS 3 ALIGN OVER TAB }
PROPERTY "My Property 18"{ EDIT Text Length 10}
PROPERTY "My Property 19"{ EDIT Text Length 10}
PROPERTY "My Property 20"{ EDIT Text Length 10}
PROPERTY "My Property 21"{ EDIT Text Length 10}
PROPERTY "My Property 22"{ EDIT Text Length 10}
PROPERTY "My Property 23"{ EDIT Text Length 10}
}
```

以下の図で、この USRPROPS.TXT コードを調べてみます。最初の図は、「定義」LAYOUT { COLS 3 ALIGN OVER TAB } 内の一番上のレイアウト・コマンドが無視されることを示しています。その理由は、このコマンドがどの CHAPTER にも割り当てられておらず、最初の "はじめに" タブ (これはデフォルトの 1 列レイアウトになるよう設定されています) のレイアウトをオーバーライドできないためです。

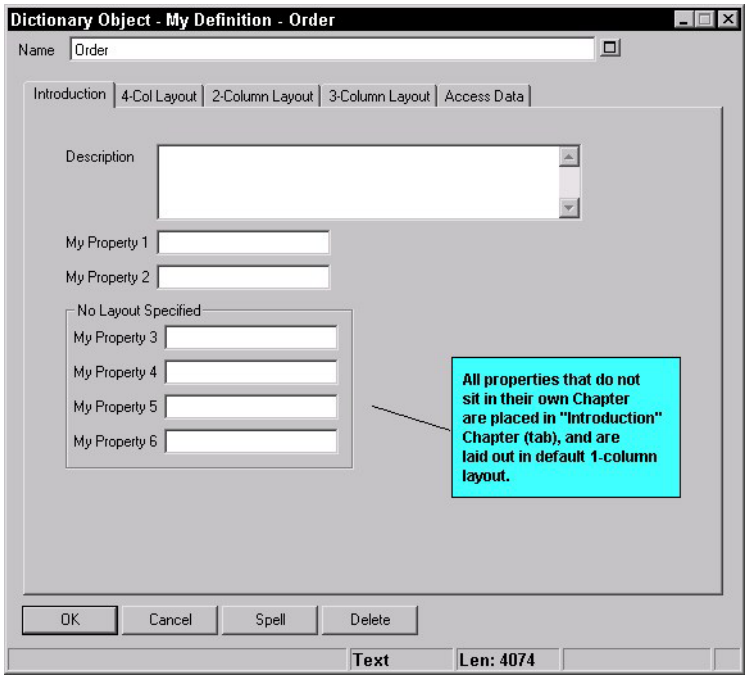
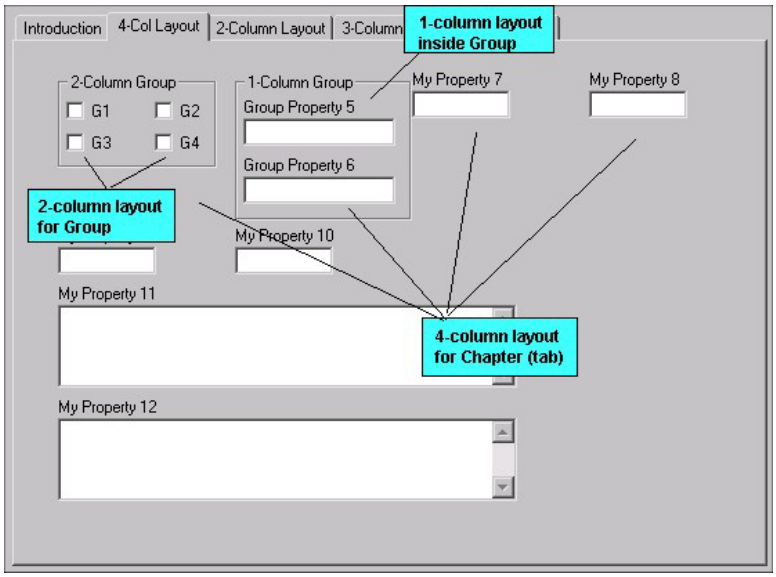


図 2-23。最初の LAYOUT コマンドは CHAPTER (タブ) に割り当てられていないため、無視されます。

ダイアログ内の 2 番目のタブは、CHAPTER “4-Col Layout” コマンドによって指定されています。そのレイアウトは 4 列になるように指定されており、各コントロールのタイトルまたはラベルは、コントロールの上に配置されます (CHAPTER "4-Col Layout" LAYOUT { COLS 4 ALIGN OVER TAB })。

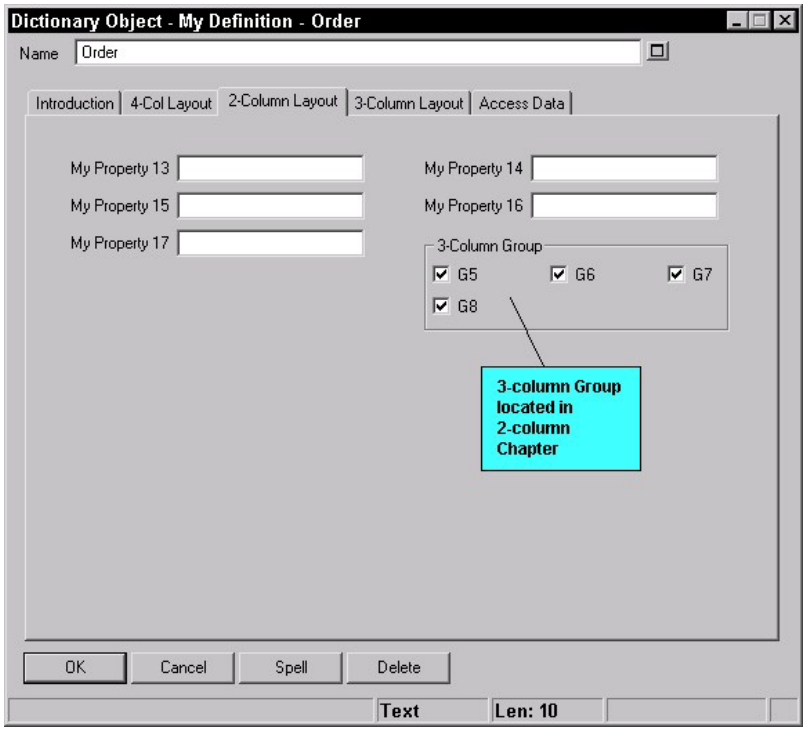
下の図から分かるのは、グループ全体 ("2-Column Group" および "1-Column Group" など) さえも、プロパティー ("My Property 7" から "My Property 10" まで) と同様に「章」に 4 列レイアウト内でレイアウトされているということです。4 列レイアウト・スキーム内に収まらないほど幅が広いプロパティーは、1 列でレイアウトされます (「長さ」が 1200 の "My Property 11" および "My Property 12" など)。

図2-24。2 列と 1 列のグループを含んでいる 4 列の「章」。



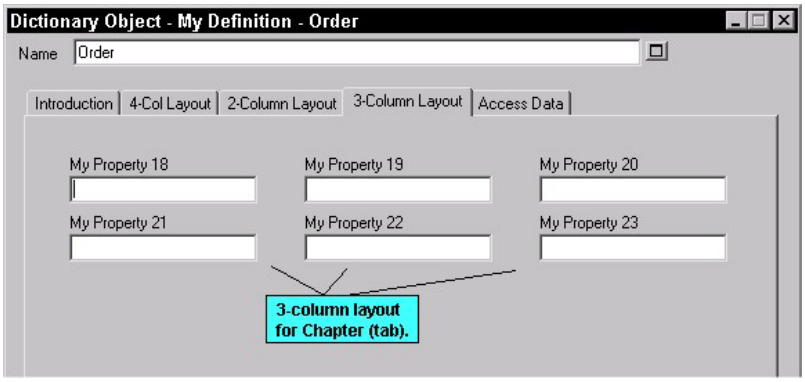
同様に 2 列の「章」には 2 列でレイアウトされるプロパティーが入っており、それには、プロパティーが 3 列でレイアウトされる「グループ」が含まれています。

図 2-25。3 列の「グループ」が入っている 2 列の「章」



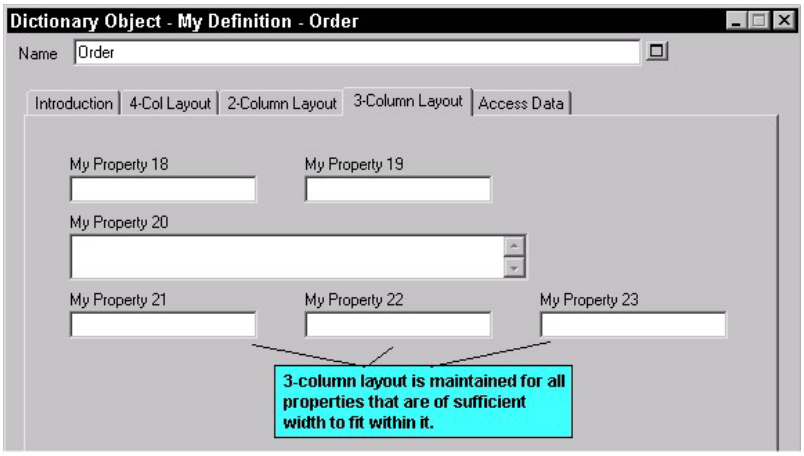
最後の「章」には、3 列レイアウトでプロパティーが入っています。これらのプロパティーは、3 列に十分に収まる幅の狭さ (「長さ」が 10) であることに注意してください。

図 2-26。3 列の「グループ」が入っている 2 列の「章」



これらのプロパティーのいずれかが 3 列レイアウトに収まらないほど幅が広ければ、そのプロパティーは他から独立して 1 列フォーマットでレイアウトされます。"My Property 20" を LENGTH 10 から LENGTH 100 に変更すると、このコントロールは下の図に示すように表示されます。ダイアログ内のこれ以外のプロパティーは、すべて 3 列のレイアウトのままとなります。

図 2-27。3 列の「グループ」と 1 つの幅の広いプロパティーが入っている 2 列の「章」。



LAYOUT コマンドの引数

LAYOUT コマンド内で使用されるサブコマンドの有効な値は、以下のとおりです。

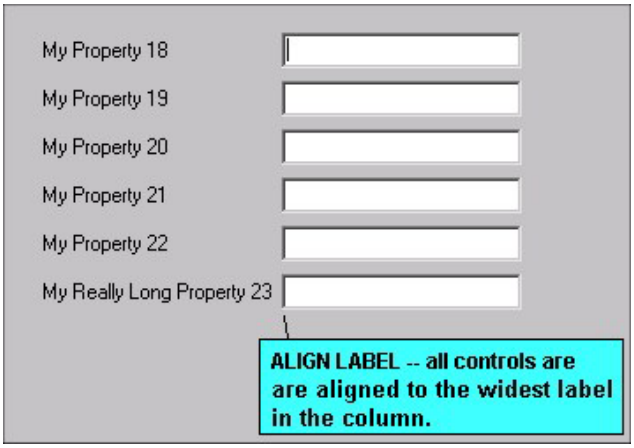
LAYOUT { [ALIGN BODY | ALIGN LABEL | ALIGN OVER] [PACK | TAB] COLS <number> **}**

サブコマンドの順序は重要ではありません。

プロパティーのタイトル (またはラベル) をコントロールに揃える

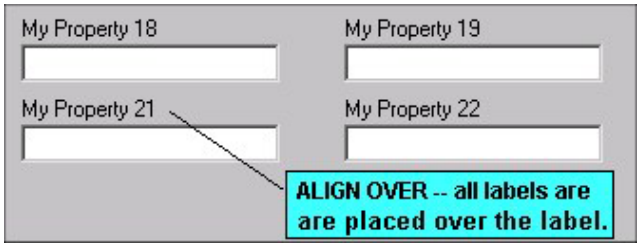
すべてのプロパティーはタイトル、または名前を持っています。LABEL コマンドを使用すると、プロパティーのラベルを変更できることを覚えておいてください。ALIGN コマンドは、プロパティーのタイトルを取得するか、プロパティーのラベルが変更された場合はそのラベルを取得して、以下のようにコントロール自体の隣の特定の位置に配置します。

1. **ALIGN BODY** および **ALIGN LABEL**: すべてのコントロールは、その列内で最も幅が広いラベルの右にスペースが 1 つ空くように桁揃えされます。



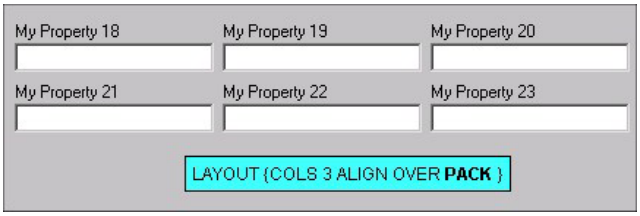
(注 – 以前、ALIGN BODY は、すべてのコントロールをラベルの右側からスペース 1 つ分の位置に置くために使用していましたが、その後、ALIGN LABEL と同じ動作に変更されました。)

2. **ALIGN OVER**: ラベルはコントロールの上に置かれます。

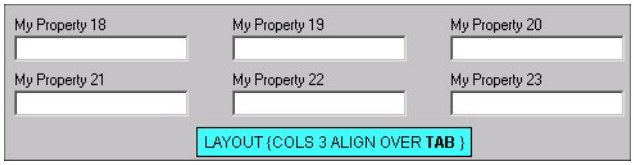


垂直位置決め

1. **PACK**: 複数の列内のコントロールとラベルのセットは、右側にある次のセットと最小のスペース量で区切られます。



2. **TAB**: 複数の列内のコントロールとラベルはタブによって区切られるため、各行の項目は、上の行の項目の直下に揃えて配列されます。



列

COLS <number_of_columns>: プロパティーを分割する列の数を制御します。

位置調整

JUSTIFY: このコマンドは SAPROPS.CFG でも USRPROPS.TXT でも使用されなくなりました。USRPROPS.TXT パーサーは、これを無視します。これは、すべてのコントロールをダイアログ・ページの左右のマージンに揃えて配列するものでした。

CHAPTER コマンドでのタブの作成

CHAPTER コマンドを使用すると、ダイアログ・ページの内容を制御し、タブを生成することができます。1 つのタブ内に表示されるより多くの情報がある場合、そのタブ内に複数のページが自動的に作成されます。

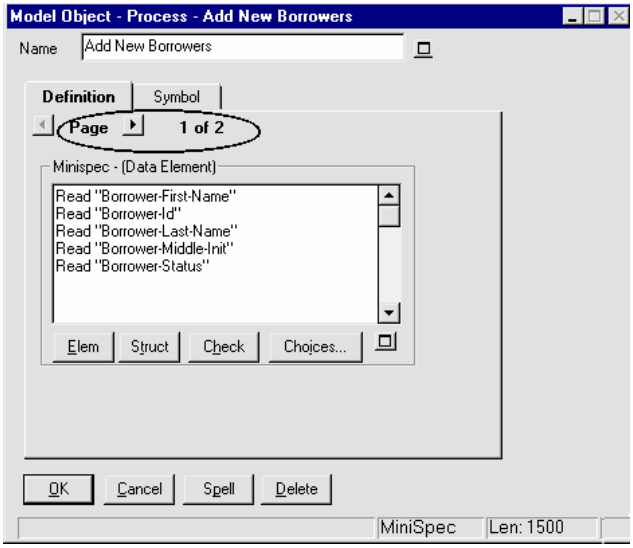


図 2-28。「定義」タブの 2 つのページを示している「モデル・オブジェクト」ダイアログ。

CHAPTER コマンドの構文と配置

タブを作成するには、CHAPTER コマンドを使用し、引数としてタブ名を (スペースを埋め込む場合は二重引用符で囲んで) 指定します。CHAPTER コマンドに左右の中括弧 { } や BEGIN .. END ステートメント・ブロックは必要ありません。

CHAPTER Name_of_Tab

または

CHAPTER “Name of Tab”

指定の中で CHAPTER コマンドの後にリストされているすべてのプロパティーは、次の CHAPTER コマンドが検出されるまで、その CHAPTER (またはタブ) 内に配列されます。**CHAPTER** コマンドは、GROUP ブロック内を除き、USRPROPS.TXT のダイアグラム、シンボル、または定義の指定内の任意のポイントに配置できます。

注: このコマンドでは「CHAPTER」という単語が、もっと明確な単語である「TAB」の代わりに使用されます。その理由は、「TAB」には USRPROPS.TXT 内で常に別の意味があるからです (これは LAYOUT コマンドの中で使用されます)。

以下の規則が CHAPTER コマンドに有効です。

- **CHAPTER** コマンドを使用せずに USRPROPS.TXT によって追加されたプロパティーは、すべての定義プロパティー・ステートメントの末尾に配置され、したがって、定義ダイアログを構成している最後の「定義」タブの最後のページに配置されます。(定義にはシンボルに関係する情報タブを 1 つ以上 含めることができ、それらは定義ダイアログの最 後にくることに注意してください。 これらの「シンボル」タブの中で、最初のタブは 多くの場合「シンボル」という名前ですが、別の名前に変更しても構いません。 シンボル・タブは、ダイアグラム上でシンボルの定 義ダイアログを開いた場合にのみ表示されます。「エクスプローラー」から定義を開いた場合には表示されません)。
- USRPROPS.TXT 内で **CHAPTER** コマンドを使用してタブを再指定することにより、SAPROPS.CFG 内で呼び出された既存のタブにプロパティーを追加できます。追加したプロパティーは、ダイアログ内のそのタブの末尾に配置されます。
- 既存の「ダイアグラム」、「シンボル」、または「定義」指定に追加した新しい **CHAPTER** コマンドのタブは、ダイアログの末尾の、既に存在するすべてのタブの後に配置されます。
- **GROUP** コマンドが必要な場合は、それを **CHAPTER** コマンドの内部でネストさせる必要があります。

CHAPTER 内での LAYOUT コマンドの使用

LAYOUT コマンドは CHAPTER コマンドの下で任意の場所に使用でき、その CHAPTER 内にあるプロパティーのすべてのコントロールに対して影響を及ぼします。1 つの CHAPTER 内に複数の LAYOUT コマンドを指定した場合、

USRPROPS.TXT パーサーはそれらすべてを拒否し、デフォルトのレイアウト (COLS 1 ALIGN LABEL) を提供します。

GROUP コマンド

GROUP コマンドは「グループ」ボックス内に一連のプロパティー・コントロールを配置するために使用します。

「グループ」コマンドの構文

このコマンドの構文は以下のとおりです。

GROUP Name_of_Group

{ <properties to be enclosed in Group>

}

または

GROUP "グループの名前"

{ <properties to be enclosed in Group>

}

上に示したように、GROUP コマンドでは左右の中括弧 { } の中にそのグループ内のプロパティーを指定する必要があります。グループの名前に埋め込みスペースが含まれている場合は、名前を二重引用符で囲む必要があります。GROUP に名前を指定しないこともでき、その場合は、左右の二重引用符の中に何も入力しません。以下に例を示します。

GROUP ““

{ <「グループ」内に囲み込むプロパティー>

}

重要な注: 「ダイアグラム」、「シンボル」、または「定義」指定の中にあるすべての「グループ」名は、異なる「章」の中に置かれる場合でも、固有でなければなりません。したがって、例えば、ある定義の 1 つの「章」の中に "x" という「グループ」を作成し、異なるプロパティーを入れた 2 番目の "x" という「グループ」を作成した場合、両方の「グループ "x"」のすべてのプロパティーは、1 つの「グループ "x"」(最初の「グループ」の「章」に置かれます) に含まれます。同じ指定の中に 2 つ以上の「グループ」を同様な名前で設定したい場合は、「グループ」の後続の設定箇所にはブランク・スペース (単数または複数) を追加します。上の例では、「グループ "x "」のようにします。

GROUP 内での LAYOUT コマンドの使用

GROUP 内に LAYOUT コマンドを指定することができます。指定した場合、その「グループ」が属する定義または CHAPTER (タブ) の LAYOUT コマンドは、そのグループのプロパティーに対してのみオーバーライドされます。

例

下の図に示す各「グループ」を作成するためには、以下の USRPROPS.TXT コードを使用します。

```
RENAME DEFINTION “User 4” To “My Group Definition”

DEFINITION “My Group Definition”
{
CHAPTER "4-Col Layout"
LAYOUT { COLS 4 ALIGN OVER TAB }
        GROUP "2-Column Group"
        {
        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 "Group Property 5"{ EDIT Text Length 10}
         PROPERTY "Group Property 6"{ EDIT Text Length 10}
         }
}
```

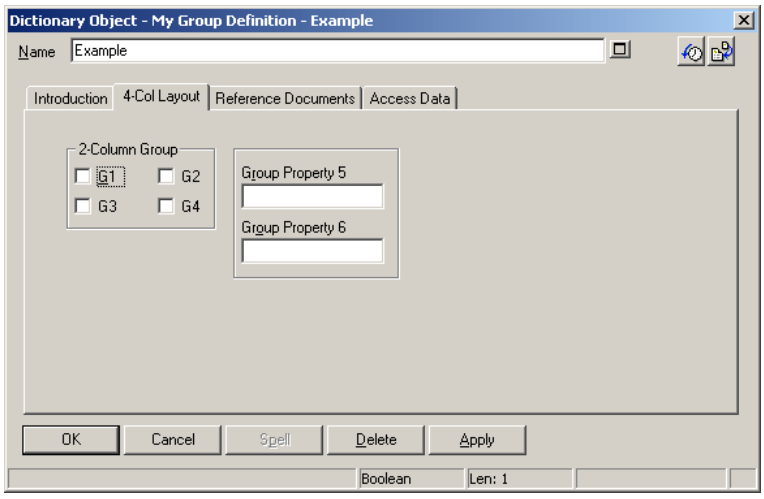


図 2-29。GROUP コマンドと LAYOUT コマンドの使用

コントロールとラベルの位置決め

正確な配置の構文は次のとおりです。

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

例 1:

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

さらに、ラベルの位置決めを指定できます。正確なラベル配置の構文は、以下のとおりです。

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

例 2:

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

上記の例 1 では、PROPPOS (4,12) によって、コントロールが、ダイアログの左上隅から水平方向に 4 Windows 単位、垂直方向に 12 Windows 単位離れた位置に配置されます。別の言い方をすれば、ダイアログの左上隅から左方向に 4 単位、および下方向に 12 単位離れた位置に配置されます。PROPSIZE (150,40) によって、コントロールの幅が 150 Windows 単位、および長さが 40 Windows 単位に設定されます。

上記の例 2 では、LABELPOS によって、コントロールのラベルが、ダイアログの左上隅から水平方向に 4 Windows 単位、および垂直方向に 2 Windows 単位離れた位置に配置されます。したがってこのラベルは、水平方向についてはダイアログの端からの距離はコントロールと同じですが、垂直方向については 10 Windows 単位分だけコントロールより上になります。

重要な注:PLACEMENT を CHAPTER 内のデフォルトの位置決めコマンドと**混用しない**でください。混用すると、位置決めの結果がおかしくなります。

「**エンティティー定義**」ダイアログ用の SAPROPS.CFG からの構文部分を、ダイアログの図と一緒に、以下に示します。PLACEMENT 構文を図と比較してください。

```
CHAPTER "SQL Server Triggers & Table Segment"
GROUP "Default Referential Integrity Triggers"
        LABEL "Default Referential Integrity" {
        LAYOUT { COLS 1 ALIGN LABEL TAB }

PROPERTY "Insert Trigger Name"
        { EDIT Text LENGTH 31
        LABEL "Insert Trigger"
        PLACEMENT {LABELPOS(4, 24)
            PROPPOS(50, 24) PROPSIZE(110, 12)} }

PROPERTY "Update Trigger Name"
        { EDIT Text LENGTH 31
        LABEL "Update Trigger"
        PLACEMENT {LABELPOS(4, 38)
            PROPPOS(50, 38) PROPSIZE(110, 12)} }

PROPERTY "Delete Trigger Name"
        { EDIT Text LENGTH 31
        LABEL "Delete Trigger"
        PLACEMENT {LABELPOS(4, 52)
            PROPPOS(50, 52) PROPSIZE(110, 12)} }
}
```

いくつかの一般的なサイズ変更規則

ほとんどの場合、長さ (X 座標) に何らかの変更を加えて、望みどおりに収まるようにする必要があります。

1. チェック・ボックスは、PROPSIZE (30, 12) です。
2. OneOf プロパティーは PROPSIZE (150, 40) です。
3. ListOf プロパティーは PROPSIZE (320, 98) です。
4. 短いテキスト・フィールドは、概ね PROPSIZE (<LENGTH*3>, 12) で、見かけ上、幅 (x) 座標は丸められます。

5. 長いテキスト・フィールド、例えば LENGTH 4074 などは、PROPSIZE (150,115) 程度です。

いくつかの一般的な配置規則

この場合も、いったんダイアログのレイアウトを表示した後に何らかの変更が必要となる可能性がありますが、以下の数値が開始点として役立つと思われます。

表 2-4。プロパティーがグループ内にありラベルがプロパティーの上にある場合の位置決めとサイズ。

PROPERTY タイプ	PLACEMENT - 左端の列
ListOf	PLACEMENT { PROPPOS (4, 24) PROPSIZE (320, 98) }
OneOf	PLACEMENT { PROPPOS (4, 24) PROPSIZE (150, 40) }
EDIT テキスト (75 文字未満)	PLACEMENT { PROPPOS (4, 24) PROPSIZE (LENGTH * 3, 12) }
PROPERTY タイプ	PLACEMENT - 右端の列
ListOf	PLACEMENT { PROPPOS (165, 24) PROPSIZE (320, 98) }
OneOf	PLACEMENT { PROPPOS (165, 24) PROPSIZE (150, 40) }
EDIT テキスト (75 文字未満)	PLACEMENT { PROPPOS (165, 24) PROPSIZE (LENGTH * 3, 12) }

表 2-4。プロパティーがグループ内にありラベルがプロパティーの上にない場合の位置決めとサイズ。

PROPERTY タイプ	PLACEMENT - 左端の列
ListOf	PLACEMENT { PROPPOS (4, 12) PROPSIZE (320, 98) }
OneOf	PLACEMENT { PROPPOS (4, 12) PROPSIZE (150, 40) }
EDIT テキスト (75 文字未満)	PLACEMENT { PROPPOS (4, 12) PROPSIZE (LENGTH * 3, 12) }
PROPERTY タイプ	PLACEMENT - 右端の列
ListOf	PLACEMENT { PROPPOS (165, 12) PROPSIZE (320, 98) }
OneOf	PLACEMENT { PROPPOS (165, 12) PROPSIZE (150, 40) }
EDIT テキスト (75 文字未満)	PLACEMENT { PROPPOS (165, 12) PROPSIZE (LENGTH * 3, 12) }

表 2-4。プロパティーがグループ内になくラベルがプロパティーの上にない場合の位置決めとサイズ。

PROPERTY タイプ	PLACEMENT - 左端の列
ListOf	PLACEMENT { PROPPOS (4, 2) PROPSIZE (320, 98) }
OneOf	PLACEMENT { PROPPOS (4, 2) PROPSIZE (150, 40) }
EDIT テキスト (75 文字未満)	PLACEMENT { PROPPOS (4, 2) PROPSIZE (LENGTH * 3, 12) }
PROPERTY タイプ	PLACEMENT - 右端の列
ListOf	PLACEMENT { PROPPOS (165, 2) PROPSIZE (320, 98) }
OneOf	PLACEMENT { PROPPOS (165, 2) PROPSIZE (150, 40) }
EDIT テキスト (75 文字未満)	PLACEMENT { PROPPOS (165, 2) PROPSIZE (LENGTH * 3, 12) }

表 2-4。プロパティーがグループ内になくラベルがプロパティーの上にある場合の位置決めとサイズ。

PROPERTY タイプ	PLACEMENT - 左端の列
ListOf	PLACEMENT { PROPPOS (4, 14) PROPSIZE (320, 98) }

OneOf	PLACEMENT { PROPPOS (4, 14) PROPSIZE (150, 40) }
EDIT テキスト **(75 文字未満)**	PLACEMENT { PROPPOS (4, 14) PROPSIZE (LENGTH * 3, 12) }
PROPERTY タイプ	PLACEMENT - 右端の列
ListOf	PLACEMENT { PROPPOS (165, 14) PROPSIZE (320, 98) }
OneOf	PLACEMENT { PROPPOS (165, 14) PROPSIZE (150, 40) }
EDIT テキスト **(75 文字未満)**	PLACEMENT { PROPPOS (165, 14) PROPSIZE (LENGTH * 3, 12) }

表 2-4。プロパティーが同じダイアログ内のプロパティーの下にある場合の位置決めとサイズ。

PROPERTY **前のプロパティーのエンドポイントに追加**	PLACEMENT - 左端の列
A **ラベルが「プロパティー B」の上にある場合は +14**	PLACEMENT { PROPPOS (4, 14) PROPSIZE (150, 40) }
B **「プロパティー C」が OneOf である場合は +4**	PROPERTY B PLACEMENT { PROPPOSE (4, 68) PROPSIZE (150, 40) }
C	PLACEMENT { PROPPOS (4, 112) PROPSIZE (150, 12) }
PROPERTY タイプ	PLACEMENT - 右端の列
D **「プロパティー E」のラベルが側面にある場合は +4**	PLACEMENT { PROPPOS (165, 14) PROPSIZE (320, 98) }
E **「プロパティー F」に LABELPOS が使用されているので +4**	PLACEMENT { PROPPOS (165, 116) PROPSIZE (150, 40) }

F	PLACEMENT { LABELPOS (165, 160) PROPPOS (165, 170) PROPSIZE (30, 12) }

注: LABELPOS はオプションです。これは、グループの LAYOUT コマンドをオーバーライドするために使用されます。y 座標は、PROPPOS y 座標より 10 単位だけ上方にあります。

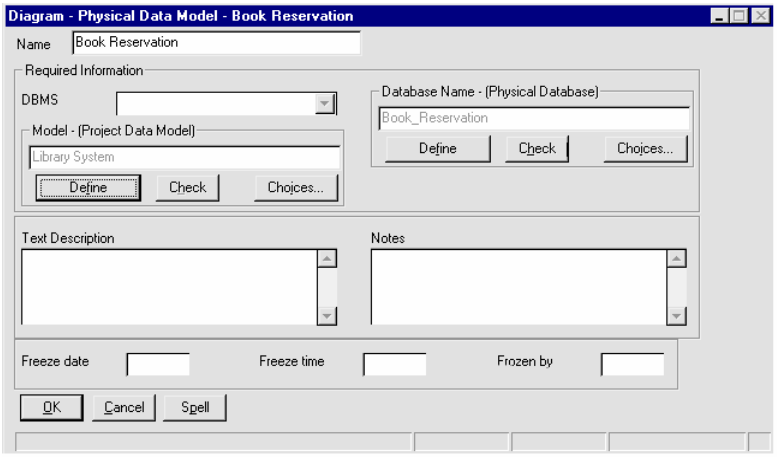


図 2-30。「章」コマンドの結果

シンボル上の値の表示の指定

シンボルは、リポジトリー内の定義を表します。その定義にはプロパティーがあります。その定義プロパティーとその値をシンボル上に表示するように指定できます。デフォルトでは、シンボルの名前 (これはシンボルとその定義のプロパティーです) が表示されます。それ以外のシンボル定義のプロパティーが表示可能であることを指定するには、各プロパティーの宣言の中で **DISPLAY** コマンドを使用します。以下に例を示します。

```
Definition “My Definition”
{
Property “My Property 1” {EDIT TEXT LENGTH 20
DISPLAY { FORMAT String LEGEND "" } }
}
```

上記のコード例では、プロパティー "My Property 1" をシンボル上に表示可能にします。定義ダイアログ内で、このプロパティーの 20 文字のテキスト・ボックスに入力したテキストはすべて、シンボルの前面に表示されます。

シンボル定義の特定のプロパティーを表示可能として指定すると、それらのプロパティーはシンボルの「**表示モード**」ダイアログ内で提供され、そのダイアログでいつでもオンまたはオフにすることができます。「表示モード」ダイアログにアクセスするには、ダイアグラム上でシンボルを選択し、**「表示」**、**「表示モード」**を選択するか、シンボルをマウスで右クリックし、「**表示モード**」を選択します。「表示モード」ダイアログを使用すると、シンボルについて表示したいすべての表示可能プロパティーを選択できます。

下の図は、エンティティー・シンボルを示しています。「変更前」の図は表示可能なすべてのプロパティーをオフにした場合、「変更後」の図は「キー・データ (Key Data)」と「非キー・データ (Non-Key Data)」をオンにした場合のシンボルを示しています。

図 2-31。プロパティーが表示されている場合と表示されていない場合の長方形シンボル

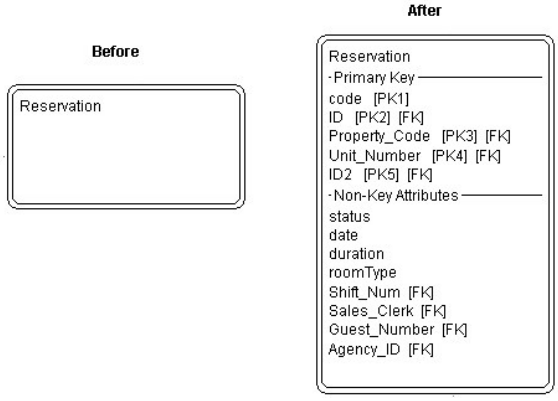
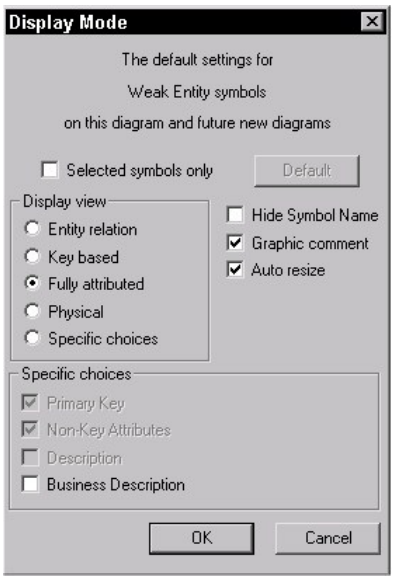
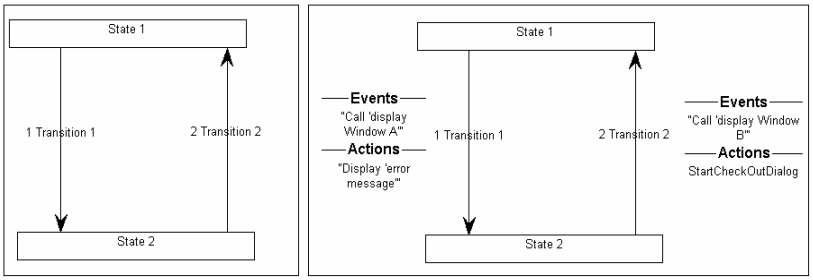


図 2-32。「表示モード」ダイアログの例



下の図に示すように、ライン・シンボルに表示可能プロパティーを指定することもできます。

図 2-33。プロパティーが表示されている場合と表示されていない場合のライン・シンボル



DISPLAY コマンドの構文

1 つの定義につき、display ステートメントは **8** つまでに制限されます。DISPLAY コマンドの構文は、以下のとおりです。

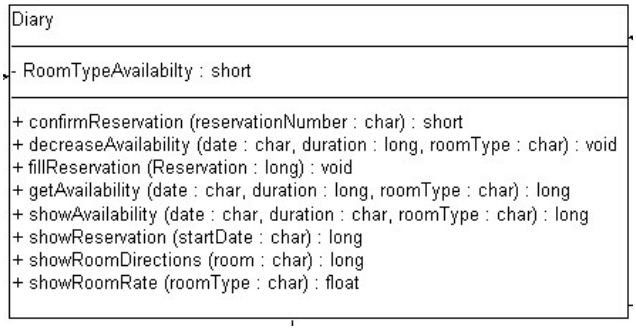
DISPLAY { FORMAT [STRING | LIST | KEY | NONKEY | COMPONENT_SCRIPT | COLUMN_SCRIPT | SCRIPT]
LEGEND " (シンボル内でのブロックのラベル表示方法) **" }**

- STRING: これはデフォルトです。これを指定すると、プロパティーの値が入力されたとおりにシンボル上に表示されます。この選択項目は、コメントを表示する場合に適しています。
- LIST: リスト内のシンボル上に項目を表示します。それぞれの空白文字は、 二重引用符で囲まれていなければ、復帰改行となります。
- KEY: このキーワードは、キーとして指定されたプロパティーに使用します。これらのキーはシンボルの別のセクションに表示されます。 例については、第 3 章の KEY キーワードを参照してください。
- NONKEY: このキーワードは非キー・プロパティーに使用できます。これらはシンボルの別のセクションに表示されます。 このキーワードは、当初は Rational System Architect のデータ・モデリング・サポートでの エンティティーとテーブルに対して使用されていました。例については、第 3 章の NONKEY キーワードを参照してください。
- COMPONENT_SCRIPT: スクリプトを呼び出し、そのスクリプトが指定する特殊なフォーマットでプロパティー値をシンボル上に表示します。スクリプト自体は、製品内にハードコーディングされているか、ユーザーが Rational System Architect の Basic 言語を使用して作成します。慣例的に、ス

クリプト自体の名前には以下のいずれかの接頭部が付けられます。

- fmtxxx - この関数自体はハードコーディングされており、変更することはできません。SAPROPS.CFG の関数の多くはこの形式です。 関数をハードコーディングするのは、Rational System Architect の全体的な応答速度を上げるためです。
- _fmtxxx - Rational System Architect のメイン実行可能ディレクトリーにある fmtscript.bas ファイルの中に存在し、SA Basic を使用してコーディングされています。

コンポーネント・スクリプトは、ListOf および ExpressionOf のプロパティーに対して使用します。スクリプトによるアクションの実行対象は、リスト内の各項目です。 例えば、クラス定義内の各クラス属性を参照し、そのクラス・シンボル上への表示方法を構成するために、コンポーネント・スクリプトが使用されます。属性のアクセス・プロパティーが private に設定されている場合は名前の前に「-」マークが、public である場合は「+」マークが付きます。また、属性名の後に戻り値の型が、前にコロンを付けて表示されます。同様に、コンポーネント・スクリプトはメソッドの表示方法も構成します。アクセスが public であれば名前の前に「+」マークが、アクセスが private であれば「-」マークが付きます。また、メソッドのパラメーターとそのタイプを、メソッド名の後に小括弧で囲んで表示します。



詳しくは、第 3 章の COMPONENT_SCRIPT および SCRIPT を参照してください。

- COLUMN_SCRIPT: COMPONENT_SCRIPT と同様に機能し、シンボル上に表示されるプロパティー値に特殊なフォーマット設定を適用するスクリプトを呼び出します。そのスクリプトは、ハードコーディングされたスクリプトであるか、ユーザーが SA Basic を使用して作成した (そして、Rational System Architect のメイン実行可能ディレクトリー内にある fmtscript.bas ファイル内に配置した) スクリプトです。列スクリプトは、物理データ・モデル内でテーブル・シンボル内の列を表示するために使用されます。このスクリプトによって実行されるアクションは、リスト内の各列に対して機能します。詳しくは、第 3 章の COLUMN_SCRIPT を参照してください。
- SCRIPT: COLUMN_SCRIPT および COMPONENT_SCRIPT と同様に機能し、シンボル上に表示されるプロパティー値に特殊なフォーマット設定を適用するスクリプトを呼び出します。SCRIPT コマンドは、ListOf でも ExpressionOf でもないプロパティーに対して使用されるスクリプトを呼び出します。そのスクリプト自体は、ハードコーディングされたスクリプトであるか、ユーザーが SA Basic を使用して作成した (そして、Rational System Architect のメイン実行可能ディレクトリー内にある fmtscript.bas ファイル内に配置した) スクリプトです。詳しくは、第 3 章の SCRIPT キーワードを参照してください。

表示される各プロパティー・グループは、分割線によって互いに分離されます。分割線上に表示されるラベル、または「凡例」は、LEGEND コマンドを使用して指定できます。凡例を入力しなければ、 プロパティー名自体がラベルになります。以下の LEGEND コマンドを使用できます。

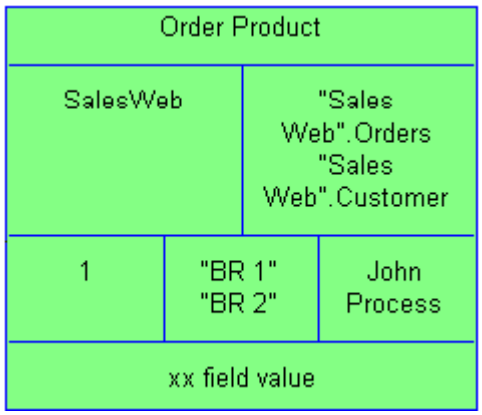
- **LEGEND “<Your Text>”:** 引用符内にどのようなテキストを配置した場合も、該当項目に値がある場合にのみ、引用符で囲まれたそのテキストは項目の上方のシンボル上に表示されます。

- **LEGEND "":** 説明なしの直線を表示します。 ただし、その項目に値がある場合に限られます。

- **LEGEND "$$FORCE$$":**シンボル上の項目の上に横のラインを表示します。このラインは、分割線として機能します。“$$FORCE$$”キーワードは、 単に“ “ を使用する場合とは異なり、プロパティー表示が表示モード・ダイアログによって抑止されていても、横のラインを表示します。

- **LEGEND “$$NONE$$”:** 該当項目の値が存在するかどうかにかかわらず、シンボル内の項目の上方に横線を表示しません。このラインは通常、分割線として機能します。

- **LEGEND “$$VFORCE$$”:** シンボル内でプロパティーを左から右の方向に配置できるようにし、それらの間に縦線を引きます。以下に例を示します。



  上記の図を作成する USRPROPS.TXT の例については、第 3 章の VFORCE キーワードを参照してください。

- **LEGEND “$$VNONE$$”:** プロパティーを左から右にレイアウトできますが、分割線を表示*しません*。例については、第 3 章の VNONE キーワードを参照してください。

表示可能な凡例の書体とフォントは、「**書式 (Format)**」メニューの下にある「**ダイアグラム書式**」、**表記**」コマンドによって制御されます。

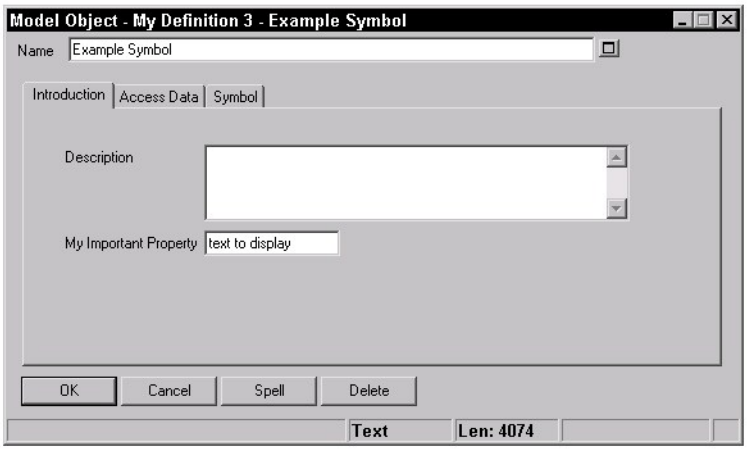
例

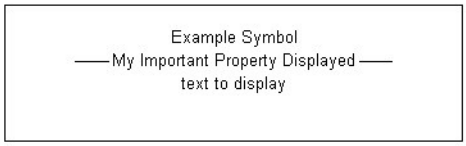
以下の例では、新しいダイアグラム・タイプ、新しいシンボル・タイプ、および新しい定義タイプを指定します。ここでは、新しいシンボル・タイプは新しい定義タイプによって定義されるよう指定し、新しいシンボル・タイプを新しいダイアグラム・タイプに割り当てます。この定義には、"My Important Property" というプロパティーを作成します。この例では、凡例 "My Important Property Displayed" が、シンボル上で、表示される値の上方にある分割線上に表示されるように指定します。

```
RENAME DIAGRAM "User 1" TO "My Diagram"
RENAME SYMBOL "User 1" TO "My Symbol 1"

SYMBOL "My Symbol 1"
{
DEFINED BY "My Definition 3"
ASSIGN TO "My Diagram"
}
DEFINITION "My Definition 3"
{
PROPERTY "My Important Property" { EDIT Text Length
20 DISPLAY { FORMAT String LEGEND "My Important
Property Displayed" }}
}
```

下の図は、そのようなダイアグラム上に描画される後続のシンボルと、プロパティー・フィールドに入力された値の表示を示しています。

図 2-34。プロパティーが表示されている場合と表示されていない場合のライン・シンボル





Key プロパティーと Keyed By プロパティーの指定

KEY プロパティーの設定

特定の定義を、他の 1 つ以上の定義に「キー設定する」ことを指定できます。キーは、エンサイクロペディア内の定義の名前空間を決定します。例えば、クラス属性定義タイプのキーは、それを含むクラス定義、およびそのクラスを含むパッケージ定義によってキー設定できます。したがって、「名前」と呼ばれる 2 つの属性を、1 つは Reservation_System パッケージ内のクラス「顧客」に属する属性として、もう 1 つは Order_System パッケージ内のクラス「製品」に属する属性として持つことができます。これら 2 つの属性は、同じ名前ではありますが、定義は明らかに異なります。

デフォルトでは、エンサイクロペディアの各モデル化要素は、既に暗黙に 3 つのものにキー設定されています。つまり、その要素の *クラス* (ここで、*クラス* は Rational System Architect 用語として使用されており、ダイアグラム、シンボル、定義のどれであるかを区別します)、その要素の *タイプ* (「UML ユースケース」図なのか、「BPMN プロセス」図なのかなど)、およびその要素の *名前* (例えば、「Reservation_System ユースケース」図と「Human_Resource_System ユースケース」図) です。

定義にキー・プロパティーを追加するには、KEY コマンドを使用します。KEY コマンドは、定義のキーにしたいプロパティーの中で指定します。KEY コマンドは、プロパティーの記述内であればどこにでも配置できますが、重要性を考慮して、プロパティーの中括弧内の最初の項目として (EDIT キーワードの直前に) 使用されるのが通例です。

注: ダイアグラムに KEY EDIT ONEOF を追加することはできません。

例 1:

```
Definition “Use Case”
{
PROPERTY "Package" { KEY EDIT …}
..}
```

例 2:

```
Definition “Use Case Step”
{
PROPERTY "Use Case Name" { KEY EDIT … }
PROPERTY "Package" { KEY EDIT …}
            …
```

注: 定義のキー・プロパティーは、ASGRID コマンドで形成されるグリッドには表示されません。 例えば、ユースケース定義内のユースケース・ステップは、ASGRID コマンドで生成されるグリッドに表示されますが、ユースケース・ステップのキー・プロパティー (それを所有するパッケージおよびユースケース) はユースケース・ステップのグリッドには表示されません。

キーであり、かつ別のオブジェクトを「指している」プロパティー (例えば、単純な TEXT または NUMERIC プロパティーではなく、LISTOF または ONEOF プロパティー) の場合、エンド・ユーザーは、Rational System Architect で作業しているときにそのプロパティーの値を入力する際に、参照されるオブジェクトのクラスおよびクラス・タイプを指定する必要があります。

以下に例を示します。

```
Definition “Business Process”
{
PROPERTY “System Use Case” {EDIT ONEOF “Use Case” …}
```

上記のステートメントは、“Use Case Name”プロパティーが“Use Case”タイプの定義を参照することを示しています。Definition は、*クラス* (Rational System Architect における *クラス*、つまり、Diagram、Symbol、または Definition) が指定されない場合のデフォルトです。

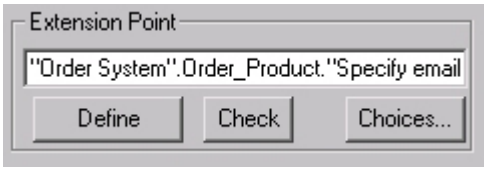
プロパティー値自体には、多くの場合、実際に参照されるオブジェクトを識別するために必要なその他の素材がすべて含まれています。プロパティーの参照対象クラス/タイプにキー・プロパティーが含まれていない場合、参照値は (クラスおよびタイプが既知であるため) オブジェクトの名前のみになりますが、参照対象クラス/タイプにキー・プロパティーが含まれている場合 (上記の例では、キー・プロパティー「package」を含む「Use Case」)、Rational System

Architect は、参照オブジェクトを正しく識別するために、それらのキー・プロパティーの値を知る必要があります。

注: 異種混合の参照プロパティーの場合は、これとは異なります。第 3 章の HETEROGENEOUS を参照してください。

これを USRPROPS.TXT 内でコーディングして Rational System Architect がエンド・ユーザーに代わってこれらの値を自動的に取得できるようにする、またはエンド・ユーザーにピリオドでキー・パーツを区切って完全修飾名を入力させる、のいずれかを行ってください。

- ユーザーに代わって Rational System Architect に値を自動的に取得させるには、KEYED BY コマンドを使用します。
- プロパティーに対して KEYED BY 節が指定されていない場合、Rational System Architect は、これらの追加キー値が参照自体で指定されると想定します。つまり、ユーザーが、ピリオドでキー値を区切って参照オブジェクトの完全修飾名を入力する必要があります (*"Order System"* パッケージ内の *Order_Product* ユースケースの *"Specify email"* ユースケース・ステップの場合、ユーザーは *"Order System".Order_Product."Specify email"* と入力する必要があります。)



注: 構文上で重要な文字 (スペースやピリオドなど) を含んでいるコンポーネントは二重引用符で囲み、Rational System Architect が参照を適切に構文解析できるようにする必要があります。

以下に、「ユースケース・ステップ」を参照する 2 つの例を示します。

Order_System.Order_Product.”Specify email”

ここで、*CorrectInvoice* は *Accounts_Payable* パッケージに属する「ユースケース」の名前です。

“Order System”.”Order Product”.”Specify email”

ここで、*Order Product* は *Order System* パッケージに属する「ユースケース」の名前です。

グループのすべてのメンバーを確実に同じタイプにするための KEYED BY の使用

KEYED BY 節には、関連するすべてのものをリストにするという用途もあります。 例えば、あるユースケース定義の “Use Case Steps”プロパティーで参照されるすべてのユースケース・ステップは、同じユースケース (“Use Case Steps” プロパティーを含むユースケース) に属します。 複数参照プロパティー (ListOf など) で参照するすべてのオブジェクトが同じ親オブジェクトに属している場合、他の 1 つ以上の他のプロパティーを使用して親オブジェクトを識別することをお勧めします。 そのような場合、KEYED BY 節を使用して、他のどのプロパティーを使用するのかを Rational System Architect に指示します。

KEYED BY 節の使用方法

要約すると、KEYED BY 節は、参照先オブジェクトのキー・コンポーネントの検出方法を指定するために、オプションで使用されます。これには、次の 2 つの主な利点があります。

1. エンド・ユーザーが参照値の完全修飾名を (ピリオドで修飾子を区切って) 指定する必要がなくなります。例えば、パッケージ ***"Order System"*** のクラス ***Customer*** の ***email*** という名前のクラス属性を参照するプロパティーの場合、エンド・ユーザーは、***"Order System".Customer.email*** と入力する代わりに、***email*** と入力するだけで済みます。
2. これを使用すると、参照値のすべてのキー・コンポーネントを同じにすることができます。 例えば、クラス定義の LISTOF “Class Attribute”プロパティーでリストされるすべての属性は、同じクラスおよび同じパッケージに属します。

一般に、KEYED BY 節には参照先オブジェクトの各キー・コンポーネントを検出する方法の指定が含まれています。KEYED BY 節には、各キー・コンポーネントの部分をコンマで区切って指定します。

例:

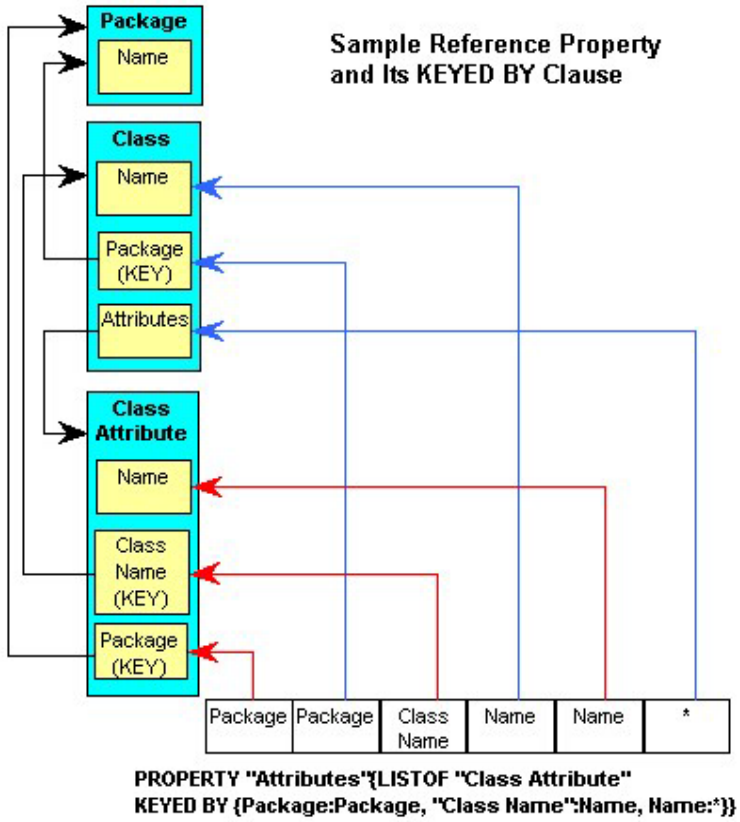
例えば、「クラス (Class)」の "属性 (Attributes)" プロパティーの KEYED BY 節は、次のようになります。

```
DEFINITION "Class"
{
…
PROPERTY "Attributes" { … LISTOF "Class Attribute"
KEYED BY {Package:Package, "Class Name":Name,
Name:* } … }
```

上記の例では、Package:Package、“Class Name”:Name、および Name:* の 3 つの*キー・コンポーネント*がコンマで区切って指定されています。これらのコンポーネントは、参照先の Class Attribute 定義を識別するために必要な 3 つの部分 (パッケージ名、クラス名、およびクラス属性名) を参照しています。これらを逆の順序で見ていくと、次のことが分かります。

- クラス属性の名前は***この***プロパティーに指定されているため (*は「ここ」を意味します)、
  Name:*
- になります。「クラス属性」定義内のキー・プロパティー "Class Name"の値は、このオブジェクトの名前に含まれているため、次のようになります。
  “Class Name”:Name
- 「クラス属性」定義内のキー・プロパティー「パッケージ」の値は、このオブジェクトの「パッケージ」プロパティーに含まれているため、次のようになります。
  Package:Package

以下の概略ダイアグラムは、上記の例での KEYED BY 節の使用方法を示しており、KEYED BY 節の概要を理解するために役立ちます。



この図は、上で述べたことを示しています。つまり、クラスの定義では、クラス属性はそのパッケージ (クラス属性のパッケージ・プロパティーに保管され、ユーザーが属しているクラスのパッケージ値から取得されます)、そのクラス名 (クラス属性の“Class Name”プロパティーに保管され、そのクラスの実際の名前から取得されます)、および名前 (クラス属性の“Name”プロパティーに保管され、そのプロパティーから取得されます) を指定することによって入力されます。

要約すると、次のようになります。

1. **参照オブジェクト**の各キー・コンポーネントについて、KEYED BY 節でコンポーネントが指定されます。
2. KEYED BY 節のコンポーネントは、コンマで区切られます。
3. 各コンポーネントは 2 つの部分からなります。
   - 最初の部分は参照オブジェクトのキー・コンポーネントを示しています。

- 2 番目の部分は、そのコンポーネントの値がどこにあるのかを示し、
- 2 つの部分はコロンで区切られています。

ただし、特定のデフォルト値を採用して KEYED BY 節を単純化することができます。コンポーネントの 2 つの部分が同じである場合、2 番目の部分は省略でき、最後のコンポーネントの 2 番目の部分を省略すると、「ここ」(つまりアスタリスク) を示すものと想定されます。したがって、Class の“Attributes”プロパティーの KEYED BY 節は、実際には次のようにコーディングできます。

```
KEYED BY {Package, "Class Name":Name, Name }
```

もちろん、KEYED BY ステートメントで使用されるすべてのプロパティーが存在している必要があります。これにより、Rational System Architect は、“Package” プロパティーと “Class Name”プロパティーが “Class Attribute” 定義に存在すること、およびそれらの両方が KEY であることを確認します。

このような LISTOF プロパティー内の共通キー・コンポーネント (例えば、"Order System".Customer.email) を入力するエンド・ユーザーの手間を省けるほかに、共通値を提供するその他のプロパティーを指定して KEYED BY 節を使用すると、**各参照で確実に同じ値が使用されるようにすることができます**。 したがって、これまで使用してきた例で、クラスの“Attributes”プロパティーで参照されるすべてのクラス属性は、同じパッケージ内の同じクラスに属することが強制されています (このクラスでは望ましい特性です)。

明瞭化および単純化のために、参照されるオブジェクトのキー・コンポーネントを区別したほうが便利な場合もあります。そのような状況では、別個のコンポーネントを提供してプロパティーを指定するために KEYED BY 節が使用されます。 実際にこのような理由から、プロパティーが KEY であり、KEY プロパティーが指定されたオブジェクトを参照している場合、Rational System Architect は、コンポーネントが異なるプロパティーであることを**必要とします**。

Key および Keyed By の例

別の定義によってキー設定されている定義

ここでは、自動車をその「ブランド (Brand)」と「モデル (Model)」によって分類するとします。"Car Brand" という新しい定義 ("Car Brand" には Ford、Volkswagen、トヨタなどがあります) と、"Car Model" という別の新しい定義 (これには、Mustang、Passat、カローラなどの値が含まれています) を作成します。

"Car Model" には、その "Car Brand" (言い換えれば、その「製造元 (Make)」) が指定されている必要があります。"Car Brand" は "Car Model" を一意に識別するために使用できるプロパティーです。それぞれの "Car Model" は、"Car Brand" を 1 つだけ持ちます。

ここでは、"Make" を "Car Model" のキー・プロパティーとします (ただし、実際にこのプロパティーに入力されるのは、定義タイプである "Car Brand" です)。また、このプロパティーを REQUIRED とします。つまり、この定義を作成するには、定義の作成を開始するダイアログで、このプロパティーに入力する必要があります。

```
RENAME DEFINITION "User 1" To "Car Brand"
RENAME DEFINITION "User 2" To "Car Model"

DEFINITION "Car Brand"
{
PROPERTY "Country of Origin"
{ EDIT  Text  LENGTH 20  }
}

DEFINITION "Car Model"
{
Property "Make"
{KEY EDIT ONEOF "Car Brand" REQUIRED}
}
```

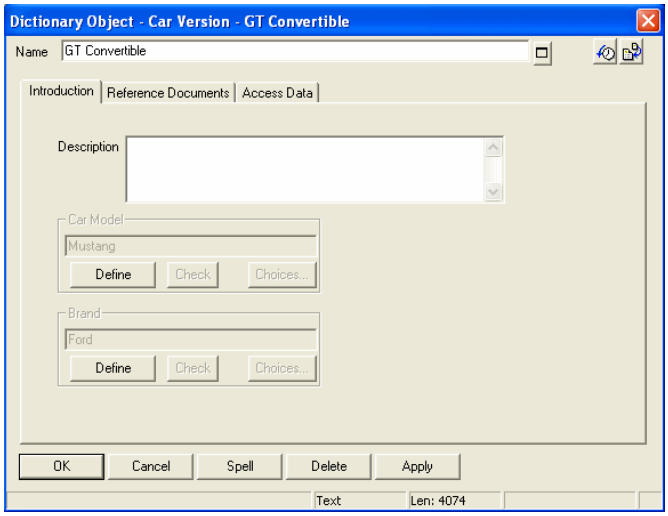
注: 上記の USRPROPS.TXT には、説明の目的で、ある問題が含まれています。それについては、このセクションのあとの部分で明らかにします。

2 つのキー・プロパティーを持つ第 3 の定義

では、さらに一歩先に進めてみましょう。それぞれの "Car Model" には、バージョンがあります。例えば、Mustang Coupe、Convertible、GT Coupe、GT Convertible、Mach 1、または SVT Cobra を購入できます。これは絶えず変化するリストとなる可能性があるので、(静的な LIST でなく) 定義

タイプにします。この定義タイプを "Car Version" と呼びます。ユーザーは、"Car Version" を作成するとき、そのバージョンの "Car Brand" と "Car Model" を指定する必要があります (GT に多数の "Car Model" が存在する可能性があるため)。

```
RENAME DEFINITION “User 3” TO “Car Version”
Definition "Car Version"
{
Property "Car Model"
{KEY Edit ONEOF "Car Model" RELATE BY "is keyed by"}
Property "Brand"
{KEY EDIT ONEOF "Car Brand" RELATE BY "is keyed by"}
}
```



キー設定定義の ListOf の作成

"Car Model" ごとに、それが備えている「Car Version」のリストを作成する必要があります。ここでは LIST OF プロパティーを作成し、ユーザーが「Car Version」を "Car Model" 定義内で入力できるようにします。「Car Versions」は、複合キーを持つ定義タイプであることに注意してください。ユーザーは "Car Version" を入力するとき、"Car Version" と "Car Brand"、および "Car Version" の "Car Model" を指定する必要があります。

```
DEFINITION "Car Model"
{
Property "Make"
{KEY EDIT ONEOF "Car Brand" RELATE BY "is keyed by"
REQUIRED}
Property "Versions" {EDIT LISTOF "Car Version"}
}
```

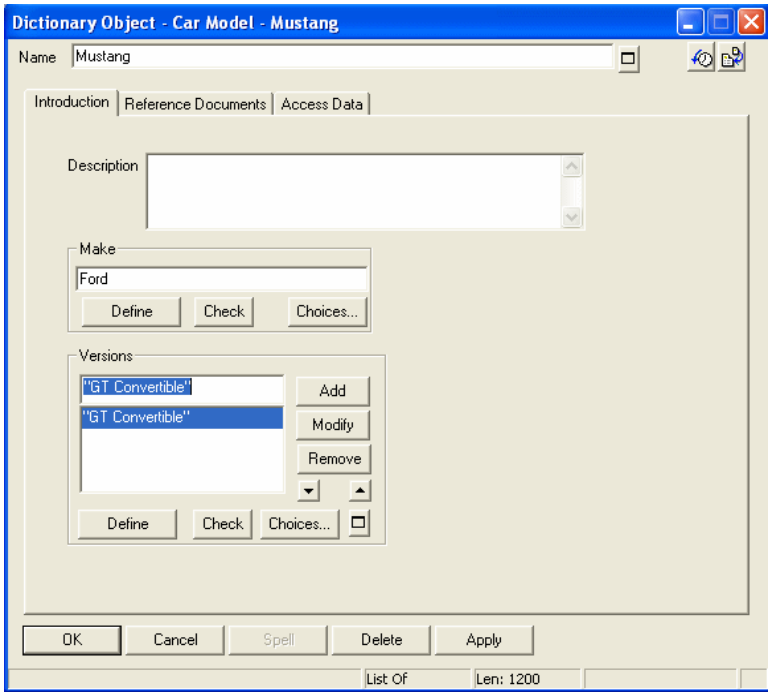
上記の USRPROPS.TXT には、**問題**があります。"Car Version" は複合キー定義です。これは、それ自体の名前をキーとして持っており、そのほかに 2 つのキー・プロパティー、つまり "Car Brand" と "Car Model" を備えています。上記の USPROPS.TXT を指定した場合、そのことを知っているかどうかは、ユーザー次第です。ユーザーは "Car Version" をその「ブランド」と「モデル」によって完全修飾し、それらをピリオドで区切って、Ford.Mustang."GT Convertible" のように入力する必要があります。

KEYED BY ステートメントの追加

ここでは、KEYED BY ステートメントを「プロパティー」"バージョン (Version)" ステートメントに追加して、それが自動的に完全なスペルで入力されるようにします。

```
DEFINITION "Car Model"
{
Property "Make"
{KEY EDIT ONEOF "Car Brand" RELATE BY "is keyed by"
REQUIRED}
Property "Versions" {EDIT LISTOF "Car Version" KEYED BY
{"Brand":"Make", "Car Model":Name, Name} }
}
```

上記の KEYED BY ステートメントの最初の部分、**KEYED BY { "Brand":"Make"** では、入力している "Car Version" 値に「Brand」と呼ばれるキー・プロパティーがあり、それを入力する必要があることを述べています。このプロパティーには、現行の定義 ("Car Model") の "Make" プロパティーから取得された値が入力されます。実際の値は、タイプ "Car Brand" の定義であることに注意してください。KEYED BY ステートメントは、定義タイプでなく、プロパティー名をリストします。



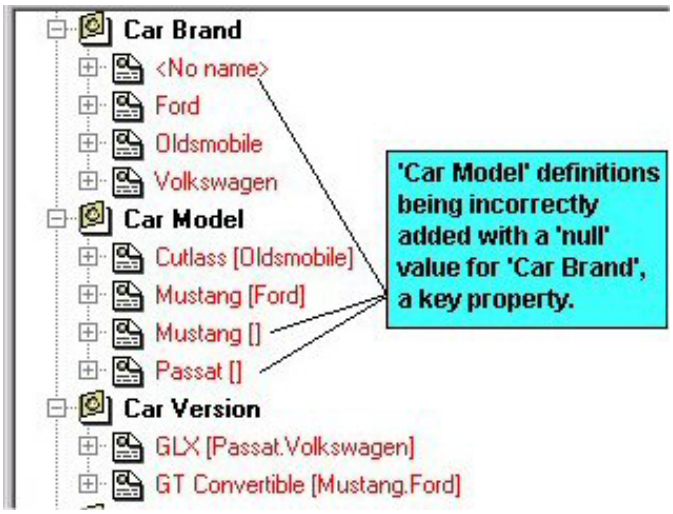
上記の参照されるプロパティー ("Brand") と、参照するキー・プロパティー ("Make") が同じものならば、両方を指定する必要はありません (つまり、それらがどちらも "Brand" であれば、KEYED BY **{ "Brand"** , ... と入力するだけで済みます。

上記の KEYED BY ステートメントの 2 番目の部分、 **"Car Model" :Name** では、入力している "Car Version" 値に "Car Model" と呼ばれる 2 番目のキー・プロパティーがあり、それを入力する必要があることを述べています。このプロパティーには、現行の定義の「Name」(開いているこの "Car Model" 定義の名前) によって取得された値が入力されます。

上記の KEYED BY ステートメントの 3 番目の部分 (最後の部分) の「Name」では、入力している "Car Version" 値の最後のキーが、すべての定義と同様に、それ自体の名前によってキー設定されることを指定しています。

したがって、Mustang という "Car Model" 定義を開いており、そのキー・プロパティー "Make" に Ford が入力されている場合、LIST OF "Car Version" プロパティーに単に GT と入力するだけで済み、新しい定義の Ford.Mustang.GT がエンサイクロペディアに追加されます。

問題は、まだあります。エンサイクロペディアに新しい "Car Version" 定義を追加するときに、"Car Brand" に Null プロパティーを持つ "Car Model" 定義が入力されることに気付きます。



この問題は、新しい "Car Version" 定義を (エクスプローラーの「新しい定義」コマンドを介して) 直接追加するときだけに発生し、"Car Model" 定義の ListOf ダイアログで追加するときには発生しません。

その理由は、"Car Version" 定義が、そのキー・プロパティーの 1 つに "Car Model" があることを指定していながら、そのプロパティーに、それ自体のキー・プロパティー "Car Brand" (これには値が入力されている必要があります) があることを指定していないためです。新しい "Car Version" 定義を追加するたびに、"Car Model" プロパティーを指定するかどうかを尋ねられます。ここでは "Car Model" プロパティーを指定していますが、その キー "Car Brand" プロパティー値の取得元は指定していません。したがって、それには何も入力されません。

これを修正するには、"Car Version" 定義の中で、その "Car Model" キー・プロパティー自体が複数のプロパティーによってキー設定されており、それらには値が入力されていなければならないことを指定する必要があります。ここでは、**KEYED BY {"Make":"Brand", Name}** 節を追加します。これは、"Car Model" 定義に "Make" というプロパティーがあり、このプロパ

ティーには、現行定義の "Brand" プロパティー内の値が入力されることを意味します。

```
Definition "Car Version"
{
Property "Car Model"
{Key Edit oneOf "Car Model" KEYED BY {"Make":"Brand", Name} RELATE BY "is keyed by"}
Property "Brand"
{KEY EDIT OneOf "Car Brand" RELATE BY "is keyed by"}
}
```

この変更を加えると、エンサイクロペディアに新しい "Car Brand" を追加したとき、偶発的に Null の "Car Brand" 定義が発生することがなくなります。

"Complete (完全)" キーワードの例

ユーザーに "Car Model" とは独立に "Car Version" の新しい定義を入力させたいと考えるかどうかは、議論の分かれるところです。例えば、"Car Version" として Si と入力した後、それがホンダのアコードを指していることを指定する人がいるでしょうか。おそらく、いません。ユーザーに "Car Brand" と "Car Model" を強制的に入力させてから、"Car Model" 内の「Car Version」の ListOf プロパティー内で「Car Version」を指定させた方が、ユーザーにとって役立つかもしれません。伝えるべきことは、これらの "Car Version" が "Car Model" に完全に属しているということです。Si があるということを伝えても、それだけではあまり意味がありません。このような場合には COMPLETE 節を使用します。

```
DEFINITION "Car Model"
{
Property "Make"
{KEY EDIT ONEOF "Car Brand" RELATE BY "is keyed by" REQUIRED}
REM "also contains a list of the versions"
Property "Versions"
{EDIT COMPLETE LISTOF "Car Version" KEYED BY {"Brand":"Make", "Car Model":Name, Name} }
}
```

リストに COMPLETE を設定すると、新しい定義を作成するときに、使用可能な定義タイプの中に "Car Version" が表示されないだけでなく、エクスプローラーから "Car Version" 定義をいっさい開くことができなくなります。開こうとすると、その定義は、その定義を含んでいる定義 (この場合は

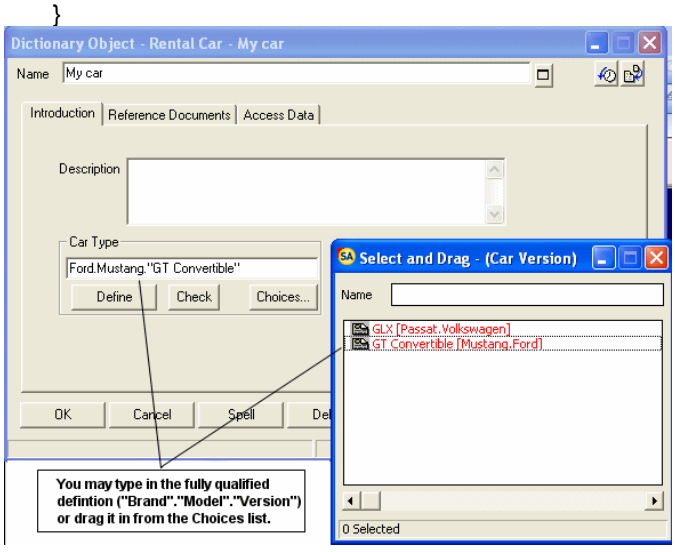
"Car Model") からのみ開くことができるというメッセージを Rational System Architect は表示します。

QUALIFIABLE の例

ここで、「自動車のレンタル (Car Rental)」を追跡するために、エンサイクロペディアに新しい定義を追加します。新しい定義タイプの「Car Rental」は、各レンタルの「自動車のタイプ (Car Type)」を追跡するためのプロパティーを含んでいます。

問題は、「レンタル・カー (Rental Car)」の場合、"Car Version" だけでなく "Car Brand" と "Car Model" の追跡も必要ないという点です。必要なのは「Car Type」という 1 つのプロパティーで、これに車種を入力し、製造元やモデルは、その車種に応じて決まるようにします。このため、「Rental Car」に「Car Type」を指定した場合、「Rental Car」定義内に自動車のブランドやモデルを保持するプロパティーはありません。この場合、QUALIFIABLE キーワードを使用します。

```
RENAME DEFINITION “User 5” TO “Rental Car”
Definition "Rental Car"
{
Property "Car Type"
{EDIT ONEOF "Car Version" KEYED BY { "Brand"
QUALIFIABLE, "Car Model" QUALIFIABLE, Name }
}
}
```



QUALIFIABLE 句を使用すると、ONEOF “Car Type”プロパティーに "Brand" と "Model" の情報が格納されます。この情報は、ピリオドで区切られて、値自体の中に保管されます。値は、「選択」ボタンを押すと開かれる「選択とドラッグ」ダイアログからドラッグするか、適切なピリオドを付けて入力することができます。

Where 節の使用例

ここでは、"Car Version" が自動車のカテゴリーに収まることを指定します。それは SUV、軽自動車 (Sub-Compact)、小型乗用車 (Compact)、中型セダン (Midsize Sedan)、大型セダン (Fullsize Sedan)、高級車 (Luxury Sedan)、トラック (Truck) のいずれでもかまいません。このリストは、かなり継続性のあるものなので、これに新しい定義タイプを作成する必要はありません。ここでは、"車両タイプ (Vehicle Types)" の「リスト」を作成します。

```
LIST "Vehicle Types"
{
Value "SUV"
Value "Sub-Compact"
Value "Compact"
Value "Midsize Sedan"
Value "Fullsize Sedan"
Value "Luxury Sedan"
Value “Convertible"
Value "Truck"
}

Definition "Car Version"
{
Property "Car Model"
{Key Edit oneOf "Car Model" KEYED BY {"Make":"Brand",
Name}  RELATE BY "is keyed by"}
Property "Brand"
{KEY EDIT OneOf "Car Brand" RELATE BY "is keyed by"}
Property "Vehicle Type"
{EDIT Text List "Vehicle Types" DEFAULT "Midsize
Sedan"}
}
```

Where 節の続き

タイプ "SUV Ad Campaign" の新しい定義を作成します。この定義のプロパティーの中で、ユーザーが特定のタイプの自動車のインスタンスを選択できるようにします。つまり、このプロパティーをフィルターに掛け、エンサイクロペディア内で、指定された条件 "Vehicle Types" = "SUV" を満たす定義インスタンスだけが含まれるようにします。このフィルター操作を提供するために、ここでは "Where" 節を使用します。

```
Definition "SUV Ad Campaign"
{
Property "SUV Type"
{ Edit OneOf "Vehicle Types" WHERE "Vehicle
Types" = "SUV" }
}
```

SAPROPS.CFG ファイル内の標準エントリーの非表示

USRPROPS.TXT 内のプロパティーを非表示にする、つまり見えないようにすることができます。既にエンサイクロペディアに情報が入力されているプロパティーを非表示にした場合に何が起きるかについて、疑問が生じる場合が少なくありません。例えば、エンサイクロペディアにデータ要素を追加し、「*事業単位*」プロパティーで、各要素に責任を負う業務領域を指定したとします。しかし、プロジェクトの途中で、「*事業単位*」が必要なくなったと決定されました。その場合には、USRPROPS.TXT を変更して、プロパティー「*事業単位*」を *invisible*、すなわち *hidden* プロパティーにするだけで済みます。このプロパティーは、データ要素定義ダイアログに表示されなくなります。

```
DEFINITION "Data Element"
{
PROPERTY "Business Unit"
{ INVISIBLE }
}
```

「*事業単位*」値を入力していたとき、それらの値はエンサイクロペディアに追加され、ENTITY.DBT というファイルに保存されていました。エンサイクロペディアの (上記の) メタモデルを変更すると、「事業単位」プロパティーはデータ要素定義ダイアログに表示されなくなりますが、以前の「事業単位」情報の項目が削除されるわけではありません。それらは、依然として .DBT ファイルの中に存在します。上記のコードを USRPROPS.TXT から除去すると、再び「*事業単位*」プロパティーがデータ要素定義ダイアログに表示され、以前に入力された値がそれぞれのデータ要素内に再表示されます。

ここで疑問が湧きます。どのようにすれば、余分で不必要なプロパティー情報を ENTITY.DBT ファイルから除去できるのでしょうか。それは、「**辞書**」メニューの下にある「**定義のエクスポート**」コマンドと「**定義のインポート**」コマンドを注意深く使用すれば行うことができます。

1. 「辞書」、「定義のエクスポート」を選択します。データ要素定義を CSV 形式でテキ

スト・ファイルにエクスポートすることを選択します。

2. その csv テキスト・ファイルを Excel などの外部エディターで開き、不要な情報が入っている列 (上記の例では、「事業単位」というタイトルが付いています) を削除します。
3. 「辞書」、「定義のインポート」を選択します。「*全フィールドを削除後新データを追加*」オプションを使用して、CSV ファイルを再インポートします。

このタスクの実行はオプションです。このタスクは、余分な値によって使用されていたメモリー・スペースを再利用したい場合にだけ必要です。

エラー・メッセージ

エラー・メッセージ

Rational System Architect は、エンサイクロペディアを開いて、その SAPROPS.CFG ファイルと USRPROPS.TXT を構文解析したとき、常にそのファイル内のステートメントに対する構文検査を行います。構文エラーがあれば、「**エラー**」ダイアログに表示されます。この「**エラー**」ダイアログには、以下のエラー・メッセージが表示される可能性があります。不等号括弧は、Rational System Architect が、例えばプロパティー名や行番号などの可変情報を挿入するポイントを示します。

< > が USRPROPS.TXT の行 < > で検出されました
< > が複数回定義されています
< > は、既にリストとして定義されています
DLL (STATBAR.DLL) をロードできません。
DLL (STATBOX.DLL) をロードできません。
章 < > は既に定義されています。
辞書クラス < > は既に定義されています。
説明
辞書
不正な引数 < > です
不正な引数 < > です。引用符で囲む必要があります。
BOOLEAN 編集に対する不正な既定値 < > です
日付編集に対する不正な既定値 < > です
数値編集に対する不正な既定値 < > です
時刻編集に対する不正な既定値 < > です
数値範囲編集に対する不正な既定値 < > です
ダイアログ \n%s をロードするためのリソースが不十分です。
無効な辞書クラス名: < >。
無効なメジャー・タイプ名:< >。
無効なリレーション名: < >。
リスト < > は既に定義されています。
リスト名 < > が定義されていません
名前 < > は既に使用されています
プロパティー編集 (OneOf、ListOf、ExpressionOf) の数が以下の < > で限度を超えています
プロパティーの数が < > の < > で限度を超えています
DISPLAY されたプロパティーの数が < > の < > で限度を超えています

リストの数が < > の < > で限度を超えています
リストの数が < > on < > で限度を超えています (最大数 =100)
数値引数 < > が範囲外です
数値引数が必要ですが、< > は以下のとおりでした
範囲外または無効な < > の長さ引数です
< > の後ファイルが途中で終わっています
前に定義済みのリスト名
プロパティー < > は既に定義されています
参照先リスト < > が定義されていません。
< > の行 <行番号> での構文エラー。
< > 編集タイプは「説明」プロパティーに対してのみ有効です
リストが多すぎます。[2]
プロパティー < > が多すぎます。[3]
リスト < > に値が多すぎます。[4]
プロパティー・ファイルを開くことができません
開始/終了 ({ }) が対になっていません
予想外のコマンド <>
不明なプロパティー DISPLAY タイプ < >
不明な辞書名 < >
不明な編集タイプ < >
不明な初期タイプ < >
不明な更新タイプ < >
警告 - RANGE が検出されましたが、最大範囲が定義されていません。
警告 - RANGE が検出されましたが、最小範囲が定義されていません。

---

[2] リストの最大数は 400 です。これには、SAPROPS と USRPROPS が含まれ、SAPROPS から実際に使用されるリストの数はエンサイクロペディアの構成によって異なります。

[3] 1 つの「ダイアグラム」、「シンボル」、または「定義」に対するプロパティーの最大数は 128 です。

[4] LIST 内の VALUE の最大数は 128 です。

エラー・メッセージのほかに、Rational System Architect はエラー・ダイアログ内に、検出した構文エラーに関する以下のような詳しい情報を配置します。

DEFINITION コマンドの検査時。
DISPLAY コマンドの検査時。
LIST コマンドの検査時。
LIST 内の **VALUE** コマンドの検査時。
DEFINITION 内の **PROPERTY** コマンドの検査時。
DEFINITION または **LIST** コマンドの有無の検査時。
続けますか?

例えば、エラー・メッセージ全体は、以下のようになります。

不明なプロパティー DISPLAY タイプ < > (**DEFINITION** コマンドの検査時)。

ランタイム編集

「ランタイム」とは、ダイアグラムを描画しているとき、特に、エンサイクロペディア項目を作成しているときを指します。辞書を追加または変更するときに表示されるダイアログは、SAPROPS.CFG および USRPROPS.TXT の制御下にあります。**EDIT** コマンドは、ユーザーがエラーのある項目を作成しないよう機能します。例えば、SAPROPS.CFG に以下のような項目があるとします。

```
PROPERTY “My Property”
{ EDIT numeric LENGTH 2 MINIMUM 1 MAXIMUM 32 }
```

この例では、"My Property" テキスト・フィールドに AB または 0 を入力すると、「**無効な値**」のエラー・メッセージが表示され、「OK」をクリックするとダイアログが閉じます。このようになるのは、このプロパティーが 1 を最小値とする数値 (英字から構成することはできません) として指定されているためです。

Rational System Architect は、以下のランタイム編集を実行します。

BOOLEAN	T、F、TRUE、FALSE のいずれかでなければならない
DATE	数値、MM/DD/YY フォーマット
式、ExpressionOf、ListOf、OneOf	2-72 ページから始まるセクションの項目を参照。
NUMERIC	数値ストリングでなければならない
TEXT	編集なし
TIME	数値、HH:MM:SS フォーマット

3

USRPROPS.TXT キーワード

はじめに

この章には、USRPROPS.TXT への変更に使用できるすべてのキーワードをアルファベット順に並べたリストが含まれています。

CHAPTER、GROUP、LABEL、LIST、および LISTONLY の各キーワードの使用については、一定の制限が適用されます。キーワードの使用に適用される具体的な制限については、各キーワードの説明を参照してください。

USRPROPS キーワード

$$FORCE$$ DISPLAY キーワードを参照してください。

$$NONE$$ DISPLAY キーワードを参照してください。

$$VFORCE$$ DISPLAY キーワードを参照してください。

$$VNONE$$ DISPLAY キーワードを参照してください。

#IFDEF IFDEF コマンドの後の引用符で囲まれた節が「プロパティー構成」ダイアログでオンにされているかどうかに基づいて、USRPROPS.TXT 内のコマンドをオンにすることができます。「プロパティー構成」ダイアログ (「ツール」、「メソッド・サポートのカスタマイズ」、「エンサイクロペディアの構成」と選択) は、エンサイクロペディアの sadeclar.cfg ファイルを変更します。 SAPROPS.CFG の解析で IFDEF 文を評価する際に実際に検査されるのは、sadeclar.cfg ファイルです。

このコマンドには、対応する終わりの #endif 文が必要です。

例:

```
#ifdef "Business Enterprise"
DEFINITION "ORGUNIT"
{
LAYOUT { COLS 2 ALIGN OVER }
PROPERTY "RowDefinition"
{ KEY EDIT OneOf "Organizational Unit" RELATE BY "is part of"
ReadOnly LABEL "Organizational Unit"}

PROPERTY "ColumnDefinition"
{ KEY EDIT OneOf "Organizational Unit" RELATE BY "is part of"
ReadOnly LABEL "Organizational Unit"}

PROPERTY "Description"
{ EDIT Text LENGTH 255 HELP "Appears in the cell of a matrix" }

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

#IFDEF (続き)

上の例で、定義タイプ “ORGUNIT”には、「**プロパティー構成**」ダイアログ (「ツール」、「メソッド・サポートのカスタマイズ」、「エンサイクロペディアの構成」) で Business Enterprise がオンになっている場合に指定されるプロパティーのみが含まれています。 これらのプロパティーに値を入力し、Business Enterprise をオフにした場合、それらの値はリポジトリー内の“ORGUNIT”の定義には残りますが、プロパティー・セットがオフになっているため、定義ダイアログには表示されません。

#IFNDEF キーワードも参照してください。

#IFNDEF

IFDEF コマンドとは逆に、IFNDEF コマンドでは、このコマンドの後の引用符で囲まれたプロパティー・リストが「プロパティー構成」ダイアログでオンになって**いない**場合に、USRPROPS.TXT 内のコマンドをオンにすることができます。このコマンドには、対応する終わりの #endif 文が必要です。

例:

```
#ifndef "Business Enterprise"
RENAME Symbol "Swim Lane" to "Org. Unit"
#endif
```

上の例では、「System Architect プロパティー構成」ダイアログ (「ツール」、「メソッド・サポートのカスタマイズ」、「エンサイクロペディアの構成」と選択) で “Enterprise Architecture”選択項目がオンになっていない場合に、すべての ‘Swim Lane’シンボルが“Org. Unit”に名前変更されます。 この変更は、これらのシンボルが使用されているダイアグラムでこれらのシンボルを選択する際に分かります。 構成ダイアログの“Enterprise Architecture”選択項目は、以前は“Business Enterprise”という名前でしたが、V9.0 のダイアログで変更されました。 しかし、その項目が SADECLAR.CFG で呼び出す内在的な switch 文は現在でも “Business Enterprise”と呼ばれます。

#INCLUDE

#include を使用すると、USRPROPS.TXT ファイル内で変更内容を別の追加ファイルに分割することができます。 これらの各ファイル内に別の include を置き、さらにその中に別の include を置く (以下同様) ことができます。パーサーが許可しているネスティングのレベルは 10 です。それを超えた場合、Rational System Architect は警告を出し、超えた分のレベルは無視します。

例:

例えば、3 つの USRPROPS.TXT ファイル (ダイアグラム用に 1 つ (例えば diagrams.txt)、定義用に 1 つ (例えば definitions.txt)、シンボル用に 1 つ (例えば symbols.txt)) を作成することができます。 この USRPROPS.TXT ファイルは以下のようになります。

```
**************************************************************************
REM "USRPROPS.TXT"
REM "Copyright IBM.  All rights reserved."
REM "Instructions for modifying this file are in the on-line help."

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

これらの各ファイル内には、リスト用のファイル (例えば lists.txt) など、他のファイルに対する #include を置くことができます。

このコマンドを使用すると、ユーザー定義データを分かりやすく再利用できるようになります。

ADDRESSABLE

Rational System Architect では、1 つ以上の (関連付け可能な) 定義によってシンボルを*関連付ける*ことができます。 関連付け可能な 13 個の定義が自動的に用意され、これらは任意のシンボルを対象にできます。*その 15 個とは、ビジネス目的、ビジネス・プロセス、変更要求、重要な成功要因、現在のデータ・コレクション、データ・クラス、成果物、機能的組織、地理的位置関係、情報要件、組織の目標、要求、 テスト計画です。* これらの関連付け可能な定義は、ダイアグラム上のシンボルを選択して、「辞書」、「関連付け」と選択することによって使用できます。また、すべての定義は、USRPROPS.TXT の構文を使用して、関連付け可能として宣言することができます。 そうすると、その定義がシンボルの関連付けで使用できるようになり、「辞書」、「関連付け」のドロップダウン・リストに表示されるようになります。

例:

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

キーワード NONADDR も参照してください。

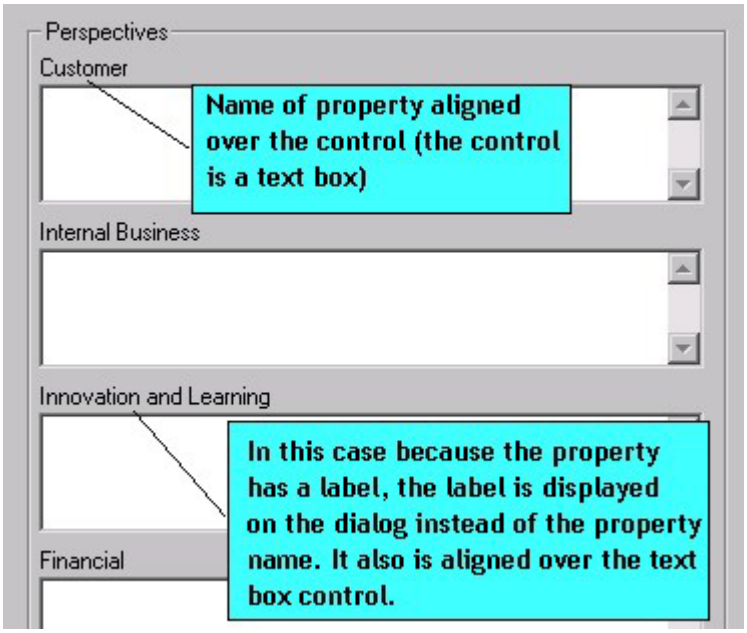
ALIGN

ダイアログ内にあるプロパティーのコントロール (リスト・ボックス、テキスト・ボックスなど) の名前 (またはラベル) の位置を指定する場合に使用します。 有効なオプションは、*BODY*、*LABEL*、および *OVER* です。

例:

```
DEFINITION "Balanced Scorecard"
{..
GROUP "Perspectives"
{
LAYOUT { COLS 2 TAB ALIGN OVER }
PROPERTY "Customer" { EDIT Text LENGTH 300 }
PROPERTY "Internal Business" { EDIT Text LENGTH 500 }
PROPERTY "Learning" { EDIT Text LENGTH 500 LABEL “Innovation and Learning”}
PROPERTY "Financial" { EDIT Text LENGTH 500 }
}
```

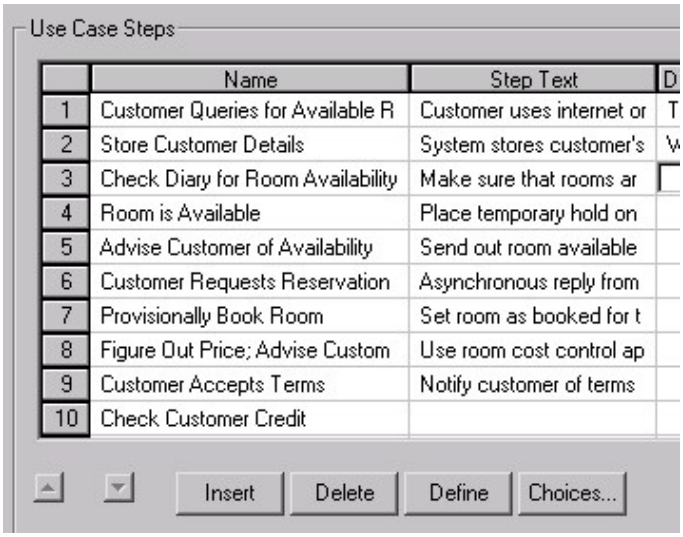
上の例では、ALIGN OVER コマンドによって、LAYOUT 文以下のすべてのプロパティーの名前またはラベルが、「Balanced Scoreboard」定義ダイアログ内のそれぞれのコントロールの上に配置されます。



キーワード BODY、LABEL、OVER、JUSTIFY、および TAB も参照してください。

ASGRID

ASGRID コマンドは、テーブル (つまりグリッド) に ListOf プロパティーがあることを示します。 ASGRID には、ListOf または ParmListOf のいずれかの編集タイプがなければなりません。 これらがない場合、Rational System Architect は、エンサイクロペディアの再オープン時に警告を出し、ASGRID キーワードを無視します。



例:

```
Definition "Use Case"
{CHAPTER "Steps"
PROPERTY "Use Case Steps" { EDIT COMPLETE ListOf "Use Case
Step" KEYED BY { "Package", "Use Case Name":Name, Name}
ASGRID LENGTH 1200 } }
```

キー付き定義での ASGRID の使用

定義のキー・プロパティーは、ASGRID コマンドによって作成されるグリッドには表示されません。 上の例では、各 Use Case Step のパッケージ名またはユースケース名はグリッドに表示されません。

ASGRID の制限

他の定義の COMPLETE ListOf にある定義を参照する LISTOF で ASGRID を使用することはできません。 したがって、例えば定義に ListOf "Attribute" を追加することはできますが、これを ASGRID 表示することはできません。 グリッドに表示可能なプロパティーの最大長は 400 です。

ASGRID (続き)

ASGRID_COUNT_FIXED

COUNT_FIXED キーワードは、ユーザーがグリッドで行の削除または挿入を行えないことを示すために、ASGRID キーワードとともに使用します。

KEY、KEYED BY、および COUNT_FIXED の各キーワードも参照してください。

ASGUID

このキーワードは、テキスト・プロパティーでのみ使用できます。 これは、プロパティーに自動的に “GUID”プロパティーの値を設定します。その後このテキスト・プロパティーは、実際の GUID プロパティーの代わりにキー・プロパティーとして使用することができます。ASGUID プロパティーは読み取り専用である必要があります。定義を再オープンすると、ASGUID プロパティーに入力が行われます。

例:

```
RENAME DEFINITION “User 1” to “MyDef”
DEFINITION “MyDef”
{
PROPERTY “MyProp”
{KEY EDIT Text LENGTH 100 ASUID READONLY}
Property “HIYA”
{EDIT Text Length 145}
}
```

ASPARMGRID

ASPARMGRID キーワードは特に Rational System Architect のデータ・モデリングで使用するために作成されたもので、特別に作成されたコードを元に動作します。このキーワードは SAPROPS.CFG にあります。USRPROPS.TXT のユーザーは使用**しないで**ください。

ASSIGN

新規または既存のダイアグラム・タイプに、新規のシンボル・タイプまたは既存のシンボル・タイプ (他のダイアグラムに既に存在するシンボル) を割り当てることができます。 シンボル・タイプは、以下の構文を使用してダイアグラム・タイプに追加することができます。

SYMBOL <symbol-type-name> [IN <diagram-type-name1>]
ASSIGN [TO] <diagram-type-name2>

例:

SYMBOL “Organizatiional Unit” IN “Organization Chart”
{
ASSIGN TO “Enterprise Direction”
}

AUDITID

このキーワードは、ユーザーが最初に Rational System Architect にサインオンする際に「**監査 Id**」ダイアログで入力する文字を表します。AUDITID は、あるプロパティーがユーザーの監査 ID を含んでいることを示す許容キーワード・タイプです。CHECKOUT AUDITID、FREEZE AUDITID、INITIAL AUDITID、および UPDATE AUDITID のそれぞれが、特別の意味を持っています。詳しくは、このテーブルに個別にリストされたそれぞれのキーワードを参照してください。

注: Rational System Architect のバージョン 9.1 以降、すべての定義の「アクセス・データ」タブに、INITIAL AUDITID (「作成時監査」というフィールドの中) および UPDATE AUDITID (「最終変更時監査」というフィールドの中) が自動的に組み込まれるようになりました。

例:

```
DIAGRAM "Data Flow Gane & Sarson"
 {
   PROPERTY "Frozen by"
 { FREEZE AUDITID }
```

「監査 ID」の他の使用例としては、定義での使用が考えられます。

例:

```
DEFINITION "X"
 {
   PROPERTY "Current Owner Name"
 { EDIT Text CHECKOUT AUDITID LENGTH 12 READONLY }
 }
```

AUTOCREATE

AUTOCREATE コマンドは、プロパティーに指定された値の背後にある定義を、そのプロパティーを含むダイアログを「OK」ボタンをクリックして閉じた直後に自動的に作成します。 AUTOCREATE キーワードを使用しない場合、プロパティーに入力された値は、そのプロパティーを含むダイアログを「OK」ボタンをクリックして閉じた後でも未定義のままです。

例:

```
DEFINITION "Physical Database"
{..
PROPERTY "Model"
{ KEY EDIT ONEOF "Project Data Model" AUTOCREATE RELATE BY
"is keyed by" READONLY }
..}
```

上の例で、Physical Database 定義の定義ダイアログには、“Project Data Model”というプロパティーが含まれています。このフィールドに値 (例えば、“Reservations”) を入力し、「OK」をクリックして Physical Database の定義ダイアログを閉じると、エンサイクロペディアに “Reservations” Project Data Model の定義が自動的に作成されます。

INITIAL USER REQUIRED および OVERRIDABLE も参照してください。

BEGIN

このキーワードは、プロパティーの定義の開始、あるいはダイアグラム、シンボル、または定義の定義を構成する一連のプロパティーの定義の開始を示します。次の構文を使用することもできます: **{**。

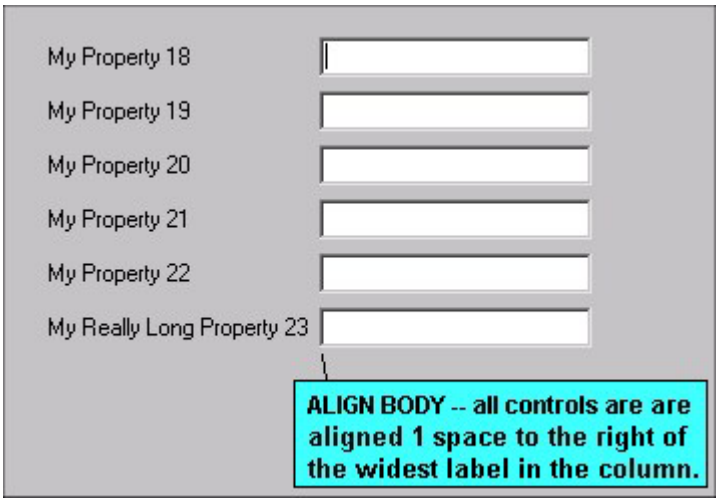
キーワード PROPERTY も参照してください。

BODY

これは、**ALIGN** コマンドで使用する引数の 1 つです。これは、すべてのコントロールを、その列の最も長いラベルの右にスペースが 1 つ空くように桁揃えするために使用されます。(これに対して、ALIGN と OVER のキーワード・ペアでは、名前がプロパティーの上に配置されます。)

例:

```
Definition "My Definiition"
{
CHAPTER “My Chapter”
LAYOUT { COLS 1 ALIGN BODY }
PROPERTY "My Property 18"{ EDIT Text Length 10}
PROPERTY "My Property 19"{ EDIT Text Length 10}
PROPERTY "My Property 20"{ EDIT Text Length 10}
PROPERTY "My Property 21"{ EDIT Text Length 10}
PROPERTY "My Property 22"{ EDIT Text Length 10}
PROPERTY "My Really Long Property 23"{ EDIT Text Length 10}
}
```



上の例では、“My Really Long Property 23”に対するコントロールが、ラベルの右側からスペース 1 つ分の位置に置かれるテキスト・ボックスです。このダイアログ上の他のプロパティーに対する他のすべてのテキスト・ボックス・コントロールは、このコントロールの位置に合わせられます。

注 – 以前、ALIGN BODY は、すべてのコントロールをラベルの右側からスペース 1 つ分の位置に置くために使用していましたが、その後、ALIGN LABEL と同じ動作に変更されました。

キーワード OVER、ALIGN、TAB、LABEL、および JUSTIFY も参照してください。

BOOLEAN

このプロパティーは「**定義**」ダイアログにチェック・ボックスとして表示されます。これには、真 (T) と偽 (F) の 2 つの値のいずれかが指定されます。

例:

以下の例で、ユーザーは真または偽を選択することによって、IDEF0 図で Hierarchical Numbering フィーチャーをオンまたはオフにすることができます。

```
DIAGRAM "IDEF0"
 {
   PROPERTY "Hierarchical Numbering"
   { EDIT Boolean LENGTH 1 DEFAULT "F" }
..
}
```

BROWSER (「エクスプローラー」)

「エクスプローラー」で個々のダイアグラム、シンボル、または定義を選択した際に、Rational System Architect の「エクスプローラー(ブラウザー)」内の「プロパティー」ボックスにプロパティーおよびその値を表示するかどうかを指定します。

以下のエクスプローラー制御ステートメントを使用できます (「オブジェクト」という語は、ダイアグラム、シンボル、または定義を意味します)。

プロパティーの指定内:

- **BROWSER {SHOW}:** そのプロパティーを含むオブジェクトを表示する際に、そのプロパティーの値を表示するように「エクスプローラー」に要求します。

オブジェクトの指定内 (ただし、プロパティーの「説明」以外):

- **BROWSER {OMITKEY}:** 本来は表示する条件にあるオブジェクトのキー・プロパティーを表示しないように「エクスプローラー」に要求します。
- **BROWSER {OMITTYPE}:** 本来は表示する条件にあるオブジェクトのタイプを表示しないように「エクスプローラー」に要求します。

オブジェクトの指定外:

- **BROWSER {OMITKEY}:** 本来は表示する条件にあるどのオブジェクトのキー・プロパティーも表示しないように「エクスプローラー」に要求します。
- **BROWSER {OMITTYPE}:** 本来は表示する条件にあるどのオブジェクトのタイプも表示しないように「エクスプローラー」に要求します。

「本来は表示する条件にある」という表現が使用されているのは、「エクスプローラー」に一部 (またはすべて) のキー・プロパティーが表示されない (オブジェクトがそのキー・オブジェクトの 1 つに従属する形で表示される場合) や、タイプが表示されない (タイプがタイプ・ヘッダーに従属する形で表示される場合) ことがよくあるためです。

BROWSER (「エクスプローラー」) (続き)

例 1:

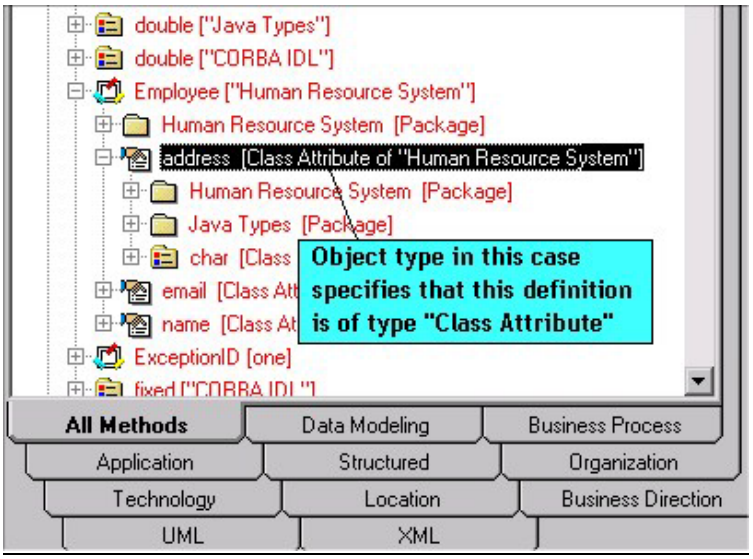
```
DEFINITION "Association  End"
{
PROPERTY "Package" { KEY EDIT OneOf "Package" RELATE BY "is keyed by" READONLY  BROWSER { SHOW }}
..}
```

上の例の場合、通常は表示されないパッケージ・プロパティーの値が「エクスプローラー」に表示されます。

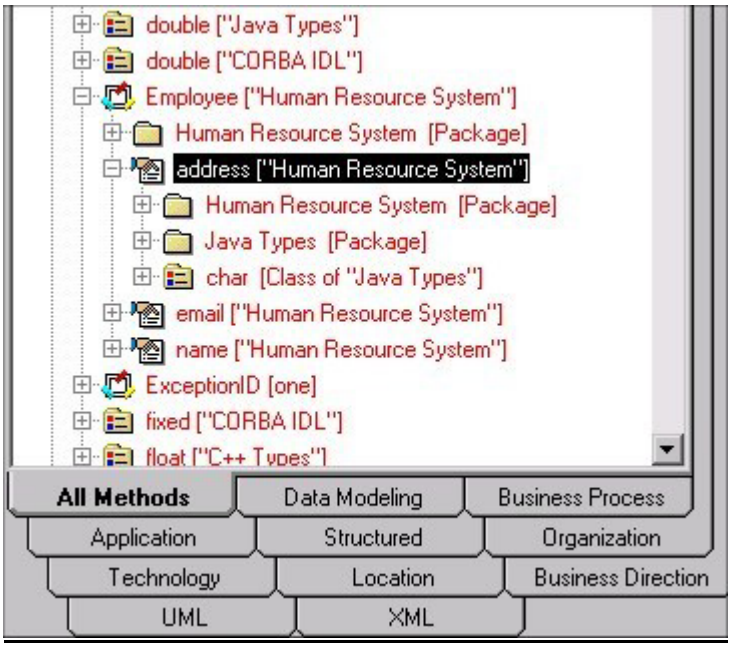
例 2:

```
DEFINITION "Class Attribute"
{
BROWSER { OMITTYPE }
..}
```

上の例で、クラス属性は “Class Attribute”タイプの定義です。これはデフォルトでは「エクスプローラー」に表示されますが、やや冗長である上に、視覚的にも邪魔であると判断されます。 BROWSER (OMITTYPE) コマンドを使用しなかった場合、「エクスプローラー」は属性を以下の図のように表示します。



BROWSER {OMITTYPE} コマンドを使用した場合、「エクスプローラー」は属性を以下の図のように表示します。



例 3:

```
DEFINITION "Association"
{
BROWSER { OMITKEY }
LAYOUT { COLS 1 ALIGN OVER   TAB }
CHAPTER "Roles"
PROPERTY "Association GUID" { KEY EDIT Text LENGTH 64
INVISIBLE READONLY}
PROPERTY "Class Roles" { EDIT COMPLETE ListOf "Association
End" KEYED BY { "Association GUID":"Association GUID",
"Association":"Name", "Package" QUALIFIABLE, "Class"
QUALIFIABLE, "Role GUID" QUALIFIABLE, "Name" }
 RELATE BY "uses" LENGTH 4096 ASGRID COUNT_FIXED
BROWSER { SHOW } }
..
}
```

上の例の場合、通常は表示されない “Class Roles”プロパティーの値が「エクスプローラー」に表示されます (関連のアタッチ先のクラスは、知っておくべき重要な情報であるためです)。

BY

DEFINED BY、*RELATED BY*、*RELATE BY*、および *KEYED BY の各式で頻繁に使用されるキーワードです。* 詳しくは、個々のキーワードの組み合わせを参照してください。

例:

```
DEFINITION "Column"
{..
PROPERTY "Database Name"
{ KEY EDIT OneOf "Database" RELATE BY "nothing" }
  ..
}
```

CHAPTER

ダイアログで***タブ***を作成します。章の各ステートメントはタブに対応しています。構文は以下のとおりです。

CHAPTER <chapter_name>

章のステートメントは、タブ内の項目をグループ化するのに左大括弧も右大括弧も必要としません。章のステートメントの下にあるすべての項目は、そのタブにグループ化されます。次のグループ化は、次の章のステートメントによって作成されます。

例:

CHAPTER "Screen Painter properties"

タブ (章) の名前の変更:

USRPROPS.TXT を使用して CHAPTER の名前を変更するには、LABEL コマンドを使用します。

例:

SAPROPS ファイルによって、クラス定義用の「ネストしたクラス」タブが提供されます:

```
DEFINITION "Class"
 { CHAPTER "Nested Classes"
  ...
 }
```

You may relabel the CHAPTER “Nested Classes” to “Fred” using the LABEL command in USRPROPS.TXT:

```
DEFINITION "Class"
 { CHAPTER "Nested Classes" LABEL "Fred"
    }
```

CHECKOUT

チェックアウトしたユーザーの監査 ID や、チェックアウトされた日付または時刻など、オブジェクトのチェックアウトに関する情報を表示します。 表示されるフィールドは常に読み取り専用です。値は Rational System Architect によって自動的に追跡されますが、ダイアログで値を表示するためには、以下のそれぞれの特性のプロパティーを追加する必要があります。

CHECKOUT Auditid
CHECKOUT Date
CHECKOUT Time

例:

```
DIAGRAM "Data Flow Gane & Sarson"
 {
   PROPERTY "Checked out by"
 { CHECKOUT AUDITID }
   PROPERTY "CheckOut Date"
 { CHECKOUT DATE }
   PROPERTY "CheckOut Time"
 { CHECKOUT TIME }
 }
```

オンライン・ヘルプの *アクセス制御* で、オブジェクトのチェックインおよびチェックアウトに関する詳しい情報を検索してください。

キーワード FREEZE も参照してください。.

COLS、列

「ダイアグラム」、「シンボル」、または「定義」の各ダイアログ内のプロパティー・グループが配置される列の数を決定します。

例:

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

COLUMN_SCRIPT

COLUMN_SCRIPT は、SA Basic で作成されたスクリプトを呼び出します。列スクリプトは、 物理データ・モデルのテーブル内の列の振る舞いに対して使用されます。スクリプトによるアクションの実行対象は、リスト内の各列です。

慣例的に、関数自体の名前には以下のいずれかの接頭部が付けられます。

- fmtxxx - この関数自体はハードコーディングされており、変更することはできません。SAPROPS.CFG の関数の多くはこの形式です。 関数をハードコーディングするのは、Rational System Architect の全体的な応答速度を上げるためです。
- _fmtxxx - Rational System Architect のメイン実行可能ディレクトリーにある fmtscript.bas ファイルの中に存在します。

独自のスクリプトの作成

独自のスクリプトの作成方法については、SCRIPT キーワードを参照してください。

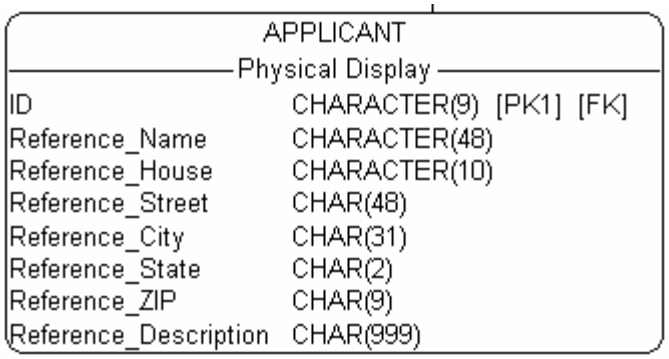
例:

```
DEFINITION "Table"
{
PROPERTY "Description"
{
ZOOMABLE EDIT ListOf Definition "Column"  FROM "Data Element"
KEYED BY {"Database Name","Owner Name","Table
Name":"Name","Name"} LENGTH 2000
DISPLAY { FORMAT Key LEGEND "Key Data" }
DISPLAY { FORMAT NonKey LEGEND "Non-Key Data" }
DISPLAY { FORMAT COLUMN_SCRIPT FmtERAttr LEGEND
"Physical Display" }
}  ..}
```

FmtERAttr は、エンティティー・リレーション図のエンティティーの属性の値、または物理図にあるテーブルの列の属性の値を返します。

FmtERAttr は、ID、NAME、ADDRESS、STREET、CITY、STATE、FIRST_5_DIGITS、ZIP CODE、および LAST_4_DIGITS を返します。

COLUMN_SCRIPT (続き)



SCRIPT、COMPONENT_SCRIPT、VALUESCRIPT、および FORMAT の各キーワードも参照してください。

COMPLETE

参照される側の定義が、参照する側の定義に属するようにします。そのため、参照される側の定義を別の定義から参照することはできません。また、参照される側の定義は、それを収容する参照する側の定義内からのみ編集することができます。

一例は、エンティティー内の属性です。この属性は、完全にエンティティーに属し (別の定義には属しません)、そのエンティティーの定義内からのみ開くことができます (例えば、属性の定義を Rational System Architect の「エクスプローラー」で直接開くことはできません)。

例:

```
DEFINITION "Entity"
{
PROPERTY "Attributes"
  {ZOOMABLE EDIT COMPLETE ListOf "Class Attribute" KEYED BY
{"Class Name":"Name", Name} ASGRID LENGTH 4096 DISPLAY
{ FORMAT List } }
..
}
```

COMPONENT_ SCRIPT

Basic で作成された関数を、SA Basic と呼ばれるものに含まれている Rational System Architect への関数呼び出しを使用して呼び出します。コンポーネント・スクリプトは、ListOf および ExpressionOf のリストに対して使用します。 スクリプトによるアクションの実行対象は、リスト内の各項目です。 例えば、構文

COMPONENT_SCRIPT *fmtomtattr* では、すべての属性および

それらの対応する C ストレージ・タイプがコロン (:) 区切りの形式で返されます。

慣例的に、関数自体の名前には以下のいずれかの接頭部が付けられます。

- fmtxxx - この関数自体はハードコーディングされており、変更することはできません。SAPROPS.CFG の関数の多くはこの形式です。 関数をハードコーディングするのは、Rational System Architect の全体的な応答速度を上げるためです。
- _fmtxxx - Rational System Architect のメイン実行可能ディレクトリーにある fmtscript.bas ファイルの中に存在します。

独自のスクリプトの作成

独自のスクリプトの作成方法については、SCRIPT キーワードを参照してください。

既存スクリプトの説明:

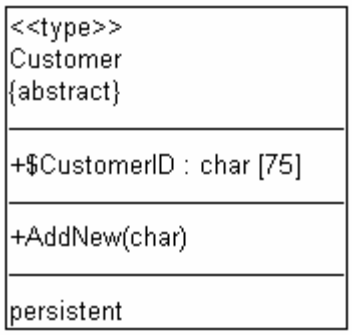
fmtUMLAttr は、すべての属性およびそれらの対応するタイプをコロン区切りの形式で返します。

fmtOMTOperation は、すべての操作およびそれらの対応する C ストレージ・タイプを括弧で囲んで (type) 返します。

FmtOMTObjInstAttr は、オブジェクトがインスタンス化されるクラスのすべての属性を返します。

FmtOMTActivity は、スクリプト **do** を返し、また、状態定義にリストされたすべてのアクティビティーのアクティビティー名を返します。

FmtOMTStateActions は、状態定義にリストされたすべての内部アクションの内部アクション名を返します。



例 (*fmtomattr* を使用):

```
Definition “Class” {
   PROPERTY "Attributes"
  { PROPERTY "Attributes" {ZOOMABLE EDIT  COMPLETE ListOf
"Class Attribute" KEYED BY {"Package", "Class Name":"Name",
Name } LENGTH 4096 ASGRID DISPLAY { FORMAT
COMPONENT_SCRIPT _FmtNewUMLAttr LEGEND "$$FORCE$$"}
LABEL "Attributes" }
 }
```

CONTROL

Control キーワードは、定義内でスイッチをセッ トアップするために TESTPROC とともに使用した場合 、Property キーワードと同等になります。

スイッチの値に応じて定義ダイアログにプロパティーが表示されるようにする方法は 2 とおりあります。「エンサイクロペディアの構成」ダイアログでエンサイクロペディアに対して設定した値を対象とした #ifdef を使用することができます (例えば、エンサイクロペディアの言語タイプを Java または C++ に設定するなど)。「エンサイクロペディアの構成」ダイアログが実際に値を設定するのは、sadeclar.cfg ファイルです。

また、それ自体は定義ダイアログ内のプロパティー (TESTPROPERTY) であるスイッチに基づいて PROPERTY がダイアログに表示されること (およびその初期値) を指定することもできます。 例えば、エンティティー内で、その DBMS タイプが Oracle または SQL Server であることを指定するとします。DBMS タイプとして設定した値に基づいて、以降のプロパティーが定義に表示されたり、表示されなかったりします。また、この値に基づいて、特定のデフォルト値が指定されます。TESTPROPERTY スイッチの指定には、TESTPROC キーワードを使用します。 PROPERTY キーワードは特定のプロパティーを最 初に定義に指定する際に使用し、定義内でそのプロパ ティーを使用する他のすべての個所には CONTROL キーワードを使用します。 REFPROP キーワードは、各 CONTROL が参照する PROPERTY を指定するために使用します。 このため、Control および RefProp キーワードは TESTPROC とともに使用されることがよくあります。

要約すると、CONTROL を使用するためには、その

CONTROL が参照する PROPERTY への初期参照が、定義の先頭になければならないということです。 REFPROP キーワードは、CONTROL キーワードとともに使用されます。

例:

```
Definition "Index"
{
CHAPTER "Modeling Properties"
{ TESTPROC TestPropertyNotValue TESTPROPERTY "DBMS"
TESTSTRING { "ORACLE 8" } }
PROPERTY "Primary Key" {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 "Modeling Properties "
  { TESTPROC TestPropertyValue TESTPROPERTY "DBMS"
TESTSTRING { "ORACLE 8" } }
CONTROL "Primary Key" { REFPROP "Primary Key" }
CONTROL “Unique” { REFPROP "Unique" }
CONTROL “Clustered” {REFPROP "Clustered"}
..
  }
```

上の例では、DBMS として Oracle 8 が選択された場合に、Index 定義に対して “Primary Key”、“Unique”、および “Clustered”の各プロパティーを指定するため、REFPROP キーワードが CONTROL キーワードとともに使用されています (これらのプロパティーは、それぞれの参照されるプロパティーとまったく同じです)。

COPY PROPERTIES FROM

このコマンドを使用すると、他の定義タイプから現行の定義タイプにプロパティーをコピーすることができます。 これにより、同種の概念を単一の定義タイプにまとめることができます。 これは定義にのみ適用されます。 構文は以下のとおりです。

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

例:

DEFINITION "Elephant"
{
…
CHAPTER "Properties copied from Change Request"
COPY PROPERTIES FROM "Change Request"
CHAPTER "Properties copied from Dependency and Node"
COPY PROPERTIES FROM "Dependency", "Node"
…
}

コピーは、入力の COPY 文に処理が及んだ時点で実行されます。上の例で、プロパティー・ファイルの Change Request、Dependency、または Node に後でプロパティーが追加されたり、プロパティー・ファイルの既存のプロパティーが後で変更されたりした場合に、それらの追加や変更がコピーされることは**ありません**。

COPYSCRIPT

このキーワードは、定義のコピーを実行する際に、特定のプロパティーに対して SABasic スクリプトを呼び出すように指定する場合に使用します。

独自のスクリプトの作成

独自のスクリプトの作成方法については、SCRIPT キーワードを参照してください。

例:

```
DEFINITION "Entity"
{
CHAPTER "Attributes"
PROPERTY "Description"
{ EDIT COMPLETELISTOF "Attribute" FROM "Data" KEYED BY
{Model, "Entity Name":"Name", "Name"} RELATE BY "uses" ASGRID
COPYSCRIPT OnCopyEntityDesc EDITCLASS
SACPropertyAttributeGrid
..}
```

COUNT_FIXED

COUNT_FIXED キーワードは、ユーザーがグリッドで行の削除または挿入を行えないことを示すために、ASGRID キーワードとともに使用します。 行の数が固定されます。

例:

```
DEFINITION "Association"
{
BROWSER { OMITKEY }
LAYOUT { COLS 1 ALIGN OVER   TAB }
CHAPTER "Roles"
PROPERTY "Association GUID" { KEY EDIT Text LENGTH 64
INVISIBLE READONLY}
PROPERTY "Class Roles" { EDIT COMPLETE ListOf "Association
End" KEYED BY { "Association GUID":"Association GUID",
"Association":"Name", "Package" QUALIFIABLE, "Class"
QUALIFIABLE, "Role GUID" QUALIFIABLE, "Name" }
RELATE BY "uses" LENGTH 4096 ASGRID COUNT_FIXED
BROWSER { SHOW } }
```

上の例で、クラス間の関連には、その関連の接続先 のクラスごとにグリッド内に 1 行ずつが割り当てられ ます (通常は 2 行ですが、その関連ラインに追加のクラスが接続された場合には、3 行以上になる場合もあります (この動作はソフトウェア内でハードコーディングされます))。COUNT_FIXED キーワードがあるために、ユーザーはグリッド内で行の追加または削除を行えません。

ユーザーがグリッドでステップの新規追加または削除を行える他のグリッド (例えば「ユースケース・ステップ」グリッド) とこのグリッドを対比してください。

DATA

これはキーワードでは***ありません***。これは、 ONEOF、LISTOF、および EXPRESSIONOF の各コマンドの引数として使用される特殊語で、 Rational System Architect のデータ辞書を構成するデータ要素とデータ構造への参照を提供します。

DATE

このキーワードは編集タイプで、 長さを 10 にする必要があります。グラフィック・ディスプレイは、 Windows に設定された日付形式に基づきます。DATE は、 プロパティーに Windows に定義された時刻形式に適した表記の日付スタンプが含まれることを示す、 正当なフィールド・タイプでもあります。

CHECKOUT DATE、FREEZE DATE、 INITIAL DATE、および UPDATE DATE のそれぞれに特別な意味があります。

例 1:

```
DIAGRAM "Data Flow Gane & Sarson"
 {
   PROPERTY "Freeze date"
 { FREEZE DATE }
```

DATE のその他の使用法が定義に存在する場合もあります。

例 2:

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

DEASSIGN

キーワード DEASSIGN は、ダイアグラム・タイプからシンボルを除去する際に使用されます。

例:

```
SYMBOL "Message Flow" in "Business Process"
{
DEASSIGN from "Business Process"
}
```

DEFAULT

Rational System Architect によってプロパティーに割り当てられる値で、 ユーザーはオーバーライドすることができます。グラフィック画面では、 既定値が最初にテキスト・ボックスに表示されたり、 チェック・ボックスを最初はオンにするか、オフにするかを決定したりします。

例:

```
PROPERTY "Not a table"
 { EDIT Boolean LENGTH 1 DEFAULT F }
```

DEFINED BY

このキーワードは、定義をシンボルに関連付けるものです。また、シンボルを別の定義に再度関連付けることもできます。

意味 1: USRPROPS.TXT のエンサイクロペディアに新しいシンボルを追加する場合、 このキーワードを使用して、関連付ける定義タイプを指定する必要があります。 USRPROPS.TXT に指定した新しいシンボルにこの節がない場合、 Rational System Architect はエンサイクロペディアを開くときに構文解析の警告を出し、 そのシンボルの定義を既定値の NULL にします。

例 1:

```
RENAME DIAGRAM "User 1" to "My Diagram"
RENAME SYMBOL "User 1" to "Direction"
RENAME DEFINITION "User 1" to " Direction"

SYMBOL "Direction"
{
DEFINED BY " Direction"
ASSIGN TO "My Diagram"
}
```

上記の例では、 新規ダイアグラム・タイプ、シンボル・タイプ、および定義タイプが USRPROPS.TXT に指定されています。 DEFINED BY キーワードを使用して、シンボル“Direction”が “Direction”定義によって定義されるように指定します。さらに、このシンボルを “My Diagram”ダイアグラムに割り当てます。(注: 新規定義“Direction”に定義文を指定することもできますが、 必須ではありません。指定しなければ、新規定義には、 デフォルト・プロパティーの “Name”および“Description”のみが付けられます。)

意味 2: DEFINED BY キーワードでは、 SAPROPS.CFG での指定とは異なる定義によってシンボルを定義することもできます。 このキーワードを使用してシンボルを別の定義に再度関連付ける際には、 参照するシンボルが表現されているダイアグラムを必ず指定してください (例えば、“Class”のシンボル“Class”と、“Component”のシンボル“Class”の場合、 最初のケースは、 クラス図のクラス・シンボル定義を再定義することを指定し、 後のケースは、コンポーネント図のクラス・シンボルを指定します)。

例 2:

```
SYMBOL Process IN "Data Flow Gane & Sarson"
{
DEFINED BY "Control Transform"
}
```

上記の例では、“Data Flow Gane & Sarson”ダイアグラムの Process シンボルが “Control Transform”によって定義されています。通常は、“Process”によって定義されます。

DEFINITION

このキーワードは、DIAGRAM または SYMBOL とは対照的に、 DEFINITION のプロパティーがリストされるブロックの最初の語です。

例:

```
DEFINITION "Data Element"
 {
   PROPERTY "Length"
 { EDIT number LENGTH 2 }
 .
 .
 .
 }
```

キーワード DIAGRAM および SYMBOL も参照してください。

DEFINITION REFERENCED IN

「OF DEFINITION REFERENCED IN」を参照してください。

DEPICT LIKE

DEPICT LIKE キーワードの組み合わせを使用して、 ダイアグラムにどのようにシンボルを描写するかを指定します。 新しいシンボルを作成するとき、およびダイアグラムでの表示方法を指定するときに、 このキーワードの組み合わせを使用できます。シンボルが別のダイアグラムのシンボルと同じように表示されるように指定できます。

DEPICT LIKE キーワードの組み合わせは、「ノード」シンボルと「ライン」シンボルで使用できます。

例 (ノード・シンボル):

```
SYMBOL "Communications Connection"
{
ASSIGN TO "OV-01 Highlevel Op. Concept"
..
DEPICT LIKE "Event Flow" IN "Data Flow Ward & Mellor"
```

例 (ライン・シンボル):

```
SYMBOL "Need Line"
{
PROPERTY "From Operational Node"
{EDIT ONEOF "Operational Node" READONLY INVISIBLE}
PROPERTY "To Operational Node"
{EDIT ONEOF "Operational Node" READONLY INVISIBLE}
DEPICT LIKE "Transition" IN "OMT State"
DEFINED BY "Need Line"
ASSIGN TO "OV-02 Op. Node Connectivity"
}
```

DEPICTIONS

ユーザーが提供するイメージ・ファイルによってどのようにシンボルを表現できるかを特定します。シンボルはビットマップまたはメタファイルで描写できます。 DIAGRAM キーワードと組み合わせて DEPICTIONS キーワードを使用することで、 シンボルをダイアグラムのワークスペース上に描写する方法を指定できます。 また、 MENU キーワードと組み合わせて DEPICTIONS キーワードを使用することで、 ツールボックスと 「描画」メニューにシンボルを描写する方法を指定することもできます。構文は以下のとおりです。

SYMBOL <symbol-type-name>

{ ...
DEPICTIONS { DIAGRAM <depiction-file> }
DEPICTIONS { MENU <depiction-file> }
...}

ここでの <depiction-file> は、ビットマップまたはメタファイルの名前とフルパスです。

例:

Rename Symbol "User 3" To "Radar
SYMBOL "Radar"
{ASSIGN To "Wireless Network"
DEPICTIONS { DIAGRAM "C:\Program Files\IBM\pictures\radar.bmp" }
DEPICTIONS { MENU "C:\Program Files\IBM\pictures\radartoolbar.bmp" }}

DEPICTIONS キーワードはリスト内でも使用できます。 これにより、選択したリストの値に基づいて、シンボルをさまざまな方法で描写することができます。

例:

List "Class Stereotypes"

{

Value "actor"**DEPICTIONS** {DIAGRAM images\slctact.wmf MENU images\slctact.bmp}

Value "boundary"**DEPICTIONS** { DIAGRAM images\slctbndy.wmf MENU images\slctbndy.bmp}

..}

DEFINITION “Class” {

PROPERTY "Stereotype" { EDIT Text LIST "Class Stereotypes" INIT_FROM_SYMBOL Default "" LENGTH 20 } ..}

DIAGRAM

DIAGRAM コマンドは 2 つの方法で使用します。

ダイアグラム・プロパティーの指定:

DIAGRAM コマンドは、DEFINITION または SYMBOL とは対照的に、 ダイアグラムのプロパティーがリストされるブロックの最初の語として使用されます。

例:

```
DIAGRAM "Booch Class"
 { PROPERTY "DGX File Name"
   { EDIT Text LENGTH 255 }
  PROPERTY "Notes"
   { EDIT Text LENGTH 4000 }
    }
```

キーワード DEFINITION および SYMBOL も参照してください。

DEPICTIONS コマンドでの使用:

「描画」ツールバーまたはメニューの場合と比べて、ここではダイアグラムのワークスペース上のシンボルを表すために使用されるグラフィックを参照しています。

例:

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

DISPLAY

ダイアグラム・シンボル上にプロパティーとその値を表示できるようにします。1 つの定義につき、display ステートメントは 37 個までに制限されます。

構文は以下のとおりです。

DISPLAY { FORMAT [STRING | LIST | KEY | NONKEY | COMPONENT_SCRIPT | COLUMN_SCRIPT | SCRIPT]
LEGEND " (シンボル内でのブロックのラベル表示方法) **" }**

以下の FORMAT キーワードのいずれかを指定するように選択できます。

STRING: プロパティーの値を入力されたとおりにシンボルに表示します。例については、STRING キーワードを参照してください。

LIST: リスト内のシンボル上に項目を表示します。それぞれの空白文字は、 二重引用符で囲まれていなければ、復帰改行となります。詳しくは、LIST キーワードを参照してください。

KEY: このキーワードは、キーとして指定されたプロパティーに使用します。これらのキーはシンボルの別のセクションに表示されます。 例および詳細については、 KEY キーワードを参照してください。

NONKEY: このキーワードは非キー・プロパティーに使用できます。これらはシンボルの別のセクションに表示されます。このキーワードは、当初は Rational System Architect のデータ・モデリング・サポートでの エンティティーとテーブルに対して使用されていました。例については、NONKEY キーワードを参照してください。

COLUMN_SCRIPT: COLUMN_SCRIPT キーワードを参照してください。

COMPONENT_SCRIPT: COMPONENT_SCRIPT キーワードを参照してください。

SCRIPT: SCRIPT キーワードを参照してください。

DISPLAY (続き)

LEGEND キーワードの後に、 シンボル内のブロックのラベル表示方法を引用符で囲んで指定します。 選択項目は以下のとおりです。

LEGEND “<Your Text>”: 引用符内にどのようなテキストを配置した場合も、該当項目に値がある場合にのみ、引用符で囲まれたそのテキストは項目の上方のシンボル上に表示されます。

LEGEND "": 説明なしの直線を表示します。 ただし、その項目に値がある場合に限られます。

**LEGEND "$$FORCE$$":**シンボル上の項目の上に横のラインを表示します。このラインは、分割線として機能します。 “$$FORCE$$”キーワードは、 単に“ “ を使用する場合とは異なり、プロパティー表示が表示モード・ダイアログによって抑止されていても、 横のラインを表示します。

LEGEND “$$NONE$$”: 該当項目の値が存在するかどうかにかかわらず、シンボル内の項目の上方に横線を表示しません。このラインは、通常、分割線として機能します。

LEGEND “$$VFORCE$$”: シンボル内でプロパティーを左から右の方向に配置できるようにし、それらの間に縦線を引きます。VFORCE キーワードを参照してください。

LEGEND “$$VNONE$$”: プロパティーを左から右にレイアウトできますが、分割線を表示 *しません*。VNONE キーワードを参照してください。

例:

```
DEFINITION "Organizational Entity"
  { PROPERTY "Incumbent Name"
    { EDIT Text LENGTH 100  HELP "Name of person
          currently in position"
  DISPLAY { FORMAT String LEGEND "" } }
```

EDIT

キーワード BEGIN (または {) と一緒に使用して、プロパティーの定義の開始を示します。キーワード EDIT は、「これが開始引数である」という意味を持ちます。

例:

```
SYMBOL "Process" IN "Data Flow Gane & Sarson"
  { PROPERTY "Short Description"
    { EDIT Text LENGTH 1500 }
  PROPERTY "Number" { EDIT Numeric LENGTH 4 }
```

EDIT COMPLETE

COMPLETE キーワードを参照してください。

EDITCLASS

このキーワードは使用しないでください。 このキーワードは、 Rational System Architect の特定の状態、エンティティー内の属性による「データ要素」プロパティーの継承に対応するように作成された特別なキーワードです。 このコマンドは、SAPROPS.CFG 内で特定の状態に対して使用されています。これが、このキーワードを適用できる唯一の状態です。それ以外の状態で使用すると、エラーの原因となります。

EDIT URLS

ListOf プロパティーを外部ドキュメントを参照可能なプロパティーであるものとして指定できます。 このコマンドによって、listof プロパティーの下部に
「開く」ボタン、「外部参照」ボタン、および「内部参照」ボタンが表示されるようになります。これらのボタンを使用して、外部ドキュメントの参照および選択、外部ハイパーリンクの入力、
エンサイクロペディアのデータベースの内部 Files テーブルの参照、外部ドキュメントまたは内部参照ドキュメントのオープンを行うことができます。

構文:

PROPERTY property name { **EDIT URLS** }

制約:

SA では、URLS プロパティーに以下の制約があります。Key 指定はできません。

例:

Definition "Use Case"
{
PROPERTY "Reference Documents" { EDIT URLS }
}

END

ダイアグラム、シンボル、または 定義の定義を構成するプロパティーあるいはプロパティー・グループの指定の終了を示します。これは、BEGIN 文と組み合わせて使用して、指定範囲を表します。 BEGIN 文および END 文の代わりに、左中括弧と右中括弧 **{ }** を使用することもできます。

例:

PROPERTY “<property_name>”
  BEGIN EDIT <edit_type> <property_parameter>
  END

式

定義の値を + 符号または空白で区切られた一連のストリングとして入力する必要があることを示します。

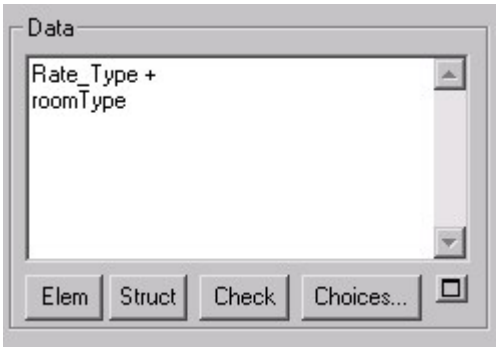
例:

```
VendorName +
VendorCity +
VendorState
```

このキーワードに代わって使用されるようになった EXPRESSIONOF キーワードも参照してください。

EXPRESSIONOF

EXPRESSIONOF を使用すると、オブジェクトへの参照を、複雑な演算子および区切り文字を使用して表現することができます。EXPRESSIONOF は通常、データ要素とデータ構造を参照する DATA という特殊な引数とともに使用されます。つまり、EXPRESSIONOF DATA の形で使用されます。このキーワードの組み合わせによって表示されるテキスト・ボックスの中に、定義の値を一連のストリングとして入力してください。定義値と定義値の間は、空白で区切ります。 規則では、個々の定義値を区切るために + 符号が使用されることになっていますが、+ 符号は必須ではありません。



例:

```
DEFINITION "Control Flag"
{
PROPERTY "Description"
{ EDIT EXPRESSIONOF DATA LENGTH 4074 LABEL "Data" }
}
```

キーワード ONEOF および LISTOF も参照してください。また、使用可能な演算子および区切り文字に関する詳細とリストについては、第 2 章の ExpressionOf のセクションを参照してください。

fmtxxx または _fmtxxx

規則により、これら 2 つの名前接頭部は、SCRIPT、COLUMN_SCRIPT、または COMPONENT_SCRIPT キーワードによって呼び出される任意の関数の名前の先頭で使用できます。関数自体 (例えば、_fmtUMLAttr) は通常、シンボルのプロパティー値 (ある属性とそのすべてのプロパティーなど) を特別なフォーマットで表示します。 命名基準は次のとおりです。

- _fmt (例えば、_fmtUMLAttr など): この関数自体は、ハードコーディングされているため、変更できません。 SAPROPS.CFG の関数の多くはこの形式です。関数をハードコーディングするのは、Rational System Architect の全体的な応答速度を上げるためです。
- fmt (例えば、fmtUMLAttr など): Rational System Architect のメイン実行可能ディレクトリー内にある、fmtscript.bas に存在します。

例:

```
DEFINITION “Class”
PROPERTY "Attributes" {ZOOMABLE EDIT  COMPLETE ListOf "Class Attribute" KEYED BY {"Package", "Class Name":"Name", Name }
LENGTH 4096 ASGRID DISPLAY { FORMAT COMPONENT_SCRIPT
_FmtNewUMLAttr LEGEND "$$FORCE$$"} LABEL "Attributes" }
```

上記の例では、クラス図の属性を特定の方法 (例えば、属性のアクセス・プロパティーが 'public' である場合にクラス・シンボルの属性の前に '+' マークを付けるなど) で表示するために使用されるスクリプトが呼び出されます。

独自の関数の作成

ユーザー独自の関数を作成する方法については、SCRIPT キーワードを参照してください。

DISPLAY、FORMAT、SCRIPT、COLUMN_SCRIPT、COMPONENT_SCRIPT、および VALUESCRIPT キーワードも参照してください。

FORCE

実際には $$FORCE$$ キーワードであり、DISPLAY キーワードとともに使用されます。

詳しくは、DISPLAY キーワードを参照してください。

FORMAT

特定の表示可能プロパティーについて、データの提示方法を指示します。

例:

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

詳しくは、DISPLAY キーワードを参照してください。

FREEZE

オブジェクトのフリーズに関する情報を、フリーズさせた人の監査 ID またはフリーズの日付または時刻を含めて表示します。 表示されるフィールドは常に読み取り専用です。値は Rational System Architect によって自動的に追跡されますが、ダイアログで値を表示するためには、以下のそれぞれの特性のプロパティーを追加する必要があります。

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

例:

```
DIAGRAM "Data Flow Gane & Sarson"
 {
    PROPERTY "Frozen by"
 { FREEZE Auditid }
    PROPERTY "Freeze Date"
 { FREEZE Date }
    PROPERTY "Freeze Time"
 { FREEZE Time }
 }
```

オブジェクトのフリーズに関する詳細については、オンライン・ヘルプの *アクセス制御* を検索してください。

CHECKOUT キーワードも参照してください。

FROM_CHOICES_ONLY

ユーザーが選択項目リストからのみ定義を選択できる (新規定義を入力できない) ように制限します。 メッセージ・ボックスが表示されて、「選択項目」リストのみから選択を行うようにユーザーに指示します。 これは ListOf と OneOf と一緒に使用します。

以下に例を示します。

```
DEFINITION "Product"
{
CHAPTER "Technical Reference Model"
PROPERTY "Status" {Zoomable EDIT Oneof "Product Status"}
Group "Involvements"{
LAYOUT { COLS 2 ALIGN OVER  }
PROPERTY "Lead Proponent" {Zoomable EDIT Oneof
"Organizational Unit" FROM_CHOICES_ONLY}
PROPERTY "Others Involved" {Zoomable EDIT Listof
"Organizational Unit" FROM_CHOICES_ONLY}
}
}
```

GROUP

特定のレイアウト・パラメーター (一連のラジオ・ボタンなど) を指定してグループ・ボックスを作成し、その中で 2 つ以上のプロパティーを指定できるようにするために使用します。

例 1:

```
GROUP "Referential Integrity"
 {
   LAYOUT { ALIGN OVER TAB COLS 3 }
     PROPERTY "Parent Delete"
 { EDIT Text LISTONLY LIST RDC
     LENGTH 15 }
 …
 }  REM "End of Group Referential Integrity"
```

Rational System Architect によって SAPROPS ファイルに既に事前定義されている GROUP 名は変更できません。

例 2:

```
DEFINITION "Class Attribute"
 { CHAPTER "Class, Source Data, Desc."
    GROUP "Source Data" {
 LAYOUT { COLS 2 ALIGN OVER TAB }PROPERTY
      "Description"
 { EDIT Text LENGTH 1500 }
```

上記の例の 3 行目で、USRPROPS.TXT ファイル内の 'GROUP "Source Data"' を 'GROUP "Original Data"' に変更しようとしても、その変更は無効になります。SAPROPS GROUP 項目に含まれているテキスト "Source Data" は指定変更されません。引き続きこれが Class Attribute Group の表示テキストとして使用されます。

ヘルプ

これは、特定のプロパティーが選択されたときに**ダイアグラム**または**定義**のダイアログの左下隅にある状況表示行に表示されるストリングです。

構文:

HELP "<text_string>"

例:

```
PROPERTY Length
 { EDIT Numeric LENGTH 2 MIN 1 MAX 99
   HELP "Length of this field"
 }
```

HETEROGENEOUS (ONEOF、LISTOF)

単一プロパティーで複数のタイプの定義を参照できるようにします。(通常のリストでは、単一タイプの定義が参照されます。) HETEROGENEOUS キーワードは、ONEOF キーワードまたは LISTOF キーワードを変更するために使用されます。

例えば、クラス・リストで「選択」ボタンをクリックすると、選択対象としてクラス定義のみが提供されます。混成リストの「選択」ボタンをクリックすると、Heterogeneous リスト節で指定したさまざまなタイプの定義 (「クラス」、「プロセス」、「エンティティー」など) が表示されます。

ONEOF の構文:

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

LISTOF の構文:

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

制限

混成リスト・プロパティーについては、いくつかの制限があります。このプロパティーは、以下のいずれかが該当するものであってはなりません (つまり、同じプロパティー内で、以下のいずれかのキーワードを HETEROGENEOUSLISTOF または HETEROGENEOUSONEOF キーワードと併用することはできません)。

- このプロパティーは KEY にすることができません。
- KEYED BY 節を含めることはできません。
- COMPLETE にすることはできません。
- FROM 節を含めることはできません。
- ASGRID にすることはできません。
- DEFAULT を含めることはできません。
- INITIAL USER REQUIRED にすることはできません。
- 制限 (REFERENCED IN または WHERE) 節を含めることはできません。
- INIT_FROM_SYMBOL 属性を含めることはできません。
- どのタイプ名も 1 回しかリストできません。

リストへの新規値の追加

ユーザーは、ほとんどの場合、「選択」ボタンをクリックすると表示される「選択とドラッグ」ブラウザーから値を混成リストにドラッグすると考えられますが、混成リストに新規の値を追加することも可能です。 ただし、混成リストに新規の値を追加する場合、その値を以下の書式の完全修飾名を使用して入力する必要があります。

ClassName:TypeName:FullyQualifiedName

場所:

- ClassName は、System Archtiect エンサイクロペディア・クラス・タイプ (Diagram、Symbol、または Definition) です。
- TypeName は、Diagram、Symbol、または Definition タイプの特定の名前 (Class (definition) または Use Case Step (definition) など) です。
- FullyQualifiedName の各部分はピリオドで区切られます。したがって、例えば、ユースケース・ステップがそのユースケースをキーとして使用し、ユースケースがそのパッケージをキーとして使用する場合、ユースケース・ステップは次のように入力されます。

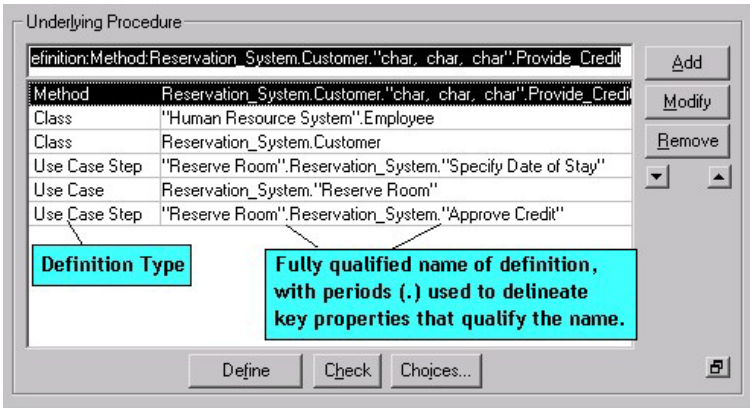
  Definition:”Use Case Step”:”Package Name”.”Use Case Name”.”Use Case Step Name”

例:

```
Definition " Procedure"
{
PROPERTY "Underlying Procedure" { EDIT
HETEROGENEOUSLISTOF " Use Case","Class", "Method", "Use Case Step" READONLY}
```

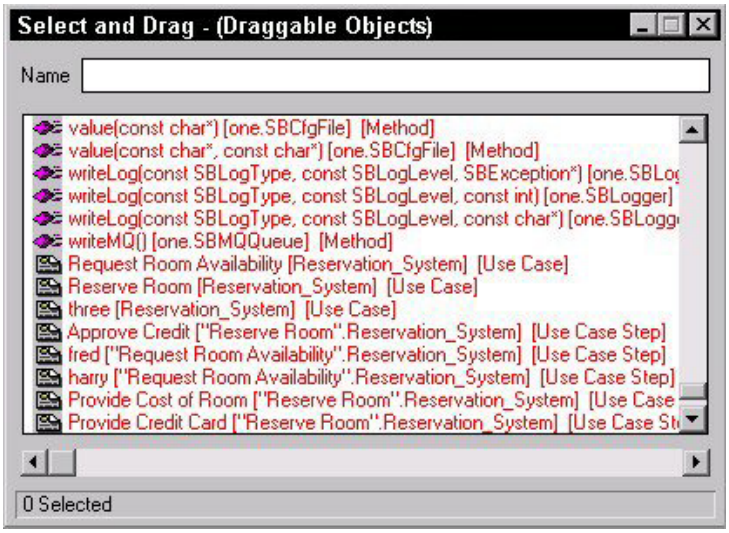
上記の例で、“Procedure”定義の “Underlying Procedure”プロパティーには、Use Case、Class、Method、および Use Case Step タイプの定義を取り込むことができます。

HETEROGENEOUSONEOF または HETEROGENEOUSLISTOF キーワードで提供されるユーザー・インターフェースは、各定義タイプの名前、およびリストにドラッグされた特定の定義の完全修飾名を含む列を表示します。

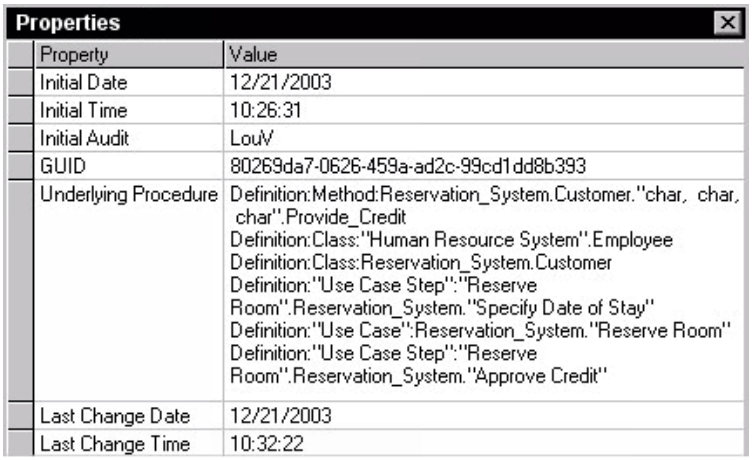


定義の名前を修飾するキー・プロパティーは、ユーザー・インターフェースで、ピリオド (.) によって相互の間を区切って表示されます。例えば、ユースケース・ステップはそれを含むユースケースをキーとして使用し、ユースケースはそれを含むパッケージをキーとして使用します。
HETEROGENEOUSONEOF フィールドまたは HETEROGENEOSLISTOF フィールドで、ユースケース・ステップは "Package Name"."Use Case Name"."Use Case Step Name" によって示されます。いずれかの項目の名前にスペースが含まれている場合、その項目は引用符で囲まれます。 例えば上記の図で、ユースケース・ステップ“Approve Credit”はユースケース Reservation_System 内にあり、 ユースケース Reservation_System はパッケージ “Reserve Room”に属しています。

混成リストの「選択」ボタンをクリックすると、呼び出し対象になるすべてのダイアグラム・タイプ、シンボル・タイプ、または定義タイプが表示され、それぞれの名前の後に大括弧で囲まれたタイプがリストされます。



「プロパティー」ウィンドウには、混成リストの値も表示されます。「プロパティー」行または列の枠をドラッグして、値全体が表示されるようにすることができます。 それぞれの値の前には、そのクラス・タイプ (ダイアグラム、シンボル、または定義)、タイプ名 (つまり、ユースケース・ステップ定義) が示され、その後に値自体が示されます。



HIDE DEFINITION

「新しい定義」および**「定義のオープン (Open Definition)」**ダイアログから対象の定義タイプを除去します。

構文:

HIDE DEFINITION <definition name>

例:

HIDE DEFINITION "SQL Server Table"

警告: 定義を非表示にするとき、特に「プロパティー構成」ダイアログ (「ツール」、「メソッド・サポートのカスタマイズ」) で選択してアクティブにしたシンボルによって使用されている定義を非表示にするときには、注意が必要です。描画しているシンボルの基礎になる定義が存在しないという状態になることがあります。

HIDE DIAGRAM

「新規ダイアグラム (Diagram New)」 および **「ダイアグラムのオープン (Diagram Open)」** ダイアログから参照対象のダイアグラム・タイプを除去します。

構文:

HIDE DIAGRAM <diagram name>

例:

HIDE DIAGRAM "Booch Process"

注: このキーワードを使用する代わりに、**「プロパティー構成」**ダイアログからダイアグラム・タイプを選択解除して、より小規模な変更を行うことができます (「ツール」、「メソッド・サポートのカスタマイズ」、「エンサイクロペディアの構成」を選択し、ダイアグラム・タイプを使用しているメソッドをオフにするか、「プロパティー構成」ダイアログの「詳細」ボタンをクリックして「選択済みダイアグラム (Selected Diagrams)」リストから「使用可能なダイアグラム」リストにダイアグラム・タイプを移動します)。

HIERARCHICAL

デフォルトではユーザー・ダイアグラムは (シンボルの) ネットワークになっていますが、ダイアグラム・タイプの記述にこのキーワードが含まれている場合、そのダイアグラム・タイプは階層ダイアグラムとして扱われます。つまり、それに割り当てられたすべてのノード・シンボルを階層内に配置することができ、その他の関連した階層機能 (階層番号付けなど) がサポートされます。

HIERARCHICAL キーワードは、ユーザー定義のダイアグラム・タイプでのみ使用することができ、既存のダイアグラム・タイプには適用できません。それ以外のコンテキストでこのキーワードを使用すると、ユーザーに対して警告が出された上で無視されます。 (ユーザー定義のダイアグラム・タイプを新規に作成する方法については RENAME DIAGRAM キーワードを参照してください。)

例:

```
RENAME DIAGRAM “User 1” to “Zoo”
RENAME SYMBOL "User 1" to "Mammals"
RENAME SYMBOL "User 2" to "Reptiles"
RENAME DEFINITION “User 1” to “Mammal”
RENAME DEFINITION “User 2” to “Reptile”

SYMBOL "Mammals"
{DEFINED by "Mammal"
ASSIGN TO "Zoo"}

SYMBOL "Reptiles"
{DEFINED by "Reptile"
ASSIGN TO "Zoo"}

DIAGRAM "Zoo"
{HIERARCHICAL
PROPERTY "Hierarchical Numbering"
{ EDIT Boolean LENGTH 1 DEFAULT "T" }
PROPERTY "First Node Number"
{ EDIT Text  Length 20 DEFAULT "1" }
}
```

上記の USRPROPS.TXT で作成されたダイアグラムは、組織図などと同じように階層性を備えています。

IFDEF

#IFDEF コマンドを参照してください。

IFNDEF

#IFNDEF コマンドを参照してください。

IN

シンボルに適用されるときに、**RENAME** コマンドのコンテキストを設定します。これは、**DEPICT LIKE** でも使用できます。

例:

例えば、Application シンボルの名前を変更するには、以下のようにします。

```
#ifdef "Business Enterprise"
RENAME SYMBOL "Application" IN "System Architecture" TO
"My Symbol"
#endif
```

あるシンボルを別のダイアグラムにあるシンボルのように見せるには、以下のようにします。

```
#ifdef "Business Enterprise"
SYMBOL "System" IN "System Context"
{
DEPICT LIKE "Process" IN "Data Flow Gane & Sarson"
}
#endif
```

INCLUDE

#INCLUDE を参照してください。

INITIAL

ダイアグラム、シンボル、または定義に、作成の際の監査 ID、日付、および時刻のスタンプを付けるために使用します。このフィールドの値は、Rational System Architect によって変更されることはありません。

可変項目:

INITIAL DATE

INITIAL TIME

INTIAL AUDITID

Rational System Architect V9 以降では、INITIAL DATE、INITIAL TIME、および INITIAL AUDITID は、各ダイアグラムや定義ダイアログの「アクセス・データ」タブにデフォルトで提供されるようになっています。これは製品にハードコーディングされています。つまり、SAPROPS.CFG 内の各定義に INITIAL キーワードは存在せず、ダイアグラム・タイプや定義タイプを新規作成する際に INITIAL キーワードを USRPROPS.TXT に追加する必要もありません。

例:

```
DEFINITION "X"
{ PROPERTY "Creation Auditid"
{ EDIT Text INITIAL AUDITID LENGTH 12 READONLY }
}
```

UPDATE キーワードも参照してください。

INIT_FROM_SYMBOL

INIT_FROM_SYMBOL キーワードは、シンボルの定義において使用します。 このキーワードは、定義内のプロパティーが最初はシンボル内の同様の名前のプロパティーから値を継承するように指定します。 これは、プロパティーがシンボルと定義の両方に存在する必要があり、その値が同じでなければならない場合に使用します。 Stereotype などのプロパティーに基づいてシンボルにユーザー提供のメタファイルを指定する場合などに必要になります。 このステレオタイプは (シンボルがダイアグラム内でどのように表示されるのかを決定するので) シンボルに指定する必要があり、また対応する定義内でも指定する必要があります。

例 1:

```
LIST "Class Stereotypes"
{
VALUE "actor" DEPICTIONS {diagram images\slctact.wmfmenu
images\slctact.bmp}
VALUE "boundary" DEPICTIONS {diagram images\slctbndy.wmfmenu
images\slctbndy.bmp}
VALUE "case worker" DEPICTIONS {diagram images\slctcwkr.wmf
menu images\slctcwkr.bmp}
}

SYMBOL "Class" in "Class"
{
PROPERTY "Stereotype" { INVISIBLE EDIT Text ListOnly List "Class
Stereotypes" DEFAULT "" LENGTH 20}..}

DEFINITION “Class”
{
PROPERTY "Stereotype" { EDIT Text LIST "Class Stereotypes"
INIT_FROM_SYMBOL Default "" LENGTH 20 } ..}
```

上記の例で、Stereotype プロパティーはクラス・シンボルの指定とクラス定義の両方で宣言されています。 このプロパティーは同じ名前になっていなければなりません。SYMBOL 内の Stereotype プロパティーによって、Rational System Architect の「描画」メニューから選択できるステレオタイプ値がドロップダウン表示されます (これらのステレオタイプ値は、LIST ステートメントの DEPICTIONS 節で指定されている bitmaps です)。「描画」ツールバーのリストからステレオタイプ・クラスを選択してダイアグラムに配置すると、そのクラスの定義が作成され、その定義のステレオタイプ・プロパティーは、シンボルに関して選択されたステレオタイプに基づいて自動的に指定されます。この値を定義内で変更すると、シンボルでも変更されます。

INIT_FROM_SYMBOL (続き)

また、このステレオタイプ値は INVISIBLE に指定されているため、クラスのシンボル・タブには表示されません。

例 2:

```
DIAGRAM “Class”
{
PROPERTY "Programming Language"{ EDIT Text Listonly LIST
"Programming Languages" Default "CORBA" LENGTH 30 INITIAL
USER REQUIRED }
}

SYMBOL "Class" in "Class"
{
PROPERTY "Package" { EDIT OneOf "Package" READONLY }
PROPERTY "Stereotype" { INVISIBLE EDIT Text ListOnly List "Class
Stereotypes" DEFAULT "" LENGTH 20}
PROPERTY "Programming Language" { INVISIBLE EDIT Text ListOnly
List "Programming Languages" DEFAULT "" LENGTH 30}
}

DEFINITION “Class”
{
PROPERTY "Package" { KEY EDIT OneOf "Package" RELATE BY "is
keyed by" READONLY}
PROPERTY "Stereotype" { EDIT Text LIST "Class Stereotypes"
INIT_FROM_SYMBOL Default "" LENGTH 20 }
PROPERTY "Programming Language"{ EDIT Text ListOnly LIST
"Programming Languages" INIT_FROM_SYMBOL Default "CORBA"
LENGTH 30 INITIAL USER REQUIRED READONLY }
}
```

上記の例で、Programming Language プロパティーがダイアグラムで指定され、Class シンボルはこのプロパティーの値をダイアグラムから継承します。 INIT_FROM_SYMBOL キーワードが指定されているため、Class シンボルの定義も、シンボルを介してこのプロパティーの定義を継承します。

Class 定義で INITIAL USER REQUIRED キーワードが指定されているため、エクスプローラーから Class 定義を作成する場合、作成時に必須プロパティーを提供する必要があります。

INITIAL USER REQUIRED

このキーワードは、モデル化要素 (ダイアグラム、シンボル、または定義) の作成時にプロパティーの値を指定**しなければならない**ことを指定します。値を指定せずに「OK」を押してダイアログを閉じようとすると、Rational System Architect は「xxx プロパティーを指定する必要があります (The xxx property must be supplied)」というメッセージを表示します。OK をクリックしてダイアログを閉じ、ダイアグラム、シンボル、または定義を作成することはできません。 プロパティーに値を指定するか、ダイアログを取り消す必要があります。

例:

DIAGRAM "Activity"
{
PROPERTY "Package" { EDIT OneOf "Package" RELATE BY "is part of" **INITIAL USER REQUIRED** OVERRIDABLE }
PROPERTY "Activity Model" { EDIT OneOf "Activity Model" ReadOnly **INITIAL USER REQUIRED** } ..}

上記の例では、「Activity」ダイアログの作成時に、ダイアグラム・ダイアログにおいて、“Package”プロパティーと “Activity Model”プロパティーの両方を指定しないと「OK」ボタンをクリックできません。

上記の例で、“Package”プロパティーには OVERRIDABLE も指定されていますが、“Activity Model”プロパティーに OVERRIDABLE は指定されていません。OVERRIDABLE キーワードは、このダイアグラムで描画されるシンボルのうち、そのダイアグラムからプロパティーの値を継承するものについてのみ有効です。

例 2:

Diagram “XML”
{..
PROPERTY "XML Schema" { Edit OneOf "XML Schema" **AUTOCREATE** Relate By "is part of" **INITIAL USER REQUIRED OVERRIDABLE READONLY** } ..}

上記の例で、READONLY キーワードを INITIAL USER REQUIRED キーワードと一緒に使用すると、ユーザーがこのプロパティーの値を入力するまでダイアログを閉じることができないこと、および初期値を指定するとこのプロパティーは読み取り専用になり、ユーザーがこの値を変更できなくなることが指定されます。 OVERRIDABLE は、このプロパティー値をダイアグラムから継承する定義についてのみ有効です。

INITIAL USER REQUIRED (続き)

したがって、INITIAL USER REQUIRED キーワードを使用した場合、ダイアグラムの作成時に XML Schema プロパティーに値を指定することが必要になります。 AUTOCREATE キーワードを使用すると、このプロパティーに指定されたすべての値の定義が自動的に作成されるようになります。 したがって、「ダイアグラム」ダイアログを閉じるために「OK」をクリックすると、定義済みの XML Schema 定義が作成されます。

OVERRIDABLE、READONLY、および AUTOCREATE キーワードも参照してください。

INVISIBLE

グラフィック・ダイアログで、プロパティーを削除することなく不可視にします。不可視プロパティーは、定義に必要であるがユーザーにとっては意味がないプロパティーを指定する場合に使用します。

例:

```
SYMBOL "Class" in "Class"
{
PROPERTY "Package" { EDIT OneOf "Package" READONLY }
PROPERTY "Stereotype" { INVISIBLE EDIT Text ListOnly List "Class
Stereotypes" DEFAULT "" LENGTH 20}
PROPERTY "Programming Language" { INVISIBLE EDIT Text ListOnly
List "Programming Languages" DEFAULT "" LENGTH 30}
}

DEFINITION “Class”
{
PROPERTY "Package" { KEY EDIT OneOf "Package" RELATE BY "is
keyed by" READONLY}
PROPERTY "Stereotype" { EDIT Text LIST "Class Stereotypes"
INIT_FROM_SYMBOL Default "" LENGTH 20 }
PROPERTY "Programming Language"{ EDIT Text ListOnly LIST
"Programming Languages" INIT_FROM_SYMBOL Default "CORBA"
LENGTH 30 INITIAL USER REQUIRED READONLY }
}
```

上記の例で、Stereotype プロパティーはクラスのシンボルと定義の両方で使用されています。 これらは同じでなければなりません。ユーザーがクラスでステレオタイプ・プロパティーを選択すると、その値は自動的にシンボルに指定され、そのシンボルの表示方法を決定するために使用されます。 しかし、クラス定義ダイアログにはステレオタイプ・プロパティーが既に指定されているため、ユーザーがクラス定義のシンボル・タブでステレオタイプ・プロパティーを見る必要はありません。両方の個所で表示してもユーザーの混乱を招くだけなので、シンボル・タブでは不可視になっています。

VISIBLE キーワードも参照してください。

JUSTIFY

このコマンドは、SAPROPS.CFG または USRPROPS.TXT では**使用されなくなりました**。USRPROPS.TXT でこのコマンドを指定してもエラーにはならず、USRPROPS.TXT パーサーによって**無視される**のみです。 これは、LAYOUT コマンドの引数の 1 つとして使用されていました。これを指定すると、すべてのコントロールがダイアログ・ページの右マージンおよび左マージンの端に位置揃えされました。

LAYOUT キーワードおよび ALIGN キーワードも参照してください。

KEY

KEY キーワードを使用して、特定のプロパティーをキーとして設定します。キーは、エンサイクロペディアにおけるモデル化要素の名前空間を判別するために使用されます。 また、KEY キーワードは、DISPLAY コマンドの FORMAT キーワードの後で使用可能な引数の 1 つでもあります。その場合の使用法については、KEY (表示用) を参照してください。

デフォルトでは、エンサイクロペディアの各モデル化要素は、そのクラス (ダイアグラム、シンボル、または定義のいずれであるのか)、タイプ (UML ユースケース・ダイアグラム、BPMN プロセス・ダイアグラムなど)、および名前 (例えば、Reservation_System ユースケース・ダイアグラムと Human_Resource_System ユースケース・ダイアグラム) によって区別されます。これらの組み込みデフォルトに加えて、定義モデル化要素として追加のキーを指定することもできます (例えば、クラス属性定義のキーとして、それを含むクラス定義、およびそのクラスを含むパッケージ定義を使用できます)。

KEY コマンドを使用するには、定義のキーとして使用したいプロパティーの中で、このコマンドを指定します。 KEY コマンドは、プロパティーの記述内であればどこにでも配置できますが、重要性を考慮して、プロパティーの中括弧内の最初の項目として (EDIT キーワードの直前に) 使用されるのが通例です。

例:

```
Definition “Use Case Step”
{
PROPERTY "Use Case Name" { KEY EDIT …  }
PROPERTY "Package" { KEY EDIT …}
…
```

KEY (続き)

キーであり、かつ別のオブジェクトを「指している」プロパティー (例えば、単純な TEXT または NUMERIC プロパティーではなく、LISTOF または ONEOF プロパティー) の場合、エンド・ユーザーは、Rational System Architect で作業しているときにそのプロパティーの値を入力する際に、参照されるオブジェクトのクラスおよびクラス・タイプを指定する必要があります。

以下に例を示します。

Definition “Business Process”
{
PROPERTY “System Use Case” {EDIT **ONEOF “Use Case”** …}

上記のステートメントは、“Use Case Name”プロパティーが “Use Case”タイプの定義を参照することを示しています。 Definition は、*クラス* (Rational System Architect における *クラス*、つまり、Diagram、Symbol、または Definition) が指定されない場合のデフォルトです。

プロパティー値自体には、多くの場合、実際に参照されるオブジェクトを識別するために必要なその他の素材がすべて含まれます。 プロパティーの参照対象クラス/タイプにキー・プロパティーが含まれていない場合、参照値は (クラスおよびタイプが既知であるため) オブジェクトの名前のみになりますが、参照対象クラス/タイプにキー・プロパティーが含まれている場合 (上記の例では、キー・プロパティー 「package」を含む「Use Case」)、Rational System Architect は、参照オブジェクトを正しく識別するために、それらのキー・プロパティーの値を知る必要があります。

これを USRPROPS.TXT 内でコーディングして Rational System Architect がエンド・ユーザーに代わってこれらの値を自動的に取得できるようにする、またはエンド・ユーザーにピリオドでキー・パーツを区切って完全修飾名を入力させる、のいずれかを行ってください。

- ユーザーに代わって Rational System Architect に値を自動的に取得させるには、KEYED BY コマンドを使用します。
- プロパティーに対して KEYED BY 節が指定されていない場合、Rational System Architect は、これらの追加キー値が参照自体で指定されると想定します。つまり、ユーザーが、ピリオドでキー値を区切って参照オブジェクトの完全修飾名を入力する必要があります (*"Order System"* パッケージ内の *Order_Product* ユースケースの *"Specify email"* ユースケース・ステップの場合、ユーザーは *"Order System".Order_Product."Specify email"* と入力する必要があります)。

注: 異種混合の参照プロパティーの場合は、これとは異なります。HETEROGENEOUS を参照してください。

KEYED BY 節には、関連するすべてのものをリストにするという用途もあります。 例えば、あるユースケース定義の “Use Case Steps”プロパティーで参照されるすべてのユースケース・ステップは、同じユースケース (“Use Case Steps”プロパティーを含むユースケース) に属します。 複数参照プロパティー (ListOf など) で参照するすべてのオブジェクトが同じ親オブジェクトに属している場合、他の 1 つ以上の他のプロパティーを使用して親オブジェクトを識別することをお勧めします。 そのような場合、KEYED BY 節を使用して、他のどのプロパティーを使用するのかを Rational System Architect に指示します。

注: 定義のキー・プロパティーは、ASGRID コマンドで形成されるグリッドには表示されません。 例えば、ユースケース定義内のユースケース・ステップは、ASGRID コマンドで生成されるグリッドに表示されますが、ユースケース・ステップのキー・プロパティー (それを所有するパッケージおよびユースケース) はユースケース・ステップのグリッドには表示されません。

注: ダイアグラムに KEY EDIT ONEOF を追加することはできません。

KEYED BY キーワードも参照してください。

KEYED BY

KEYED BY 節は、参照されたオブジェクトのキー・コンポーネントの検出方法を指定するために、オプションで使用されます。 KEYED BY 節には、各キー・コンポーネントの部分をコンマで区切って指定します。

KEYED BY 節には、次の 2 つの利点があります。

1. エンド・ユーザーが参照値の完全修飾名を (ピリオドで修飾子を区切って) 指定する必要がなくなります。例えば、パッケージ ***"Order System"*** のクラス ***Customer*** の ***email*** という名前のクラス属性を参照するプロパティーの場合、エンド・ユーザーは、***"Order System".Customer.email*** と入力する代わりに、***email*** と入力するだけで済みます。
2. これを使用すると、参照値のすべてのキー・コンポーネントを同じにすることができます。 例えば、クラス定義の LISTOF “Class Attribute”プロパティーでリストされるすべての属性は、同じクラスおよび同じパッケージに属します。

例:

例えば、Class の “Class Attribute”プロパティーの KEYED BY 節は次のようになります。

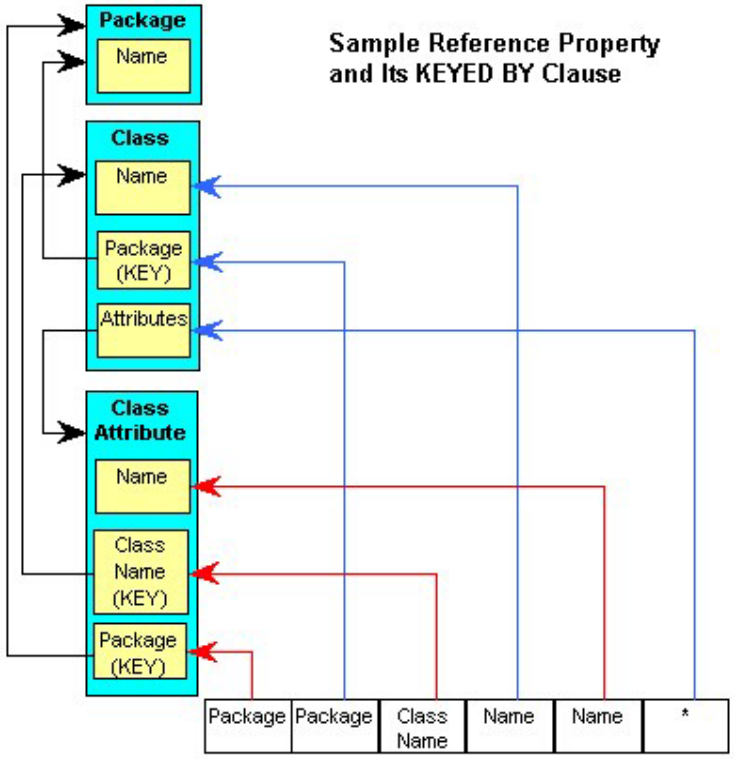
```
DEFINITION "Class"
{
…
PROPERTY "Attributes" { … LISTOF "Class Attribute"
KEYED BY {Package:Package, "Class Name":Name, Name:* } … }
```

上記の例では、Package:Package、“Class Name”:Name、および Name:* の 3 つの *キー・コンポーネント*がコンマで区切って指定されています。これらのコンポーネントは、参照先の Class Attribute 定義を識別するために必要な 3 つの部分 (パッケージ名、クラス名、およびクラス属性名) を参照しています。これらを逆の順序で見ていくと、次のことが分かります。

- クラス属性の名前は***この***プロパティーに指定されているため (*は「ここ」を意味します)、**Name:***
- クラス属性定義内のキー・プロパティー“Class Name” の値はこのオブジェクトの名前に含まれているため、**“Class Name”:Name**
- クラス属性定義内のキー・プロパティー Package の値はこのオブジェクトのパッケージ・プロパティーで指定されているため、**Package:Package**

KEYED BY (続き)

以下の概略ダイアグラムは、上記の例での KEYED BY 節の使用方法を示しており、KEYED BY 節の概要を理解するために役立ちます。



この図は、上で述べたことを示しています。つまり、クラスの定義では、クラス属性はそのパッケージ (クラス属性のパッケージ・プロパティーに保管され、ユーザーが属しているクラスのパッケージ値から取得されます)、そのクラス名 (クラス属性の“Class Name”プロパティーに保管され、そのクラスの実際の名前から取得されます)、および名前 (クラス属性の“Name”プロパティーに保管され、そのプロパティーから取得されます) を指定することによって入力されます。

KEYED BY (続き)

要約すると、次のようになります。

1. **参照オブジェクト**の各キー・コンポーネントについて、KEYED BY 節でコンポーネントが指定されます。
2. KEYED BY 節のコンポーネントは、コンマで区切られます。
3. 各コンポーネントは 2 つの部分からなります。
    - 最初の部分は参照オブジェクトのキー・コンポーネントを示しています。
    - 2 番目の部分は、そのコンポーネントの値がどこにあるのかを示し、
    - 2 つの部分はコロンで区切られています。

ただし、特定のデフォルト値を採用して KEYED BY 節を単純化することができます。コンポーネントの 2 つの部分が同じである場合、2 番目の部分は省略でき、最後のコンポーネントの 2 番目の部分を省略すると、「ここ」(つまりアスタリスク) を示すものと想定されます。したがって、Class の“Attributes”プロパティーの KEYED BY 節は、実際には次のようにコーディングできます。

KEYED BY {Package, "Class Name":Name, Name }

もちろん、KEYED BY ステートメントで使用されるすべてのプロパティーが存在している必要があります。これにより、Rational System Architect は、“Package” プロパティーと “Class Name”プロパティーが “Class Attribute” 定義に存在すること、およびそれらの両方が KEY であることを確認します。

このような LISTOF プロパティー内の共通キー・コンポーネントをコーディングする手間を省くほかに、共通値を提供するその他のプロパティーを指定して KEYED BY 節を使用すると、**各参照で確実に同じ値が使用されるようにすることができます**。したがって、これまで使用してきた例で、クラスの “Attributes”プロパティーで参照されるすべてのクラス属性は、同じパッケージ内の同じクラスに属することが強制されています (このクラスでは望ましい特性です)。

明瞭化および単純化のために、参照されるオブジェクトのキー・コンポーネントを区別したほうが便利な場合もあります。そのような状況では、別個のコンポーネントを提供してプロパティーを指定するために KEYED BY 節が使用されます。実際にこのような理由から、プロパティーが KEY であり、KEY プロパティーが指定されたオブジェクトを参照している場合、Rational System Architect は、コンポーネントが異なるプロパティーであることを**必要とします**。

KEYED BY (続き)

名前以外の一部のキー・コンポーネント値については、その値を参照自体に指定するほうが、別のプロパティーから取り出すよりも望ましいことがしばしばあります。 これは、値を提供する適切なプロパティーが存在しない場合、またはプロパティーのすべての参照でキー・コンポーネントを同じにすることが望ましくない場合が該当します。 この場合は、QUALIFIABLE キーワードを使用します。例えば、クラス定義には次のプロパティーが含まれます。

PROPERTY "Operations" {Edit … ParmListOf "Method"
KEYED BY {"Package","Class Name":Name,"Formal Parameters"
QUALIFIABLE, Name } … }

これは、参照されるメソッドの“Package”キー・プロパティーと “Class Name”キー・プロパティーの値が、それぞれ、クラスの“Package”プロパティーおよび Name から得られるのに対し、メソッドの“Formal Parameters”プロパティーの値とそれらのメソッドの名前がクラスの“ Operations”プロパティー自体から得られることを示しています。 したがって、それぞれの参照には、“Formal Parameters”プロパティーの値と名前の値の 2 つのコンポーネントが、ピリオドで区切って指定されます。

Rational System Architect では、KEYED BY 節を含むキー・プロパティーで　QUALIFIABLE キーワードを指定**しない**ようにする必要があります。これは、上で述べた明瞭化と単純化を実現するためです。

KEY (表示用)

KEY は、**DISPLAY** コマンドの FORMAT の後で引数として使用することもできます。キーとして指定されたプロパティーは、そのシンボルの別のセクションで表示されます。

例:

```
DEFINITION "Entity"
{
PROPERTY "Description"
{ EDIT COMPLETE LISTOF "Attribute" FROM "Data Element"
KEYED BY {Model, "Entity Name":"Name", "Name"} RELATE BY
"uses" ASGRID COPYSCRIPT OnCopyEntityDesc EDITCLASS
SACPropertyAttributeGrid Label "Attribute List" LENGTH 4096
ZOOMABLE DISPLAY { FORMAT KEY LEGEND "Primary Key" }
DISPLAY { FORMAT NONKEY LEGEND "Non-Key Attributes" }
...}
```

上記の例で、FORMAT KEY コマンドは、ユーザーが 1 次キーとして指定したすべての属性を、エンティティー・シンボルの「Primary Key」凡例の下に配置します。FORMAT NONKEY コマンドは、すべての非 1 次キー属性をエンティティー・シンボルの「Non-Key Attributes」凡例の下に配置します。

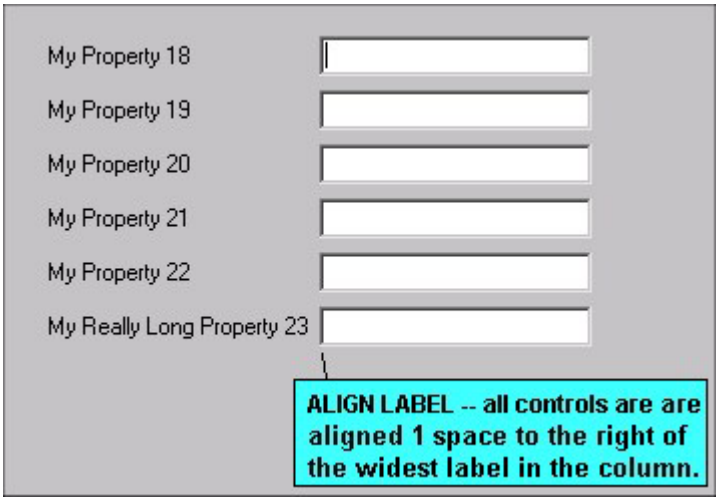
LABEL

LABEL コマンドを使用する目的は、次の 2 つです。

目的 1: LABEL は、{xe "Label:Keyword"}**ALIGN** コマンドで使用する引数の 1 つです。これは、すべてのコントロールを、その列の最も長いラベルの右にスペースが 1 つ空くように桁揃えするために使用されます。(これに対して、ALIGN と OVER のキーワード・ペアでは、名前がプロパティーの上に配置されます。)

例:

```
Definition "My Definiition"
{
CHAPTER “My Chapter”
LAYOUT { COLS 1 ALIGN LABEL }
PROPERTY "My Property 18"{ EDIT Text Length 10}
PROPERTY "My Property 19"{ EDIT Text Length 10}
PROPERTY "My Property 20"{ EDIT Text Length 10}
PROPERTY "My Property 21"{ EDIT Text Length 10}
PROPERTY "My Property 22"{ EDIT Text Length 10}
PROPERTY "My Really Long Property 23"{ EDIT Text Length 10}
}
```



上の例では、“My Really Long Property 23”に対するコントロールが、ラベルの右側からスペース 1 つ分の位置に置かれるテキスト・ボックスです。このダイアログ上の他のプロパティーに対する他のすべてのテキスト・ボックス・コントロールは、このコントロールの位置に合わせられます。

ALIGN キーワード、BODY キーワード、および OVER キーワードも参照してください。

LABEL

(続き)

目的 2: LABEL は、ダイアログ内のタブ (章)、グループ、またはプロパティーの名前を**指定し直す**ために使用されます。 SAPROPS で定義されたプロパティー名は除去できません。ただし、そのプロパティーに関して表示されるテキストは、USRPROPS.TXT 内で LABEL コマンドを使用することによって変更できます。

例 2:

```
DIAGRAM "Data Flow Diagram" {
  PROPERTY "Event Label Prefix"
    { EDIT Text LENGTH 10 }
  PROPERTY "Key Letters"
    { EDIT text LENGTH 10 LABEL "Process Prefix" } ..}
```

上記のコードを USRPROPS.TXT に追加してエンサイクロペディアを再オープンすると、“Key Letters”コントロールのラベルとして "Processing Prefix" という語句が表示されるようになります。

定義内のグループの名前変更

LABEL コマンドを使用してグループを名前変更することができます。 空のテキスト・ストリング (“ “) を指定すると、グループ・ボックスには語句が表示されなくなります。

例 2:

```
DEFINITION “Attribute” {
GROUP "other stuff" LABEL "" }
```

章の名前変更については、CHAPTER コマンドを参照してください。 ALIGN キーワードと BODY キーワードも参照してください。

LABELPOS

PLACEMENT コマンドのパラメーターであり、「ダイアグラム」ダイアログ、「シンボル」ダイアログ、または「定義」ダイアログにおけるプロパティーの名前 (つまりラベル) の正確な位置を指定するために使用されます。LABELPOS パラメーターには 2 つの引数があります。Windows 単位での (ダイアログの上部からの) 水平位置と、Windows 単位での (ダイアログの左からの) 垂直位置です。

構文:

PLACEMENT { **LABELPOS(4, 52)** PROPPOS (horizontal-positioning, vertical-positioning) PROPSIZE (width, height) }

例:

```
DEFINITION “My Definition”
{
PROPERTY "Table Name" { EDIT Text LENGTH 31 PLACEMENT
{ LABELPOS (4, 24) PROPPOS (20, 24) PROPSIZE(150, 12)} }
}
```

上記の例で、LABELPOS と PROPPOS に同じ y 座標 (24) が指定されています。これは、それらの文字がともに、ダイアログの上部から 24 単位の位置に配置されることを意味しています。つまり、ラベルはコントロールの (上方ではなく) 左に表示されます。 また、PROPPOS の x 座標と LABELPOS の x 座標の差 (20 - 4 = 16) により、プロパティー・コントロールの開始点の左に 10 文字のラベル名 “Table Name”を表示するための十分な余裕 (16 -10 = 6 単位) が確保されます。

重要: 配置およびサイジングに関する一般的なヒントについては、本書の第 2 章を参照してください。

PLACEMENT、PROPPOS、PROPSIZE、および FORMAT キーワードも参照してください。

LAYOUT

このキーワードで、「ダイアグラム」ダイアログ、「シンボル」ダイアログ、または「定義」ダイアログにおけるプロパティーのレイアウトを指定します。(「シンボル」ダイアログは「定義」ダイアログに最後のタブとして組み込まれています。)

LAYOUT コマンドの左および右大括弧の間に、その LAYOUT コマンドで呼び出されるすべてのプロパティーのレイアウトを指定する引数を指定します。そのダイアログのプロパティーをいくつの列に配置するかや、プロパティーをどのように位置合わせするかについても指定できます。

「ダイアグラム」ダイアログ、「シンボル」ダイアログ、または「定義」ダイアログに、複数の LAYOUT コマンドを指定しなければならない場合があります。 ダイアログ全体について LAYOUT コマンドを指定して、ダイアログの各グループ内、またはダイアログの各タブ (章) 内、あるいはその両方で指定を変更することができます。

構文:

LAYOUT { [alignment_criteria] [PACK_TAB_criteria] [Number of Columns] [JUSTIFY] }

具体的には、次のようになります。

LAYOUT {[ALIGN BODY | ALIGN LABEL | ALIGN OVER]
[PACK | TAB] [COLS <number>] [JUSTIFY] }

例:

```
SYMBOL "Object" IN "Sequence"
{
LAYOUT { COLS 2 ALIGN OVER }
PROPERTY "Package" { EDIT OneOf "Package" READONLY }
PROPERTY "Class" { EDIT OneOf "Class" KEYED BY { "Package",
Name } REQUIRED READONLY }
..}
```

上記の例では、オブジェクトのシンボル・ダイアログ内のすべてのプロパティーは 2 列に配置されます。

ALIGN、BODY、LABEL、OVER、PACK、TAB、COLS、および JUSTIFY キーワードも参照してください。

LEGEND

長方形のシンボル内に表示可能な、プロパティー名を指定変更するためのプロパティーのストリングです。

例:

```
PROPERTY "Description"
  { EDIT ListOf Data
    DISPLAY { FORMAT Key LEGEND "Key data" }
  }
```

構文:

LEGEND “<Your Text>”: 引用符内にどのようなテキストを配置した場合も、該当項目に値がある場合にのみ、引用符で囲まれたそのテキストは項目の上方のシンボル上に表示されます。

LEGEND "": 説明なしの直線を表示します。 ただし、その項目に値がある場合に限られます。

**LEGEND "$$FORCE$$":**シンボル上の項目の上に横のラインを表示します。このラインは、分割線として機能します。“$$FORCE$$”キーワードは、 単に“ “ を使用する場合とは異なり、プロパティー表示が表示モード・ダイアログによって抑止されていても、 横のラインを表示します。

LEGEND “$$NONE$$”: 該当項目の値が存在するかどうかにかかわらず、シンボル内の項目の上方に横線を表示しません。このラインは、通常、分割線として機能します。

LEGEND “$$VFORCE$$”: シンボル内でプロパティーを左から右の方向に配置できるようにし、それらの間に縦線を引きます。VFORCE キーワードを参照してください。

LEGEND “$$VNONE$$”: プロパティーを左から右にレイアウトできますが、分割線を表示*しません*。VNONE キーワードを参照してください。

DISPLAY キーワードも参照してください。

LENGTH

ユーザーがプロパティー・フィールドに入力できる文字の数を指定します。

例:

```
PROPERTY "From Entity"
  { EDIT TEXT LENGTH 80 }
```

上記の例で、*From Entity* には 80 文字まで指定できます。

LINES

プロパティー・フィールドの行数を深さで設定します。

例:

```
DEFINITION "Constructor"
 { CHAPTER "Desc., Formal Parm"
 GROUP "" {
 LAYOUT { COLS 2 TAB ALIGN OVER }
 PROPERTY "Formal Parameters"   { KEY EDIT Text LENGTH 1020 }
 PROPERTY "Initializer List"    { EDIT Text LENGTH 1000 LINES 4 }
```

このキーワードは、自動的に提供されるスペースをデフォルトよりも大きくしたり小さくしたりしたい場合にのみ役立ちます。

ZOOMABLE キーワードも参照してください。

リスト

USRPPROPS.TXT 内のリスト・キーワードには 2 つの目的があります。 デフォルト長は 1200 です。

目的 1: LIST キーワードは、指定可能なテキスト値のリストを設定します。 このキーワードは USRPROPS.TXT ファイルの先頭 (そこに、指定可能な値のリストを指定します) と、そのリストを使用するプロパティー内の 2 個所で定義する必要があります。すべてのリスト指定ステートメントは、USRPROPS.TXT ファイルの先頭 (すべての DIAGRAM、DEFINITION、または SYMBOL 指定ステートメントの前) に指定する必要があります。

例:

```
List "Method Stereotypes"
{
VALUE "Get"
VALUE "Let"
VALUE "Set"
}
DEFINITION “Method” {..
PROPERTY "Stereotype"{ EDIT Text LIST "Method Stereotypes"
Default "" LENGTH 30 } …}
```

ラジオ・ボタンとドロップダウン・リスト

Rational System Architect は、LIST ステートメント内の値の数が 4 以下の場合には、リストを自動的にラジオ・ボタン選択項目のリストとして表示します。 値の数が 5 以上の場合には、リストは自動的にドロップダウン・リスト・ボックスとして表示されます。 ユーザーは、ドロップダウン・リスト・ボックスに独自の値を入力できます。リストされる値が 4 以下のときにドロップダウン・リスト・ボックスを表示したい場合には、LISTONLYCOMBO キーワードを使用してください。

目的 2: LIST キーワードは、**DISPLAY** コマンド内の FORMAT の後で引数として指定することもできます。 LIST キーワードを指定すると、項目がリスト内のシンボルの上に表示されます。空白文字を二重引用符で囲まない場合、それぞれの空白文字ごとに改行が行われます。

例:

```
DEFINITION "Operational Node"
{PROPERTY "Operational Activities" {EDIT LISTOF "Operational
Activity" LENGTH 2000 DISPLAY {FORMAT LIST Legend
"Activities"} ..}
```

LISTONLY キーワードおよび LISTONLYCOMBO キーワードも参照してください。

LISTOF

プロパティーに指定できるタイプの 1 つ。EDIT キーワードと一緒に使用して、そのプロパティーが**他の定義のリスト**を参照することを指定します。例えば、あるクラスに、クラス属性のリストである Attributes というプロパティーが含まれているとします。クラス属性は、それ自体が定義タイプであり、独自のプロパティーのセットが割り当てられています。これに対して、アクセス・タイプと呼ばれるクラスのプロパティーは、Public、Private、Protected などの単純なテキスト選択項目のリストです。(この単純なテキスト・リストを定義するには、LIST コマンドが使用されます。LIST を参照してください。) また、あるプロパティーが他の定義を 1 つのみ参照することを指定する ONEOF と対比してください。その例として、あるクラスに Package という 1 つのプロパティーが含まれ、そのプロパティーが、クラスが入っている 1 つのパッケージが指定する場合が挙げられます。Package はそれ自体が定義です。

LISTOF は EDIT キーワードと一緒に使用します。構文は以下のとおりです。

PROPERTY "Your Property" { EDIT LISTOF "Referenced Definition Type" } LENGTH 1200}

多くの場合、ASGRID キーワードが LISTOF とともに使用されます。ASGRID は定義のリストをグリッドで表示します。これを使用しない場合、定義はデフォルトのリスト構造でリストされます。LISTOF は、ZOOMABLE キーワードおよび COMPLETE キーワードと一緒に使用することがあります (これらについては、本章の別の個所で説明します)。LISTOF プロパティーについては、LENGTH キーワードはデフォルトで 1200 に設定されます。LENGTH は、プロパティー・フィールドにユーザーが入力できる文字数を指定します。この場合には、リストに収容可能な定義の名前の合計文字数です。

例:

DEFINITION "Use Case"
{
PROPERTY "Preconditions" { ZOOMABLE EDIT ListOf "Pre/Post Condition" LENGTH 1200 }...}

ONE OF、EXPRESSIONOF、COMPLETE、および ZOOMABLE キーワードも参照してください。

LISTONLY

プロパティーの値が、表示リスト (USRPROPS.TXT ファイルの上部にある LIST キーワードを介して作成されます) から取得される必要があることを示します。ユーザーは、このリストに独自の値を入力することは***できません***。

例:

```
List "Importance"
{
Value "Mandatory"
Value "Strongly Desired"
Value "Should Have"
Value "Icing on the Cake"
Value “Not Important”
}

Definition "Use Case Step"
{
PROPERTY "Importance" {Edit Text ListOnly List "Importance"
Length 20 Default "Should Have" }
..}
```

上記の例では、ユースケース・ステップ定義ダイアログで、リストがドロップダウン・リストとして提供されます。このドロップダウン・リストには、ユーザー独自の項目を入力することができます。リスト・ステートメントには、値が 5 つあることに注意してください。値が 4 つ以下の場合、ユースケース・ステップ定義ダイアログ内のリストは、トグル・ボックスでの選択として提供されます。リストされる値が 4 つ以下しかない場合に、ドロップダウン・リストを表示したい場合には、LISTONLYCOMBO キーワードを使用してください。

LIST キーワードおよび LISTONLYCOMBO キーワードも参照してください。

LISTONLYCOMBO

LIST の値の数に関係なく、ドロップダウン・リストを提供します。 また、ユーザーがリストに独自の値を入力することはできません。

LISTONLYCOMBO キーワードを使用すると、LIST コマンドで提供されない機能が提供されます。LIST コマンドを使用する場合、Rational System Architect は、LIST ステートメント内の値の数が 4 以下の場合には、リストを自動的にチェック・ボックス選択項目のリストとして表示します。値の数が 5 以上の場合には、リストは自動的にドロップダウン・リスト・ボックスとして表示されます。 ユーザーは、ドロップダウン・リスト・ボックスに独自の値を入力できます。リストされる値が 4 以下のときにドロップダウン・リスト・ボックスを表示したい場合には、LISTONLYCOMBO キーワードを使用してください。

例:

```
List "Importance"
{
Value "Mandatory"
Value "Strongly Desired"
Value "Should Have"
}

Definition "Use Case Step"
{
PROPERTY "Importance" {EDIT TEXT LISTONLYCOMBO LIST
"Importance" LENGTH 20 DEFAULT "Should Have" }
..}
```

上記の例では、List ステートメントに 3 つの値しか指定されていませんが、ユースケース・ステップ定義ダイアログではリストがドロップダウン・リストとして指定されます。 単純な LIST ステートメントを使用した場合には、値が 5 個未満であるため、値はトグル・ボックスとして表示されます。

LIST キーワードおよび LISTONLY キーワードも参照してください。

MAX; MAXIMUM

数値として定義されたプロパティーに指定可能な最大値を示します。 数値フィールドには、数値のみを入力できます。

例:

```
PROPERTY Length
  { EDIT numeric LENGTH 2 MINIMUM 1 MAXIMUM 99 }
```

MENU

ダイアグラム・ワークスペースではなく「描画」メニューおよび「描画」ツールバーでシンボル を表示するために使用するグラフィックを参照します。

例:

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

上記の例で、画像 satellitetoolbar.bmp は「無線ネットワーク (Wireless Network)」ダイアグラムの「描画」メニューに配置されます。

DEPICTIONS キーワードも参照してください。

MIN; MINIMUM

数値として定義されるプロパティーの最小値を指定します。 数値フィールドには、数値のみを入力できます。

例:

```
PROPERTY Length
  { EDIT numeric LENGTH 2 MINIMUM 1 MAXIMUM 99 }
```

MINISPEC

Rational System Architect で、ミニスペックはプロセス・シンボルの処理ロジックを表すステートメントです。 ミニスペックは、一般に構造化英語と呼ばれる公式の構文を使用して記述されます。 MINISPEC キーワードは、EDIT キーワードとともに使用されます。

例:

```
DEFINITION "Process"
{
PROPERTY "Description"
{ ZOOMABLE EDIT MINISPEC LENGTH 1500 LABEL "Minispec" }
...}
```

ミニスペックはプロセス・シンボルの処理ロジックを表すステートメントです。具体的には、プロセスによって入力データを出力データに変換する方法を表します。

次に、ミニスペック・ステートメントの例を示します。

```
If ISBN number brand new,
Create "ISBN MASTER LIST"
Else
Update "Borrower Request"
```

Rational System Architect は、ミニスペック用語を使用して、プロセスの入出力フローと、データ・フロー上のデータ要素およびデータ構造を比較する (バランス) ことができます。 このバランス機能を使用するには、システムによりテキストを逐語的に分析し、重要用語を探す必要があります。 重要用語は、単一引用符または二重引用符で囲むことでフラグ付けされます。 システムがすべての用語を考慮するように選択することもできますし、システムが考慮対象としてフラグ付けされた重要用語のみを考慮するように選択することもできます。

デフォルトで、システムは引用符によってフラグ付けされた重要用語のみを考慮します。次に例を示します。

```
Compute "extended_cost" = "unit_cost" times "quantity"
```

システムが二重引用符で囲まれた用語だけでなく、ミニスペック内のすべての用語を考慮するようにするには、SA2001.INI ファイルで MinispecUsesQuotes を「N」に設定する必要があります。このように設定した場合は、上記のサンプル・ミニスペックは次のように記述できます。

```
Compute extended_cost = unit_cost times  quantity
```

NAME

定義のキーの一部が、オブジェクト自体の名前であり、親オブジェクトの名前である可能性もあることを示すために使用します。

例:

```
DEFINITION "SQL Server Trigger"
{
PROPERTY "Table Name"
{ EDIT OneOf Definition "Table" RELATE BY "is keyed by" KEYED BY
{"Database Name", "Owner Name","Table Name":Name, Name} }
...}
```

上記の例では、トリガーの定義内のキーである「Table Name」というプロパティーは、そのトリガーが定義されている表の名前です。このトリガー自体の名前もキーの一部です。

NODESC;
NODESCRIPTION

このキーワードは、その定義に DESCRIPTION プロパティーがないことを指定します。

構文:

```
DEFINITION <def_name>
  { NODESC
  }
```

例:

```
DEFINITION "XML Attribute Type"
{
NODESC
PROPERTY "Data Type" { Edit Text LIST "XML Data Type" Length
100 }
PROPERTY "Required" { Edit Text List "XML yesno" Length 100 }
PROPERTY "Default" { Edit Text Length 1000 }
..}
```

Rational System Architect では、NODESC キーワードを使用して 「説明」フィールドが除去されている定義タイプがいくつかあります。 これらの定義タイプでは、「説明」は不要であり、ユーザーにとって邪魔になるだけです。これらの例としては、トリガー・テンプレート、 テーブル・シノニム、テーブル、ストアード・プロシージャー、ビューなどがあります。

NONADDR、NONADDRESSABLE

アドレス・リストから定義を除去するために使用されます。「アドレス可能」として事前定義された 13 個の定義タイプがあります。「アドレス可能」とは、 ダイアグラム上のシンボルをそれらの定義タイプに「関連付ける」ことができることを意味します (任意のシンボルを選択して、「辞書」メニューから「関連付け」を選択して、定義タイプを選択します)。「アドレス可能」として指定された定義タイプは、一般に要件、ルール、テスト計画など、ダイアグラム上のシンボルが「関連付けられている」または満たされているものです。これらの定義のいずれかを「辞書」メニューの「関連付け」 ドロップダウン・メニューから除去するには、USRPROPS.TXT 内のステートメントを変更します。

例:

```
DEFINITION "Change Request"
  {
    NONADDR
  }
```

キーワード ADDRESSABLE も参照してください。

NONE

実際には $$NONE$$ キーワードであり、DISPLAY キーワードとともに使用されます。詳しくは、DISPLAY キーワードを参照してください。

NONKEY

Display コマンドで FORMAT の後に指定できる引数の 1 つ。キーとして指定されていない要素は、そのシンボルの別のセクションに表示することができます。

例:

```
DEFINITION "Entity"
{
PROPERTY "Description"
{ EDIT COMPLETE LISTOF "Attribute" FROM "Data Element"
KEYED BY {Model, "Entity Name":"Name", "Name"} RELATE BY
"uses" ASGRID COPYSCRIPT OnCopyEntityDesc EDITCLASS
SACPropertyAttributeGrid Label "Attribute List" LENGTH 4096
ZOOMABLE DISPLAY { FORMAT KEY LEGEND "Primary Key" }
DISPLAY { FORMAT NONKEY LEGEND "Non-Key Attributes" }
...}
```

上記の例では、FORMAT NONKEY コマンドによって、プライマリー・キーとして指定されていないすべての属性が、エンティティー・シンボル上の「キーでない属性」凡例の下に配置されます。

NOTHING

RELATE BY NOTHING コマンドで使用されます。

RELATE BY を参照してください。

NUMERIC

これは許可されたプロパティー・タイプの 1 つです。 このキーワードはそのプロパティーが数値であることを指定し、そのフィールドには数字 (および正/負の符号) のみを入力できます。LENGTH ステートメントは、そのフィールドに入力できる数字の桁数を指定します。 そのフィールドには、数字と正/負の符号のみを入力でき、小数点や他の文字を入力することはできません。

例:

```
SYMBOL "Process" IN "Data Flow Gane & Sarson"
  { PROPERTY "Short Description"
    { EDIT Text LENGTH 1500 }
     PROPERTY "Number" { EDIT Numeric LENGTH 4 }
```

OF DEFINITION REFERENCED IN

定義の限定リストを指定します。プロパティーの「選択」ボタンをクリックすると、このリストの中から選択できます。 このキーワードを使用して、EDIT LISTOF または EDIT ONEOF ステートメントにさらに細かい条件を加えます。 特定の参照定義に属している定義のみが表示されるように指定できます。

例 1:

```
DEFINITION "Object"
{
PROPERTY "Package" { KEY EDIT OneOf "Package" RELATE BY "is
   keyed by" READONLY}
PROPERTY "Class" { KEY EDIT OneOf "Class" KEYED BY
   { "Package", Name } RELATE BY "is keyed by" READONLY }
PROPERTY "Attributes" { ZOOMABLE EDIT LISTOF  "Class
   Attribute" OF DEFINITION REFERENCED IN "Class"
  KEYED BY {"Package", "Class Name":"Class", Name} LENGTH
   4096 DISPLAY {FORMAT COMPONENT_SCRIPT
   _FmtNewUMLObjInstAttr LEGEND "$$FORCE$$"} }
..}
```

上記の例では、オブジェクトの属性グリッドで「選択」ボタンをクリックすると、そのオブジェクトに含まれているクラスの属性のみが表示されます。OF DEFINITION REFERENCED IN が参照している「Class」プロパティーも、上記のようにオブジェクトの定義で指定される必要があります。(ちなみに、上記の例ではまた、KEYED BY {"Package", "Class Name":"Class", Name} ステートメントによって、属性自体が、そのパッケージ、クラス名、および独自の属性名でキー設定されるように指定されています。)

例 2:

```
DEFINITION "Message"
{
PROPERTY "To Class"  { KEY EDIT OneOf "Class" }
...
PROPERTY "Operation"  { EDIT ParmOneOf "Method" OF
DEFINITION REFERENCED IN "To Class"  KEYED BY { "Class
Name" : "To Class", Name, "Formal Parameters"} LENGTH 1000}
..}
```

上記の例では、メッセージ・ラインの定義が詳細化され、メッセージ・ラインの終点であるオブジェクトのクラスの

メソッドのみが表示されるようになっています。これらは「To Class」のメソッドです。このことが可能になる理由は、メッセージ・ラインの終点であるオブジェクト・シンボル (オブジェクト・ライフライン) にはその参照クラスのプロパティーが含まれており、メッセージ行にはそのプロパティー「To Class」が含まれているからです。 これは OMT シーケンス図の定義です。UML シーケンス図には、この例にはない別のキー設定 (パッケージによる) が存在します。 UML メッセージ・ライン定義の例については、キーワード OF DEFINITION AND SUPERS REFERENCED IN を参照してください。

キーワード OF DEFINITION AND SUPERS REFERENCED IN も参照してください。

OF DEFINITION AND SUPERS REFERENCED IN

定義の限定リストを指定します。プロパティーの「選択」ボタンをクリックすると、このリストの中から選択できます。この限定リストは、特定の定義の要素を参照し、**また**、その定義の継承元である (継承ラインでアタッチされている) 他の定義の要素を参照します。このキーワードは、EDIT LISTOF または EDIT ONEOF ステートメントにさらに詳細な条件を追加するために、UML モデリングで使用されます。

例:

```
DEFINITION "Message/Stimulus"
{
PROPERTY "To Package" { KEY EDIT OneOf "Package" RELATE
  BY "is keyed by" READONLY}
PROPERTY "To Class" { KEY EDIT OneOf "Class" KEYED BY
  { "Package":"To Package", Name } }
PROPERTY "To Object"{ KEY EDIT OneOf "Object" KEYED BY
  { "Package":"To Package", "Class":"To Class",Name} }
PROPERTY "Operation" { EDIT ParmOneOf "Method"
  OF DEFINITION AND SUPERS REFERENCED IN "To Class"
  KEYED BY { "Package" QUALIFIABLE,"Class Name"
  QUALIFIABLE,"Formal Parameters" QUALIFIABLE ,Name}
  LENGTH 1000 DISPLAY {FORMAT COMPONENT_SCRIPT
  _FmtNewUMLEventOperation LEGEND "$$NONE$$" } LABEL
  "Method" HELP "Choose a method from the proper class" }
```

上記の例では、メッセージ定義内の「選択」をクリックすると、メッセージ・ラインがアタッチされているクラス (「From Class」ではなく「To Class」) のメソッドが得られるとともに、そのクラスのスーパークラス (継承ラインでそのクラスに接続されているクラス) である任意のクラスの任意のメソッドが得られます。

キーワード OF DEFINITION REFERENCED IN も参照してください。

ONEOF

プロパティーに指定できるタイプの 1 つ。このキーワードを EDIT キーワードとともに使用して、そのプロパティーが別の定義タイプの定義の**いずれか**を参照することを指定します。

例:

SYMBOL "Relation" IN "Entity Relation"
{
PROPERTY "From Entity" **{ EDIT ONEOF Entity** READONLY **}**
PROPERTY "To Entity" { **EDIT ONEOF Entity** READONLY }
}

上記の例では、2 つのエンティティー間の Relation ラインは、その Symbol タブ上に、このラインが接続しているエンティティーを含んでいます (このラインの終点のエンティティーと始点のエンティティーの両方)。それぞれの場合に、1 つのエンティティーのみが表示されます。 この情報は自動的に提供されるため (Rational System Architect は始点と終点の情報を追跡管理します)、プロパティーは READONLY に設定されます。

例 2:

Definition “Extends”
{
PROPERTY "Use Case Steps" { ZOOMABLE **EDIT ONEOF "Use Case Step"** KEYED BY {"Model Name":"Model Name", "Use Case Name":"From Use Case", Name}
}

上記の例では、Extends ラインの定義には、拡張 (このラインの接続先である他の Use Case への拡張) が行われる Use Case Step (参照 Use Case 内) への参照が含まれています。 Extends 定義内のこのプロパティーの「選択」をクリックすると、すべての Use Case Step のリストが表示されますが、プロパティー・フィールドには 1 つの Use Case Step をドラッグして取り込むためのスペースしかありません (これとは対照的に LISTOF では複数の定義をドラッグして取り込めます)。

キーワード LISTOF および EXPRESSIONOF も参照してください。

OVER

ALIGN コマンドの引数であり、プロパティーの名前 (またはラベル) をプロパティーのコントロール (テキスト・フィールドやドロップダウン・リスト・ボックスなど) の上に配置します。

例:

```
Definition "Use Case Step"
{
Chapter "My Properties"
LAYOUT { COLS 2 ALIGN OVER TAB }
PROPERTY "Importance" {Edit Text ListOnlycombo List "Importance"
Length 20 Default "Should Have" }
PROPERTY "Number" { EDIT Numeric LENGTH 4  LABEL “Ranking”}
}
```

上記の例では、タブ (Chapter キーワード) のすべてのプロパティーは、各プロパティーの名前またはラベルがそのコントロールの上に配置されるようにレイアウトされます。 Number プロパティーのラベルは「Ranking」に変更されます。このプロパティーのラベルは、単純な数値フィールドであるそのコントロールの上に配置されます。 Importance プロパティーの名前は、ドロップダウン・リストであるそのコントロールの上に配置されます。

このキーワードとは対照的に、BODY キーワードは、プロパティーの名前またはラベルをコントロールの左側に配置します。

BODY、TAB、ALIGN、LABEL、および JUSTIFY というキーワードも参照してください。

OVERRIDABLE

このキーワードを使用すると、ダイアグラムから継承された定義の読み取り専用プロパティーを、読み取り専用であるにもかかわらず、その定義の初回作成時に変更する (または書き込みを行う) ことができます。OVERRIDABLE はダイアグラム・レベルで***のみ***使用され、そのダイアグラム上に描画されたシンボルを表す定義に属しているプロパティーを、そのシンボルの初回作成時に変更できることを指定します (読み取り専用プロパティーである場合でも)。

例:

```
Diagram "XML"
{
CHAPTER "Diagram"
PROPERTY "XML Schema" { EDIT ONEOF "XML Schema"
AUTOCREATE RELATE BY "is part of" INITIAL USER REQUIRED
OVERRIDABLE READONLY }
..}

DEFINITION "XML Element"
{
..
PROPERTY "XML Schema" { Key Edit ONEOF "XML Schema" Relate
By "is keyed by" Initial User Required Readonly }
..}
```

上記の例では、XML Element 定義はその「XML Schema」プロパティーの値を、この定義で定義している XML Element シンボルから継承し、このシンボルはその「XML Schema」プロパティーの値をこのシンボルが配置されているダイアグラムから継承します。XML Element シンボルをダイアグラム・ワークスペースに初めて配置するときに、XML Element 定義の「XML Schema」プロパティーの値を変更できます。「OK」をクリックしてこの定義を閉じてから再び開くと、「XML Schema」プロパティーが読み取り専用になり変更できなくなります。

このキーワードは、XML Element 定義の指定ではなく、Diagram の指定で使用されます。

PACK

LAYOUT コマンド内で垂直位置を制御します。このコマンドは、複数の列内のコントロールとラベルのセットと、そのすぐ右側にあるコントロールとラベルのセットとの間に最小限のスペースを空けることで、これらのセットを区切ります。

例:

```
GROUP "Power Builder Headings/Labels" {
  LAYOUT { COLS 2 ALIGN OVER PACK }
```

キーワード LAYOUT および TAB も参照してください。

PARENT

オブジェクト指向と手法に関する用語としての PARENT は、現在のオブジェクトの継承元となるオブジェクトを 指定します。現在のオブジェクトは、その継承元オブジェクトからすべてのキー・プロパティーを継承します。 PARENT のステートメントには RELATE BY "is keyed by" を含める必要があります。

例:

```
DEFINITION "Use Case Step"
{
PROPERTY "Use Case Name" { KEY EDIT OneOf "Use Case"
  PARENT RELATE BY "is keyed by" READONLY  HELP "Name of
  Owning Use Case" }
...}
```

PARMONEOF

このキーワードは、SAPROPS.CFG 内の特定のケースでのみ使用し、USRPROPS.TXT 内では**使用しないでください**。このキーワードは、Rational System Architect 構文内の参照プロパティーが一般的な UML 操作のように表示されることを指定します。 具体的には、修飾された参照プロパティーが表示される際に、キーの修飾部分は、Rational System Architect で通常表示される二重引用符ではなく括弧で囲まれ、参照先のオブジェクトの名前 が先頭に配置されるようにキーの順序が変更されます。 例えば、メソッドは **“int, char”.meth** ではなく **meth(int, char)** と表示されます。

例:

```
DEFINITION "Activity Model"
{ ..
PROPERTY "Operation" { EDIT PARMONEOF"Method" OF
DEFINITION REFERENCED IN "Active Class" KEYED BY
{ "Package" QUALIFIABLE, "Class Name":"Active Class", "Formal
Parameters" QUALIFIABLE, Name}  LENGTH 1000 DISPLAY
{ FORMAT  STRING LEGEND "$$NONE$$" } LABEL "Method" HELP
"Specify class and then click choices button" }
}
```

PARMLISTOF

このキーワードは、SAPROPS.CFG 内の特定のケースでのみ使用し、USRPROPS.TXT 内では**使用しないでください**。このキーワードは PARMONEOF と同じ意味ですが、メソッドのグリッドなどのプロパティーの参照リストに適用されます。このキーワードは、Rational System Architect 構文内の参照プロパティーが一般的な UML 操作のように表示されることを指定します。具体的には、修飾された参照プロパティーが表示される際に、キーの修飾部分は、Rational System Architect で通常表示される二重引用符ではなく括弧で囲まれ、参照先のオブジェクトの名前 が先頭に配置されるようにキーの順序が変更されます。 例えば、メソッドは **“int, char”.meth** ではなく **meth(int, char)** と表示されます。

例:

```
Definition “Class” {
CHAPTER "Methods"
PROPERTY "Operations" { ZOOMABLE EDIT
COMPLETE_ALLOW_NEW PARMLISTOF "Method" KEYED BY
{ "Package", "Class Name":Name, "Formal Parameters"
QUALIFIABLE, Name }
LENGTH 1200 ASGRID DISPLAY { FORMAT COMPONENT_SCRIPT
_FmtNewUMLOperation LEGEND "$$FORCE$$"} LABEL "Methods" }
```

PLACEMENT

このコマンドを使用して、DIAGRAM、SYMBOL、または DEFINITION ダイアログ上のプロパティーの正確な配置を指定します。 PLACEMENT コマンドには、次のパラメーターがあります。

LABELPOS (x, y): プロパティーの名前 (またはラベル) の左上隅の始点を指定します。x は水平位置 (ダイアログの左端からの距離) を指定し、y は垂直位置 (ダイアログの上端からの距離) を指定します。 x と y はどちらも Windows 単位です。

PROPPOS (x, y): ダイアログ上のプロパティーのコントロールの左上隅の始点を指定します。x は水平位置 (ダイアログの左端からの距離) を指定し、y は垂直位置 (ダイアログの上端からの距離) を指定します。 x と y はどちらも Windows 単位です。

PROPSIZE (x, y): コントロールの長方形サイズを指定します。x はコントロールの幅を指定し、y は高さを指定します (Windows 単位)。

例:

```
DEFINITION “Class”
{
CHAPTER "Entity Information”
LAYOUT { COLS 2 TAB ALIGN OVER }
PROPERTY "Table Name" { EDIT Text LENGTH 31
   PLACEMENT {PROPPOS (4,  24) PROPSIZE(150, 12)} }
PROPERTY "Naming Prefix" { EDIT Text LENGTH 8 LABEL
   "Column Prefix" HELP "Prefix of column name"
   PLACEMENT {PROPPOS (175,  24) PROPSIZE(40, 12)} }
..
}
```

上記の例では、PLACEMENT コマンドによってタブ (CHAPTER) の LAYOUT コマンドがオーバーライドされます。Table Name プロパティーのテキスト・ボックス・コントロールは、Class 定義ダイアログ (Entity Information タブ) 上に配置され、その位置の始点は、ダイアログの左端からの距離が 4 Windows 単位で、ダイアログの上端からの距離が 24 Windows 単位となります。このテキスト・ボックスは、幅が 150 単位で高さが 12 単位です。

重要: 配置およびサイジングに関する一般的なヒントについては、本書の第 2 章を参照してください。

PROPPOS、PROPSIZE、および LABELPOS の各キーワードも参照してください。

PROPERTY

このキーワードに続いて、ダイアグラム、シンボル、または定義の特性を設定する引数を指定します。 PROPERTY キーワードの後ろには、プロパティー名を引用符で囲んで記述する必要があります。 それに続いて、{} という括弧の内側に、または BEGIN ステートメントと END ステートメントの間にプロパティーの特性を指定する必要があります。

構文:

PROPERTY “<property_name>”
  { EDIT <edit_type> <property_parameter> **}**

または

PROPERTY “<property_name>”
  BEGIN EDIT <edit_type> <property_parameter>
  END

PROPPOS、PROPSIZE

DIAGRAM、SYMBOL、または DEFINITION ダイアログ上のプロパティーの正確な配置を指定するために使用する PLACEMENT コマンドのパラメーター・ペア。 PROPPOS コマンドは、Windows 単位での水平位置 (ダイアログの上端からの距離) と、Windows 単位での垂直位置 (ダイアログの左端からの距離) という 2 つの引数をとります。 PROPSIZE コマンドも x と y という 2 つの引数をとり、これらはそれぞれプロパティーのコントロールの幅と高さを Windows 単位で指定します。

構文:

PLACEMENT { **PROPPOS (horizontal-positioning, vertical-positioning) PROPSIZE (width, height)** }

例:

```
DEFINITION “My Definition”
{
PROPERTY "Table Name" { EDIT Text LENGTH 31
PLACEMENT {PROPPOS (4, 24) PROPSIZE(150, 12)} }
}
```

上記の例では、Table Name プロパティーの始点 (そのテキスト・ボックスの左上隅) を定義ダイアログの左端および上端からそれぞれ 4 Windows 単位および 24 Windows 単位離れた位置に配置します。 このテキスト・ボックスは、幅が 150 Windows 単位で長さが 12 Windows 単位です。 このステートメントでは、このテキスト・ボックスに割り当てられる名前 (またはラベル「Table Name」) に関しては何も指定していません。何も指定されていないため、ラベルはデフォルトでテキスト・ボックスの左側に配置されます。 ラベルの位置を変更するには、FORMAT コマンドまたは PLACEMENT {LABELPOS} コマンドを使用します。

重要: 配置およびサイジングに関する一般的なヒントについては、本書の第 2 章を参照してください。

PLACEMENT、LABELPOS、および FORMAT の各キーワードも参照してください。

PUBLISHER

このキーワードを使用すると、プロパティーの値を **SA Information Publisher** の出力で公開するかどうかを指定できます。このキーワードは、**PUBLISHER SHOW** および **PUBLISHER ORDER** という 2 つの引数をとります。

構文:

PROPERTY "Some user property" {
PUBLISHER
{
SHOW (YES|NO) ‘ デフォルトは YES
ORDER nnnn ‘ デフォルトはゼロ (ソートしない)}
}

PUBLISHER ORDER

これは **PUBLISHER** コマンドに対する引数であり、**SA/Publisher** の公開出力におけるプロパティー値の表示順序を指定できます。 この引数は、**PUBLISHER SHOW** 引数とともに使用します。

構文:

PROPERTY "Some user property" { ... **PUBLISHER** {SHOW YES|NO **ORDER** nnnn } ...}

デフォルト値は 0 (ソートしない) です。

例:

DEFINITION "Business Requirement"
{
PROPERTY "Benefit" { EDIT Text LENGTH 50 **PUBLISHER {ORDER 2 }** }
PROPERTY "Status" { EDIT Text LENGTH 50 **PUBLISHER {ORDER 1 }** }
PROPERTY "Difficulty" { EDIT Text LENGTH 50 **PUBLISHER {ORDER 3 }** }
PROPERTY "Assigned to" { EDIT Text LENGTH 50 **PUBLISHER {ORDER 4}** }
}

PUBLISHER SHOW

これは **PUBLISHER** コマンドに対する引数であり、**SA/Publisher** の出力でプロパティーの値を公開するかどうかを指定できます。 **PUBLISHER_ORDER** コマンドをこのキーワードとともに使用することもできます。

構文:

PROPERTY "Some user property" {.....PUBLISHER {SHOW YES|NO} ... }

デフォルト値は YES です。

例:

```
DEFINITION "Business Requirement"
{
PROPERTY "Benefit" { EDIT Text LENGTH 50 PUBLISHER {SHOW NO} }
}
```

上記の例では、Benefit プロパティーは、レポートによって出力されるように指定されている場合でも、SA/Publisher によって生成される Web サイトには表示されません。

QUALIFIABLE

QUALIFIABLE は、参照元オブジェクト内の他のプロパティーから 参照先オブジェクトの 1 つ以上のキー・コンポーネントを取得する必要がなく、そのプロパティー自体で指定できる参照プロパティーで使用します。このキーワードは、すべてのキー・データを参照元定義のプロパティー内に格納できないときに、参照先定義の名前をキー・プロパティーによって修飾する必要がある場合に使用します。

次の KEYED BY 節の例を見てください。

KEYED BY {key_component-1: property_name_1, name}

key_component_1 の値を property_name_1 から取得すること、およびその結果として参照プロパティーに参照オブジェクトの名前のみが格納されることを指定しています。もう 1 つの例として次の KEYED BY 節を見てください。

KEYED BY {key_component-1 QUALIFIABLE, name}

この KEYED BY 節は、key_component_1 の値をこのプロパティー (この KEYED BY 節を持つプロパティー) から取得すること、およびその結果として参照プロパティーに参照オブジェクトの名前と key_component_1 の両方の値が格納されることを指定しています。このような条件では、名前の値はピリオドによって key-component_1 の値と区切られます。

例:

```
PROPERTY "Operations”
{ ZOOMABLE EDIT ParmListOf "Method”
KEYED BY {"Class Name":Name, "Formal Parameters"
QUALIFIABLE, Name}
LENGTH 1200
ASGRID
DISPLAY { FORMAT COMPONENT_SCRIPT FmtUMLOperation
LEGEND "$$FORCE$$"}
}
```

READONLY

そのプロパティーが読み取り可能で、変更不可であることを指定します。 READONLY は、SAPROPS 内で、 ソフトウェアによって挿入されるがユーザーに表示される必要がある値を持つプロパティーで使用されます。このキーワードは、Initial AuditId、Date and Time、および Update AuditId、Date and Time には常に使用されます。リレーション・ライン、制約、およびシンボル同士を結ぶ他のライン (ラインの From シンボルと To シンボルが重要なもの) は常に READONLY です。

例:

```
SYMBOL "Link" IN "Booch (94) Object"
REM "defined by Object Link"
{
PROPERTY "From Class"
{ EDIT OneOf "Class" READONLY }
PROPERTY "From Object"
{ EDIT OneOf "Object" KEYED BY {"Class":"From
Class", Name}  READONLY }
PROPERTY "To Class"
{ EDIT OneOf "Class" READONLY }
PROPERTY "To Object"
{ EDIT OneOf "Object" KEYED BY {"Class":"To
Class", Name}  READONLY }
...}
```

REFPROP

各プロパティーは、1 つの定義内で一度だけ使用できます (ただしそのプロパティーが #ifdef で囲まれている場合は除きます)。 同じプロパティーを 1 つの定義内で複数回使用するためには、Control および RefProp キーワードを使用する必要があります。 このため、CONTROL および REFPROP キーワードは TESTPROC とともに使用されることがよくあります。

Control を使用するには、 定義の先頭に、Control が参照するプロパティーに対する最初の参照が必要です。REFPROP キーワードは、Control キーワードとともに使用されます。

例:

```
Definition "Index"
{
CHAPTER "Modeling Properties"
  { TESTPROC TestPropertyNotValue TESTPROPERTY
"DBMS" TESTSTRING { "ORACLE 8" } }
  PROPERTY "Primary Key"
  {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 "Modeling Properties"
  { TESTPROC TestPropertyValue TESTPROPERTY "DBMS"
TESTSTRING { "ORACLE 8" } }
  Control "Primary Key"
  { REFPROP "Primary Key" }
  Control Unique
  { REFPROP "Unique" }
  Control Clustered
  {REFPROP "Clustered"}
  ...
  }
```

CONTROL キーワードも参照してください。

RELATE (BY)、RELATED (BY)

参照プロパティーのデフォルト関係タイプは「uses」です。このデフォルトの関係を異なる関係 (「keyed by」など) または何の関係も持たない状態 (「RELATE BY nothing」と 記述します) に指定変更するには、RELATE BY キーワードを使用します。

RELATE BY キーワードでは次の関係を使用できます。

Nothing: 何の関係もありません。

Uses – デフォルトです。定義に定義が含まれていることを意味します。

Explained By: シンボルが定義によって説明されることを意味します。

Defined By: シンボルが定義によって定義されることを意味します。

Is A: ある定義がある定義の「インスタンスである (is an instance of)」ことを意味します (例えば、列はデータ要素です)。

Identifies: あるオブジェクトが別のオブジェクトを識別することを意味します。

Comprises: あるオブジェクトが複数のオブジェクトで構成されていることを意味します (例えば、モデルはエンティティーや関係などで構成されており、カテゴリーは複数のカテゴリー関係で構成されています)。

Originated From – オブジェクトが定義から発生していることを意味します。

Is Based On: オブジェクトが定義 (通常はデータ要素) に基づいていることを意味します。

Is Part Of: ある定義がある定義の一部であることを意味します。これは OneOf または ListOf とともに使用されます。

Is Keyed by: ある定義がある定義によって識別されることを意味します。

ユーザー定義関係: 20 個のユーザー定義関係タイプ (USER 1 から USER 20) もあり、これらは RENAME コマンドを使用して作成された場合に使用できます (例: RENAME RELATION “USER 1” to “XXXX”)。

例:

```
Definition “Use Case Step”
{
PROPERTY "Use Case Name" { KEY EDIT ONEOF "Use Case"
KEYED BY {"Package", Name} RELATE BY "is keyed by" READONLY
HELP "Name of Owning Use Case" }
PROPERTY "Package" { KEY EDIT ONEOF "Package" RELATE BY
"is keyed by" READONLY}}
```

上記の例では、1 つ目の KEY EDIT は「Use Case Name」プロパティーが Use Case Step 定義のキー・プロパティーであることを示します。この「Use Case Name」プロパティーは Use Case 定義を参照しています (これは「ONEOF "Use Case"」によって指定されています)。その Use Case (ステップがキー設定されている先の Use Case) の完全なキーを指定する必要があります。この場合は、KEYED BY コマンドを使用してキー (Use Case が格納されているパッケージ、および Use Case 自体の Name) を指定しています。 最後に、Use Case Step がこの Use Case によってキー設定されていることを指定する必要があります (これは「RELATE BY “is keyed by”」コマンドによって実行される処理です)。参照プロパティーのデフォルトの関係タイプは「uses」であるため、RELATE BY 節が追加されます。異なる関係タイプ (「keyed by」など) を希望する場合は、デフォルトを指定変更する必要があります。

上記の例の 2 つ目の KEY EDIT は、「Package」プロパティーが Use Case Step 定義のキーでもあることを指定しています。具体的には、Use Case Step はパッケージ定義によってキー設定されています。ただしパッケージ定義自体は、それ自身の名前を除いて、どの追加プロパティーによってもキー設定されていないため、KEYED BY コマンドは使用されていません。 パッケージは、「is keyed by」関係によって Use Case Step に関連付けられています。

RELATION

プロパティーとその参照対象との間の実質的な関係。

RELATE [BY]、RELATED [BY] の各キーワードも参照してください。

REM、REMARK

このコマンドの後ろに記述された、単一引用符または二重引用符で囲まれたテキストを無視します。

例:

```
 GROUP "Connections"
   { LAYOUT { COLS 2 TAB ALIGN OVER }
     PROPERTY "From Entity"
   { KEY EDIT OneOf "Entity" RELATE BY "is keyed by"
         READONLY}
     PROPERTY "To Entity"
   { KEY EDIT OneOf "Entity"  RELATE BY "is keyed by"
         READONLY}
   } REM "End of group Connections"
```

RENAME

Rational System Architect で通常使用されている名前以外の名前によるオブジェクトの参照を可能にします。

例:

```
RENAME SYMBOL "Control Transform"
   IN "Data Flow Ward & Mellor" TO "Process"
RENAME DIAGRAM "Data Flow Ward & Mellor"
   TO "Ward Mellor"
```

RENAME DEFINITION

150 個のユーザー提供定義を使用できるようにします。これらの 150 個のユーザー提供定義には User 1 から User 150 という名前が付けられています。 RENAME DEFINITION コマンドを使用すると、これらのうち任意の数のユーザー提供定義を新しい名前に変更して、実質的に新しい定義タイプを作成できます。 RENAME DEFINITION ステートメントは、USRPROPS.TXT ファイルの先頭付近に列記する必要があります。

例:

```
Rename Definition "User 10" to "System Requirement "
```

RENAME DIAGRAM

20 個のユーザー提供ダイアグラムを使用できるようにします。これらの 20 個のユーザー提供ダイアグラムには User 1 から User 20 という名前が付けられています。 RENAME DIAGRAM コマンドを使用すると、これらのうち任意の数のユーザー提供ダイアグラムを新しい名前に変更して、実質的に新しいダイアグラム・タイプを作成できます。 RENAME DIAGRAM ステートメントは、USRPROPS.TXT ファイルの先頭付近に列記する必要があります。

例:

Rename Diagram "User 10" to "Requirements Hierarchy"

RENAME SYMBOL

150 個のユーザー・シンボルを使用できるようにします。. これらの 150 個のユーザー提供シンボルには User 1 から User 150 という名前が付けられています。 RENAME SYMBOL コマンドを使用すると、これらのうち任意の数のユーザー提供シンボルを新しい名前に変更して、実質的に新しいシンボル・タイプを作成できます。 RENAME SYMBOL ステートメントは、USRPROPS.TXT ファイルの先頭付近に列記する必要があります。

例:

Rename Symbol "User 10" to "System Requirement"

REQUIRED

ダイアグラムまたは定義を作成可能にするために、ユーザーがプロパティーを入力する必要があることを指定します。そのプロパティーは、そのダイアグラムまたは定義の初期「**名前**」ダイアログに自動的に表示されます。

例:

この例では、XML Element Entity 定義を作成するために、ユーザーは XML Schema プロパティーの値を入力する必要があります。

```
DEFINITION "XML Element Entity"
{
PROPERTY "XML Schema" { Key Edit ONEOF "XML Schema"
Relate By "is keyed by" Required
Readonly }
}
```

RETAIN STYLE

このキーワードは、ユーザーが提供するメタファイルが System Architect で使用されるときに、その元のグラフィカル・スタイルと色付けが保持されるように指示します。 このキーワードは DEPICTIONS 節とともに使用されます。

ユーザーが提供する外部イメージを使用してダイアグラム上にシンボルを表現する場合、デフォルトでは、Rational System Architect 内の他のシンボルと同様に、これらのシンボルの特徴 (塗りつぶす色や線の色など) を指定できます。 RETAIN STYLE キーワードを DEPICTIONS 節で指定した場合は、ユーザー定義シンボルの色は変更されずに元のままになります。

例:

```
LiST "Node Stereotypes"
{
Value "Client" DEPICTIONS {diagram images\client.wmf menu
images\client.bmp}
Value "Database" DEPICTIONS {diagram images\database.wmf menu
images\database.bmp}
Value "Firewall" DEPICTIONS {diagram RETAIN STYLE
images\firewall.wmf menu images\firewall.bmp}

SYMBOL "Node" in "Deployment"
{
PROPERTY "Stereotype" { INVISIBLE EDIT Text ListOnly List "Node
Stereotypes" DEFAULT "" LENGTH 32}
}

DEFINITION "Node"
{
PROPERTY "Stereotype"
{ EDIT Text LIST "Node Stereotypes" Default "" LENGTH 32 }
```

上記の例では、firewall.wmf を使用して Deployment ダイアグラム上のノード・シンボルを表現できます (このノードのステレオタイプが「Firewall」に設定されている場合)。ユーザーが提供するメタファイル firewall.wmf (ユーザーによってエンサイクロペディアのデータベースの FILES テーブルに追加されたもの) は、ダイアグラム上に描画される際に、Rational System Architect の外部で使用されていたのとまったく同じ色を使用して描画され、Rational System Architect のカラー・ツールを使用してこれらの色を変更することはできません。

SACPropertyOnOfBase

EDITCLASS SACPropertyOneOfBase コマンドで使用します。 **このキーワードの組み合わせを使用しないでください。** このキーワードの組み合わせは、Rational System Architect の特定の状態、あるエンティティー内の 属性による「データ・エレメント」プロパティーの継承に対して特別に設計されたものです。 この状態に対して使用される SAPROPS.CFG 内で、このキーワードの組み合わせが使われます。 それが、このキーワードの組み合わせを適用できる唯一の状態です。 それ以外の状態で使用すると、エラーの原因となります。

スクリプト

SA Basic で書かれたスクリプトを呼び出します。 このスクリプトは、 ListOf および ExpressionOf のどちらでもないプロパティーに使用されます。スクリプトは、プロパティーの値を取ってアクションを 実行します (通常は、ダイアグラム上のシンボルに関する特定タイプの注釈を表示します)。スクリプト自体の命名基準は、次のとおりです。

- _fmt (例えば、_fmtUMLAttr など): この関数自体は、ハードコーディングされているため、変更できません。 SAPROPS.CFG の関数の多くはこの形式です。関数をハードコーディングするのは、Rational System Architect の全体的な応答速度を上げるためです。
- fmt (例えば、fmtUMLAttr など): Rational System Architect のメイン実行可能ディレクトリー内にある、fmtscript.bas に存在します。

独自の関数の作成

シンボルに特定の方法で項目を表示したり、 特定の値を計算したりする、独自の関数を作成できます。作成するスクリプトを fmtscript.bas ファイル に***配置しない***でください。このファイルは、Rational System Architect を インストールまたは更新するたびに上書きされます。独自の関数を作成する場合、作成した関数は **usr_fn.bas** ファイルに配置します。このファイルはデフォルトでは提供されないため、作成して Rational System Architect のメイン・ディレクトリー (<C>:\Program Files\IBM\Rational\11.3.1\System Architect Suite\System Architect) に配置する必要があります。(sarules.bas ファイルには、usr_fn.bas に対する #include があります。)

SAPROPS.CFG で呼び出される大部分の関数 (ハードコーディングされており、 原則として名前の最初に _fmtUMLAttr のようにアンダースコアーが付く) については、 fmtscript.bas ファイル内に同等の関数呼び出し (アンダースコアーなし) があります。 独自の関数を作成したい場合は、fmtscript.bas 内のスクリプトを参考として使用できます。

既存の関数の説明:

FmtOMTAbstractClass は、abstract プロパティーが設定されていればスクリプト **{abstract}** を返しますが、 設定されていなければ何も返しません。

FmtBOOClassConstraint は、制約が指定されていれば、その制約の名前を ***{constraint name}*** のように中括弧で囲んで返しますが、 指定されていなければ何も返しません。

SCRIPT (続き)

FmtOMTObjInstClass は、クラスが指定されていれば、そのクラスの名前を **(*class name*)** のように括弧で囲んで返しますが、 指定されていなければ何も返しません。

FmtEntryAction は、設定されているものがあれば、 スクリプト **entry /** とその入状時アクションの名前を返しますが、 これらが設定されていなければ何も返しません。

FmtExitAction は、設定されているものがあれば スクリプト **exit /** とその退状時アクションの名前を返しますが、 これらが設定されていなければ何も返しません。

FmtOMTTransition は、以下のような値が状態遷移定義で 設定されている場合には、それらの値を以下のような句読記号内に入れて返します。 **(*attribute name*) [*condition*] /*action***

例:

```
CHAPTER "OMT Object-oriented"
GROUP "OMT Object-oriented"  {
..
PROPERTY "Abstract"
{ EDIT Boolean LENGTH 1 DEFAULT "F" DISPLAY { FORMAT
SCRIPT FmtOMTAbstractClass  LEGEND "$$FORCE$$"} }
...
PROPERTY "Constraints" { EDIT Text LENGTH 500 DISPLAY
{ FORMAT SCRIPT FmtBOOClassConstraint  LEGEND
"$$NONE$$"} }
} REM "end of group OMT Object-oriented"
```

キーワード FORMAT、COLUMN_SCRIPT、COMPONENT_SCRIPT、および fmtxxx も参照してください。

STRING

Display コマンドで、キーワード FORMAT の後に指定するデフォルト引数です。*string* を使用すると、辞書項目の内容が入力されているとおりに画面上に表示されます。

例:

```
SYMBOL "Relation" IN "Shlaer Information Model"
{
PROPERTY "Description"  { EDIT Text LENGTH 100 }
PROPERTY "Reverse Phrase"  { EDIT Text LENGTH 65 DISPLAY
{ FORMAT STRING LEGEND "$$NONE$$" } }
..}
```

キーワード FORMAT および DISPLAY も参照してください。

SUPERS

キーワード OF DEFINITION AND SUPERS REFERENCED IN を参照してください。

SYMBOL

SYMBOL は、DEFINITION または DIAGRAM と対照的に、シンボルのプロパティーがリストされるブロックの最初のワードです。

例:

```
SYMBOL "Entity" IN "Entity Relation"
{
PROPERTY "Description"  { EDIT Text LENGTH 500 }
…
}
```

キーワード DIAGRAM および DEFINITION も参照してください。

TAB

このキーワードは、**LAYOUT** コマンドの垂直位置決めを制御します。 複数列上の一連のコントロールやラベルをタブで区切ることにより、各行の項目が上の行の項目の直下に配列されます。

例:

```
GROUP "SQL Server Schema Check Constraint"
{
LAYOUT { TAB ALIGN Over COLS 2 }
PROPERTY "SQL Server Check Constraint Name"
{ EDIT Text LENGTH 30 Label "Constraint Name"}
PROPERTY "SQL Server Check Constraint"
{ EDIT TEXT LENGTH 256 LINES 10 LABEL "Constraint Check"}
} REM "End of Schema Check Constraint group "
```

キーワード LAYOUT および PACK も参照してください。

TESTPROC、TESTPROPERTY、TESTSTRING コマンド群

TESTPROC、TESTPROPERTY、および TESTSTRING コマンド群は、 ダイアグラムごとに、プロパティーに条件付き機能を提供します。このコマンド群は、#ifdef ではエンサイクロペディア全体のレベル に基づいて条件付き機能が提供される点を除けば、#ifdef と同様の機能を提供します。 TESTPROC コマンド群は**ダイアグラム・プロパティー**の一部として機能します。ダイアグラムのプロパティー内で 「テスト・プロパティー」に値が選択されると、定義には特定のプロパティー・セットが含まれます。

TESTPROC コマンド群の主な用途は、論理データ・モデル内で、選択した RDBMS に応じて データ・モデリング・プロパティー・セットを指定することです。

TESTPROC はテスト・プロシージャーを表します。 TESTPROC キーワードに続けて指定可能な値は 2 つ (**TestPropertyValue** お よび **TestPropertyNotValue**) あります。 TESTPROC に続けて TestPropertyValue を指定した場合、「プロパティーをテストして、 それが指定された TESTSTRING の値の 1 つに一致する場合には、この TESTPROC セクションのプロパティーを対象の定義に適用する」ことを意味します。TESTPROC に続けて TestPropertyNotValue を指定した場合、「プロパティーをテストして、 それが指定された TESTSTRING の値の 1 つに一致**しない**場合には、 この TESTPROC セクションのプロパティーを対象の定義に適用する」ことを意味します。 **TestPropertyValue** および **TestPropertyNotValue** には大/小文字の区別があるため、 指定されたとおりの大/小文字を使用する必要があります。 すべて小文字、またはすべて大文字を使用した場合、大/小文字の区別は機能しなくなります。

TESTPROPERTY は、照会されるダイアグラム・プロパティーです。

TESTSTRING は、照会される値です。 このストリングには、1 つ以上の値をリストできます。

Controls および RefProps: 定義では、1 つのプロパティーを 1 度しか使用できません (プロパティーが #ifdef で囲まれている場合を除く)。 同じプロパティーを定義の中で複数回使用するには、Control および RefProp キーワードを使用する必要があります。このため、CONTROL および REFPROP キーワードは TESTPROC とともに使用されることがよくあります。 Control を使用するには、定義の先頭に、Control が参照するプロパティーに対する最初の参照が必要です。

TESTPROC、TESTPROPERTY、TESTSTRING コマンド群 (続き)

例:

```
Definition "Index"
{
CHAPTER "Modeling Properties"
  { TESTPROC TestPropertyNotValue TESTPROPERTY "DBMS"
TESTSTRING { "ORACLE 8" } }
  PROPERTY "Primary Key"
  {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 "Modeling Properties "
  { TESTPROC TestPropertyValue TESTPROPERTY "DBMS"
TESTSTRING { "ORACLE 8" } }
  Control "Primary Key"
  { REFPROP "Primary Key" }
  Control Unique
  { REFPROP "Unique" }
  Control Clustered
  {REFPROP "Clustered"}
  ...
  }
```

TestPropertyValue TESTPROC、TESTPROPERTY、TESTSTRING コマンド群を参照してください。

TestPropertyNotValue TESTPROC、TESTPROPERTY、TESTSTRING コマンド群を参照してください。

TEXT

これは、指定可能なフィールド・タイプです。テキストとしての定義は、リストや、ユーザーの入力した英数字に基づいて行われます。

例:

```
DEFINITION "Relationship"
{
CHAPTER "Relations and Connections"
GROUP "Relation"
{ LAYOUT { COLS 2 TAB ALIGN OVER }
PROPERTY "Role" { EDIT Text LENGTH 31 }
PROPERTY "Role Prefix"  { EDIT Text LENGTH 31 }
}
```

TIME

使用可能なフィールド・タイプの 1 つで、Windows に定義されている時刻形式に適合する表記のタイム・スタンプがプロパティーに含まれることを示します。 CHECKOUT TIME、FREEZE TIME、INITIAL TIME および UPDATE TIME にはそれぞれ特別な意味があります。

例:

```
DIAGRAM "Data Flow Gane & Sarson"
{
PROPERTY "Freeze time"
{ FREEZE  TIME }
..
}
```

TIME の他の利用法が定義に存在することがあります。

例:

```
DEFINITION "X" {
PROPERTY "Creation Time"
{ EDIT Text INITIAL TIME LENGTH 12 READONLY }
}
```

TO

Rename コマンドで使用され、オブジェクトの元の名前を新しい名前と区別します。

例:

```
RENAME SYMBOL Class IN "Booch Class"
  TO "Booch Class"
```

更新

プロパティーが変更されると、システムにより自動的にフィールドが更新されることを示す許容フィールド・タイプです。デフォルトで、*「監査 ID」*、*「更新日時」*、および *「更新時刻」*に使用されます。

UPDATE キーワードは、 LAST CHANGED キーワードと同じ情報を提供します。両方とも定義が最後に変更された時間を指定します。 つまり、誰かが定義ダイアログを開いて、変更し(スペースを追加したり、いずれかのプロパティー の文字を削除したり、文字を除去した後でその文字を再度追加して元に戻したりして、定義を「修正」し)、 「保存」ボタンをクリックして変更を保存した日時を指定します。ユーザーが定義ダイアログを開いて何もせずに「保存」をクリックした場合には、定義は変更されなかった (「修正」されなかった) ため、変更とは見なされません。(注: ユーザーが定義を開くと、定義はそのユーザーによって一時的に「ロック」されます。ユーザーがその定義を変更せず、保存もキャンセルも行わなかった場合、定義は変更されていないため、LAST CHANGED または UPDATE プロパティーはこれを認識しません。ただし、Rational System Architect は、内部的に、誰が最後に定義を「ロックした」かをトラッキングします。この 最終ロック情報は、USRPROPS キーワードを使用して入手することはできません。)

Rational System Architect V9 から、 LAST CHANGED AUDITID、LAST CHANGED DATE、および LAST CHANGED TIME は、 各ダイアグラムまたは定義ダイアログの「アクセス・データ」タブで、デフォルトで提供されています。 これは、製品にハードコーディングされています。つまり、SAPROPS.CFG 内の各定義に LAST CHANGED キーワードはありません。 また、新しく作成するダイアグラム・タイプや定義タイプの USRPROPS.TXT にこのキーワードを追加する必要もありません。

例:

```
DEFINITION "X"
  { PROPERTY "Modified Time"
    { EDIT Text UPDATE TIME LENGTH 12 READONLY }
  }
```

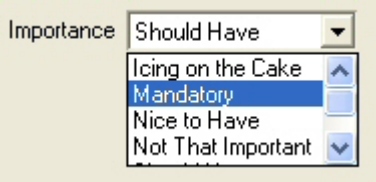
INITIAL キーワードも参照してください。

VALUE

このキーワードは、LIST の値ストリングの値を与えます。

例:

```
List "Importance"
{
VALUE "Mandatory"
VALUE "Strongly Desired"
VALUE "Should Have"
VALUE "Icing on the Cake"
VALUE "Not Important"
}

Definition "Use Case Step"
{
PROPERTY "Importance" {EDIT TEXT LIST "Importance" LENGTH 20
DEFAULT "Should Have" }
}
```



上記の例では、新規リストが USRPROPS.TXT (このファイルの先頭) に作成されます。 リストには、5 つの値が割り当てられています。 後に USRPROPS.TXT のユースケース・ステップの定義で、このリストは、プロパティー“Importance”で使用されます。このタイプのリストでは、ユーザーは、"Importance" フィールドにユーザー独自の値を入力できます。

キーワード LIST、LISTONLY、および LISTONLYCOMBO を参照してください。

VALUESCRIPT

VALUESCRIPT は、SA Basic で書かれた関数を呼び出します。 VALUESCRIPT には、オープン・ダイアログのプロパティー値の整合性検査の実行が含まれています。 この関数は、通常、ダイアログのプロパティーに設定された値を計算し、リアルタイムでダイアログの他のプロパティーの値に必要な変更を行って、整合性規則が実行されるようにします。

ユーザー独自の機能の作成

ユーザー独自の機能を作成し、オープン・ダイアログのプロパティー値の整合性検査を実行することができます。 ユーザー独自の機能の作成方法については、SCRIPT キーワードを参照してください。

例:

```
DEFINITION "Index"
{
PROPERTY Unique
{ PLACEMENT {PROPPOS(84, 0) PROPSIZE(100, 12)} EDIT Boolean
LENGTH 1 VALUESCRIPT ProcessIndexUnique DEFAULT "F" }
.. }
```

上記の例では、関数 ProcessIndexUnique が呼び出されます。これは、fmtscript.bas ファイルにあります。 この関数は、Oracle が選択された DBMS である場合のみに呼び出されます。 この関数は、索引のビットマップ・プロパティーがオンに切り替わっているかどうかを確認し、オンに切り替わっている場合、ProcessIndexUnique 関数により、索引のプロパティーをオフに切り替えます。 この理由は、Oracle では、索引は、ビットマップ索引として指定されている場合、一意になることはできないためです。

例:

```
Definition “Data Element”
{
PROPERTY "SQL Data Type"
{ EDIT text LIST "Standard Data Types" LENGTH 30 VALUESCRIPT
ProcessSQLDataType LABEL "Data Type" "Type" "DT" PLACEMENT
{PROPPOS(4, 26) PROPSIZE(80, 12)} }
..}
```

上記の例では、ProcessSQLDataType は、データ要素がそのタイプを基礎となるデータ・ドメインから継承しているかどうかを確認し、継承している場合には自動的にそのタイプを入力します。 タイプを継承しておらず、ユーザーがタイプ・フィールドを空のままにしている場合、ユーザーが Enter キーを押すか、属性グリッドのフィールドを変更すると、関数は自動的にデフォルトとして文字 10 を入力します。

VFORCE

シンボル内に、デフォルトである水平線ではなく、縦線を描くことができます。 VFORCE は、左から右へプロパティーを配置し、縦線で分離します。 (注: VNONE コマンドでも同様のことを行いますが、縦線は表示されません。)

構文:

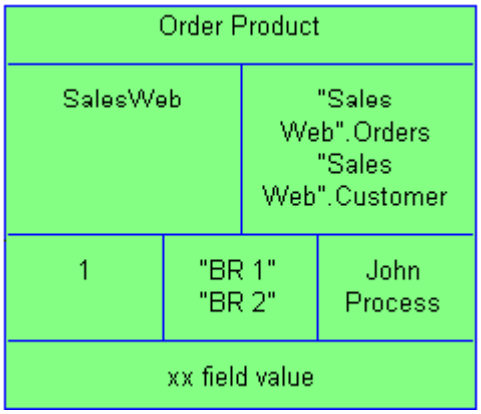
{FORMAT String LEGEND **"$$VFORCE$$"**}

例:

```
DEFINITION "Elementary Business Process"
{
PROPERTY "Supporting Applications"
{  Edit Listof "Application" Label "Applications" LENGTH 2000 HELP
"Must be entered through Matrix"  READONLY DISPLAY { FORMAT
String LEGEND "$$FORCE$$"} }
PROPERTY "Referenced Data"
{ EDIT ListOf "Entity" KEYED BY {Model QUALIFIABLE,
Name} LENGTH 5000 READONLY DISPLAY { FORMAT String
LEGEND "$$VFORCE$$"}}
..}
```

リストされた最初のプロパティーにあるのは、 VFORCE ではなく FORCE であることに注意してください。 最初のプロパティーの右に、続いて配置するプロパティーには、VFORCE が指定されます。以下の図では、"Sales Web".Orders および "Sales Web".Customer 値が、「参照データ」用のボックスにリストされます。 VFORCE コマンドを使用して、「必要なアプリケーション (Supporting Applications)」プロパティー・ボックスの右にこのボックスを表示させます。図では、値 SalesWeb がリストされています。

VFORCE (続き)



また、VFORCE コマンドは、「BR 1」と「BR 2」、および John のプロセスを含むボックスを値 1 を含むボックスの右に表示させるためにも使用されることに注意してください。ただし、これは提供された USRPROPS.TXT サンプルには示されていません。

VISIBLE

プロパティーが SAPROPS.CFG で INVISIBLE と表示されていても、キーワード VISIBLE を使用すると、定義ダイアログにそのプロパティーが表示されます。

例 (SAPROPS):

```
DEFINITION "Watcom Stored Procedure"
        { CHAPTER "Keys and Parameters"
    PROPERTY "Owner Name"
        { EDIT Text KEY  LENGTH 31 }
    PROPERTY "Procedure Number"
        { INVISIBLE EDIT Numeric LENGTH 9 }
    PROPERTY "Description"
        { EDIT Text  LENGTH 400 }
```

例 (USRPROPS):

```
DEFINITION "Watcom Stored Procedure"
    PROPERTY "Procedure Number"
        { VISIBLE }
```

キーワード INVISIBLE も参照してください。

VNONE

実際には $$VNONE$$ キーワードであり、DISPLAY キーワードとともに使用されます。詳しくは、DISPLAY キーワードを参照してください。

シンボル内に、デフォルトである水平線ではなく、縦線を描くことができます。 VNONE は、左から右へプロパティーを分離して配置しますが、プロパティー間に縦線を表示しません。 (注: VFORCE コマンドでも同様のことを行いますが、縦線が表示されます。)

構文:

{FORMAT String LEGEND **"$$VNONE$$"**}

例:

以下の USRPROPS.TXT スニペットの例では、リストされた最初のプロパティーに FORCE が指定されます。 最初のプロパティーの右に、(分離線なしで) 続いて配置するプロパティーには、VNONE が指定されます。

例:

```
DEFINITION "Elementary Business Process"
{
PROPERTY "Supporting Applications"
{  Edit Listof "Application" Label "Applications" LENGTH 2000 HELP
"Must be entered through Matrix"  READONLY DISPLAY { FORMAT
String LEGEND "$$FORCE$$"} }
PROPERTY "Referenced Data"
{ EDIT ListOf "Entity" KEYED BY {Model QUALIFIABLE,
Name} LENGTH 5000 READONLY DISPLAY { FORMAT String
LEGEND "$$VNONE$$"}}

..}
```

WHERE

「選択」ダイアログに、定義の名前付きプロパティーの固定値を含む定義のみを表示します。

例:

```
Rename Definition "User 1" To "Aircraft Type"
Rename Definition "User 2" To "Filtered Aircraft"


List "Engine"
{
  Value "Propeller"
  Value "Jet"
  Value "Glider"
}

Definition "Aircraft Type"
{
  Property "Engine Type"
   { EDIT Text List "Engine" Length 48 }
}
Definition "Filtered Aircraft"
{
   Property "Selected Aircraft Type"
        { edit listof "Aircraft Type" WHERE "Engine Type" = "Jet"}
}
```

上記の USRPROPS.TXT が、エンサイクロペディアに適用され、以下の Aircraft Type 定義がエンサイクロペディアで作成された場合、

```
Mustang (engine = Propeller)
Spitfire (engine = Propeller)
F-16 Fighting Falcon (engine = Jet)
F-86 Sabre (engine = Jet)
```

タイプ Filtered Aircraft (例えば、指定された最 新のジェット戦闘機など) の新規定義の作成の際に、 この定義の「**選択**」ボタンをクリックすると、F-16 Fighting Falcon および F-86 Sabre という 2 つの選択項目の みが表示されます。Propeller に設定された Engine Type についての定義はすべて「**選択**」リストに表示されません。

ZOOMABLE

リスト・ボックスに、ダイアログ・ページ全体に広がる大きさまでボックスを拡大できるボタンを配置します。

例:

```
PROPERTY "User Roles"
  {ZOOMABLE EDIT ListOf "User Role with Access Rights"
  LENGTH 1500 LABEL "User Role(s)"}
```

キーワード LINES も参照してください。

4

IBM サポート

はじめに

問題のトラブルシューティングに役立つセルフ・ヘルプ方式の情報リソースおよびツールが多数あります。ご使用の製品に問題がある場合、以下の方法で対処できます。

ご使用の製品のリリース情報を参照し、既知の問題、回避策、およびトラブルシューティング情報を確認します。

問題の解決に利用できるダウンロードまたはフィックスがないか確認します。

使用可能な知識ベースを検索して、問題の解決策が既に文章化されていないかを確認します。
それでも支援が必要な場合は、IBM®ソフトウェア・サポートに問い合わせて、問題を報告してください。

IBM Rational ソフトウェア・サポートへのお問い合わせ

セルフ・ヘルプ・リソースで問題を解決できない場合は、IBM® Rational® ソフトウェア・サポートにお問い合わせください。

注: 従来からの Telelogic のお客様は、すべてのサポート・リソースを 1 つの参照サイト (http://www.ibm.com/software/rational/support/telelogic/

前提条件

IBM Rational Software Support に問題を送信するには、Passport Advantage® の有効なソフトウェア保守契約を締結している必要があります。Passport Advantage は、IBM の包括的なソフトウェア・ライセンスおよびソフトウェア保守 (製品のアップグレードおよび技術サポート) オファリングです。パスポート・アドバンテージのオンライン登録は、http://www.ibm.com/software/lotus/passportadvantage/howtoenroll.html

- Passport Advantage について詳しくは、以下の Passport Advantage FAQ サイトにアクセスしてください: http://www.ibm.com/software/lotus/passportadvantage/brochures_faqs_quickguides.html
- 他にご質問などございましたら、IBM 担当員にお問い合わせください。

問題をオンラインで (IBM の Web サイトから) IBM Rational ソフトウェア・サポートに送信するには、以下のようにしてください。

- IBM Rational Software Support Web サイトでユーザーとして登録します。登録について詳しくは、http://www.ibm.com/software/support/ を参照してください。
- サービス要求ツールで許可された呼び出し元としてリストに入れます。

その他の情報

Rational ソフトウェア製品に関するニュース、イベント、およびその他の情報については、以下の IBM Rational ソフトウェア Web サイトにアクセスしてください:
http://www.ibm.com/software/rational/

問題の送信

問題を IBM Rational ソフトウェア・サポートに送信するには、以下のようにしてください。

1. 問題がビジネスに及ぼす影響を判別します。IBM に問題を報告するときには、重大度レベルを提示するように求められます。そのため、問題がビジネスに及ぼす影響を理解し、査定しておく必要があります。

重大度レベルを決定するには、以下の表を使用します。

重大度	説明
1	問題が業務に重大な影響を及ぼします。プログラムを使用できないため、運用上の重大な影響が発生します。この状態は即時に解決する必要があります。
2	この問題は、業務に大きな影響を及ぼします。プログラムは使用可能ですが、その機能は極度に限定されます。
3	この問題は、業務に多少の影響を及ぼします。プログラムは使用可能ですが、あまり重要でない機能 (運用上の重大な影響が発生しない) は使用できません。
4	この問題は、業務に最小限の影響を及ぼします。問題が運用に及ぼす影響がほとんどないか、または問題に対する適切な回避策が既に実施されています。

2. 問題を説明し、その背景となる情報を収集します。IBM に対して問題を説明するときには、できる限り具体的な情報を提供してください。IBM Rational Software Support スペシャリストの支援によりお客様が問題を効

率的に解決できるように、関連する背景情報をすべて提出してください。時間を節約するために、次の質問に対する答えを準備しておいてください。

- 問題が発生したときに実行中だったソフトウェアのバージョンは何か。
- 正確な製品の名前とバージョンを判別するには、以下から適切な方法を選択してください。
  - IBM Installation Manager を開始して、**「ファイル」** > **「インストール済みパッケージの表示 (View Installed Packages)」**をクリックします。パッケージ・グループを展開して、パッケージを選択し、パッケージ名とバージョン番号を確認します。
  - 製品を始動し、**「ヘルプ」** > **「バージョン情報」**をクリックし、製品名とバージョン番号を確認します。
- オペレーティング・システムとバージョン番号は何か (すべてのサービス・パックまたはパッチを含む)。
- 問題の徴候に関連するログ、トレース、およびメッセージはあるか。
- 問題を再現することができるか。再現できる場合、どのステップを実行すると問題が再現するか。
- システムに変更を加えたか。例えば、ハードウェア、オペレーティング・システム、ネットワーキング・ソフトウェア、あるいは他のシステム・コンポーネントを変更したか。

3. この問題のために現在予備手段を使用していますか。使用している場合は、問題を報告するときに予備手段について説明できるように準備してください。
4. 次のいずれかの方法で、IBM Rational ソフトウェア・サポートに問題を送信します。

- オンライン: IBM Rational Software Support Web サイト https://www.ibm.com/software/rational/support/ にアクセスします。Rational サポート・タスク・ナビゲーターで、「**Open Service Request**」をクリックします。電子問題報告ツールを選択して、Problem Management Record (PMR) を開き、問題についての説明を入力します。

- サービス要求をオープンにする方法について詳しくは、以下のサイトにアクセスしてください: http://www.ibm.com/software/support/help.html

- IBM Support Assistant を使用して、オンライン・サービス要求をオープンにすることもできます。詳しくは、http://www.ibm.com/software/support/isa/faq.html を参照してください。
- 電話による方法: お住まいの国または地域の電話番号については、各国の連絡先をリストした IBM ディレクトリー (http://www.ibm.com/planetwide/) にアクセスして、お住まいの国名または地域名をクリックしてください。

- IBM 担当員を介して: IBM Rational Software Support にオンラインまたは電話でアクセスできない場合は、IBM 担当員にお問い合わせください。必要に応じて、IBM 担当員がお客様に代わってサービス要求をオープンすることができます。国ごとの完全な連絡先情報については、http://www.ibm.com/planetwide/ を参照してください。

5

付録:

はじめに

この章では、IBM® Rational® System Architect®の法的な使用および商標について説明します。

この章のトピック	ページ
特記事項	5-2
商標	5-5

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものであり、本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒242-8502
神奈川県大和市下鶴間1623番14号
日本アイ・ビー・エム株式会社
法務・知的財産 知的財産権ライセンス渉外

〒242-8502 神奈川県大和市下鶴間1623番14号
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

Intellectual Property Dept. for Rational Software
IBM Corporation
1 Rogers Street
Cambridge, MA 02142
U.S.A

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権ライセンス

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。 このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生的創作物にも、次のように、著作権表示を入れていただく必要があります。

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は 表示されない場合があります。

商標

IBM、IBM ロゴおよび ibm.com は、世界の多くの国で登録された International Business Machines Corp. の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、
www.ibm.com/legal/copytrade.htmlhttp://www.ibm.com/legal/copytrade.html

Microsoft および Windows は、Microsoft Corporation の米国およびその他の国における商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。