IBM Rational

Handbuch zur Erweiterbarkeit von System Architect USRPROPS.TXT

Release 11.3.1

Lesen Sie vor der Verwendung dieser Informationen den Abschnitt "Bemerkungen" auf Seite 5-1 des Anhangs.

Diese Ausgabe gilt so lange für IBM® Rational® System Architect®, Version 11.3.1 und alle nachfolgenden Releases und Bearbeitungen, bis in einer neuen Ausgabe ein anderslautender Hinweis erfolgt.

Inhaltsverzeichnis

1

Erweitern des Metamodells einer Enzyklopädie von System Architect Metamodel

Einführung

Dieses Kapitel stellt die Mechanismen vor, mit denen Sie das Metamodell einer Enzyklopädie von IBM® Rational® System Architect® mithilfe der USRPROPS.TXT-Datei erweitern können.

Erweitern von Rational*System Architect*

Rational System Architect kann auf verschiedene Weise erweitert und angepasst werden. Das Zeichnungsverhalten kann durch eine Reihe von Optionen im Tool selbst und in der SA2001.INI-Datei angepasst werden. So können etwa die Symbolleisten angepasst werden, die Matrixeditoren, die Berichte usw. Rational System Architect verfügt darüber hinaus über eine integrierte Unterstützung von Microsoft Visual Basic for Applications. Dadurch kann der Benutzer native Makros schreiben, die innerhalb von Rational System Architect ausgeführt werden können und mit denen zahlreiche Funktionen verbunden sind. So können beispielsweise nützliche Dienstprogramme hinzugefügt oder sogar das Verhalten des Tools gesteuert werden.

Erweitern des Metamodells mithilfe der USRPROPS.TXT-Datei

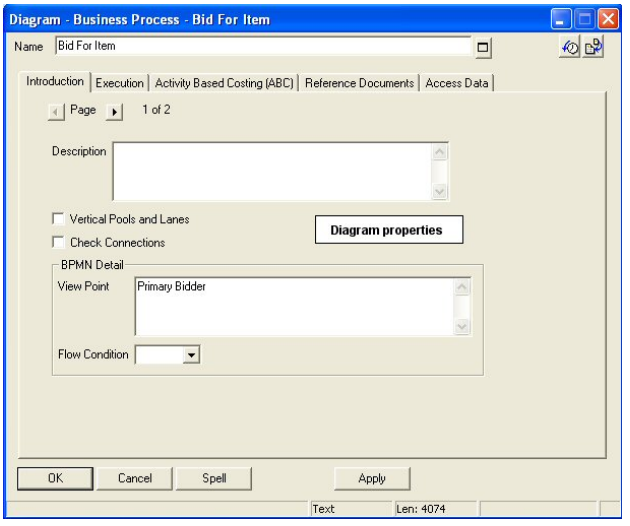
Das ist aber noch nicht alles. Rational System Architect verfügt darüber hinaus über eine äußerst leistungsstarke Funktion: Der Benutzer hat die Möglichkeit, das zugrunde liegende Metamodell anzupassen und zu erweitern und kann somit Einfluss darauf nehmen, wie Informationen in der Enzyklopädie gespeichert werden. Das standardmäßige Metamodell einer Rational System Architect-Enzyklopädie ist in der SAPROPS.CFG-Datei festgelegt (der Haupteigenschaftsdatei von **S**ystem **A**rchitect). Diese steuert zum Beispiel die Symbole in den Diagrammen, die Beziehung zwischen den Symbolen und ihre jeweilige Definition sowie die Eigenschaften von Symbolen, Definitionen und Diagrammen. Benutzermodifikationen am Metamodell werden in einer Textdatei (USRPROPS.TXT) vorgenommen. Diese wird, zusammen mit der SAPROPS.CFG-Datei, beim Laden einer Enzyklopädie syntaktisch analysiert. Dabei wird die SAPROPS.BIN-Datei erstellt. Die USRPROPS.TXT-Datei überschreibt die SAPROPS.CFG-Datei. Sie können die USRPROPS.TXT-Datei mithilfe einer systemeigenen Scripting-Sprache von Rational System Architect bearbeiten und auf diese Weise das Metamodell einer Enzyklopädie anpassen oder erweitern.

Metamodell der Enzyklopädie von Rational System Architect

Inhalt des Metamodells

Das Metamodell modelliert, wie Rational System Architect die Diagramme, Symbole und Definitionen speichert, die Sie während Ihrer Arbeit erstellen. Das System Architect-Metamodell umfasst sämtliche Diagrammtypen, Symboltypen und Definitionstypen, die Eigenschaften, die in diesen Typen enthalten sind, sowie verschiedene Beziehungen, die zwischen diesen Modellierungselementen bestehen.

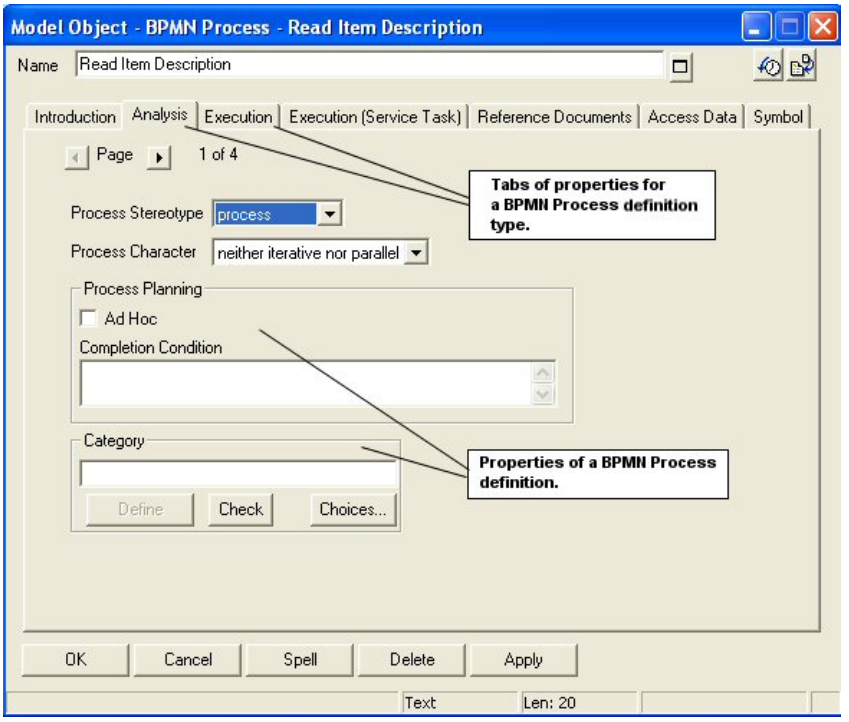
Ein Beispiel für einen *Diagrammtyp* ist ein Geschäftsprozessdiagramm. Dieses verfügt über bestimmte *Eigenschaften*, die etwa angeben, ob Pools und Lanes horizontal oder vertikal angezeigt werden sollen (Vertikale Pools und Lanes), ob Liniensymbolverbindungen im Diagramm während des Zeichnens automatisch geprüft werden sollen (Verbindungen überprüfen) usw.



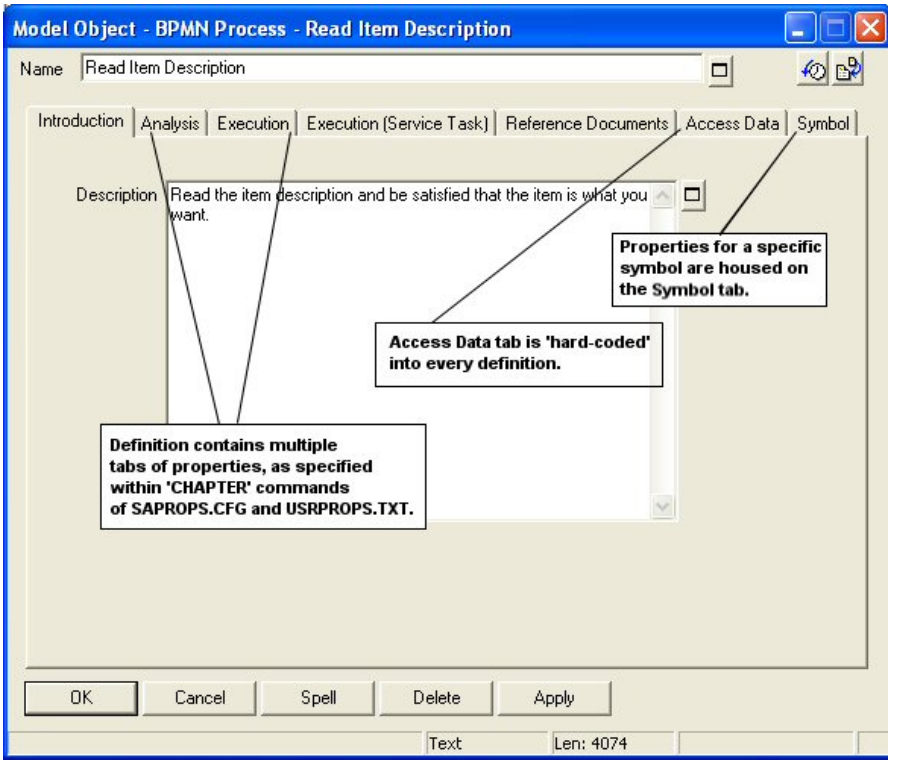
Ein Beispiel für einen *Symboltyp* ist das BPMN-Prozesssymbol. Das BPMN-Prozesssymbol wird in ein Geschäftsprozessdiagramm gezeichnet. Das Diagramm *enthält* Symbole und Symbole sind in einem Diagramm

enthalten. Dies ist ein Beispiel für zwei der zahlreichen Beziehungen innerhalb des Metamodells der Enzyklopädie. Ein Beispiel für einen *Definitionstyp* ist die BPMN-Prozessdefinition. Die BPMN-Prozessdefinition wird grafisch durch ein BPMN-Prozesssymbol dargestellt. Die meisten Definitionen werden durch ein Symbol dargestellt, jedoch nicht alle. Die Definitionstypen "Attribut" und "Methode" etwa werden in Diagrammen nicht durch ein Symbol dargestellt. Sie sind beide in einem Klassendefinitionstyp (einer anderen Beziehung) *enthalten.*

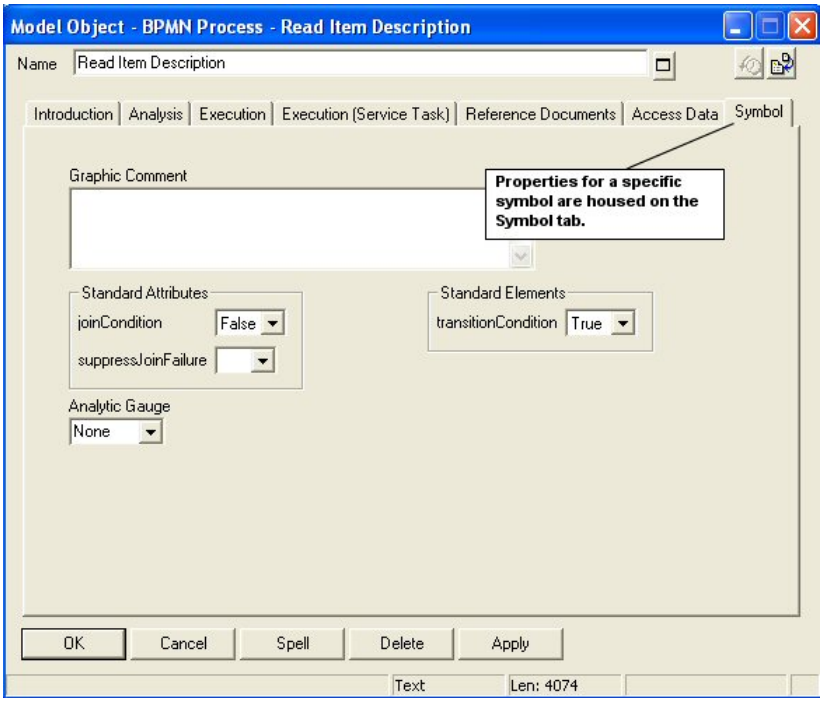
Wie ein Diagrammtyp weist auch jeder Definitionstyp Eigenschaften auf. Wenn Sie eine Definition im Explorer von Rational System Architect öffnen, werden diese Eigenschaften – unterteilt in entsprechende Registerkarten und Gruppen – im Definitionsdialogfenster angezeigt.



Wie Diagramm- und Definitionstypen weist auch jeder Symboltyp Eigenschaften auf. Wenn Sie das zugrunde liegende Dialogfenster eines Symbols öffnen (doppelklicken Sie dazu im Diagrammarbeitsbereich auf ein Symbol, oder klicken Sie mit der rechten Maustaste auf das Symbol, und wählen Sie "Bearbeiten", oder wählen Sie das Symbol aus, und wählen Sie "Bearbeiten, Symboltyp"), sehen Sie die Eigenschaften der zugrunde liegenden Definition, die durch das Symbol dargestellt wird (diese sind mit den Eigenschaften identisch, die angezeigt werden, wenn Sie die Definition in Explorer öffnen). **Zusätzlich** wird eine Symbolregisterkarte angezeigt.



In der Symbolregisterkarte sind die spezifischen Eigenschaften des Symbols enthalten.



Jedes Symbol, das Sie in ein Diagramm zeichnen, ist eine separate Instanz, die auf ein und dieselbe Definition verweist. Sie können also ein Prozesssymbol mit dem Namen "Artikelbezeichnung lesen" in ein Geschäftsprozessdiagramm zeichnen (und für das Symbol die Farbe Rot wählen) und ein weiteres Prozesssymbol mit dem Namen "Artikelbezeichnung lesen" in ein anderes Geschäftsprozessdiagramm zeichnen (und für das Symbol die Farbe Grün wählen). Wenn Sie nun eine Änderung in der Definition "Artikelbezeichnung lesen" vornehmen, indem Sie beispielsweise einen Begriff in deren Beschreibungseigenschaft einfügen, dann wirkt sich diese Änderung gleichermaßen beim Öffnen der Definition des grünen oder des roten Symbols mit dem Namen "Artikelbezeichnung lesen" aus. Es handelt sich also um zwei verschiedene Symbole, die über ein und dieselbe zugrunde liegende Definition verfügen.

Beziehungen zwischen Diagrammtypen, Symboltypen und Definitionstypen (und untereinander) können äußerst komplex sein. In Rational System Architect beispielsweise gehört ein Klassendiagramm zu dem Paket, in dem es erstellt wurde. Zwischen einer Klasse und einem Paket besteht eine BELONGS TO-Beziehung. Darüber hinaus steht eine Klasse in einer KEYED TO-Beziehung zu dem Paket, dem sie angehört. Die KEYED TO-Beziehung bewirkt die Eindeutigkeit des Namensbereichs der Klasse. So hat eine Klasse mit dem Namen "Person" im Paket "Human_Resources" einen ganz anderen Inhalt als die Klasse "Person" im Paket "Hotel_Reservation". Die KEYED BY-Beziehung ist eine weitere Form der Beziehung zwischen einer Klasse und einem Paket. Analog gehört eine Methode zu einer Klasse, die wiederum zu einem Paket gehört. Eine Methode wird ebenfalls durch ihre Klasse angegeben, die wiederum durch ihr Paket angegeben wird. Darüber hinaus kann der Benutzer für Klassensymbole auf diesem Klassendiagramm auch untergeordnete Diagramme erstellen (z. B. ein Statusdiagramm). In diesem Fall ist das Statusdiagramm dem Klassensymbol untergeordnet und stellt damit eine weitere Art der Beziehung dar.

Vorgehensweise zum Ändern des Metamodells

Im Lieferumfang von Rational System Architect ist ein voreingestelltes Metamodell mit Diagrammen, Symbolen, Definitionen und Beziehungen enthalten. Sie können das Metamodell so übernehmen oder es nach Ihren eigenen Wünschen erweitern oder anpassen. Im Rahmen dieser Anpassung können Sie entweder bereits vorhandene Inhalte ändern oder eigene Diagramm-, Symbol- und Definitionstypen hinzufügen.

Physische Form des Metamodells einer Enzyklopädie: Die Dateien SAPROPS.CFG und USRPROPS.TXT

Das Metamodell aller Enzyklopädien in Rational System Architect wird durch zwei Dateien bestimmt: Die SAPROPS.CFG-Datei (die Eigenschaftskonfigurationsdatei von System Architect) und die USRPROPS.TXT-Datei (die Eigenschaftendatei des Benutzers). Diese beiden Dateien sind in der Dateientabelle jeder Enzyklopädie enthalten.

Die SAPROPS.CFG-Datei weist das Standardmetamodell auf, das von IBM für jede Enzyklopädie, die mit einer bestimmten Version des Produkts verwendet wird, angegeben wird. Bei der USRPROPS.TXT-Datei handelt es sich – abgesehen von einigen Kommentaranweisungen (REM oder Erinnerung) – standardmäßig um eine leere Datei. Der Benutzer kann der USRPROPS.TXT-Datei Code hinzufügen, um das Metamodell zu ändern.

Beim Öffnen einer Enzyklopädie führt Rational System Architect eine syntaktische Analyse der SAPROPS.CFG-Datei und dann der USRPROPS.TXT-Datei durch und erstellt anschließend eine SAPROPS.BIN-Datei. Bei der Erstellung der SAPROPS.BIN-Datei überschreibt bzw. ergänzt der in der USRPROPS.TXT-Datei angegebene Inhalt die SAPROPS.CFG-Spezifikation. Die SAPROPS.BIN-Datei ist die Datei, die für die Darstellung des Metamodells für den Benutzer herangezogen wird.

Es gibt einige wichtige Elemente des Metamodells, die Sie nicht mithilfe der USRPROPS.TXT-Datei in der SAPROPS.CFG-Datei überschreiben können:

- Sie können keine LIST- und keine LISTONLY-Verweise entfernen, die in der SAPROPS.CFG-Datei angegeben wurden. Sie können jedoch den Text ändern, der in der Liste oder dem Listenfeld angezeigt wird.
- Sie können keine Beschriftungen entfernen, die in der SAPROPS.CFG-Datei angegeben wurden. Sie können jedoch den Text ändern, der als Beschriftung angezeigt wird.

Die Masterkopien SAPROPS.CFG und USRPROPS.TXT

Abgesehen von der Dateientabelle einer jeden Enzyklopädie, ist eine Masterkopie der SAPROPS.CFG- und der USRPROPS.TXT-Datei auch im Hauptverzeichnis für ausführbare Dateien von Rational System Architect vorhanden (in der Regel unter <C>:\Programme\IBM\Rational\System Architect Suite\11.3.1\System Architect). Bei der erstmaligen Erstellung einer Enzyklopädie werden die im Hauptverzeichnis für ausführbare Dateien von Rational System Architect vorhandenen Masterkopien der SAPROPS.CFG- und der USRPROPS.TXT-Datei automatisch in der Dateientabelle der Enzyklopädie abgelegt. Wenn Sie also den Inhalt der USRPROPS.TXT-Datei im Hauptverzeichnis von Rational System Architect ändern, ändern Sie damit das Metamodell für alle weiteren Enzyklopädien, die Sie erstellen. Aus diesem Grund stellen die meisten Benutzer sicher, dass die im Hauptverzeichnis für ausführbare Dateien von Rational System Architect enthaltene, übergeordnete USRPROPS.TXT-Datei über alle für ihre jeweiligen Unternehmens- und Projektstandards erforderlichen Eigenschaften verfügt.

Zunächst ist die übergeordnete USRPROPS.TXT-Datei nahezu leer. Sie enthält lediglich einige Kommentare im Kopfsatz der Datei, denen ein **REM**-Befehl (Erinnerung oder Kommentar) vorangestellt ist.

CONFIG.PRP-Datei

Rational System Architect stellt eine dritte Datei, die CONFIG.PRP-Datei, bereit. Diese ist eine genaue Kopie der SAPROPS.CFG-Datei. Die CONFIG.PRP-Datei befindet sich im Hauptverzeichnis für ausführbare Dateien von Rational System Architect (in der Regel unter <C>:\Programme\IBM\Rational\System Architect Suite\11.3.1\System Architect). Mithilfe der CONFIG.PRP-Datei können Sie die ebenfalls in

der SAPROPS.CFG-Datei enthaltenen Befehle und Eigenschaften anzeigen, ausschneiden und kopieren, ohne jedoch befürchten zu müssen, die SAPROPS.CFG-Datei selbst versehentlich zu zerstören. Es bietet sich an, Befehle aus der CONFIG.PRP-Datei auszuschneiden oder zu kopieren und in die USRPROPS.TXT-Datei einzufügen und erforderliche Änderungen dort vorzunehmen.

Auswählen der Diagramm- und Eigenschaftengruppen für eine Enzyklopädie

Abgesehen von der Möglichkeit, das Metamodell mithilfe der USRPROPS.TXT-Datei zu ändern, können Sie über das Konfigurationsdialogfenster von Rational System Architect auch auswählen, welche Diagramm- und Eigenschaftengruppen für eine Enzyklopädie zu einem bestimmten Zeitpunkt aktiviert werden (Aufrufen des Konfigurationsdialogfensters über "Tools, Methodenunterstützung anpassen, Enzyklopädiekonfiguration").

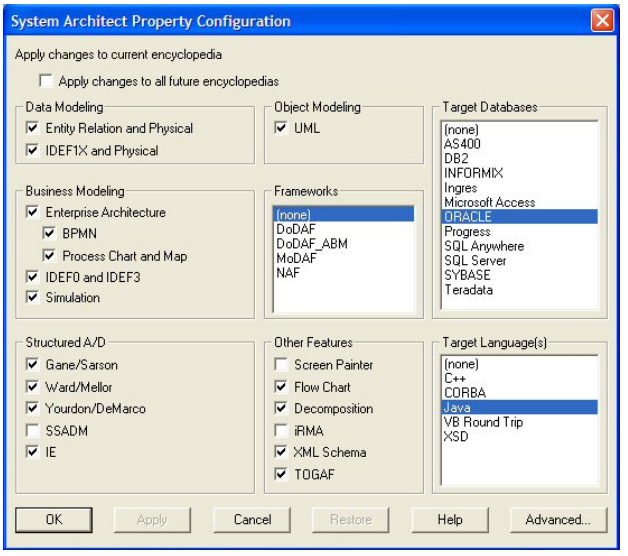


Abbildung 1-1. Dialogfenster für Eigenschaftskonfiguration: Auswählen der Diagrammtypen und anderer nützlicher Diagramme für die jeweilige Enzyklopädie

Sie können Diagramm- und Eigenschaftengruppen aktivieren und inaktivieren und über die Schaltfläche "Erweitert" des Dialogfensters weitere Feineinstellungen vornehmen und bestimmen, welche Diagramm- und Eigenschaftengruppen in einer Enzyklopädie aktiv sind.

SADECLAR.CFG-Datei

Die Optionen, die Sie im Dialogfenster für die Eigenschaftskonfiguration auswählen, wirken sich unmittelbar auf den Inhalt der SADECLAR.CFG-Datei aus, die sich in der

Dateientabelle einer jeden Enzyklopädie befindet. Auf diese Datei wiederum wird in der SAPROPS.CFG- und der USRPROPS.TXT-Datei über #IFDEF-Anweisungen verwiesen (Anmerkung: kein Leerzeichen zwischen '#' und 'IF'). Wenn beispielsweise die SAPROPS.CFG-Datei #IFDEF-Anweisungen für UML aufweist und die UML-Modellierungsmethode im obigen Dialogfenster (und damit in der SADECLAR.CFG-Datei) aktiviert wurde, sorgen die #IFDEF-Anweisungen in der SAPROPS.CFG-Datei wiederum dafür, dass die entsprechenden Eigenschaften für UML-Diagramme aktiviert bzw. inaktiviert werden.

Wie aus der Abbildung unten hervorgeht, werden anhand der Optionen im Dialogfenster für die Eigenschaftskonfiguration (durch die Optionen in der SADECLAR.CFG-Datei aktiviert bzw. inaktiviert werden) die für die Enzyklopädie zu verwendenden Eigenschaften in der SAPROPS.CFG-Datei gefiltert. Benutzermodifikationen, die in der USRPROPS.TXT-Datei vorgenommen werden, werden am Anfang der gefilterten SAPROPS.CFG-Datei syntaktisch analysiert, um eine SAPROPS.BIN-Datei zu erstellen. Diese stellt das Metamodell für eine Enzyklopädie bereit, sobald Rational System Architect ausgeführt wird. Jedes Mal, wenn Sie ein Eigenschafts- oder Definitionsdialogfenster aufrufen oder einen Bericht ausführen, greift Rational System Architect auf die SAPROPS.BIN-Datei zu, um die Eigenschaften des jeweiligen Modellelements abzurufen, das Sie gerade definieren.

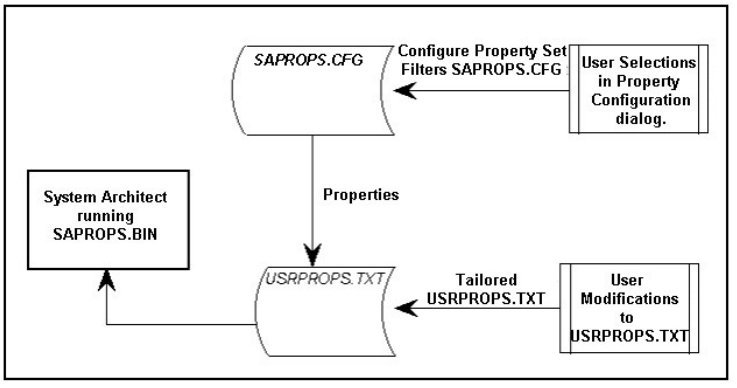


Abbildung 1-2. Zusammenhang zwischen der SAPROPS.CFG- und der USRPROPS.TXT-Datei und den Benutzereinstellungen für Diagramme, Eigenschaften und Modellierungsverfahren

Sie können die SADECLAR.CFG-Datei aus der Dateientabelle einer Enzyklopädie exportieren und in einem beliebigen Texteditor öffnen, um die spezifischen Eigenschaftengruppen anzuzeigen, die für die Aktivierung bzw. Inaktivierung in #IFDEF-Anweisungen der USRPROPS.TXT-Datei zur Verfügung stehen.

Einige entsprechen nicht genau den Begriffen und Bezeichnungen im Dialogfenster für die Eigenschaftskonfiguration. So heißt die Option "Unternehmensarchitektur" (Enterprise Architecture) beispielsweise in der SADECLAR.CFG-Datei "Unternehmen" (Business Enterprise). Folglich wäre die #IFDEF-Anweisung "Enterprise Architecture" in der USRPROPS.TXT-Datei ohne Bedeutung und würde einen Parsing-Fehler hervorrufen. Die Anweisung muss richtig "Unternehmen" (Business Enterprise) lauten.

Abbildung 1-3. Der Inhalt der SADECLAR-Datei wird für #IFDEF-(In-) Aktivierungen in der USRPROPS.TXT-Datei herangezogen.

```
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

Ändern des Metamodells mithilfe der USRPROPS.TXT-Datei

Einführung

Dieses Kapitel beschreibt die Theorien und Mechanismen, die dem erweiterbaren Metamodell von Rational System Architect zugrundeliegen.

Aufrufen und Bearbeiten der USRPROPS.TXT-Datei

Die USRPROPS.TXT-Datei kann in jedem beliebigen Texteditor bearbeitet werden. Die einzige Voraussetzung ist, dass sie als TEXT-Datei gespeichert wird.

Aufrufen der übergeordneten USRPROPS.TXT-Datei

Wie bereits an früherer Stelle dieses Kapitels erwähnt, wird die übergeordnete USRPROPS.TXT-Datei automatisch in allen neuen Enzyklopädien abgelegt, die Sie erstellen. Viele Organisationen ändern die übergeordnete USRPROPS.TXT-Datei so, dass alle neuen Enzyklopädien dieselben Metamodellerweiterungen aufweisen. So ändern Sie die übergeordnete USRPROPS.TXT-Datei:

- Wählen Sie "Tools, Benutzereigenschaften anpassen, USRPROPS.TXT (übergeordnete) bearbeiten" oder
- Navigieren Sie zum Verzeichnis <C>:\Programme\IBM\Rational\System Architect Suite\11.3.1\System Architect, und öffnen Sie die dort befindliche USRPROPS.TXT-Datei.

Aufrufen der USRPROPS.TXT-Datei einer Enzyklopädie

Die USRPROPS.TXT-Datei einer Enzyklopädie befindet sich in der SQL Server-Datenbank der Enzyklopädie. Damit Sie sie bearbeiten können, müssen Sie sie zunächst aus der Dateientabelle der Datenbank exportieren. Nach dem Bearbeiten der Datei müssen Sie sie in die Dateientabelle der Datenbank zurückimportieren und Ihre Enzyklopädie erneut öffnen (damit die SAPROPS.CFG- und die USRPROPS.TXT-Datei syntaktisch analysiert werden können).

Es gibt mehrere Möglichkeiten, um die USRPROPS.TXT-Datei einer Enzyklopädie aufzurufen:

- Sie können entweder die systemeigene Funktion von Rational System Architect zum Exportieren/ Importieren der USRPROPS.TXT-Datei verwenden (über "Tools, Benutzereigenschaften anpassen, USRPROPS.TXT (Enzyklopädie) exportieren") oder

- Sie verwenden das Dienstprogramm **Enzyklopädie-dateimanager** von Rational System Architect (über "Tools, Enzyklopädiedateimanager") oder
- Sie verwenden **SAEM** (außerhalb von Rational System Architect; wählen Sie dazu "Start, Programme, IBM Rational, IBM Rational Lifecycle Solutions Tools, IBM Rational System Architect 11.3.1, SAEM", und lesen Sie die SAEM-Hilfedatei).

Verwenden der systemeigenen Funktion von Rational System Architect zum Exportieren/Importieren der USRPROPS.TXT-Datei:

Gehen Sie folgendermaßen vor, um die USRPROPS.TXT-Datei mithilfe der systemeigenen Export-/Importfunktion von Rational System Architect zu bearbeiten:

1. Wählen Sie "Tools, Benutzereigenschaften anpassen, USRPROPS.TXT (Enzyklopädie) exportieren".
2. Wählen Sie im angezeigten Dialogfenster **Benutzer-eigenschaften exportieren** ein Verzeichnis aus, in das Sie die USRPROPS.TXT-Datei exportieren möchten. Klicken Sie auf die Schaltfläche "Speichern". Die USRPROPS.TXT-Datei wird im angegebenen Verzeichnis gespeichert und automatisch in Notepad geöffnet.
3. Nachdem Sie die Datei bearbeitet haben, wählen Sie "Tools, Benutzereigenschaften anpassen, USRPROPS.TXT (Enzyklopädie) importieren", um die modifizierte USRPROPS.TXT-Datei in die Dateientabelle der Datenbank der Enzyklopädie zurückzuimportieren.
4. Öffnen Sie die Enzyklopädie erneut, um eine Syntaxanalyse der SAPROPS.CFG-Datei und der modifizierten USRPROPS.TXT-Datei durchzuführen.

Verwenden des Enzyklopädiedateimanagers:

Gehen Sie folgendermaßen vor, um die USRPROPS.TXT-Datei mithilfe des Enzyklopädiedateimanagers zu bearbeiten:

1. Wählen Sie "Tools, Enzyklopädiedateimanager".

2. Stellen Sie im Dialogfenster des Enzyklopädiedatei-managers sicher, dass die Option "Export" in der linken unteren Ecke aktiviert ist. Wählen Sie in der Liste "Datei zum Exportieren auswählen" die USRPROPS.TXT-Datei aus, und geben Sie über die Schaltfläche '…' der Eigenschaft "Ausgewählte Datei exportieren in" an, in welches Verzeichnis Sie die Datei exportieren möchten.
3. Ändern Sie die Datei in einem Texteditor, und importieren Sie die Datei anschließend mithilfe des Enzyklopädiedateimanagers zurück in die Dateientabelle der Enzyklopädiedatenbank.
4. Öffnen Sie die Enzyklopädie erneut, um eine Syntaxanalyse der SAPROPS.CFG-Datei und der modifizierten USRPROPS.TXT-Datei durchzuführen.

Verwenden von SAEM:

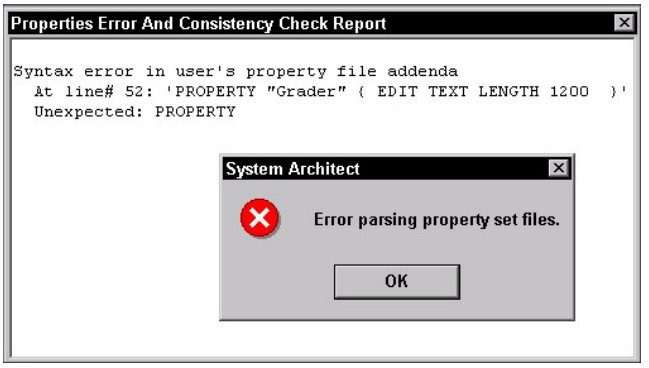
Sie können die USRPROPS.TXT-Datei auch mithilfe von SAEM aufrufen und bearbeiten. Lesen Sie in der SAEM-Hilfedatei die entsprechenden Anweisungen nach, um die Verbindung zu einem Server herzustellen, eine Datenbank auszuwählen und die Dateien aus der Datenbank zu exportieren bzw. in die Datenbank zu importieren.

Neuladen der Eigenschafts-dateien

Wie bereits bei den obigen Schritten erwähnt, müssen Sie die Enzyklopädie erneut mit Rational System Architect öffnen, nachdem Sie eine modifizierte USRPROPS.TXT-Datei in eine Enzyklopädie zurückimportiert haben. Durch das erneute Öffnen der Enzyklopädie werden die SAPROPS.CFG- und die USRPROP.TXT-Datei syntaktisch analysiert und eine binäre SAPROPS.BIN-Datei erstellt, die für die Darstellung des Metamodells verwendet wird. Werden dabei keine Fehler festgestellt, werden die am Metamodell vorgenommenen Änderungen sofort übernommen.

Stellt Rational System Architect bei der Syntaxanalyse der USRPROPS.TXT-Datei einen Fehler im Code fest, gibt das System entweder eine Warnung oder eine Fehlernachricht aus. Rational System Architect öffnet zwar im Anschluss an eine Warnung die Enzyklopädie, **nicht** aber wenn ein Fehler aufgetreten ist. Es wird in etwa folgende Nachricht angezeigt:

Abbildung 2-1. Fehlerdialog Eigenschaftendatei



Bei Auftreten eines Fehlers können Sie nicht mit der systemeigenen Funktion von Rational System Architect zum Exportieren/Importieren der USRPROPS.TXT-Datei auf die fehlerhafte USRPROPS.TXT-Datei zugreifen (wenn Sie "Tools, Benutzereigenschaften anpassen" auswählen, ist die Option "USRPROPS.TXT (Enzyklopädie) exportieren" ausgegraut).

Wenn Sie die USRPROPS.TXT-Datei im Anschluss an einen Fehler aufrufen und bearbeiten möchten, müssen Sie entweder den Enzyklopädiedateimanager von Rational System Architect (über "Tools, Enzyklopädiedateimanager") oder SAEM verwenden (wählen Sie außerhalb von Rational System Architect "Start, Programme, IBM Rational, IBM Rational Lifecycle Solutions Tools, IBM Rational System Architect 11.3.1, SAEM", und rufen Sie die SAEM-Hilfedatei auf).

Zusammensetzung und Syntax

Wie bei den meisten Programmiersprachen setzt sich die Sprachsyntax der USRPROPS.TXT-Datei aus mehreren Zeichenfolgen zusammen. Als Trennung zwischen den einzelnen Zeichenfolgen ist mindestens ein *Leerzeichen* erforderlich (dazu zählen Tabulatoren, Kommata, Zeilenumbrüche und gewöhnliche Leerzeichen). Wenn mehrere Leerzeichen unmittelbar hintereinander folgen, z. B. ein Zeilenumbruch und ein Tabulatorzeichen, werden die Zeichen zusammengefasst und als ein Zeichen interpretiert.

Wenn eine Zeichenfolge ein oder mehrere eingebettete Leerzeichen enthält, müssen Sie die Zeichenfolge in doppelte Anführungszeichen setzen. Schreiben Sie beispielsweise **"Data Flow"** und nicht **Data Flow**.

Schlüsselwörter

Die Sprache innerhalb der USRPROPS.TXT-Datei basiert auf einem bestimmten Satz aus ***Schlüsselwörtern***. Je nach Platzierung wird ein Schlüsselwort entweder als ***Befehl*** oder als ***Argument*** interpretiert. Die in der USRPROPS.TXT-Datei zugelassenen Schlüsselwörter sind in **Kapitel 3, USRPROPS.TXT-Schlüsselwörter** aufgeführt.

Groß- und Kleinschreibung von Schlüsselwörtern

Bei Schlüsselwörtern in der USRPROPS.TXT-Datei ist die Groß- und Kleinschreibung ***nicht*** relevant. Sie können also Großbuchstaben, Kleinbuchstaben oder beides zusammen verwenden. In diesem Handbuch und in der USRPROPS.TXT-Musterdatei wurden die Befehle und Schlüsselwörter lediglich zur besseren Lesbarkeit in Großbuchstaben geschrieben. Beispiele für Befehle:

BEGIN oder Begin oder BegiN
EDIT oder Edit
LIST oder List oder LiST
LISTONLY oder Listonly oder ListOnly
RENAME oder Rename oder ReName usw.

Befehle

Befehle sind *immer* Schlüsselwörter und stehen *immer* am Anfang einer neuen Wortfolge. Bei der Syntaxanalyse der USRPROPS.TXT-Datei erkennt Rational System Architect, dass es sich bei der ersten Zeichenfolge der Datei um einen

gültigen Schlüsselwortbefehl handelt. Jeder Befehl muss von einer bekannten Anzahl von Argumentzeichenfolgen (keine, eine oder mehrere) gefolgt werden. Danach muss ein weiterer Befehl erkannt werden.

Argumente

Zeichenfolgen im Anschluss an einen Befehl werden als "Argumente" bezeichnet. Einige Argumente sind Schlüsselwörter. Andere wiederum setzen sich aus Textzeichenfolgen zusammen, die den Namen von Diagrammen, Symbolen, Definitionen, Eigenschaften, Listenwerten, Beschriftungen und Hilfezeichenfolgen usw. darstellen, die in den nachfolgenden Dialogfenstern angezeigt werden. Beispiele:

- LIST **"Processor Scheduling**"

  "Processor Scheduling" ist kein Schlüsselwort. Der Begriff wird als Argument für den obigen Ausdruck verwendet.

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

  "KEY" ist ein Schlüsselwort. Der Begriff wird als Argument für den obigen Ausdruck verwendet.

Groß- und Kleinschreibung bei Argumenten, die als Textzeichenfolge fungieren

Wie bereits erwähnt, ist die Groß- und Kleinschreibung bei Schlüsselwörtern ***nicht*** relevant. Bei Argumenten allerdings, die als Textzeichenfolge fungieren, ***muss*** die Groß- und Kleinschreibung beachtet werden. Wenn beispielsweise das obige Argument LIST "Processor Scheduling" verwendet werden soll, müssen Verweise auf diese Liste in der SAPROPS.CFG-Datei bzw. in der USRPROPS.TXT-Datei exakt gleich geschrieben werden. Wird beispielsweise folgende Liste festgelegt:

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

Dann muss ein gültiger Verweis auf diese Liste exakt gleich geschrieben werden.

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

Bei nachfolgender Syntax wird jedoch die Fehlernachricht 'List "PROCESSOR SCHEDULING" not Found.' ausgegeben.

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

Gleichermaßen müssen Eigenschaften, auf die in Berichten verwiesen wird, dieselbe Schreibweise und Groß- und Kleinschreibung aufweisen, wie der Eintrag in der SAPROPS.CFG-Datei und/oder der USRPROPS.TXT-Datei.

Gruppieren von Befehlen zur Erstellung von Modellierungs-elementen

Mithilfe von geschweiften Klammern **{ }** oder BEGIN- und END-Befehlen können Befehle gruppiert werden, um Modellierungselemente zu erstellen.

Diagramme, Symbole und Definitionen

Das Repository von Rational System Architect unterstützt drei wesentliche Modellierungselemente, die manchmal auch als *Verzeichnisklassen* bezeichnet werden, nämlich ***Diagramme***, ***Symbole*** (die in Diagrammen enthalten sind) und ***Definitionen*** (die durch Symbole dargestellt sein können, aber nicht müssen). Die Struktur BEGIN .. END bzw. die geschweiften Klammern { } dienen der inhaltlichen Festlegung dieser Modellierungselemente. Beispiel:

Diagram "Name des Diagrammtyps**"**
{
[Inhalt]
}

Symbol "Name des Symboltyps**"**
{
[Inhalt]
}

Definition "Name des Definitionstyps**"**
{
[Inhalt]
}

oder

Diagram "Name des Diagrammtyps**"**
BEGIN
[Inhalt]
END

usw.

Eigenschaften

Der Inhalt der Modellierungselemente setzt sich aus Eigenschafts- und Layoutbefehlen zusammen. Die Struktur BEGIN .. END bzw. die geschweiften Klammern { } dienen der Gruppierung von Eigenschaftsbefehlen. Beispiel:

```
Definition "Name des Definitionstyps"
{
PROPERTY { [Bestimmung der Eigenschaft] }
PROPERTY { [Bestimmung der Eigenschaft] }
…
}
```

Bei bestimmten Schlüsselwörtern, durch die Klauseln innerhalb einer Eigenschaft erstellt werden, müssen außerdem geschweifte Klammern verwendet werden, um die Argumente des Befehls, z. B. des KEYED BY-Befehls, abzugrenzen.

```
Definition "Name des Definitionstyps"
{
PROPERTY { [Bestimmung der Eigenschaft] KEYED BY {
[Klausel] } }
…
}
```

Layout

Für den LAYOUT-Befehl sind ebenfalls geschweifte Klammern bzw. die Anweisung BEGIN .. END erforderlich.

```
Definition "Name des Definitionstyps"
{
LAYOUT { [Bestimmung des Layouts] }
PROPERTY { [Bestimmung der Eigenschaft] }
PROPERTY { [Bestimmung der Eigenschaft] }
…
}
```

Kapitel

Die Eigenschaften in einem Dialogfenster können darüberhinaus in Form von Registerkarten und Gruppen zusammengefasst werden. Registerkarten werden über den CHAPTER-Befehl festgelegt. Dieser Befehl ***darf keine*** geschweiften Klammern oder die Anweisung BEGIN .. END enthalten. Er gruppiert lediglich die darunter angesiedelten Eigenschaften in eine Spezifikation innerhalb einer Registerkarte (innerhalb des sich ergebenden Dialogfensters) bis zum nächsten CHAPTER-Befehl in der Spezifikation.

```
Definition "Name des Definitionstyps"
{
LAYOUT { [Bestimmung des Layouts] }
```

```
CHAPTER "First Tab"
PROPERTY { [Bestimmung der Eigenschaft]  }
PROPERTY { [Bestimmung der Eigenschaft]  }
…
CHAPTER "Second Tab"
PROPERTY { [Bestimmung der Eigenschaft]  }
PROPERTY { [Bestimmung der Eigenschaft]  }
…
}
```

Gruppen

Im Gegensatz zum CHAPTER-Befehl sind bei Gruppen geschweifte Klammern oder die Anweisung BEGIN .. END ***erforderlich***.

```
Definition "Name des Definitionstyps"
{
LAYOUT { [Bestimmung des Layouts] }
CHAPTER "First Tab"
PROPERTY { [Bestimmung der Eigenschaft]  }
PROPERTY { [Bestimmung der Eigenschaft]  }
GROUP "Zusammengehörige Elemente"
{
PROPERTY { [Bestimmung der Eigenschaft]  }
PROPERTY { [Bestimmung der Eigenschaft]  }
}
…
}
```

Listen

Sie können in einer Enzyklopädie auch Listen angeben. Dabei kann es sich um voreingestellte Listen mit Textwerten handeln oder um Listen mit Definitionen, die Sie beim Modellieren erstellen. Listen mit Definitionen, die Sie während des Modellierens erstellen, werden mithilfe der Befehle ONEOF, LISTOF und EXPRESSIONOF innerhalb einer Eigenschaftsanweisung generiert und im weiteren Verlauf dieser Dokumentation näher erörtert. Textlisten werden durch Angeben der Listenwerte als separate Listenanweisung erstellt, wobei die Werte zwischen geschweiften Klammern bzw. innerhalb der Anweisung BEGIN.. END platziert werden. Die LIST-Anweisung wird normalerweise an den Anfang der USRPROPS.TXT-Datei platziert. Auf sie wird innerhalb der jeweiligen Eigenschafts-spezifikation eines Diagramms, eines Symbols oder einer Definition verwiesen.

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

Anmerkung zur Syntax

Einrückungen und Zeilenumbrüche dienen ausschließlich der besseren Lesbarkeit und sind für den USRPROPS.TXT-Prozessor insoweit von Bedeutung, als sie zwischen den einzelnen Zeichenfolgen als Trennungsleerzeichen fungieren. Das obige Beispiel könnte wie folgt dargestellt werden:

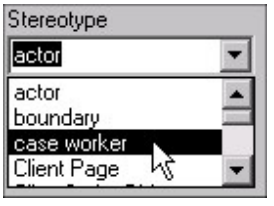
```
LIST "List of Things"   {    VALUE One    VALUE
Two    VALUE "Two and a Half"    VALUE }
```

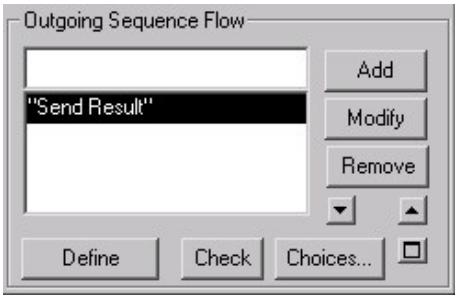
Wenngleich dieses Format die Zwecke von Rational System Architect vollends erfüllt, ist es für die USRPROPS.TXT-Datei wahrscheinlich eher ungeeignet und sollte daher vermieden werden.

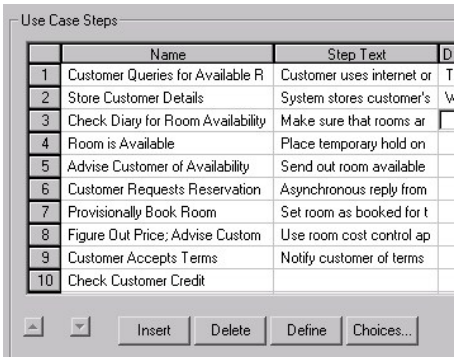
Dialogsteuerelemente

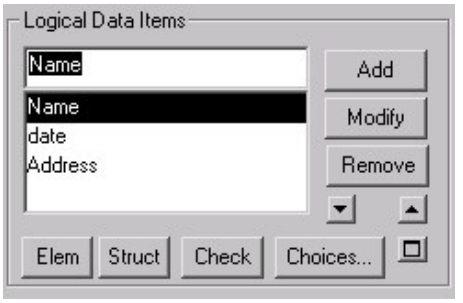
Die Tabelle unten beschreibt Dialogsteuerelemente, die über entsprechende Befehle in der USRPROPS.TXT-Datei erstellt werden können.

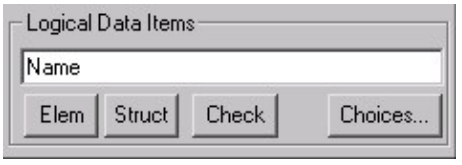
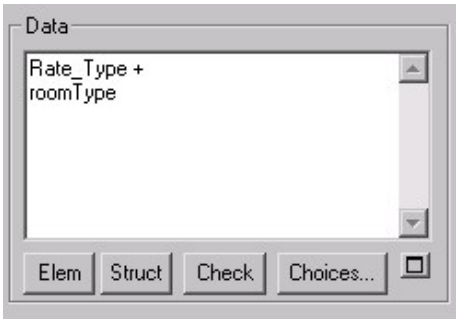
Tabelle 2-2. Über Eigenschafts-ausdrücke generierte Steuerelemente

Argumenttyp	Generiertes Steuerelement
LIST-Befehl für weniger als fünf Werte	Gruppenfeld mit einem **Optionsfeld** pro Wert:  **Wichtige Anmerkung:** Über den LISTONLYCOMBO-Befehl können Sie die Erzeugung einer Liste mit weniger als fünf Werten als Dropdown-Liste erzwingen. Weitere Informationen zu diesem Schlüsselwort finden Sie in Kapitel 3.
LIST-Befehl für fünf oder mehr Werte	Dropdown-Listenfeld: 

BOOLESCH	Kontrollkästchen. Zutreffende Werte werden durch ein Häkchen dargestellt, bei nicht zutreffenden Werten ist das Feld leer. Die unten aufgeführte Eigenschaft "Virtual" beispielsweise wird folgendermaßen als boolescher Wert dargestellt: PROPERTY "Virtual" { EDIT **BOOLEAN** LENGTH 1 DEFAULT "F" } 
LISTOF "[Definition oder Diagrammtyp]"	Gruppe mit Dropdown-Listenfeld mit den Schaltflächen **Neu**, **Hinzufügen**, **Entfernen**, **Pfeil nach unten** und **Pfeil nach oben** seitlich und den drei Schaltflächen **Definieren**, **Überprüfen** und **Optionen** unten. 
ONEOF "[Definition oder Diagrammtyp]"	Gruppe mit Textfeld und den drei Schaltflächen **Definieren**, **Überprüfen** und **Optionen** 

ASGRID	Dieser Befehl wird entweder mit den Befehlen LISTOF oder ONEOF verwendet, um ein Raster mit Werten darzustellen. Beispiel: PROPERTY "Use Case Steps" { EDIT COMPLETE LISTOF "Use Case Step" KEYED BY { "Package", "Use Case Name":Name, Name} **ASGRID** LENGTH 1200 } 
EXPRESSIONOF "[Definition oder Diagrammtyp]"	Nicht möglich.

LISTOF DATA	DATA ist ein besonderer Begriff. Er liefert eine Liste mit Datenelementen und Datenstrukturen in der Enzyklopädie (eine Datenstruktur ist eine Gruppe von Datenelementen). Das Steuerelement LISTOF DATA ist ein ganz besonderes Steuerelement. Es handelt sich dabei um eine Gruppe mit einem Dropdown-Listenfeld mit den seitlichen Schaltflächen **Neu**, **Hinzufügen**, **Entfernen**, **Pfeil nach unten** und **Pfeil nach oben** und den vier Schaltflächen **Element**, **Struktur**, **Überprüfen** und **Optionen** unten. 

ONEOF DATA	Wie oben bereits erwähnt, handelt es sich bei DATA um einen besonderen Begriff. Er liefert eine Liste mit Datenelementen und Datenstrukturen in der Enzyklopädie. Eine ONEOF DATA-Klausel liefert eine Gruppensteuerung, die aus einem Textfeld in der Mitte, in dem das Datenelement oder die Datenstruktur angegeben werden kann, und den vier Schaltflächen **Element**, **Struktur**, **Überprüfen** und **Optionen** besteht. 
EXPRESSIONOF DATA	Wie oben bereits erwähnt, handelt es sich bei DATA um einen besonderen Begriff. Er liefert eine Liste mit Datenelementen und Datenstrukturen in der Enzyklopädie. EXPRESSIONOF DATA liefert einen Textbereich, in dem die Datenelemente oder Datenstrukturen eingegeben werden können, sowie die vier Schaltflächen **Element**, **Struktur**, **Überprüfen** und **Optionen**. 

Sortieren und Anordnen von USRPROPS.TXT-Änderungen

Die allgemeine Reihenfolge der Abschnitte in der SAPROPS.CFG-Datei sieht wie folgt aus:

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

  LAYOUT-Befehlsunterabschnitt (Standard für das gesamte Definitionsdialogfenster)

  CHAPTER-Befehlsunterabschnitt

  GROUP-Befehlsunterabschnitt

  LAYOUT-Befehlsunterabschnitt

  PROPERTY-Befehlsunterabschnitt

Zwar sind alle Einträge in der USRPROPS.TXT-Datei optional, jedoch sollten Sie eine ähnliche Anordnung wie in der SAPROPS.CFG-Datei einhalten und ggf. den Befehlsabschnitt **RENAME** am Anfang der Datei hinzufügen. Die allgemeine Reihenfolge der Abschnitte in der USRPROPS.TXT-Datei sollte wie folgt aussehen:

- **RENAME**-Befehlsabschnitt (in diesem Abschnitt können Sie Benutzerdiagramme, Benutzersymbole und Benutzerdefinitionen umbenennen, um Ihre eigenen Diagramm-, Symbol- oder Definitionstypen zu erstellen (siehe Seite 2-32)
- **LIST**-Befehlsabschnitt (siehe Seite 2-30)
- **DIAGRAM**-Befehlsabschnitt (siehe Seite 2-61)
- **SYMBOL**-Befehlsabschnitt (siehe Seite 2-63)

- **DEFINITION**-Befehlsabschnitt
  (siehe Seite 2-67)

  CHAPTER-Befehlsunterabschnitt
  (siehe Seite 2-97)

  GROUP-Befehlsunterabschnitt
  (siehe Seite 2-100)

  LAYOUT-Befehlsunterabschnitt
  (siehe Seite 2-100)

  PROPERTY-Befehlsunterabschnitt
  (siehe Seite 2-71)

.

Regeln für das Modifizieren der USRPROPS.TXT-Datei

Beim Modifizieren der USRPROPS.TXT-Datei sind folgende Regeln zu beachten:

1. Einträge in der USRPROPS.TXT-Datei verstehen sich als Ergänzungen oder Ersetzungen für Einträge in der SAPROPS.CFG-Datei.
2. Ein Eintrag in der USRPROPS.TXT-Datei muss mit der entsprechenden LIST-, RENAME-, DIAGRAM-, SYMBOL- oder DEFINITION-Anweisung beginnen.
3. USRPROPS.TXT-Einträge, die eine *Ergänzung* der SAPROPS.CFG-Datei darstellen, werden am Ende des jeweiligen Abschnitts eingefügt. So wird ein LIST-Block, der noch nicht in der SAPROPS.CFG-Datei vorhanden ist, in der Regel nach allen anderen LIST-Blöcken in der Datei SAPROPS.CFG hinzugefügt.
4. Falls kein **CHAPTER**-Befehl enthalten ist, werden USRPROPS.TXT-Einträge am Ende des jeweiligen Dialogfensters eingefügt. So wird beispielsweise eine neue Eigenschaft für eine Klassendefinition nach allen anderen Eigenschaften im Definitions-dialogfenster der Klasse hinzugefügt.

5. Wenn ein **CHAPTER**-Befehl, der bereits in der SAPROPS.CFG-Datei enthalten ist, in die USRPROPS.TXT-Datei eingefügt wird, werden die USRPROPS.TXT-Einträge an das Ende des bereits vorhandenen Kapitels (bzw. der Registerkarte) angehängt.
6. Wenn ein **GROUP**-Befehl, der bereits in der SAPROPS.CFG-Datei enthalten ist, in die USRPROPS.TXT-Datei eingefügt wird, werden die USRPROPS.TXT-Einträge an das Ende der bereits vorhandenen Gruppe angehängt.
7. Der **GROUP**-Befehl generiert ein Gruppenfeld (ein gebräuchliches Windows-Steuerelement), in das alle nachfolgenden Steuerelemente zu platzieren sind. Wenn zu viele Einträge vorgenommen werden, sodass die Größe der Gruppe über die Bildschirmgröße hinausgeht, werden die überzähligen Eigenschaften nicht aufgenommen und nicht angezeigt. Beim Öffnen der Enzyklopädie wird eine entsprechende Warnung angezeigt.
8. Wenn eine Eigenschaft zu einer Gruppe hinzugefügt wird, die hinsichtlich der Eigenschaften in der SAPROPS.CFG-Datei **PLACEMENT**-Befehle aufweist, muss der **PLACEMENT**-Befehl auch für die neuen Eigenschaften verwendet werden, die zur USRPROPS.TXT-Datei hinzugefügt werden.

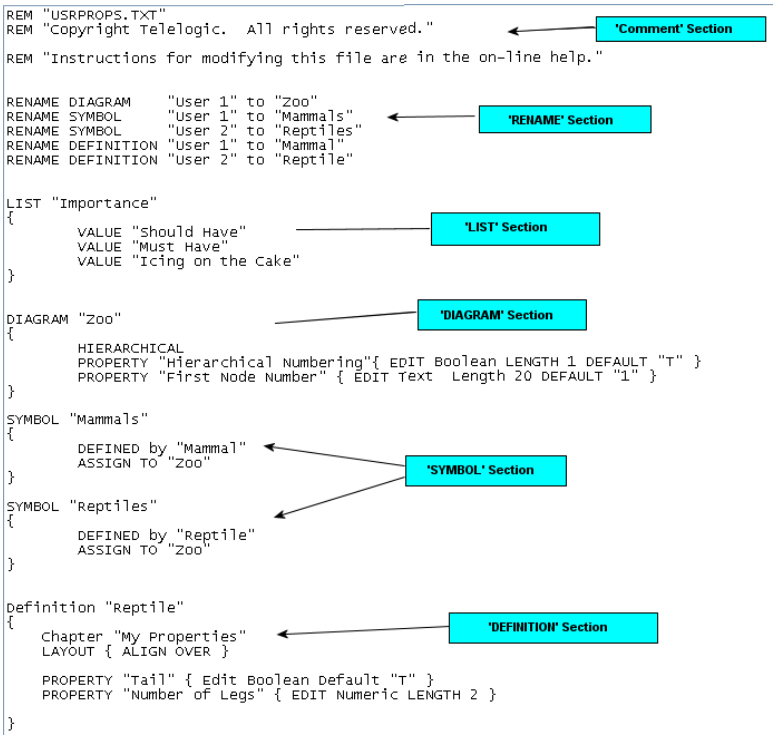
Layout von USRPROPS.TXT-Code

Wenn Sie weder über eine USRPROPS.TXT- noch über eine SAPROPS.CFG-Datei verfügen, so weist dennoch jedes Diagramm, jedes Symbol und jede Definition einen *Namen* und eine *Beschreibung* der Eigenschaft auf. Die Standard-werte für die *Beschreibung* werden später in diesem Abschnitt behandelt. Wie bereits erwähnt, ist der gesamte Text der SAPROPS.CFG-Datei in der CONFIG.PRP-Datei enthalten. Es handelt sich um eine Standard-ASCII-Textdatei,

deren Einträge als Modell für Änderungen und Ergänzungen in der USRPROPS.TXT-Datei verwendet werden können.

Wie Sie den eigentlichen Code in der USRPROPS.TXT-Datei selbst anordnen, bleibt Ihnen überlassen. Wir empfehlen eine Tabulatorstruktur, sodass der Beginn von LIST-, DIAGRAM-, SYMBOL- und DEFINITION-Anweisungen einfacher zu erkennen ist. Es kann sein, dass die Tabulatoren in unterschiedlichen Texteditoren unterschiedlich dargestellt werden. Wenn Sie beispielsweise Microsoft Word als Texteditor verwenden und die USRPROPS.TXT-Datei später in einem anderen Texteditor öffnen, sind die mit Word gesetzten Tabulatoren möglicherweise ganz anders platziert.

Abbildung 2-2. Beispiel für die Darstellung von USRPROPS.TXT-Code



Beispiel für die Vornahme von Änderungen in der USRPROPS.TXT-Datei

In diesem Abschnitt werden wir Änderungen an einer Definition vornehmen, die bereits in der SAPROPS.CFG-Datei vorhanden ist. Der folgende Code ist in der SAPROPS.CFG-Datei vorhanden:

```
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" }
```

Die Abbildung unten zeigt das Dialogfenster **Verzeichnis-objekt** für den obigen Definitionsblock (dieses wird in diesem Handbuch meist als **Definitionsdialogfenster** bezeichnet).

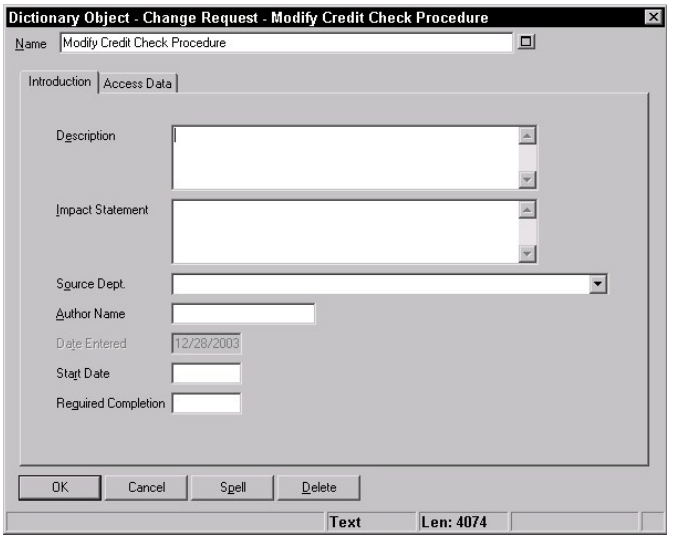


Abbildung 2-3. Definitionsdialogfenster für Änderungsanforderung gemäß Definition in der Eigenschaftengruppendatei der Masterkonfiguration

Hier ist zu beachten, dass die Registerkarte **Einführung** vorhanden ist, obwohl diese von der SAPROPS.CFG-Datei nicht ausdrücklich über den CHAPTER-Befehl aufgerufen wurde. Wenn kein CHAPTER-Befehl angegeben wurde, erstellt Rational System Architect automatisch eine Standardregisterkarte mit dem Titel **Einführung**. Die Registerkarte **Zugriffsdaten** ist in der Software fest codiert und wurde nicht in der SAPROPS.CFG-Datei angegeben.

Vornehmen von Änderungen mithilfe der USRPROPS.TXT-Datei

Änderungen an der Änderungsanforderungsdefinition werden durch Hinzufügen des nachfolgenden Codes in die USRPROPS.TXT-Datei und anschließendes Öffnen der Enzyklopädie vorgenommen.

```
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" }
}
```

Die Tabelle enthält eine Erläuterung zu den einzelnen Zeilen des obigen USRPROPS.TXT-Codes und veranschaulicht den jeweiligen Effekt.

Tabelle 2-1. Effekt von USRPROPS.TXT-Einträgen

USRPROPS.TXT-Eintrag	Effekt
DEFINITION "Change Request" {	Legt eine Änderung an der Definition "Change Request" fest.
LAYOUT { COLS 2 TAB ALIGN LABEL }	Legt eine zweispaltige Anordnung der Eigenschaften unterhalb des LAYOUT-Befehls fest (bis das Ende der Definition oder ein neuer LAYOUT-Befehl erreicht wird).
PROPERTY "Author Name" { LABEL "Client Division" }	Ändert eine bereits vorhandene Eigenschaft (hier die Beschriftung innerhalb des Feldes von *Author Name* in *Client Division*
PROPERTY "Supervising Manager" { EDIT TEXT LENGTH 45 }	Fügt eine neue Eigenschaft zum Dialogfenster hinzu (hier das Textfeld *Supervising Manager*).
PROPERTY "On time" { EDIT BOOLEAN LENGTH 1 DEFAULT "T" } }	Fügt eine neue Eigenschaft zum Dialogfenster hinzu (hier ein Kontrollkästchen, um anzugeben, ob die Änderungsanforderung die Terminvorgaben erfüllt).

Jetzt importieren wir die modifizierte USRPROPS.TXT-Datei in die Rational System Architect-Enzyklopädie und öffnen die Definition einer Änderungsanforderung, um die Änderungen im Dialogfenster anzuzeigen. Die Informationen in der Registerkarte **Einführung** sind jetzt übrigens auf zwei Seiten verteilt.

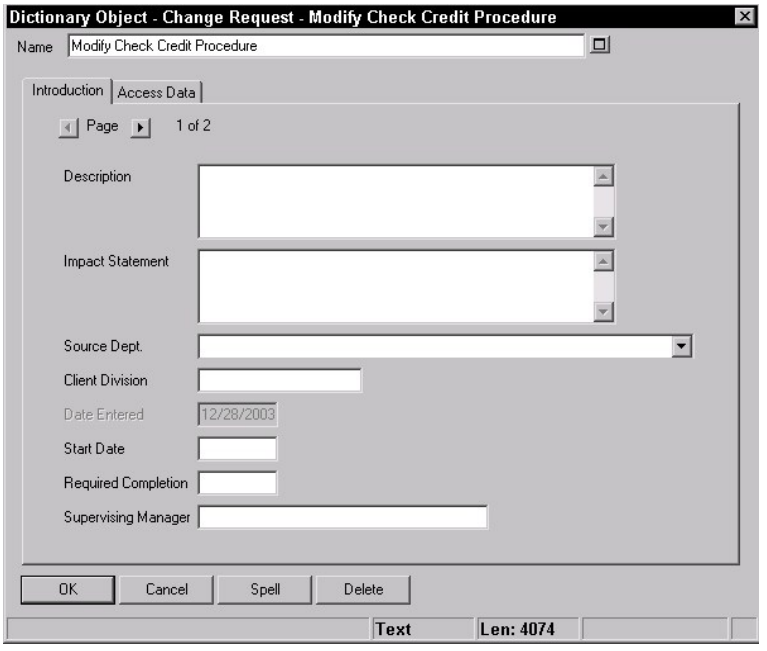
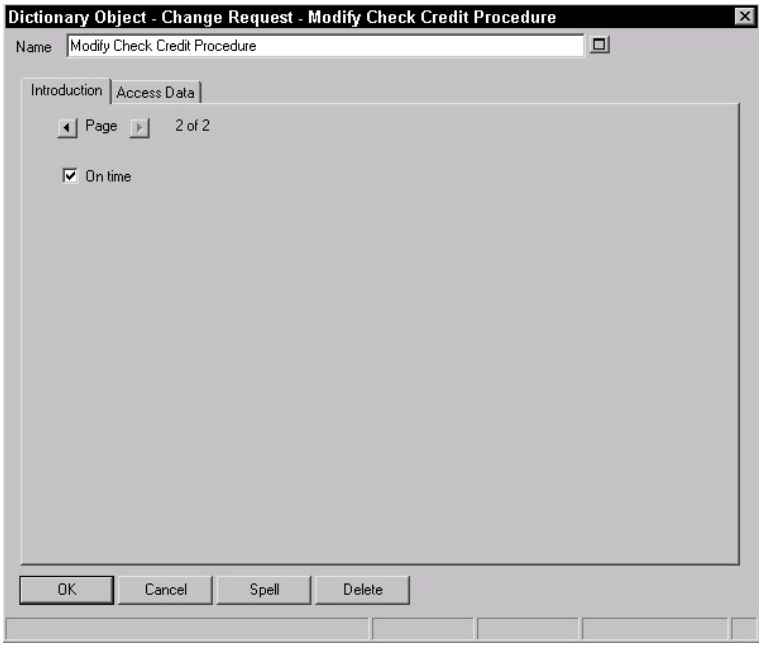


Abbildung 2- 4. Definitionsdialog-fenster für Änderungs-anforderung nach der Modifizierung über die USRPROPS.TXT-Datei



Die Informationen in der Registerkarte **Einführung** sind deshalb auf zwei Seiten verteilt, weil wir zwei neue Eigenschaften hinzugefügt haben. Sie werden am Ende der Definition hinzugefügt (und nicht am Ende der Registerkarte **Zugriffsdaten**. Diese zählt nicht, da sie fest codiert und nicht in der SAPROPS.CFG-Datei enthalten ist).

Einzige Änderung

Wie Sie sehen, haben wir nicht die gesamte, bereits in der SAPROPS.CFG-Datei vorhandene, PROPERTY-Anweisung erneut in die USRPROPS.TXT-Datei eingegeben. Abgesehen von etwaig neu hinzugefügten Anweisungen müssen wir nur die spezifischen Anweisungen eingeben, die wir ändern möchten. Auch bei den Anweisungen, die wir ändern, also neu eingeben, ist nur der Teil der Anweisung hinzuzufügen, der sich tatsächlich ändert. In unserem Beispiel lautete die neu eingegebene (also geänderte) SAPROPS.CFG-Anweisung:

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

In der USRPROPS.TXT-Datei wollten wir lediglich die Beschriftung der Eigenschaft ändern, also war nur Folgendes einzugeben:

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

Die Länge der Eigenschaft und die Tatsache, dass es sich um Text handelt (und nicht um eine numerische oder boolesche Darstellung), bleiben davon unberührt. Im Dialogfenster hat sich lediglich die Beschriftung links neben dem Steuerelement geändert.

Weitere Änderung und Warnung

Fügen wir nun im Rahmen einer weiteren Änderung den unten fett dargestellten Text zum USRPROPS.TXT-Code hinzu:

```
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" }
}
```

Die Erläuterung dieser Änderung erfolgt unten:

USRPROPS.TXT-Eintrag	Effekt
PROPERTY "Impact Statement" { EDIT TEXT LENGTH **100** }	Versucht, eine bereits vorhandene Eigenschaft zu ändern (hier: das Volumen von *Impact Statement* von 1000 Zeichen auf 100 zu reduzieren).

Wenn wir diese USRPROPS.TXT-Datei zurück in unsere Enzyklopädie importieren und die Enzyklopädie erneut öffnen, gibt Rational System Architect eine Warnung aus:

```
Warnung: Addendum in Eigenschaftendatei
des Benutzers zwischen Zeile 70 und
Zeile 72. Unzulässiger Versuch, die
Länge einer Eigenschaft zu kürzen.
Ursprüngliche Länge beibehalten.
```

Rational System Architect lässt keine Verkürzung eines Feldes zu, lediglich eine Verlängerung. Der Grund dafür ist, dass der Benutzer möglicherweise bereits Informationen in ein Textfeld eingegeben hat, die bei einer Verkürzung des Feldes verloren gingen. Dadurch würde sich die Menge der Informationen, die die Enzyklopädie für diese Eigenschaft zu einem späteren Zeitpunkt aufnehmen kann, reduzieren.

Rational System Architect gibt die Warnung aus, ignoriert den fehlerhaften Code und öffnet die Enzyklopädie. Wie bereits erwähnt, kann die Enzyklopädie im Falle einer Fehlernachricht erst nach Korrektur der USRPROPS.TXT-Datei geöffnet werden.

Wenn wir alternativ versuchen würden, die Länge des Feldes "Impact Statement" zu verlängern, würde Rational System Architect diese Änderung problemlos akzeptieren.

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

Definieren einer Werteliste

Sie können eine Liste mit Einträgen festlegen, die dem Benutzer in Form einer Dropdown-Liste oder Kontrollkästchenliste in einem Dialogfenster angeboten wird. Die Werte dieser Liste müssen in einer LIST-Definition angegeben werden. Auf die LIST-Definition wird anschließend in dem Diagramm, in dem Symbol oder in der Definition verwiesen, in dem/der sie verwendet wird. Die Listen müssen zuerst in der USRPROPS.TXT-Datei angelegt werden. Erst danach können Einträge für Diagramme, Symbole oder Definitionen vorgenommen werden, die auf diese Listen verweisen.

Die Verwaltung der USRPROPS.TXT-Datei ist einfacher, wenn alle LIST-Definitionen am Anfang der Datei stehen, im Anschluss an etwaige RENAME-Befehle.

Syntax der LIST-Definition

Eine LIST-Definition beginnt mit dem Schlüsselwort **LIST**, gefolgt von einer Zeichenfolge (dem Argument), die den Namen der Liste darstellt. Namen mit eingebetteten Leerzeichen sind in doppelte Anführungszeichen zu setzen. Die **LIST**-Definition ist zwischen geschweifte Klammern zu setzen **{ }** oder alternativ in eine **BEGIN...END**-Struktur einzubetten. Innerhalb der Klammern werden die Werte der Liste angegeben, die jeweils durch das Befehlsschlüsselwort **VALUE** aufgerufen werden. Wenn ein Wert über ein oder mehrere eingebettete Leerzeichen verfügt, ist der Wert in doppelte Anführungszeichen zu setzen.

```
LIST  list_name
{
VALUE value_name_1
VALUE value_name_2

…
}
```

Beispiel:

```
List "Method Stereotypes"
{
VALUE Get
VALUE Let
VALUE Set
```

```
VALUE "Stereotyp mit eingebetteten Leerzeichen"
}

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

Einrückungen und Zeilenumbrüche dienen ausschließlich der besseren Lesbarkeit und sind für den USRPROPS.TXT-Prozessor insoweit von Bedeutung, als sie zwischen den einzelnen Zeichenfolgen als Trennungsleerzeichen fungieren. Das obige Beispiel kann wie folgt geschrieben werden:

```
LIST "Method Stereotypes"  {   VALUE get   VALUE let
VALUE set  VALUE "Stereotyp mit eingebetteten
Leerzeichen"  }
```

Wenngleich dieses Format die Zwecke von Rational System Architect vollends erfüllt, ist es für die USRPROPS.TXT-Datei wahrscheinlich eher ungeeignet und sollte daher vermieden werden.

Kontrollkästchen und Dropdown-Listen

Rational System Architect zeigt eine Liste automatisch als Kontrollkästchenliste an, wenn die Anzahl der Werte in der LIST-Anweisung vier oder weniger beträgt. Beträgt die Anzahl der Werte fünf oder mehr, wird die Liste automatisch als Dropdown-Listenfeld angezeigt. Der Benutzer kann einen eigenen Wert in ein Dropdown-Listenfeld eingeben. Wenn Sie ein Dropdown-Listenfeld anlegen möchten, obwohl Sie nur über vier oder weniger als vier Listenwerte verfügen, verwenden Sie das Schlüsselwort LISTONLYCOMBO.

Eingeben von eigenen Werten

Wenn die Liste als Dropdown-Liste bereitgestellt wird, kann der Benutzer entweder einen der vier Werte aus der Liste wählen oder einen eigenen Wert eingeben (es sei denn, es wurde der Befehl LISTONLY oder LISTONLYCOMBO verwendet. Siehe hierzu Kapitel 3, LISTONLY- bzw. LISTONLYCOMBO-Befehl).

Umbenennen von bereits vorhandenen Diagramm-, Symbol- und Definitionstypen

Jede DIAGRAM-, SYMBOL- oder DEFINITION-Anweisung muss sich auf ein Objekt beziehen, das Rational System Architect bekannt ist.

Wenn sich die Namen in der bereitgestellten SAPROPS.CFG-Datei als ungeeignet erweisen, haben Sie die Möglichkeit, diese an Ihre individuellen Projekt- oder Unternehmensstandards anzupassen. Die RENAME-Anweisungen sollten am Anfang der USRPROPS.TXT-Datei, noch vor allen anderen Befehlen und Anweisungen, eingegeben werden. Die allgemeine Syntax des **RENAME**-Befehls sieht wie folgt aus:

```
RENAME class_name from_type_name TO to_type_name
```

Durch die folgenden drei Anweisungen werden ein Diagramm, ein Symbol und eine Definition umbenannt:

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

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

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

Abbildung 2- 5. Das Pulldown-Menü zur Auswahl des Typs zeigt nicht *Data Flow Gane & Sarson* an, sondern *Data Flow Chris & Trish*

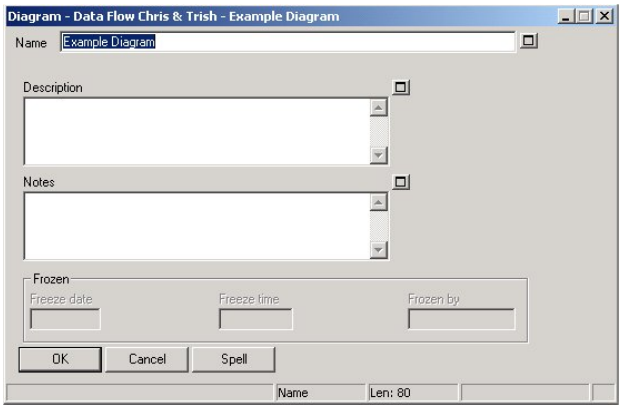


Abbildung 2- 6. Auch das Dialogfenster mit den Diagramm-eigenschaften zeigt als Titel *DFD Chris & Trish* an und nicht *DFD Gane & Sarson*

Der **RENAME SYMBOL**-Befehl kann von Entwicklern verwendet werden, die mit Ward & Mellor-DFDs arbeiten und den Namen *Process* (Prozess) gegenüber *Transform* (Umwandlung) vorziehen: Klicken Sie in einem beliebigen Ward & Mellor-DFD auf *Steuerungsumwandlung*. Doppelklicken Sie anschließend auf das Symbol im Diagramm, um das Dialogfenster **Diagramm <Typ> <Name>** anzuzeigen. Der Symboltyp lautet *Steuerungsumwandlung*.

Zum Umbenennen des Symbols muss folgender Befehl in die USRPROPS.TXT-Datei eingegeben werden:

RENAME class_name from_type_name IN
from_diagram_name TO to_type_name

Beispiel:

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

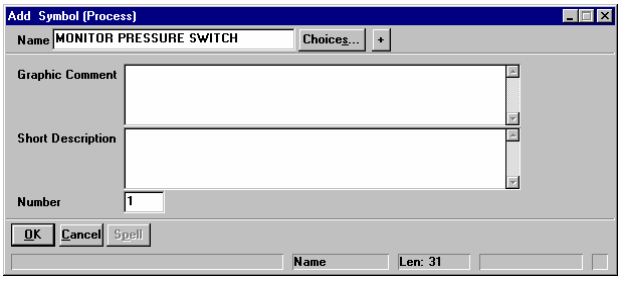


Abbildung 2-7. Dialogfenster mit Symboleigenschaften vor der Umbenennung

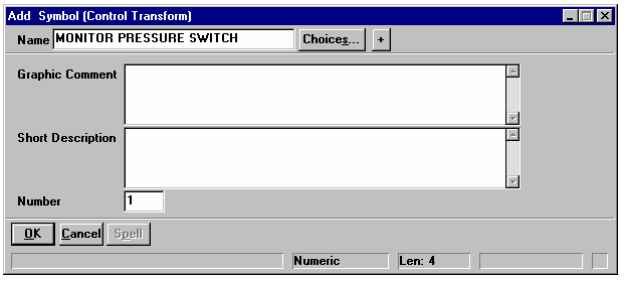


Abbildung 2-8. Dialogfenster mit Symboleigenschaften nach der Umbenennung

Klicken Sie im Menü **Bearbeiten** erneut auf **Bearbeiten <Symboltyp>**, während die Option *Steuerungsumwandlung* markiert ist. Beachten Sie den Titel des Dialogfensters **zum Ändern der Definition**: Die Definition lautet *Prozess und* nicht *Steuerungsumwandlung*. D. h. also, die Definition des Symbols *Steuerungsumwandlung* entspricht der Definition *Prozess*. Angenommen, Sie verwenden DFD Ward & Mellor und nicht DFD Gane & Sarson und möchten, dass der Definitionsname dem Symbolnamen entspricht.

Die Syntax des **RENAME**-Befehls für eine Definition sieht wie folgt aus:

RENAME class_name from_type_name TO to_type_name

RENAME DEFINITION "Process" TO "Control Transform"

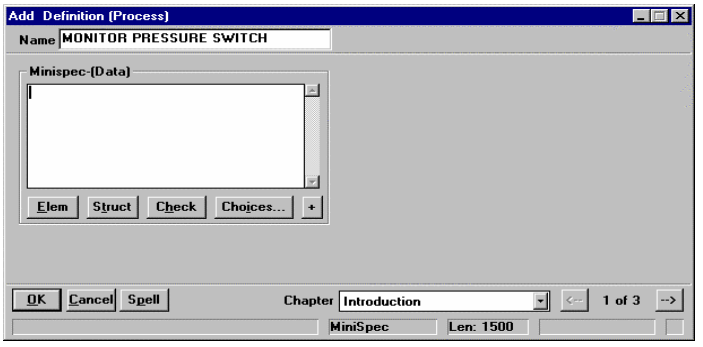


Abbildung 2-9. Dialogfenster für Symboldefinition vor der Umbenennung

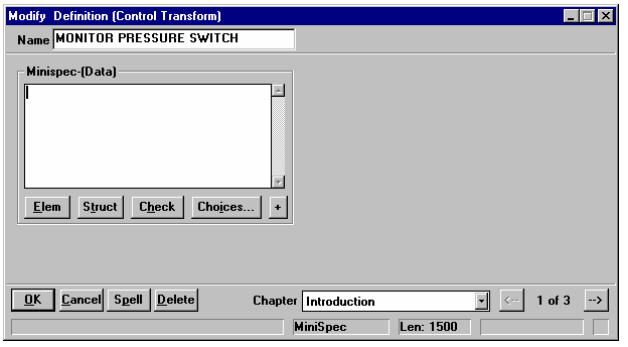


Abbildung 2-10. Dialogfenster für Symboldefinition nach der Umbenennung

Andererseits, wenn Sie sowohl DFD Gane & Sarson verwenden (wobei die Symbole für *Prozess* der Definition *Prozess* entsprechen) als auch DFD Ward & Mellor (wobei die Symbole für *Steuerungsumwandlung* der Definition *Prozess* entsprechen), möchten Sie möglicherweise nur die Definitionen für *Steuerungsumwandlungen* umbenennen, nicht aber die Definitionen aller Prozesse. Diese Umbenennung wird durch folgende USRPROPS.TXT-Einträge erreicht:

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

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

Wenn die Definition "Steuerungsumwandlung" keinen Eigenschaftensatz enthält, hat sie nur die Eigenschaft *Beschreibung*. In unserem Beispiel wurde der folgende Definitionsblock, der mit dem von *Prozess* identisch ist, zur USRPROPS.TXT-Datei hinzugefügt. Sie können natürlich beliebige für Ihre Zwecke geeignete Eigenschaften verwenden und müssen nicht die Eigenschaften einer bereits vorhandenen Definition kopieren.

```
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-Befehl und Berichte

Der **RENAME**-Befehl wirkt sich auch auf die Methode aus, mit der Sie Berichte erstellen. Überall dort, wo der frühere Name verwendet wurde, muss nun der neue Name verwendet werden. Im Berichtssystem der grafischen Benutzerschnittstelle müssen Sie den Eigenschaftennamen des Diagramms, des Symbols oder der Definition erneut auswählen, nachdem er in der SAPROPS.CFG-Datei geändert wurde.

---

[1] Es stehen Ihnen 150 "User n"-Definitionen zur Verfügung, beginnend mit User 1.

Vorher:

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

Wenn Sie den Bericht im Texteditor öffnen (Dialogfenster **Berichte**, Befehl **EDIT**) wird Folgendes angezeigt.

Nachher:

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

Erstellen neuer Diagramm-, Symbol- und Definitionstypen

In einer Rational System Architect-Enzyklopädie können Sie neue Diagramm-, Symbol- und Definitionstypen erstellen. Verwenden Sie den speziell dafür vorgesehenen RENAME-Befehl, um einen bereits bestehenden Diagramm-, Symbol- oder Definitionstyp umzubenennen. Auch in diesem Fall sollte der RENAME-Befehl an den Anfang der USRPROPS.TXT-Datei platziert werden, direkt unterhalb der ersten REM-Anweisungen (Erinnerung oder Kommentar).

Erstellen neuer Diagramme

In einer Rational System Architect-Enzyklopädie können Sie bis zu 50 neue Diagrammtypen erstellen. Zum Hinzufügen eines neuen Diagrammtyps müssen Sie in der USRPROPS.TXT-Datei einen der 50 generischen Diagrammtypen umbenennen, die Ihnen zur Verfügung stehen: User 1 bis User 50. Die Syntax lautet wie folgt:

```
RENAME DIAGRAM "User 1" TO My_Diagram
```

Beachten Sie, dass Sie den Namen in doppelte Anführungszeichen setzen müssen, wenn in dem von Ihnen erstellten neuen Diagramm-, Symbol- oder Definitionstyp eingebettete Leerzeichen enthalten sein sollen. Beispiel:

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

Nachdem Sie den neuen Diagrammtyp erstellt haben, können Sie die Symboltypen angeben, die in dem Diagramm verfügbar sein sollen. Sie können entweder neue Symboltypen erstellen, oder dem neuen Diagrammtyp Symbole zuweisen, die bereits in einem anderen Diagramm verfügbar sind. Darauf gehen wir im folgenden Abschnitt (Zuweisen eines Symboltyps zu einem Diagrammtyp) näher ein.

Standardmäßig handelt es sich bei Benutzerdiagrammen um Netzwerke (von Symbolen). Sie können jedoch auch ein Benutzerdiagramm des Typs **Hierarchisch** festlegen. Für hierarchische Diagramme gelten in Rational System Architect bestimmte Zeichnungsregeln, mit denen Sie die Symbole in

einer Hierarchie verbinden und automatisch Liniensymbole zeichnen können. Weitere zugehörige hierarchische Funktionen (wie z. B. die hierarchische Nummerierung) werden ebenfalls unterstützt. Verwenden Sie das Schlüsselwort HIERARCHICAL, um ein Diagramm als hierarchisches Diagramm festzulegen. Beispiel:

DIAGRAM "Zoo" {**HIERARCHICAL**}

Erstellen neuer Symbole

Sie können bis zu 150 neue Symboltypen in einer Rational System Architect-Enzyklopädie erstellen. Zum Hinzufügen eines neuen Symboltyps müssen Sie einen der 150 generischen Symboltypen umbenennen, die Ihnen zur Verfügung stehen: User 1 bis User 150. Die Syntax sieht wie folgt aus:

RENAME SYMBOL "User 3" to "Neuer Name"

Festlegen neuer Liniensymbole

Ein Liniensymbol ist eine Linie, die zwischen zwei Symbolen gezeichnet werden kann, z. B. eine Beziehungslinie, eine Vererbungslinie, eine Assoziationslinie, eine Flusslinie usw. In einer Enzyklopädie können Sie einen neuen Liniensymboltyp erstellen. Sie müssen angeben, dass dieser wie ein bereits vorhandener Liniensymboltyp in einem anderen Diagramm aussehen und sich entsprechend verhalten soll. Verwenden Sie dabei denselben RENAME SYMBOL-Befehl wie für ein reguläres ('Knoten-') Symbol. Später müssen Sie jedoch in der USRPROPS.TXT-Datei über den DEPICT LIKE-Befehl angeben, wie das Liniensymbol gezeichnet werden soll.

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

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

Erstellen neuer Definitionen

In einer Rational System Architect-Enzyklopädie können Sie bis zu 150 neue Definitionstypen erstellen. Zum Hinzufügen eines neuen Definitionstyps müssen Sie einen der 150 generischen Definitionstypen umbenennen, die Ihnen zur Verfügung stehen: User 1 bis User 150. Die Syntax sieht wie folgt aus:

```
RENAME DEFINITION "User 3" to "whatever"
```

Ein Symbol stellt in der Regel einen Definitionstyp dar. Informationen dazu finden Sie im nachfolgenden Abschnitt (Zuweisen eines Definitionstyps zu einem Symboltyp).

Zuweisen eines Symboltyps zu einem Diagrammtyp

Sie können neuen oder bereits bestehenden Diagrammtypen einen neuen oder einen bereits vorhandenen Symboltyp (Symbole, die bereits in einem anderen Diagramm vorhanden sind) zuweisen. Mit folgender Syntax können Diagrammtypen Symboltypen zugewiesen werden:

ASSIGN <Symboltyp-Name> [IN <Diagrammtyp-Name1>] **TO** <Diagrammtyp-Name2>

Symboltypen können Diagrammtypen auch innerhalb der Symbolspezifikation zugwiesen werden, indem die Schlüsselwortkombination **ASSIGN .. TO** wie folgt verwendet wird:

SYMBOL <Symboltyp-Name> [IN <Diagrammtyp-Name1>] **{ASSIGN TO** <Diagrammtyp-Name1>**}**

Beispiel

Durch die USRPROPS.TXT-Datei unten wird ein neuer Diagrammtyp (Wireless Network-Diagramm) erstellt. Dieser Diagrammtyp verfügt über drei neue Symboltypen, die in dem Diagramm verwendet werden können (einen Satelliten, einen Computer und einen Server) sowie über einen bereits bestehenden Symboltyp, und zwar ein Statussymbol aus einem "Ward & Mellor-Statusübergangsdiagramm" (also kein Statussymbol aus einem UML- oder IDEF3-Statusdiagramm, usw):

```
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"}
```

Anmerkung: Lesen Sie auch den Abschnitt *Einschränkungen für das Zuweisen von Symboltypen zu Diagrammtypen*.

Zuweisen eines Liniensymboltyps zu einem Diagrammtyp

Wie bereits in diesem Abschnitt erwähnt, handelt es sich bei einem Liniensymbol um eine Linie, die zwischen zwei Symbolen gezeichnet werden kann, z. B. eine Beziehungslinie, eine Vererbungslinie, eine Assoziationslinie, eine Flusslinie usw. In einer Enzyklopädie können Sie einen neuen Liniensymboltyp erstellen. Sie müssen angeben, dass dieser wie ein bereits vorhandener Liniensymboltyp in einem anderen Diagramm aussehen und sich entsprechend verhalten soll. Verwenden Sie dazu denselben RENAME SYMBOL-Befehl wie für ein reguläres ('Knoten-') Symbol. Später müssen Sie jedoch in der USRPROPS.TXT-Datei über den DEPICT LIKE-Befehl angeben, wie das Liniensymbol gezeichnet werden soll.

Benutzerdefinierte Symbole (User 1 bis User 150) werden sowohl für reguläre ('Knoten-') Symbole als auch für Liniensymbole bereitgestellt. Stellen Sie daher sicher, dass Sie nicht dieselbe Symbolnummer für zwei verschiedene Symbole verwenden.

Beispiel

Im Beispiel unten fügen wir das fett gedruckte neue Liniensymbol zur USRPROPS.TXT-Datei hinzu:

```
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" }
```

Anmerkung: Lesen Sie auch den Abschnitt *Einschränkungen für das Zuweisen von Symboltypen zu Diagrammtypen*.

Einschränkungen für das Zuweisen von Symboltypen zu Diagrammtypen

Für das Zuweisen von Symboltypen zu Diagrammtypen gelten folgende Einschränkungen:

Bei folgenden Diagrammtypen ist keine Zuweisung möglich:

- DB2 - physisch
- Entitätenbeziehung
- Logisches Datenmodell
- Logische Ansicht
- Physisches Datenmodell

Folgende Symbole können nicht zugeordnet werden, da sie besonderen Code in Rational System Architect enthalten:

- Symbol "Assoziative Entität" im Diagramm "Entitätenbeziehung"
- Symbol "Entität" im Diagramm "Entitätenbeziehung"
- Symbol "Identifizierende Beziehung" im Diagramm "Entitätenbeziehung"
- Symbol "Inkonsistente Beziehung" im Diagramm "Entitätenbeziehung"
- Symbol "Nicht identifizierende Beziehung" im Diagramm "Entitätenbeziehung"
- Symbol "Nicht spezifische Beziehung" im Diagramm "Entitätenbeziehung"
- Symbol "Super-Sub-Beziehung" im Diagramm "Entitätenbeziehung"
- Symbol "Schwache Entität" im Diagramm "Entitätenbeziehung"
- Symbol "Assoziation" im Diagramm "OMT-Objektmodell"
- Symbol "Klasse" im Diagramm "OMT-Objektmodell"
- Symbol "Identifizierende Bedingung" im Diagramm "Physisches Datenmodell"
- Symbol "Nicht identifizierende Bedingung" im Diagramm "Physisches Datenmodell"

- Symbol "Tabelle" im Diagramm "Physisches Datenmodell"
- Symbol "Klasse" im Diagramm "UML-Klasse"
- Symbol "Schnittstelle" im Diagramm "UML-Klasse"
- Symbol "Akteur" im Diagramm "UML-Anwendungsfall"
- Symbol "Grenze" im Diagramm "UML-Anwendungsfall"
- Symbol "Fallmitarbeiter" im Diagramm "UML-Anwendungsfall"
- Symbol "Steuerung" im Diagramm "UML-Anwendungsfall"
- Symbol "Entität" im Diagramm "UML-Anwendungsfall"
- Symbol "Mitarbeiter" im Diagramm "UML-Anwendungsfall"
- Zusätzlich können folgende Symbole keinem anderen Diagramm zugewiesen werden, weil ihre Definition mit dem Schlüssel "Modell" versehen ist:
- Symbol "Zugriffspfad" im Diagramm "Entitätenbeziehung"
- Symbol "Beziehung" im Diagramm "Entitäten-beziehung"
- Symbol "Beziehungsraute" im Diagramm "Entitätenbeziehung"
- "Individu" in diagram "Modèle Conceptuel des Données"
- "Relation Ligne" in diagram "Modèle Conceptuel des Données"
- Symbol "Verschlüsselter Eingangspunkt" im Diagramm "SSADM-Datenstruktur"
- Symbol "Nicht verschlüsselter Eingangspunkt" im Diagramm "SSADM-Datenstruktur"
- Symbol "Beziehung" im Diagramm "SSADM-Datenstruktur"
- Zusätzlich können folgende BPMN-Symbole keinem anderen Diagrammtyp zugeordnet werden:
- Symbol "Pool" im Diagramm "Geschäftsprozess"
- Symbol "Lane" im Diagramm "Geschäftsprozess"

Anmerkung: Bestimmte Symbole verfügen über einen besonderen Code und werden zudem durch den Schlüssel "Modell" angegeben. Diese Symbole werden nur in der ersten Symbolliste angezeigt.

Viele Symbole können normalerweise in mehreren Diagrammtypen angesiedelt sein. Für die Symbole in den obigen Listen wird immer nur ein Diagrammtyp angezeigt.

Zuweisen eines Definitionstyps zu einem Symboltyp

Wenn Sie neue Symbole zu einer Enzyklopädie in der USRPROPS.TXT-Datei hinzufügen möchten, müssen Sie mithilfe dieses Schlüsselworts angeben, mit welchem Definitionstyp diese verbunden sind. Wenn ein neues in der USRPROPS.TXT-Datei angegebenes Symbol nicht mit dieser Klausel versehen ist, gibt Rational System Architect beim Öffnen der Enzyklopädie eine Syntaxanalysewarnung aus und verwendet für das Symbol die Standarddefinition Null, welche lediglich aus der Beschreibungseigenschaft besteht.

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

Beispiel

Im Beispiel unten ist für den Symboltyp "Satellite" der Definitionstyp "Satellite" angegeben (die Tatsache allein, dass es sich zufällig um denselben Namen handelt, reicht nicht aus).

```
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" }
```

Abbilden eines Symbols mit einer Bitmap- oder Metadatei

Sie können ein Symbol mit einer von Ihnen bereitgestellten Bitmap- (.bmp) oder Windows Metadatei (.wmf) abbilden. Sie können angeben, wie ein Symbol im Diagrammarbeitsbereich sowie im Toolbox- und Zeichenmenü abgebildet werden soll, indem Sie der Deklaration des Symbols eine Abbildungsklausel hinzufügen. Gehen Sie dabei folgendermaßen vor:

SYMBOL <Symboltyp-Name>

{ ...
DEPICTIONS { DIAGRAM
<Abbildungsdatei> }
DEPICTIONS { MENU <Abbildungsdatei> }
...}

Der DIAGRAM-Befehl gibt an, welche Abbildungsdatei im Diagrammarbeitsbereich zum ***Zeichnen*** verwendet wird. Für den DIAGRAM-Befehl verwenden Sie am besten eine ***Windows-Metadatei (.WMF)***. Da es sich um ein Vektorbild handelt, können Sie es genau skalieren, indem Sie die Ziehpunkte zur Größenanpassung verwenden. Sie können für den DIAGRAM-Befehl auch BMP-Dateien verwenden, diese lassen sich jedoch nicht so gut skalieren.

WMF-Dateien sind Vektordateien, d. h. sie enthalten mathematische Formeln, die festlegen, wie ein Bild am Bildschirm darzustellen ist. Ein großer Vorteil dieses Formats ist die Skalierbarkeit, die keinerlei Einfluss auf die Bildqualität hat. WMF-Dateien werden beim Vergrößern oder Verkleinern nicht verschwommen bzw. kantig angezeigt.

Der **MENU**-Befehl gibt die Abbildungsdatei an, die für die Darstellung in den ***Symbolleisten***, ***Menüs*** und in anderen Bereichen verwendet werden soll. Es ist die Grafik, auf die Sie klicken, um ein zu zeichnendes Symbol auszuwählen. Für die Symbolleisten eignen sich **Bitmapbilder** am besten,

da keine Skalierung erforderlich ist. Eine bewährte Vorgehensweise ist es, für jedes Symbol, das Sie in der Symbolleiste darstellen möchten, ein Bitmapbild mit einer Auflösung von ***16x16 Pixel*** zu erstellen.

BMP-Dateien sind Rasterdateien, d. h. sie enthalten Informationen über jedes einzelne Pixel des Bildes. Zwar können Sie Bitmapdateien zur Wiedergabe von Bildern verwenden, allerdings werden diese beim Vergrößern verschwommen bzw. beim Verkleinern kantig angezeigt.

Die **<Abbildungsdatei>** stellt den Namen und den vollständigen Pfad der Bitmap- oder Metadatei dar. Sie können zwar ein Verzeichnis außerhalb Ihres Enzyklopädiepfads angeben, es empfiehlt sich jedoch, die Bitmap- und Metadateien direkt zur Dateientabelle der Enzyklopädiedatenbank hinzuzufügen.

Gehen Sie folgendermaßen vor, um eigene Abbildungsdateien zu einer Enzyklopädie hinzuzufügen:

1. **Nehmen Sie die erforderlichen Änderungen in der USRPROPS.TXT-Datei vor.** Beispiel für einen entsprechenden Code:

   ```
   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. **Importieren Sie Ihre BMP- und WMF-Dateien in die Dateientabelle der Enzyklopädie**. Zum Importieren Ihrer benutzerdefinierten Grafikdateien in die Dateientabelle der Enzyklopädiedatenbank können Sie entweder den Enzyklopädiedateimanager von Rational System Architect verwenden (über "Tools, Enzyklopädiedateimanager") oder SAEM (Start, Programme, IBM Rational, IBM Rational Lifecycle Solutions Tools, IBM Rational System Architect 11.3.1, SAEM, Hilfedatei) oder Enterprise Manager von Microsoft. Bei Verwendung des Enzyklopädiedateimanagers kann

immer nur eine Datei gleichzeitig importiert werden. Wenn Sie mehrere Grafikdateien in die Dateientabelle importieren möchten, empfehlen wir die Verwendung von SAEM.

Die Namen der von Ihnen importierten Dateien sollten mit Ihrem USRPROPS.TXT-Code übereinstimmen. Im obigen Beispiel haben wir einen relativen Pfad verwendet, indem wir überhaupt keinen Pfad angegeben haben. Wir haben lediglich die Dateien SATELLITE.WMF und SATELLITE.BMP aufgeführt. Das bedeutet, dass die Abbildungsdateien direkt in die Dateientabelle der Datenbank importiert werden sollen.

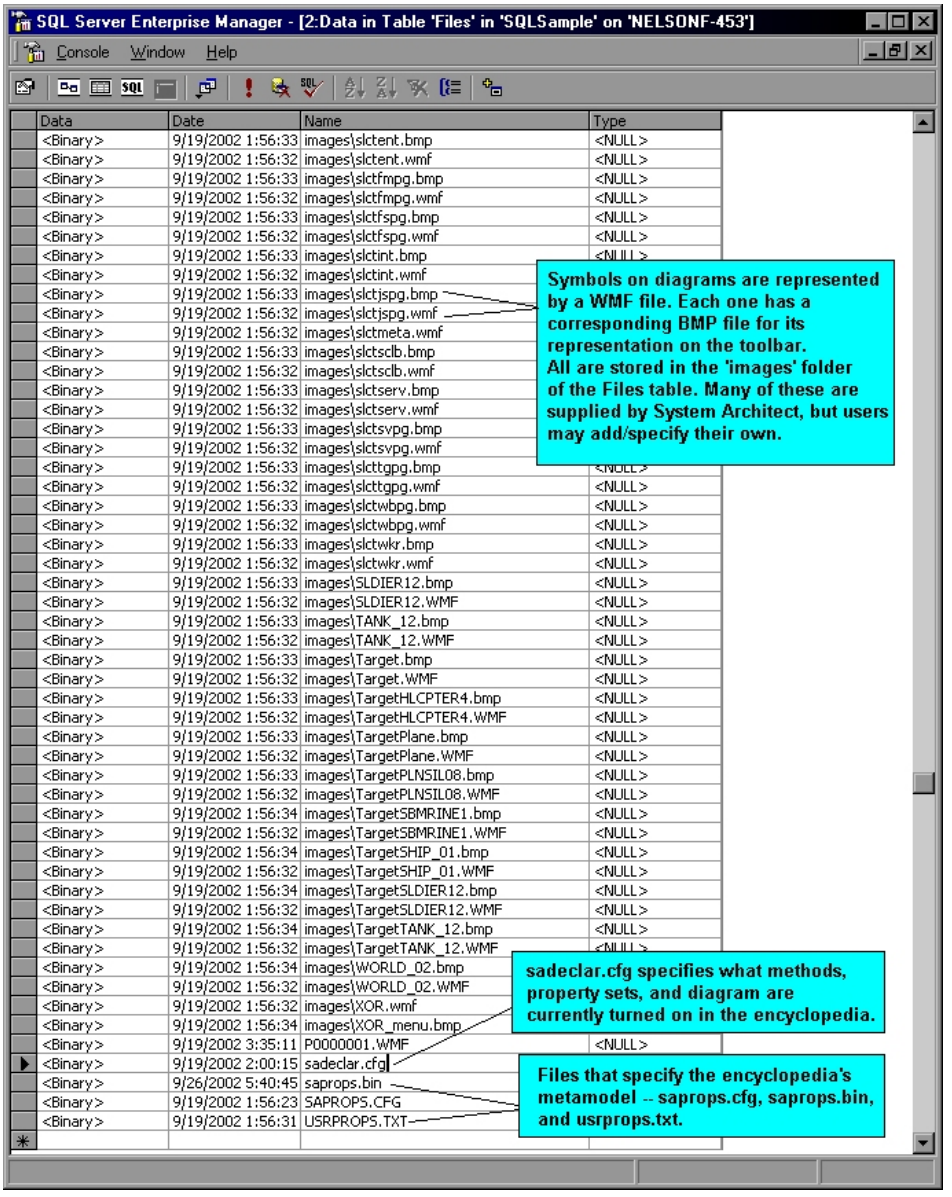
Empfehlung: Wir empfehlen Ihnen die Anwendung einer etablierten Konvention von Rational System Architect. Hängen Sie an den Namen Ihrer Abbildungsdatei den Zusatz 'images/' an, um zu simulieren, dass sich jede Abbildungsdatei in einem Unterverzeichnis der Dateientabelle mit der Bezeichnuung 'images' befindet. Wenn Sie zum Importieren mehrerer Dateien SAEM verwenden, stellen Sie sicher, dass die Dateien in einem Verzeichnis mit der Bezeichnung 'images' abgelegt sind. Dieses Verzeichnis kann sich an einem beliebigen Speicherort Ihres Computers befinden. SAEM hängt automatisch den Zusatz 'images/' an den Anfang des Grafikdateinamens an, wenn die Datei aus einem Verzeichnis mit der Bezeichnung 'images' importiert wird. Analog sollten Sie den Zusatz 'images\' in der USRPROPS.TXT-Datei vor dem Namen der jeweiligen Abbildungsdatei einfügen. Bezogen auf das obige Beispiel sähe dies folgendermaßen aus:

```
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 }
}
```

Die Anwendung dieser Strategie hat zwei Vorteile. Erstens bietet sie eine gewisse Unabhängigkeit bei der Namensvergabe und eine Strategie für die logische Gruppierung von benutzerdefinierten Bildern. Zweitens ist sie konsistent mit der Verarbeitungsweise von Bildern beim Erstellen neuer Enzyklopädien. Rational System Architect greift auf alle Grafiken im Verzeichnis ...\System Architect\images zu, legt sie in der Dateientabelle der neuen Enzyklopädie ab und weist ihnen einen Namen mit dem Zusatz 'images\' zu.

Die nachfolgende Abbildung zeigt eine Dateientabelle einer Enzyklopädiedatenbank. Anhand dieser Tabelle können Sie erkennen, inwieweit das Präfix 'images\' vor den einzelnen Abbildungsdateien eine logische Gruppierung der Bilder ermöglicht.

Abbildung 2-11. Dateientabelle einer Enzyklopädie-datenbank



3. **Öffnen Sie die Enzyklopädie erneut, um die Änderungen zu übernehmen**.

Angeben von Abbildungsdateien für neue Enzyklopädien

Beim Erstellen einer neuen Enzyklopädie haben Sie zwei Möglichkeiten: Entweder Sie erstellen zuerst die neue Enzyklopädie und importieren anschließend eine oder mehrere benutzerdefinierte Grafikdateien mithilfe von SAEM, dem Enzyklopädiedateimanager oder Enterprise Manager von SQL Server (wie oben beschrieben) oder Sie legen zuerst Ihre benutzerdefinierten Bilder in das Hauptverzeichnis von Rational System Architect mit der Bezeichnung ***images*** ab (im Hauptsoftwareverzeichnis unter <C>:\Programme\IBM\Rational\11.3.1\System Architect Suite\System Architect\images) und erstellen anschließend die Enzyklopädie. Rational System Architect greift auf die Grafiken im Hauptverzeichnis 'images' zu und legt sie in der Dateientabelle der neu erstellten Enzyklopädie ab.

Wenn Sie möchten, dass dieselben benutzerdefinierten Grafikdateien in allen von Ihnen oder anderen Teammitgliedern neu erstellten Enzyklopädien abgelegt werden, gehen Sie folgendermaßen vor:

1. **Kopieren Sie die BMP- und WMF-Dateien und fügen Sie sie in das Unterverzeichnis 'images' von Rational System Architect ein**. Legen Sie die BMP- und WMF-Dateien vor dem Erstellen einer neuen Enzyklopädie in das Verzeichnis 'images' innerhalb des Hauptprogrammverzeichnisses von Rational System Architect ab. Dies sollten auch alle Teammitglieder tun, die später einmal neue Enzyklopädien erstellen werden. Diese Dateien werden nun automatisch in die Dateientabelle der später erstellten Enzyklopädie abgelegt. Rational System Architect hängt an jeden Dateinamen den Zusatz 'images\' an. Eine Bilddatei mit dem Namen FRED.BMP wird demnach in der Dateientabelle der neuen Enzyklopädie unter dem Namen

IMAGES\FRED.BMP angelegt. Dieses Verfahren ist schneller, als zuerst die Enzyklopädie zu erstellen und anschließend die benutzerdefinierten Grafikdateien in die Enzyklopädie zu importieren.

2. **Nehmen Sie die erforderlichen Änderungen in der USRPROPS.TXT-Datei vor.** Verwenden Sie dazu den DEPICTIONS-Befehl (und optional den RETAIN STYLE-Befehl). Informationen zum Vornehmen der erforderlichen Codeänderungen finden Sie in der Hilfedatei von Rational System Architect. Beispiel für einen entsprechenden Code:

```
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. **Öffnen Sie die Enzyklopädie erneut, um die Änderungen zu übernehmen.**

Benutzerdefinierte Symboldarstellung auf Basis des Eigenschaftswerts

Sie können *basierend auf dem Wert einer Eigenschaft* der Symboldefinition angeben, wie ein Symbol gezeichnet werden soll. In UML handelt es sich bei dieser Eigenschaft im Allgemeinen um ein Stereotyp. Diese Funktionalität gilt jedoch durchgängig für alle Symboltypen – nicht nur für UML-Symbole – und nicht nur für die Stereotypeigenschaft.

Zur Aktivierung dieser Funktion wird die **DEPICTIONS**-Klausel direkt innerhalb einer **LIST**-Anweisung in der USRPROPS.TXT-Datei verwendet.

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

Beispiel

Im folgenden Beispiel wird eine neue Liste für Knotenstereotypen angegeben. Diese Stereotypen werden für ein Knotensymbol auf einem UML-Implementierungsdiagramm angewendet. Auf diese Weise kann der Benutzer mithilfe eigener Grafikdateien, die er zuvor in die Dateientabelle der Enzyklopädiedatenbank importiert hat, ein Knotensymbol zeichnen.

```
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}
```

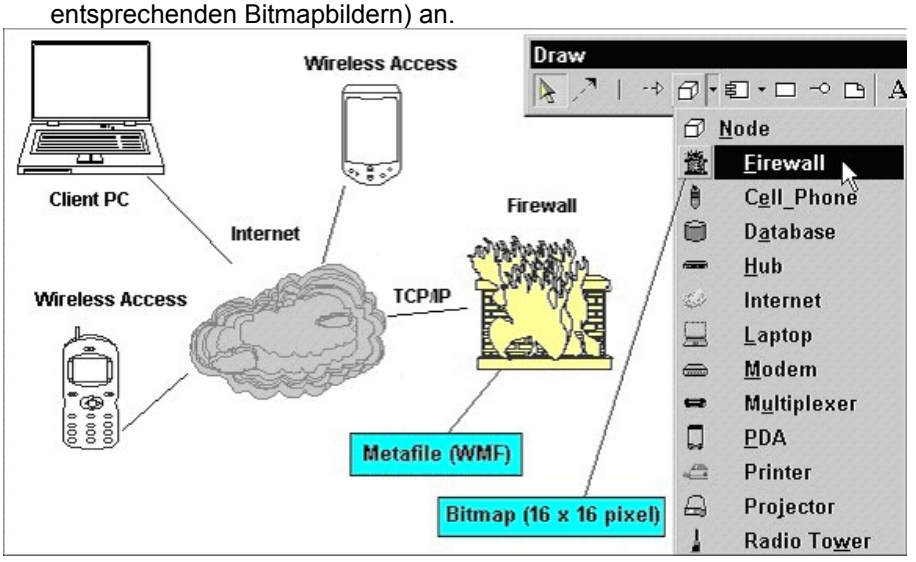
```
Value "Multiplexer"  DEPICTIONS {DIAGRAM
images\multiplexer.wmf MENU images\multiplexer.bmp}
Value "PDA"  DEPICTIONS {DIAGRAM images\pda.wmf MENU
images\pda.bmp}
Value "Printer"   DEPICTIONS {DIAGRAM images\printer.wmf
MENU images\printer.bmp}
Value "Projector" DEPICTIONS {DIAGRAM
images\projector.wmf 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  }
}
```

Beachten Sie, dass in der obigen USRPROPS.TXT-Beispieldatei sowohl im Symbol als auch in der Definition eines Knotens auf die Liste "Node Stereotypes" verwiesen wird. Im Symbol wurde die Eigenschaft UNSICHTBAR hinterlegt. Das Symbol verweist auf das

Stereotyp, das für die dem Symbol zugrunde liegende Definition angegeben wurde.

Der obige USRPROPS.TXT-Code verändert die Symbolleiste eines Implementierungsdiagramms und zeigt eine Dropdown-Liste der verfügbaren Stereotypen (mit den entsprechenden Bitmapbildern) an.

Abbildung 2-12. Vom Benutzer bereitgestellte Abbildungsdateien



Der Benutzer kann ein Stereotyp auswählen und in das Diagramm übertragen, wo es durch die entsprechende WMF-Datei dargestellt wird.

Nach dem Zeichnen des Symbols stehen Ihnen durch Klicken mit der rechten Maustaste auf das jeweilige Symbol folgende Optionen zur Verfügung:

- Anzeigen als <Knoten>
- Mit Stereotypen ausstatten (Piktogramm der Metadatei wird rechts neben dem Symbolnamen platziert)
- Gemäß Stereotyp anzeigen

Nach dem Zeichnen des Symbols können Sie die Farben der Metadatei angeben, genau wie bei jedem anderen Symbol in

Rational System Architect, indem Sie die Symbolleiste mit den Symbolstilarten verwenden (oder das Symbol auswählen und dann Format, Symbolformat, Symbolstil wählen). Sie können u. a. die Linienfarbe, Füllfarbe und Schriftfarbe angeben.

Stil beibehalten

Sie können angeben, dass eine von Ihnen bereitgestellte Metadatei bei der Verwendung in Rational System Architect ihren ursprünglichen Grafikstil und ihre ursprüngliche Farbgebung beibehält. Dazu müssen Sie das Schlüsselwort RETAIN STYLE verwenden. Beispiel:

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

Wenn die vom Benutzer bereitgestellte FIREWALL.WMF-Metadatei in das Diagramm übertragen wird, weist sie genau die gleiche Farbgebung auf wie außerhalb von Rational System Architect und kann auch nicht durch die Farbtools von Rational System Architect geändert werden.

Darstellbare Eigenschaften abgebildeter Symbole

Rational System Architect ermöglicht Ihnen mithilfe des Schlüsselworts "DISPLAY" die Angabe von bis zu 37 Eigenschaften in einem Symbol. Dies gilt auch für Symbole, die anhand von benutzerseitig bereitgestellten Abbildungsdateien abgebildet werden.

Weitere Informationen finden Sie im Abschnitt *Festlegen der Anzeige von Werten auf Symbolen* (im weiteren Verlauf dieses Kapitels) oder in Kapitel 3 im Abschnitt über das Schlüsselwort DISPLAY.

Angeben von Eigenschaften für Diagramme, Symbole und Definitionen

In jeder Enzyklopädie von **Rational System Architect** gibt es drei Klassen: *Diagramm*, *Symbol* und *Definition*. Jede Klasse kann mit einem eigenen Eigenschaftensatz definiert werden.

Die folgende Tabelle enthält alle obligatorischen und optionalen Einträge außerhalb einer Diagramm-, Symbol- oder Definitionsanweisung der USRPROPS.TXT-Datei.

Tabelle 2-3. Obligatorische und optionale Einträge für eine Diagramm-, Symbol- oder Definitionsspezifikation

Eintrag	Obligatorisch Optional	Anmerkung
DIAGRAM { } oder DIAGRAM BEGIN END oder SYMBOL { } oder SYMBOL BEGIN END oder DEFINTION { } oder DEFINITION BEGIN END	Obligatorisch	Eröffnet und beendet die Deklaration.
CHAPTER chapter_name	Optional	Schließt nachfolgende Eigenschaften in ein bereits vorhandenes Kapitel ein oder fügt ein neues Kapitel hinzu.

Eintrag	Obligatorisch Optional	Anmerkung

GROUP group_name
{
PROPERTY prop_name
PROPERTY prop_name
}
	Optional	Platziert alle nachfolgenden Eigenschaften innerhalb einer Gruppe für die Layoutsteuerung.

LAYOUT { alignment_criteria PACK_TAB_criteria COLS no_of_columns JUSTIFY }	Optional	[Align Body | Align Label | Align Over] [Pack | Tab] COLS <number>

PROPERTY property_name
{
}
	Obligatorisch	Sie können '{' bzw. BEGIN und '}' bzw. END verwenden.

Angeben von Eigenschaften für Diagrammtypen

Die Standardeigenschaft aller Diagramme lautet *Beschreibung*. Die Beschreibung ist als Textfeld mit 4074 Zeichen definiert. Diagrammeigenschaften sind Eigenschaften, die der Benutzer für das gesamte Diagramm einstellen kann, z. B. ob Swimlanes (oder Pools) vertikal oder horizontal angezeigt werden. Unten ist ein typisches Dialogfenster mit **Diagrammeigenschaften** dargestellt.

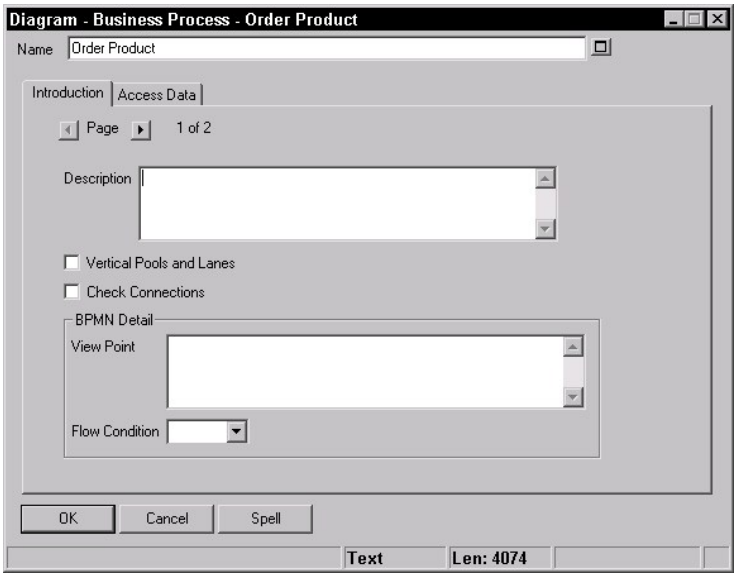


Abbildung 2- 13. Dialogfenster mit Diagrammeigenschaften

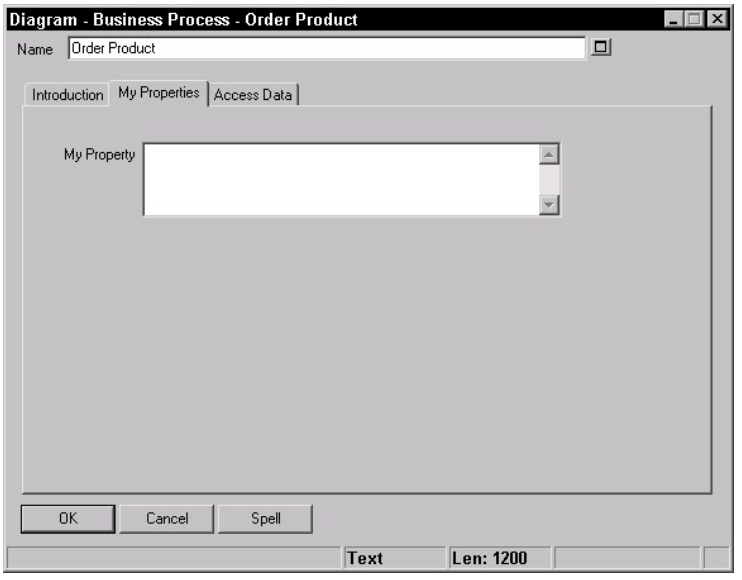
Mit folgender Syntax können Sie einem Diagramm weitere Eigenschaften hinzufügen:

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

So ändert sich beispielsweise durch Einfügen der nachfolgenden Anweisungen in die USRPROPS.TXT-Datei das Dialogfenster mit den **Diagrammeigenschaften** für den Diagrammtyp "Geschäftsprozess" wie in der Abbildung unten dargestellt:

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

Abbildung 2-14. Neues Dialogfenster mit Diagramm-eigenschaften



Angeben von Eigenschaften für Symboltypen

Symboleigenschaften sind in der Registerkarte "Symbol" im Definitionsdialogfenster des Symbols enthalten.

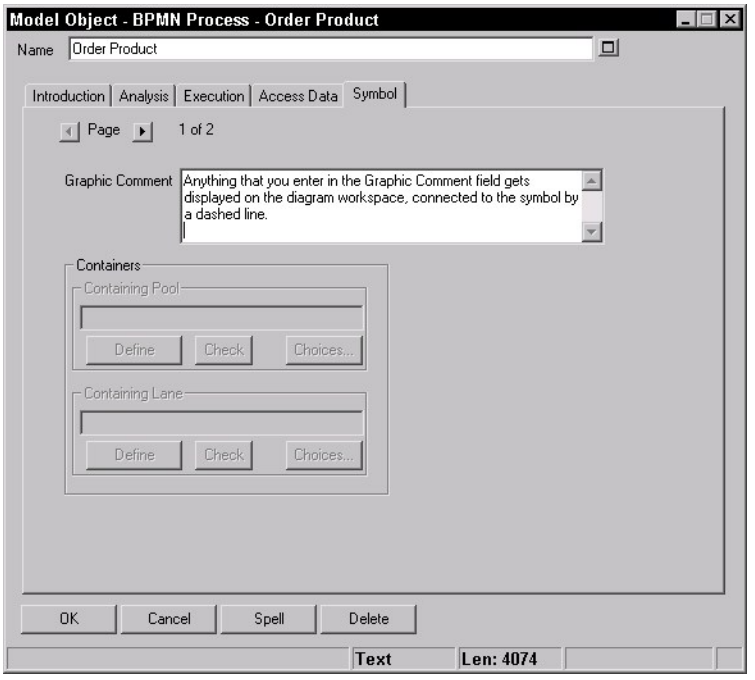


Abbildung 2-15. Dialogfenster mit Symboleigenschaften mit der Standardeigenschaft *Grafikkommentar*

Grafikkommentar

Die Standardeigenschaft aller Symbole lautet *Grafikkommentar*. Der *Grafikkommentar* ist als Textfeld mit 4074 Zeichen definiert. Text, den Sie in das Grafikkommentarfeld eingeben, wird als Kommentar im Diagrammarbeitsbereich angezeigt und durch eine Linie mit dem Symbol verbunden. Die Linie wird nur dann gezeichnet, wenn der Grafikkommentar einen bestimmten Abstand zum Symbol überschreitet. Sie können diesen Abstand anpassen, indem Sie das Symbol auswählen und dann "Format, Diagrammformat, Notation" wählen und die Optionen unter "Linie zu fernem Text" anpassen.

Sie können auch angeben, dass der Grafikkommentar innerhalb des Symbols angezeigt wird (wählen Sie das Symbol aus, und wählen Sie dann "Format, Symbolformat, Textposition" und inaktivieren Sie die Option "Grafikkommentar außerhalb platzieren"). Sie können die Anzeige des Grafikkommentars auch vollständig aktivieren/inaktivieren. Klicken Sie dazu mit der rechten Maustaste auf das Symbol, wählen Sie "Anzeigemodus", und inaktivieren Sie die Option "Grafikkommentar".

Hinzufügen weiterer Eigenschaften zu einem Symbol

Mit folgender Syntax können Sie einem Symbol weitere Eigenschaften hinzufügen:

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

Sie müssen den Diagrammtyp angeben, in dem das Symbol, auf das Sie verweisen, enthalten ist. Ein Symbol kann in vielen verschiedenen Diagrammtypen angezeigt werden.

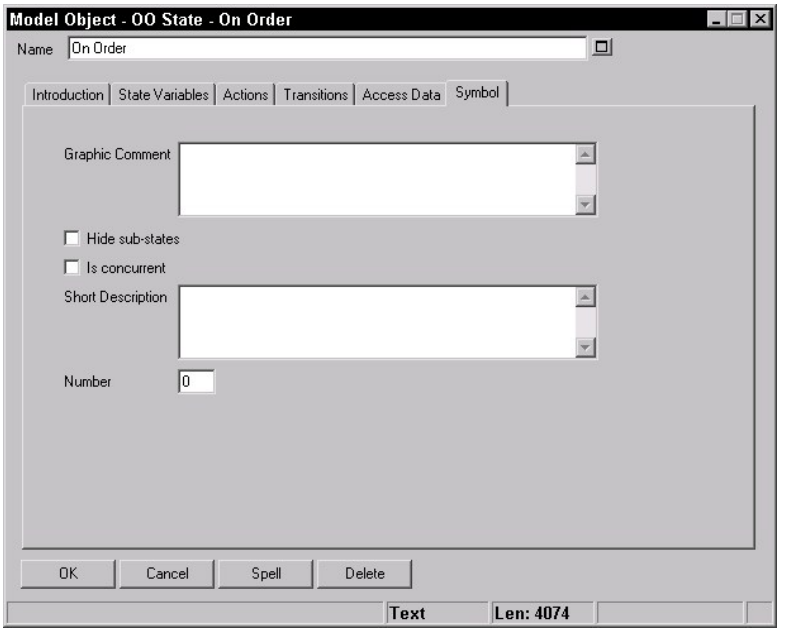
Beispiel

Wir nehmen folgende Änderungen in der USRPROPS.TXT-Datei vor:

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

Durch diese Änderungen werden die Eigenschaften *Kurzbeschreibung* und *Nummer* zum Statussymbol eines UML-Statusdiagramms hinzugefügt. Der *Grafikkommentar* ist immer verfügbar. Das geänderte Dialogfenster ist unten dargestellt:

Abbildung 2-16. Neues Dialogfenster mit Diagramm-eigenschaften



Einige Symboltypen sind auf mehreren verschiedenen Diagrammen vorhanden. Anknüpfend an das obige Beispiel gibt es in Rational System Architect aber auch Typen von Statusdiagrammen mit Statussymbolen, wie z. B. das IDEF3-Objektstatusübergangsdiagramm, das OV-06b Op-Statusübergangsdiagramm und das Ward & Mellor-Statusübergangsdiagramm. Wenn die Eigenschaften *Short Description* (Kurzbeschreibung)und *Number* (Nummer) in diesen drei Typen angezeigt werden sollen, muss der Eigenschaftenblock dreimal hinzugefügt werden, nämlich einmal pro Diagrammtyp.

```
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 }
}
```

Angeben von Eigenschaften für Definitionstypen

Alle Definitionen verfügen über einen *Namen*. Zusätzlich weist jede Definition die Standardeigenschaft *Beschreibung* auf, auf die später in diesem Abschnitt näher eingegangen wird. Es gibt eigentlich kein typisches **Definitionsdialog-fenster**, da Definitionen in der Regel innerhalb einer Methodologie und innerhalb eines Typs eindeutig sind. Unten ist ein Dialogfenster für eine Klassendefinition dargestellt.

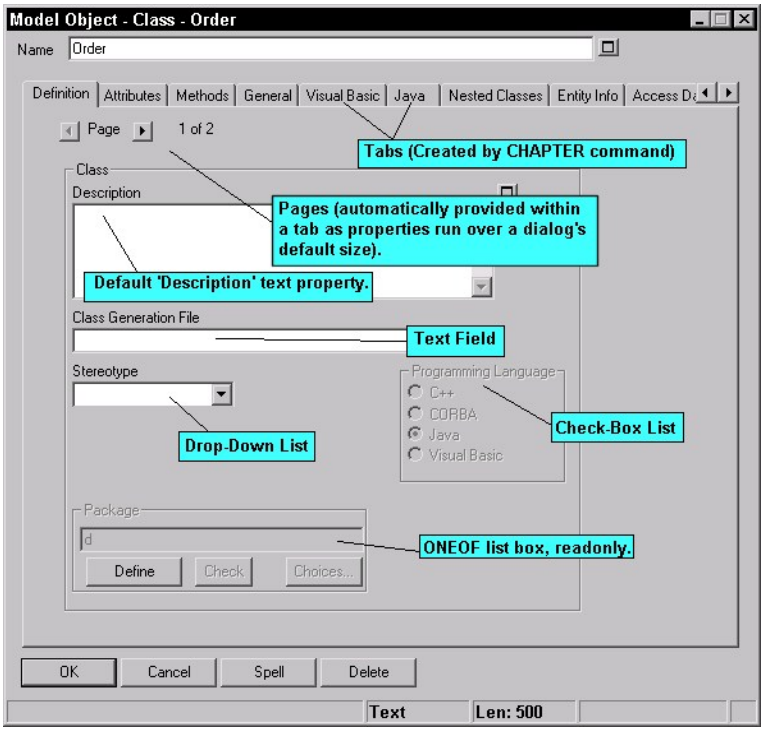


Abbildung 2-17. Dialogfenster für ein Modellobjekt (Definitionstyp "Klasse")

Syntax

Ein Definitionsblock beginnt mit dem Schlüsselwort *DEFINITION*, gefolgt von einer Zeichenfolge (dem Argument), die den Namen des Definitionstyps darstellt. Der Name muss Rational System Architect bekannt sein. Entweder ist er in der SAPROPS.CFG-Datei enthalten, falls Sie eine bereits vorhandene Definition ändern oder ergänzen, oder Sie haben den Namen mithilfe der Befehle RENAME "User 1" bis RENAME "User 150" angelegt (zur Erstellung eines neuen Definitionstyps müssen Sie einen dieser 150 vom Benutzer bereitgestellten Definitionstypen umbenennen). Namen von Definitionstypen, die eingebettete Leerzeichen aufweisen, müssen in doppelte Anführungszeichen gesetzt werden (z. B. "User 1").

Die Verzeichnisdefinition wird zwischen die Schlüsselwörter `BEGIN...END` gesetzt (oder alternativ in geschweifte Klammern **{ }**). Innerhalb der Klammern befindet sich ein Satz aus Definitionsbefehlen, wobei jeder Befehl aus dem Befehlsschlüsselwort `PROPERTY`, gefolgt von den zugehörigen Argumenten, besteht. Wenn Sie das Dialogfenster **Verzeichnisobjekt** aufrufen, werden dessen Seiten mit den im Definitionsblock bezeichneten Eigenschaften befüllt.

Jeder PROPERTY-Eintrag verfügt über eine eigene Teildefinition, die wiederum mit dem Schlüsselwort in geschweifte Klammern gesetzt ist {**EDIT...** }. Jede Definition besteht aus Wortfolgen, die wiederum aus Schlüsselwörtern bestehen, wie z. B. *Boolean, Date, Expression, ExpressionOf, ListOf, Minispec, Numeric, OneOf, Text* und *Time*. Weitere Informationen über Eigenschaftsdefinitionen finden Sie weiter hinten in diesem Abschnitt.

Grundsätzlich können Sie einer Definition mit folgender Syntax weitere Eigenschaften hinzufügen:

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

Beispiel:

```
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 }
..}
```

Im obigen Beispiel wurde die standardmäßige erste Registerkarte der Definition mit dem Namen "Einführung" (sofern dieser nicht anderweitig festgelegt wurde) über den Befehl CHAPTER "Definition" in "Definition" geändert.

Beschreibung

Wie zu Beginn dieses Abschnitts erwähnt, weist jede Definition die Standardeigenschaft *Beschreibung* auf. Soweit in der SAPROPS.CFG-Datei nichts Anderweitiges festgelegt wurde, ist die *Beschreibung* als Textfeld mit 4074 Zeichen definiert. Sie können das Feld für eine Beschreibung in der USRPROPS.TXT-Datei vergrößern. Dazu genügt es, die Beschreibungseigenschaft neu anzugeben und die Anzahl der Zeichen zu erhöhen. Beispiel:

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

Mit dem obigen Beispiel wird festgelegt, dass die Beschreibungseigenschaft einer Klasse bis zu 16.000 Zeichen enthalten kann, wobei nur die ersten fünf Zeilen im Dialogfenster der Klassendefinition angezeigt werden.

Wichtige Anmerkung: Es gibt Definitionstypen innerhalb von Rational System Architect, bei denen die Beschreibungseigenschaft für ganz bestimmte Zwecke verwendet wird. So wurde beispielsweise die Definition einer Entität neu definiert als LISTOF "Attribute" FROM "Data. Die Eigenschaft *Beschreibung* wurde für Prozessdefinitionen neu definiert als *Minispec*, da der Dateninhalt von Prozessen im Allgemeinen aus Minispezifikationen, strukturiertem Englisch, Pseudocode usw. besteht. Bei jedem dieser Fälle wurde auch die Beschreibungseigenschaft in "Attribute" bzw. "Minispec" geändert. Das folgende Beispiel veranschaulicht die Spezifikation für eine Prozessdefinition in der SAPROPS.CFG-Datei:

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

Zu beachten ist, dass der Name der Eigenschaft bei der Erstellung von Berichten *Description* (Beschreibung)lautet und bei Verweisen entsprechend zu bezeichnen ist.

Eigenschaftsanweisungen

Eigenschaftsanweisungen werden innerhalb von Diagramm-, Symbol- oder Definitionsspezifikationen festgelegt. Die Syntax einer Eigenschaftsanweisung sieht wie folgt aus:

```
PROPERTY Eigenschaft-Name
{ EDIT bearbeiten-Typ
}
```

Die folgende Tabelle enthält alle obligatorischen und optionalen Einträge für eine Eigenschaftsanweisung in der USRPROPS.TXT-Datei.

Tabelle 2-3. Obligatorische und optionale Einträge für eine Eigenschafts-anweisung

Eintrag	Obligatorisch Optional	Anmerkung

PROPERTY property_name
{
}
	Obliga-torisch	Sie können entweder geschweifte Klammern {..} oder die Anweisung BEGIN .. END verwenden.

EDIT edit-type	Optional	[Boolean | Date | ExpressionOf DATA | ListOf "dictionary-type" | Minispec | Numeric | OneOf "dictionary-type" | Text | Time]
LABEL label_string	Optional	Name des Steuer-elements im Dialog-fenster; ersetzt den standardmäßigen Namen der Eigen-schaft.

Eintrag	Obligatorisch Optional	Anmerkung
LENGTH length-argument	Optional	Maximale Länge des Felds in Zeichen 0 < numerisch < 4095
LIST list-name	Optional	Gibt an, dass bei Auswahl der Eigenschaft zur Eingabe durch den Benutzer eine Liste des Listennamens angezeigt wird. Der Benutzer kann entweder einen Wert aus der Liste wählen oder einen eigenen Wert eingeben.
LISTONLY LIST list-name	Optional	Gibt an, dass ausschließlich ein Eintrag aus einer optional angezeigten Liste zulässig ist.
MINIMUM numeric MAXIMUM numeric	Optional	Minimal/maximal zulässiger numerischer Wert des Felds
DISPLAY { FORMAT format-type LEGEND legend-name }	Optional	Definiert eine von 37 möglichen anzeigbaren Eigenschaften für ein Symbol
DEFAULT default_string	Optional	Falls kein Benutzereintrag erfolgt, wird der Eintrag der Zeichenfolge verwendet.

Tabelle 2-3. Obligatorische und optionale Einträge für eine Eigenschafts-anweisung (Fortsetzung)

Eintrag	Obligatorisch Optional	Anmerkung
READONLY	Optional	Unterbindet alle Eingaben über die Tastatur oder eine angezeigte Liste
INVISIBLE VISIBLE	Optional	Blendet die Eigenschaft aus oder ein. Mit diesem Eintrag können Sie den Wert in der SAPROPS.CFG-Datei umkehren.
CHECKOUT initial-type	Optional	Wert, der auto-matisch ausgeführt wird, wenn der Verzeichniseintrag ausgecheckt wird. [DATE | TIME | AUDITID]
FREEZE initial-type	Optional	Wert, der auto-matisch ausgeführt wird, wenn der Verzeichniseintrag blockiert wird. [DATE | TIME | AUDITID]

Eintrag	Obligatorisch Optional	Anmerkung
INITIAL initial-type	Optional	Wert, der automatisch ausgeführt wird, wenn der Verzeichniseintrag erstmalig aufgerufen und gespeichert wird. [DATE | TIME | AUDITID]
UPDATE update-type	Optional	Wert, der automatisch jedes Mal ausgeführt wird, wenn der Verzeichniseintrag aufgerufen und gespeichert wird. [DATE | TIME | AUDITID]
HELP	Optional	35-40 Zeichen, die in der Statusleiste des Dialogfensters angezeigt werden, wenn sich das Steuerelement im Fokus befindet

Verwenden von ListOf, OneOf und ExpressionOf

Die Schlüsselwörter ListOf, OneOf und ExpressionOf bilden ein äußerst leistungsstarkes Konzept, das im gesamten Metamodell von Rational System Architect zur Anwendung kommt. Mit jedem dieser Schlüsselwörter können Sie angeben, dass eine Eigenschaft einer Definition auf einen anderen Objekttyp verweisen soll – sei es auf ein Diagramm, ein Symbol oder eine Definition.

So können Sie beispielsweise angeben, dass eine Klasse eine Liste mit Methoden enthalten soll (ListOf-Befehl), eine Nachricht zwischen zwei Objekten auf eine Methode im aufrufenden Objekt verweisen soll (OneOf-Befehl), oder ein Prozess die auf Daten ausgeführten Prozeduren ausdrücken soll (ExpressionOf-Befehl). In den nachfolgenden Abschnitten werden wir uns der Reihe nach mit diesen drei Ausdrücken befassen.

Anmerkung: Wie bei allen Schlüsselwörtern, die in der USRPROPS.TXT-Datei angegeben werden, ist die Groß- und Kleinschreibung der Schlüsselwörter ListOf, OneOf und ExpressionOf nicht relevant. Mit Ausnahme dieses Abschnitts werden in diesem Handbuch für alle Schlüsselwörter Großbuchstaben verwendet. In diesem Fall kommt jedoch die Schlüsselwortfunktion in der Schreibweise ListOf, OneOf und ExpressionOf besser zur Geltung als in der Schreibweise LISTOF, ONEOF und EXPRESSIONOF.

ListOf

Mit dem ListOf-Befehl können Sie angeben, dass eine Eigenschaft eine Liste anderer Objekte enthalten soll, z. B. Diagramme, Symbole oder Definitionen. So enthält beispielsweise eine Klasse eine Eigenschaft mit der Bezeichnung "Attribute", die aus einer Liste mit Klassenattributen besteht. Ein Klassenattribut ist an sich ein eigener Definitionstyp mit einem eigenen Satz aus Eigenschaften. Der Objekttyp, auf den verwiesen werden soll, muss bereits in der SAPROPS.CFG-Datei oder am Anfang der USRPROPS.TXT-Datei definiert worden sein.

Sie können die ListOf-Eigenschaft mit einer einfachen Textliste vergleichen. Die Anzahl der Elemente in der ListOf-Liste steigt mit dem Hinzufügen von Definitionen zum Repository durch die Benutzer. Bei einer einfachen Liste ist die Anzahl der Elemente in der dem Benutzer angezeigten Liste statisch (und basiert auf der LIST-Anweisung im Kopfsatz der USRPROPS.TXT-Datei).

Die Syntax des ListOf-Befehls sieht wie folgt aus:

```
PROPERTY "Ihre Eigenschaft" { EDIT LISTOF
<"Verweisdefinitionsytp"> LENGTH 1200}
```

Filtern der Liste der Einträge

Die Liste der Einträge innerhalb der Liste für einen ListOf-Befehl kann gefiltert werden. Es sind verschiedene Filterschlüsselwörter verfügbar, wie z. B. OF DEFINITION REFERENCED IN und OF DEFINITION AND SUPERS REFERENCED IN. Weitere Informationen zu diesen Schlüsselwörtern finden Sie in Kapitel 3.

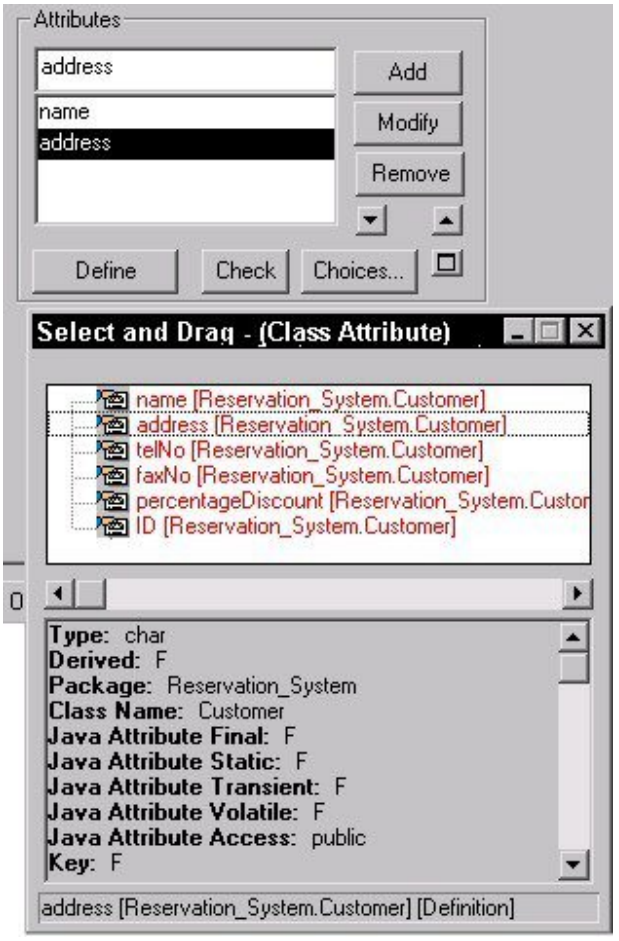
Beispiel:

Das Beispiel unten veranschaulicht den Code für eine Objektdefinition, in der die Eigenschaft "Attribute" enthalten ist, die wiederum aus einer Liste mit Klassenattributen (ListOf "Class Attributes") besteht. "Klassenattribute" ist ein weiterer Definitionstyp, der in der SAPROPS.CFG-Datei definiert ist.

```
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 "$$FORCE$$"}  }
..}
```

Die Abbildung unten zeigt das ListOf-Standarddialogfenster, wie es durch den LISTOF-Befehl erstellt wird. Es enthält die Schaltfläche "Optionen", über die eine Liste aller Definitionen des Typs, auf den verwiesen wird (in diesem Fall des Typs "Klassenattribut"), in der Enzyklopädie angezeigt werden kann. Durch den Befehl OF DEFINITION REFERENCED IN im obigen Code wird angegeben, dass nur die Klassen-attribute angezeigt werden sollen, die in der Objektklasse enthalten sind.

Abbildung 2-18. Beispiel für eine LISTOF-Liste



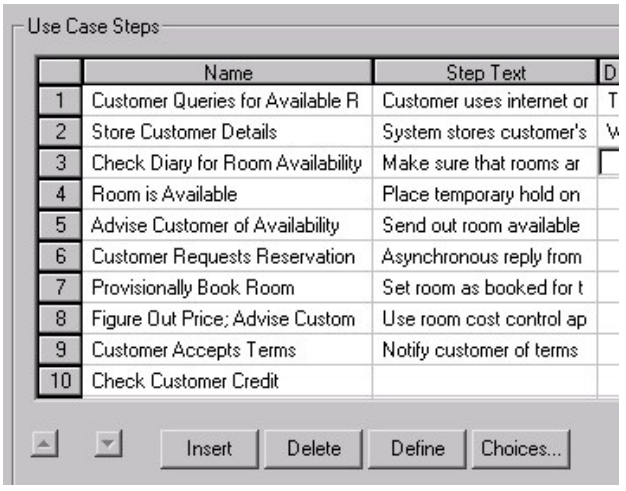
Erstellen eines ListOf-Rasters

Sie können die Elemente einer ListOf-Eigenschaft als Raster darstellen, indem Sie das Schlüsselwort ASGRID hinzufügen.

Beispiel:

```
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 }
}
```

Abbildung 2-19. Beispiel für eine LISTOF ASGRID-Liste



Heterogene Listen für ListOf

Bei einer typischen ListOf-Anweisung wird eine Liste eines Objekttyps bereitgestellt. Mithilfe des Schlüsselworts HETEROGENEOUSLISTOF können Sie jedoch auch eine Liste erstellen, die auf mehrere Objekttypen verweist.

Beispiel:

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

Im obigen Beispiel kann die Eigenschaft "Underlying Procedure" (zugrunde liegende Prozedur) der Definition "Procedure" (Prozedur) mit Definitionen des Typs "Use Case" (Anwendungsfall) und/oder "Class" (Klasse) und/oder "Method" (Methode) und/oder "Use Case Step" (Anwendungsfallschritt) befüllt werden.

Weitere Informationen zum Befehl HETEROGENEOUSLISTOF finden Sie in Kapitel 3.

OneOf

Ein OneOf-Listenfeld enthält ein Listenfeld, über das der Benutzer ein einziges Objekt aus einer Objektliste eines bestimmten Typs auswählen kann (Diagramm, Symbol oder Definition). Der Objekttyp, auf den verwiesen werden soll, muss bereits in der SAPROPS.CFG-Datei oder am Anfang der USRPROPS.TXT-Datei definiert worden sein.

Beispiel:

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

Abbildung 2-20. Beispiel für ein ONEOF-Listenfeld

Filtern der Liste der Einträge

Genau wie bei der ListOf-Liste kann die Liste der Einträge innerhalb der Liste für einen ListOf-Befehl gefiltert werden. Es sind verschiedene Filterschlüsselwörter verfügbar, wie z. B. OF DEFINITION REFERENCED IN und OF DEFINITION AND SUPERS REFERENCED IN. Weitere Informationen zu diesen Schlüsselwörtern finden Sie in Kapitel 3.

Hetereogene OneOf-Liste

Bei einer typischen OneOf-Anweisung wird eine Liste eines Objekttyps bereitgestellt. Genau wie bei der ListOf-Liste, können Sie mithilfe des Schlüsselworts HETEROGENEOUS**ONE**OF auch eine OneOf-Liste erstellen, die auf mehrere Objekttypen verweist.

Weitere Informationen zu diesen Schlüsselwörtern finden Sie in Kapitel 3.

ExpressionOf

ExpressionOf ermöglicht das Ausdrücken von Verweisen auf Objekte, die komplexe Operatoren und Begrenzer verwenden. Zwar sieht Rational System Architect die Verwendung von ExpressionOf für den Verweis auf Datenelemente und Datenstrukturen (DATA) vor, jedoch ist dies nicht die einzige mögliche Verwendung.

Der Eintrag der mit ExpressionOf definierten Verweise in ein Dialogfenster erfolgt mithilfe folgender Syntax:

A + B + C

oder

A +
B +
C

oder

A
B
C

Die Elemente können in einer oder mehreren Zeilen eingegeben werden, wobei die Trennung zwischen den einzelnen Elementen durch ein Leerzeichen erfolgt. Gemäß Konvention werden die einzelnen Datenelemente durch ein Pluszeichen (+) voneinander getrennt, dies ist jedoch nicht zwingend erforderlich.

Beim Angeben von Ausdrücken können die folgenden besonderen Operatoren und Begrenzer verwendet werden:

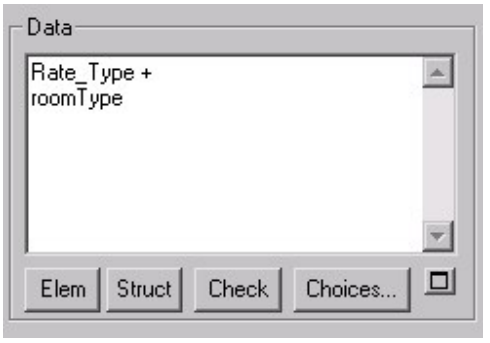
Tabelle 2-5. Besondere Operatoren und Begrenzer für das Angeben von Ausdrücken

+	und (optional)
[...|...]	entweder-oder
{...}	Iterationen von
i{...}j	Iterationen von 'i' nach 'j' zulassen
(...)	Die Komponente zwischen den Zeichen ist optional.
@	Die Komponente ist ein Schlüsselfeld.
@n	Die Komponente ist das n-te Element der Elemente eines Verbundschlüssels.
...	Der Text zwischen den Zeichen ist ein Kommentar.
/.../	Der Text zwischen den Zeichen ist ein Kommentar, ist aber von Bedeutung für den Schemagenerator.

Unterausdrücke können mit anderen Ausdrücken verschachtelt werden. So kann beispielsweise ITERATIONS OF zwischen EITHER und OR eingeschlossen werden.

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

Abbildung 2-21. Beispiel für ein EXPRESSIONOF-Listenfeld



ZOOMABLE-Befehl

Mit dem **ZOOMABLE**-Befehl kann der Benutzer die Größe eines Listenfelds vorübergehend erweitern, um größere Textblöcke besser eingeben oder anzeigen zu können. Üblicherweise wird dieser Befehl in Prozessdefinitionen verwendet, in die in der Regel Minispezifikationen eingegeben werden, oder in der Beschreibungseigenschaft einer Entität, wo die Fremdschlüsselinformationen häufig ziemlich lang sind.

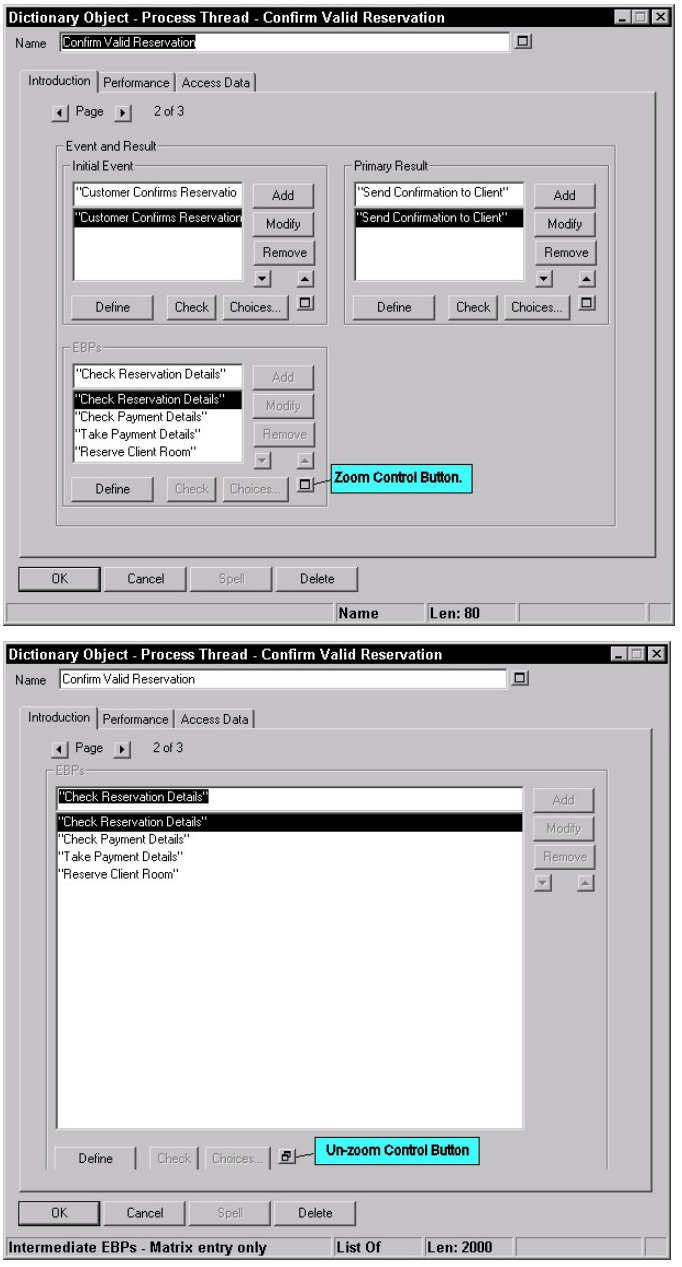
Der Befehl wird wie folgt in die USRPROPS.TXT-Datei eingegeben:

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

Durch den **ZOOMABLE**-Befehl wird in der rechten Ecke des Listenfelds eine kleine Schaltfläche hinzugefügt. Die Schaltfläche weist ein Pluszeichen auf, wenn das Feld nicht vergrößert ist, und ein Minuszeichen, wenn das Feld vergrößert ist.

Der Effekt des Befehls wird in den beiden Darstellungen in Abbildung 2-22 veranschaulicht. Das erste Bild zeigt den Bereich mit der Minispezifikation im nicht vergrößerten Zustand. Im unteren Bild wurde das Listenfeld vergrößert und nimmt die gesamte Seite des Dialogfensters ein.

Abbildung 2-22.
Listenfeld in normaler Größe und vergrößert



Ändern der Darstellung von Dialogfenstern

Die Anzeige der Steuerelemente und ihrer Beschriftungen innerhalb des Dialogfensters kann bis zu einem gewissen Grad konfiguriert werden. Die Beschriftung beispielsweise kann entweder oberhalb, direkt neben oder in einem gewissen Abstand zu einem Steuerelement angezeigt werden, wobei sich der Abstand aus der längsten Beschriftung innerhalb einer Gruppe ergibt.

Es ist nicht erforderlich, die Anzahl der Steuerelemente zu ermitteln, die auf einer Dialogseite untergebracht werden können. Rational System Architect errechnet die Anzahl der Steuerelemente, die auf einer Dialogseite untergebracht werden können, automatisch anhand des für die Anzeige verfügbaren Platzes und unterteilt das Dialogfenster ggf. in mehrere Seiten, die über einen Seitenpfeil in der oberen linken Ecke des Dialogfensters angezeigt werden können.

Sie können eine eigene Anordnung der Steuerelemente in Ihrem Dialogfenster festlegen. Mit dem **LAYOUT**-Befehl können Sie Spalten und die Positionierung von Beschriftungen für Steuerelemente und deren Textausrichtung angeben. Mit dem **CHAPTER**-Befehl können Sie Registerkarten erstellen und mit dem **GROUP**-Befehl Gruppen von Steuerelementen für bestimmte Eigenschaften. Darüber hinaus können Sie die exakte Platzierung der einzelnen Steuerelemente und Beschriftungen für jede beliebige Seite eines Dialogfensters angeben, indem Sie bestimmte Steuerelemente für die Positionierung verwenden (Anweisungen finden Sie auf Seite 2-103, *Positionierung von Steuerelementen und Beschriftungen*).

LAYOUT-Befehl

Mithilfe des LAYOUT-Befehls können Sie angeben, wie viele Spalten mit Steuerelementen für Eigenschaften in einem Dialogfenster angelegt werden sollen, wo der Titel (oder die Beschriftung) des jeweiligen Steuerelements platziert werden soll (links neben dem Steuerelement oder darüber) usw.

Die Verwendung des LAYOUT-Befehls ist optional. Wenn Sie den Befehl nicht verwenden, stellt Rational System Architect das Standardlayout bereit. Beim Standardlayout sind alle Steuerelemente in einer Spalte angeordnet. Der Name (oder die Beschriftung) der einzelnen Steuerelemente wird links neben dem Steuerelement angezeigt (ALIGN LABEL-Befehl).

Sie können einen LAYOUT-Befehl innerhalb von Kapiteln (entspricht einer Registerkarte in dem sich ergebenden Dialogfenster) und GRUPPEN einer Diagramm-, Symbol- oder Definitionsspezifikation angeben. Der LAYOUT-Befehl wirkt sich in Kapiteln und Gruppen folgendermaßen aus:

Innerhalb eines Kapitels: Sie können einen eindeutigen LAYOUT-Befehl für jedes Kapitel einer Diagramm-, Symbol- oder Definitionsspezifikation angeben. Alle Steuerelemente für Eigenschaften sowie auch vollständige Gruppen werden gemäß dem LAYOUT-Befehl für das Kapitel angeordnet. Wenn Sie innerhalb eines Kapitels mehrere LAYOUT-Befehle angeben, werden diese vollständig ignoriert und stattdessen das Standardlayout verwendet.

Innerhalb einer GRUPPE: Sie können innerhalb einer Gruppe einen LAYOUT-Befehl angeben, um die Eigenschaften der Gruppe gemäß der LAYOUT-Spezifikation für die Gruppe anzuordnen. Wenn Sie mehrere LAYOUT-Befehle innerhalb einer Gruppe angeben, werden diese vollständig ignoriert und stattdessen das Standardlayout verwendet.

Beispielhafte Anordnung von LAYOUT-Befehlen und ihre Wirkung:

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

```
    PROPERTY – Anordnung (in Kapitel) gemäß LAYOUT 1
    PROPERTY – Anordnung (in Kapitel) gemäß LAYOUT 1
    GROUP  – Anordnung (in Kapitel) gemäß LAYOUT 1
      LAYOUT 2
        PROPERTY – Anordnung (in Gruppe) gemäß LAYOUT 2
        PROPERTY – Anordnung (in Gruppe) gemäß LAYOUT 2
    GROUP  – Anordnung (in Kapitel) gemäß LAYOUT 1
      LAYOUT 3
        PROPERTY – Anordnung (in Gruppe) gemäß LAYOUT 3
        PROPERTY – Anordnung (in Gruppe) gemäß LAYOUT 3
  CHAPTER 2
   LAYOUT 4
      PROPERTY – Anordnung (in Kapitel) gemäß LAYOUT 4
  CHAPTER 3
   LAYOUT 5
      PROPERTY – Anordnung gemäß Standardlayout, da zwei
LAYOUT-Befehle (5 und 6) im Kapitel vorhanden sind
   LAYOUT 6
```

Layout der Registerkarte "Einführung"

Beachten Sie, dass das erste Kapitel eines Diagramm-, Symbol- oder Definitionsdialogfensters, in dem die Beschreibungseigenschaft enthalten ist, gemäß dem Standardlayout angeordnet wird. Bei diesem einspaltigen Layout befindet sich die Beschriftung "Beschreibung" links neben dem Textfeld.

Standardlayout-verhalten

Wenn ein Steuerelement für eine Eigenschaft so breit ist, dass es nicht in die angegebene Spaltenstruktur eines LAYOUT-Befehls passt, wird dieses Steuerelement in einer eigenen Spalte angeordnet, damit es im Dialogfenster bzw. in der Gruppe untergebracht werden kann. Die anderen Steuerelemente, deren Breite mit dem LAYOUT-Befehl kompatibel ist, werden entsprechend dem LAYOUT-Befehl angeordnet.

Angenommen, Sie haben für eine Gruppe ein vierspaltiges Layout angegeben und die Gruppe ist ihrerseits in einem Kapitel (einer Registerkarte) untergebracht, für das ein zweispaltiges Layout angegeben wurde. Wenn nun eine der Eigenschaften der Gruppe zu breit für den verfügbaren Raum ist, die anderen Eigenschaften jedoch in dem vierspaltigen Layout Platz fänden, wird die zu breite Eigenschaft separat angeordnet und die anderen Eigenschaften entsprechend dem für die Gruppe angegebenen vierspaltigen Layout.

Beispiel

Im folgenden Beispiel untersuchen wir den Effekt des LAYOUT-Befehls in CHAPTER- und GROUP-Anweisungen innerhalb einer neuen, vom Benutzer festgelegten, Definition.

```
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}
}
```

Der USRPROPS.TXT-Code wird in den nachfolgenden Abbildungen genauer untersucht. Die erste Abbildung macht deutlich, dass der oberste LAYOUT-Befehl der Definition (LAYOUT { COLS 3 ALIGN OVER TAB }) ignoriert wird, weil er keinem Kapitel zugewiesen ist und er das Layout der ersten Registerkarte "Einführung", für das standardmäßig ein einspaltiges Layout verwendet wird, nicht überschreiben kann.

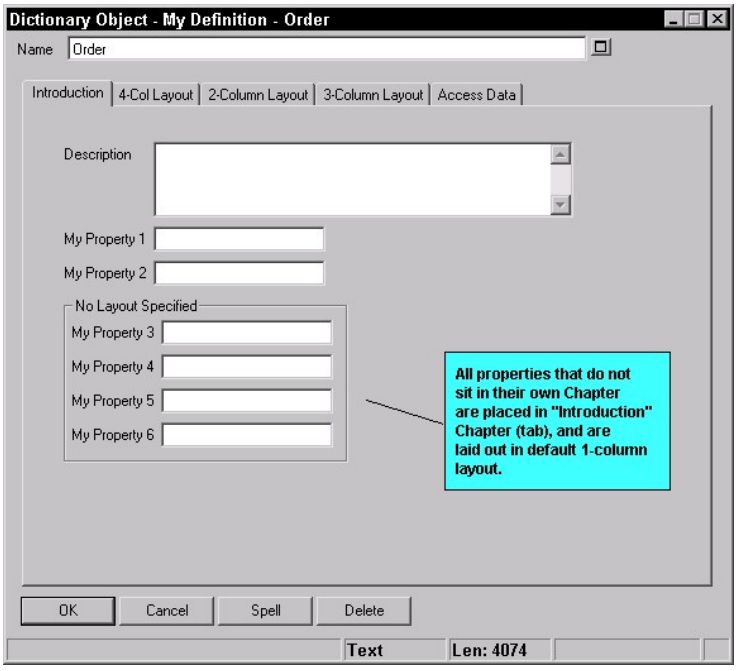
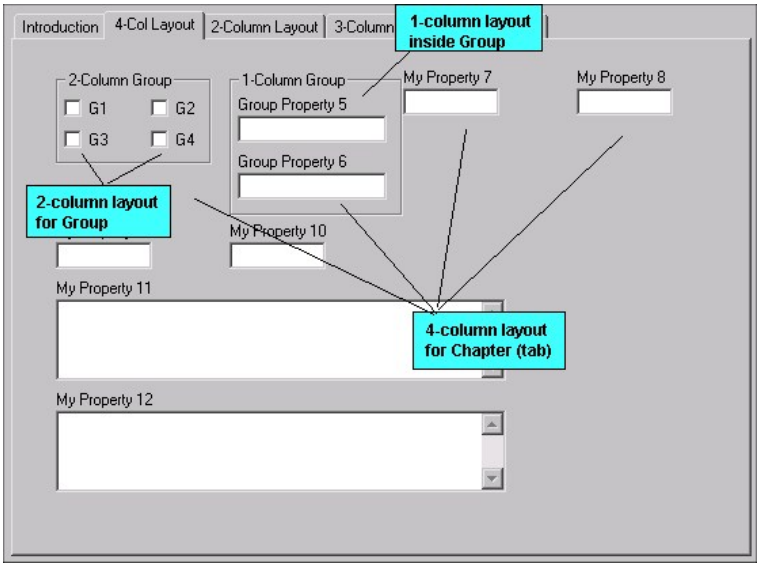


Abbildung 2-23. Der erste LAYOUT-Befehl wird ignoriert, weil er keinem Kapitel (keiner Registerkarte) zugewiesen ist.

Die zweite Registerkarte des Dialogfensters wird durch den Befehl CHAPTER "4-Col Layout" festgelegt. Bei diesem vierspaltigen Layout werden der Titel bzw. die Beschreibung der einzelnen Steuerelemente oberhalb des Steuerelements angeordnet (CHAPTER "4-Col Layout" LAYOUT { COLS 4 ALIGN OVER TAB }.

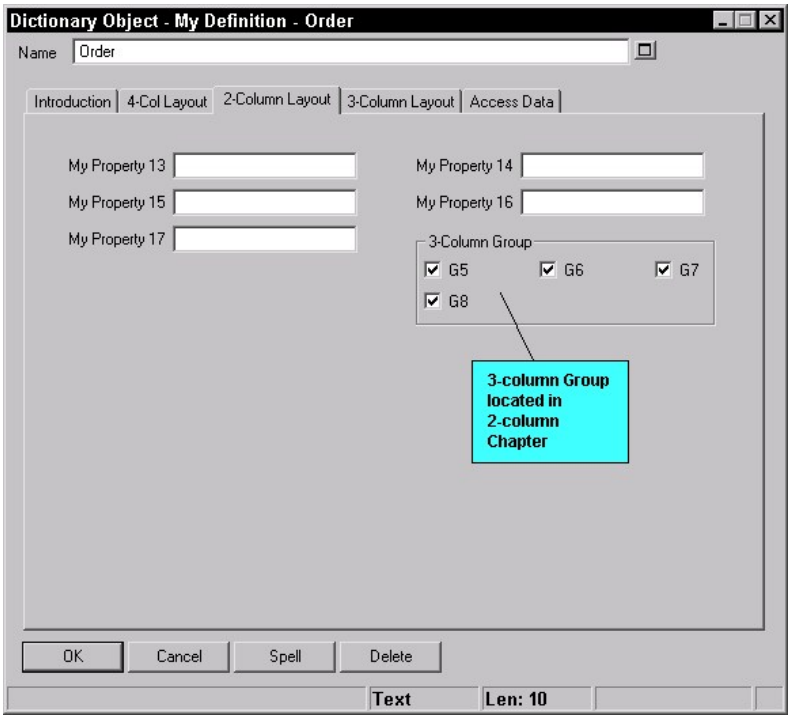
Wie Sie der Abbildung unten entnehmen können, werden sogar ganze Gruppen (wie z. B. "2-Column Group" und "1-Column Group") im Kapitel innerhalb eines vierspaltigen Layouts angeordnet, ebenso wie die Eigenschaften (wie z. B. "My Property 7" bis "My Property 10"). Eigenschaften, die zu breit für das vierspaltige Layout sind, werden in einer Spalte angeordnet (wie z. B. "My Property 11" und "My Property 12", beide mit der Länge 1200).

Abbildung 2-24. Vierspaltiges Kapitel mit ein- und zweispaltigen Gruppen (1-Column Groups und 2-Column Groups)



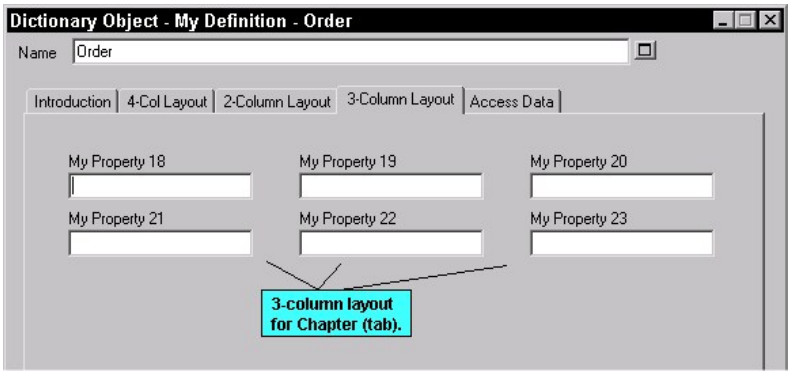
Das zweispaltige Kapitel enthält ebenfalls Eigenschaften, die in zwei Spalten angeordnet sind, darunter eine Gruppe, deren Eigenschaften in drei Spalten angeordnet sind.

Abbildung 2-25. Zweispaltiges Kapitel mit dreispaltiger Gruppe



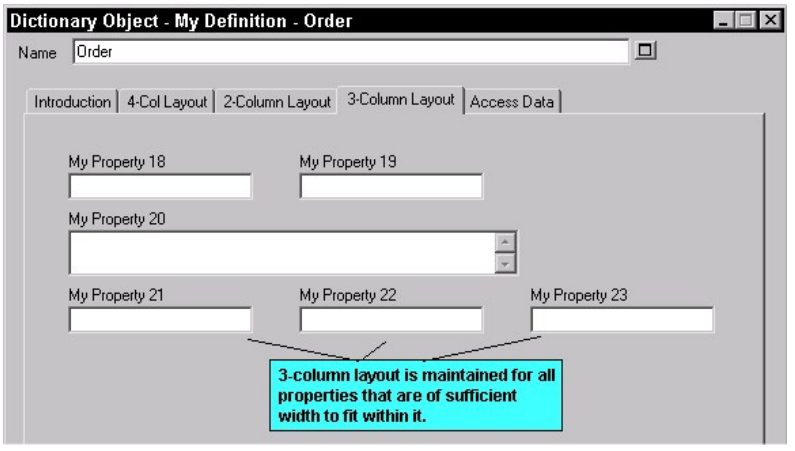
Das letzte Kapitel enthält Eigenschaften mit einer dreispaltigen Anordnung. Hier sind die Eigenschaften schmal genug (Länge 10), um in drei Spalten Platz zu finden.

Abbildung 2-26. Zweispaltiges Kapitel mit dreispaltiger Gruppe



Falls eine dieser Eigenschaften zu breit für das dreispaltige Layout wäre, würde diese unabhängig von den anderen Eigenschaften in einem einspaltigen Format angeordnet werden. Die Änderung von "My Property 20" von LENGTH 10 in LENGTH 100 bewirkt, dass das Steuerelement für diese Eigenschaft wie in der Abbildung unten angezeigt wird. Die anderen Eigenschaften des Dialogfensters werden nach wie vor in einem dreispaltigen Format angezeigt.

Abbildung 2-27. Zweispaltiges Kapitel mit dreispaltiger Gruppe und einer zu breiten Eigenschaft



LAYOUT-Befehls-argumente

Die gültigen Werte für die im **LAYOUT**-Befehl verwendeten Unterbefehle lauten:

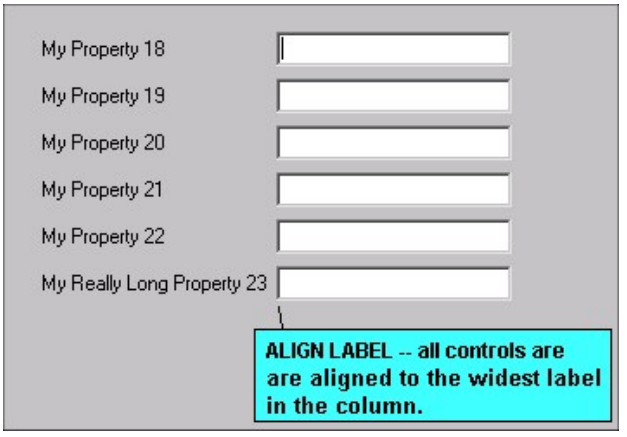
LAYOUT { [ALIGN BODY | ALIGN LABEL | ALIGN OVER] [PACK | TAB] COLS <number> **}**

Die Reihenfolge der Unterbefehle ist unerheblich.

Ausrichten von Eigenschaftstiteln (oder Beschriftungen) am zugehörigen Steuerelement

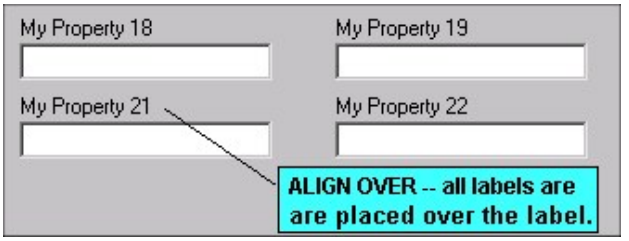
Jede Eigenschaft verfügt über einen Titel oder Namen. Wie Sie wissen, können Sie eine Eigenschaft mithilfe des LABEL-Befehls umbenennen. Durch den ALIGN-Befehl werden Titel einer Eigenschaft bzw. deren Beschriftung, falls sie umbenannt wurde, an eine bestimmte Position in der Nähe des Steuerelements platziert. Dabei gibt es folgende Möglichkeiten:

1. **ALIGN BODY** und **ALIGN LABEL**: Alle Steuerelemente werden im Abstand von einem Leerzeichen rechts neben der längsten Beschriftung der Spalte angeordnet.



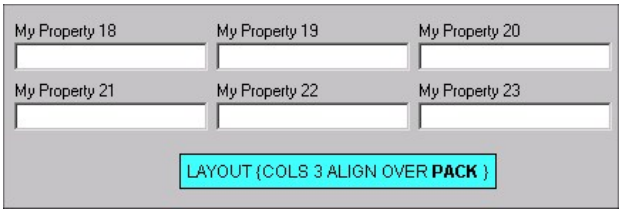
(Anmerkung: Mit ALIGN BODY wurden früher alle Steuerelemente im Abstand von einem Leerzeichen rechts neben der Beschriftung angeordnet. Mittlerweile sind ALIGN BODY und ALIGN LABEL identisch.)

2. **ALIGN OVER**: Die Beschriftung wird oberhalb des Steuerelements angeordnet.

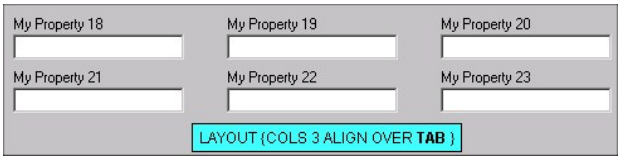


Vertikale Positionierung

1. **PACK***:* Sätze mit Steuerelementen und Beschriftungen in mehreren Spalten, werden rechts durch einen minimalen Abstand vom jeweils nächsten Satz getrennt.



2. **TAB**: Die Steuerelemente und Beschriftungen in mehreren Spalten werden durch Tabulatoren voneinander getrennt, sodass die Einträge in den einzelnen Zeilen bündig angeordnet sind.



Spalten

COLS <number_of_columns>: Steuert die Anzahl der Spalten, in die die Eigenschaften unterteilt werden

Ausrichten

JUSTIFY: Dieser Befehl wird für die SAPROPS.CFG- und die USRPROPS.TXT-Datei nicht mehr verwendet. Er wird vom USRPROPS.TXT-Parser ignoriert. Der Befehl diente früher dazu, alle Steuerelemente an der linken und rechten Seite des Dialogfensters auszurichten.

Erstellen von Registerkarten mit dem CHAPTER-Befehl

Mit dem **CHAPTER**-Befehl können Sie den Inhalt einer Dialogseite steuern und *Registerkarten* erstellen. Wenn in der Registerkarte mehr Informationen enthalten sein sollen, als angezeigt werden können, werden automatisch mehrere Seiten innerhalb der Registerkarte erstellt.

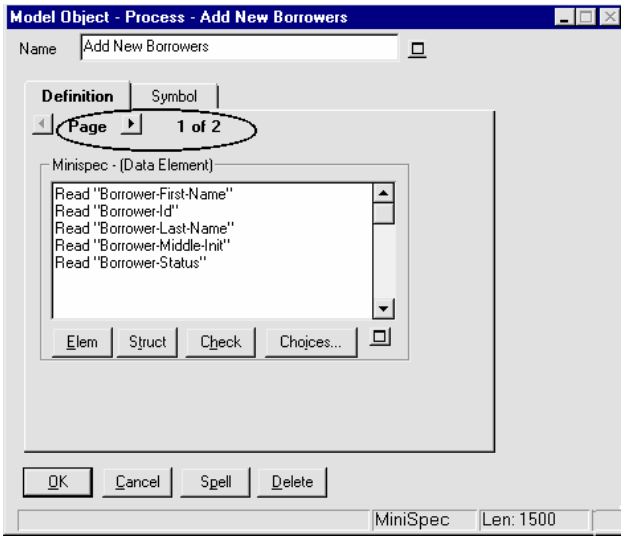


Abbildung 2-28. Das Dialogfenster "Modellobjekt" verfügt über eine zweiseitige Registerkarte "Definition".

Syntax und Positionierung des CHAPTER-Befehls

Wenn Sie eine Registerkarte erstellen möchten, müssen Sie den CHAPTER-Befehl verwenden und den Namen der Registerkarte als Argument angeben (setzen Sie den Namen in doppelte Anführungszeichen, falls dieser über eingebettete Leerzeichen verfügt). Beim CHAPTER-Befehl ist die Verwendung von geschweiften Klammern { } bzw. eines BEGIN .. END-Anweisungsblocks nicht erforderlich.

CHAPTER Name_der_Registerkarte

oder

CHAPTER "Name der Registerkarte"

Alle Eigenschaften, die nach dem CHAPTER-Befehl in der Spezifikation aufgeführt werden, werden dem Kapitel (bzw. der Registerkarte) zugeordnet, bis ein neuer CHAPTER-Befehl vorliegt. Der **CHAPTER**-Befehl kann an einer beliebigen Stelle innerhalb einer Diagramm-, Symbol- oder Definitionsspezifikation der USRPROPS.TXT-Datei eingefügt werden, jedoch nicht innerhalb eines GROUP-Blocks.

Anmerkung: Für diesen Befehl wurde der Begriff CHAPTER gewählt und nicht der offensichtlichere Begriff TAB, da diesem innerhalb der USRPROPS.TXT-Datei schon immer eine andere Bedeutung zukam (wird im LAYOUT-Befehl verwendet).

Für den CHAPTER-Befehl gelten folgende Regeln:

- Eine Eigenschaft, die über die USRPROPS.TXT-Datei ohne die Verwendung des **CHAPTER**-Befehls hinzugefügt wird, wird an das Ende aller Eigenschaftsanweisungen für die Definition platziert, also auf die letzte Seite der letzten Registerkarte "Definition", aus denen das Definitionsdialogfenster besteht. (Beachten Sie, dass eine Definition mehrere mit dem Symbol verbundene Informationsregisterkarten aufweisen kann, die sich ganz am Ende des Definitionsdialogfensters befinden. Die erste der Symbolregisterkarten trägt häufig den Namen "Symbol", kann jedoch umbenannt werden. Die Symbolregisterkarten werden nur angezeigt, wenn Sie das Definitionsdialogfenster eines Symbols in einem Diagramm öffnen. Wenn Sie die Definition in Explorer öffnen, werden die Registerkarten nicht angezeigt.)
- Sie können einer bereits vorhandenen und in der SAPROPS.CFG-Datei aufgerufenen Registerkarte eine Eigenschaft hinzufügen, indem Sie die Registerkarte mithilfe eines **CHAPTER**-Befehls in der USRPROPS.TXT-Datei neu festlegen. Die von Ihnen hinzugefügten Eigenschaften werden an das Ende der Registerkarte des Dialogfensters gestellt.

- Registerkarten für neue **CHAPTER**-Befehle, die Sie zu einer bereits vorhandenen Diagramm-, Symbol- oder Definitionsspezifikation hinzufügen, werden im Anschluss an die bereits vorhandenen Registerkarten an das Ende des Dialogfensters gestellt.
- Wenn Sie einen **GROUP**-Befehl verwenden möchten, müssen Sie diesen in einen **CHAPTER**-Befehl verschachteln.

Verwenden des LAYOUT-Befehls innerhalb eines Kapitels

Der LAYOUT-Befehl kann an beliebiger Stelle unter einem CHAPTER-Befehl eingefügt werden und wirkt sich auf alle Steuerelemente für Eigenschaften innerhalb des Kapitels aus. Wenn Sie mehrere LAYOUT-Befehle in einem Kapitel angeben, weist der USRPROPS.TXT-Parser alle LAYOUT-Befehle zurück und verwendet stattdessen das Standardlayout (COLS 1 ALIGN LABEL).

GROUP-Befehl

Mit dem GROUP-Befehl können Sie einen Satz aus Steuerelementen für Eigenschaften in ein Gruppenfeld platzieren.

GROUP-Befehlssyntax

Die Syntax des Befehls sieht wie folgt aus:

GROUP Name_der_Gruppe

{ <in die Gruppe einzuschließende Eigenschaften>

}

oder

GROUP "Name der Gruppe"

{ <in die Gruppe einzuschließende Eigenschaften>

}

Wie aus obiger Syntax hervorgeht, müssen beim GROUP-Befehl die in die Gruppe einzuschließenden Eigenschaften in geschweiften Klammern { } angegeben werden. Wenn der Name der Gruppe eingebettete Leerzeichen aufweist, müssen Sie den Namen in doppelte Anführungszeichen setzen. Sie können auch keinen Namen für die Gruppe angeben, indem Sie doppelte Anführungszeichen ohne Textinhalt angeben. Z. B.:

GROUP ""

{ <in die Gruppe einzuschließende Eigenschaften>

}

Wichtige Anmerkung: Alle Gruppennamen innerhalb einer Diagramm-, Symbol- oder Definitionsspezifikation müssen eindeutig sein, auch wenn sie sich in unterschiedlichen Kapiteln befinden. Wenn Sie beispielsweise in einem Kapitel einer Definition eine Gruppe mit dem Namen "x" erstellen und anschließend eine zweite Gruppe "x" mit anderen Eigenschaften erstellen, werden alle Eigenschaften beider Gruppen mit dem Namen "x" in einer Gruppe "x" zusammengefasst, die im ersten Kapitel der Definition abgelegt wird. Wenn Sie innerhalb einer Spezifikation mehrere Gruppen mit demselben Namen erstellen möchten, fügen Sie ein bzw. mehrere Leerzeichen zwischen den einzelnen Instanzen der Gruppe ein. Auf das obige Beispiel bezogen, wäre der Name wie folgt zu ergänzen: "x ".

Verwenden des LAYOUT-Befehls innerhalb einer Gruppe

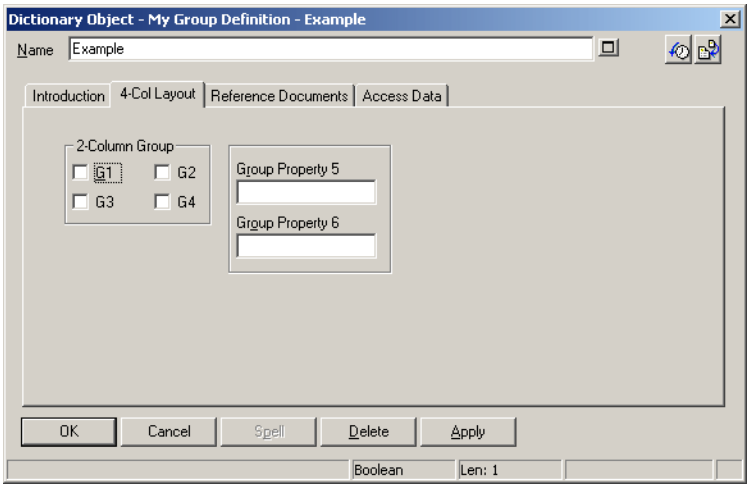
Sie können innerhalb einer Gruppe einen LAYOUT-Befehl angeben. In diesem Fall überschreibt er den LAYOUT-Befehl der Definition bzw. des Kapitels (der Registerkarte), in der sich die Gruppe befindet, jedoch lediglich bezogen auf die Eigenschaften innerhalb der Gruppe.

Beispiel

Mit dem folgenden USRPROPS.TXT-Code können die in der Abbildung unten dargestellten Gruppen erstellt werden.

```
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}
       }
}
```

Abbildung 2-29. Verwendung der Befehle GROUP und LAYOUT



Positionieren von Steuerelementen und Beschriftungen

Die Syntax für eine präzise Platzierung sieht wie folgt aus:

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

Beispiel 1:

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

Zusätzlich können Sie die Positionierung der Beschriftung angeben. Die Syntax für die präzise Platzierung einer Beschriftung sieht wie folgt aus:

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

Beispiel 2:

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

In Beispiel 1 oben werden die Steuerelemente durch PROPPOS (4,12) vier Fenstereinheiten horizontal und zwölf Fenstereinheiten vertikal angeordnet, beginnend oben links im Dialogfenster. Anders ausgedrückt: Die Steuerelemente werden von der oberen linken Ecke des Dialogfensters ausgehend, um vier Einheiten nach links und um zwölf Einheiten nach unten angeordnet. Durch die Angabe PROPSIZE (150,40) wird die Größe des Steuerelements mit 150 Fenstereinheiten Breite und 40 Fenstereinheiten Länge bestimmt.

In Beispiel 2 oben wird durch LABELPOS die Beschriftung des Steuerelements, ausgehend von der oberen linken Ecke des Dialogfensters, vier Fenstereinheiten horizontal und zwei Fenstereinheiten vertikal angeordnet. Die Beschriftung hat daher denselben Abstand zur Ecke des Dialogfensters, ist jedoch zehn Fenstereinheiten oberhalb des Steuerelements angeordnet.

Wichtige Anmerkung: Innerhalb eines Kapitels sollten Sie PLACEMENT-Befehle und Standardpositionierungsbefehle **nicht kombiniert verwenden**. Anderenfalls erhalten Sie nicht das gewünschte Ergebnis.

Es folgen ein Ausschnitt der Syntax der SAPROPS.CFG-Datei für das Dialogfenster **Entitätsdefinition**.

```
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)} }
}
```

Allgemeine Regeln für die Dimensionierung

In den meisten Fällen werden Sie einige Änderungen an der Länge (X-Koordinate) vornehmen müssen, um die gewünschte Größe zu erhalten.

1. Für Kontrollkästchen gilt PROPSIZE (30, 12).
2. Für OneOf-Eigenschaften gilt PROPSIZE (150, 40).
3. Für ListOf-Eigenschaften gilt PROPSIZE (320, 98).

4. Kurze Textfelder entsprechen in etwa PROPSIZE (<LENGTH*3>, 12), wobei die Breitenkoordinate (X) aus optischen Gründen abgerundet wird.
5. Lange Textfelder, wie z. B. LENGTH 4074, entsprechen in etwa PROPSIZE (150,115).

Allgemeine Regeln für die Platzierung

Auch hier werden Sie vermutlich einige Änderungen vornehmen müssen, nachdem Sie das Layout des Dialogfensters zum ersten Mal angezeigt haben. Die Angaben unten sind jedoch für den Anfang recht hilfreich.

Tabelle 2-4 Positionierung und Größe, wenn sich die Eigenschaft in einer Gruppe befindet und die Beschriftung oberhalb der Eigenschaft angeordnet ist

EIGENSCHAFTSTYP	PLATZIERUNG - linke Spalte
ListOf	PLACEMENT { PROPPOS (4, 24) PROPSIZE (320, 98) }
OneOf	PLACEMENT { PROPPOS (4, 24) PROPSIZE (150, 40) }
EDIT Text (weniger als 75 Zeichen)	PLACEMENT { PROPPOS (4, 24) PROPSIZE (LENGTH * 3, 12) }
EIGENSCHAFTSTYP	PLATZIERUNG - rechte Spalte
ListOf	PLACEMENT { PROPPOS (165, 24) PROPSIZE (320, 98) }
OneOf	PLACEMENT { PROPPOS (165, 24) PROPSIZE (150, 40) }
EDIT Text (weniger als 75 Zeichen)	PLACEMENT { PROPPOS (165, 24) PROPSIZE (LENGTH * 3, 12) }

Tabelle 2-4. Positionierung und Größe, wenn sich die Eigenschaft in einer Gruppe befindet und die Beschriftung nicht oberhalb der Eigenschaft angeordnet ist

EIGENSCHAFTSTYP	PLACEMENT - linke Spalte
ListOf	PLACEMENT { PROPPOS (4, 12) PROPSIZE (320, 98) }
OneOf	PLACEMENT { PROPPOS (4, 12) PROPSIZE (150, 40) }
EDIT Text **(weniger als 75 Zeichen)**	PLACEMENT { PROPPOS (4, 12) PROPSIZE (LENGTH * 3, 12) }
EIGENSCHAFTSTYP	PLATZIERUNG - rechte Spalte
ListOf	PLACEMENT { PROPPOS (165, 12) PROPSIZE (320, 98) }
OneOf	PLACEMENT { PROPPOS (165, 12) PROPSIZE (150, 40) }
EDIT Text **(weniger als 75 Zeichen)**	PLACEMENT { PROPPOS (165, 12) PROPSIZE (LENGTH * 3, 12) }

Tabelle 2-4 Positionierung und Größe, wenn sich die Eigenschaft nicht in einer Gruppe befindet und die Beschriftung nicht oberhalb der Eigenschaft angeordnet ist

EIGENSCHAFTSTYP	PLATZIERUNG - linke Spalte
ListOf	PLACEMENT { PROPPOS (4, 2) PROPSIZE (320, 98) }
OneOf	PLACEMENT { PROPPOS (4, 2) PROPSIZE (150, 40) }
EDIT Text **(weniger als 75 Zeichen)**	PLACEMENT { PROPPOS (4, 2) PROPSIZE (LENGTH * 3, 12) }
EIGENSCHAFTSTYP	PLATZIERUNG - rechte Spalte
ListOf	PLACEMENT { PROPPOS (165, 2) PROPSIZE (320, 98) }
OneOf	PLACEMENT { PROPPOS (165, 2) PROPSIZE (150, 40) }
EDIT Text **(weniger als 75 Zeichen)**	PLACEMENT { PROPPOS (165, 2) PROPSIZE (LENGTH * 3, 12) }

Tabelle 2-4 Positionierung und Größe, wenn sich die Eigenschaft nicht in einer Gruppe befindet und die Beschriftung oberhalb der Eigenschaft angeordnet ist

EIGENSCHAFTSTYP	PLATZIERUNG - linke Spalte
ListOf	PLACEMENT { PROPPOS (4, 14) PROPSIZE (320, 98) }
OneOf	PLACEMENT { PROPPOS (4, 14) PROPSIZE (150, 40) }
EDIT Text **(weniger als 75 Zeichen)**	PLACEMENT { PROPPOS (4, 14) PROPSIZE (LENGTH * 3, 12) }
EIGENSCHAFTSTYP	PLATZIERUNG - rechte Spalte
ListOf	PLACEMENT { PROPPOS (165, 14) PROPSIZE (320, 98) }
OneOf	PLACEMENT { PROPPOS (165, 14) PROPSIZE (150, 40) }
EDIT Text **(weniger als 75 Zeichen)**	PLACEMENT { PROPPOS (165, 14) PROPSIZE (LENGTH * 3, 12) }

Tabelle 2-4 Positionierung und Größe, wenn Eigenschaften innerhalb eines Dialogfensters untereinander angeordnet sind

EIGENSCHAFT **an den Endpunkt der vorangehenden Eigenschaft anhängen**	PLATZIERUNG - linke Spalte
A **+14 falls Beschriftung OBERHALB von Eigenschaft B**	PLACEMENT { PROPPOS (4, 14) PROPSIZE (150, 40) }
B **+ 4 falls Eigenschaft C OneOf**	PROPERTY B PLACEMENT { PROPPOSE (4, 68) PROPSIZE (150, 40) }
C	PLACEMENT { PROPPOS (4, 112) PROPSIZE (150, 12) }

EIGENSCHAFTSTYP	PLATZIERUNG - rechte Spalte
D **+ 4 falls Beschriftung von Eigenschaft E seitlich**	PLACEMENT { PROPPOS (165, 14) PROPSIZE (320, 98) }
E **+ 4 da Verwendung von LABELPOS für Eigenschaft F**	PLACEMENT { PROPPOS (165, 116) PROPSIZE (150, 40) }
F	PLACEMENT { LABELPOS (165, 160) PROPPOS (165, 170) PROPSIZE (30, 12) }

Anmerkung: Die Verwendung von LABELPOS ist optional. LABELPOS dient der Überschreibung des LAYOUT-Befehls für die Gruppe. Die Y-Koordinate ist um zehn Einheiten höher als die Y-Koordinate von PROPPOS.

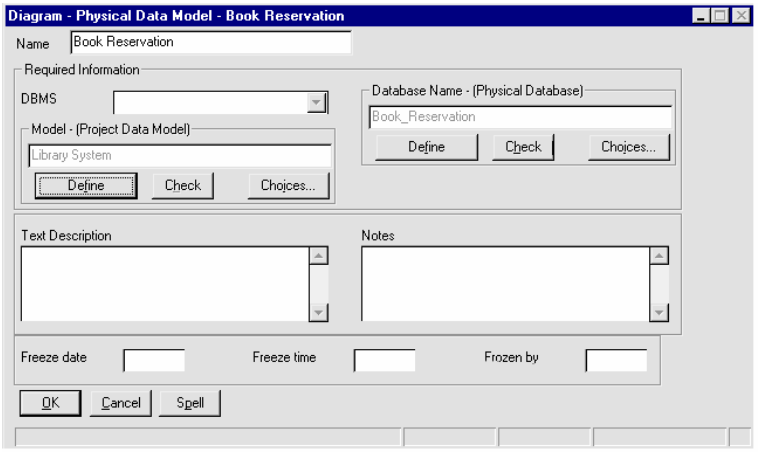


Abbildung 2-30. Ergebnis der CHAPTER-Befehle

Festlegen der Anzeige von Werten in Symbolen

Ein Symbol stellt eine Definition im Repository dar. Diese Definition weist bestimmte Eigenschaften auf. Sie können festlegen, dass diese Definitionseigenschaften und ihre Werte in einem Symbol angezeigt werden. Standardmäßig wird der Name eines Symbols (der eine Eigenschaft des Symbols und von dessen Definition darstellt) angezeigt. Wenn Sie festlegen möchten, dass andere Eigenschaften der Symboldefinition anzeigbar sein sollen, müssen Sie in der Deklaration der einzelnen Eigenschaften den **DISPLAY**-Befehl verwenden. Beispiel:

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

Der obige Beispielcode bewirkt, dass "My Property 1" (Meine Eigenschaft 1) auf dem Symbol anzeigbar ist. Text, den Sie in das 20 Zeichen große Textfeld der Eigenschaft im Dialogfenster eingeben, wird auf dem Symbol angezeigt.

Nachdem Sie bestimmte Eigenschaften einer Symboldefinition als anzeigbar festgelegt haben, werden diese Eigenschaften im Dialogfenster **Anzeigemodus** des Symbols angezeigt, wo sie jederzeit aktiviert oder inaktiviert werden können. Sie können das Dialogfenster "Anzeigemodus" aufrufen, indem Sie ein Symbol in einem Diagramm auswählen und anschließend **Ansicht, Anzeigemodus** wählen oder mit der rechten Maustaste auf ein Symbol klicken und die Option **Anzeigemodus** auswählen. Mithilfe des Dialogfensters "Anzeigemodus" können Sie die anzeigbaren Eigenschaften auswählen, die Sie auf dem Symbol anzeigen möchten.

Die Abbildung unten zeigt ein Entitätssymbol. Im linken Bild sind die anzeigbaren Eigenschaften des Entitätssymbols inaktiviert, im rechten Bild sind sie aktiviert.

Abbildung 2-31. Rechteckiges Symbol einmal mit und einmal ohne Anzeige der Eigenschaften

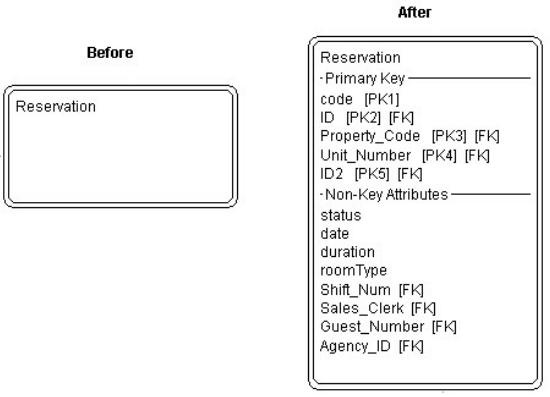
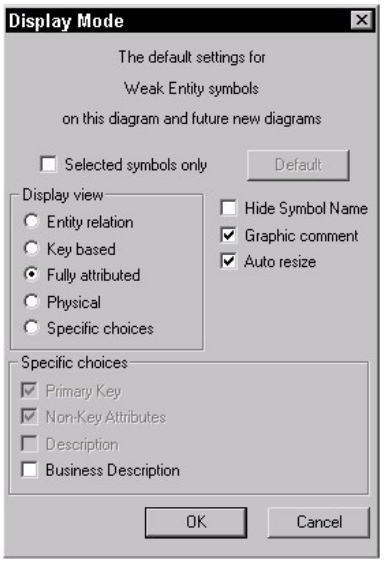


Abbildung 2-32. Beispiel-Dialogfenster "Anzeigemodus"



Sie können auch, wie in der Abbildung unten dargestellt, für Liniensymbole anzeigbare Eigenschaften festlegen.

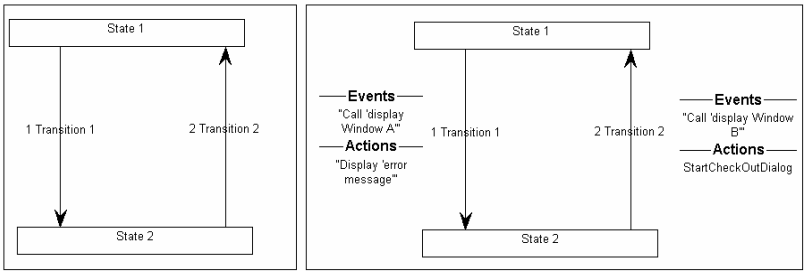


Abbildung 2-33. Liniensymbol einmal mit und einmal ohne Anzeige der Eigenschaften

Syntax des DISPLAY-Befehls

Jede Definition ist auf **acht** Anzeigeanweisungen begrenzt. Die Syntax des DISPLAY-Befehls sieht wie folgt aus:

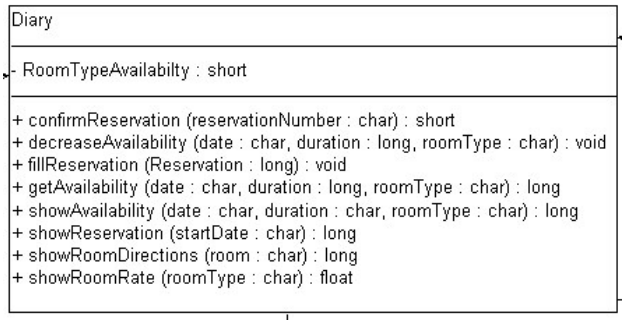
DISPLAY { FORMAT [STRING | LIST | KEY | NONKEY | COMPONENT_SCRIPT | COLUMN_SCRIPT | SCRIPT]
LEGEND " (Beschriftung des Blocks im Symbol) **" }**

- STRING: Dies ist die Standardeinstellung. Sie bewirkt, dass die Werte der Eigenschaft auf dem Symbol exakt so angezeigt werden, wie sie eingegeben wurden. Dies bietet sich an, wenn Sie Kommentare anzeigen möchten.
- LIST: Bewirkt, dass die Elemente auf dem Symbol in Form einer Liste angezeigt werden. Jedes Leerzeichen bewirkt einen Zeilenumbruch, es sei denn, es ist in doppelte Anführungszeichen gesetzt.
- KEY: Verwenden Sie dieses Schlüsselwort für Eigenschaften, die als Schlüssel designiert sind. Sie werden in einem gesonderten Abschnitt des Symbols angezeigt. Ein Beispiel finden Sie in Kapitel 3 im Abschnitt über das Schlüsselwort KEY.
- NONKEY: Dieses Schlüsselwort kann für Eigenschaften ohne Schlüsselfunktion verwendet werden. Sie werden in einem gesonderten Abschnitt des Symbols angezeigt. Dieses Schlüsselwort wurde ursprünglich für Entitäten und Tabellen in der Datenmodellierungsfunktion von Rational System Architect verwendet. Ein Beispiel finden Sie in Kapitel 3 im Abschnitt über das Schlüsselwort NONKEY.
- COMPONENT_SCRIPT: Ruft ein Script auf, durch das der Eigenschaftswert in einem vom Script vorgegebenen besonderen

Format auf dem Symbol angezeigt wird. Das Script ist entweder im Produkt fest codiert oder kann vom Benutzer mithilfe der Basic-Sprache von Rational System Architect erstellt werden. Gemäß Konvention wird dem Script eines der folgenden Präfixe vorangestellt:

- fmtxxx – Die Funktion selbst ist fest codiert und kann nicht geändert werden. Dies trifft auf die meisten Funktionen in der SAPROPS.CFG-Datei zu. Die feste Codierung von Funktionen wird deshalb vorgenommen, um die Reaktionszeit von Rational System Architect insgesamt zu verbessern.
- _fmtxxx – Ist in der FMTSCRIPT.BAS-Datei im Hauptverzeichnis für ausführbare Dateien von Rational System Architect vorhanden und wurde mit SA Basic codiert.

Die Komponentenscripts werden für ListOf- und ExpressionOf-Eigenschaften verwendet. Die vom Script ausgeführte Aktion wirkt sich auf alle Elemente der Liste aus. So wird beispielsweise ein Komponentenscript dazu verwendet, um die Klassenattribute in einer Klassendefinition zu analysieren und festzulegen, wie es auf dem Klassensymbol angezeigt wird. Wenn die Zugriffseigenschaft des Attributs auf privat eingestellt ist, wird vor dem Namen ein Minuszeichen (-) eingefügt bzw. ein Pluszeichen (+), wenn die Zugriffseigenschaft auf öffentlich gesetzt ist. Der Rückgabetyp des Attributs wird nach dem Attributnamen, hinter einem Doppelpunkt angezeigt. Analog wird durch das Komponentenscript festgelegt, wie eine Methode angezeigt wird. Ist ihr Zugriff öffentlich, wird dem Namen ein Pluszeichen (+) vorangestellt; ist der Zugriff privat, wird ein Minuszeichen (-) vorangestellt. Zusätzlich werden die Parameter der Methode und deren Typ in Klammern an das Ende des Methodennamens angefügt.



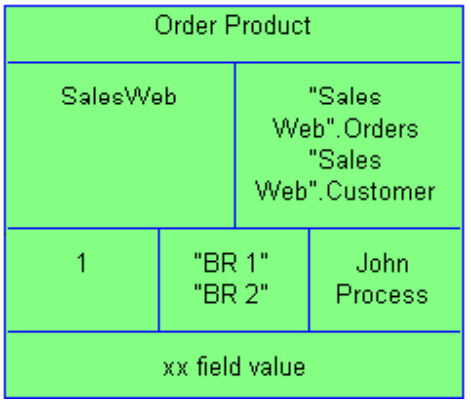
Weitere Informationen finden Sie in Kapitel 3 unter COMPONENT_SCRIPT und SCRIPT.

- COLUMN_SCRIPT: Funktioniert wie COMPONENT_SCRIPT und ruft ein Script auf, durch das dem auf einem Symbol angezeigten Eigenschaftswert ein besonderes Format zugewiesen wird. Das Script kann ein fest codiertes Script sein oder ein Script, das der Benutzer mithilfe von SA Basic geschrieben hat (und in der Datei FMTSCRIPT.BAS im Hauptverzeichnis für ausführbare Dateien von Rational System Architect vorhanden ist). Die Spaltenscripts werden verwendet, um Spalten in Tabellensymbolen in einem physischen Datenmodell anzuzeigen. Die vom Script ausgeführte Aktion wirkt sich auf alle Spalten der Liste aus. Weitere Informationen finden Sie in Kapitel 3 unter COLUMN_SCRIPT.
- SCRIPT: Funktioniert wie COLUMN_SCRIPT und COMPONENT_SCRIPT und ruft ein Script auf, durch das dem auf einem Symbol angezeigten Eigenschaftswert ein besonderes Format zugewiesen wird. Der SCRIPT-Befehl ruft Scripts auf, die für Eigenschaften verwendet werden, die weder ListOf- noch ExpressionOf-Eigenschaften sind. Das Script kann ein fest codiertes Script sein oder ein Script, das der Benutzer

mithilfe von SA Basic geschrieben hat (und in der Datei FMTSCRIPT.BAS im Hauptverzeichnis für ausführbare Dateien von Rational System Architect vorhanden ist). Ein Beispiel finden Sie in Kapitel 3 im Abschnitt über das Schlüsselwort SCRIPT.

Die angezeigten Eigenschaftengruppen werden durch eine Trennlinie voneinander getrennt. Mithilfe des LEGEND-Befehls können Sie angeben, dass auf der Trennlinie eine Beschriftung oder eine "Legende" angezeigt wird. Wenn kein LEGEND-Befehl angegeben wird, entspricht die Beschriftung dem Namen der Eigenschaft. Folgende LEGEND-Befehle sind verfügbar:

- **LEGEND "<Ihr Text>":** Text, den Sie zwischen die doppelten Anführungszeichen einfügen, wird auf dem Symbol oberhalb des Eintrags angezeigt, sofern ein Wert für den Eintrag vorhanden ist.
- **LEGEND "":** Zeigt eine gerade Linie ohne Text an, sofern ein Wert für den Eintrag vorhanden ist
- **LEGEND "$$FORCE$$":** Zeigt eine horizontale Linie oberhalb des Eintrags auf dem Symbol an. Diese Linie fungiert als Trennlinie. Das Schlüsselwort "$$FORCE$$" unterscheidet sich insofern von der bloßen Verwendung der doppelten Anführungszeichen " ", als es die Anzeige einer horizontalen Linie erzwingt, obwohl die Anzeige der Eigenschaft im Dialogfenster "Anzeigemodus" unterdrückt wurde.
- **LEGEND "$$NONE$$":** Zeigt keine horizontale Linie oberhalb des Eintrags im Symbol an, unabhängig davon, ob Werte für den Eintrag vorhanden sind oder nicht. Diese Linie fungiert normalerweise als Trennlinie.
- **LEGEND "$$VFORCE$$":** Ermöglicht die Anordnung von Eigenschaften von links nach rechts innerhalb eines Symbols und grenzt diese durch vertikale Linien voneinander ab. Beispiel:



Die USRPROPS.TXT-Beispieldatei, mit der das obige Bild erstellt wurde, finden Sie in Kapitel 3 im Abschnitt zum Schlüsselwort VFORCE.

- **LEGEND "$$VNONE$$":** Ermöglicht die Anordnung von Eigenschaften von links nach rechts, jedoch *ohne* Trennlinie. Ein Beispiel finden Sie in Kapitel 3 im Abschnitt über das Schlüsselwort VNONE.

Schriftbild und Schriftart der anzeigbaren Legenden können Sie über den Befehl **Diagrammformat**, **Notation** im Menü **Format** festlegen.

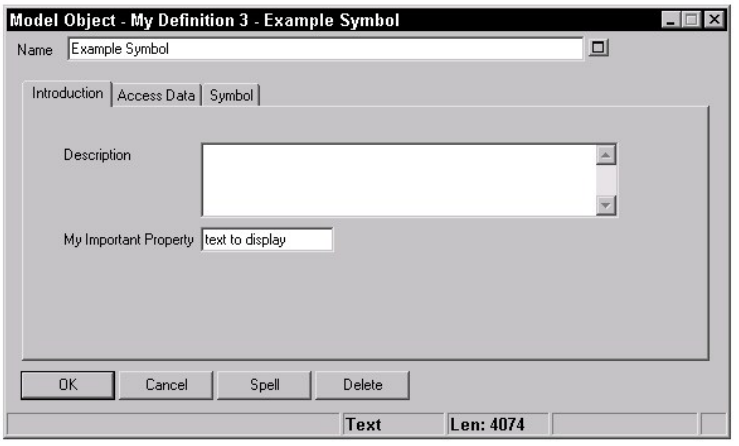
Beispiel:

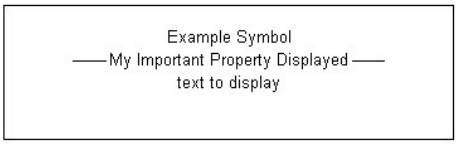
Im folgenden Beispiel werden ein neuer Diagrammtyp, ein neuer Symboltyp und ein neuer Definitionstyp festgelegt. Wir geben an, dass der neue Symboltyp durch den neuen Definitionstyp definiert werden soll und weisen den neuen Symboltyp dem neuen Diagrammtyp zu. Für die Definition erstellen wir die Eigenschaft "My important Property" (Meine wichtige Eigenschaft). Wir geben an, dass die Legende "My Important Property Displayed" (Anzeige meiner wichtigen Eigenschaft) im Symbol, auf der Trennlinie oberhalb des angezeigten Werts, angezeigt werden soll.

```
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" }}
}
```

Die Abbildung unten zeigt das Symbol, das in einem solchen Diagramm gezeichnet wurde, und den angezeigten Wert, wie er in das Eigenschaftsfeld eingegeben wurde.

Abbildung 2-34. Liniensymbol einmal mit und einmal ohne Anzeige der Eigenschaften





Angeben von KEY-Eigenschaften und KEYED BY-Eigenschaften

Erstellen von KEY-Eigenschaften

Sie können angeben, dass eine bestimmte Definition für eine oder mehrere andere Definitionen als Schlüssel designiert sein soll. Ein Schlüssel bestimmt den Namensbereich einer Definition in der Enzyklopädie. Der Definitionstyp "Klassenattribut" beispielsweise wird durch seine übergeordnete Klassendefinition sowie durch die übergeordnete Paketdefinition dieser Klasse bestimmt. Sie können also über zwei Attribute mit der Bezeichnung "Name" (Name) verfügen, wovon eines zur Klasse "Customer" (Kunde) im Paket "Reservation_System" (Reservierung System) gehört und das andere zur Klasse "Product" (Produkt) im Paket "Order_System" (Auftrag_System). Die beiden Attribute sind trotz ihrer Namensgleichheit völlig unterschiedliche Definitionen.

Standardmäßig wird jedes Modellierungselement in einer Enzyklopädie bereits im Hintergrund durch drei Dinge bestimmt: Seine *Klasse* (hier ist der Begriff *Klasse* im Sinne von Rational System Architect zu verstehen, wobei nach Diagramm, Symbol und Definition unterschieden wird), seinen *Typ* (Diagramm für UML-Anwendungsfall, BPMN-Prozessdiagramm usw.) und seinen *Namen* (z. B. ob es sich um ein Diagramm für den Anwendungsfall Reservation_System oder den Anwendungsfall Human_Resource_System handelt).

Mithilfe des KEY-Befehls können Sie einer Definition KEY-Eigenschaften hinzufügen. Den KEY-Befehl geben Sie innerhalb der Eigenschaft an, die Sie als Schlüssel für eine Definition festlegen möchten. Der KEY-Befehl kann innerhalb der Beschreibung einer Eigenschaft nahezu überall eingefügt werden. Aufgrund seiner Bedeutung ist es jedoch üblich, ihn als erstes Element innerhalb der geschweiften Klammern der Eigenschaft, direkt vor das Schlüsselwort EDIT, einzufügen.

Anmerkung: Es ist nicht möglich, KEY EDIT ONEOF einem Diagramm hinzuzufügen.

Beispiel 1:

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

Beispiel 2:

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

Anmerkung: KEY-Eigenschaften einer Definition werden nicht in einem über den Befehl ASGRID erstellten Raster angezeigt. Bei der Definition eines Anwendungsfalls etwa werden die Schritte des Anwendungsfalls in einem Raster abgebildet, das über den Befehl ASGRID erstellt wurde. Die KEY-Eigenschaften dieser Schritte (Eignerpaket und Anwendungsfall) werden jedoch nicht im Raster angezeigt.

Bei einer Eigenschaft, die einen Schlüssel darstellt und auf eines oder mehrere andere Objekte verweist, z. B. eine LISTOF- oder ONEOF-Eigenschaft (keine einfache Text- oder Zahleneigenschaft), muss der Benutzer die Klasse und den Klassentyp der Objekte, auf die verwiesen wird, angeben, wenn er beim Arbeiten mit Rational System Architect einen Wert für die Eigenschaft eingibt.

Beispiel:

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

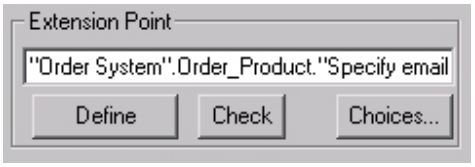
Die Anweisung oben gibt an, dass sich die Eigenschaft "Use Case Name" (Anwendungsfallname) auf eine Definition des Typs "Use Case" (Anwendungsfall) bezieht. Die Definition ist der Standardwert, wenn keine *Klasse* angegeben wurde (*Klasse* im Sinne von Rational System Architect: Diagramm, Symbol oder Definition).

Der Eigenschaftswert selbst enthält meist das verbleibende Material, das für die Identifizierung des oder der Objekte benötigt wird, auf das/die verwiesen wird. Wenn die Klasse/der Typ der Eigenschaft, auf die/den verwiesen wird, keine KEY-Eigenschaften aufweist, besteht der Verweiswert lediglich aus dem Namen des Objekts (da Klasse und Typ bekannt sind). Wenn aber die Klasse/der Typ, auf den verwiesen wird, KEY-Eigenschaften aufweist (wie z. B. "Use Case" im obigen Beispiel, mit der KEY-Eigenschaft "Package"), muss Rational System Architect den Wert dieser KEY-Eigenschaften kennen, um das Verweisobjekt richtig identifizieren zu können.

Anmerkung: Heterogene Verweiseigenschaften sind in diesem Zusammenhang etwas anderes. Weitere Informationen finden Sie in Kapitel 3 unter dem Stichwort HETEROGEN.

Sie codieren dies entweder in der Datei USRPROPS.TXT, sodass Rational System Architect automatisch die Werte für den Endbenutzer abruft, oder Sie erzwingen, dass der Endbenutzer den vollständig qualifizierten Namen, mit Punkten als Trennzeichen für die Schlüsselabschnitte, eingibt.

- Bei Verwendung des KEYED BY-Befehls ruft Rational System Architect den Wert für Benutzer automatisch ab.
- Wenn für die Eigenschaft keine KEYED BY-Klausel gegeben ist, geht Rational System Architect davon aus, dass diese zusätzlichen Schlüsselwerte im Verweis selbst enthalten sind. Anders ausgedrückt: Der Benutzer muss den vollständig qualifizierten Namen des Verweisobjekts eingeben und dabei Punkte als Trennzeichen zwischen den Schlüsselwerten verwenden. Für einen Anwendungsfallschritt mit dem Namen "*Specify email" (E-Mail angeben)*, in einem Anwendungsfall mit dem Namen *Order_Product* (Auftrag_Produkt), in einem Paket mit dem Namen "*Order System"* (Auftrag System), müsste der Benutzer Folgendes eingeben: "*Order System".Order_Product."Specify email"*.



Anmerkung: Wenn eine Komponente ein syntaktisch relevantes Zeichen enthält (z. B. ein Leerzeichen oder einen Punkt), muss dieses in doppelte Anführungszeichen gesetzt werden, damit Rational System Architect eine ordnungsgemäße Syntaxanalyse des Verweises durchführen kann.

Hier zwei Beispiele für Verweise auf einen Anwendungsfallschritt:

Order_System.Order_Product."Specify email"

wobei *CorrectInvoice* (RichtigeRechnung) der Name des zum Paket *Accounts_Payable* (Buchhaltung_Kreditoren) gehörigen Anwendungsfalls ist.

"Order System"."Order Product"."Specify email"

wobei *Order Product* (Auftrag Produkt) der Name des zum Paket *Order System* (Auftrag System) gehörigen Anwendungsfalls ist.

Mit KEYED BY sicherstellen, dass alle Mitglieder einer Gruppe denselben Typ aufweisen

Eine andere Verwendung der KEYED BY-Klausel ist die Erstellung einer Liste zusammengehöriger Elemente. So gehören beispielsweise alle Anwendungsfallschritte, auf die in der Eigenschaft "Anwendungsfallschritte" einer Anwendungsfalldefinition verwiesen wird, zum selben Anwendungsfall, nämlich zu dem, der die Eigenschaft "Anwendungsfallschritt" aufweist. Wenn eine Eigenschaft mit mehreren Verweisen (wie z. B. ListOf) auf Objekte verweist, die alle demselben übergeordneten Objekt angehören, empfiehlt es sich, das übergeordnete Objekt anhand von einem oder mehreren zusätzlichen Eigenschaften zu identifizieren. In diesem Fall gibt die KEYED BY-Klausel an, welche weiteren Eigenschaften Rational System Architect verwenden soll.

Verwendung der KEYED BY-Klausel

Zusammenfassend lässt sich festhalten, dass die KEYED BY-Klausel optional verwendet werden kann, um anzugeben, wie die Schlüsselkomponenten eines oder mehrerer Objekte, auf das/die verwiesen wird, gefunden werden können. Die Verwendung der Klausel hat zwei wesentliche Vorteile:

1. Der Benutzer muss nicht den vollständig qualifizierten Namen eines Verweiswerts (mit Punkten als Trennzeichen zwischen den Qualifikationsmerkmalen) eingeben. Im Falle einer Eigenschaft beispielsweise, die auf ein Klassenattribut mit dem Namen ***email*** (E-Mail) der Klasse ***Customer*** (Kunde) des Pakets ***"Order System"*** (Auftrag System) verweist, muss der Benutzer nicht ***"Order System".Customer.email*** eingeben, sondern einfach nur ***email***.
2. Sie kann verwendet werden, um sicherzustellen, dass alle Schlüsselkomponenten eines Verweiswerts identisch sind. So weist etwa die Eigenschaft LISTOF "Class Attribute" in einer Klassendefinition eine Liste mit Attributen auf, die alle derselben Klasse und demselben Paket angehören.

Eine KEYED BY-Klausel weist in der Regel eine Spezifikation auf, die Aufschluss darüber gibt, wie die einzelnen Schlüsselkomponenten des oder der Objekte, auf das/die verwiesen wird, gefunden werden können. Die KEYED BY-Klausel weist durch Kommata getrennte Abschnitte für die einzelnen Schlüsselkomponenten auf.

Beispiel:

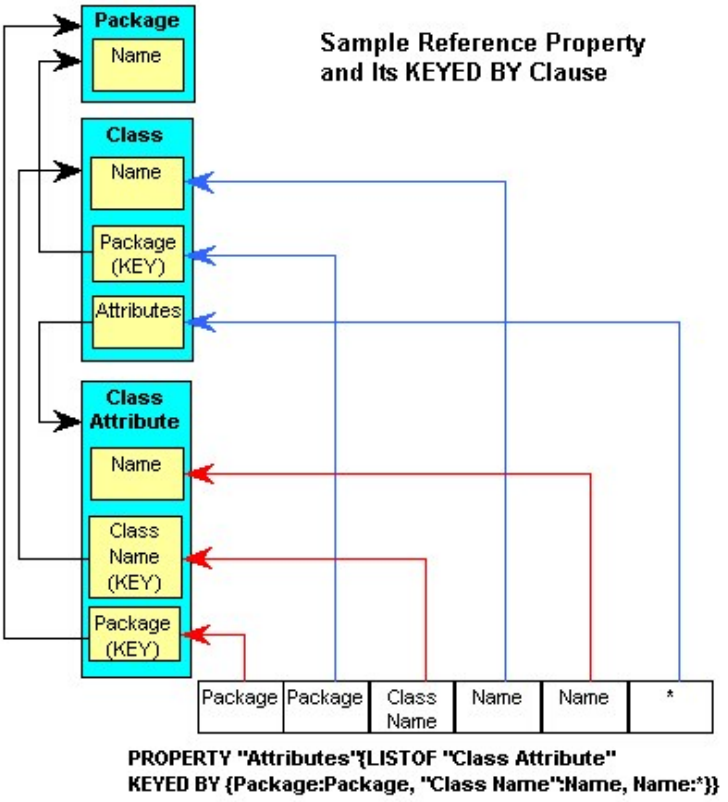
Die KEYED BY-Klausel der Eigenschaft "Attributes" der Klasse würde wie folgt aussehen:

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

Im obigen Beispiel lauten die drei durch Kommata getrennten *Schlüsselkomponenten* Package:Package, "Class Name":Name und Name:*. Die Komponenten verweisen auf die drei Abschnitte, die erforderlich sind, um die Klassenattributdefinitionen, auf die verwiesen wird, zu identifizieren: den Paketnamen, den Klassennamen und den Klassenattributnamen. In umgekehrter Reihenfolge betrachtet, wird Folgendes angegeben:

- Der Name des Klassenattributs ist in ***dieser*** Eigenschaft (* also "hier") enthalten, sprich:
  Name:*
- Der Wert der KEY-Eigenschaft "Class Name" in der Klassenattributdefinition ist im Namen dieses Objekts enthalten, sprich:
  "Class Name":Name
- Der Wert der KEY-Eigenschaft "Package" in der Klassenattributdefinition ist in der Paketeigenschaft dieses Objekts enthalten, sprich:
  Package:Package

Die folgende schematische Darstellung zeigt die Verwendung der KEYED BY-Klausel im obigen Beispiel und veranschaulicht das Wesen der KEYED BY-Klausel im Allgemeinen.



Das Schema veranschaulicht, was weiter oben bereits beschrieben wurde, nämlich, dass ein Klassenattribut in die Definition einer Klasse eingegeben wird, indem sein Paket (gespeichert in der Paketeigenschaft des Klassenattributs und abrufbar über den Paketwert der Klasse, in der Sie sich befinden), sein Klassenname (gespeichert in der Eigenschaft "Class Name" (Klassenname) des Klassenattributs und abrufbar über den tatsächlichen Namen der Klasse) und sein Name (gespeichert in der Eigenschaft "Name" des Klassenattributs und abrufbar über sich selbst) angegeben wird.

Zusammenfassung:

1. Die KEYED BY-Klausel verfügt für jede Schlüsselkomponente des **Verweisobjekts** über eine Komponente.
2. Die Komponenten der KEYED BY-Klausel werden durch ein Komma voneinander getrennt.
3. Jede Komponente besteht aus zwei Teilen:
   - Der erste Teil identifiziert die Schlüsselkomponente des Verweisobjekts.
   - Der zweite Teil gibt an, wo diese Komponente enthalten ist.
   - Beide Teile sind durch einen Doppelpunkt voneinander getrennt.

Zur Vereinfachung der KEYED BY-Klausel können jedoch bestimmte Standardwerte vorausgesetzt werden. Wenn beide Teile der Komponente identisch sind, kann der zweite Teil weggelassen werden. Wird der zweite Teil der letzten Komponente weggelassen, wird unterstellt, dass er "hier" zu finden ist, dargestellt durch den Stern. In der Praxis wird die KEYED BY-Klausel der Eigenschaft "Attribute" der Klasse daher wie folgt codiert:

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

Natürlich müssen alle in der KEYED BY-Anweisung verwendeten Eigenschaften vorhanden sein. Rational System Architect prüft daher, ob eine Eigenschaft "Paket" und eine Eigenschaft "Klassenname" in der Definition "Klassenattribut" vorhanden ist und beide als KEY-Eigenschaft designiert sind.

Abgesehen von der Tatsache, dass die Verwendung einer KEYED BY-Klausel dem Endbenutzer die Eingabe gemeinsamer Schlüsselkomponenten in eine LISTOF-Eigenschaft erspart (z. B. "Order System".Customer.email), stellt die KEYED BY-Klausel bei Verwendung anderer Eigenschaften zur Bereitstellung gemeinsamer Werte sicher, **dass für jeden Verweis dieselben Werte verwendet werden**. In unserem Beispiel gehören alle Klassenattribute, auf die in der Eigenschaft "Attributes" der Klasse verwiesen wird, zwangsweise derselben Klasse innerhalb desselben Pakets an, was in diesem Fall wünschenswert ist.

In anderen Situationen bietet es sich aus Gründen der Verdeutlichung und Übersicht an, die Schlüsselkomponenten des Verweisobjekts zu separieren. In so einem Fall werden durch die KEYED BY-Klausel die Eigenschaften designiert, durch die die separaten Komponenten bereitgestellt werden. Wenn eine Eigenschaft als KEY-Eigenschaft designiert ist und auf ein Objekt mit KEY-Eigenschaften verweist, **verlangt** Rational System Architect deshalb sogar, dass die Komponenten in separaten Eigenschaften vorhanden sind.

Beispiele für KEY-Eigenschaften und KEYED BY-Eigenschaften

Definition, die durch eine andere angegeben ist

Angenommen, Sie möchten Fahrzeuge nach "Brand" (Marke) und "Model" (Modell) kategorisieren. Dazu erstellen Sie eine neue Definition mit dem Namen "Car Brand" (Fahrzeugmarke) (Fahrzeugmarken sind z. B. Ford, Volkswagen, Toyota usw.) und eine weitere Definition mit dem Namen "Car Model" (Fahrzeugmodell) (die Werte, wie z. B. Mustang, Passat oder Corolla aufweist).

Für jedes "Car Model" (Fahrzeugmodell) muss eine "Car Brand" (Fahrzeugmarke) (auch "Make" (Fabrikat) genannt) angegeben sein. "Car Brand" ist eine Eigenschaft, mit der Sie das "Car Model" eindeutig identifizieren können. Jedes "Car Model" ist ausschließlich einer "Car Brand" zugewiesen.

Wir legen "Make" als KEY-Eigenschaft von "Car Model" fest, wobei der eigentliche Definitionstyp, der für diese Eigenschaft eingetragen wird, "Car Brand" ist. Wir designieren die Eigenschaft außerdem als REQUIRED, d. h. die Eigenschaft muss bei der Erstellung der Definition im Eröffnungsdialogfenster angegeben werden, damit die Definition erstellt wird.

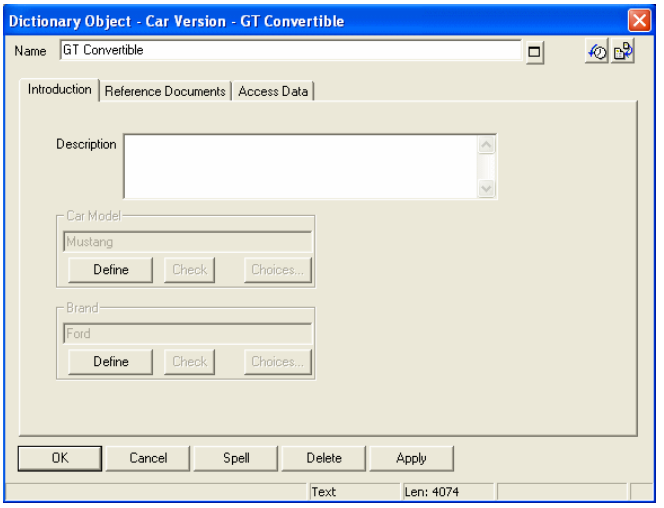
```
RENAME DEFINITION "User 1" To "Car Brand"
RENAME DEFINITION "User 2" To "Car Model"

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

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

Anmerkung: In die obige USRPROPS.TXT-Datei haben wir absichtlich ein Problem eingebaut, mit dem wir uns später in diesem Abschnitt befassen werden.

Dritte Definition mit zwei KEY-Eigenschaften

Gehen wir einen Schritt weiter. Für jedes Fahrzeugmodell gibt es eine Version, z. B. Mustang Coupé, Cabrio, GT Coupé, GT Cabrio, Mach 1 oder SVT Cobra. Diese Liste kann sich laufend verändern, deshalb legen wir einen Definitionstyp an und keine statische Liste. Der Name des Definitionstyps lautet "Car Version" (Fahrzeugversion). Wenn der Benutzer eine Fahrzeugversion erstellt, muss er gleichzeitig die Fahrzeugmarke und das Fahrzeugmodell angeben (da es auch andere Fahrzeugmodelle der Version GT geben kann).

```
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"}
}
```



Erstellen einer ListOf-Liste einer KEYED-Definition

Für jedes Fahrzeugmodell erstellen wir eine Liste der Fahrzeugversionen, die das Modell anbietet. Wir erstellen eine ListOf-Eigenschaft, die es dem Benutzer ermöglicht, in die Definition "Car Model" Fahrzeugversionen einzugeben. Beachten Sie, dass "Car Version" ein Definitionstyp mit einem Verbundschlüssel ist. Wenn der Benutzer die Fahr-

zeugversion eingibt, muss er also auch die Fahrzeugmarke und das Fahrzeugmodell der Fahrzeugversion eingeben.

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

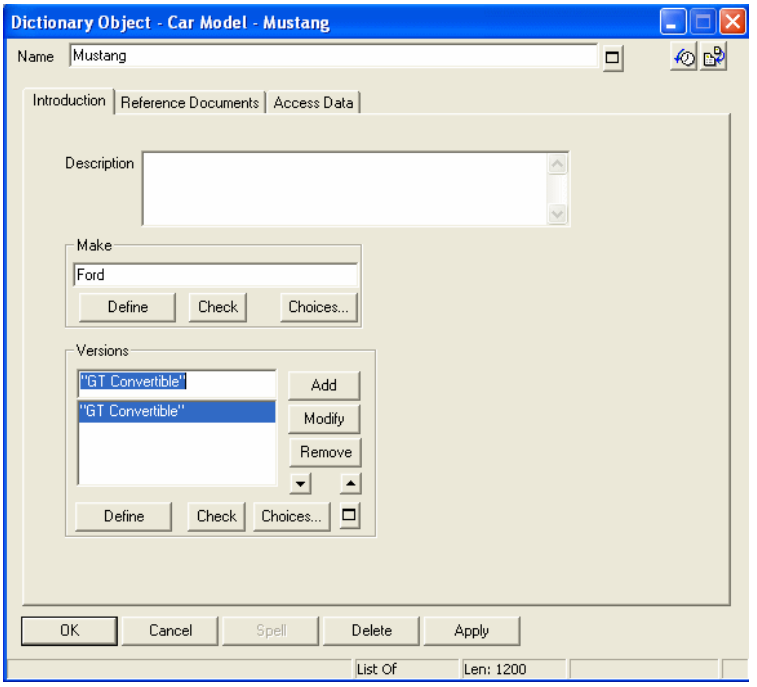
Bei der obigen USRPROPS.TXT-Datei gibt es ein **Problem**. "Car Version" ist eine Definition mit einem Verbundschlüssel: Sie hat neben ihrem eigenen Namen als Schlüssel noch zwei weitere KEY-Eigenschaften, nämlich "Car Brand" und "Car Model". Bei der Codierung der obigen USPROPS.TXT-Datei muss der Benutzer das von sich aus wissen. Der Benutzer muss die Fahrzeugversion eingeben und zwar vollständig qualifiziert durch deren Marke und Modell und einem Trennpunkt dazwischen, z. B. Ford.Mustang."GT Cabrio".

Hinzufügen der KEYED BY-Anweisung

Wir fügen die KEYED BY-Anweisung zur Anweisung der Eigenschaft "Version" hinzu, damit diese automatisch eingetragen wird.

```
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} }
}
```

Im ersten Teil der obigen KEYED BY-Anweisung **KEYED BY {"Brand":"Make"** geben wir an, dass der Wert von "Car Version", den wir eingeben, eine zu übernehmende KEY-Eigenschaft mit dem Namen "Brand" aufweist. Diese Eigenschaft wird mit einem Wert befüllt, den das System über die Eigenschaft "Make" der Definition, in der wir uns gerade befinden (nämlich "Car Model") bezieht. Beachten Sie, dass der eigentliche Wert eine Definition des Typs "Car Brand" ist. Durch die KEYED BY-Anweisung werden Namen von Eigenschaften aufgeführt und keine Definitionstypen.



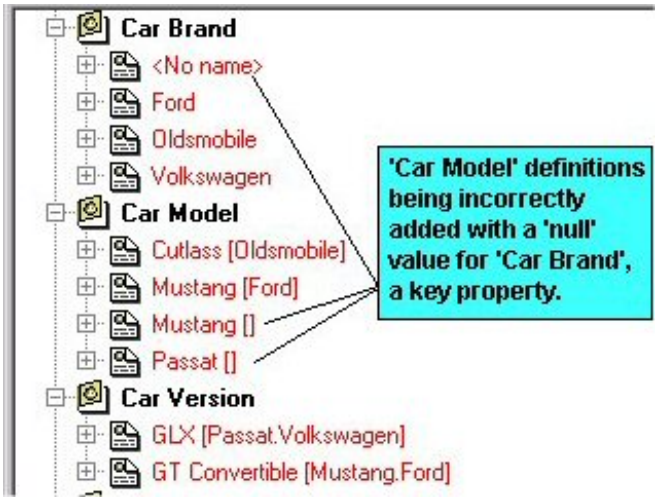
Wenn die Eigenschaft, auf die verwiesen wird ("Brand"), und die verweisende KEY-Eigenschaft ("Make") oben identisch wären, müssten wir nicht beides angeben. Wenn beide Eigenschaften den Namen "Brand" hätten, müssten wir lediglich Folgendes eingeben: KEYED BY **{"Brand",** ….

Im zweiten Teil der obigen KEYED BY-Anweisung, **"Car Model":Name**, geben wir an, dass der Wert "Car Version", den wir eingeben, eine zweite zu übernehmende KEY-Eigenschaft mit dem Namen "Car Model" aufweist. Diese Eigenschaft wird mit einem Wert befüllt, den das System über den Namen der aktuell geöffneten Definition bezieht (nämlich "Car Model").

Im dritten und letzten Teil der obigen KEYED BY-Anweisung (Name) geben wir an, dass der letzte Schlüssel des Werts "Car Version", den wir eingeben, durch seinen eigenen Namen bestimmt wird, wie dies bei allen Definitionen der Fall ist.

Wenn wir uns also in der Definition "Car Model" mit dem Namen "Mustang" befinden, die eine KEY-Eigenschaft "Make" aufweist, die mit "Ford" befüllt ist, müssen wir in die Eigenschaft LIST OF "Car Version" lediglich "GT" eingeben, damit die neue Definition "Ford.Mustang.GT" zur Enzyklopädie hinzugefügt wird.

Es stellt sich jedoch noch folgendes Problem dar: Wenn wir neue Definitionen für "Car Version" zur Enzyklopädie hinzufügen, stellen wir fest, dass Definitionen für "Car Model" eingetragen werden, die keine Eigenschaft "Car Brand" aufweisen.



Dies ist nur dann der Fall, wenn wir eine neue Definition "Car Version" direkt (über den Befehl "Neue Definition" in Explorer) eingeben, nicht aber, wenn wir diese im ListOf-Dialogfenster in der Definition "Car Model" hinzufügen.

Der Grund dafür ist, dass die Definition "Car Version" zwar angibt, dass eine ihrer KEY-Eigenschaften "Car Model" ist, nicht jedoch, dass diese Eigenschaft eine eigene KEY-Eigenschaft ("Car Brand") aufweist, die mit einem Wert zu befüllen ist. Jedes Mal, wenn wir eine neue Definition "Car Version" hinzufügen, werden wir aufgefordert, eine Eigenschaft "Car Model" anzugeben. Wir geben zwar die Eigenschaft "Car Model" an, nicht jedoch, woher diese den Wert für ihre KEY-

Eigenschaft "Car Brand" bezieht. Daher wird für diese Eigenschaft kein Wert eingetragen.

Zur Behebung dieses Problems müssen wir in der Definition "Car Version" angeben, dass die zugehörige KEY-Eigenschaft "Car Model" über eigene KEY-Eigenschaften verfügt, für die Werte zu übernehmen sind. Wir fügen die Klausel **KEYED BY {"Make":"Brand", Name**} hinzu. Damit geben wir an, dass die Definition "Car Model" eine Eigenschaft mit dem Namen "Make" aufweist, für die der Wert der Eigenschaft "Brand" aus der derzeit geöffneten Definition zu übernehmen ist.

```
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"}
}
```

Nachdem wir diese Änderung vorgenommen haben, werden beim Hinzufügen einer neuen Fahrzeugversion zur Enzyklopädie keine Definitionen für "Car Brand" ohne Inhhalt mehr angezeigt.

Beispiel für das Schlüsselwort COMPLETE

Es ist zu überlegen, ob der Benutzer die Möglichkeit haben soll, neue Definitionen für Fahrzeugversionen, unabhängig vom Fahrzeugmodell, einzugeben. Würde ein Benutzer "Si" als Fahrzeugversion eingeben und dann angeben, dass er sich auf einen "Honda Accord" bezieht? Wahrscheinlich nicht. Es kann für den Endbenutzer auch eine Erleichterung darstellen, wenn Sie ihn veranlassen, die Fahrzeugmarke und das Fahrzeugmodell einzugeben und anschließend Fahrzeugversionen in der ListOf-Eigenschaft der Fahrzeugversion in der Definition "Car Model" anzugeben. Damit geben Sie vor, dass eine Fahrzeugversion vollständig einem Fahrzeugmodell zuzuordnen ist. Die Bezeichnung "Si" allein sagt nicht sehr viel über das betreffende Fahrzeug aus. Dies erreichen wir, indem wir die COMPLETE-Klausel verwenden.

```
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} }
}
```

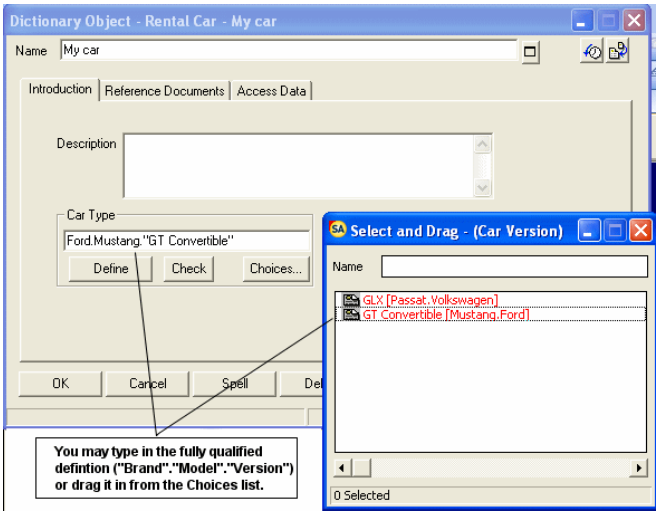
Nachdem Sie die COMPLETE-Klausel für eine Liste angegeben haben, wird die Definition "Car Version" weder bei der Erstellung einer neuen Definition bei den verfügbaren Definitionstypen aufgeführt, noch können Sie die Definition "Car Version" im Explorer öffnen. Wenn Sie versuchen, die Definition zu öffnen, gibt Rational System Architect eine Nachricht aus und weist darauf hin, dass die Definition nur über die übergeordnete Definition geöffnet werden kann, in diesem Fall über die Definition "Car Model".

Qualifizierbares Beispiel

Fügen wir nun eine neue Definition zur Enzyklopädie hinzu, um Fahrzeugvermietungen nachzuverfolgen. Der neue Definitionstyp "Car Rental" (Fahrzeugvermietung) weist eine Eigenschaft zur Verfolgung des jeweiligen Fahrzeugtyps bei der Fahrzeugvermietung auf.

Das Problem ist, dass wir bei einem Mietfahrzeug weder die Fahrzeugmarke noch das Fahrzeugmodell noch die Fahrzeugversion verfolgen möchten. Wir möchten lediglich die Eigenschaft "Car Type" (Fahrzeugtyp) erstellen, in die wir einen Fahrzeugtyp eingeben können und die dann die Werte für Fabrikat und Modell selbstständig bezieht. Wenn wir also einen Fahrzeugtyp für ein Mietfahrzeug angeben, verfügen wir innerhalb der Definition "Rental Car" (Mietfahrzeug) über keinerlei Eigenschaft, in der wir Marke und Modell des Fahrzeugs hinterlegen können. Wir verwenden daher das Schlüsselwort QUALIFIABLE.

```
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 }
}
}
```



Die Wortfolge QUALIFIABLE sorgt dafür, dass die Eigenschaft ONEOF "Car Type" die Informationen "Brand" und "Model" speichert. Die Informationen werden im Wert selbst gespeichert und durch Punkte getrennt. Sie können entweder die Schaltfläche "Optionen" betätigen und Werte aus dem anschließend angezeigten Dialogfenster "Auswählen und Ziehen" durch Ziehen übernehmen oder die Werte mit den entsprechenden Punktzeichen eingeben.

Beispiel für die Verwendung der WHERE-Klausel

Wir möchten angeben, dass eine Fahrzeugversion einer bestimmten Fahrzeugkategorie zugeordnet wird, z. B. der Kategorie SUV, Sub-Compact, Compact, Midsize Sedan oder Fullsize Sedan, Luxusfahrzeug oder Lkw. Da diese Liste relativ konstant bleibt, müssen wir keinen neuen Definitionstyp für sie erstellen. Wir erstellen eine Liste "Vehicle Types" für die Fahrzeugtypen.

```
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-Klausel Fortsetzung

Wir erstellen eine neue Definition des Typs "SUV Ad Campaign". In einer Eigenschaft mit dieser Definition sollen die Benutzer Fahrzeuge eines bestimmten Typs auswählen können. Diese Eigenschaft soll also so gefiltert werden, dass sie nur die Definitionen der Enzyklopädie enthält, die die Bedingung 'vehicle type' = 'SUV' erfüllen. Für diesen Filtervorgang verwenden wir die WHERE-Klausel.

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

Ausblenden von Standardeinträgen in der SAPROPS.CFG-Datei

Sie haben die Möglichkeit, Eigenschaften in der USRPROPS.TXT-Datei auszublenden. Es taucht häufig die Frage auf, was geschieht, wenn eine Eigenschaft ausgeblendet wird, für die bereits Informationen in die Enzyklopädie eingegeben worden sind. Angenommen, Sie haben bereits Datenelemente zur Enzyklopädie hinzugefügt und die zuständigen Geschäftsbereiche für diese Elemente in der Eigenschaft *Business Unit* angegeben. Mitten im Projekt wird nun beschlossen, dass die Eigenschaft *Business Unit* nicht mehr benötigt wird. In diesem Fall müssen Sie die Eigenschaft *Business Unit* lediglich als *unsichtbare* bzw. *ausgeblendete* Eigenschaft festlegen, indem Sie eine Änderung in der USRPROPS.TXT-Datei vornehmen. Anschließend wird die Eigenschaft im Definitionsdialogfenster des Datenelements nicht mehr angezeigt.

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

Als Sie die Werte für die Eigenschaft *Business Unit* eingegeben haben, wurden diese zur Enzyklopädie hinzugefügt und in der ENTITY.DBT-Datei gespeichert. Die Änderung am Metamodell der Enzyklopädie (siehe oben) bewirkt, dass die Eigenschaft "Business Unit" nicht mehr im Definitionsdialogfenster des Datenelements angezeigt wird. Durch die Änderung werden jedoch nicht die zuvor an der Eigenschaft vorgenommenen Dateneinträge gelöscht. Diese sind nach wie vor in der DBT-Datei vorhanden. Wenn Sie den obigen Code wieder aus der USRPROPS.TXT-Datei entfernen, wird die Eigenschaft *Business Unit* erneut im Definitionsdialogfenster des Datenelements angezeigt und auch die bereits eingegebenen Werte werden wieder im jeweiligen Datenelement angezeigt.

Wie können Sie nun tatsächlich nicht mehr benötigte Eigenschaftsinformationen aus der ENTITY.DBT-Datei löschen? Sie können die Informationen löschen, indem Sie, mit gebotener Vorsicht, die Optionen **Definitionen exportieren** und **Definitionen importieren** im Menü **Verzeichnis** verwenden.

1. Wählen Sie "Verzeichnis, Definitionen exportieren". Wählen Sie die gewünschten Datenelementdefinitionen aus, und exportieren Sie sie im CSV-Format in eine Textdatei.
2. Öffnen Sie die CSV-Textdatei in einem externen Editor (z. B. Excel), und löschen Sie die Spalte mit den nicht mehr benötigten Informationen (im obigen Beispiel wäre dies die Spalte "Business Unit").
3. Wählen Sie "Verzeichnis, Definitionen importieren". Importieren Sie die CSV-Datei mithilfe der Option *Alle Felder löschen und dann neue Daten hinzufügen* zurück.

Dieser Vorgang ist optional und muss nur durchgeführt werden, wenn Sie den Speicherplatz benötigen, der durch die überflüssigen Werte belegt wird.

Fehlernachrichten

Fehler-nachrichten

Wenn Rational System Architect eine Enzyklopädie öffnet und ein Parsing der SAPROPS.CFG- und der USRPROPS.TXT-Datei durchführt, werden die Anweisungen in den Dateien auf ihre Syntax überprüft. Etwaig festgestellte Syntaxfehler werden in einem **Fehler**-Dialogfenster angezeigt. Dieses **Fehler**-Dialogfenster kann die unten aufgeführten Fehlernachrichten aufweisen. Die Klammern weisen auf Stellen hin, in die Rational System Architect variable Informationen einfügt, wie z. B. einen Eigenschaftsnamen oder eine Zeilennummer.

< > found on Line < > of USRPROPS.TXT
< > has been defined more than once
< > is already defined as a List
Cannot load DLL (STATBAR.DLL).
Cannot load DLL (STATBOX.DLL).
Chapter < > is already defined.
Dictionary class < > is already defined.
Description
Dictionary
Illegal argument < >
Illegal argument < > - must be quoted
Illegal default < > for Boolean edit
Illegal default < > for date edit
Illegal default < > for numeric edit
Illegal default < > for time edit
Illegal default for < > ranged numeric edit
Insufficient resources to load dialog \n%s.
Invalid Dictionary class Name: < >.
Invalid Major Type Name: < >.
Invalid Relation Name: < >.
List < > is already defined.
List-name < > not defined
Name < > already in use
Number of property edits (OneOf, ListOf, ExpressionOf) exceeds limit with < > on
Number of properties exceeds limit with < > on < >
Number of DISPLAYed properties exceeds limit with < > on <

>
Number of lists exceeds limit with < > on < >
Number of lists exceeds limit with < > on < > (max=100)
Numeric argument < > out of range
Numeric argument expected but < > was
Out of range or invalid < > length argument
Premature end of file after < >
Previously defined list-name
Property < > is already defined
Referenced List < > is not defined.
Syntax Error in < > Line <line #>.
The < > edit type is only valid for the 'Description' property
Too many Lists.[2]
Too many Properties < >.[3]
Too many Values in List < >.[4]
Unable to open property file
Unbalanced begin-end or { }
Unexpected command < >
Unknown property DISPLAY type < >
Unknown dictionary name < >
Unknown edit-type < >
Unknown initial-type < >
Unknown update-type < >
Warning - RANGE found but no maximum range defined.
Warning - RANGE found but no minimum range defined.

---

[2] Die maximale Listenanzahl beträgt 400. Dies bezieht sich auf die SAPROPS.CFG- und die USRPROPS.TXT-Datei, wobei die Anzahl der tatsächlich von der SAPROPS.CFG-Datei genutzten Listen von der Enzyklopädiekonfiguration abhängig ist.

[3] Die maximale Anzahl von Eigenschaften für ein Diagramm, ein Symbol oder eine Definition beträgt 128.

[4] Die maximale Anzahl von Werten in einer Liste beträgt 128.

Zusätzlich zur Fehlernachricht zeigt Rational System Architect weitere Informationen zum festgestellten Syntaxfehler im Fehlerdialogfenster an:

while checking a **DEFINITION** command.
while checking a **DISPLAY** command.
while checking a **LIST** command.
while checking a **VALUE** command in a **LIST**.
while checking a **PROPERTY** command in a **DEFINITION**.
while checking for a **DEFINITION** or **LIST** command.
Would you like to continue?

Die vollständige Fehlernachricht kann beispielsweise wie folgt aussehen:

Unknown property DISPLAY type < > while checking a **DEFINITION** command.

Laufzeitbearbeitungen

Die Laufzeit ist die Zeit, in der Sie Diagramme zeichnen, und vor allem die Zeit, in der Sie Einträge in die Enzyklopädie vornehmen. Die Dialogfenster, die angezeigt werden, während Sie das Verzeichnis ergänzen oder ändern, werden von der SAPROPS.CFG- und der USRPROPS.TXT-Datei gesteuert. Die **EDIT**-Befehle verhindern die Eingabe unzulässiger Werte durch den Benutzer. Angenommen, die SAPROPS.CFG-Datei weist folgenden Eintrag auf:

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

In diesem Fall würde die Fehlernachricht **Invalid Value** (Ungültiger Wert) angezeigt werden, wenn Sie"AB" oder "0" in das Textfeld "My Property" eingeben und auf OK klicken, um das Dialogfenster zu schließen. Die Fehlernachricht wird erzeugt, weil die Eigenschaft als numerisch festgelegt wurde (keine Buchstaben enthalten darf) und mindestens den Wert 1 aufweisen muss.

Rational System Architect führt folgende Laufzeitbearbeitungen durch:

BOOLEAN	Muss auf T, F, TRUE oder FALSE lauten
DATE	Numerisch, im Format MM/TT/JJ
Expression, ExpressionOf, ListOf, OneOf	Siehe Einträge im entsprechenden Abschnitt ab Seite 2-75
NUMERIC	Muss numerische Zeichenfolge sein
TEXT	Keine Bearbeitung
TIME	Numerisch, im Format HH:MM:SS

3

USRPROPS.TXT-Schlüsselwörter

Einführung

Dieses Kapitel enthält eine alphabetische Liste aller Schlüsselwörter, die Sie verwenden können, um Änderungen an USRPROPS.TXT vorzunehmen.

Für die Verwendung der folgenden Schlüsselwörter gelten bestimmte Einschränkungen: CHAPTER, GROUP, LABEL, LIST und LISTONLY. Eine Erläuterung der jeweiligen Einschränkungen, die bei der Verwendung der Schlüsselwörter gelten, finden Sie bei den entsprechenden Schlüsselwörtern.

USRPROPS-Schlüsselwörter

$$FORCE$$ Siehe Schlüsselwort DISPLAY.

$$NONE$$ Siehe Schlüsselwort DISPLAY.

$$VFORCE$$ Siehe Schlüsselwort DISPLAY.

$$VNONE$$ Siehe Schlüsselwort DISPLAY.

#IFDEF Mit diesem Schlüsselwort können Sie Befehle in USRPROPS.TXT abhängig davon aktivieren, ob die in Anführungszeichen gesetzte Klausel nach dem IFDEF-Befehl im Dialogfenster "Eigenschaftskonfiguration" aktiviert wurde. Über das Dialogfenster "Eigenschaftskonfiguration" ("Tools", "Methodenunterstützung anpassen", "Enzyklopädiekonfiguration") wird die Datei "sadeclar.cfg" in einer Enzyklopädie geändert. Tatsächlich wird eigentlich die Datei "sadeclar.cfg" aktiviert, wenn die IFDEF-Anweisung beim Parsen von SAPROPS.CFG ausgewertet wird.

Dieser Befehl erfordert eine passende schließende #ENDIF-Anweisung.

Beispiel:

```
#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 (Fortsetzung)

Im obigen Beispiel weist der Definitionstyp "ORGUNIT" nur dann die angegebenen Eigenschaften auf, wenn im Dialogfenster **Eigenschaftskonfiguration** ("Tools", "Methodenunterstützung anpassen", "Enzyklopädiekonfiguration") der Eintrag "Unternehmen" ("Business Enterprise") aktiviert ist. Wenn Sie Werte für diese Eigenschaften eingeben und anschließend "Unternehmen" inaktivieren, verbleiben die Werte in der Definition von "ORGUNIT" im Repository, werden jedoch nicht im Definitionsdialogfenster angezeigt, da die Eigenschaftengruppe inaktiviert ist.

Siehe auch das Schlüsselwort #IFNDEF.

#IFNDEF

Im Gegensatz zum IFDEF-Befehl können Sie mit IFNDEF Befehle in USRPROPS.TXT aktivieren, wenn die nach dem Befehl in Anführungszeichen gesetzte Eigenschaft **nicht** im Dialogfenster "Eigenschaftskonfiguration" aktiviert wurde. Dieser Befehl erfordert eine passende schließende #ENDIF-Anweisung.

Beispiel:

#ifndef "Business Enterprise"

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

#endif

Wenn im obigen Beispiel die Auswahl "Unternehmensarchitektur" im Dialogfenster "Eigenschaftskonfiguration" von System Architect ("Tools", "Methodenunterstützung anpassen", "Enzyklopädiekonfiguration") nicht aktiviert ist, werden alle "Swim Lane"-Symbole umbenannt in "Org.Unit". Diese Änderung macht sich bemerkbar, wenn Sie ein solches Symbol in einem beliebigen Diagramm, in dem es verwendet wird, auswählen. Die Auswahl "Unternehmensarchitektur" im Konfigurationsdialog hieß vormals "Unternehmen", wurde jedoch in Version 9.0 im Dialogfenster geändert. Die zugrunde liegende Switchanweisung, die in SADECLAR.CFG aufgerufen wird, heißt jedoch weiterhin "Unternehmen".

#INCLUDE

#INCLUDE kann in der Datei USRPROPS.TXT verwendet werden, um Änderungen in separate, zusätzliche Dateien aufzugliedern. In jeder dieser Dateien können weitere Dateien eingeschlossen sein, die wiederum weitere Dateien einschließen usw. Die im Parser noch zulässige Verschachtelungsebene beträgt 10. Geht die Verschachtelung noch darüber hinaus, erhalten Sie von Rational System Architect eine Warnung. Die folgenden Ebenen werden ignoriert.

Beispiel:

Sie können beispielsweise drei USRPROPS.TXT-Dateien erstellen, eine für Diagramme (im Beispiel "diagrams.txt" genannt), eine für Definitionen (im Beispiel "definitions.txt" genannt) und eine für Symbole (im Beispiel "symbols.txt" genannt). In diesem Fall sieht die Datei USRPROPS.TXT folgendermaßen aus:

```
*************************************************************************
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"
*************************************************************************
```

In jede dieser Dateien können Sie #INCLUDE-Befehle für andere Dateien einschließen, z. B. eine Datei für Listen (beispielsweise "lists.txt").

Mit diesem Befehl wird eine kohärente Wiederverwendbarkeit benutzerdefinierter Daten ermöglicht.

ADDRESSABLE

In Rational System Architect können Symbole möglicherweise über eine oder mehrere (adressierbare) Definitionen *aufgerufen* werden. Dreizehn adressierbare Definitionen sind automatisch angegeben und können jedem beliebigen Symbol zugewiesen werden: *Unternehmensziel, Geschäftsprozess, Änderungsanforderung, Kritischer Erfolgsfaktor, aktuelle Archivgruppe mit Daten, Datenklasse, Liefergegenstand, Funktionale Organisation, Standort, Informationsanforderung, Unternehmensziele*, *Anforderungen und Testpläne.* Diese adressierbaren Definitionen stehen zur Verfügung, wenn Sie ein Symbol in einem Diagramm auswählen und den Eintrag "Verzeichnis" > "Adressen" auswählen. Darüber hinaus können alle Definitionen über die Syntax in USRPROPS.TXT als adressierbar deklariert werden. Dadurch wird die Definition zum Adressieren eines Symbols verfügbar gemacht und in der Dropdown-Liste "Adressen" unter "Verzeichnis" gespeichert.

Beispiel:

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

Siehe auch das Schlüsselwort NONADDR.

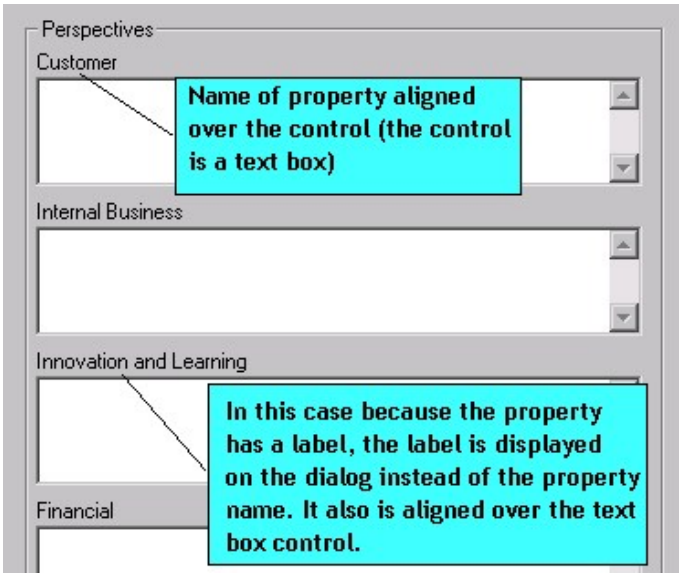
ALIGN

Mit diesem Schlüsselwort wird die Positionierung des Namens (oder der Beschriftung) eines Eigenschaftensteuerelements (Listenfeld, Textfeld etc.) in einem Dialogfenster angegeben. Gültige Optionen lauten: *BODY*, *LABEL* und *OVER*.

Beispiel:

```
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 }
}
```

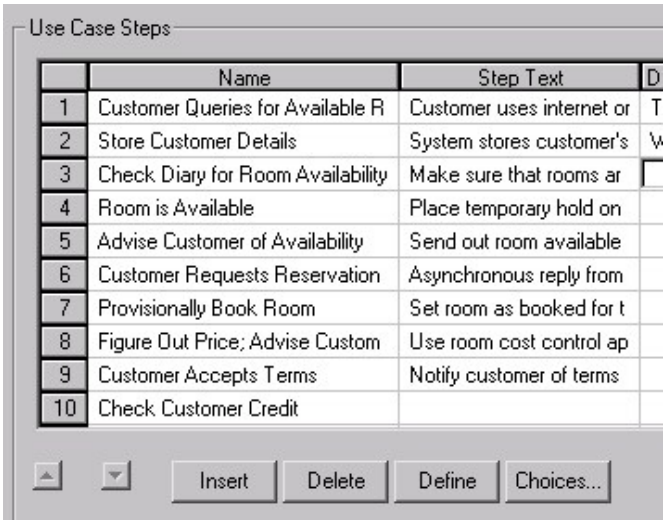
Im obigen Beispiel werden mit dem Befehl ALIGN OVER die Namen oder Beschriftungen aller Eigenschaften unterhalb der LAYOUT-Anweisung über das jeweilige Steuerelement im Definitionsdialogfenster "Balanced Scorecard" platziert.



Siehe auch die Schlüsselwörter BODY, LABEL, OVER, JUSTIFY und TAB

ASGRID

Der Befehl "ASGRID" gibt an, dass eine Eigenschaft vom Typ "ListOf" in einer Tabelle oder einem Raster dargestellt wird. ASGRID muss einen Bearbeitungstyp "ListOf" oder "ParmListOf" aufweisen. Anderenfalls gibt Rational System Architect eine Warnung aus, wenn Sie die Enzyklopädie erneut öffnen, und ignoriert das Schlüsselwort ASGRID.



Beispiel:

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 } }

Verwendung von ASGRID mit verschlüsselten Definitionen

Schlüsseleigenschaften einer Definition werden in einem Raster, das mit einem ASGRID-Befehl gebildet wurde, nicht angezeigt. Im obigen Beispiel wird der Paketname für jeden Schritt des Anwendungsfalls oder der Name des Anwendungsfalls nicht im Raster angezeigt.

ASGRID (Fortsetzung)

Einschränkung von ASGRID

Sie können "ASGRID" nicht in einer Anweisung vom Typ "LISTOF" verwenden, die auf eine Definition verweist, die Teil von "COMPLETE ListOf" einer anderen Definition ist. So können Sie beispielsweise ein Attribut "ListOf" zu einer Definition hinzufügen, dieses wird jedoch mit ASGRID nicht angezeigt. Die maximale Länge einer Eigenschaft, die im RASTER angezeigt werden kann, beträgt 400.

ASGRID COUNT_FIXED

Das Schlüsselwort COUNT_FIXED wird zusammen mit dem Schlüsselwort ASGRID verwendet, um festzulegen, dass der Benutzer keine Zeilen in einem Raster löschen oder einfügen kann.

Siehe auch die Schlüsselwörter KEY und KEYED BY und COUNT_FIXED.

ASGUID

Dieses Schlüsselwort kann nur mit Texteigenschaften verwendet werden. Damit wird automatisch eine Eigenschaft mit dem Wert der "GUID"-Eigenschaft belegt. Diese Texteigenschaft kann anschließend als Schlüsseleigenschaft anstelle der tatsächlichen GUID-Eigenschaft verwendet werden. Die Eigenschaft ASGUID muss schreibgeschützt sein. Wenn Sie die Definition erneut öffnen, ist die ASGUID-Eigenschaft bereits eingetragen.

Beispiel:

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

ASPARMGRID

Das Schlüsselwort ASPARMGRID wurde speziell für die Verwendung bei der Datenmodellierung in Rational System Architect erzeugt und funktioniert mit speziell erstelltem Code. Dieses Schlüsselwort befindet sich in der Datei SAPROPS.CFG und **darf nicht** von Benutzern in USRPROPS.TXT verwendet werden.

ASSIGN

Sie können neue oder bereits (in einem anderen Diagramm) vorhandene Symboltypen zu neuen oder bereits vorhandenen Diagrammtypen zuweisen. Symboltypen können mit der folgenden Syntax zu Diagrammtypen hinzugefügt werden:

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

Beispiel:

```
SYMBOL "Organizational Unit" IN "Organization Chart"
{
ASSIGN TO "Enterprise Direction"
}
```

AUDITID

Dieses Schlüsselwort stellt die Zeichen dar, die im Dialogfenster **Prüf-ID** eingegeben wurden, als sich der Benutzer zum ersten Mal bei Rational System Architect angemeldet hat. AUDITID ist ein zulässiger Schlüsselworttyp, mit dem angegeben wird, dass eine Eigenschaft die Prüf-ID des Benutzers aufweist. CHECKOUT AUDITID, FREEZE AUDITID, INITIAL AUDITID und UPDATE AUDITID haben jeweils besondere Bedeutungen. Weitere Informationen finden Sie bei den jeweiligen Schlüsselwörtern, die separat in dieser Tabelle aufgeführt sind.

Anmerkung: Ab Version 9.1 von Rational System Architect sind INITIAL AUDITID (in einem Feld mit dem Namen "Anfangsprüfung" bereitgestellt) und UPDATE AUDITID (in einem Feld mit dem Namen "Letzte Änderungsprüfung) bereitgestellt) automatisch auf der Registerkarte "Zugriffsdaten" jeder Definition enthalten.

Beispiel:

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

Andere Verwendungszwecke für die Prüf-ID liegen möglicherweise in beliebigen Definitionen vor.

Beispiel:

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

AUTOCREATE

Mit dem Befehl AUTOCREATE wird automatisch eine Definition hinter dem Wert erstellt, der in einer Eigenschaft angegeben wurde, sobald Sie auf die Schaltfläche "OK" klicken, um das darin enthaltene Dialogfenster zu schließen. Wenn Sie das Schlüsselwort AUTOCREATE nicht verwenden, bleibt der in eine Eigenschaft eingegebene Wert nach dem Klicken auf die Schaltfläche "OK" zum Schließen des Dialogfensters undefiniert.

Beispiel:

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

Im obigen Beispiel weist das Definitionsdialogfenster einer Definition für eine physische Datenbank eine Eigenschaft mit der Bezeichnung "Projektdatenmodell" auf. Wenn Sie einen Wert in dieses Feld eingeben (z. B. "Reservierungen") und anschließend auf "OK" klicken, um das Definitionsdialogfenster "Physische Datenbank" zu schließen, wird die Projektdaten-modelldefinition "Reservierungen" automatisch in der Enzyklopädie erstellt.

Siehe auch INITIAL USER REQUIRED und OVERRIDABLE.

BEGIN

Dieses Schlüsselwort weist auf den Anfang der Definition einer Eigenschaft oder der Eigenschaftengruppe hin, die die Definition eines Diagramms, Symbols oder einer Definition bilden. Sie können auch die folgende Syntax verwenden: **{**.

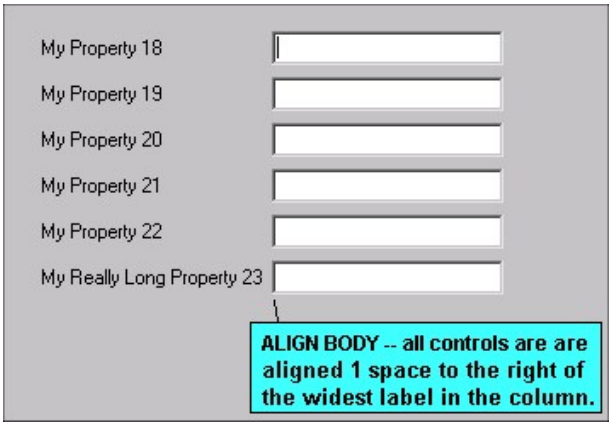
Siehe auch das Schlüsselwort PROPERTY.

BODY

Hierbei handelt es sich um eines der im Befehl **ALIGN** verwendeten Argumente. Damit werden alle Steuerelemente an einer Stelle rechts von der breitesten Beschriftung in dieser Spalte ausgerichtet. (Im Gegensatz dazu wird mit dem Schlüsselwortpaar ALIGN OVER der Name oberhalb der Eigenschaft platziert.)

Beispiel:

```
Definition "My Definition"
{
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}
}
```



Im obigen Beispiel stellt das Steuerelement "My Really Long Property 23" ein Textfeld dar, das eine Stelle rechts neben der Beschriftung platziert wird. Alle anderen Textfeldsteuerelemente für andere Eigenschaften im Dialogfenster werden an diesem Steuerelement ausgerichtet.

Anmerkung: Vormals wurde ALIGN BODY verwendet, um alle Steuerelemente eine Stelle rechts neben der Beschriftung zu positionieren, dies wurde in der Folge jedoch geändert, sodass das Schlüsselwort jetzt mit ALIGN LABEL identisch ist.

Siehe auch die Schlüsselwörter OVER, ALIGN, TAB, LABEL und JUSTIFY.

BOOLEAN

Wird in einem **Definition**sdialogfenster als Kontrollkästchen angezeigt. Diese Eigenschaft verfügt über einen der beiden folgenden Werte: True (T) oder False (F).

Beispiel:

Im folgenden Beispiel kann der Benutzer die Funktionen zur hierarchischen Nummerierung in einem IDEF0-Diagramm aktivieren bzw. inaktivieren, indem er "True" oder "False" auswählt.

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

BROWSER (Explorer)

Mit diesem Schlüsselwort wird festgelegt, ob eine Eigenschaft und ihr Wert im Feld "Eigenschaften" des Explorers (Browsers) von Rational System Architect angezeigt werden, wenn das jeweilige Diagramm, Symbol oder die entsprechende Definition im Explorer ausgewählt wird.

Folgende Explorer-Steueranweisungen sind zulässig (das Wort "Objekt" wird in der Bedeutung Diagramm, Symbol oder Definition verwendet):

Im Rahmen der Spezifikation einer Eigenschaft:

- **BROWSER {SHOW}:** Erzwingt, dass der Explorer den Wert dieser Eigenschaft anzeigt, wenn das Objekt, das diese Eigenschaft aufweist, angezeigt wird.

Im Rahmen der Spezifikation eines Objekts, aber nicht in der Beschreibung einer Eigenschaft:

- **BROWSER {OMITKEY}:** Erzwingt, dass der Explorer die Schlüsseleigenschaften des Objekts unter Bedingungen, unter denen diese eigentlich angezeigt werden, nicht anzeigt.
- **BROWSER {OMITTYPE}:** Erzwingt, dass der Explorer den Typ des Objekts unter Bedingungen, unter denen dieser eigentlich angezeigt wird, nicht anzeigt.

Nicht im Rahmen der Spezifikation eines Objekts:

- **BROWSER {OMITKEY}:** Erzwingt, dass der Explorer die Schlüsseleigenschaften eines beliebigen Objekts unter Bedingungen, unter denen diese eigentlich angezeigt werden, nicht anzeigt.
- **BROWSER {OMITTYPE}:** Erzwingt, dass der Explorer den Typ eines beliebigen Objekts unter Bedingungen, unter denen dieser eigentlich angezeigt wird, nicht anzeigt.

Der Begriff "unter Bedingungen, unter denen dieser eigentlich angezeigt wird" wird oben verwendet, da der Explorer häufig bestimmte (oder alle) Schlüsseleigenschaften nicht anzeigt (wenn das Objekt als untergeordnetes Objekt zu einem der dazugehörigen Schlüsselobjekte angezeigt wird) und häufig den Typ ebenfalls nicht anzeigt (wenn dieser untergeordnet zu einem Typenheader angezeigt wird).

BROWSER (Fortsetzung für Explorer)

Beispiel 1:

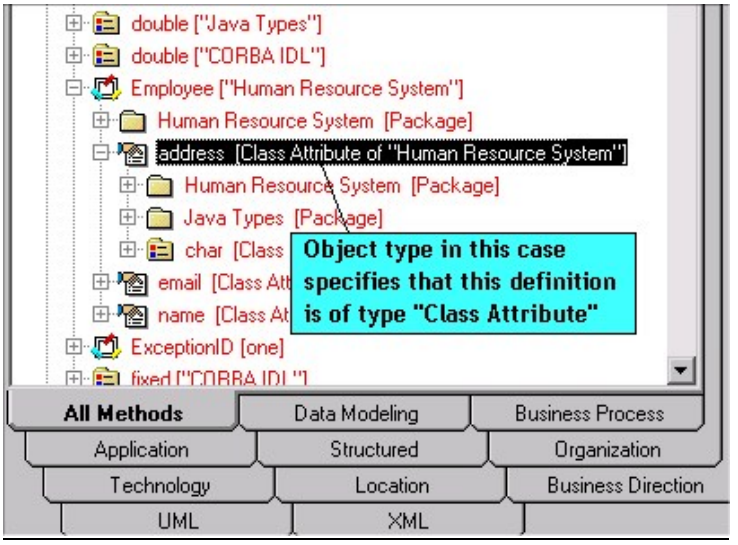
```
DEFINITION "Association  End"
{
PROPERTY "Package" { KEY EDIT OneOf "Package" RELATE BY "is keyed by" READONLY  BROWSER { SHOW }}
..}
```

In diesem Beispiel wird der Wert der Eigenschaft "Package" im Explorer angezeigt, obwohl dieser Wert normalerweise nicht angezeigt würde.

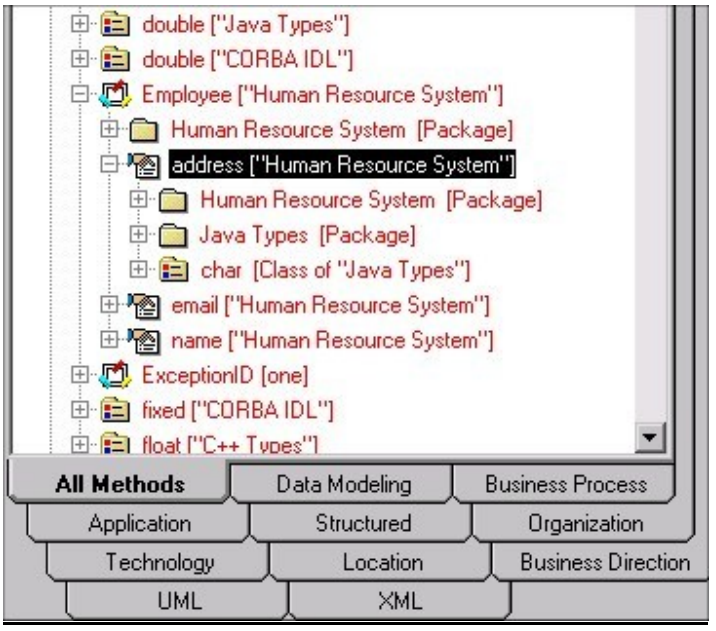
Beispiel 2:

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

Im obigen Beispiel ist ein Klassenattribut eine Definition vom Typ "Klassenattribut". Dieses wird standardmäßig im Explorer angezeigt, was eigentlich redundant ist und den Benutzer möglicherweise stört. Wird der Befehl BROWSER (OMITTYPE) nicht verwendet, zeigt der Explorer die Attribute an, die im nachstehenden Diagramm aufgeführt sind.



Wenn Sie den Befehl "BROWSER {OMITTYPE}" verwenden, zeigt der Explorer Attribute so an, wie im folgenden Diagramm dargestellt.



Beispiel 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 } }
..
}
```

In diesem Beispiel wird der Wert der Eigenschaft "Class Roles" im Explorer angezeigt (da es für den Benutzer wichtig ist, zu wissen, an welche Klassen eine Assoziation angehängt wird), obwohl dieser Wert normalerweise nicht angezeigt würde.

BY

Ein häufig verwendetes Schlüsselwort, wie an den folgenden Ausdrücken zu erkennen ist: *DEFINED BY, RELATED BY, RELATE BY* und *KEYED BY.* Weitere Informationen finden Sie unter der speziellen Schlüsselwortkombination.

Beispiel:

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

CHAPTER

Erzeugt ***Registerkarten*** in einem Dialogfenster. Jede CHAPTER-Anweisung entspricht einer Registerkarte. Verwenden Sie folgende Syntax:

CHAPTER <Kapitelname>

Die Kapitelanweisung erfordert keine öffnenden oder schließenden Klammern, um die Elemente auf der Registerkarte zu gruppieren. Alle Elemente, die unter eine Kapitelanweisung fallen, werden auf dieser Registerkarte gruppiert. Die nächste Gruppierung wird mit der nächsten Kapitelanweisung erstellt.

Beispiel:

CHAPTER "Screen Painter properties"

Ändern des Namens einer Registerkarte (Kapitel):

Mit dem LABEL-Befehl können Sie den Namen eines KAPITELS (CHAPTER) über USRPROPS.TXT ändern.

Beispiel:

Die Datei SAPROPS stellt eine Registerkarte "Verschachtelte Klassen" für eine Klassendefinition zur Verfügung:

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

Sie können das KAPITEL "Verschachtelte Klassen" über den LABEL-Befehl in USRPROPS.TXT in "Fred" umbenennen:

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

CHECKOUT

Dieser Befehl zeigt Informationen zum Auschecken eines Objekts an, wie z. B. die Prüf-ID (AUDITID) des Benutzers, der das Objekt ausgecheckt hat, oder das Datum (DATE) und die Uhrzeit (TIME), zu dem/der das Objekt ausgecheckt wurde. Die angezeigten Felder sind immer SCHREIBGESCHÜTZT. Die Werte werden automatisch von Rational System Architect protokolliert. Um die Werte jedoch in einem Dialogfenster anzuzeigen, müssen Sie Eigenschaften mit den folgenden Merkmalen hinzufügen:

CHECKOUT Auditid
CHECKOUT Date
CHECKOUT Time

Beispiel:

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

Weitere Informationen zum Ein- und Auschecken von Objekten finden Sie in der Onlinehilfe unter *Zugriffssteuerung*.

Siehe auch das Schlüsselwort FREEZE.

COLS, COLUMNS

Legt die Anzahl der Spalten fest, in denen eine Gruppe von Eigenschaften in einem Dialogfenster "Diagramm", "Symbol" oder "Definition"platziert wird.

Beispiel:

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

COLUMN_SCRIPT

Mit COLUMN_SCRIPT wird ein Script aufgerufen, das in SA Basic geschrieben wurde. Die COLUMN-Scripts werden für das Verhalten von Spalten in Tabellen in einem physischen Datenmodell verwendet. Die durch das Script ausgelöste Aktion wird auf jede Spalte in der Liste angewendet.

Gemäß der Konvention erhält die Funktion selbst eines der folgenden Präfixe:

- fmtxxx – Die Funktion selbst ist fest codiert und kann nicht geändert werden. Die meisten Funktionen in SAPROPS.CFG sind fest codiert. Dadurch wird die Antwort von Rational System Architect insgesamt schneller.
- _fmtxxx – Ist in der Datei "fmtscript.bas" im ausführbaren Hauptverzeichnis in Rational System Architect vorhanden.

Erstellen eines eigenen Scripts

Weitere Informationen zum Erstellen eigener Scripts finden Sie unter dem Schlüsselwort SCRIPT.

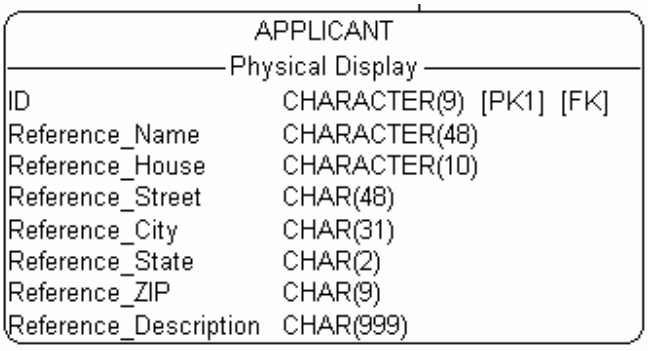
Beispiel:

```
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 gibt Werte für Attribute in Entitäten von Diagrammen mit Entitätenbeziehungen oder in Tabellenspalten in physischen Diagrammen zurück.

FmtERAttr gibt ID, NAME, ADDRESS, STREET, CITY, STATE, FIRST_5_DIGITS, ZIP CODE und LAST_4_DIGITS zurück.

COLUMN_SCRIPT (Fortsetzung)



Siehe auch die Schlüsselwörter SCRIPT, COMPONENT_SCRIPT, VALUESCRIPT und FORMAT.

COMPLETE

Wenn Sie dieses Schlüsselwort verwenden, gehört die Definition, auf die verwiesen wird, zur verweisenden Definition, sodass nicht mit einer anderen Definition auf die Definition verwiesen werden kann, auf die die verweisende Definition verweist. Diese Definition kann nur über die verweisende Definition bearbeitet werden.

Ein Beispiel ist ein Attribut in einer Entität, die vollständig zu einer Entität (und nicht zu einer anderen Definition) gehört. Das Attribut kann nur über die Entitätsdefinition geöffnet werden (Sie können eine Attributdefinition beispielsweise nicht direkt im Explorer von Rational System Architect öffnen).

Beispiel:

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

COMPONENT_ SCRIPT

Hiermit wird eine in Basic geschriebene Funktion mithilfe von Funktionsaufrufen für Rational System Architect aufgerufen, die Bestandteil von SA Basic sind. Die Komponentenscripts werden für "ListOf"- und "ExpressionOf"-Listen verwendet. Die durch das Script ausgelöst Aktion wird auf jedes Element in der Liste angewendet. Mit der Syntax **COMPONENT_SCRIPT** ***fmtomtattr*** werden beispielsweise alle Attribute und ihre entsprechenden C-Speichertypen, getrennt durch einen Doppelpunkt (:), zurückgegeben.

Gemäß der Konvention erhält die Funktion selbst eines der folgenden Präfixe:

- fmtxxx – Die Funktion selbst ist fest codiert und kann nicht geändert werden. Die meisten Funktionen in SAPROPS.CFG sind fest codiert. Dadurch wird die Antwort von Rational System Architect insgesamt schneller.
- _fmtxxx – Ist in der Datei "fmtscript.bas" im ausführbaren Hauptverzeichnis in Rational System Architect vorhanden.

Erstellen eines eigenen Scripts

Weitere Informationen zum Erstellen eigener Scripts finden Sie unter dem Schlüsselwort SCRIPT.

Erläuterung bereits vorhandener Scripts:

fmtUMLAttr gibt alle Attribute und die entsprechenden Typen getrennt durch einen Doppelpunkt zurück.

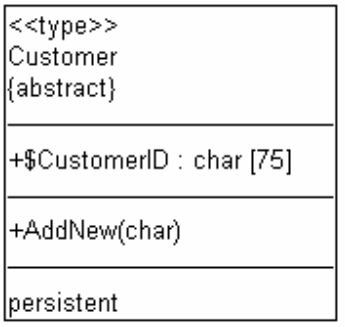
fmtOMTOperation gibt alle Vorgänge und die entsprechenden C-Speichertypen in runden Klammern (Typ) zurück.

FmtOMTObjInstAttr gibt alle Attribute für die Klasse zurück, die ein Objekt instanziiert.

FmtOMTActivity gibt das Script **do**: und den Namen der Aktivität für alle in einer Statusdefinition aufgeführten Aktivitäten zurück.

FmtOMTStateActions gibt den Namen der internen Aktion für alle in einer Statusdefinition aufgeführten internen Aktionen zurück.

COMPONENT_ SCRIPT (Fortsetzung)



Beispiel (mit *fmtomattr*):

```
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

Das Schlüsselwort CONTROL ist äquivalent zum Schlüsselwort PROPERTY, wenn es mit TESTPROC verwendet wird, um einen Switch innerhalb einer Definition zu konfigurieren.

Es gibt zwei Möglichkeiten anzugeben, dass eine Eigenschaft in einem Definitionsdialogfenster abhängig vom Wert eines Switches angezeigt wird. Sie können #IFDEF-Schlüsselwörter verwenden, die für Werte gelten, die Sie für die Enzyklopädie im Dialogfenster "Enzyklopädiekonfiguration" festlegen (z. B. indem Sie den Sprachtyp der Enzyklopädie auf Java oder C++ setzen). Im Dialogfenster "Enzyklopädiekonfiguration" werden tatsächlich Werte in der Datei "sadeclar.cfg" festgelegt.

Sie können außerdem festlegen, dass eine EIGENSCHAFT basierend auf einem Switch in einem Dialogfenster angezeigt wird (und welchen Anfangswert diese aufweist), der selbst wiederum eine Eigenschaft (TESTPROPERTY) im Definitionsdialogfenster ist. Sie können beispielsweise innerhalb einer Entität angeben, dass der entsprechende Datenbankmanagementsystemtyp "Oracle" oder "SQL Server" lautet. Nachfolgende Eigenschaften werden in der Definition entweder angezeigt oder nicht angezeigt und haben bestimmte Standardwerte,

die auf dem Wert basieren, den Sie für den Datenbankmanagementsystemtyp festgelegt haben. Mit dem Schlüsselwort TESTPROC können Sie den TESTPROPERTY-Switch festlegen. Sie verwenden das Schlüsselwort PROPERTY, wenn Sie zum ersten Mal eine bestimmte Eigenschaft in der Definition und das CONTROL-Schlüsselwort für jedes weitere Auftreten dieser Eigenschaft in der Definition festlegen. Mit dem Schlüsselwort REFPROP können Sie angeben, welches PROPERTY-Schlüsselwort auf die einzelnen CONTROL-Schlüsselwörter verweist. Daher werden die Schlüsselwörter CONTROL und REFPROP häufig in Verbindung mit TESTPROC verwendet.

Zusammengefasst muss, damit ein CONTROL-Schlüsselwort verwendet wird, ein Ausgangsverweis auf die Eigenschaft (PROPERTY) vorhanden sein, auf die CONTROL verweist, und zwar an erster Stelle der Definition. Das Schlüsselwort REFPROP wird in Verbindung mit dem Schlüsselwort CONTROL verwendet.

Beispiel:

```
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"}
..
  }
```

Im obigen Beispiel wird das Schlüsselwort REFPROP in Verbindung mit dem Schlüsselwort CONTROL verwendet, um anzugeben, dass die Eigenschaften "Primärschlüssel", "Eindeutig" und "In Clustern" für die Indexdefinition bereitgestellt werden, wenn Oracle 8 als Datenbankmanagementsystem ausgewählt wurde. Diese Eigenschaften entsprechen exakt der Eigenschaft, auf die verwiesen wird.

COPY PROPERTIES FROM

Mit diesem Befehl können Sie Eigenschaften aus anderen Definitionstypen in den aktuellen Definitionstyp kopieren. Damit können Sie ähnliche Konzepte in einem einzelnen Definitionstyp konsolidieren. Dies gilt nur für Definitionen. Die Syntax lautet folgendermaßen:

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

Beispiel:

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"
…
}

Der Kopiervorgang erfolgt an der Stelle in der Eingabe, an der die Kopieranweisung gefunden wird. Wenn im obigen Beispiel weiter unten in den Eigenschaftendateien Eigenschaften zu Änderungsanforderungen, Abhängigkeiten oder Knoten hinzugefügt werden, werden die hinzugefügten Daten und Änderungen **nicht** kopiert.

COPYSCRIPT

Mit diesem Schlüsselwort wird ein SA Basic-Script angegeben, das für eine bestimmte Eigenschaft aufgerufen werden soll, wenn eine Kopie der Definition erstellt wird.

Erstellen eines eigenen Scripts

Weitere Informationen zum Erstellen eigener Scripts finden Sie unter dem Schlüsselwort SCRIPT.

Beispiel:

```
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

Das Schlüsselwort COUNT_FIXED wird zusammen mit dem Schlüsselwort ASGRID verwendet, um festzulegen, dass der Benutzer keine Zeilen in einem Raster löschen oder einfügen kann. Die Anzahl der Zeilen ist festgelegt.

Beispiel:

```
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 } }
```

Im obigen Beispiel weist eine Assoziation zwischen Klassen eine Zeile im Raster für jede Klasse auf, der die Assoziation zugeordnet ist (in der Regel zwei, möglicherweise auch drei oder mehr, wenn zusätzliche Klassen an die Assoziationslinie angehängt sind - dieses Verhalten ist in der Software fest codiert). Aufgrund des Schlüsselworts COUNT_FIXED können Benutzer keine Zeilen im Raster hinzufügen oder löschen.

Im Gegensatz dazu gibt es andere Raster, beispielsweise das Raster "Anwendungsfallschritt", in dem Benutzer neue Schritte hinzufügen können oder Schritte aus dem Raster löschen können.

DATA

Hierbei handelt es sich ***nicht*** um ein Schlüsselwort. Dies ist ein spezielles Wort, das als Argument der Befehle ONEOF, LISTOF und EXPRESSIONOF verwendet wird und auf Datenelemente und Datenstrukturen verweist, die das Datenverzeichnis von Rational System Architect bilden.

DATE

Bei diesem Schlüsselwort handelt es sich um einen Bearbeitungstyp, dessen Länge 10 betragen muss. Die grafische Darstellung basiert auf dem in Windows festgelegten Datumsformat. DATE ist außerdem ein zulässiger Feldtyp, mit dem angegeben wird, dass eine Eigenschaft eine Datums-zeitmarke in der Notation aufweist, die dem in Windows definierten Zeitformat entspricht.

CHECKOUT DATE, FREEZE DATE, INITIAL DATE und UPDATE DATE haben jeweils besondere Bedeutungen.

Beispiel 1:

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

Andere Verwendungszwecke für DATE liegen möglicherweise in beliebigen Definitionen vor.

Beispiel 2:

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

DEASSIGN

Mit dem Schlüsselwort DEASSIGN können Sie Symbole aus einem Diagrammtyp entfernen.

Beispiel:

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

DEFAULT

Der Wert, den Rational System Architect einer Eigenschaft zuordnet und der vom Benutzer überschrieben werden kann. In der Grafikanzeige wird der Standardwert zunächst in einem Textfeld angezeigt; oder der Standardwert legt fest, ob ein Kontrollkästchen aktiviert oder inaktiviert ist.

Beispiel:

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

DEFINED BY

Mit diesem Schlüsselwort wird eine Definition einem Symbol zugeordnet. Damit können Sie zudem ein Symbol erneut einer anderen Definition zuordnen.

Bedeutung 1: Wenn Sie neue Symbole zu einer Enzyklopädie in USRPROPS.TXT hinzufügen, müssen Sie mithilfe dieses Schlüsselworts angeben, welchem Definitionstyp sie zugeordnet sind. Wenn diese Klausel bei einem neuen, in USRPROPS.TXT angegebenen Symbol fehlt, gibt Rational System Architect beim Öffnen der Enzyklopädie eine Parsing-Warnung aus und übernimmt standardmäßig die Nulldefinition für das Symbol.

Beispiel 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"
}
```

Im obigen Beispiel wurden in USRPROPS.TXT ein neuer Diagrammtyp, ein neuer Symboltyp und ein neuer Definitionstyp festgelegt. Das Schlüsselwort DEFINED BY wird verwendet, um anzugeben, dass das Symbol "Richtung" (Direction) durch die Definition "Richtung" definiert wird. Außerdem wird das Symbol dem Diagramm "My Diagram" zugeordnet. (Anmerkung: Sie können auch eine Definitionsanweisung für die neue Definition "Richtung" angeben, dies ist jedoch nicht obligatorisch. Wenn Sie nichts anderes angeben, weist die neue Definition einfach die Standardeigenschaften "Name" und "Beschreibung" auf.)

Bedeutung 2: Mit dem Schlüsselwort DEFINED BY können Sie auch ein Symbol durch eine andere Definition definieren als in SAPROPS.CFG angegeben. Wenn Sie dieses Schlüsselwort zum erneuten Zuordnen eines Symbols zu einer anderen Definition verwenden, müssen Sie angeben, in welchem Diagramm das Symbol, auf das Sie verweisen, vorhanden ist (z. B. Symbol "Klasse" in "Klasse" im Gegensatz zum

DEFINED BY (Fortsetzung)

Symbol "Klasse" in "Komponente" - im ersten Fall geben wir an, dass wir die Klassensymboldefinition in einem Klassendiagramm definieren; im zweiten Fall wird das Klassensymbol in einem Komponentendiagramm definiert.

Beispiel 2:

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

Im obigen Beispiel wird das Prozesssymbol in einem Diagramm "Datenfluss (Gane & Sarson)" jetzt durch "Steuerelement-umwandlung" definiert. In der Regel wird es durch "Prozess" definiert.

DEFINITION

Dieses Schlüsselwort ist das erste Wort in einem Block, in dem die Eigenschaften einer DEFINITION im Gegensatz zu einem DIAGRAMM oder einem SYMBOL aufgeführt sind.

Beispiel:

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

Siehe auch die Schlüsselwörter DIAGRAM und SYMBOL.

DEFINITION REFERENCED IN

Siehe "OF DEFINITION REFERENCED IN".

DEPICT LIKE

Die Schlüsselwortkombination DEPICT LIKE wird verwendet, um anzugeben, wie ein Symbol in einem Diagramm dargestellt wird. Sie können diese Schlüsselwortkombination beim Erstellen eines neuen Symbols verwenden, um anzugeben, wie dieses in einem Diagramm dargestellt werden soll. Sie können beispielsweise angeben, dass es wie ein Symbol in einem anderen Diagramm angezeigt werden soll.

Sie können die Schlüsselwortkombination DEPICT LIKE mit Knoten- und Liniensymbolen verwenden.

Beispiel (Knotensymbol):

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

Beispiel (Liniensymbol):

```
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

Bestimmt, wie ein Symbol mit einer von Ihnen bereitgestellten Bilddatei dargestellt werden kann. Sie können ein Symbol mit einem Bitmap oder einer Metadatei darstellen. Mit dem Schlüsselwort DEPICTIONS in Kombination mit dem Schlüsselwort DIAGRAM können Sie angeben, wie dieses Symbol im Diagrammarbeitsbereich dargestellt werden soll. Wenn Sie das Schlüsselwort DEPICTIONS in Kombination mit dem Schlüsselwort MENU verwenden, können Sie festlegen, wie das Symbol in der Symbolleiste und im Menü Zeichnen dargestellt wird. Verwenden Sie folgende Syntax:

```
SYMBOL <symbol-type-name>
        { ...
        DEPICTIONS { DIAGRAM <depiction-file> }
        DEPICTIONS { MENU <depiction-file> }
        ...}
```

wobei "<depiction-file>" der Name und der vollständige Pfad eines Bitmap oder einer Metadatei ist.

Beispiel:

```
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" }}
```

Sie können das Schlüsselwort DEPICTIONS auch in einer Liste verwenden, damit das Symbol je nach dem in der Liste ausgewählten Wert jeweils unterschiedlich dargestellt wird.

Beispiel:

```
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

Der Befehl DIAGRAM wird auf zwei verschiedene Arten angewendet.

Angeben von Diagrammeigenschaften:

Der DIAGRAM-Befehl wird als erstes Wort in einem Block verwendet, in dem die Eigenschaften eines Diagramms im Gegensatz zu einer DEFINITION oder einem SYMBOL aufgeführt sind.

Beispiel:

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

Siehe auch die Schlüsselwörter DEFINITION und SYMBOL.

Bei Verwendung mit dem Befehl DEPICTIONS:

Verweist auf die Grafik, die zum Darstellen eines Symbols in einem Diagrammarbeitsbereich im Vergleich zur Symbolleiste oder zum Menü "Zeichnen" verwendet wird.

Beispiel:

```
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

Mit diesem Schlüsselwort werden eine Eigenschaft und ihr Wert in einem Diagrammsymbol darstellbar. Es liegt eine Begrenzung von 37 Anzeigeanweisungen für eine Definition vor.

Die Syntax lautet folgendermaßen:

DISPLAY { FORMAT [STRING | LIST | KEY | NONKEY | COMPONENT_SCRIPT | COLUMN_SCRIPT | SCRIPT] **LEGEND "** (Kennzeichnung des Blocks im Symbol) **" }**

Sie haben die Option, eines der folgenden FORMAT-Schlüsselwörter anzugeben:

STRING: Hiermit werden die Werte der Eigenschaft genau wie eingegeben im Symbol dargestellt. Ein Beispiel finden Sie beim Schlüsselwort STRING.

LIST: Hiermit werden die Elemente für das Symbol in einer Liste angezeigt – jedes Leerzeichen führt zu einer neuen Zeile, es sei denn, das Leerzeichen wird in Anführungszeichen gesetzt. Weitere Informationen finden Sie beim Schlüsselwort LIST.

KEY: Verwenden Sie dieses Schlüsselwort für Eigenschaften, die als Schlüssel designiert sind. Sie werden in einem separaten Abschnitt des Symbols angezeigt. Ein Beispiel und weitere Informationen finden Sie beim Schlüsselwort KEY.

NONKEY: Sie können dieses Schlüsselwort für Eigenschaften ohne Schlüsselfunktion verwenden. Sie werden in einem separaten Abschnitt des Symbols angezeigt. Dieses Schlüsselwort wurde ursprünglich für Entitäten und Tabellen in der Datenmodellierungsunterstützung von Rational System Architect verwendet. Ein Beispiel finden Sie beim Schlüsselwort NONKEY.

COLUMN_SCRIPT: Siehe das Schlüsselwort COLUMN_SCRIPT.

COMPONENT_SCRIPT: Siehe das Schlüsselwort COMPONENT_SCRIPT.

SCRIPT:. Siehe das Schlüsselwort SCRIPT.

DISPLAY (Fortsetzung)

Innerhalb der Anführungszeichen nach dem Schlüsselwort LEGEND geben Sie an, wie der Block im Symbol gekennzeichnet wird. Folgende Optionen stehen zur Verfügung:

LEGEND "<Ihr Text>": Beliebiger Text, den Sie in Anführungszeichen setzen, wird oberhalb der Eingabe im Symbol angezeigt, jedoch nur, wenn ein Wert für die Eingabe vorliegt.

LEGEND "": Zeigt eine gerade Linie ohne Worte an, jedoch nur, wenn ein Wert für die Eingabe vorliegt.

LEGEND "$$FORCE$$": Zeigt eine horizontale Linie oberhalb der Eingabe im Symbol an. Diese Linie dient als Trennlinie. Das Schlüsselwort "$$FORCE$$" unterscheidet sich von der einfachen Verwendung von " " dadurch, dass die Anzeige einer horizontalen Linie zwingend ist, selbst wenn die Eigenschaftenanzeige über das Dialogfenster für den Anzeigemodus unterdrückt wird.

LEGEND "$$NONE$$": Zeigt keine horizontale Linie oberhalb der Eingabe im Symbol an, unabhängig davon, ob Werte für die Eingabe vorliegen. Diese Linie dient normalerweise als Trennlinie.

LEGEND "$$VFORCE$$": Ermöglicht das Verschieben von Eigenschaften innerhalb von Symbolen von links nach rechts und zieht vertikale Linien dazwischen. Siehe das Schlüsselwort VFORCE.

LEGEND "$$VNONE$$": Ermöglicht das Verschieben von Eigenschaften von links nach rechts, stellt jedoch *keine* Trennlinie bereit. Siehe das Schlüsselwort VNONE.

Beispiel:

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

EDIT

Dieses Schlüsselwort gibt in Verbindung mit dem Schlüsselwort BEGIN (oder {) den Anfang der Definition einer Eigenschaft an. Das Schlüsselwort EDIT hat die Bedeutung "Dies ist das Anfangsargument".

Beispiel:

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

EDIT COMPLETE

Siehe das Schlüsselwort COMPLETE.

EDITCLASS

Verwenden Sie dieses Schlüsselwort nicht. Dieses Schlüsselwort ist ein spezielles Schlüsselwort, das insbesondere für bestimmte Situationen in Rational System Architect entwickelt wurde, nämlich die Übernahme von Datenelementeigenschaften durch ein Attribut in einer Entität. Dieser Befehl wird in der Datei SAPROPS.CFG für diese Situation verwendet. Dieses Schlüsselwort kann nur in dieser Situation angewendet werden. Eine Verwendung in anderen Situationen kann Fehler verursachen.

EDIT URLS

Sie können angeben, dass eine "ListOf"-Eigenschaft als Eigenschaft designiert ist, die auf externe Dokumente verweisen kann. Der Befehl führt dazu, dass Schaltflächen im unteren Bereich der "ListOf"-
Eigenschaft angezeigt werden, und zwar die Schaltflächen "Öffnen", "Extern durchsuchen" und "Intern durchsuchen". Mit diesen Schaltflächen können Sie externe Dokumente durchsuchen und auswählen oder externe
Hyperlinks eingeben oder die interne Dateitabelle der Enzyklopädiedatenbank durchsuchen oder ein Dokument mit externen oder internen Verweisen öffnen.

Syntax:

PROPERTY property name { **EDIT URLS** }

Einschränkungen:

SA erzwingt folgende Einschränkungen für die URLS-Eigenschaft. -- Muss kein Schlüssel sein.

Beispiel:

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

END

Dieses Schlüsselwort weist auf das Ende der Spezifikation einer Eigenschaft oder einer Eigenschaftengruppe hin, die die Definition eines Diagramms, eines Symbols oder einer Definition bilden. Es wird mit der BEGIN-Anweisung kombiniert, um die Spezifikation einzuschließen. Anstelle der BEGIN- und END-Anweisungen können Sie auch linke und rechte geschweifte Klammern **{ }** verwenden.

Beispiel:

```
PROPERTY "<property_name>"
   BEGIN EDIT <edit_type> <property_parameter>
   END
```

EXPRESSION

Mit diesem Schlüsselwort wird angegeben, dass der Wert der Definition als eine Reihe von Zeichenfolgen eingegeben werden muss, die durch ein Pluszeichen oder ein Leerzeichen getrennt werden.

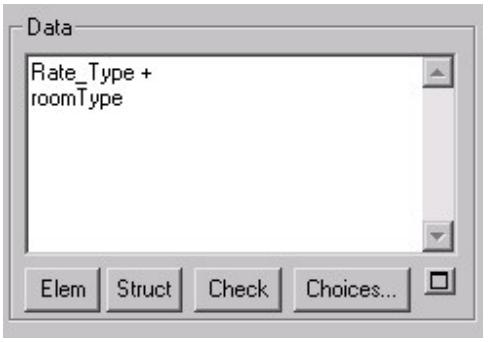
Beispiel:

```
VendorName +
VendorCity +
VendorState
```

Siehe auch das Schlüsselwort EXPRESSIONOF, das dieses Schlüsselwort ersetzt hat.

EXPRESSIONOF

Mit EXPRESSIONOF können Sie mithilfe komplexer Operatoren und Begrenzer Verweise auf Objekte ausdrücken. EXPRESSIONOF wird normalerweise mit dem speziellen Argument DATA verwendet, das auf Datenelemente und Datenstrukturen verweist, d. h. EXPRESSIONOF DATA. Mit dieser Schlüsselwortkombination wird ein Textfeld bereitgestellt, in dem Definitionswerte als Reihe von Zeichenfolgen eingegeben werden. Die Abtrennung zwischen den einzelnen Definitionswerten erfolgt durch Leerzeichen. Gemäß der Konvention wird ein Pluszeichen (+) verwendet, um die einzelnen Definitionswerte zu trennen, dies ist jedoch nicht erforderlich.



Beispiel:

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

Weitere Informationen und eine Liste der verwendbaren Operatoren und Begrenzer finden Sie bei den Schlüsselwörtern ONEOF und LISTOF sowie in Kapitel 2 im Abschnitt zum Thema "ExpressionOf".

fmtxxx oder _fmtxxx

Diese beiden Namenspräfixe werden in der Regel am Anfang des Namens einer beliebigen Funktion verwendet, die mit den Schlüsselwörtern SCRIPT, COLUMN_SCRIPT oder COMPONENT_SCRIPT aufgerufen wird. Die Funktion selbst (z. B. _fmtUMLAttr) stellt üblicherweise eine spezielle Formatierungsdarstellung eines Eigenschaftswertes (beispielsweise ein Attribut und alle dazugehörigen Eigenschaften) im Symbol bereit. Die Namenskonvention lautet folgendermaßen:

- _fmt (z. B. _fmtUMLAttr): Die Funktion selbst ist nur fest codiert verfügbar und kann nicht geändert werden. Die meisten Funktionen in SAPROPS.CFG sind fest codiert. Dadurch wird die Antwort von Rational System Architect insgesamt schneller.
- fmt (z. B. fmtUMLAttr): Ist in der Datei "fmtscript.bas" im ausführbaren Hauptverzeichnis von Rational System Architect vorhanden.

Beispiel:

```
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" }
```

Im obigen Beispiel wird ein Script aufgerufen, mit dem Attribute in einem Klassendiagramm auf bestimmte Art und Weise angezeigt werden (wenn z. B. die Zugriffseigenschaft eines Attributs "Öffentlich" lautet, wird im Klassensymbol vor dem Attribut ein "+" eingegeben usw.)

Erstellen eigener Funktionen

Weitere Informationen zum Erstellen eigener Funktionen finden Sie beim Schlüsselwort SCRIPT.

Siehe auch die Schlüsselwörter DISPLAY, FORMAT, SCRIPT, COLUMN_SCRIPT, COMPONENT_SCRIPT und VALUESCRIPT.

FORCE

Hierbei handelt es sich eigentlich um das Schlüsselwort $$FORCE$$, das zusammen mit dem Schlüsselwort DISPLAY verwendet wird.

Weitere Informationen finden Sie beim Schlüsselwort DISPLAY.

FORMAT

Mit diesem Schlüsselwort wird angegeben, wie Daten für eine bestimmte anzeigbare Eigenschaft dargestellt werden sollen.

Beispiel:

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

Weitere Informationen finden Sie beim Schlüsselwort DISPLAY.

FREEZE

Zeigt Informationen zum Blockieren eines Objekts an, z. B. die AUDIT ID (Prüf-ID) derjenigen Person, die das Objekt blockiert hat, sowie DATE (Datum) oder TIME (Zeit) der Blockierung. Die angezeigten Felder sind immer SCHREIBGESCHÜTZT. Die Werte werden automatisch von Rational System Architect protokolliert. Um die Werte jedoch in einem Dialogfenster anzuzeigen, müssen Sie Eigenschaften mit den folgenden Merkmalen hinzufügen:

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

Beispiel:

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

Weitere Informationen zum Blockieren von Objekten finden Sie in der Onlinehilfe unter *Zugriffssteuerung*.

Siehe auch das Schlüsselwort CHECKOUT.

FROM_CHOICES_ONLY

Sorgt dafür, dass ein Benutzer nur eine Definition aus der Liste der Optionen auswählt, ohne eine neue Definition eingeben zu können. Es wird ein Nachrichtenfeld angezeigt, in dem der Benutzer zur Auswahl aus der Liste "Optionen" aufgefordert wird. Dieses Schlüsselwort wird mit "ListOf" und "OneOf" verwendet.

Beispiel:

```
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

Mit diesem Schlüsselwort wird ein Gruppenfeld mit bestimmten Layoutparametern erstellt, beispielsweise mit einer Reihe von Optionsfeldern, in denen sich mehrere Eigenschaften befinden.

Beispiel 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"
```

Sie können keinen Gruppennamen ändern, der bereits durch Rational System Architect in der SAPROPS-Datei vordefiniert wurde.

Beispiel 2:

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

Wenn Sie in der dritten Zeile des obigen Beispiels in der Datei USRPROPS.TXT versuchen, 'GROUP "Source Data"' in 'GROUP "Original Data"' zu ändern, haben die Änderungen keine Auswirkungen. Der im SAPROPS GROUP-Eintrag enthaltene Text "Source Data" wird nicht überschrieben. Dieser Text ist weiterhin der Anzeigetext für die Klassenattributgruppe.

HELP

Diese Zeichenfolge wird in der Statuszeile in der ***linken unteren Ecke*** eines Dialogfensters **Diagramm** oder **Definition** angezeigt, wenn eine vorgegebene Eigenschaft ausgewählt wurde.

Syntax:

HELP "<text_string>"

Beispiel:

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

HETEROGENEOUS (ONEOF, LISTOF)

Mit diesem Schlüsselwort kann eine einzelne Eigenschaft auf Definitionen mehr als eines Typs verweisen. (Eine normale Liste verweist auf Definitionen eines einzelnen Typs.) Das Schlüsselwort HETEROGENEOUS wird verwendet, um eines der Schlüsselwörter ONEOF oder LISTOF zu ändern.

Wenn Sie beispielsweise auf die Schaltfläche "Optionen" einer Klassenliste klicken, werden nur die Klassendefinitionen für die Auswahl bereitgestellt. Wenn Sie auf die Schaltfläche "Optionen" einer heterogenen Liste klicken, erhalten Sie verschiedene Typen von Definitionen, die Sie in der Listenklausel "Heterogen" angegeben haben, z. B. "Klasse", "Prozess", "Entität" usw.

Syntax für ONEOF:

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

Syntax für LISTOF:

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

Einschränkungen

Für eine heterogene Listeneigenschaft gibt es bestimmte Einschränkungen. Gleichzeitig kann die Eigenschaft keine der folgenden Merkmale erfüllen (d. h., Sie können keines der folgenden Schlüsselwörter zusammen mit HETEROGENEOUSLISTOF oder HETEROGENEOUSONEOF in derselben Eigenschaft verwenden):

- Die Eigenschaft darf nicht KEY sein.
- Sie darf keine KEYED BY-Klausel enthalten.
- Sie darf nicht beendet sein (COMPLETE).
- Sie darf keine FROM-Klausel aufweisen.
- Sie darf nicht ASGRID sein.
- Sie darf keinen Standardwert aufweisen (DEFAULT).
- Es darf kein Erstbenutzer erforderlich sein (INITIAL USER REQUIRED).
- Sie darf keine Einschränkungsklausel (REFERENCED IN oder WHERE) enthalten.
- Sie darf nicht die Zuordnung INIT_FROM_SYMBOL aufweisen.
- Kein Typname darf mehrmals aufgeführt sein.

HETEROGENEOUS (ONEOF, LISTOF) (Fortsetzung)

Hinzufügen neuer Werte zur Liste

In der Regel ziehen Benutzer zwar durch Klicken auf die Schaltfläche "Optionen" Werte aus dem Browser "Auswählen und ziehen" in eine heterogene Liste, sie können jedoch auch neue Werte zur heterogenen Liste hinzufügen. Um jedoch neue Werte in eine heterogene Liste aufzunehmen, müssen die Benutzer die neuen Werte mit dem vollständig qualifizierten Namen im folgenden Format eingeben:

ClassName:TypeName:FullyQualifiedName

Wobei:

- "ClassName" der Klassentyp der System Architect-Enzyklopädie ist, also Diagramm, Symbol oder Definition.
- "TypeName" der spezifische Name des Diagramm-, Symbol- oder Definitionstyps ist, z. B. "Klasse" (Definition) oder "Anwendungsfallschritt" (Definition).
- Jeder Teil des vollständig qualifizierten Namens ("FullyQualifiedName") durch Punkte getrennt wird, sodass beispielsweise ein Anwendungsfallschritt, auf den mit dem zugehörigen Anwendungsfall verwiesen wird, auf den wiederum mit dem entsprechenden Paket verwiesen wird, folgendermaßen eingegeben werden muss:

  Definition:"Anwendungsfallschritt":"Paketname"."Anwendungsfallname"."Anwendungsfallschrittname"

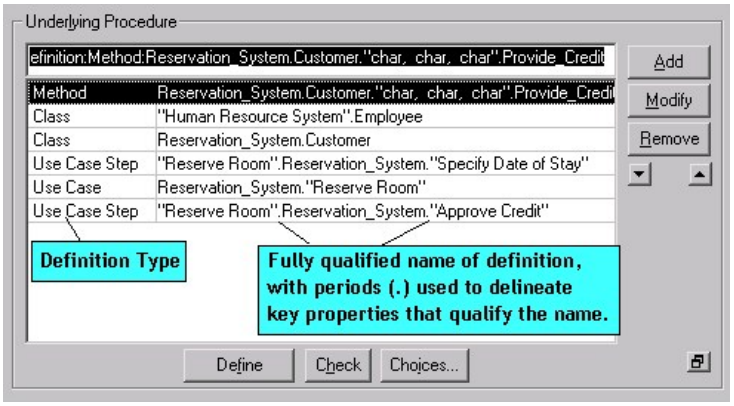
Beispiel:

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

Im obigen Beispiel kann die Eigenschaft für die zugrunde liegende Prozedur der Definition "Prozedur" mit Definitionen des Typs "Anwendungsfall", "Klasse", "Methode" und "Anwendungsfallschritt" belegt werden.

Die durch das Schlüsselwort HETEROGENEOUSONEOF oder HETEROGENEOUSLISTOF bereitgestellte Benutzerschnittstelle zeigt eine Spalte an, die den Namen jedes Definitionstyps sowie den vollständig qualifizierten Namen der jeweiligen, in die Liste gezogenen Definition enthält.

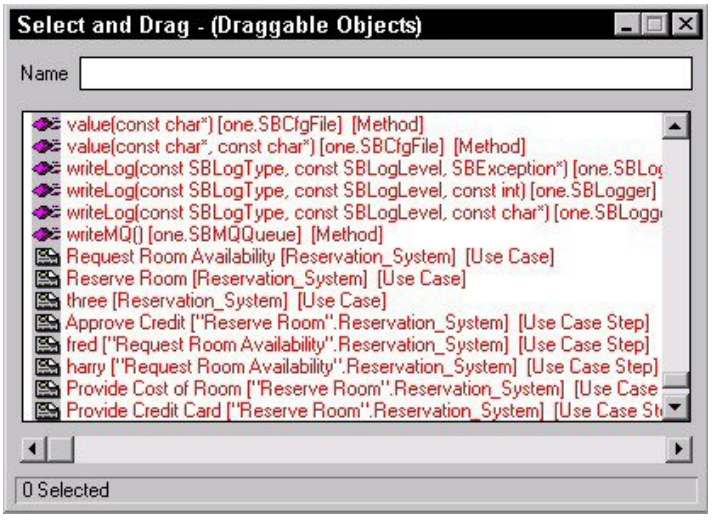
HETEROGENEOUS (ONEOF, LISTOF) (Fortsetzung)



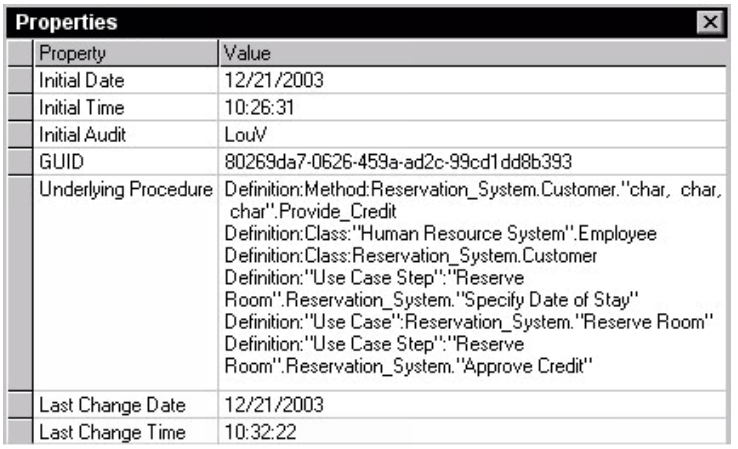
Die Schlüsseleigenschaften, die den Namen einer Definition qualifizieren, werden in der Benutzerschnittstelle bereitgestellt und sind durch Punkte (.) voneinander getrennt. Beispielsweise verweist ein Anwendungsfallschritt auf den übergeordneten Anwendungsfall, der wiederum auf das übergeordnete Paket verweist. Im Feld HETEROGENEOUSONEOF oder HETEROGENEOSLISTOF erfolgt die Darstellung des Anwendungsfallschritts durch "Paketname"."Anwendungsfallname"."Anwendungsfallschrittname". Wenn im Namen eines Elements Leerzeichen integriert sind, wird dieses Element in Anführungszeichen gesetzt. In der obigen Abbildung befindet sich beispielsweise der Anwendungsfallschritt "Kredit genehmigen" im Anwendungsfall "Reservierungssystem", das zum Paket "Raum reservieren" gehört.

HETEROGENEOUS (ONEOF, LISTOF) (Fortsetzung)

Wenn Sie für eine heterogene Liste auf die Schaltfläche "Optionen" klicken, werden alle aufgerufenen Diagramm-, Symbol- oder Definitionstypen dargestellt und der zugehörige Typ in eckigen Klammern hinter den Namen gesetzt.



Das Fenster "Eigenschaften" stellt ebenfalls Werte einer heterogenen Liste dar. Sie können an den Rändern der Eigenschaftenzeilen oder -spalten ziehen, um die Werte vollständig anzuzeigen. Jedem Wert sind der entsprechende Klassentyp (Diagramm, Symbol oder Definition), Typname (d. h. die Definition "Anwendungsfallschritt") und der Wert selbst vorangestellt.



HIDE DEFINITION

Mit diesem Schlüsselwort wird der Definitionstyp, auf den verwiesen wird, aus den Dialogfenstern **Neue Definition** und **Definition öffnen** entfernt.

Syntax:

HIDE DEFINITION <Definitionsname>

Beispiel:

HIDE DEFINITION "SQL Server Table"

ACHTUNG: Gehen Sie beim Ausblenden von Definitionen sehr vorsichtig vor, insbesondere wenn diese von Symbolen verwendet werden, die Sie durch Ihre Auswahl im Dialogfenster "Eigenschafts-konfiguration" ("Tools", "Methodenunterstützung anpassen") aktiviert haben. Möglicherweise tritt eine Situation ein, in der Sie Symbole ohne zugrunde liegende Definitionen zeichnen.

HIDE DIAGRAM

Mit diesem Schlüsselwort wird der Diagrammtyp, auf den verwiesen wird, aus den Dialogfenstern **Neues Diagramm** und **Diagramm öffnen** entfernt.

Syntax:

HIDE DIAGRAM <Diagrammname>

Beispiel:

HIDE DIAGRAM "Booch Process"

Anmerkung: Anstatt dieses Schlüsselwort zu verwenden, besteht eine weniger drastische Änderung darin, einfach die Auswahl des Diagrammtyps im Dialogfenster **Eigenschafts-konfiguration** aufzuheben (wählen Sie hierzu "Tools", "Methodenunterstützung anpassen", "Enzyklopädiekonfigu-ration" aus, und inaktivieren Sie entweder die Methode für den Diagrammtyp, oder klicken Sie im Dialogfenster "Eigenschafts-konfiguration" auf die Schaltfläche "Erweitert" und verschieben den Diagrammtyp aus der Liste "Ausgewählte Diagramme" in die Liste "Verfügbare Diagramme").

HIERARCHICAL

Standardmäßig sind Benutzerdiagramme Netzwerke (aus Symbolen). Wenn jedoch dieses Schlüsselwort in die Beschreibung eines Diagrammtyps eingeschlossen wird, wird der Diagrammtyp als hierarchisches Diagramm behandelt. Das bedeutet, dass alle diesem Diagramm zugewiesenen Knotensymbole in einer Hierarchie angeordnet werden können und weitere zusammengehörige hierarchische Funktionalitäten (z. B. die hierarchische Nummerierung) unterstützt werden.

Das Schlüsselwort HIERARCHICAL kann nur mit benutzerdefinierten Diagrammtypen verwendet werden - es kann nicht auf bestehende Diagrammtypen angewendet werden. In jedem anderen Kontext wird es nach einer Warnung an den Benutzer ignoriert. (Weitere Informationen zum Erstellen eines neuen benutzerdefinierten Diagrammtyps finden Sie beim Schlüsselwort RENAME DIAGRAM.)

Beispiel:

```
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" }
}
```

Das von USRPROPS.TXT im obigen Beispiel erstellte Diagramm ist hierarchisch - und mit einem Organigramm etc. vergleichbar.

IFDEF

Siehe #IFDEF-Befehl.

IFNDEF Siehe Befehl #IFNDEF.

IN Mit diesem Schlüsselwort wird der Kontext für den Befehl **RENAME** erstellt, wenn dieser auf ein Symbol angewendet wird. Es kann auch für **DEPICT LIKE** verwendet werden.

Beispiele:

Umbenennen eines Anwendungssymbols:

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

Ein Symbol soll aussehen wie ein Symbol in einem anderen Diagramm:

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

INCLUDE Siehe #INCLUDE.

INITIAL

Wird verwendet, um ein Diagramm, ein Symbol oder eine Definition mit der Angabe AUDIT ID (Prüf-ID), DATE (Datum) und TIME (Uhrzeit) für die Erstellung zu versehen. Der Wert dieses Feldes wird von Rational System Architect nie geändert.

Varianten:

INITIAL DATE

INITIAL TIME

INTIAL AUDITID

Ab Version 9 von Rational System Architect werden INITIAL DATE, INITIAL TIME und INITIAL AUDITID standardmäßig auf der Registerkarte "Zugriffsdaten" jedes Diagramms oder Definitionsdialogfensters bereitgestellt. Diese Einstellung ist fest im Produkt codiert - d. h. das Schlüsselwort INITIAL ist nicht in jeder Definition in SAPROPS.CFG vorhanden, und Sie müssen es auch nicht für neue Diagramm- oder Definitionstypen, die Sie erstellen, zu USRPROPS.TXT hinzufügen.

Beispiel:

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

Siehe auch das Schlüsselwort UPDATE.

INIT_FROM_SYMBOL

Das Schlüsselwort INIT_FROM_SYMBOL wird innerhalb einer Definition verwendet, die ein Symbol definiert. Es gibt an, dass eine Eigenschaft in der Definition zunächst seinen Wert aus einer ähnlich benannten Eigenschaft im Symbol übernimmt. Dieses Schlüsselwort wird in Fällen verwendet, in denen eine Eigenschaft sowohl im Symbol als auch in der Definition vorhanden sein und denselben Wert aufweisen muss. Dies ist erforderlich, wenn vom Benutzer bereitgestellte Metadateien für ein Symbol anhand einer Eigenschaft, beispielsweise "Stereotyp", angegeben werden. Das Stereotyp muss für das Symbol (denn dadurch wird bestimmt, wie das Symbol im Diagramm dargestellt wird) und in der entsprechenden Definition angegeben werden.

Beispiel 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 } ..}
```

Im obigen Beispiel wird die Eigenschaft "Stereotype" (Stereotyp) sowohl in der Spezifikation des Klassensymbols als auch in der Klassendefinition angegeben. Sie muss denselben Wert besitzen. Die Eigenschaft "Stereotyp" im SYMBOL führt dazu, dass mögliche Stereotypwerte für die Auswahl in einer Dropdown-Liste im Menü "Zeichnen" von Rational System Architect angezeigt werden (dabei handelt es sich um Bitmaps, die durch die DEPICTIONS-Klausel in der LIST-Anweisung festgelegt werden). Wenn Sie eine mit Stereotyp versehene Klasse aus der Liste in der Symbolleiste "Zeichnen" ausgewählt und im Diagramm abgelegt haben, wird die Definition der Klasse erstellt und die entsprechende Stereotypeigenschaft automatisch von dem für das Symbol ausgewählten Stereotyp belegt. Anmerkung: Wenn Sie diesen Wert in der Definition ändern, wird er auch im Symbol geändert.

INIT_FROM_SYMBOL (Fortsetzung)

Beachten Sie außerdem, dass dieser Stereotypwert nicht auf der Registerkarte "Symbol" der Klasse angezeigt wird, da er ausgeblendet wurde (INVISIBLE).

Beispiel 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 }
}
```

Im obigen Beispiel ist die Eigenschaft "Programmiersprache" ("Programming Language") im Diagramm vorhanden, und das Symbol "Klasse" übernimmt den Wert dieser Eigenschaft aus dem Diagramm. Die Definition des Klassensymbols übernimmt ebenfalls den Wert dieser Eigenschaft über das Symbol, und zwar aufgrund des Schlüsselworts INIT_FROM_SYMBOL.

Wenn eine Klassendefinition über den Explorer erstellt wird, MUSS aufgrund des Schlüsselworts INITIAL USER REQUIRED die erforderliche Eigenschaft zum Zeitpunkt der Erstellung in der Klassendefinition angegeben werden.

INITIAL USER REQUIRED

Mit diesem Schlüsselwort wird angegeben, dass zum Zeitpunkt der Erstellung des Modellierungselements (entweder Diagramm, Symbol oder Definition) ein Wert für die Eigenschaft angegeben werden **muss**. Wenn Sie keinen Wert angeben und versuchen, das Dialogfenster durch Klicken auf "OK" zu schließen, erhalten Sie von Rational System Architect eine Nachricht "Die Eigenschaft xxx muss angegeben werden.". Sie können nicht auf "OK" klicken, um das Dialogfenster zu schließen und das Diagramm, das Symbol oder die Definition zu erstellen. Sie müssen entweder einen Wert für die Eigenschaft angeben oder das Dialogfenster abbrechen.

Beispiel:

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** } ..}

Im obigen Beispiel müssen Sie für die Eigenschaften "Paket" ("Package") und "Aktivitätsmodell" ("Activity Model") Werte angeben, bevor Sie beim Erstellen eines Aktivitätsdiagramms im Diagrammdialogfenster auf die Schaltfläche "OK" klicken können.

Beachten Sie, dass im obigen Beispiel die Eigenschaft "Paket" überschrieben werden kann (OVERRIDABLE), während dies für die Eigenschaft "Aktivitätsmodell" nicht gilt. Das Schlüsselwort OVERRIDABLE wirkt sich nur bei Symbolen in diesem Diagramm aus, die Werte der Eigenschaft aus dem Diagramm übernehmen.

Beispiel 2:

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

INITIAL USER REQUIRED (Fortsetzung)

Im obigen Beispiel wird mit dem Schlüsselwort READONLY, das in Verbindung mit dem Schlüsselwort INITIAL USER REQUIRED verwendet wird, angegeben, dass das Dialogfenster erst geschlossen werden kann, wenn vom Benutzer ein Wert für diese Eigenschaft eingegeben wurde. Außerdem wird festgelegt, dass die Eigenschaft nach der Eingabe des Anfangswerts schreibgeschützt wird und vom Benutzer nicht mehr geändert werden kann. OVERRIDABLE wirkt sich nur auf die Definitionen aus, die diesen Eigenschaftswert aus dem Diagramm übernehmen.

Das Schlüsselwort INITIAL USER REQUIRED erfordert die Angabe eines Werts für die Eigenschaft "XML Schema" (XML-Schema) bei der Erstellung des Diagramms. Mit dem Schlüsselwort AUTOCREATE wird automatisch eine Definition für jeden in diese Eigenschaft eingegebenen Wert erstellt. Wenn daher der Benutzer auf "OK" klickt, um das Dialogfenster "Diagramm" zu schließen, wird eine definierte Definition "XML Schema" erstellt.

Siehe auch die Schlüsselwörter OVERRIDABLE, READONLY und AUTOCREATE.

INVISIBLE

Blendet eine Eigenschaft im Grafikdialogfenster aus, ohne sie zu löschen. Ausgeblendete Eigenschaften werden in Situationen verwendet, in denen eine Eigenschaft für eine Definition erforderlich, jedoch für den Benutzer ohne Bedeutung ist.

Beispiel:

```
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 }
}
```

Im obigen Beispiel wird die Eigenschaft "Stereotyp" zusammen mit dem Symbol und der Definition einer Klasse verwendet. Sie muss übereinstimmen. Benutzer können die Eigenschaft "Stereotyp" in der Klasse auswählen. Dieser Wert wird dann automatisch für das Symbol übernommen, in dem die Eigenschaft verwendet wird, um anzugeben, wie das Symbol angezeigt wird. Der Benutzer muss die Eigenschaft "Stereotyp" jedoch nicht auf der Registerkarte "Symbol" der Klassendefinition sehen, da sich diese bereits im Dialogfenster "Klassendefinition" befindet. Wenn die Eigenschaft an beiden Stellen angezeigt wird, wird der Benutzer verwirrt. Daher wird sie ausgeblendet.

Siehe auch das Schlüsselwort VISIBLE.

JUSTIFY

Dieser Befehl wird in SAPROPS.CFG oder USRPROPS.TXT **nicht länger verwendet**. Wenn Sie ihn in USRPROPS.TXT angeben, wird kein Fehler verursacht, sondern der Befehl wird einfach vom USRPROPS.TXT-Parser **ignoriert**. Dieser Befehl war vorher eines der Argumente des LAYOUT-Befehls. Wenn er verwendet wurde, wurden alle Steuerelemente am rechten und linken Rand des Dialogfensters aufgeführt.

Siehe auch die Schlüsselwörter LAYOUT und ALIGN.

KEY

Das Schlüsselwort KEY dient zum Erstellen einer Eigenschaft als Schlüssel. Mit Schlüsseln werden die Namensbereiche von Modellierungselementen in der Enzyklopädie bestimmt. Das Schlüsselwort KEY besitzt auch eine zweite Verwendung – es ist eines der zulässigen Argumente, die auf das Schlüsselwort FORMAT im Befehl DISPLAY folgen. Weitere Informationen zum letzten Verwendungszweck finden Sie unter KEY (Verwendung bei DISPLAY).

Standardmäßig unterscheidet sich jedes Modellierungselement in einer Enzyklopädie durch seine Klasse (ob es ein Diagramm, ein Symbol oder eine Definition ist), seinen Typ (ob es sich um ein UML-Anwendungsfalldiagramm, ein BPMN-Prozessdiagramm etc. handelt) und seinen Namen (z. B. das Anwendungsfalldiagramm "Reservierungssystem" im Vergleich zum Anwendungsfalldiagramm "Personalsystem"). Zusätzlich zu diesen integrierten Standardwerten können Sie weitere Schlüssel für ein Definitionsmodellierungselement angeben - z. B. auf eine Klassenattributsdefinition wird von der entsprechenden Klassendefinition verwiesen, auf die wiederum die entsprechende Paketdefinition verweist.

Um den KEY-Befehl zu verwenden, geben Sie diesen in der Eigenschaft an, die als Schlüssel einer Definition dienen soll. Der KEY-Befehl kann an fast jeder Stelle innerhalb der Beschreibung einer Eigenschaft eingefügt werden, da er jedoch eine so große Bedeutung hat, empfiehlt es sich, ihn an die erste Stelle innerhalb der geschweiften Klammern der Eigenschaft zu stellen - direkt vor das Schlüsselwort EDIT.

KEY (Fortsetzung)

Beispiel:

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

Bei einer Eigenschaft, die einen Schlüssel darstellt und auf eines oder mehrere andere Objekte verweist, z. B. eine LISTOF- oder ONEOF-Eigenschaft (keine einfache Text- oder Zahleneigenschaft), muss der Benutzer die Klasse und den Klassentyp der Objekte, auf die verwiesen wird, angeben, wenn er beim Arbeiten mit Rational System Architect einen Wert für die Eigenschaft eingibt.

Beispiel:

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

Mit der obigen Anweisung wird angegeben, dass die Eigenschaft "Anwendungsfallname" ("Use Case Name") auf eine Definition des Typs "Anwendungsfall" verweist. Die Definition ist der Standardwert, wenn keine *Klasse* angegeben wird (*Klasse* im Sinne von Rational System Architect, also ein Diagramm, ein Symbol oder eine Definition).

Der Eigenschaftswert selbst enthält alles weitere Material, das erforderlich ist, um die Objekte, auf die verwiesen wird, zu identifizieren. Wenn die Verweisklasse bzw. der Verweistyp der Eigenschaft keine Schlüsseleigenschaften aufweist, wird als Referenzwert einfach der Name des Objekts verwendet (da Klasse und Typ nicht bekannt sind), wenn die Verweisklasse bzw. der Verweistyp jedoch Schlüsseleigenschaften besitzt (beispielsweise "Anwendungsfall" im obigen Beispiel, das über die Schlüsseleigenschaft "Paket" verfügt), muss Rational System Architect die Werte dieser Schlüsseleigenschaften kennen, um das Verweisobjekt ordnungsgemäß zu ermitteln.

Sie codieren dies entweder in der Datei USRPROPS.TXT, sodass Rational System Architect automatisch die Werte für den Endbenutzer abruft, oder Sie erzwingen, dass der Endbenutzer den vollständig qualifizierten Namen, mit Punkten als Trennzeichen für die Schlüsselabschnitte, eingibt.

KEY (Fortsetzung)

- Bei Verwendung des KEYED BY-Befehls ruft Rational System Architect den Wert für Benutzer automatisch ab.
- Wenn keine KEYED BY-Klausel für die Eigenschaft angegeben ist, erwartet Rational System Architect diese zusätzlichen Schlüsselwerte im Verweis - d. h. der Benutzer muss den vollständig qualifizierten Namen des Verweisobjekts eingeben und die Schlüsselwerte durch Punkte trennen (bei einem Anwendungsfallschritt mit der Bezeichnung *"Specify email"* in einem Anwendungsfall *Order_Product* in einem Paket "*Order System*" muss der Benutzer "*Order System".Order_Product".Specify email"* eingeben.

Anmerkung: Heterogene Verweiseigenschaften unterscheiden sich in dieser Hinsicht. Siehe HETEROGENEOUS.

Eine andere Verwendung der KEYED BY-Klausel ist die Erstellung einer Liste zusammengehöriger Elemente. Beispielsweise gehören alle Anwendungsfallschritte, auf die in der Eigenschaft "Anwendungsfallschritte" einer Anwendungsfalldefinition verwiesen wird, zum selben Anwendungsfall - in diesem Fall zu dem, der die Eigenschaft "Anwendungsfallschritte" aufweist. Wenn eine Eigenschaft mit mehreren Verweisen (wie ListOf) auf Objekte verweist, die alle zum selben übergeordneten Objekt gehören, empfiehlt es sich, eine oder mehrere andere Eigenschaften zum Bestimmen des übergeordneten Objekts zu verwenden. In diesen Situationen wird mit der KEYED BY-Klausel Rational System Architect mitgeteilt, welche Eigenschaften verwendet werden sollen.

Anmerkung: Schlüsseleigenschaften einer Definition werden in einem Raster, das mit einem ASGRID-Befehl gebildet wurde, nicht angezeigt. Beispielsweise werden in einer Anwendungsfalldefinition Anwendungsfallschritte in einem Raster dargestellt, das mit einem ASGRID-Befehl erstellt wurde. Die Schlüsseleigenschaften der Anwendungsfallschritte jedoch (zu denen "Paket" und "Anwendungsfall" gehören) werden im Raster der Anwendungsfallschritte nicht angezeigt.

Anmerkung: Es ist nicht möglich, ein KEY EDIT ONEOF-Schlüsselwort zu einem Diagramm hinzuzufügen.

Siehe auch das Schlüsselwort KEYED BY.

KEYED BY

Eine KEYED BY-Klausel wird optional verwendet, um anzugeben, wie die Schlüsselkomponenten eines Verweisobjekts gefunden werden können. Die KEYED BY-Klausel weist einen Anteil jeder Schlüsselkomponente getrennt durch ein Komma auf.

Die KEYED BY-Klausel bietet zwei Vorteile:

1. Sie sorgt dafür, dass der Endbenutzer nicht mehr den vollständig qualifizierten Namen eines Referenzwerts (mit Punkten als Trennzeichen) eingeben muss. Beispielsweise muss der Endbenutzer bei einer Eigenschaft, die auf ein Klassenattribut mit der Bezeichnung ***E-Mail*** der Klasse ***Kunde*** des Pakets ***"Bestellsystem"*** verweist, nicht mehr ***"Bestellsystem".Kunde.E-Mail***, sondern einfach nur ***E-Mail*** eingeben.
2. Sie kann verwendet werden, um sicherzustellen, dass alle Schlüsselkomponenten eines Referenzwerts identisch sind. Beispielsweise enthält die LISTOF-Eigenschaft "Klassenattribut" in einer Klassendefinition eine Liste der Attribute, die alle zur selben Klasse und zum selben Paket gehören.

Beispiel:

Die KEYED BY-Klausel der Eigenschaft "Klassenattribut" der Klasse kann beispielsweise folgendermaßen lauten:

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

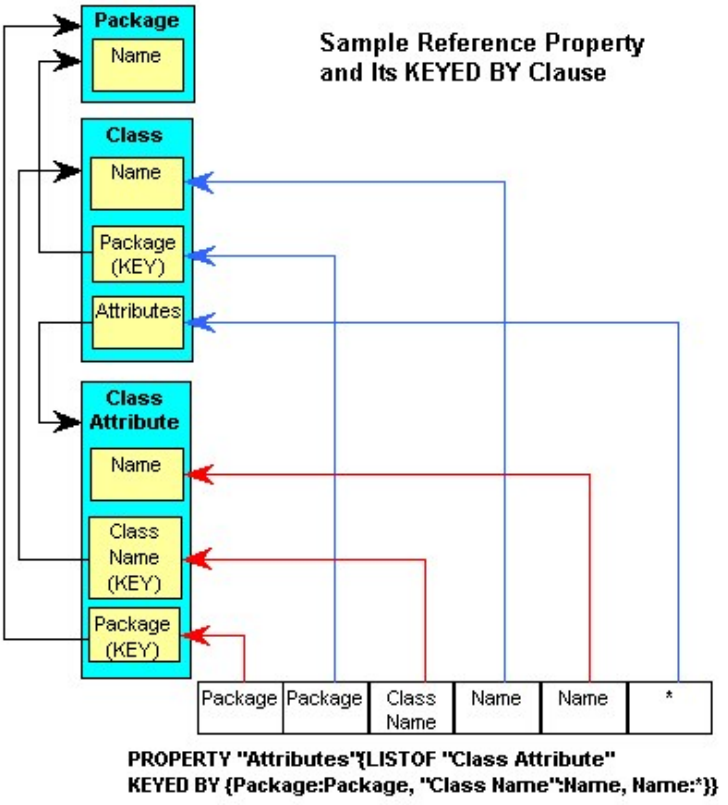
Im obigen Beispiel lauten die drei *Schlüsselkomponenten* (durch Kommata getrennt) wie folgt: .Package:Package, "Class Name":Name, and Name:*. Diese Komponenten beziehen sich auf die drei Teile, die erforderlich sind, um die Definitionen für das Klassenattribut zu bestimmen, auf die verwiesen wird, nämlich den Paketnamen, den Klassennamen und den Klassenattributnamen. Bei umgekehrter Reihenfolge wird Folgendes angegeben:

- Der Name des Klassenattributs befindet sich in ***dieser*** Eigenschaft (* bedeutet "hier"), also: **Name:***

KEYED BY (Fortsetzung)

- Der Wert der Schlüsseleigenschaft "Klassenname" in der Klassenattributdefinition befindet sich in diesem Objektnamen, also: **"Class Name":Name**.
- Der Wert der Schlüsseleigenschaft "Paket" in der Klassenattributdefinition befindet sich in der Paket-eigenschaft dieses Objekts, also: **Package:Package**

In der folgenden schematischen Darstellung wird gezeigt, wie die KEYED BY-Klausel im obigen Beispiel verwendet wird. Außerdem kann die Darstellung ein allgemeines Verständnis der Klausel vermitteln.



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

In der schematischen Darstellung wird gezeigt, was oben beschrieben wurde - in die Definition einer Klasse wird ein Klassenattribut durch Angabe des zugehörigen Pakets

KEYED BY (Fortsetzung)

(gespeichert in der Paketeigenschaft des Klassenattributs und übernommen vom Paketwert der aktuellen Klasse), ihres Klassennamens (gespeichert in der Eigenschaft "Klassenname" des Klassenattributs und übernommen vom tatsächlichen Namen der Klasse) und des Namens (gespeichert in der Eigenschaft "Name" des Klassenattributs und von dort übernommen) eingegeben.

Zusammenfassung:

1. Für jede Schlüsselkomponente des **Verweisobjekts** weist die KEYED BY-Klausel eine Komponente auf.
2. Die Komponenten der KEYED BY-Klausel werden durch Kommata getrennt.
3. Jede Komponente hat zwei Bestandteile:
    - Mit dem ersten Teil wird die Schlüsselkomponente des Verweisobjekts angegeben.
    - Mit dem zweiten Teil wird angegeben, wo sich der Wert dieser Komponente befindet.
    - Beide Teile werden durch einen Doppelpunkt getrennt.

Es können jedoch bestimmte Standardwerte übernommen werden, um die KEYED BY-Klausel zu vereinfachen. Wenn die beiden Teile der Komponente übereinstimmen, kann der zweite Teil weggelassen werden, und wenn der zweite Teil der letzten Komponente weggelassen wird, wird angenommen, dass diese sich "hier" befindet, deshalb der Stern. Daher wird in der Praxis die KEYED BY-Klausel der Eigenschaft "Attribute" der Klasse folgendermaßen codiert:

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

Natürlich müssen alle in der KEYED BY-Anweisung verwendeten Eigenschaften vorhanden sein. Deshalb überprüft Rational System Architect, ob eine Eigenschaft "Paket" und eine Eigenschaft "Klassenname" in der Definition "Klassenattribut" vorhanden sind und ob beide KEY-Eigenschaften sind. Zusätzlich zum verringerten Aufwand beim Codieren gebräuchlicher Schlüsselkomponenten in einer LISTOF-Eigenschaft wie dieser, wird durch das Bereitstellen einer KEYED BY-Klausel bei Verwendung anderer Eigenschaften zum Bereitstellen einheitlicher Werte **sichergestellt, dass dieselben Werte für jeden Verweis verwendet werden**. Daher müssen im

KEYED BY (Fortsetzung)

verwendeten Beispiel alle Klassenattribute, auf die in der Eigenschaft "Attribute" der Klasse verwiesen wird, zur selben Klasse im selben Paket gehören - in diesem Fall ein wünschenswertes Merkmal.
In anderen Fällen ist es wünschenswert, dass die Schlüsselkomponenten des Objekts, auf das verwiesen wird, aus Gründen der Klarheit und Einfachheit getrennt sind. Unter solchen Umständen wird eine KEYED BY-Klausel zur Bezeichnung der Eigenschaften verwendet, die die einzelnen Komponenten bereitstellen. Wenn daher eine Eigenschaft vom Typ KEY ist und auf ein Objekt mit KEY-Eigenschaften verweist, macht es Rational System Architect **erforderlich,** dass sich die Komponenten in separaten Eigenschaften befinden.
Häufig sollen bestimmte Schlüsselkomponentenwerte zusätzlich zu den Namen im Verweis selbst bereitgestellt und nicht aus einer anderen Eigenschaft übernommen werden. Dies ist möglicherweise der Fall, wenn keine geeignete Eigenschaft zum Bereitstellen eines Werts vorhanden ist oder wenn die Schlüsselkomponente nicht für alle Verweise in der Eigenschaft denselben Wert aufweisen soll. In diesem Fall wird das Schlüsselwort QUALIFIABLE verwendet. Beispielsweise ist in der Klassendefinition folgende Eigenschaft vorhanden:

PROPERTY "Operations" {Edit … ParmListOf "Method"
KEYED BY {"Package","Class Name":Name,"Formal Parameters"
QUALIFIABLE, Name } … }

Damit wird angegeben, dass zwar die Werte der Schlüsseleigenschaften "Paket" und "Klassenname" der Methoden, auf die verwiesen wird, jeweils von Eigenschaft und Name des "Pakets" der Klasse übernommen werden sollen, die Werte der Eigenschaft "Formale Parameter" der Methoden und ihre Namen jedoch aus der Eigenschaft "Vorgänge" der Klasse selbst übernommen werden müssen. Daher enthält jeder Verweis zwei Komponenten, den Wert der Eigenschaft "Formale Parameter" und den Wert des Namens, getrennt durch einen Punkt.

Beachten Sie, dass in Rational System Architect bei KEY-Eigenschaften mit einer KEYED BY-Klausel **nicht** das Schlüsselwort QUALIFIABLE verwendet werden darf. Dies gilt aus den oben genannten Gründen der Klarheit und Einfachheit.

KEY (Verwendung für Anzeige)

KEY ist ebenfalls eines der zulässigen Argumente, die nach FORMAT im **DISPLAY**-Befehl stehen können. Eigenschaften, die als Schlüssel designiert sind, werden in einem separaten Abschnitt des Symbols angezeigt.

Beispiel:

```
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" }
...}
```

Im obigen Beispiel werden mit dem Befehl FORMAT KEY alle Attribute, die vom Benutzer als Primärschlüssel designiert werden, unter der Legende "Primärschlüssel" bei einem Entitätssymbol abgelegt. Mit dem Befehl FORMAT NONKEY werden alle Attribute ohne Primärschlüsselfunktion unter der Legende "Attribute ohne Schlüssel" beim Entitätssymbol abgelegt.

LABEL

Der **LABEL**-Befehl wird für zwei Zwecke verwendet.

Zweck 1: LABEL ist eines der Argumente des **ALIGN**-Befehls. Damit werden alle Steuerelemente an einer Stelle rechts von der breitesten Beschriftung in dieser Spalte ausgerichtet. (Im Gegensatz dazu wird mit dem Schlüsselwortpaar ALIGN OVER der Name oberhalb der Eigenschaft platziert.)

Beispiel:

```
Definition "My Definition"
{
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}
}
```

Im obigen Beispiel stellt das Steuerelement "My Really Long Property 23" ein Textfeld dar, das eine Stelle rechts von der Beschriftung platziert wird. Alle anderen Textfeldsteuerelemente für andere Eigenschaften im Dialogfenster werden an diesem Steuerelement ausgerichtet.

Siehe auch die Schlüsselwörter ALIGN, BODY und OVER.

LABEL

(Fortsetzung)

Zweck 2: Mit dem Befehl LABEL werden Registerkarten (Kapitel), Gruppen oder Eigenschaften in einem Dialogfenster **umbenannt**. Sie können einen in SAPROPS definierten Eigenschaftsnamen nicht entfernen. Sie können jedoch mithilfe des LABEL-Befehls in USRPROPS.TXT den Text ändern, der für die Eigenschaft angezeigt wird.

Beispiel 2:

```
DIAGRAM "Data Flow Diagram" {
  PROPERTY "Event Label Prefix"
    { EDIT Text LENGTH 10 }
  PROPERTY "Key Letters"
    { EDIT text LENGTH 10 LABEL "Process Prefix" } ..}
```

Wenn Sie den obigen Code zu USRPROPS.TXT hinzufügen (und Ihre Enzyklopädie erneut öffnen), wird als Beschriftung des Steuerelements "Schlüsselbuchstaben" ("Key Letters") die Bezeichnung "Präfixverarbeitung" ("Process Prefix") angezeigt.

Umbenennen einer Gruppe in einer Definition
Mit dem LABEL-Befehl können Sie eine Gruppe umbenennen. Wenn Sie eine leere Textzeichenfolge (" ") angeben, wird für das Gruppenfeld nichts angezeigt.

Beispiel 2:

```
DEFINITION "Attribute" {
GROUP "other stuff" LABEL "" }
```

Weitere Informationen zum Umbenennen eines KAPITELS finden Sie beim CHAPTER-Befehl. Siehe auch die Schlüsselwörter ALIGN und BODY.

LABELPOS

Hierbei handelt es sich um einen Parameter des PLACEMENT-Befehls, mit dem Sie die exakte Position eines Eigenschaftsnamens (oder einer Beschriftung) im Dialogfenster DIAGRAMM, SYMBOL oder DEFINITION angeben können. Der Befehl LABELPOS weist zwei Argumente auf – die horizontale Position (vom oberen Rand des Dialogfensters) in Windows-Einheiten und die vertikale Position (vom linken Rand des Dialogfensters) in Windows-Einheiten.

Syntax:

PLACEMENT { **LABELPOS(4, 52)** PROPPOS (horizontal-positioning, vertical-positioning) PROPSIZE (width, height) }

Beispiel:

```
DEFINITION "My Definition"
{
PROPERTY "Table Name" { EDIT Text LENGTH 31 PLACEMENT {
LABELPOS (4, 24) PROPPOS (20, 24) PROPSIZE(150, 12)} }
}
```

Beachten Sie im obigen Beispiel, dass LABELPOS und PROPPOS dieselben y-Koordinaten aufweisen (24) - das bedeutet, dass die Oberkante der Buchstaben bei beiden 24 Einheiten vom oberen Rand des Dialogfensters entfernt sind. Das heißt, die Beschriftung befindet sich links neben dem Steuerelement (nicht darüber). Beachten Sie außerdem, dass die Differenz zwischen der x-Koordinate von PROPPOS und der x-Koordinate von LABELPOS (20 - 4 = 16) ausreichend Platz (16 - 10 = 6 Einheiten) für die aus 10 Zeichen bestehende Beschriftung ("Table Name") lässt, da diese links neben dem Ausgangspunkt der Anfangsposition des Steuerelements der Eigenschaft liegt.

Wichtig: Allgemeine Informationen zur Platzierung und Tipps zur Dimensionierung finden Sie in Kapitel 2 dieses Handbuchs.

Siehe auch die Schlüsselwörter PLACEMENT, PROPPOS, PROPSIZE und FORMAT.

LAYOUT

Mit diesem Schlüsselwort wird das Layout der Eigenschaften in einem Diagramm-, Symbol- oder Definitionsdialogfenster angegeben. (Beachten Sie, dass das Symboldialogfenster als letzte Registerkarte im Definitionsdialogfenster enthalten ist.)

Zwischen den eckigen Klammern des LAYOUT-Befehls verwenden Sie Argumente zum Festlegen des Layouts aller Eigenschaften, die mit diesem LAYOUT-Befehl gesendet wurden. Sie können angeben, wie viele Spalten das Layout für die Eigenschaften des Dialogfensters aufweisen soll und wie die Eigenschaften ausgerichtet sein sollen.

Sie können mehrere LAYOUT-Befehle für ein Diagramm-, Symbol- oder Definitionsdialogfenster angeben. Sie können auch einen LAYOUT-Befehl für ein gesamtes Dialogfenster angeben und/oder diesen innerhalb jeder GRUPPE in einem Dialogfenster oder innerhalb jeder Registerkarte (KAPITEL) in einem Dialogfenster außer Kraft setzen.

Syntax:

LAYOUT { [alignment_criteria] [PACK_TAB_criteria] [Number of Columns] [JUSTIFY] }

Oder spezieller:

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

Beispiel:

```
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 }
..}
```

Im obigen Beispiel erhalten alle Eigenschaften im Symboldialogfenster des Objekts ein Layout mit zwei Spalten.

Siehe auch die Schlüsselwörter ALIGN, BODY, LABEL, OVER, PACK, TAB, COLS und JUSTIFY.

LEGEND

Hierbei handelt es sich um die Zeichenfolge der anzeigbaren Eigenschaft in einem rechteckigen Symbol, mit der der Eigenschaftsname überschrieben wird.

Beispiel:

```
PROPERTY "Description"
  { EDIT ListOf Data
    DISPLAY { FORMAT Key LEGEND "Key data" }
  }
```

Syntax:

LEGEND "<Ihr Text>": Beliebiger Text, den Sie in Anführungszeichen setzen, wird oberhalb der Eingabe im Symbol angezeigt, jedoch nur, wenn ein Wert für die Eingabe vorliegt.

LEGEND "": Zeigt eine gerade Linie ohne Worte an, jedoch nur, wenn ein Wert für die Eingabe vorliegt.

LEGEND "$$FORCE$$": Zeigt eine horizontale Linie oberhalb der Eingabe im Symbol an. Diese Linie dient als Trennlinie. Das Schlüsselwort "$$FORCE$$" unterscheidet sich von der einfachen Verwendung von " " dadurch, dass die Anzeige einer horizontalen Linie zwingend ist, selbst wenn die Eigenschaftenanzeige über das Dialogfenster für den Anzeigemodus unterdrückt wird.

LEGEND "$$NONE$$": Zeigt keine horizontale Linie oberhalb der Eingabe im Symbol an, unabhängig davon, ob Werte für die Eingabe vorliegen. Diese Linie dient in der Regel als Trennlinie.

LEGEND "$$VFORCE$$": Ermöglicht das Verschieben von Eigenschaften innerhalb von Symbolen von links nach rechts und zieht vertikale Linien dazwischen. Siehe das Schlüsselwort VFORCE.

LEGEND "$$VNONE$$": Ermöglicht das Verschieben von Eigenschaften von links nach rechts, stellt jedoch *keine* Trennlinie bereit. Siehe das Schlüsselwort VNONE.

Siehe auch das Schlüsselwort DISPLAY.

LENGTH

Gibt die Anzahl der Zeichen an, die der Benutzer im Eigenschaftenfeld eingeben kann.

Beispiel:

```
PROPERTY "From Entity"
  { EDIT TEXT LENGTH 80 }
```

Im obigen Beispiel darf *From Entity* maximal 80 Zeichen lang sein.

LINES

Legt die Anzahl Zeilen in der Verschachtelungstiefe für ein Eigenschaftenfeld fest.

Beispiel:

```
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 }
```

Dieses Schlüsselwort ist nur sinnvoll, wenn der automatisch bereitgestellte Platz deutlich größer oder kleiner sein soll als die Standardeinstellung.

Siehe auch das Schlüsselwort ZOOMABLE.

LIST

Das Schlüsselwort LIST erfüllt in der Datei USRPPROPS.TXT zwei Zwecke. Die Standardlänge beträgt 1200.

Zweck 1: Mit dem Schlüsselwort LIST wird eine Liste möglicher Textwerte erstellt. Es muss in zwei Bereichen definiert werden, und zwar ganz oben in der Datei USRPROPS.TXT, wo Sie die Liste möglicher Werte angeben, und innerhalb der Eigenschaft, die die Liste verwendet. Alle Spezifikationsanweisungen für die Liste müssen an erster Stelle der Datei USRPROPS.TXT vor allen Spezifikationsanweisung für DIAGRAMME, DEFINITIONEN oder SYMBOLE stehen.

LIST (Forsetzung)

Beispiel:

```
List "Method Stereotypes"
{
VALUE "Get"
VALUE "Let"
VALUE "Set"
}
DEFINITION "Method"  {..
PROPERTY "Stereotype"{ EDIT Text LIST "Method Stereotypes"
Default "" LENGTH 30  }  …}
```

Optionsfelder im Vergleich zu Dropdown-Listen

Rational System Architect zeigt automatisch eine Liste mit Optionsfeldern an, wenn die Anzahl der Werte in der LIST-Anweisung vier oder weniger beträgt. Liegen fünf oder mehr Werte vor, wird die Liste automatisch als Dropdown-Listenfeld angezeigt. Die Benutzer können eigene Werte in einem Dropdown-Listenfeld eingeben. Wenn Sie ein Dropdown-Listenfeld wünschen, jedoch nur vier oder weniger LIST-Werte vorhanden sind, verwenden Sie das Schlüsselwort LISTONLYCOMBO.

Zweck 2: Das Schlüsselwort LIST ist ebenfalls eines der zulässigen Argumente, die nach FORMAT im **DISPLAY**-Befehl stehen können. Mit dem Schlüsselwort LIST werden die Elemente für das Symbol in einer Liste angezeigt - jedes Leerzeichen führt zu einer neuen Zeile, es sei denn, das Leerzeichen wird in Anführungszeichen gesetzt.

Beispiel:

```
DEFINITION "Operational Node"
{PROPERTY "Operational Activities" {EDIT LISTOF "Operational
Activity" LENGTH 2000 DISPLAY {FORMAT LIST Legend "Activities"}
..}
```

Siehe auch die Schlüsselwörter LISTONLY und LISTONLYCOMBO.

LISTOF

Hierbei handelt es sich um einen der zulässigen Typen für eine Eigenschaft. Das Schlüsselwort wird zusammen mit dem Schlüsselwort EDIT verwendet, um anzugeben, dass die Eigenschaft auf eine **Liste mit anderen Definitionen verweist**. Beispielsweise enthält eine Klasse eine Eigenschaft mit der Bezeichnung "Attribute". Dabei handelt es sich um eine Liste mit Klassenattributen. Klassenattribut ist an und für sich schon ein Definitionstyp mit einem eigenen Satz Eigenschaften. Im Gegensatz dazu gibt es die Eigenschaft einer Klasse mit der Bezeichnung "Zugriffstyp", bei der es sich um eine Liste mit einfachen Textauswahlmöglichkeiten handelt, beispielsweise "Öffentlich", "Privat", "Geschützt" etc. (Mit dem LIST-Befehl wird diese einfache Textliste definiert; siehe LIST.) LISTOF unterscheidet sich von ONEOF, mit dem angegeben wird, dass eine Eigenschaft auf genau eine andere Definition verweist - ein Beispiel ist, dass eine Klasse eine Eigenschaft mit der Bezeichnung "Paket" enthält, die angibt, dass sich darin ein Paket befindet; "Paket" ist eine Definition in und von sich selbst.

LISTOF wird zusammen mit dem Schlüsselwort EDIT verwendet. Die Syntax lautet folgendermaßen:

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

Das Schlüsselwort ASGRID wird häufig mit LISTOF verwendet. ASGRID stellt die Liste der Definitionen in einem Raster dar. Wenn es nicht verwendet wird, werden die Definitionen in einer standardmäßigen Listenstruktur aufgeführt. LISTOF wird manchmal mit dem Schlüsselwort ZOOMABLE und mit COMPLETE verwendet (an anderer Stelle in diesem Kapitel beschrieben). Bei einer LISTOF-Eigenschaft wird das Schlüsselwort LENGTH standardmäßig auf 1200 gesetzt. Mit LENGTH wird angegeben, wie viele Zeichen der Benutzer im Eigenschaftenfeld eingeben kann - in diesem Fall handelt es sich dabei um die Gesamtzahl der Zeichen der Namen von Definitionen, die in die Liste passen.

Beispiel:

```
DEFINITION "Use Case"
{
PROPERTY "Preconditions" { ZOOMABLE EDIT ListOf "Pre/Post Condition" LENGTH 1200  }...}
```

Siehe auch die Schlüsselwörter ONE OF, EXPRESSIONOF, COMPLETE und ZOOMABLE.

LISTONLY

Mit diesem Schlüsselwort wird angegeben, dass die Werte für eine Eigenschaft aus der angezeigten Liste (die mit dem Schlüsselwort LIST ganz oben in der Datei URSPROPS.TXT erstellt wurde) übernommen werden müssen - es ist ***nicht zulässig***, dass der Benutzer eigene Werte in die Liste eingibt.

Beispiel:

```
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" }
..}
```

Im obigen Beispiel wird die Liste im Definitionsdialogfenster "Anwendungsfallschritt" als Dropdown-Liste bereitgestellt, in der Sie eigene Eingaben vornehmen können. Beachten Sie, dass sich in der Listenanweisung fünf Werte befinden. Bei vier oder weniger Werten würde die Liste im Definitionsdialogfenster "Anwendungsfallschritt" als Auswahl mit mehreren Optionsfeldern bereitgestellt. Wenn Sie eine Dropdown-Liste wünschen, selbst wenn nur vier oder weniger Listenwerte vorhanden sind, verwenden Sie das Schlüsselwort LISTONLYCOMBO.

Siehe auch die Schlüsselwörter LIST und LISTONLYCOMBO.

LISTONLYCOMBO

Stellt eine Dropdown-Liste bereit, und zwar unabhängig davon, wie viele LIST-Werte enthalten sind. Außerdem kann der Benutzer keine eigenen Werte in die Liste eingeben.

Das Schlüsselwort LISTONLYCOMBO bietet eine Funktionalität, die der LIST-Befehl nicht aufweist: Beim Verwenden des LIST-Befehls zeigt Rational System Architect eine Liste automatisch als Liste von Kontrollkästchen an, wenn die Anzahl der Werte in der LIST-Anweisung vier oder weniger beträgt. Liegen fünf oder mehr Werte vor, wird die Liste automatisch als Dropdown-Listenfeld angezeigt. Die Benutzer können eigene Werte in einem Dropdown-Listenfeld eingeben. Wenn Sie ein Dropdown-Listenfeld wünschen, jedoch nur vier oder weniger LIST-Werte vorhanden sind, verwenden Sie das Schlüsselwort LISTONLYCOMBO.

Beispiel:

```
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" }
..}
```

Im obigen Beispiel wird die Liste im Definitionsdialogfenster "Anwendungsfallschritt" als Dropdown-Liste bereitgestellt, auch wenn nur drei Werte in der Listenanweisung vorhanden sind. Wenn Sie die einfache LIST-Anweisung verwendet hätten, wären die Werte als Optionsfelder angezeigt worden, da weniger als fünf Werte vorliegen.

Siehe auch die Schlüsselwörter LIST und LISTONLY.

MAX; MAXIMUM

Gibt die maximal zulässige Zahl für eine Eigenschaft an, die als numerisch definiert ist. In einem numerischen Feld können nur Zahlen eingegeben werden.

Beispiel:

```
PROPERTY Length
  { EDIT numeric LENGTH 2 MINIMUM 1 MAXIMUM 99  }
```

MENU

Verweist auf die Grafik, die zum Darstellen eines Symbols im Menü und in der Symbolleiste "Zeichnen" verwendet wird, und zwar im Vergleich zum Diagrammarbeitsbereich.

Beispiel:

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

Im obigen Beispiel wird das Bild "satellitetoolbar.bmp" im Menü "Zeichnen" des Diagramms "Drahtloses Netzwerk" ("Wireless Network") positioniert.

Siehe auch das Schlüsselwort DEPICTIONS.

MIN; MINIMUM

Gibt die minimale zulässige Zahl für eine Eigenschaft an, die als numerisch definiert ist. In einem numerischen Feld können nur Zahlen eingegeben werden.

Beispiel:

```
PROPERTY Length
  { EDIT numeric LENGTH 2 MINIMUM 1 MAXIMUM 99  }
```

MINISPEC

In Rational System Architect ist "Minispec" die Anweisung, mit der die Verarbeitungslogik eines Prozesssymbols ausgedrückt wird. Minispezifikationen werden mithilfe einer formalen Syntax geschrieben, die häufig als "strukturiertes Englisch" bezeichnet wird. Das Schlüsselwort MINISPEC wird zusammen mit dem Schlüsselwort EDIT verwendet.

MINISPEC (Fortsetzung)

Beispiel:

```
DEFINITION "Process"
{
PROPERTY "Description"
{ ZOOMABLE EDIT MINISPEC LENGTH 1500 LABEL "Minispec" }
...}
```

Eine Minispezifikation ist eine Anweisung, mit der die Verarbeitungslogik eines Prozesssymbols ausgedrückt wird, d. h. es wird angegeben, wie der Prozess die Eingabedaten in Ausgabedaten umwandelt.

Nachstehend finden Sie ein Beispiel für eine MINISPEC-Anweisung:

```
If ISBN number brand new,
Create "ISBN MASTER LIST"
Else
Update "Borrower Request"
```

Rational System Architect kann die Ein- und Ausgabedatenflüsse eines Prozesses mithilfe der Minispec-Wörter in Bezug auf Datenelemente und Datenstrukturen in Datenflüssen neu ausrichten. Für die Lastausgleichsfunktion ist eine Wort-für-Wort-Analyse des Textes für die Suche nach signifikanten Wörtern erforderlich. Signifikante Wörter werden entweder durch einfache oder durch doppelte Anführungszeichen markiert. Sie können festlegen, dass das System jedes Wort oder nur die signifikanten markierten Wörter berücksichtigt.

Standardmäßig werden vom System nur die signifikanten Wörter berücksichtigt, die speziell mit Anführungszeichen markiert sind, z. B.:

```
Compute "extended_cost" = "unit_cost" times "quantity"
```

Wenn das System jedes in den Minispezifikationen enthaltene Wort und nicht nur die zwischen Anführungszeichen berücksichtigen soll, müssen Sie "MinispecUsesQuotes" in der Datei "SA2001.INI" auf "N" setzen. Die obige Beispielminispezifikation kann dann folgendermaßen geschrieben werden:

```
Compute extended_cost = unit_cost times quantity
```

NAME

Mit diesem Schlüsselwort wird angegeben, dass ein Teil des Schlüssels einer Definition der Name des Objekts ist und daher auch der Name des übergeordneten Objekts sein kann.

Beispiel:

```
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} }
...}
```

Im obigen Beispiel ist der Schlüssel der Eigenschaft "Tabellenname" ("Table Name") in der Definition eines Triggers der Name der Tabelle, in der dieser Trigger definiert ist. Der Name des Triggers ist ebenfalls Teil des Schlüssels.

NODESC; NODESCRIPTION

Mit diesem Schlüsselwort wird angegeben, dass die Definition nicht über eine BESCHREIBUNG der Eigenschaft verfügt.

Syntax:

```
DEFINITION <def_name>
  { NODESC
  }
```

Beispiel:

```
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 }
..}
```

Es gibt eine Reihe von Definitionstypen in Rational System Architect, für die das Feld DESCRIPTION durch die Verwendung des Schlüsselworts NODESC entfernt wurde. Dabei handelt es sich um Definitionstypen, für die keine Beschreibung erforderlich ist und bei denen eine Beschreibung den Benutzer nur stören würde. Beispiele sind "Triggervorlage", "Tabellensynonym", "Tabelle", "Gespeicherte Prozeduren" und "Ansichten".

NONADDR, NONADDRESSABLE

Mit diesem Schlüsselwort werden Definitionen aus der Adressliste entfernt. Es gibt 13 Definitionstypen, die als "Adressierbar" vordefiniert wurden, d. h. Sie können ein Symbol in einem Diagramm mit diesen Definitionstypen "adressieren" (wählen Sie ein beliebiges Symbol und anschließend "Verzeichnis", "Adressen" und den Definitionstyp aus). "Adressierbare" Definitionstypen sind in der Regel Elemente wie Anforderungen, Regeln, Testpläne etc., also Elemente, mit denen das Symbol im Diagramm "adressiert" wird (d. h., mit denen die Anforderungen des Symbols erfüllt werden). Um eine dieser Definitionen aus dem Verzeichnis im Dropdown-Menü **Adressen** zu entfernen, ändern Sie die Anweisung in USRPROPS.TXT.

Beispiel:

```
DEFINITION "Change Request"
  {
    NONADDR
  }
```

Siehe auch das Schlüsselwort ADDRESSABLE.

NONE

Dieses Schlüsselwort heißt eigentlich $$NONE$$ und wird zusammen mit dem Schlüsselwort DISPLAY verwendet. Weitere Informationen finden Sie beim Schlüsselwort DISPLAY.

NONKEY

Dies ist eines der zulässigen Argumente, die nach FORMAT im **Display**-Befehl stehen können. Elemente, die nicht als Schlüssel designiert sind, können in einem separaten Abschnitt des Symbols angezeigt werden.

Beispiel:

```
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" }
...}
```

Im obigen Beispiel werden mit dem Befehl FORMAT NONKEY alle Attribute, die nicht als Primärschlüssel designiert sind, unter der Legende "Attribute ohne Schlüssel" ("Non-Key Attributes") bei einem Entitätssymbol abgelegt.

NOTHING

Wird im Befehl RELATE BY NOTHING verwendet.

Siehe RELATE BY.

NUMERIC

Hierbei handelt es sich um einen der zulässigen Typen für eine Eigenschaft. Damit wird angegeben, dass die Eigenschaft eine Zahl ist und nur Zahlen (und das Plus- oder Minuszeichen) in das Feld eingegeben werden dürfen. Mit der LENGTH-Anweisung wird festgelegt, wie viele Zahlen in das Feld eingegeben werden können. Der Benutzer kann keine Dezimalzeichen oder ähnliche Zeichen in das Feld eingeben. Nur Ziffern sowie Plus- und Minuszeichen sind zulässig.

Beispiel:

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

OF DEFINITION REFERENCED IN

Hiermit können Sie eine eingeschränkte Liste von Definitionen angeben, aus denen Sie eine Auswahl treffen können, wenn Sie auf die Schaltfläche "Optionen" für eine Eigenschaft klicken. Das Schlüsselwort wird verwendet, um eine EDIT LISTOF- oder EDIT ONEOF-Anweisung zu verbessern. Sie können angeben, dass nur die Definitionen aufgeführt werden, die zu einer bestimmten Verweisdefinition gehören.

Beispiel 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 "$$FORCE$$"}  }
..}
```

Wenn Sie im obigen Beispiel im Attributraster eines Objekts auf die Schaltfläche "Optionen" klicken, werden nur die Attribute bereitgestellt, die in der im Objekt enthaltenen Klasse verfügbar sind. Die Eigenschaft "Klasse", auf die OF DEFINITION REFERENCED IN verweist, muss ebenfalls in der Objekt-definition angegeben sein (siehe obiges Beispiel). (Darüber hinaus wird, wie ebenfalls im Beispiel gezeigt, für das Attribut selbst angegeben, dass darauf mit dem zugehörigen Paket, dem Klassennamen und dem eigenen Attributnamen über die Anweisung KEYED BY {"Package", "Class Name":"Class", Name} verwiesen wird.)

Beispiel 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}
..}
```

OF DEFINITION REFERENCED IN (Forsetzung)

Im obigen Beispiel wird die Definition einer Nachrichtenzeile so eingegrenzt, dass nur die Methoden der Klasse des Objekts aufgeführt werden, in das die Nachrichtenzeile gezogen wird. Dies sind die Methoden von "Zielklasse". Dies ist möglich, da das Objektsymbol (die Objektlebenslinie), in das die Nachrichtenzeile gezogen wird, Eigenschaften für die Verweisklasse enthält, und da die Nachrichtenzeile die Eigenschaft "Zielklasse" enthält. Beachten Sie, dass es sich hierbei um eine Definition für ein OMT-Ablaufdiagramm handelt; das UML-Ablaufdiagramm weist zusätzliche Schlüssel (je nach Paket) auf, die im Beispiel nicht gezeigt werden. Ein Beispiel für die Definition einer UML-Nachrichtenzeile finden Sie beim Schlüsselwort OF DEFINITION AND SUPERS REFERENCED IN.

Siehe auch das Schlüsselwort OF DEFINITION AND SUPERS REFERENCED IN.

OF DEFINITION AND SUPERS REFERENCED IN

Hiermit können Sie eine eingeschränkte Liste mit Definitionen angeben, aus denen Sie beim Klicken auf die Schaltfläche "Optionen" für eine Eigenschaft auswählen können. Diese eingeschränkte Liste verweist auf Elemente einer bestimmten Definition **und** Elemente aller anderen Definitionen, aus denen Werte übernommen werden (sind über eine Vererbungslinie angehängt). Das Schlüsselwort wird verwendet, um eine EDIT LISTOF- oder EDIT ONEOF-Anweisung zu verbessern und wird bei der UML-Modellierung eingesetzt.

Beispiel:

```
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" }
```

Wenn Sie im obigen Beispiel in der Nachrichtendefinition auf "Optionen" klicken, erhalten Sie Methoden der Klasse, an die die Nachrichtenzeile angehängt ist (die "Zielklasse", nicht die "Quellenklasse"), sowie alle Methoden jeder Klasse, die eine Superklasse dieser Klasse darstellt (und über eine Vererbungslinie mit dieser Klasse verbunden ist).

Siehe auch das Schlüsselwort OF DEFINITION REFERENCED IN.

ONEOF

Hierbei handelt es sich um einen der zulässigen Typen für eine Eigenschaft. Das Schlüsselwort wird zusammen mit dem Schlüsselwort EDIT verwendet, um anzugeben, dass die Eigenschaft auf **eine der** Definitionen eines anderen Definitionstyps verweist.

Beispiel:

```
SYMBOL "Relation" IN "Entity Relation"
{
PROPERTY "From Entity" { EDIT ONEOF Entity READONLY }
PROPERTY "To Entity"  { EDIT ONEOF Entity READONLY }
}
```

Im obigen Beispiel enthält die Beziehungslinie zwischen zwei Entitäten auf der zugehörigen Registerkarte "Symbol" die Entitäten, mit denen sie verbunden ist, und zwar sowohl die Entität, zu der die Linie gezogen wird, als auch die, von der aus die Linie gezogen wird. In jedem Fall wird nur eine Entität aufgeführt. Diese Informationen werden automatisch bereitgestellt (Rational System Architect protokolliert Quell- und Zieldaten), daher ist die Eigenschaft SCHREIBGESCHÜTZT.

Beispiel 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}
}
```

Im obigen Beispiel enthält die Definition hinter der Zeile "Erweiterungen" ("Extends") einen Verweis auf den Anwendungsfallschritt (im Referenzanwendungsfall), in dem die Erweiterung (zum anderen Anwendungsfall, mit dem die Zeile verbunden ist) stattfindet. Wenn Sie in der Definition "Erweiterungen" für diese Eigenschaft auf "Optionen" klicken, erhalten Sie eine Liste mit allen Anwendungsfallschritten. Es steht jedoch nur ausreichend Platz im Eigenschaftsfeld zur Verfügung, um einen Anwendungsfallschritt in das Feld zu ziehen (im Gegensatz zu LISTOF, bei dem mehrere Definitionen in das Feld gezogen werden können).

Siehe auch die Schlüsselwörter LISTOF und EXPRESSIONOF.

OVER

Hierbei handelt es sich um ein Argument des Befehls **ALIGN**, mit dem der Name der Eigenschaft (oder ihre Beschriftung) über dem Steuerelement der Eigenschaft platziert wird (beispielsweise ein Textfeld, ein Dropdown-Listenfeld etc.).

Beispiel:

```
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"}
}
```

Im obigen Beispiel sind alle Eigenschaften der Registerkarte (Schlüsselwort CHAPTER) so dargestellt, dass der Name oder die Beschriftung jeder Eigenschaft über dem Steuerelement platziert wird. Die Eigenschaft "Zahl" ("Number") wird in "Rangfolge" ("Ranking") umbenannt. Die entsprechende Beschriftung wird über dem zugehörigen Steuerelement platziert, bei dem es sich um ein einfaches numerisches Feld handelt. Bei der Eigenschaft "Stellenwert" ("Importance") wird der Name über dem zugehörigen Steuerelement platziert, das eine Dropdown-Liste darstellt.

Im Gegensatz hierzu wird beim Schlüsselwort BODY der Name bzw. die Beschriftung der Eigenschaft links neben dem Steuerelement platziert.

Siehe auch die Schlüsselwörter BODY, TAB, ALIGN, LABEL und JUSTIFY.

OVERRIDABLE

Mit diesem Schlüsselwort kann eine schreibgeschützte Eigenschaft einer Definition, die aus dem Diagramm übernommen wurde, bei der ursprünglichen Erstellung geändert (oder beschrieben) werden, obwohl sie eigentlich schreibgeschützt ist. OVERRIDABLE wird **nur** auf der Diagrammebene verwendet, um anzugeben, dass eine Eigenschaft, die zu einer Definition gehört, die wiederum ein im Diagramm gezeichnetes Symbol darstellt, bei der ersten Erstellung des Symbols geändert werden kann, auch wenn dieses schreibgeschützt ist.

Beispiel:

```
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 }
..}
```

Im obigen Beispiel übernimmt die Definition "XML-Element" ihren Wert für die Eigenschaft "XML-Schema" aus dem Symbol "XML-Element", das sie definiert. Dieses wiederum übernimmt den Wert aus der zugehörigen Eigenschaft "XML-Schema" aus dem Diagramm, in dem es abgelegt ist. Wenn Sie das Symbol "XML-Element" zu Beginn unten im Diagrammarbeitsbereich ablegen, können Sie den Wert der Eigenschaft "XML-Schema" der Definition "XML-Element" ändern. Sobald Sie auf "OK" klicken, um die Definition zu schließen, und diese anschließend erneut öffnen, werden Sie feststellen, dass die Eigenschaft "XML-Schema" schreibgeschützt ist und nicht mehr geändert werden kann.

Beachten Sie, dass das Schlüsselwort in der Diagrammspezifikation, nicht in der Definitionsspezifikation für "XML-Element" verwendet wird.

PACK

Steuert die vertikale Positionierung innerhalb des **LAYOUT**-Befehls. Mit diesem Befehl werden Gruppen von Steuerelementen und Beschriftungen in mehreren Spalten von der Gruppe aus Steuerelementen/Beschriftungen, die sich direkt rechts daneben befindet, durch den mindestens benötigten Platz getrennt.

Beispiel:

```
GROUP "Power Builder Headings/Labels" {
   LAYOUT { COLS 2 ALIGN OVER PACK }
```

Siehe auch die Schlüsselwörter LAYOUT und TAB.

PARENT

In der objekt- und methodenorientierten Terminologie wird hiermit das Objekt bezeichnet, von dem das aktuelle Objekt alle Schlüsseleigenschaften übernimmt. In der Anweisung muss 'RELATE BY "is keyed by"' enthalten sein.

Beispiel:

```
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

Dieses Schlüsselwort wird nur in bestimmten Fällen in SAPROPS.CFG verwendet und **darf nicht** in USRPROPS.TXT verwendet werden. Mit diesem Schlüsselwort wird angegeben, dass eine Verweiseigenschaft in der Rational System Architect-Syntax wie eine übliche UML-Operation angezeigt wird. Das heißt, eine qualifizierte Verweiseigenschaft wird mit runden Klammern um den qualifizierten Teil des Schlüssels angezeigt, anstatt wie sonst in Rational System Architect üblich mit Anführungszeichen. Außerdem ist die Reihenfolge der Schlüssel geändert, sodass der Name des Verweisobjekts als Erstes angezeigt wird. Methoden werden beispielsweise als **meth(int, char)** anstelle von **"int, char".meth** angezeigt.

Beispiel:

```
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

Dieses Schlüsselwort wird nur in bestimmten Fällen in SAPROPS.CFG verwendet und **darf nicht** in USRPROPS.TXT verwendet werden. Es ist identisch mit PARMONEOF, wird jedoch auf die Verweisliste von Eigenschaften, beispielsweise ein Methodenraster, angewendet. Mit diesem Schlüsselwort wird angegeben, dass eine Verweiseigenschaft in der Rational System Architect-Syntax wie eine übliche UML-Operation angezeigt wird. Das heißt, eine qualifizierte Verweiseigenschaft wird mit runden Klammern um den qualifizierten Teil des Schlüssels angezeigt, anstatt wie sonst in Rational System Architect üblich mit Anführungszeichen. Außerdem ist die Reihenfolge der Schlüssel geändert, sodass der Name des Verweisobjekts als Erstes angezeigt wird. Methoden werden beispielsweise als **meth(int, char)** anstelle von **"int, char".meth** angezeigt.

Beispiel:

```
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

Mit diesem Befehl wird die exakte Position der Eigenschaften in einem Dialogfenster DIAGRAMM, SYMBOL oder DEFINITION angegeben. Der PLACEMENT-Befehl weist folgende Parameter auf:

LABELPOS (x, y) – gibt den Ausgangspunkt der linken oberen Ecke eines Eigenschaftsnamens (oder der Beschriftung) an. Mit dem x wird die horizontale Position (von der linken Ecke der Dialogfensters) und mit dem y die vertikale Position (vom oberen Rand des Dialogfensters) angegeben. Sowohl x als auch y sind Windows-Einheiten.

PROPPOS (x, y) – gibt den Ausgangspunkt der linken oberen Ecke eines Steuerelements einer Eigenschaft in einem Dialogfenster an. Mit dem x wird die horizontale Position (von der linken Ecke der Dialogfensters) und mit dem y die vertikale Position (vom oberen Rand des Dialogfensters) angegeben. Sowohl x als auch y sind Windows-Einheiten.

PROPSIZE (x, y) – gibt die rechtwinklige Größe des Steuerelements an. Mit x wird die Breite und mit y die Höhe des Steuerelements in Windows-Einheiten angegeben.

Beispiel:

```
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)} }
..
}
```

Im obigen Beispiel setzt der Befehl PLACEMENT den LAYOUT-Befehl für die Registerkarte (CHAPTER) außer Kraft. Das Textfeld der Eigenschaft "Tabellenname" ("Table Name") wird im Definitionsdialogfenster "Klasse" ("Class") auf der Registerkarte "Entitätsinformationen" ("Entity Information") an einer Stelle platziert, die sich 4 Windows-Einheiten vom linken Rand und 24 Windows-Einheiten vom oberen Rand des Dialogfensters befindet. Das Textfeld ist 150 Einheiten breit und 12 Einheiten hoch.

PLACEMENT (Fortsetzung)

Wichtig: Allgemeine Informationen zur Platzierung und Tipps zur Dimensionierung finden Sie in Kapitel 2 dieses Handbuchs.

Siehe auch die Schlüsselwörter PROPPOS, PROPSIZE und LABELPOS.

PROPERTY

Mit diesem Schlüsselwort beginnt das Argument, das ein Merkmal eines Diagramms, eines Symbols oder einer Definition erstellt. Nach dem Schlüsselwort PROPERTY muss der Name der Eigenschaft in Anführungszeichen gesetzt werden. Anschließend müssen Sie die Merkmale der Eigenschaft entweder zwischen geschweifte Klammern oder zwischen die Anweisungen BEGIN und END setzen.

Syntax:

PROPERTY "<property_name>"
{ EDIT <edit_type> <property_parameter> **}**

Oder

PROPERTY "<property_name>"
BEGIN EDIT <edit_type> <property_parameter>
END

PROPPOS, PROPSIZE

Hierbei handelt es sich um ein Parameterpaar des PLACEMENT-Befehls, mit dem Sie die exakte Position von Eigenschaften in einem Dialogfenster DIAGRAMM, SYMBOL oder DEFINITION angeben können. Der Befehl PROPPOS weist zwei Argumente auf – die horizontale Position (vom oberen Rand des Dialogfensters) in Windows-Einheiten und die vertikale Position (vom linken Rand des Dialogfensters) in Windows-Einheiten. Der PROPSIZE-Befehl weist ebenfalls zwei Argumente auf, nämlich x und y, mit denen die Breite und Höhe des Steuerelements der Eigenschaft in Windows-Einheiten angegeben wird.

Syntax:

PLACEMENT { **PROPPOS (horizontal-positioning, vertical-positioning) PROPSIZE (width, height)** }

Beispiel:

```
DEFINITION "My Definition"
{
PROPERTY "Table Name" { EDIT Text LENGTH 31
PLACEMENT {PROPPOS (4, 24) PROPSIZE(150, 12)} }
}
```

Im obigen Beispiel wird der Anfang der Eigenschaft "Tabellenname" ("Table Name") (die linke untere Ecke des Textfeldes) 4 Windows-Einheiten vom linken Rand und 24 Windows-Einheiten vom oberen Rand des Definitionsdialogfensters platziert. Das Textfeld ist ebenfalls 150 Windows-Einheiten breit und 12 Windows-Einheiten hoch. Mit dieser Anweisung werden keine Angaben über den Namen (oder die Beschriftung, in diesem Fall "Tabellenname") gemacht, der zu diesem Textfeld gehört. Da keine Angaben hierzu gemacht werden, wird die Beschriftung standardmäßig links neben dem Textfeld platziert. Sie können die Positionierung der Beschriftung mit dem FORMAT-Befehl oder dem Befehl PLACEMENT {LABELPOS} ändern.

Wichtig: Allgemeine Informationen zur Platzierung und Tipps zur Dimensionierung finden Sie in Kapitel 2 dieses Handbuchs.

Siehe auch die Schlüsselwörter PLACEMENT, LABELPOS und FORMAT.

PUBLISHER

Mit diesem Schlüsselwort können Sie angeben, ob die Werte einer Eigenschaft in der Ausgabe des **SA Information Publisher** veröffentlicht werden sollen. Es weist zwei Argumente auf - **PUBLISHER SHOW** und **PUBLISHER ORDER**.

Syntax:

PROPERTY "Some user property" {
PUBLISHER
{
SHOW (YES|NO) ‘ default is YES
ORDER nnnn ‘ default is zero (do not sort)}
}

PUBLISHER ORDER

Mit diesem Argument des **PUBLISHER**-Befehls können Sie die Reihenfolge angeben, in der die Werte der Eigenschaft in der veröffentlichten Ausgabe von **SA/Publisher** angezeigt werden. Das Argument wird zusammen mit dem Argument **PUBLISHER SHOW** verwendet.

Syntax:

PROPERTY "Some user property" { ... **PUBLISHER** {SHOW YES|NO **ORDER** nnnn } ...}

Standardwert ist Null (nicht sortieren).

Beispiel:

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

Mit diesem Argument des **PUBLISHER**-Befehls können Sie angeben, ob die Werte einer Eigenschaft in der Ausgabe von **SA/Publisher** veröffentlicht werden sollen. Sie können auch den Befehl **PUBLISHER ORDER** mit diesem Schlüsselwort verwenden.

Syntax:

PROPERTY "Some user property" {.....PUBLISHER {SHOW YES|NO} ... }

Der Standardwert ist YES.

Beispiel:

DEFINITION "Business Requirement"
{
PROPERTY "Benefit" { EDIT Text LENGTH 50 **PUBLISHER** {**SHOW NO**} }
}

Im obigen Beispiel wird die Eigenschaft "Vorteil" ("Benefit") nicht auf einer Website angezeigt, die mit SA/Publisher erzeugt wurde, selbst wenn diese Eigenschaft für die Ausgabe in einem Bericht markiert wurde.

QUALIFIABLE

QUALIFIABLE wird in einer Verweiseigenschaft verwendet, in der eine oder mehrere Schlüsselkomponenten der Verweisobjekte nicht aus anderen Eigenschaften im Verweisobjekt übernommen werden dürfen, jedoch möglicherweise in der Eigenschaft selbst bereitgestellt werden können. Das Schlüsselwort wird verwendet, wenn alle Schlüsseldaten nicht innerhalb der Eigenschaften einer Referenzdefinition gespeichert werden können, aber der Name der Referenzdefinition durch die Schlüsseleigenschaft qualifiziert werden muss.

Ein Beispiel ist folgende KEYED BY-Klausel:

KEYED BY {key_component-1: property_name_1, name}

Diese Klausel gibt an, dass der Wert für "key_component_1" aus "property_name_1" übernommen werden soll. Daher würde die Verweiseigenschaft nur den Namen des Verweisobjekts enthalten. Ein weiteres Beispiel ist folgende KEYED BY-Klausel:

KEYED BY {key_component-1 QUALIFIABLE, name}

Diese Klausel gibt an, dass der Wert für "key_component_1" aus dieser Eigenschaft übernommen werden soll, d. h. der Eigenschaft mit dieser KEYED BY-Klausel. Daher kann die Verweiseigenschaft die Werte sowohl der Namen als auch der Schlüsselkomponente 1 ("key-component-1") der Verweisobjekte enthalten. Unter diesen Bedingungen werden die Werte der Namen von den Werten der Schlüsselkomponente 1 durch Punkte getrennt.

Beispiel:

```
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

Gibt an, dass eine Eigenschaft zwar gelesen, jedoch nicht geändert werden kann. READONLY wird in SAPROPS für Eigenschaften verwendet, deren Wert von der Software eingefügt wird, jedoch für den Benutzer sichtbar sein soll. Dieses Schlüsselwort wird stets bei "Initial AuditId" - "Datum und Uhrzeit" sowie bei "Update AuditID" - "Datum und Uhrzeit" verwendet. Beziehungslinien, Bedingungen und andere Linien, die Symbole verknüpfen, bei denen die Quell- und Zielsymbole der Linie signifikant sind, sind immer SCHREIBGESCHÜTZT.

Beispiel:

```
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

Eine Eigenschaft kann nur einmal in einer Definition verwendet werden (es sei denn, sie wird zwischen #IFDEF gesetzt). Wenn der Benutzer dieselbe Eigenschaft mehrmals in einer Definition verwenden möchte, muss er die Schlüsselwörter CONTROL und REFPROP verwenden. Daher werden die Schlüsselwörter CONTROL und REFPROP häufig in Verbindung mit TESTPROC verwendet.

Damit ein CONTROL-Schlüsselwort verwendet wird, muss ein Ausgangsverweis auf die Eigenschaft vorhanden sein, auf die CONTROL verweist, und zwar an erster Stelle der Definition. Das Schlüsselwort REFPROP wird in Verbindung mit dem Schlüsselwort CONTROL verwendet.

Beispiel:

```
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"}
  ...
  }
```

Siehe auch das Schlüsselwort CONTROL.

RELATE (BY), RELATED (BY)

Der Standardbeziehungstyp für eine Verweiseigenschaft lautet "Verwendet". Mit dem Schlüsselwort RELATE BY können Sie diesen Standardwert durch eine andere Beziehung (z. B. "Mit Schlüssel") oder keine Beziehung (dann muss "RELATE BY nichts" codiert werden) ersetzen.

Folgende Beziehungen können zusammen mit dem Schlüsselwort RELATE BY verwendet werden:

Nichts – keine Beziehung.

Verwendet – Standardeinstellung. Bedeutet, dass die Definition eine Definition enthält.

Erläutert durch – Bedeutet, dass ein Symbol durch eine Definition erläutert wird.

Definiert durch – Bedeutet, dass ein Symbol durch eine Definition definiert wird.

Ist ein – Bedeutet, dass eine Definition eine Instanz einer Definition ist (z. B. ist eine Spalte ein Datenelement).

Bestimmt – Bedeutet, dass ein Objekt ein anderes bestimmt.

Umfasst – Bedeutet, dass ein Objekt mehrere Objekte umfasst (z. B. umfasst ein Modell Entitäten, Beziehungen usw. Eine Kategorisierung umfasst Kategoriebeziehungen).

Stammt aus – Bedeutet, dass ein Objekt aus einer Definition stammt.

Basiert auf – Bedeutet, dass ein Objekt auf einer Definition (in der Regel einem Datenelement) basiert.

Ist Teil von – Bedeutet, dass eine Definition Teil einer Definition ist. Wird mit "OneOf" oder "ListOf" verwendet.

Ist Schlüssel von – Bedeutet, dass eine Definition von einer Definition bestimmt wird.

Benutzerdefinierte Beziehungen – Es gibt zudem 20 benutzerdefinierte Beziehungstypen (USER 1 bis USER 20), die zur Verfügung stehen, wenn sie der Benutzer über einen RENAME-Befehl erstellt, z. B. RENAME RELATION "USER 1" to "XXXX".

RELATE (BY), RELATED (BY) (Fortsetzung)

Beispiel:

```
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}}
```

Im obigen Beispiel wird mit dem ersten KEY EDIT angegeben, dass die Eigenschaft "Anwendungsfallname" ("Use Case Name") eine Schlüsseleigenschaft der Definition "Anwendungsfallschritt" ("Use Case Step") ist. Diese Eigenschaft "Anwendungsfallname" verweist auf eine Definition "Anwendungsfall" - dies wird mit ONEOF "Anwendungsfall" angegeben. Sie müssen den vollständigen Schlüssel dieses Anwendungsfalls angeben (auf den der Schritt verwiesen wird). In diesem Fall verwenden Sie den Befehl KEYED BY, um die Schlüssel anzugeben, die das Paket darstellen, in dem der Anwendungsfall enthalten ist, sowie den Namen des Anwendungsfalls selbst. Schließlich müssen Sie angeben, dass mit diesem Anwendungsfall auf den Anwendungsfallschritt verwiesen wird. Dieser Vorgang erfolgt durch den Befehl RELATE BY "ist Schlüssel von". Die RELATE BY-Klausel wird hinzugefügt, da der Standardbeziehungstyp für eine Verweiseigenschaft "Verwendet" lautet. Wenn ein anderer Beziehungstyp gewünscht wird (z. B. "ist Schlüssel von"), muss die Standardeinstellung außer Kraft gesetzt werden.

Mit dem zweiten Schlüsselwort KEY EDIT im obigen Beispiel wird angegeben, dass die Eigenschaft "Paket" ebenfalls ein Schlüssel der Definition "Anwendungsfallschritt" ist - d. h. auf den Anwendungsfallschritt wird mit einer Paketdefinition verwiesen. Beachten Sie jedoch, dass auf eine Paketdefinition selbst keine zusätzlichen Eigenschaften außer dem eigenen Namen verweisen, daher wird der Befehl KEYED BY nicht verwendet. Das Paket bezieht sich über die Beziehung "ist Schlüssel von" auf einen Anwendungsfallschritt.

RELATION

Die aktive Beziehung zwischen einer Eigenschaft und den zugehörigen Verweisen.

Siehe auch die Schlüsselwörter RELATE [BY], RELATED [BY].

REM, REMARK

Text, der nach diesem Befehl eingegeben und in einfache oder doppelte Anführungszeichen gesetzt wird, wird ignoriert.

Beispiel:

```
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

Dieser Befehl ermöglicht Verweise auf ein Objekt über einen anderen Namen als denjenigen, der normalerweise von Rational System Architect verwendet wird.

Beispiel:

```
RENAME SYMBOL "Control Transform"
  IN "Data Flow Ward & Mellor" TO "Process"
RENAME DIAGRAM "Data Flow Ward & Mellor"
  TO "Ward Mellor"
```

RENAME DEFINITION

Dieses Schlüsselwort ermöglicht die Verwendung von 150 vom Benutzer bereitgestellten Definitionen. Diese Definitionen heißen "Benutzer 1" bis "Benutzer 150". Mit dem Befehl RENAME DEFINITION können Sie eine beliebige Anzahl dieser vom Benutzer bereitgestellten Definitionen umbenennen und somit neue Definitionstypen erstellen. RENAME DEFINITION-Anweisungen müssen relativ weit oben in der Datei USRPROPR.TXT aufgeführt werden.

Beispiel:

```
Rename Definition "User 10" to "System Requirement "
```

RENAME DIAGRAM

Dieses Schlüsselwort ermöglicht die Verwendung von 20 vom Benutzer bereitgestellten Diagrammen. Diese Diagramme heißen "Benutzer 1" bis "Benutzer 20". Mit dem Befehl RENAME DIAGRAM können Sie eine beliebige Anzahl dieser vom Benutzer bereitgestellten Diagramme umbenennen und somit neue Diagrammtypen erstellen. RENAME DIAGRAM-Anweisungen müssen relativ weit oben in der Datei USRPROPR.TXT aufgeführt werden.

Beispiel:

Rename Diagram "User 10" to "Requirements Hierarchy"

RENAME SYMBOL

Dieses Schlüsselwort ermöglicht die Verwendung von 150 Benutzersymbolen. Diese vom Benutzer bereitgestellten Symbole heißen "Benutzer 1" bis "Benutzer 150". Mit dem Befehl RENAME SYMBOL können Sie eine beliebige Anzahl dieser vom Benutzer bereitgestellten Symbole umbenennen und somit neue Symboltypen erstellen. RENAME SYMBOL-Anweisungen müssen relativ weit oben in der Datei USRPROPS.TXT aufgeführt werden.

Beispiel:

Rename Symbol "User 10" to "System Requirement"

REQUIRED

Mit diesem Schlüsselwort wird angegeben, dass eine Eigenschaft vom Benutzer angegeben werden muss, damit das Diagramm oder die Definition erstellt werden kann. Die Eigenschaft wird automatisch im ersten Dialogfenster **Name** für das Diagramm oder die Definition angezeigt.

Beispiel:

Bei diesem Beispiel muss der Benutzer einen Wert für die Eigenschaft "XML-Schema" eingeben, damit die Definition "XML-Elemententität" erstellt werden kann.

```
DEFINITION "XML Element Entity"
{
PROPERTY "XML Schema" { Key Edit ONEOF "XML Schema"
Relate By "is keyed by" Required
Readonly }
}
```

RETAIN STYLE

Mit diesem Schlüsselwort wird angegeben, dass die vom Benutzer bereitgestellten Metadateien bei der Verwendung im Tool ihr ursprüngliches grafisches Format und ihre Farbgebung beibehalten. Dieses Schlüsselwort wird zusammen mit der DEPICTIONS-Klausel verwendet.

Wenn Sie externe, vom Benutzer bereitgestellte Bilder verwenden, um Symbole in einem Diagramm darzustellen, bestimmt das Standardverhalten, dass Sie die Merkmale dieser Symbole, wie z. B. die Füllfarbe oder die Linienfarbe, genauso wie bei den anderen Symbolen in Rational System Architect angeben können. Wenn Sie das Schlüsselwort RETAIN STYLE in der DEPICTIONS-Klausel angeben, bleiben die Farben des benutzerdefinierten Symbols unverändert und können auch nicht geändert werden.

Beispiel:

```
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 }
```

Im obigen Beispiel kann "firewall.wmf" für die Abbildung eines Knotensymbols in einem Implementierungsdiagramm verwendet werden, wenn das Stereotyp des Knotens auf "Firewall" gesetzt ist. Beim Zeichnen im Diagramm wird die vom Benutzer bereitgestellt Metadatei "firewall.wmf" (die vom Benutzer zur Dateitabelle der Enzyklopädiedatenbank hinzugefügt wurde) mit genau denselben Farben gezeichnet, wie sie außerhalb von Rational System Architect verwendet werden. Diese können mit den Farbtools von Rational System Architect nicht geändert werden.

SACPropertyOnOfBase	Dieses Schlüsselwort wird im Befehl EDITCLASS SACPropertyOneOfBase verwendet. **Verwenden Sie diese Schlüsselwortkombination nicht.** Diese Kombination wurde speziell für bestimmte Situationen in Rational System Architect entwickelt, nämlich die Übernahme von Datenelementeigenschaften durch ein Attribut in einer Entität. Diese Schlüsselwortkombination wird in der Datei SAPROPS.CFG für diese Situation verwendet. Diese Schlüsselwortkombination kann nur in dieser Situation angewendet werden. Eine Verwendung in anderen Situationen kann Fehler verursachen.

SCRIPT

Hiermit wird ein Script aufgerufen, das in SA Basic geschrieben wurde. Das Script wird für Eigenschaften verwendet, die weder vom Typ "ListOf" noch vom Typ "ExpressionOf" sind. Ein Script übernimmt den Wert einer Eigenschaft und wandelt ihn in eine Aktion um, üblicherweise um einen bestimmten Anmerkungstyp zu einem Symbol in einem Diagramm anzuzeigen. Die Namenskonvention für das Script lautet folgendermaßen:

- _fmt (z. B. _fmtUMLAttr): Die Funktion selbst ist nur fest codiert verfügbar und kann nicht geändert werden. Die meisten Funktionen in SAPROPS.CFG sind fest codiert. Dadurch wird die Antwort von Rational System Architect insgesamt schneller.
- fmt (z. B. fmtUMLAttr): Ist in der Datei "fmtscript.bas" im ausführbaren Hauptverzeichnis von Rational System Architect vorhanden.

Erstellen eigener Funktionen

Sie können eigene Funktionen erstellen, um Elemente auf bestimmte Weise in einem Symbol anzueigen oder um einen bestimmten Wert zu berechnen. Die von Ihnen erstellen Funktionen ***dürfen nicht*** in "fmtscript.bas" gespeichert werden, da diese Datei bei jeder Neuinstallation oder Aktualisierung von Rational System Architect überschrieben wird. Wenn Sie eigene Funktionen erstellen, müssen Sie diese in einer Datei namens **usr_fn.bas** speichern, die Sie erstellen (wird nicht standardmäßig bereitgestellt) und im Hauptverzeichnis von Rational System Architect speichern müssen (<C>:\Programme\IBM\Rational\11.3.1\System Architect Suite\System Architect). (Die Datei "sarules.bas" verfügt über ein "#include" zur Datei "usr_fn.bas".)

Die meisten in SAPROPS.CFG aufgerufenen Funktionen (die fest codiert sind und deren Name üblicherweise mit einem Unterstrich beginnt, z. B. _fmtUMLAttr) verfügen über einen äquivalenten Funktionsaufruf in der Datei "fmtscript.bas" (ohne Unterstrich). Wenn Sie eine eigene Funktion erstellen möchten, können Sie die Scripts in der Datei "fmtscript.bas" als Leitfaden verwenden.

SCRIPT (Fortsetzung)

Erläuterung bereits vorhandener Funktionen:

FmtOMTAbstractClass gibt das Script **{abstract}** zurück, wenn diese Eigenschaft festgelegt wurde; anderenfalls wird nichts zurückgegeben.

FmtBOOClassConstraint gibt den Namen der Bedingung in geschweiften Klammern zurück, d. h. **{*constraint name*}**, wenn ein solcher festgelegt wurde; anderenfalls wird nichts zurückgegeben.

FmtOMTObjInstClass gibt den Namen der Klasse in runden Klammern zurück, d. h. **(*class name*)**, wenn ein solcher festgelegt wurde; anderenfalls wird nichts zurückgegeben.

FmtEntryAction gibt das Script **entry /** und den Namen der Eintragsaktion zurück, wenn ein solcher festgelegt wurde; anderenfalls wird nichts zurückgegeben.

FmtExitAction gibt das Script **exit /** und den Namen der Exitaktion zurück, wenn ein solcher festgelegt wurde; anderenfalls wird nichts zurückgegeben.

FmtOMTTransition gibt die folgenden Werte innerhalb der folgenden Interpunktionszeichen zurück, wenn diese in einer Statusübergangsdefinition festgelegt wurden: **(*Attributname*) [*Bedingung*] /*Aktion***

Beispiele:

```
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"
```

Siehe auch die Schlüsselwörter FORMAT, COLUMN_SCRIPT, COMPONENT_SCRIPT und fmtxxx.

STRING

Hierbei handelt es sich um das Standardargument, das nach dem Schlüsselwort FORMAT im **Display**-Befehl verwendet wird. Bei Verwendung von *STRING* wird der Inhalt des Verzeichniseintrags in der Anzeige genau so dargestellt, wie er eingegeben wurde.

Beispiel:

```
SYMBOL "Relation" IN "Shlaer Information Model"
{
PROPERTY "Description"  { EDIT Text LENGTH 100 }
PROPERTY "Reverse Phrase"  { EDIT Text LENGTH 65 DISPLAY {
FORMAT STRING LEGEND "$$NONE$$" } }
..}
```

Siehe auch die Schlüsselwörter FORMAT und DISPLAY.

SUPERS

Siehe auch das Schlüsselwort OF DEFINITION AND SUPERS REFERENCED IN.

SYMBOL

Dieses Schlüsselwort ist das erste Wort in dem Block, in dem die Eigenschaften eines Symbols im Gegensatz zu einer DEFINITION oder einem DIAGRAMM aufgeführt sind.

Beispiel:

```
SYMBOL "Entity" IN "Entity Relation"
{
PROPERTY "Description"  { EDIT Text LENGTH 500 }
…
}
```

Siehe auch die Schlüsselwörter DIAGRAM und DEFINITION.

TAB

Mit diesem Schlüsselwort wird die vertikale Positionierung im **LAYOUT**-Befehl gesteuert, indem die Gruppen von Steuerelementen und Beschriftungen in mehreren Spalten durch Tabulatoren so getrennt werden, dass die Einträge in jeder Zeile direkt unterhalb der Einträge in der Zeile darüber aufgelistet werden.

Beispiel:

```
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 "
```

Siehe auch die Schlüsselwörter LAYOUT und PACK.

Befehlsgruppe TESTPROC, TESTPROPERTY, TESTSTRING

Die Befehlsgruppe TESTPROC, TESTPROPERTY und TESTSTRING stellt eine bedingte Funktionalität für Eigenschaften auf einer diagrammweisen Basis bereit. Die Funktionen dieser Befehlsgruppe sind mit denen von #IFDEF vergleichbar, nur dass die bedingte Funktionalität von #IFDEF auf der Ebene einer Enzyklopädie basiert. Die Befehlsgruppe TESTPROC wird aus einer **Diagrammeigenschaft** ausgearbeitet, d. h. eine Definition enthält eine bestimmte Eigenschaftengruppe, wenn ein Wert für eine "Testeigenschaft" innerhalb der Eigenschaften des Diagramms ausgewählt wurde.

Die TESTPROC-Befehlsgruppe wird insbesondere in logischen Datenmodellen verwendet, um die Eigenschaftengruppen für die Datenmodellierung abhängig vom ausgewählten RDBMS (Managementsystem für relationale Datenbanken) anzugeben.

TESTPROC steht für Testverfahren. Es gibt zwei Werte, die nach einem TESTPROC-Schlüsselwort stehen können: **TestPropertyValue** und **TestPropertyNotValue**. Wenn nach TESTPROC der Wert "TestPropertyValue" folgt, bedeutet dies Folgendes: "Teste die Eigenschaft; wenn sie mit einem der in TESTSTRING festgelegten Werte übereinstimmt, wende die Eigenschaften in diesem TESTPROC-Abschnitt auf die entsprechende Definition an". Wenn nach TESTPROC der Wert "TestPropertyNotValue" folgt, bedeutet dies Folgendes: "Teste die Eigenschaft; wenn sie **nicht** mit einem der in TESTSTRING festgelegten Werte übereinstimmt, wende die Eigenschaften in diesem TESTPROC-Abschnitt auf die entsprechende Definition an". Die Groß-/Kleinschreibung muss bei Verwendung von **TestPropertyValue** und **TestPropertyNotValue** beachtet werden, d. h. die Groß-/Kleinschreibung muss genau dem vorgegebenen Wert entsprechen. Der Vorgang funktioniert nicht, wenn Sie alles in Klein- oder alles in Großbuchstaben schreiben.

TESTPROPERTY ist die Diagrammeigenschaft, die abgefragt wird.

TESTSTRING sind die abgefragten Werte. Sie können einen oder mehr Werte in der Zeichenfolge auflisten.

CONTROL und REFPROP: Eine Eigenschaft kann nur einmal in einer Definition verwendet werden (es sei denn, sie wird zwischen "#ifdef" gesetzt). Wenn der Benutzer dieselbe

Befehlsgruppe TESTPROC, TESTPROPERTY, TESTSTRING (Fortsetzung)

Eigenschaft mehrmals in einer Definition verwenden möchte, muss er die Schlüsselwörter CONTROL und REFPROP verwenden. Daher werden die Schlüsselwörter CONTROL und REFPROP häufig in Verbindung mit TESTPROC verwendet. Damit ein CONTROL-Schlüsselwort verwendet wird, muss ein Ausgangsverweis auf die Eigenschaft vorhanden sein, auf die CONTROL verweist, und zwar an erster Stelle der Definition.

Beispiel:

```
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

Siehe Befehlsgruppe TESTPROC, TESTPROPERTY, TESTSTRING.

TestPropertyNotValue

Siehe Befehlsgruppe TESTPROC, TESTPROPERTY, TESTSTRING.

TEXT

Hierbei handelt es sich um einen zulässigen Feldtyp. Die als Text definierte Definition kann aus einer Liste stammen, oder es kann sich um beliebige, vom Benutzer eingegebene alphanumerische Zeichen handeln.

Beispiel:

```
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

Hierbei handelt es sich um einen zulässigen Feldtyp, mit dem angegeben wird, dass eine Eigenschaft eine Uhrzeitmarke in der Notation aufweist, die dem in Windows definierten Zeitformat entspricht. CHECKOUT TIME, FREEZE TIME, INITIAL TIME und UPDATE TIME haben jeweils besondere Bedeutungen.

Beispiel:

```
DIAGRAM "Data Flow Gane & Sarson"
{
PROPERTY "Freeze time"
{ FREEZE  TIME }
..
}
```

Andere Verwendungszwecke für TIME liegen möglicherweise in beliebigen Definitionen vor.

Beispiel:

```
DEFINITION "X" {
PROPERTY "Creation Time"
{ EDIT Text INITIAL TIME LENGTH 12 READONLY }
}
```

TO

Wird im Befehl **Rename** verwendet, um den ursprünglichen Namen des Objekts vom neuen Namen zu trennen.

Beispiel:

```
RENAME SYMBOL Class IN "Booch Class"
  TO "Booch Class"
```

UPDATE

Hierbei handelt es sich um einen zulässigen Feldtyp, mit dem angegeben wird, dass das System das Feld beim Ändern der Eigenschaft automatisch aktualisiert. Er wird standardmäßig bei *Audit ID, Update Date* und *Update Time* verwendet.

Das Schlüsselwort UPDATE stellt dieselben Informationen bereit wie das Schlüsselwort LAST CHANGED. Mit beiden Schlüsselwörtern wird der Zeitpunkt der letzten Änderung einer Definition angegeben, d. h. dass jemand ein Definitionsdialogfenster geöffnet hat, eine Änderung vorgenommen hat (die Definition also in gewisser Weise "verunreinigt" hat, z. B. durch Hinzufügen eines Leerzeichens oder Löschen eines Buchstabens in einer der Eigenschaften oder durch Entfernen und erneutes Hinzufügen eines Buchstabens) und anschließend auf die Schaltfläche SAVE geklickt hat, um die Änderung zu speichern. Wenn ein Benutzer ein Definitionsdialogfenster öffnet, jedoch keine Änderungen vornimmt und dann auf SAVE klickt, wird die Definition nicht geändert. (Anmerkung: Wenn eine Definition von einem Benutzer geöffnet wird, wird diese temporär von diesem Benutzer gesperrt. Wenn der Benutzer keine Änderungen vornimmt und die Definition entweder speichert oder den Vorgang abbricht, wurde die Definition nicht geändert, und die Eigenschaften LAST CHANGED oder UPDATE protokollieren diesen Benutzer nicht. In Rational System Architect wird jedoch intern verfolgt, wer eine Definition als Letzter gesperrt hat. Diese Informationen über das letzte Sperren stehen nicht über die USRPROPS-Schlüsselwörter zur Verfügung.)

Ab Rational System Architect V9 werden die Schlüsselwörter LAST CHANGED AUDITID, LAST CHANGED DATE und LAST CHANGED TIME standardmäßig auf der Registerkarte "Zugriffsdaten" jedes Diagrammdialogfensters oder Definitionsdialogfensters zur Verfügung gestellt. Diese Einstellung ist fest im Produkt codiert - d. h. das Schlüsselwort LAST CHANGED ist nicht in jeder Definition in SAPROPS.CFG vorhanden, und Sie müssen es auch nicht für neue Diagramm- oder Definitionstypen, die Sie erstellen, zu USRPROPS.TXT hinzufügen.

Beispiel:

```
DEFINITION "X"
  { PROPERTY "Modified Time"
    { EDIT Text UPDATE TIME LENGTH 12 READONLY }
  }
```

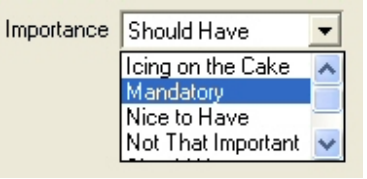
Siehe auch das Schlüsselwort INITIAL.

VALUE

Dieses Schlüsselwort wird einer Wertefolge in einer LISTE vorangestellt.

Beispiel:

```
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" }
}
```



Im obigen Beispiel wird eine neue Liste in USRPROPS.TXT (an erster Stelle der Datei) erstellt. Der Liste werden fünf Werte zugewiesen. Weiter unten in der Datei USRPROPS.TXT, in der Definition eines Anwendungsfallschritts, wird diese Liste in der Eigenschaft "Stellenwert" ("Importance") bereitgestellt. Beachten Sie, dass bei dieser Art von Liste der Benutzer eigene Werte in das Feld "Stellenwert" eingeben kann.

Siehe auch die Schlüsselwörter LIST, LISTONLY und LISTONLYCOMBO.

VALUESCRIPT

Mit VALUESCRIPT wird eine in SA Basic geschriebene Funktion aufgerufen. Zu VALUESCRIPT gehört die Umsetzung von Konsistenzprüfungen für Eigenschaftswerte in einem offenen Dialogfenster. Die Funktionen berechnen in der Regel für Eigenschaften in einem Dialogfenster festgelegte Werte und nehmen die erforderlichen Änderungen an Werten in anderen Eigenschaften im Dialogfenster in Echtzeit vor, damit bestimmte Konsistenzregeln eingehalten werden.

Erstellen eigener Funktionen

Sie können eigene Funktionen erstellen, um Konsistenzprüfungen für Eigenschaftswerte in einem offenen Dialogfenster umzusetzen. Weitere Informationen zum Erstellen eigener Funktionen finden Sie unter dem Schlüsselwort SCRIPT.

Beispiel:

```
DEFINITION "Index"
{
PROPERTY Unique
{ PLACEMENT {PROPPOS(84, 0) PROPSIZE(100, 12)} EDIT Boolean
LENGTH 1 VALUESCRIPT ProcessIndexUnique DEFAULT "F" }
.. }
```

Im obigen Beispiel wird die Funktion "ProcessIndexUnique" aufgerufen. Sie befindet sich in der Datei "fmtscript.bas". Diese Funktion wird nur aufgerufen, wenn im Datenbankmanagementsystem "Oracle" ausgewählt wurde. Mit der Funktion wird überprüft, ob die Eigenschaft "Bitmap" für den Index aktiviert ist. Ist dies der Fall, wird mit der Funktion "ProcessIndexUnique" die Eigenschaft "Index" inaktiviert, falls diese aktiviert war. Dies geschieht, da in Oracle ein Index nicht eindeutig sein kann, wenn er als Bitmap-Index angegeben wurde.

Beispiel:

```
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)} }
..}
```

VALUESCRIPT (Fortsetzung)

Im obigen Beispiel wird mit "ProcessSQLDataType" überprüft, ob das Datenelement den Typ aus einer zugrunde liegenden Datendomäne übernimmt. In diesem Fall wird der Wert automatisch eingetragen. Wenn der Typ nicht übernommen wird und der Benutzer das Typfeld leer lässt, wird von der Funktion automatisch "Zeichen 10" als Standardwert eingetragen, wenn der Benutzer die Eingabetaste betätigt oder Felder im Attributraster ändert.

VFORCE

Ermöglicht es Ihnen, vertikale Linien innerhalb von Symbolen zu zeichnen. Die Standardeinstellung sind horizontale Linien. Mit VFORCE erfolgt das Layout der Eigenschaften von links nach rechts, getrennt durch eine vertikale Linie. (Anmerkung: Der Befehl VNONE hat dieselbe Funktion, zeigt die vertikale Linie jedoch nicht an.)

Syntax:

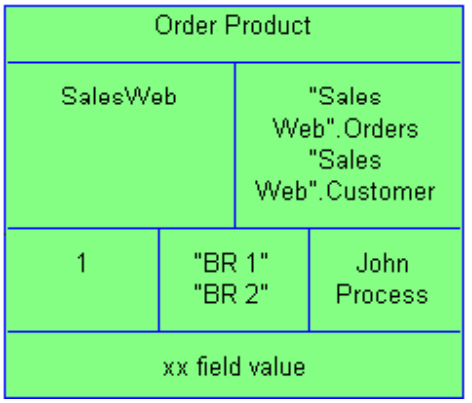
{FORMAT String LEGEND **"$$VFORCE$$"**}

Beispiel:

```
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 "$$VFORCE$$"}}
..}
```

Beachten Sie, dass die erste aufgelistete Eigenschaft nicht VFORCE, sondern nur FORCE enthält. Nachfolgende Eigenschaften, die Sie rechts neben der ersten Eigenschaft aufführen möchten, erhalten die Spezifikation VFORCE. In der nachstehenden Abbildung werden die Werte "Sales Web.Orders" und "Sales Web.Customer" in einem Feld für "Referenzdaten" aufgeführt. Durch den Befehl VFORCE wurde erreicht, dass dieses Feld rechts neben dem Eigenschaftenfeld "Supporting Applications" angezeigt wird, das in der Abbildung den Wert "SalesWeb" enthält.

VFORCE (Fortsetzung)



Beachten Sie außerdem, dass der Befehl VFORCE auch verwendet wurde, damit die Felder mit "BR 1" und "BR 2" sowie "John Process" rechts neben dem Feld mit dem Wert 1 angezeigt werden. Dies wird jedoch im USRPROPS.TXT-Beispiel nicht angezeigt.

VISIBLE

Wenn eine Eigenschaft in SAPROPS.CFG als INVISIBLE bezeichnet wird, kann sie mithilfe des Schlüsselworts VISIBLE im Definitionsdialogfenster sichtbar gemacht werden.

Beispiel (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 }
```

Beispiel (USRPROPS):

```
DEFINITION "Watcom Stored Procedure"
   PROPERTY "Procedure Number"
      { VISIBLE }
```

Siehe auch das Schlüsselwort INVISIBLE.

VNONE

Hierbei handelt es sich eigentlich um das Schlüsselwort $$VNONE$$, das zusammen mit dem Schlüsselwort DISPLAY verwendet wird. Weitere Informationen finden Sie beim Schlüsselwort DISPLAY.

Ermöglicht es Ihnen, vertikale Linien innerhalb von Symbolen zu zeichnen. Die Standardeinstellung sind horizontale Linien. Mit VNONE erfolgt das Layout der Eigenschaften von links nach rechts, getrennt durch eine vertikale Linie, die jedoch nicht angezeigt wird. (Anmerkung: Der Befehl VFORCE hat dieselbe Funktion, zeigt die vertikale Linie jedoch an.)

Syntax:

{FORMAT String LEGEND **"$$VNONE$$"**}

Beispiel:

Beachten Sie, dass im nachstehenden Beispielausschnitt von USRPROPS.TXT für die erste aufgeführte Eigenschaft FORCE angegeben ist. Nachfolgende Eigenschaften, die Sie (ohne Trennlinie) rechts neben der ersten Eigenschaft aufführen möchten, erhalten die Spezifikation VNONE.

Beispiel:

```
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

Zeigt nur die Definitionen im Dialogfenster "Optionen" an, die einen festen Wert in einer benannten Eigenschaft der Definition enthalten.

Beispiel:

```
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"}
}
```

Wenn das obige USRPROPS.TXT-Beispiel auf eine Enzyklopädie angewendet wird, werden folgende Definitionen von Flugzeugtypen (Aircraft Type) in der Enzyklopädie erstellt:

Mustang (engine = Propeller)
Spitfire (engine = Propeller)
F-16 Fighting Falcon (engine = Jet)
F-86 Sabre (engine = Jet)

Wenn Sie dann nach der Erstellung einer neuen Definition des Typs "Filtered Aircraft" (beispielsweise mit der Bezeichnung "Current Jet Fighters") auf die Schaltfläche **Optionen** für diese Definition klicken, werden nur zwei Optionen angezeigt, nämlich "F-16 Fighting Falcon" und "F-86 Sabre"; alle Definitionen, bei denen der Motortyp ("Engine") auf "Propeller" gesetzt wurde, werden in der Liste **Optionen** nicht angezeigt.

ZOOMABLE

Hiermit wird eine Schaltfläche in einem Listenfeld platziert, mit der das Feld erweitert werden kann, um das gesamte Dialogfenster auszufüllen.

Beispiel:

```
PROPERTY "User Roles"
  {ZOOMABLE EDIT ListOf "User Role with Access Rights"
  LENGTH 1500 LABEL "User Role(s)"}
```

Siehe auch das Schlüsselwort LINES.

4

IBM Unterstützungsfunktion

Einführung

Es gibt eine Reihe von Informationsquellen zur Selbsthilfe und Tools, die Ihnen bei der Fehlerbehebung von Nutzen sein können. Wenn bei Ihrem Produkt ein Problem auftritt, haben Sie folgende Möglichkeiten:

Lesen Sie die Versionshinweise zu Ihrem Produkt. Dort finden Sie Informationen zu bekannten Problemen, Problemlösungen und zur Fehlerbehebung.

Überprüfen Sie, ob ein Download oder Fix zum Beheben des Fehlers verfügbar ist.

Durchsuchen Sie die verfügbaren Wissensdatenbanken, um zu überprüfen, ob die Behebung dieses Fehlers bereits dokumentiert ist. Wenn Sie weiterhin Hilfe benötigen, wenden Sie sich an den IBM® Software Support und melden Sie den Fehler.

Themen in diesem Kapitel	Seite

IBM Rational-Software-unterstützung kontaktieren

Wenn Sie mithilfe der Informationsquellen zur Selbsthilfe einen Fehler nicht beheben können, wenden Sie sich an die IBM® Rational®-Softwareunterstützung.

Anmerkung: Wenn Sie ein ehemaliger Telelogic Kunde sind, stehen Ihnen alle Unterstützungsressourcen auf einer zentralen Website zur Verfügung: http://www.ibm.com/software/rational/support/telelogic/

Voraus-setzungen

Wenn Sie ein Problem an die IBM Rational-Softwareunterstützung melden möchten, müssen Sie über einen aktiven Passport Advantage®-Softwarewartungsvertrag verfügen. Passport Advantage ist das umfassende Softwarelizenz- und Softwarewartungsangebot für Produktupgrades und technische Unterstützung von IBM. Sie können sich online unter der folgenden Adresse für Passport Advantage registrieren: http://www.ibm.com/software/lotus/passportadvantage/howtoenroll.html

- Weitere Informationen zu Passport Advantage finden Sie in den FAQs zu Passport Advantage unter http://www.ibm.com/software/lotus/passportadvantage/brochures_faqs_quickguides.html.
- Weitere Unterstützung erhalten Sie von Ihrem IBM Ansprechpartner.

Gehen Sie wie folgt vor, um einen Fehler online (von der IBM Website) an die IBM Rational-Softwareunterstützung zu senden:

- Registrieren Sie sich auf der Website für die IBM Rational-Softwareunterstützung als Benutzer. Details zur Registrierung finden Sie hier: http://www.ibm.com/software/support/.
- Sie müssen im Serviceanforderungstool als "Authorized Caller" (autorisierter Anrufer) aufgeführt sein.

Sonstige Informationen

Produktneuheiten, Veranstaltungshinweise und sonstige Informationen zur Rational-Software finden Sie unter: http://www.ibm.com/software/rational/.

Fehler senden

Gehen Sie wie folgt vor, um einen Fehler an die IBM Rational-Softwareunterstützung zu senden:

1. Bestimmen Sie die Auswirkungen des Fehlers auf die Geschäftsabläufe. Beim Melden eines Problems an IBM werden Sie aufgefordert, einen Schweregrad anzugeben. Dazu müssen Sie den durch das Problem verursachten Einfluss auf die Geschäftsabläufe nachvollziehen und bewerten.

Verwenden Sie zum Bestimmen des Schweregrads die folgende Tabelle.

Schweregrad	Beschreibung
1	Der Fehler ist geschäftskritisch: Sie können das Programm nicht verwenden, was sich auf die Abläufe kritisch auswirkt. Die Situation erfordert eine sofortige Lösung.
2	Der Fehler hat große Auswirkungen auf die Geschäftsabläufe: Das Programm kann verwendet werden, ist jedoch stark eingeschränkt.
3	Der Fehler hat zu einem bestimmten Teil Auswirkungen auf die Geschäftsabläufe: Das Programm kann verwendet werden, aber weniger signifikante (keine vorgangs-kritischen) Funktionen sind nicht verfügbar.
4	Der Fehler hat minimale Auswirkungen auf die Geschäftsabläufe: Der Fehler hat geringe Auswirkungen auf Vorgänge oder es wurde eine vernünftige Umgehung des Fehlers implementiert.

2. Beschreiben Sie den Fehler und erfassen Sie Hintergrundinformationen. Beschreiben Sie IBM das Problem so genau wie möglich. Fügen Sie relevante Hintergrundinformationen hinzu, damit Ihnen die Experten der IBM Rational-Softwareunterstützung helfen können, das Problem effizient zu lösen. Halten Sie folgende Informationen bereit, damit dies noch schneller möglich ist:

    - Welche Softwareversion haben Sie ausgeführt, als das Problem aufgetreten ist?

    - Bestimmen Sie den genauen Produktnamen und die Produktversion, indem Sie die für Sie anwendbare Option wählen:
        - Starten Sie IBM Installation Manager, und klicken Sie auf **File** > **View Installed Packages**. Erweitern Sie eine Paketgruppe, und wählen Sie ein Paket aus, um den Paketnamen und die Versionsnummer anzuzeigen.
        - Starten Sie Ihr Produkt, und klicken Sie auf **Hilfe** > **Info**, um den Namen des Produktangebots und die Versionsnummer anzuzeigen.

    - Welches Betriebssystem verwenden Sie in welcher Version (einschließlich Service-Packs und Programmkorrekturen)?

    - Verfügen Sie über Protokolle, Traces und Nachrichten in Zusammenhang mit den Fehlersymptomen?

    - Können Sie das Problem reproduzieren? Falls ja, welche Schritte sind dazu erforderlich?

    - Haben Sie Änderungen am System vorgenommen? Haben Sie beispielsweise Änderungen an der Hardware, am Betriebssystem, an der Netzsoftware oder an anderen Systemkomponenten vorgenommen?

3. Verwenden Sie derzeit eine Ausweichlösung für das Problem? Falls ja, beschreiben Sie diese ggf. bei der Meldung des Problems.

4. Senden Sie den Fehlerbericht auf eine der folgenden Weisen an die IBM Rational-Softwareunterstützung:

    - Online: Öffnen Sie die Website der IBM Rational-Softwareunterstützung unter https://www.ibm.com/software/rational/support/. Klicken Sie im Bereich Rational Support Task Navigator auf **Open Service Request**. Wählen Sie das elektronische Fehlerberichtstool aus, und öffnen Sie einen PMR (Problem Management Record), um das Problem zu beschreiben.

    - Informationen zum Anlegen einer Serviceanforderung finden Sie unter http://www.ibm.com/software/support/help.html.

    - Sie können auch über den IBM Support Assistant eine Online-Serviceanforderung anlegen. Weitere Informationen finden Sie unter http://www.ibm.com/software/support/isa/faq.html.
    - Telefonisch: Konsultieren Sie das IBM Verzeichnis mit den weltweiten Ansprechpartnern, um die für Sie relevante Telefonnummer zu ermitteln. Öffnen Sie das Verzeichnis unter http://www.ibm.com/planetwide/, und klicken Sie auf Ihr Land bzw. Ihre Region.

    - Über Ihren IBM Ansprechpartner: Wenn Sie die IBM Rational-Softwareunterstützung elektronisch oder telefonisch nicht erreichen können, wenden Sie sich an Ihren IBM Ansprechpartner. Bei Bedarf kann Ihr IBM Ansprechpartner eine Serviceanforderung für Sie anlegen. Vollständige Informationen für die einzelnen Länder finden Sie unter http://www.ibm.com/planetwide/.

5

Anhang:

Einführung

Dieses Kapitel umfasst Informationen zur rechtmäßigen Nutzung und zu den Marken von IBM® Rational® System Architect®.

Bemerkungen

Die vorliegenden Informationen wurden für Produkte und Services entwickelt, die auf dem deutschen Markt angeboten werden. Möglicherweise bietet IBM die in dieser Dokumentation beschriebenen Produkte, Services oder Funktionen in anderen Ländern nicht an. Informationen über die gegenwärtig im jeweiligen Land verfügbaren Produkte und Services sind beim zuständigen IBM Ansprechpartner erhältlich. Hinweise auf IBM Lizenzprogramme oder andere IBM Produkte bedeuten nicht, dass nur Programme, Produkte oder Services von IBM verwendet werden können. Anstelle der IBM Produkte, Programme oder Services können auch andere, ihnen äquivalente Produkte, Programme oder Services verwendet werden, solange diese keine gewerblichen oder anderen Schutzrechte von IBM verletzen. Die Verantwortung für den Betrieb von Produkten, Programmen und Services anderer Anbieter liegt beim Kunden.

Für in dieser Dokumentation beschriebene Erzeugnisse und Verfahren kann es IBM Patente oder Patentanmeldungen geben. Mit der Auslieferung dieses Handbuchs ist keine Lizenzierung dieser Patente verbunden. Lizenzanforderungen sind schriftlich an folgende Adresse zu richten (Anfragen an diese Adresse müssen auf Englisch formuliert werden):

IBM Director of Licensing
IBM Europe, Middle East and Africa
Tour Descartes 2, avenue Gambetta
France

Trotz sorgfältiger Bearbeitung können technische Ungenauigkeiten oder Druckfehler in dieser Veröffentlichung nicht ausgeschlossen werden. Die hier enthaltenen Informationen werden in regelmäßigen Zeitabständen aktualisiert und als Neuausgabe veröffentlicht. IBM kann ohne weitere Mitteilung jederzeit Verbesserungen und/oder Änderungen an den in dieser Veröffentlichung beschriebenen Produkten und/oder Programmen vornehmen.

Verweise in diesen Informationen auf Websites anderer Anbieter werden lediglich als Service für den Kunden bereitgestellt und stellen keinerlei Billigung des Inhalts dieser Websites dar. Das über diese Websites verfügbare Material ist nicht Bestandteil des Materials für dieses IBM Produkt. Die Verwendung dieser Websites geschieht auf eigene Verantwortung.

Werden an IBM Informationen eingesandt, können diese beliebig verwendet werden, ohne dass eine Verpflichtung gegenüber dem Einsender entsteht.

Lizenznehmer des Programms, die Informationen zu diesem Produkt wünschen mit der Zielsetzung: (i) den Austausch von Informationen zwischen unabhängig voneinander erstellten Programmen und anderen Programmen (einschließlich des vorliegenden Programms) sowie (ii) die gemeinsame Nutzung der ausgetauschten Informationen zu ermöglichen, wenden sich an folgende Adresse:

Intellectual Property Dept. for Rational Software
IBM Corporation

1 Rogers Street

Cambridge, MA 02142
U.S.A

Die Bereitstellung dieser Informationen kann unter Umständen von bestimmten Bedingungen - in einigen Fällen auch von der Zahlung einer Gebühr - abhängig sein.

Die Lieferung des im Dokument aufgeführten Lizenzprogramms sowie des zugehörigen Lizenzmaterials erfolgt auf der Basis der IBM Rahmenvereinbarung bzw. der Allgemeinen Geschäftsbedingungen von IBM, der IBM Internationalen Nutzungsbedingungen für Programmpakete oder einer äquivalenten Vereinbarung.

Alle in diesem Dokument enthaltenen Leistungsdaten stammen aus einer kontrollierten Umgebung. Die Ergebnisse, die in anderen Betriebsumgebungen erzielt werden, können daher erheblich von den hier erzielten Ergebnissen abweichen. Einige Daten stammen möglicherweise von Systemen, deren Entwicklung noch nicht abgeschlossen ist. Eine Gewährleistung, dass diese Daten auch in allgemein verfügbaren Systemen erzielt werden, kann nicht gegeben werden. Darüber hinaus wurden einige Daten unter Umständen durch Extrapolation berechnet. Die tatsächlichen Ergebnisse können

davon abweichen. Benutzer dieses Dokuments sollten die entsprechenden Daten in ihrer spezifischen Umgebung prüfen.

Alle Informationen zu Produkten anderer Anbieter stammen von den Anbietern der aufgeführten Produkte, deren veröffentlichten Ankündigungen oder anderen allgemein verfügbaren Quellen. IBM hat diese Produkte nicht getestet und kann daher keine Aussagen zu Leistung, Kompatibilität oder anderen Merkmalen machen. Fragen zu den Leistungsmerkmalen von Produkten anderer Anbieter sind an den jeweiligen Anbieter zu richten.

Diese Veröffentlichung enthält Beispiele für Daten und Berichte des alltäglichen Geschäftsablaufes. Sie sollen nur die Funktionen des Lizenzprogramms illustrieren; sie können Namen von Personen, Firmen, Marken oder Produkten enthalten. Alle diese Namen sind frei erfunden; Ähnlichkeiten mit tatsächlichen Namen und Adressen sind rein zufällig.

Copyrightlizenz

Diese Veröffentlichung enthält Musteranwendungsprogramme, die in Quellensprache geschrieben sind und Programmiertechniken in verschiedenen Betriebsumgebungen veranschaulichen. Sie dürfen diese Musterprogramme kostenlos kopieren, ändern und verteilen, wenn dies zu dem Zweck geschieht, Anwendungsprogramme zu entwickeln, zu verwenden, zu vermarkten oder zu verteilen, die mit der Anwendungsprogrammierschnittstelle für die Betriebsumgebung konform sind, für die diese Musterprogramme geschrieben werden. Diese Beispiele wurden nicht unter allen denkbaren Bedingungen getestet. Daher kann IBM die Zuverlässigkeit, Wartungsfreundlichkeit oder Funktion dieser Programme weder zusagen noch gewährleisten.

Kopien oder Teile der Musterprogramme bzw. daraus abgeleiteter Code müssen folgenden Copyrightvermerk beinhalten:

© (Name Ihrer Firma) (Jahr). Teile des vorliegenden Codes wurden aus Musterprogrammen der IBM Corporation abgeleitet. © Copyright IBM Corp. 2000 2009.

Wird dieses Buch als Softcopy (Book) angezeigt, erscheinen keine Fotografien oder Farbabbildungen.

Marken

IBM, das IBM Logo und ibm.com sind Marken oder eingetragene Marken der International Business Machines Corp., die in zahlreichen Gerichtsbarkeiten weltweit registriert sind. Andere Produkt- oder Servicenamen können das Eigentum von IBM oder anderen Unternehmen sein. Eine aktuelle Liste der IBM Marken finden Sie auf der Webseite "Copyright and trademark information" unter www.ibm.com/legal/copytrade.html

Microsoft und Windows sind Marken der Microsoft Corporation in den Vereinigten Staaten und/oder anderen Ländern.

Weitere Unternehmen-, Produkt- oder Servicenamen in diesem Dokument können Marken anderer Hersteller sein.