

Oracle® Retail Predictive Application Server
Configuration Tools User Guide
Release 12.0.6

October 2007

Copyright © 2007, Oracle. All rights reserved.

Primary Author: Gary O'Hara

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software—Restricted Rights (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

Value-Added Reseller (VAR) Language

- (i) the software component known as **ACUMATE** developed and licensed by Lucent Technologies Inc. of Murray Hill, New Jersey, to Oracle and imbedded in the Oracle Retail Predictive Application Server – Enterprise Engine, Oracle Retail Category Management, Oracle Retail Item Planning, Oracle Retail Merchandise Financial Planning, Oracle Retail Advanced Inventory Planning and Oracle Retail Demand Forecasting applications.
- (ii) the **MicroStrategy** Components developed and licensed by MicroStrategy Services Corporation (MicroStrategy) of McLean, Virginia to Oracle and imbedded in the MicroStrategy for Oracle Retail Data Warehouse and MicroStrategy for Oracle Retail Planning & Optimization applications.
- (iii) the **SeeBeyond** component developed and licensed by Sun Microsystems, Inc. (Sun) of Santa Clara, California, to Oracle and imbedded in the Oracle Retail Integration Bus application.
- (iv) the **Wavelink** component developed and licensed by Wavelink Corporation (Wavelink) of Kirkland, Washington, to Oracle and imbedded in Oracle Retail Store Inventory Management.
- (v) the software component known as **Crystal Enterprise Professional and/or Crystal Reports Professional** licensed by Business Objects Software Limited (“Business Objects”) and imbedded in Oracle Retail Store Inventory Management.
- (vi) the software component known as **Access Via™** licensed by Access Via of Seattle, Washington, and imbedded in Oracle Retail Signs and Oracle Retail Labels and Tags.
- (vii) the software component known as **Adobe Flex™** licensed by Adobe Systems Incorporated of San Jose, California, and imbedded in Oracle Retail Promotion Planning & Optimization application.
- (viii) the software component known as **Style Report™** developed and licensed by InetSoft Technology Corp. of Piscataway, New Jersey, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration application.
- (ix) the software component known as **i-net Crystal-Clear™** developed and licensed by I-NET Software Inc. of Berlin, Germany, to Oracle and imbedded in the Oracle Retail Central Office and Oracle Retail Back Office applications.
- (x) the software component known as **WebLogic™** developed and licensed by BEA Systems, Inc. of San Jose, California, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration application.
- (xi) the software component known as **DataBeacon™** developed and licensed by Cognos Incorporated of Ottawa, Ontario, Canada, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration application.

Contents

Preface	xi
Audience	xi
Related Documents	xi
Customer Support	xi
Review Patch Documentation	xi
Oracle Retail Documentation on the Oracle Technology Network	xi
Conventions	xii
1 Introduction	1
Overview	1
Configuration Tools Business Process	1
Sample Configurations	2
Know the Configuration Manager	2
Navigating the Configuration Tools	3
2 Configuration Manager	5
Overview	5
Projects	5
Hierarchy	5
Data Interface	5
Styles	6
Solutions	6
Measures	6
Expressions, Rules, and Rule Groups	6
Workbooks and Worksheets	7
Wizards	7
Task List	7
How RPAS Uses Solution Configurations	8
The RPAS Calculation Engine	8
Aggregation and Spreading	8
RPAS Functions	8
Right-Click Menus in the Configuration Manager	9
Setting Tools Preferences	10
3 Projects	13
Working with Projects	13
Overview	13
Create a Project	13
Save Changes to a Project	14
Using “Save As” to Save a Project Using a Different Name	15
Open an Existing Project	15
Open an Existing Project from an Older Version of the Configuration Tools	17
Close a Project	18
Hierarchies	18
Overview	18
The Hierarchy Definition Tool	19
Working with Hierarchies	20
Overview	20
Create a New Hierarchy	20
Specify Hierarchy Information	21
Delete a Hierarchy	23

Edit a Hierarchy	24
Working with Position Formats	25
Overview	25
Specifying the Position Format	25
Working with Dimensions	27
Overview	27
Create a Dimension	27
Specify Dimension Information	28
Delete a Dimension	31
Edit a Dimension	31
Create a Hierarchy Branch	32
Data Interface Tool	33
Overview	33
Specify the Data Interface for a Measure	33
Edit Data Interface Information for a Measure	35
Delete Data Interface Information for a Measure	37
Working with Styles	37
Overview	37
The Style Tool	37
Style Attributes	37
Create a Style	39
Remove a Style	40
Edit a Style	41
4 Solutions	43
Working with Solutions	43
Overview	43
Create a Solution	43
Copy a Solution	44
Rename a Solution	45
Move a Solution	45
Delete a Solution	46
Measures and Components	46
Overview	46
Measure Properties	47
Working with Components	48
Measure Component Design	48
Create a Major Component	49
Create a Minor Component	50
Specify Component Information	51
Edit the Components	62
Working with Measures	65
Overview	65
Realize and Unrealize Measures	65
Rename a Measure	67
Show Measures by Component	67
Show All Measures	67
Hide Measures by Component	68
Hide All Measures	68
Sort Measures by Property Value	69
Filter Measures by Property Value	69
Imported Measures	70
Overview	70

Import a Measure.....	71
Remove an Imported Measure from a Solution	72
Rule Sets	73
Overview	73
Create a Rule Set	73
Add Attributes to a Rule Set.....	74
Delete a Rule Set.....	76
Edit Properties of a Rule Set	76
Rule Groups.....	77
Overview	77
Types of Rule Groups.....	77
Rule Group Validation	78
Multiple Refresh Rule Groups	78
Rule Group Transitions.....	79
Create a Rule Group	80
Add Attributes to a Rule Group	82
Delete a Rule Group	83
Edit Properties of a Rule Group.....	84
Copy a Rule Group.....	85
Measure Validation in the Rule Tool	86
Rules	87
Overview.....	87
Create a Rule and Add It to a Rule Group	87
Add an Existing Rule to a Rule Group	93
Apply a Rule Pattern to Create New Rules or to Update Existing Rules.....	95
Delete a Rule from All Rule Groups.....	98
Remove a Rule from a Rule Group.....	100
Edit Properties of a Rule	100
Rename All Rules in a Rule Group.....	102
Filter Rules in a Rule Group	104
Reordering Rules in a Rule Group	105
Auto Generate Load and Commit Rules	105
Copy Selected Rules to Another Rule Group.....	106
Find and Replace Measures in the Copied Rules	108
Expressions and Rules.....	110
Non-Conforming Recalc Measures	110
Reorder an Expression in a Rule.....	111
Edit an Expression in a Rule.....	111
Delete an Expression from a Rule.....	113
Add an Expression to a Rule	114
RPAS Functions, Procedures, Keywords, and Modifiers	114
Workbooks.....	114
Overview	114
Workbook Tabs	115
Worksheets	115
Rule Groups.....	115
Wizards	115
Create a Workbook.....	115
Overview of Participation Measures.....	128
Configure Extended Measures.....	129
The Usage and Args Properties	130
Edit Workbook Properties	132
Remove a Workbook	132

Working with the Rule Group Simulator	133
Overview	133
About the Rule Group Simulator	133
Invoking the Rule Group Simulator	135
Filtering the Measures Table	136
Changing the Edited Status of Measures	137
Using the Upstream and Downstream Panes	137
Exiting the Rule Group Simulator	138
Working with Workbook Tabs	139
Overview	139
Create a Workbook Tab	139
Edit Workbook Tab Properties	140
Remove a Workbook Tab	140
Working with Worksheets	141
Overview	141
Create a Worksheet	142
Specify Which Measures Are on a Worksheet	144
Specify the Sequence of Measures on a Worksheet	146
Edit Worksheet Properties	148
Remove a Worksheet	148
Wizards	148
Overview	148
Create a Wizard Group	150
Create a Wizard Page	151
Edit Control Properties	152
5 System Preferences	153
Overview	153
Global Domain	153
Overview	153
Measure Data	154
Hypersparse	154
Multi-Language	155
Setting Tools Preferences	155
Set Configuration Properties	157
6 Configuration Utilities	159
Overview	159
Configuration Converter	159
Overview	159
Launching the Converter	159
Converting a Configuration	160
Functional Library Manager	161
Overview	161
Launching the Functional Library Manager	162
Adding a Function Library to Be Validated in the Configuration Tools	162
Removing a Function Library from being Validated in the Configuration Tools	163
Report Generator	163
Overview	163
Generate a Report	164
7 Building and Patching Domains	165
Build a Domain	165

Overview	165
Prerequisites	165
Client-Side Procedures	165
Server-Side Procedures	167
Manually Building the Domain Using a Command-Line Interface	169
Validate Domain Build Results	170
Building the Domain using the GUI Installer	171
After Building the Domain	175
Patching a Domain	175
A Appendix: Global Domain Technical Information	177
About Domains	177
B Appendix: Calculation Engine User Guide	179
Overview	179
Measure Definition and Base Intersections	180
Data Types	180
Base Intersection	180
Aggregation and Spreading Types	180
Aggregation	181
Overview	181
Aggregation Types	182
Spreading	184
Introduction	184
Locks and Spreading around Locked and Changed Cells	184
Spreading Methods	185
Multi-Level Spreading	188
Hierarchical Protection Processing	189
The Spreading of Recalc Type Measures	190
Expressions, Rules, and Rule Groups	191
Introduction	191
Expressions	191
Rules	192
Rule Groups	193
Rule Group Transitions	194
The Calculation Cycle	195
Introduction	195
Protection Processing	195
Determining What to Calculate	197
Determining the Calculation Sequence	198
Cycle Groups	199
Synchronized Measures	200
Overview	200
Synchronized Inventory Examples	201
Elapsed Periods	202
Elapsed Periods and Spreading	202
Non-Conforming Expressions	203
Introduction	203
Handling of Non-conforming Expressions	203
C Appendix: Rules Function Reference Guide	205
Overview	205
Introduction	205
Syntactical Approach	206

Specification of Hierarchy, Dimension, or Position	207
Function Inverses	207
Functions with Multiple Results	207
Special Handling for Functions	208
Error Handling	208
if	208
prefer	209
Non-Conforming Measures	210
Definition	210
Functional Keywords	212
Overview	212
Calendar Index Functional Keywords	212
Session Keywords	213
Calendar Hierarchical Date Keywords	214
Modifiers	214
Overview	214
master	214
aggtype	215
level	215
old	216
Description of Functions	218
Calendar Index Functions	218
Index and Position Functions	221
Time Series Functions	224
Hierarchical Functions and Procedures	231
Normalization and Resizing Functions	239
Other Functions and Procedures	240
String Functions	251
Math Functions	252
D Appendix: Aggregation and Spread Types.....	255
Supported Aggregation Types	255
Supported Spread Types	257
Arithmetic Operators	258
Unary Operators	258
Binary Operators	258

Preface

Oracle Retail Configuration Guides are designed so that you can view and understand the application's "behind-the-scenes" processing, including such information as the following:

- Key system administration configuration settings
- Technical architecture
- Functional integration dataflow across the enterprise
- Batch processing

Audience

This document is intended for the users and administrators of Oracle Retail Predictive Application Server. This may include merchandisers, buyers, and business analysts.

Related Documents

For more information, see the following documents in the Oracle Retail Predictive Application Server Release 12.0.6 documentation set:

- Oracle Retail Predictive Application Server Release Notes
- Oracle Retail Predictive Application Server User Guide

Customer Support

- <https://metalink.oracle.com>

When contacting Customer Support, please provide:

- Product version and program/module name.
- Functional and technical description of the problem (include business impact).
- Detailed step-by-step instructions to recreate.
- Exact error message received.
- Screen shots of each step you take.

Review Patch Documentation

For a base release ("0" release, such as 12.0), Oracle Retail strongly recommends that you read all patch documentation before you begin installation procedures. Patch documentation can contain critical information related to the base release, based on new information and code changes that have been made since the base release.

Oracle Retail Documentation on the Oracle Technology Network

In addition to being packaged with each product release (on the base or patch level), all Oracle Retail documentation is available on the following Web site:

http://www.oracle.com/technology/documentation/oracle_retail.html

Documentation should be available on this Web site within a month after a product release. Note that documentation is always available with the packaged code on the release date.

Conventions

Navigate: This is a navigate statement. It tells you how to get to the start of the procedure and ends with a screen shot of the starting point and the statement “the Window Name window opens.”

Note: This is a note. It is used to call out information that is important, but not necessarily part of the procedure.

This is a code sample
It is used to display examples of code

A [hyperlink](#) appears like this.

Introduction

Overview

The Oracle Retail Predictive Application Server (RPAS) configuration tools provide a flexible means to configure and build RPAS-based applications with customer-specific business parameters. The configuration tools provide a streamlined, user-friendly interface to utilize RPAS functionality. Once a configuration is created, an installer script is used to build an RPAS domain.

The configuration tools consist of an integrated set of task-specific configuration aids that are used to configure a solution template or to modify an existing solution template.

RPAS functionality is exposed to the configuration tools via Application Programming Interfaces (APIs).

A configuration is typically created and maintained by an application administrator or solution expert. Details of the configuration are stored locally on the administrator's PC or on the network. Once the configuration is complete, the administrator uses the configuration to create a new domain or update an existing domain.

Users of the configured solution will access the RPAS domain by using the RPAS client that is installed on their machines. The domain that they access will represent the business process and environment that was configured in the solution by the Configuration Administrator together with the appropriate data.

Once the domain is created, administrative RPAS processes (such as hierarchy maintenance and user administration) are accomplished by an RPAS administrator using the utilities on the server.

Note: For more information on RPAS administration and installation, refer to the *RPAS Administration Guide* and the *RPAS Installation Guide*.

Configuration Tools Business Process

The business processes performed by the Configuration Tools include the following:

- Set up system properties
- Create a project
- Create solutions
- Configure hierarchies and dimensions
- Configure measures and measure components
- Configure rules sets, rule groups, and rules
- Configure workbooks, workbook tabs, and worksheets
- Configure wizards
- Define interfaces used to import data
- Build an RPAS domain

Sample Configurations

The examples illustrated in this document utilize a Sample Configuration, which is delivered with the RPAS platform and can be installed along with the RPAS and Configuration Tools. The Sample Configuration may not match every illustration because RPAS and Tools software versions might vary between users. The illustrations are meant to provide a context to the reader. Detailed documentation of the Sample Configuration is provided in the *RPAS Installation Guide*.

Know the Configuration Manager

As you create new configurations; you name the configuration, the project, and the solution in the configuration. You can drill down through the configuration to work in specific areas. The icons in the Configuration Manager help you know where you are working, and navigation statements in this document use these icons to help you follow procedures. If an area of the configuration has been modified, its icon will contain a modification flag , and it must be saved if the modifications are required.

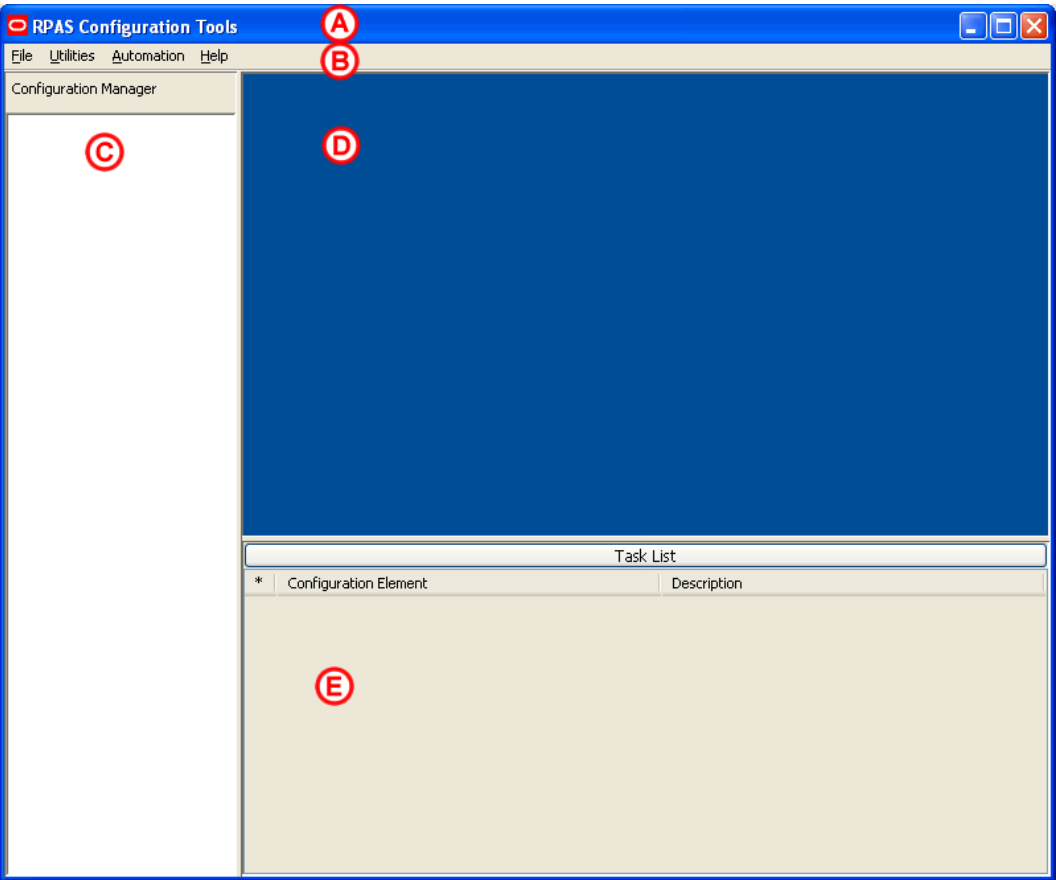
Configuration Manager	Icon Name	Window Displayed in the Workspace
 Sample_Config	 Project	None
 Hierarchies	 Hierarchies	Hierarchy Definition window
 Data Interface	 Data Interface	Data Interface Manager window
 Styles	 Styles	Style Definition Window
 Solution_1	 Solution	None
 Measures	 Measures	Measure Manager window
 Rules	 Rules	Rule Definition window
 Workbooks	 Workbooks	Workbook Designer window
 Wizards	 Wizards	Wizard Designer window

Navigating the Configuration Tools

The Configuration Tools uses a Windows GUI with many of the standard components found in other Windows applications. These include:

- Drop-down menus
- Toolbars
- Active buttons
- Right-click functionality

Once the Configuration Tools are launched, the configuration administrator has access to the RPAS Configuration Tools window.



Configuration Tools Window

The primary elements in the application window are as follows:

Element	Purpose
Title Bar (A)	▪ Displays the product name ▪ The three buttons at the far right on the title bar allow for the application window to be minimized, restored, maximized, and closed
Menu Bar (B)	▪ Contains the menus that are used in the Configuration Tools ▪ Each menu contains a set of commands that allow the configuration administrator to operate the Configuration Tools

Element	Purpose
Configuration Manager (C)	▪ Displays information about configurations, projects, and solutions that are currently in use ▪ Configuration information is not displayed until a configuration is opened.
Workspace (D)	▪ As different elements are selected of a configuration in the Configuration Manager, the related windows are displayed in the workspace.
Task List (E)	▪ Displays errors and warnings within the configuration

Configuration Manager

Overview

The Configuration Manager is the starting point for creating a new configuration or for opening an existing configuration. It provides a high-level view of all the components that are necessary to configure an RPAS application, and it is used to navigate to the various tools that are used to configure those components.

The Configuration Manager is the core of the RPAS Configuration Tools, and it provides an overall view of the configuration components. Each configuration contains one project and one or more solutions.

Projects

Each project represents a single, logical RPAS domain although it may become several physical domains in a “global domain” environment. A project contains one or more solutions. The meaning of “solution” is application specific.

Hierarchy



The user may access the Hierarchy Definitions window by either selecting the icon or **Hierarchy** from the Configuration Manager. For each project, a single or multiple hierarchies may be created and dimensions are defined within each hierarchy. Hierarchies are the structures used by an organization to describe the relationships that exist between the dimensions. The following hierarchies are automatically created when a new project is defined:

- Calendar
- Product
- Location

Users may create and define the individual dimensions for these hierarchies and for any additional hierarchies that may be desired.

Note: The RPAS (system) names for the default hierarchies (CLND, PROD, LOC, and ADMU) cannot be changed, but the default labels (Calendar, Product, and Location) can be changed. The ADMU label cannot be changed.

Data Interface



The user may access the Data Interface Manager by either selecting the icon or **Data Interface** from the Configuration Manager. The Data Interface Tool is used to define the format of data interface files and provide some data interface parameters, such as directions to RPAS on how to deal with data that is sourced below its base intersection. The tool sets measure attributes that are referenced when loading measure data into the domain. The information entered into the data interface will be referenced when the “loadmeas” utility is used to load data for a measure.

Styles

 The user may access the Style Definition window by either selecting the icon or “Styles” from the Configuration Manager. The style tool is used to define styles that specify how the data for a measure is to be displayed within the RPAS Client. Styles consist of a number of attributes; such as text font, size, and color as well as specifications of precision, alignment of text within the cell. These styles may then be assigned to measures within the Measure and Workbook Tools.

Solutions

 A solution corresponds to an application configuration (for example, Financial Planning or Item Planning). For each solution, the following are configured:

- Measures
- Rule sets / rule groups / rules
- Workbooks / worksheets
- Wizards (optional)

Measures

 Measures (multidimensional variables) are any item of data that can be represented on a grid in a worksheet. Measures are the data points used in the customer’s business process.

Expressions, Rules, and Rule Groups

 Rules are collections of expressions (the basis of all calculations) that describe the relationships between measures. They are evaluated by the RPAS calculation engine during a calculation. Rules can consist of multiple expressions as the following example represents:

- Expression 1: $\text{ReceiptUnits} = \text{ReceiptValue} / \text{ReceiptPrice}$
- Expression 2: $\text{ReceiptValue} = \text{ReceiptUnits} * \text{ReceiptPrice}$
- Expression 3: $\text{ReceiptPrice} = \text{ReceiptValue} / \text{ReceiptUnits}$

The collection of expressions represents a rule. These three expressions state the relationship between ReceiptUnits, ReceiptValue, and ReceiptPrice. Each expression solves for a different measure.

A rule group is a collection of rules that are treated as a unit by the calculation engine. The rules in the rule group must be considered together to satisfy the calculation requirements for a specific business process. The sequence of rules in a rule group determines the calculation sequence unless the sequence is forced.

A rule group is a collection of rules that are treated as a unit by the calculation engine. The rules in the rule group must be considered together to satisfy the calculation requirements for a specific business process. The sequence of rules in a rule group determines the calculation sequence unless the sequence is forced.

A rule set is a collection of rule groups that is used for organizational purposes by the tools.

Workbooks and Worksheets



A workbook is the multidimensional framework that is used to perform specific business functions, such as creating a merchandise plan and reviewing available data. Workbooks are easily viewed and manipulated.

A workbook can contain any number of multidimensional spreadsheets (called worksheets) to present data. Measures and rules are used to define and calculate the measure data. All of these components work together to facilitate the viewing and analysis of business functions. The configuration tools allow the configuration administrator to configure workbook templates incorporating these various components.

Wizards



A wizard is a feature that guides the user through the process of building a new workbook. A wizard displays successive dialogs that require the user to answer a sequence of questions or enter information regarding the content of the workbook. Responses to these questions are used to format and populate the workbook. The layout of these wizard dialogues could be defined using the wizard tool. However, each workbook may use a standard wizard configuration, eliminating the need for the configuration administrator to access the Wizard Designer.

The main purpose of the wizard is to allow the end user to make choices regarding the scope of the workbook. For example, the first wizard might ask the user to select the SKUs to include in the workbook. The second might ask the user to select the stores to include in the workbook, and the third wizard might ask the user to select the dates to include in the workbook. At the end of the series of wizards, a workbook will be created that has data for the SKUs, stores, and dates that the user selected.

Task List

The Task List provides a centralized view of errors and warnings that are issued as a result of information input by the user. Use the information in the Task List as a guide for correcting errors or omissions in the Project.

The Task List title bar serves as a status indicator. If the title “Task List” is displayed in red, the Task List contains items that need the user’s attention. If there are no errors or warnings, the title will be displayed in black.

The title bar can also be used to show or hide the Task List. If the Task List is visible, click anywhere on the title bar to hide the list and move the title bar to the bottom of the window. If it is hidden, click anywhere on the title bar to display the Task List. The amount of space used by the task list sub-pane can also be changed by dragging the separator above the task list title bar.

The Task List has three columns:

1. The first column indicates the nature of the Task List item. A  indicates an error. A  indicates a warning.
2. The second column identifies the configuration element involved.
3. The third column provides a description of the issue.

Errors typically indicate definite validation problems, which are shown in red in the tool where the configuration setting is made. When the user fixes the erroneous condition, the Task List automatically removes the error listing for that condition. Warnings indicate the possibility of a problem occurring, and the user is advised to inspect the suspected element to ensure that everything is in order. Since warnings are more general than

errors the Task List will not remove them automatically. The user is provided with options to remove errors and warnings via a right-click menu.

How RPAS Uses Solution Configurations

The RPAS Calculation Engine

The RPAS calculation engine is a very powerful and flexible engine that is built to support On-Line Analytical Processing (OLAP) type calculations against a multi-dimensional model.

In the OLAP model, individual pieces of data (called cells) apply to a single position in one or more hierarchies or dimensions. Cells typically reference:

- A measure
- A calendar or time hierarchy
- Other hierarchies, such as product and location

The measure is fundamentally different to the other hierarchies, because measures represent the fundamental events or measurements that are being recorded. The positions in the other hierarchies provide a context for the measurement: where, when, what, and so on. Measures relate to one another through rules and expressions. Positions in all the other hierarchies relate to each other through hierarchical relationships.

Aggregation and Spreading

The RPAS calculation engine is designed to be robust and extensible, but in complete control of the calculation process. It enforces integrity of the data by ensuring that all known relationships between cells are always enforced. Much of the logic of the processing of rules and rule groups depends on this basic principal. RPAS supports two different forms of relationships between cells:

1. Hierarchical relationships that require aggregation and spreading
2. Measure relationships that require rules and expressions

Aggregation and spreading are basic capabilities of the engine that do not require coding by the implementer, other than the selection of aggregation and spreading types to use for a measure. Hierarchical relationships, such as weeks rolling up to months or stores rolling up to regions, require the aggregation of data values from lower levels in a hierarchy to higher levels by using a variety of methods as appropriate to the measure.

To enable such data to be manipulated at higher levels, RPAS supports spreading the changes, which also uses a variety of methods.

The inherent relationships between measures can be modeled through a rich rule and expression syntax. Modeling these relationships takes most of the effort in configuring an application model.

RPAS Functions

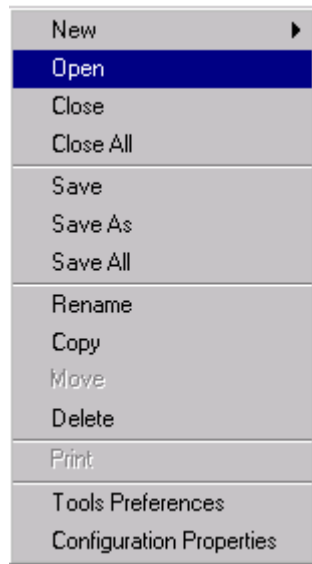
- RPAS Functions are mechanisms for performing operations within an expression that are controlled and executed by the calculation engine.
- Most functions have only one output.
- The calculation engine controls and executes the evaluation of a function.
- Functions may be used in long expressions with other functions and keywords.
- The data that can be referenced is limited to the scope of the workbook.

See Appendix B – Calculation Engine Users Guide – for a comprehensive definition of the RPAS calculation engine and how it is used when configuring a solution.

See Appendix C – Rules Function Reference Guide – for details about standard RPAS functions.

Right-Click Menus in the Configuration Manager

The right-click menu may be accessed by right-clicking in any location within the Configuration Manager. Each available selection from the right-click menu is described in the following sections.

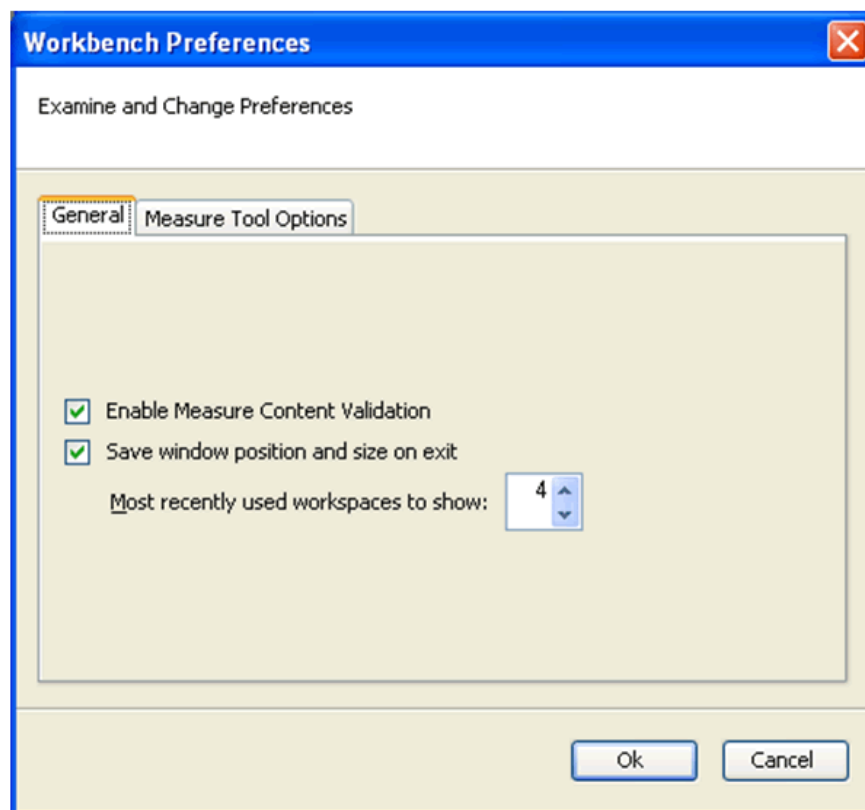


Example: Right-click Menu

Setting Tools Preferences

Settings made here will apply to all configuration projects created or viewed with the tool.

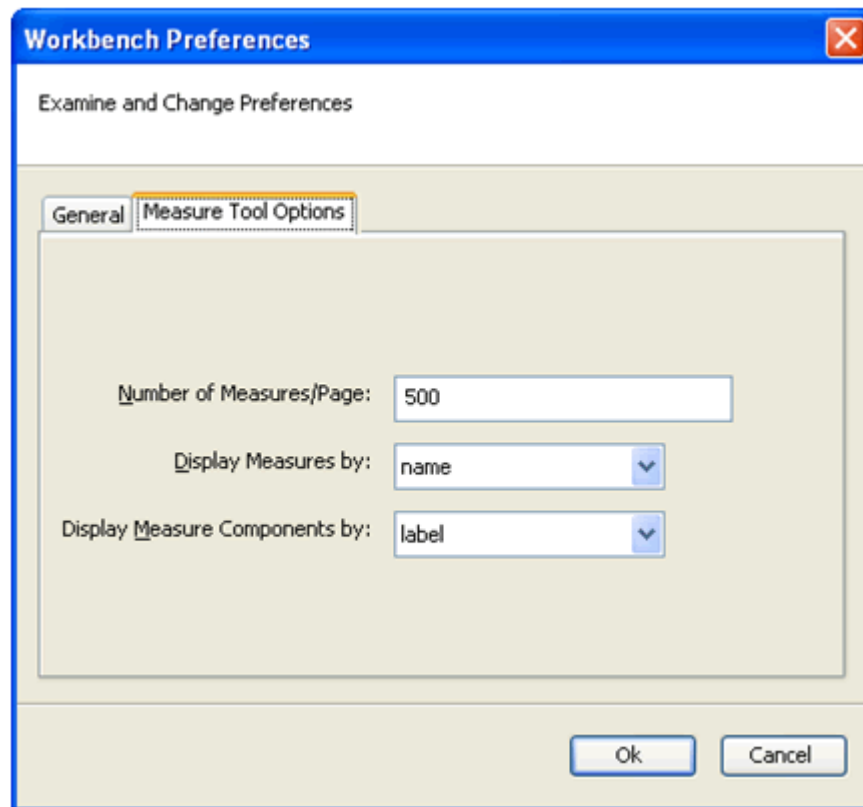
Navigate: From the File menu, select **Tools Preferences**. The Workbench Preferences window opens.



Workbench Preference Window

1. Select the **General** tab.
 - **Enable Measure Content Validation** – Activating this checkbox enables the immediate validation of measure properties when configuration measure information is created or modified. This process can impact the performance of the RPAS Configuration Tools. If this box is not checked, the manual Measure Content Validation icon is enabled on the Rule Definition toolbar. See the Rule Definition Tool for details.
 - **Save window position and size on exit** – Activating this checkbox enables the RPAS Configuration Tools to be launched in either full or minimized view based on the status in which the configuration administrator last exited the application. If the application is exited in a minimized view, the size of the window is also maintained when the RPAS Configuration Tools is re-launched.
 - **Most Recently used workspaces to show** – Use the up and down arrows to specify the number of configurations to be displayed in the Most Recently Used list displayed in the File menu dialog. This list allows the configuration administrator easy selection of a recently viewed configuration.

2. Select the **Measure Manager Options** tab.



Workbook Preferences - Measure Tool Options

- **Number of Measures/Page** – Select the number of measures to display per page in the Measure Manager Tool. The default is 500.
 - **Display Measures by** – Displays measures either by their name or label in various locations of the Tools. The default setting is “name.”
 - **Display Measure Components by** – Display measure components (in the Measure Manager tool) either by name or by label. The default setting is “label.”
3. Click **OK** to save any changes and close the window.

Working with Projects

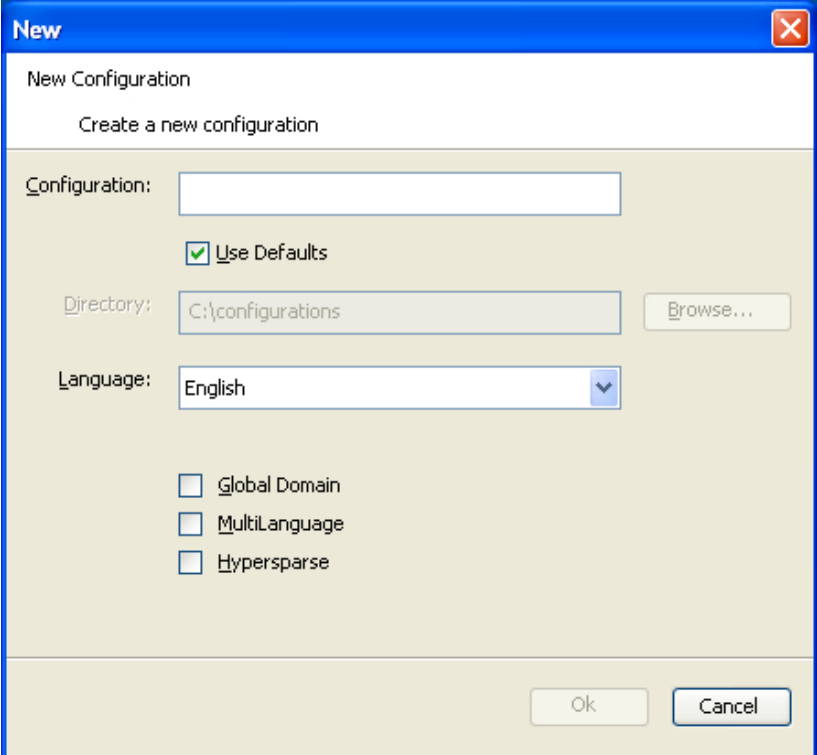
Overview

A project is used to configure the structure of a domain. Each project represents a single, logical RPAS domain although it may become several physical domains in a “global domain” environment. The hierarchies, dimensions, and styles are defined within a project and are available for use within all solutions in a given project.

Note: A solution can use a subset of the hierarchies and dimensions defined within the project. Within a project, you can also define certain properties (which are part of the Data Interface Tool) that describe how measures will be loaded into the domain.

Create a Project

Navigate: From the File menu or right-click menu in the Configuration Manager, select **New – Project**. The New Window opens.



The screenshot shows a Windows-style dialog box titled "New" with a close button (X) in the top right corner. The dialog has a light beige background and a blue border. Inside, the text "New Configuration" is at the top, followed by "Create a new configuration". Below this, there are three main sections: 1. "Configuration:" with an empty text box. 2. "Directory:" with a text box containing "C:\configurations" and a "Browse..." button to its right. 3. "Language:" with a dropdown menu showing "English". At the bottom, there are three unchecked checkboxes: "Global Domain", "MultiLanguage", and "Hypersparse". At the very bottom right, there are "Ok" and "Cancel" buttons.

New Dialog Box

1. In the Configuration field, enter the name of the configuration/project.
2. Select the Use Defaults checkbox to use the default paths that are currently active.
The default path is the most recent path where the project was last saved. If you want to specify an alternate location, uncheck Use Defaults and select the path.
3. Select the language that you want the configuration to be created in. The default is English.
4. Select the options for Global Domain, Multilanguage, and Hypersparse as necessary. Refer to the "System Preferences" section for additional information about these options.

Note: The Global Domain and Multi-Language settings must be defined before the domain is built. A domain can be converted to or from a hypersparse domain after the domain is built. Changes to the Global Domain and MultiLanguage properties are ignored when modifying the configuration of an existing domain.

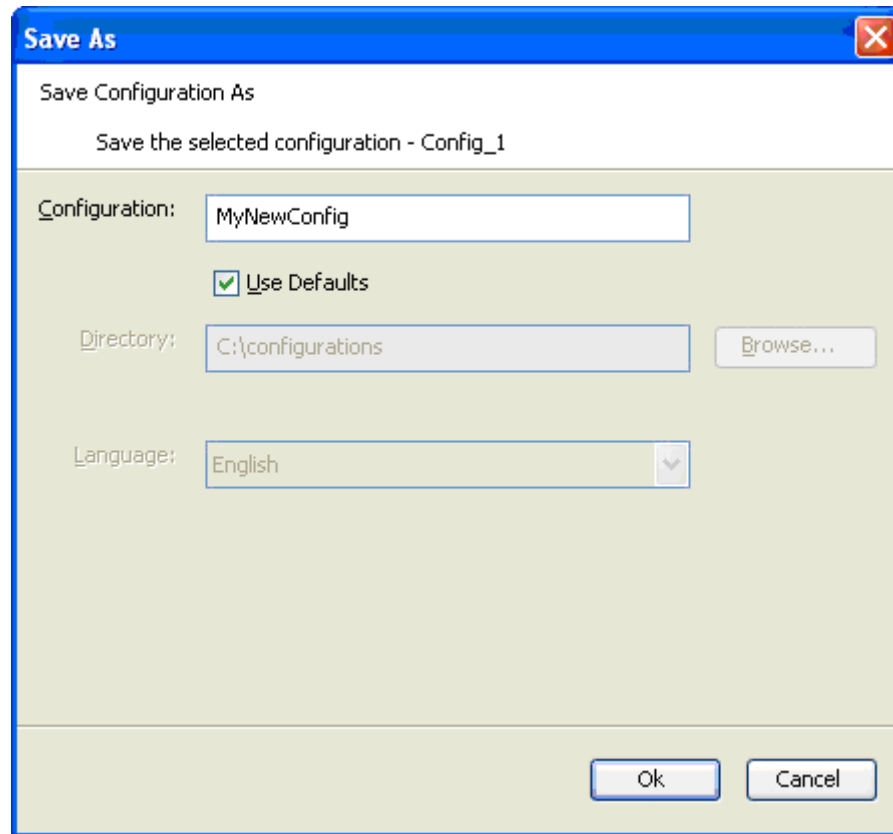
5. Click **OK** to save any changes and close the window.

Save Changes to a Project

Navigate: From the File menu or right-click in the Configuration Manager window, select **Save**. This will save the project under the same name that was used to create it and within the same directory.

Using “Save As” to Save a Project Using a Different Name

Navigate: From the File menu or right-click in the Configuration Manager window, select **Save As**. The Save As dialog appears:



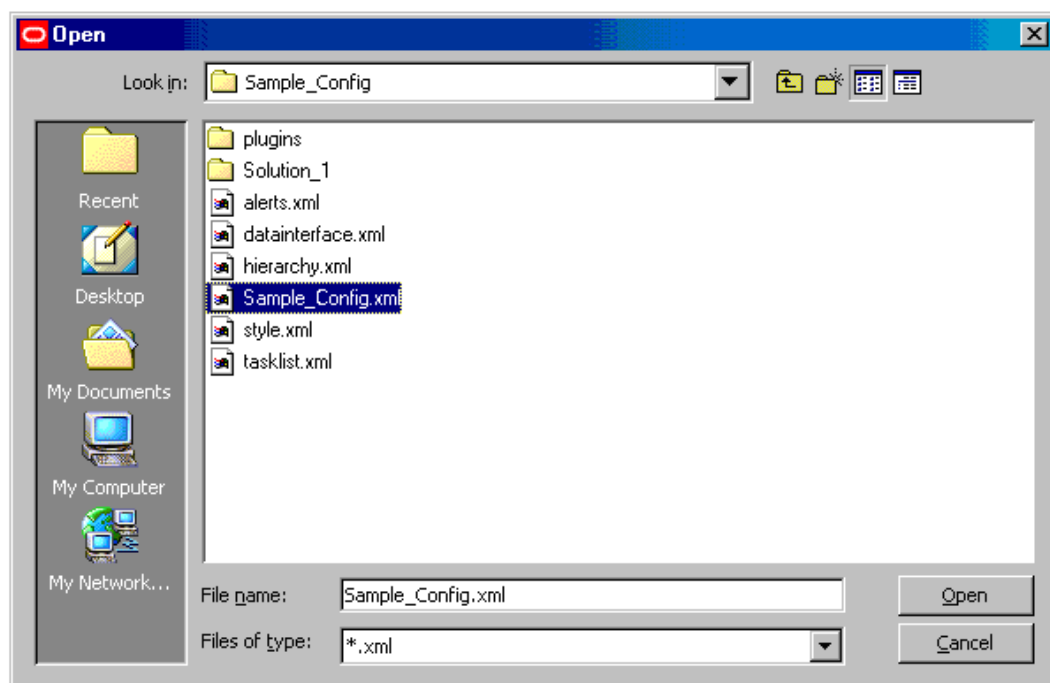
Save As Dialog Box

1. In the Configuration field, enter the new project name.
2. Click **OK** to save the project to the path specified in the Default directory. Deselect Use Defaults and either enter the new directory in the box labeled "Directory" or click the **Browse** button and select the path for the new location.
3. Click **OK** to save the project.

Open an Existing Project

Navigate: From the File menu or right-click in the Configuration Manager window, select **Open**. The Open dialog box appears.

1. Choose one of the following methods:
 - a. Browse to the directory where your project is saved. Select the file whose name is the same as the project with an '.xml' extension and click **Open**. The project will now be opened in the configuration manager.

**Open Dialog Box**

- b. To open a project that was recently opened, select the project from the recently used projects list in the File menu. These projects will be in a numbered list where the most recently used project is first in the list. Click on the project, and it will open in the configuration manager. The number of projects that appears in the most recently used list may be changed by using the Tools Preferences option of the File menu.

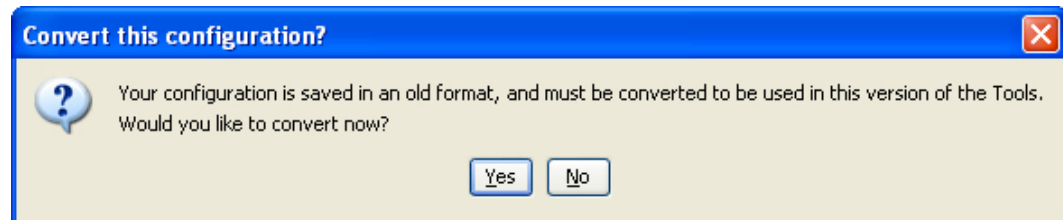
Open an Existing Project from an Older Version of the Configuration Tools

Projects that were created or saved in a previous major version of the Configuration Tools must be converted before they can be viewed or modified.

Navigate: From the File menu or right-click from the Configuration Manager, select **Open**. The Open dialog box appears.

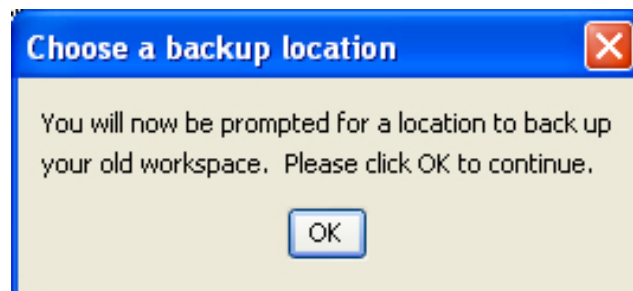
1. Choose one of the following methods:
 - a. Browse to the directory where your project is saved. Select the file whose name is the same as the project with a '.xml' extension and click Open. The project opens in the configuration manager.
 - b. To open a project that was recently opened, select the project from the recently used projects list in the File Menu. These projects will be in a numbered list where the most recently used project is first in the list. Click on the project, and it opens in the configuration manager. The number of projects that appears in the most recently used list may be changed by using the Tools Preferences option of the File menu.

A message box appears:



Convert This Configuration Message Box

2. To convert at a later time without currently viewing or modifying the project, click **No**. Click **Yes** to convert the project so it can be viewed or modified. If you click **Yes**, the following dialog will be displayed

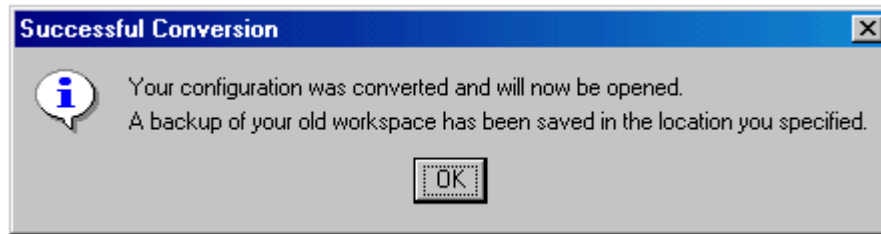


Choose a Backup Location Message Box

3. Click **OK**. The Open dialog box appears and prompts you for a location to place a backup of the project that is to be converted.

Note: The directory that you enter here must not already contain a directory whose name is the name of your configuration. For example, to put a backup of a configuration named "MyConfig" in a directory "C:\Backups," "C:\Backups\MyConfig" must not currently exist.

4. Select the directory to store the backup, and click **Open**. The conversion process will begin. If the conversion successfully completes the following message will be displayed. Select **OK** to continue and view the Project.

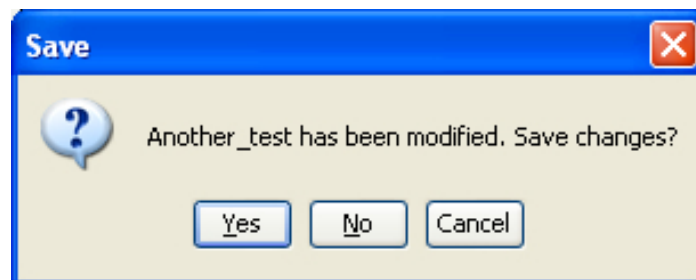


Successful Conversion Dialog Box

An error message appears if this process fails. The original Project will remain untouched and it will not open.

Close a Project

Navigate: From the File menu or right-click from the Configuration Manager, select **Close**. If no changes were made to the Project, the project will be closed. If changes were made, the Save dialog box appears.



- Click **Yes** to save the changes made to the project and close it.
- Click **No** to discard the changes made to the project since the last save and close it.
- Click **Cancel** to return to the configuration manager without closing the project.

Hierarchies

Overview

A hierarchy is a top-to-bottom set up of parent-child relationships between groups belonging to the same entity (for example; Time = years, months, weeks, and days). A hierarchy provides a means to define relationships between dimensions by aggregates (roll ups, and alternate roll ups). The following hierarchies are automatically created and cannot be deleted within the Configuration Tools:

- CLND (Calendar)
- PROD (Product)
- LOC (Location)
- ADMU (ADMU)

ADMU is not a user defined hierarchy, so no one can create or modify it. ADMU is built by RPAS, and the configuration tools make it available for use in configurations. ADMU is the "user" hierarchy, and it exists to allow a measure to use the dimension "user" as part of its base intersection.

You create and define dimensions for these hierarchies and for any additional hierarchies that you create.

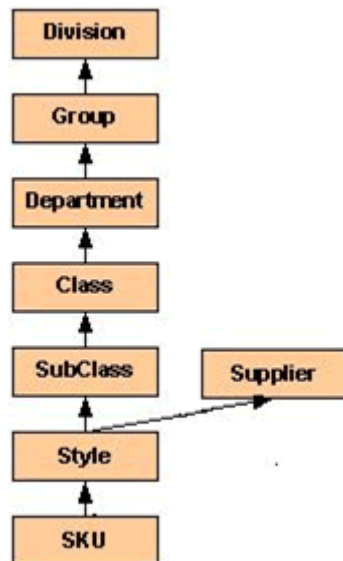
Note: The *names* for the automatically generated hierarchies (CLND, PROD, LOC, and ADMU) cannot be changed, but the default user *labels* for CLND, PROD, and LOC (Calendar, Product, and Location) can be changed. The user *label* of ADMU cannot be changed. The CLND and ADMU hierarchies must exist in all domains, but PROD and LOC are not mandatory. If there are no dimensions created for these hierarchies, the hierarchies will not be created in the resulting domain.

The Hierarchy Definition Tool

The Hierarchy Definition Tool allows configuration administrators to define and construct hierarchies, dimensions each hierarchy, and the relationships between dimensions. It also:

- Provides a visual representation of a hierarchy and its dimensions
- Provides a means to define the hierarchy data load file
- Allows existing hierarchies/dimensions to be reused in a new solution in the same project

The following diagram represents a typical structure of an organization's product hierarchy.



In this example, the Style dimension has two parents: Subclass and Supplier. Each position in the Style dimension will have a parent position in both the SubClass and Supplier dimension.

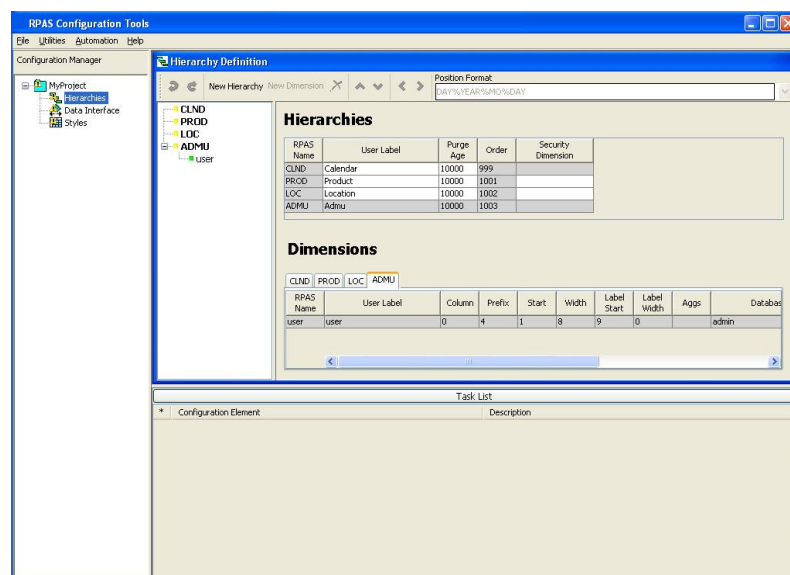
Working with Hierarchies

Overview

When a project is created, the Calendar (CLND), Product (PROD), Location (LOC), and User (ADMU) hierarchies are automatically created by default. Additional hierarchies and dimensions can also be created.

Create a New Hierarchy

Navigate: From Hierarchies within the Configuration Manager or within the left pane of the Hierarchy Definition window, right-click and select **New Hierarchy** from the menu.



- To change the name of the newly created hierarchy in the left pane of the Hierarchy Definition window:
 - Either right-click on the hierarchy name, and select **Rename**.
 - Double click the hierarchy name to enable it.
- Enter the new name.

Note: The RPAS name can only be up to four characters long.

- Press **Enter** or click outside the hierarchy name.

Specify Hierarchy Information

1. In the left-hand pane, select the name of the hierarchy that you want to specify information for.
2. In the Hierarchies area, enter the following:
 - **RPAS Name** – The RPAS internal name of the hierarchy.

Note: You cannot change the name of the default hierarchies.

- **User Label** – The hierarchy label that is displayed to RPAS users.
- **Purge Age** – The purge age determines when a position and its corresponding measure data are removed from a domain. Specifically, it represents the number of days before the data is purged from the last time the position was included in the hierarchy input file that is loaded with the loadhier utility (most commonly on a nightly or weekly basis). Setting this value to zero means that a position and all of its data will be immediately purged if it is not included in the hierarchy file.

Note: The value set in this field serves as the default value to use when loading the corresponding hierarchy. This value can be overwritten by one of the arguments of the loadhier utility each time the utility is called.

Example 1: A purge age of “0” will purge positions the first night they are not in the input file.

Example 2: A purge age of “1000” will purge the positions the 1000th night after they are last seen on the input file.

- **Order** – Hierarchy order determines the ordering of dimension fields in the physical storage of data in the RPAS domain. This ordering is the traversal order of data for calculations, which relates to how RPAS iterates over data when performing calculations. Data in the domain is stored in multi-dimensional arrays with each dimension belonging to a different hierarchy.

For performance reasons, the Calendar hierarchy (and therefore all of its dimensions) is always the “innermost” dimension and defaults to the number 999, which cannot be edited. The ordering of any hierarchy can be changed with the exception of Calendar (CLND). The lower the order number, the nearer the hierarchy is to the innermost dimension.

Consider the following example for the Calendar, Product, and Location hierarchies:

- CLND order = 999
- PROD order = 1001
- LOC order = 1002
- Two products: P1 and P2
- Two locations: L1 and L2
- Two calendar periods: C1 and C2

The sequence of physically storing and iterating over the data with calendar as the innermost dimension and location as the outermost dimension would be:

L1/P1/C1
L1/P1/C2
L1/P2/C1
L1/P2/C2
L2/P1/C1
L2/P1/C2
L2/P2/C1
L2/P2/C2

With Calendar being the innermost dimension, data is first processed for all positions in the Calendar hierarchy and for the first position of the other hierarchies. In this example, data would be processed for all calendar positions for the first product and first location. Then all calendar positions for the second product and first location, and so on.

It is recommended that customers order their hierarchies with Calendar as the innermost dimension (required), followed by other hierarchies in order of importance/traversal – most commonly Product, Location, then other hierarchies (if applicable).

The Order column also indicates the order in which the hierarchy information is expected in the file used for measure data loading purposes.

Note: The values “1000” or “1020” are not used as a hierarchy order as they are used internally by RPAS.

- **Security Dimension** – Selecting a Security Dimension for a hierarchy enables position-level security in the domain for the corresponding hierarchy. Any dimension along any hierarchy except the Calendar hierarchy is valid. For example, if the security dimension for the product hierarchy is set to “Dept” (Department Level Security); within the domain, access to departments can be granted or denied by the administrator for individual users, user groups, or all users. If position-level security is to be enabled in RPAS, select the security level. Refer to the *RPAS Administration Guide* for additional information about position-level security.

3. Click the up  button and down  button to move the hierarchy to the desired location, which changes the Order hierarchy property.

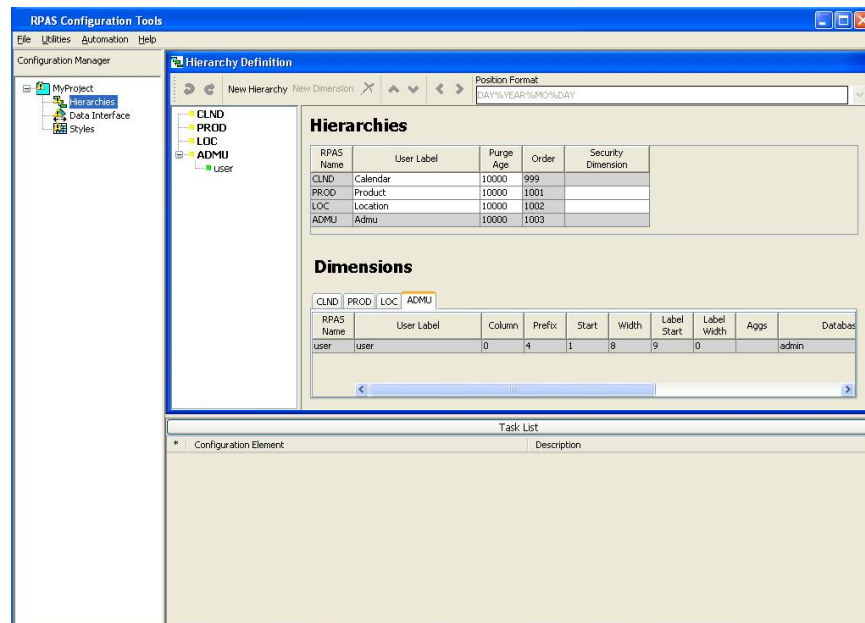
Note: These buttons are used for defining the relationships between dimensions, so do not attempt to use these buttons to re-sequence hierarchies.

You can also reorder the hierarchy by dragging and dropping in the hierarchy tree pane. The order numbers are automatically changed and generated regardless of which technique you use.

Note: The name of a hierarchy cannot be edited if it is shaded gray; however, you can change other attributes (for instance, the label). CLND is always the innermost dimension, and ADMU is always the outermost dimension. The order of the other hierarchies (PROD, LOC, and others that you created) can be changed.

Delete a Hierarchy

Navigate: In the Configuration Manager, select  **Project** –  **Hierarchies**. The Hierarchy Definition window opens in the workspace.

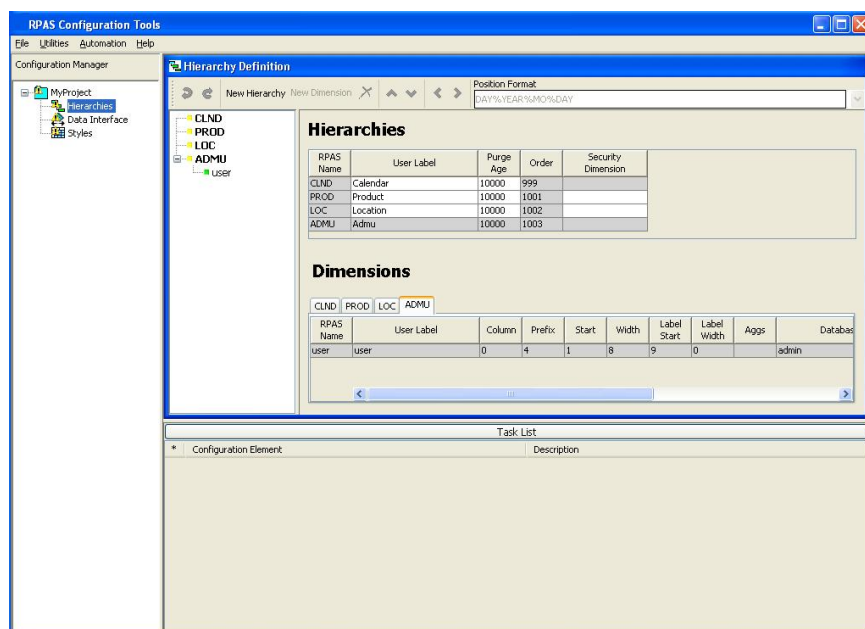


1. In the Hierarchies area, select the hierarchy that you want to delete.
2. Perform one of the following:
 - a. Click the **Delete**  button. The hierarchy is removed.
 - b. Press the **Delete** key from your keyboard.
 - c. Use the right-click menu to select **Remove Selected Item**.

Note: You cannot delete the CLND, PROD, LOC, or ADMU hierarchies.

Edit a Hierarchy

Navigate: In the Configuration Manager, select  **Project** –  **Hierarchies**. The Hierarchy Definition window opens in the workspace.



1. In the Hierarchies area, select the hierarchy that you want to edit.
2. Click on the field that you want to update.
3. Enter the new information.
4. To change the order of the hierarchy, click the up  button or down  button as necessary.

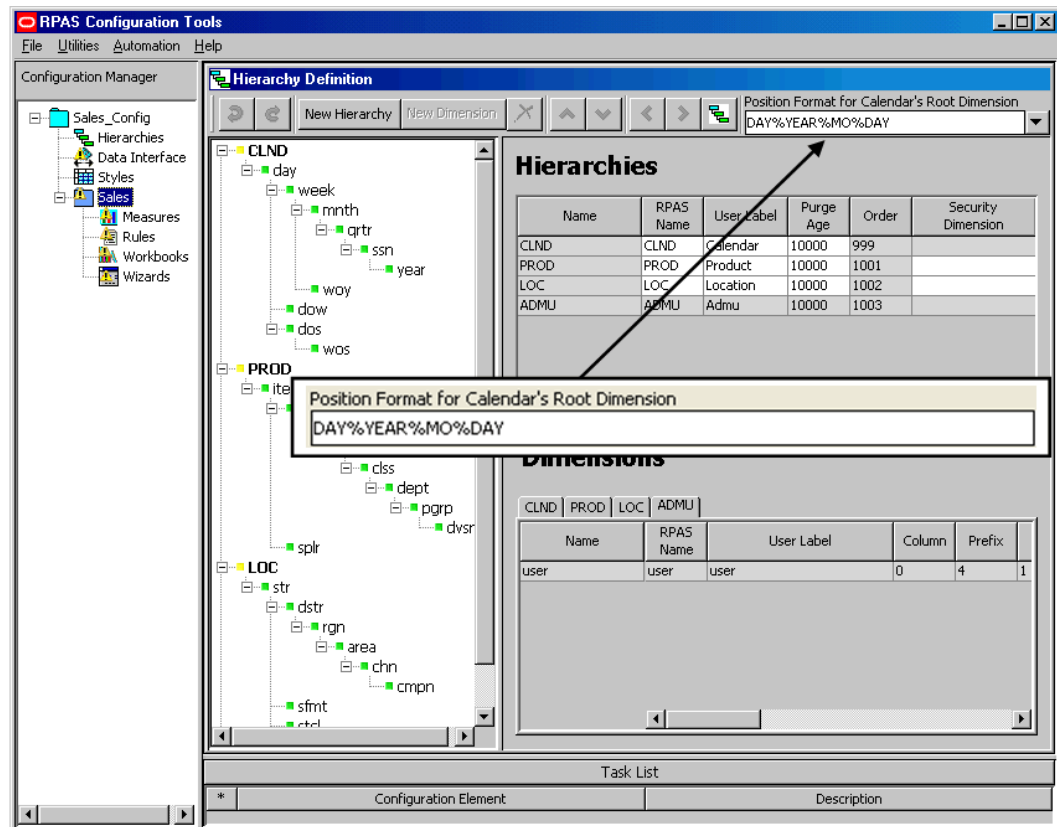
Working with Position Formats

Overview

The Position Format is the date/time format used for the names of positions in the root dimension of the CLND (Calendar) hierarchy (typically 'day'). Positions in the root dimension of the CLND hierarchy need names in a special format for RPAS to map abstract positions to actual dates and times in order to support time-aware calculations. See "Appendix B – Calculation Engine Users Guide" and "Appendix C – Rules Function Reference Guide" for more information.

Specifying the Position Format

Navigate: In the Configuration Manager, select **Project – Hierarchies**. The Hierarchy Definition window opens in the workspace.



Example of Position Format in Hierarchy Definition Window

The Position Format field is on the upper right hand side of the Hierarchy Definition toolbar.

Position Format for Calendar's Root Dimension

DAY%YEAR%MO%DAY

This is a combo box that is populated with some of the more commonly used formats. However, the configuration administrator may also type directly in the combo box if a different format is desired.

Specify the position format as a concatenated sequence of strings and arguments using the following syntax.

Position Format Syntax

- %YEAR – four-digit Gregorian calendar year
- %YR – two-least significant digits of the year (for example, 05 is 2005)
- %MO – two-digit representation of month (for example, 01 is January)
- %MON – three-character abbreviation of the month name
- %MONTH – varying length full name of the month up to nine characters

Note: Even when configuring a solution in another language, RPAS expects the month names and abbreviations (%MONTH and %MON) used in the position names to be in English (for example, Jan, Feb, Mar, and so on).

- %DAY – two-digit representation of the day of the month (for example, 01 is the first day of the month)
- %HR – two-digit representation of the hour of the day (for example, 22 is 10 p.m.)
- %MIN – two-digit representation of minutes past the hour
- %SEC – two-digit representation of seconds past the current minute
- %MSEC – three-digit milliseconds past the current second

The Position Format is NOT case sensitive, so %YEAR is the same as %year.

Note: The resulting position names in the root dimension of the CLND hierarchy must start with an alphabetic character. So the Position Format must either start with a literal string like 'DAY,' or it must start with %MONTH or %MON.

The length of the position name should not exceed 24 characters. The Position Format field performs validation on the Position Format in order to enforce this limitation. For example, the Position Format DAY%YEAR%MONTH%DAY evaluates to a total of 18 characters (3 for the literal, 4 for the year, 9 for month, and 2 for day).

Examples

- Format: DAY%YEAR%MO%DAY
A position that represents the 31st January 2006 would have the name DAY20060131
- Format: d%YR%MON%DAY
A position that represents the 31st January 2006 would have the name d06Jan31

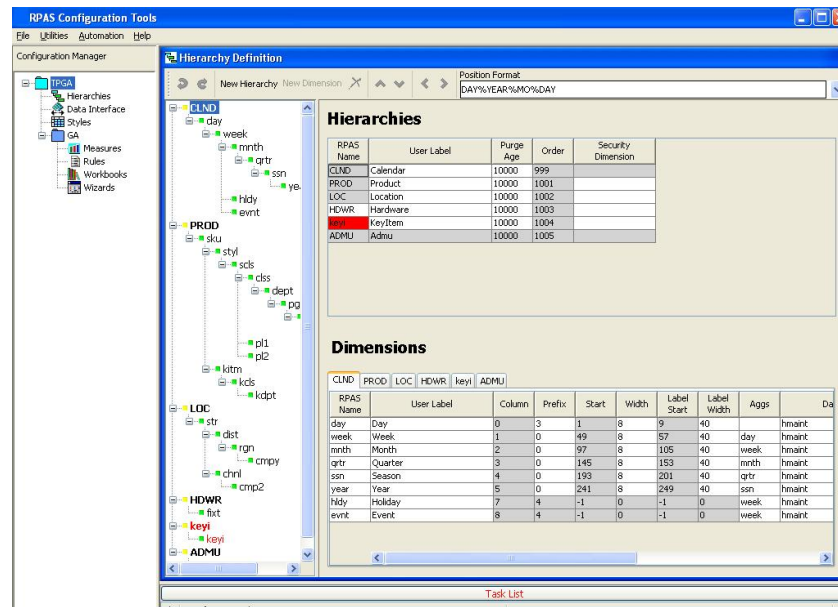
Working with Dimensions

Overview

Dimensions are the components within a hierarchy that define the structure and rollup within a hierarchy. For example, the dimensions for a calendar hierarchy can be day, week, month, and year; or they can be accounting periods.

Create a Dimension

Navigate: In the Configuration Manager, select  **Project** –  **Hierarchies**. The Hierarchy Definition window opens in the workspace.



1. Select the hierarchy or dimension under which to create the new dimension.

Note: When this document uses terms like “top” and “bottom” or “over” and “under,” those terms are to be interpreted visually. For example, the top dimension visually is the root dimension, which is the lowest dimension in the hierarchy. The highest dimensions in the hierarchy are at the bottom end of the hierarchy branches. For instance, Day may be the root dimension of a Calendar hierarchy, but it falls directly under CLND.

2. Perform one of the following options:
 - a. From the Hierarchy Definition right-click menu, select **New Dimension**.
 - b. Click the **New Dimension** button on the toolbar.
 - c. Press the **Insert** key.

You must create the first dimension (which becomes the root dimension) for a new hierarchy when positioned on the hierarchy. Once the root dimension is defined, you cannot add further dimensions under the hierarchy, and you must add them under other dimensions. For example, after you add “Day” to the CLND hierarchy, you are unable select CLND again and add “Hour.” However, you can add “Week” under the “Day” dimension. You can only have one root dimension per hierarchy.

Note: Certain processes that support the purging of data or positions and the mapping of real dates/times to positions require a dimension in the CLND hierarchy that is named “day” and which represents the day level. Such a dimension should be defined, although the user label can be changed from ‘day’ if that is needed for localization purposes.

Specify Dimension Information

Navigate: In the Configuration Manager, select  **Project** –  **Hierarchies**. In the dimension area (hierarchy tree structure), select the dimension that you want to specify information for.

OR

In the Dimension window, select the tab for the hierarchy that contains the dimension that you want to specify information for.

OR

In the Hierarchy window, select the hierarchy that contains the dimension that you want to specify information for.

1. In the Dimension window, select or double-click in the field box to edit. Scroll to the right to see all of the fields. You can also resize the columns by placing the cursor over the column until the double sided arrow appears. Drag the column until the desired width is reached.

Note: Much of the information specified for the dimensions pertains to the data file from which hierarchies and dimensions will be loaded.

2. Other than the RPAS name, the columns can be reordered by dragging and dropping the headings. The column positions will return to the default order when the session is closed.

3. Specify the following dimension information:

- **RPAS Name** – The RPAS internal name of the dimension.

Note: The RPAS name can contain a maximum of four characters; such as SKU, Styl, Clss, and Scls.

- **User Label** – The dimension label that is displayed to RPAS users.

Note: Any string is allowed, but no single or double quotes are allowed.

- **Column** – Identifies the order in which the dimension's positions fall in the meta-data load file. Use the left  and right  buttons to change the order and the column value of the dimensions. Changing the column value will not impact the hierarchy structure or aggregation paths.

Note: The up  and down  buttons are used for defining the sequence of hierarchies.

Example:

Change the column values if the dimensions in the data load file are not in the same order as in the tree structure. Dimensions will be moved up or down in the table without impacting the aggregates.

- **Prefix** – The number of characters (0 to 4) for an automatic prefix to be put on all position names loaded and stripped from all position names exported. The prefix is taken from the internal dimension name. If the number of characters is greater than the dimension name, an underscore is used to pad the prefix. If the number of characters is less than the dimension name, the left-most characters from the dimension name are used. The default is 4. For example, if data loads and exports use numeric SKU identifiers, such as 123456, the internal representation in RPAS could be to prefix this identifier with "SKU," such as SKU123456.

Note: Use of a prefix is optional. There is no prefix if a prefix of 0 is defined.

- **Start** – This field identifies the start position of the position names for this dimension in the hierarchy load file. It is calculated from entries in the width column.
- **Width** – The width of position names for this dimension in the hierarchy load file.
- **Label Start** – Identifies the start position for position labels for this dimension in the hierarchy load file. This field is calculated from entries in the label width column.
- **Label Width** – Width of position labels for this dimension in the hierarchy load file.
- **Aggs** – Establishes the relationship of the dimension with the other dimensions in the same hierarchy. Specifically, this field references the child dimension that aggregates up to this dimension (the parent dimension). This field is not applicable if the dimension is the root dimension. It can be edited by using the drop down list, or you can drag and drop dimensions in the hierarchy pane.

- **Database** – The name of the database in which the dimension information is stored. The default value is “hmaint.” For each position in the dimension, dimension information stored by RPAS includes its internal and external names, label, and the name of parent positions in all higher dimensions.

Note: The database property for a dimension cannot be patched once the domain is built. It is therefore important to establish the databases to hold dimension information correctly when the configuration is first built.

Note: The size of any RPAS database, including databases used to store dimension information, cannot exceed 2 GB. For very large implementations, it may be necessary to store dimension information for different dimensions in different databases.

- **User Dimension** – When selected, this field indicates that the dimension is user maintained. Positions and position mappings (parent-child relationships) for user-defined dimensions are established in the RPAS administrative workbook template, “Hierarchy Maintenance.” This meta-data cannot be loaded like regular (non-user-defined) dimensions in the hierarchy load process.
- **Translate** – When selected, this field enables the position labels for the dimension to be translated into multiple languages (if using a multi-lingual environment, which is set as a Workspace Property for a given project). Positions are loaded into the domain in the native language of the domain via the standard hierarchy load process. Position labels for additional languages are loaded into special measures that are used in multi-lingual domains. With the proper setup, these translated position labels can be displayed in workbooks in the RPAS client instead of the loaded position labels.

Contact Oracle’s Translation department for additional information about multi-lingual domains and localization.

Note: This option must be selected before building a domain for it to take effect.

- **Buffer % Low and Buffer % High** – These properties are used to enable ‘dummy positions’ for the dimension, and to establish the size of the buffer of dummy positions. The appropriate values to use for the Buffer % Low and Buffer % High depend on the rebuffering policy for the implementation. See the *RPAS Administration Guide* for more information.

Delete a Dimension

Navigate: In the Configuration Manager, select  **Project** –  **Hierarchies**. The Hierarchy Definition window opens in the workspace.

1. In the Hierarchy Definition window or in the Dimension Information Frame, select the dimension that you want to delete.
2. Click the Delete  button, or press the Delete key.

Note: If you delete a dimension, all of the dimensions that are structurally dependant on it (its parents, grandparents, and so on) are also deleted. This process cannot be undone.

Note: You cannot delete the user dimension, which is contained in the ADMU hierarchy.

Edit a Dimension

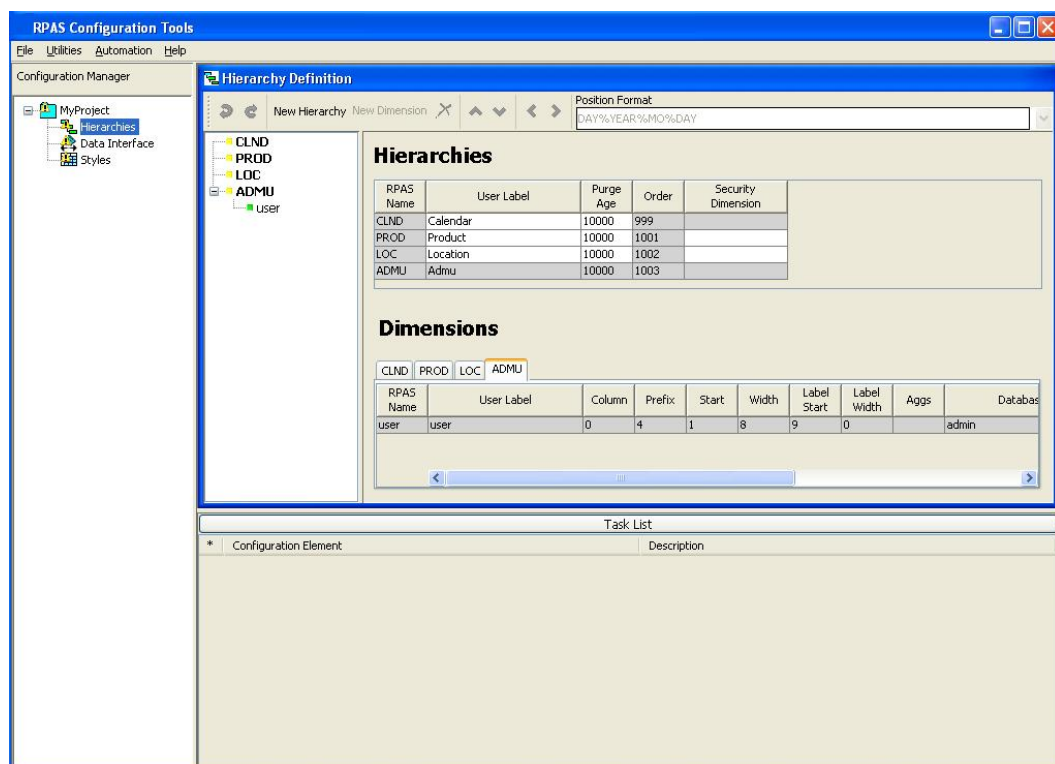
Navigate: In the Configuration Manager, select  **Project** –  **Hierarchies**. The Hierarchy Definition window opens in the workspace.

1. In the Dimensions area, select the dimension that you want to edit.
2. Click on the field you that want to updated.
3. Enter the new information.
4. To change the order of the dimension, click the left  button or right  button to change the order of the dimensions.

Note: The  button and the  button are used to change the order of hierarchies only.

Create a Hierarchy Branch

Navigate: In the Configuration Manager, select  **Project** –  **Hierarchies**. The Hierarchy Definition window opens in the workspace.

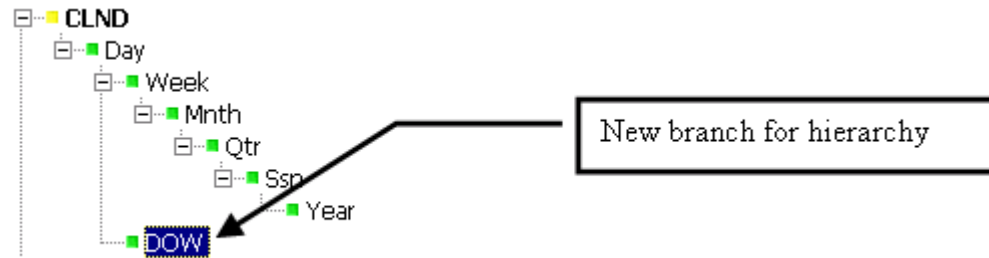


1. In the Hierarchy Definition window or in the Dimension area, select the dimension that will be the base of the hierarchy branch.

Note: Make sure that you select a dimension that will have more than one parent (that is, where the branch starts), and create another parent dimension. Branches can never join together. For example, you cannot have style-subclass-class and style-supplier-class.

2. Perform one of the following options:
 - From the Hierarchy Definition right-click menu, select **New Dimension**.
 - Click the New Dimension button on the toolbar.
 - Press the Insert key.

The new dimension will be created below the selected dimension.



Example of Branch Hierarchy

- Complete the Dimension Information table for the new dimension. The system sets the aggs field to the dimension that was selected to be the base dimension.

Dimension Attributes

Dimension attributes allow for additional information to be defined for the positions of a given dimension. These attributes are defined in the Measure Tool.

Data Interface Tool

Overview

The data interface tool is used to specify information about measures that will be loaded into the domain. This includes the intersection into which data will be loaded and information about the input file.

Note: Data can only be loaded into stored, realized measures in the domain. Therefore, only such measures can be used in the data interface tool. Refer to the section, “Work with measures” (in the Measures overview section of the Solutions topic).

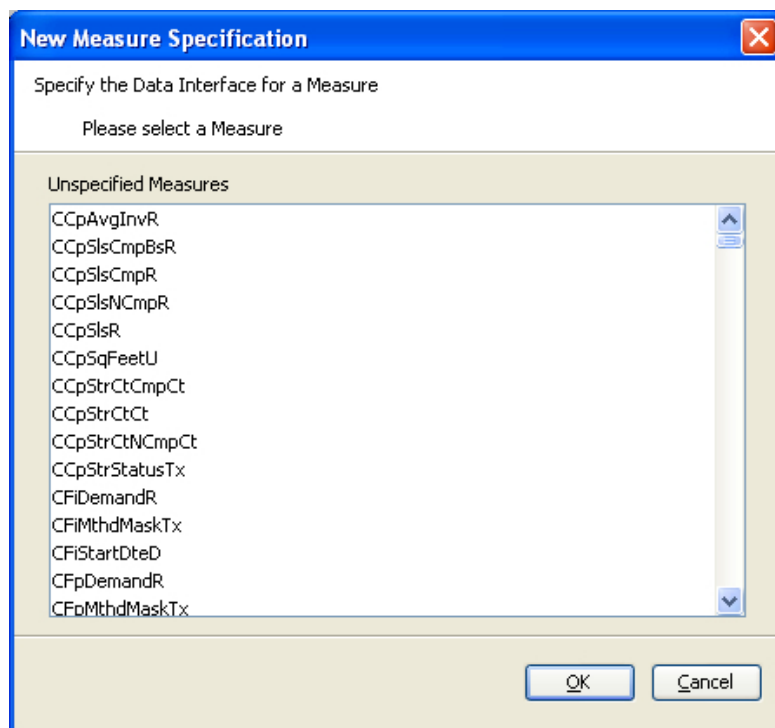
Specify the Data Interface for a Measure

Navigate: In the Configuration Manager, select  **Project** –  **Data Interface**. The Data Interface Manager window opens in the workspace.

Data Interface Manager			
New Meas		Delete Meas	
Measure Name	Load Intersection	File Name	Sta
WpBOPR	Day_scjschnl	measdata	
WpSalesR	Day_scjschnl	measdata	
WpMarkdownR	Day_scjschnl	measdata	
WpReceiptsR	Day_scjschnl	measdata	
WpEOPR	Day_scjschnl	measdata	

Data Interface Manager Window

- Click **New Meas**. The New Measure Specification window opens.

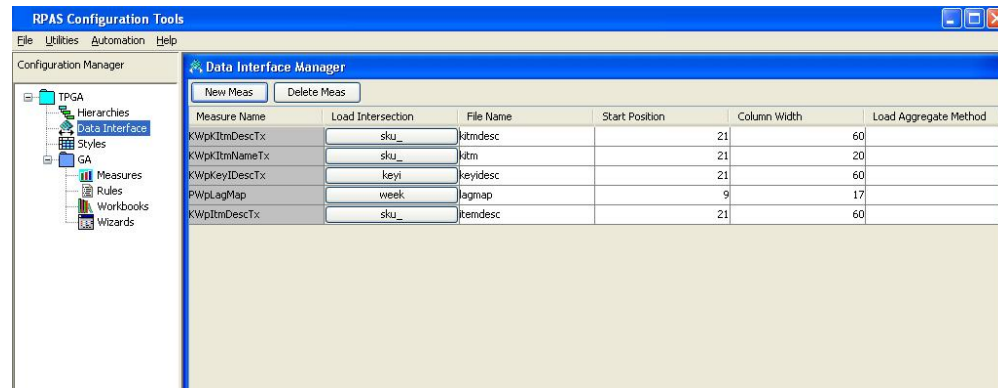


Note: This is a filtered list of all possible measures in the configuration. Measures will be displayed in this list if they are: realized, stored (has a defined database), and not already defined in the data interface tool.

2. Select the measure whose data interface you are defining.
3. Click **OK**. The measure is displayed in the Data Interface Manager window.

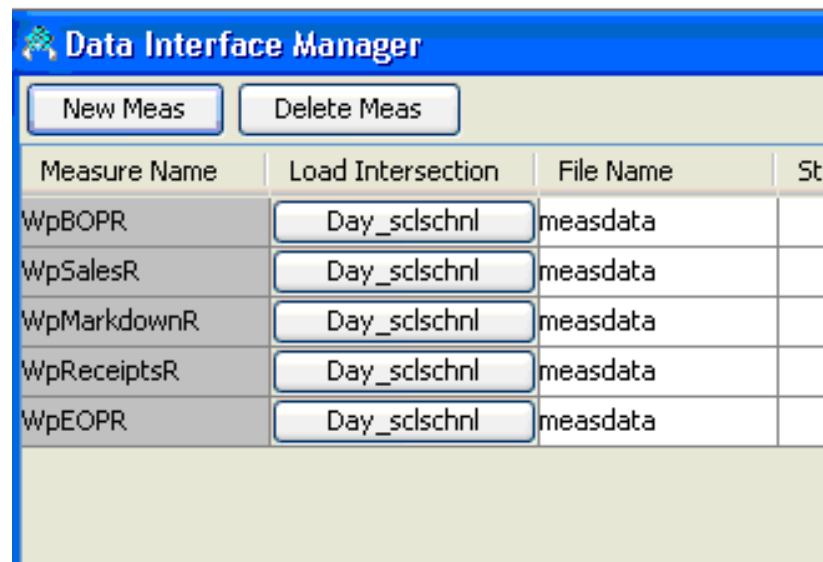
Edit Data Interface Information for a Measure

Navigate: In the Configuration Manager, select  **Project** –  **Data Interface**. The Data Interface Manager window opens in the workspace.



By default, the Load Intersection field is populated with the base intersection of the measure. If the data for a given measure is being loaded at a lower intersection than the base intersection of the measure, this value can be overridden to specify the intersection at which the data will be loaded. For example, you would override this field and select the “day” dimension if the data being loaded is representative of the “day” dimension but into a measure with a base intersection of week.

In this illustration, the Load Intersection is “Day_sclshnl.”



1. Click on the **Load Intersection** field to change its value. The Select Intersection window opens.

Note: The load intersection can only be overwritten to load the data into the measure from below the base intersection.

2. To specify the load intersection:
 - a. Select appropriate dimensions using the list boxes.

Note: Only those dimensions that are at the same level as the base intersection or below will be displayed for the load intersection.

- b. Click **OK** to save any changes and close the window.
3. In the File name field, enter the file name from which data for the measure will be loaded.
4. In the Start Position field, enter the character position in the file where the measure data starts. The start position defaults to the sum of all the dimension widths in the load intersection of the measure +1, and the value specified in the field must be that default or higher.
5. In the Column Width field, enter the number of characters in the file that will contain the measure data.

Note: The column width defaults to 8, but can be changed.

If the Load Intersection was overridden to specify that the data is to be loaded from an intersection below the measure's base intersection, the measure's default aggregation method is used to aggregate the data unless the Load Aggregate field is populated to specify an alternate aggregation method. Select the appropriate aggregation method by clicking on the drop-down box.

Note: Hybrid is not supported for the load aggregation for a measure.

Delete Data Interface Information for a Measure

Navigate: In the Configuration Manager, select  **Project** –  **Data Interface**. The Data Interface Manager window opens in the workspace.

1. Select the measure whose data interface parameters you want to delete.
2. Click **Delete Meas.** You are prompted to confirm your choice.
3. Click **Yes.** The measure is removed from the table.

Working with Styles

Overview

It is possible for the user of the RPAS Client to modify the appearance of the data displayed for a given measure in a grid. Text font, size, and color may all be changed. Many attributes, such as precision (for decimal data types), alignment of the value in the cell, and the cell border may also change.

Using the Style Tool, it is possible to define styles that may be applied to measures. These predefined styles may specify any of a body of attributes that determine the appearance of the data within the client. It is then possible to specify a measure as using one of these pre-defined styles. The measure will then be displayed according to the specifications for that style.

Note: The RPAS Client is not aware of styles which are a configuration convenience. In the client, the individual properties are maintained individually. A style can therefore be thought of as a mechanism to easily set many individual properties.

The Style Tool

The Style Tools provides the following features:

- Allows the creation and management of named styles. New styles are generated as sub-styles of existing styles.
- Allows the specification of the attributes of named styles. Style attributes follow an inheritance scheme in which any unspecified attribute will inherit a value from its parent style, if that style has a specification.
- Allows the specified styles to be visible to the Measure and Workbook Tools, where measures are marked as using a style.

Style Attributes

A number of attributes may be specified for a style. These attributes will determine how the data for a measure that uses the style is displayed within the RPAS Client. Style attributes follow an inheritance framework in which an attribute defined in one style will be defined for all of that styles children if they do not override that specification. The attributes of a style that may be specified are:

- **Name** – The name of the style. This is used in the Measure and Workbook Tools to assign a style to a measure. Since styles are a configuration convenience, style names are not visible in the RPAS client.

- **Prefix** – A cell value in the RPAS client will be prefixed with this string. For example, a prefix could be “\$” to denote US currency values. The prefix can be any character sequence, not exceeding seven characters.
- **Suffix** – A cell value in the RPAS client will be suffixed with this string. For example, a suffix could be “%” to denote that the value in the field is a percentage of something. The prefix can be any character sequence, not exceeding seven characters.
- **Scale Factor** – A cell value in the RPAS client could use a scale factor for display purposes. A value that is calculated as a fraction could be displayed as a percent by selecting the scale factor to be 0.01 (the UI divides by the scale factor).
For example, if the value in a cell is 0.5, the scale factor would have to be 0.01 for the cell to display 50.

The value entered in the field should be greater than zero.

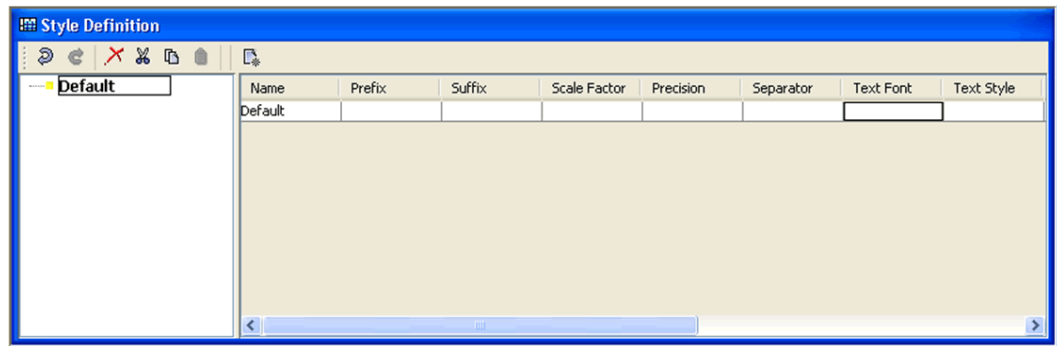
- **Precision** – Precision is the number of significant digits to be displayed in the cell of the RPAS client. Setting this value to 3 indicates to the client to always display 3 positions after the decimal. The value entered in the field should be greater than zero.
- **Separator** – A cell value in the RPAS client could be formatted to have separators in the value. The separator and the format are defined by the regional settings on the computer. For example, when a separator is used, a value of 1000 would be displayed as 1,000 or 1.000 or other formats depending on the regional settings.
- **Text Font** – The font of cell value in the RPAS client (for example, “Times New Roman,” “Arial,” and so on).
- **Text Style** – The display style of the text value in the RPAS client (for example, “Bold,” “Italic,” and so on).
- **Text Size** – The font size used to display the cell value in the RPAS client if (for example, 12 pt).
- **Text Color** – The color of the cell values in the RPAS client.
- **Background** – The background color of the cells in the RPAS client.

Note: In the RPAS client, the measure formatting background color for a measure takes priority over the ‘read/write’ background color that can be set for the application. Therefore, the RPAS client ‘read/write’ color will not be seen if styles are used through the tools. If a specific read/write color is desired for all measures, set it as the background color of the default style, and do not override it for any other styles. On the other hand, the RPAS client ‘read only’ background color takes priority over the measure formatting background color so that ‘protection processing’ will be visible.

- **Alignment** – Alignment of cell values within the cells in the RPAS client (Left, Center, and Right).
- **Border Style** – The cell values could have predefined borders in the RPAS client. Border style determines the kind of borders (like single line, dotted line, and so on) and where the borders should be relative to the cell value (top, bottom, left, right, or any combination of these).
- **Border Color** – The color of the border lines for cell values.

Create a Style

Navigate: In the Configuration Manager, select  **Project** –  **Styles**. The Style Definition window opens in the workspace.



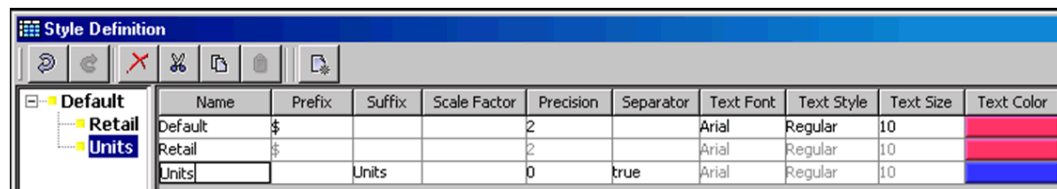
Style Definition Window

1. Select the Style that will be the parent of the new style. All styles must ultimately be descendants of the Default Style.
2. Perform one of the following:
 - Click the Create style button in the toolbar.
 - Select **New Style** from the right-click menu.
 - Hit the Insert key.

If the Style Attributes for Default are populated, all of its descendants will inherit the same attributes unless the configuration administrator specifies new attributes for the new styles. A new style will be created with inherited attribute values for all the properties set in its parent style'.

The inherited style attribute values are displayed with lighter shade (gray) to differentiate from the un-inherited values (black). Notice that the style attributes for the style 'Default' -are shown in black while the style attributes for the style 'Percent' are in gray.

3. Change the values of any of the new style's attributes where a different value is required than that which has been inherited. For those attributes that have been overwritten from the "Default" value will be display in black while those that have not been changed will remain gray to indicate they are inherited.

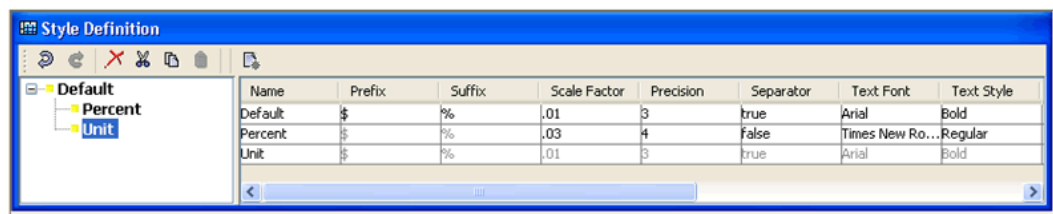


Remove a Style

Navigate: In the Configuration Manager, select  **Project** –  **Styles**. The Style Definition window opens in the workspace.

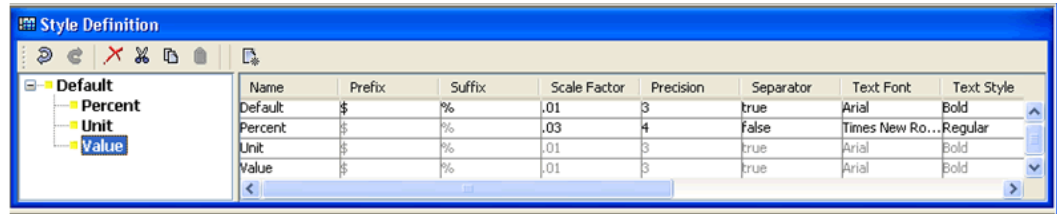
1. Select the Style to be removed.
2. Perform one of the following
 - Click the **Delete Style** button in the toolbar.
 - Select **Remove** from the right-click menu.
 - Press the Delete key.

The selected style and all of its child styles will be removed from the style description tool. Any measures using a deleted style will be displayed as invalid.



Edit a Style

Navigate: In the Configuration Manager, select  **Project** –  **Styles**. The Style Definition window opens in the workspace.



1. Select the Style to be edited.
2. Select the property of the style to be edited. Depending on the property selected, one of the following will be displayed:
 - a pop-up color chooser (for all color selection properties like font color, background color, and so on)
 - a pop-up dialogue (for Borders)
 - a drop down (for alignment, font, and text style)
 - a free flow text cursor (for all other properties)
3. Make the selection, and enter the value for the property.
If the edited value is changed to the same as its parent's value for an attribute, the value is automatically changed to inherit from the parent, and it is displayed as gray rather than black.
4. If the value is to be deleted to not contain any value, select the property, and hit the **Delete** key.

Solutions

Working with Solutions

Overview

A solution corresponds to an application configuration (for example, Merchandise Financial Planning or Item Planning). Each project can contain one or more solutions. For each solution, you define measures, rules, workbooks, and wizards. Once you create a solution, you can move it from one project to another.

Note: Some solutions, such as Curve, Forecast, and Promote, have configuration procedures that are specific to those solutions. For more information, see the RPAS Solution Extension Guide.

Create a Solution

Navigate: From the File menu, select **New – Solution**. The New window opens.

The screenshot shows a 'New' dialog box with a blue title bar. The main area is light beige. At the top, it says 'New Solution'. Below that is a red stop icon and the text 'Application name has not been specified.' There is a text field labeled 'Name' and a list box labeled 'Project' containing 'MyConfig'. Below the list box is a message box that says 'Select project for new solution.' At the bottom are 'Ok' and 'Cancel' buttons.

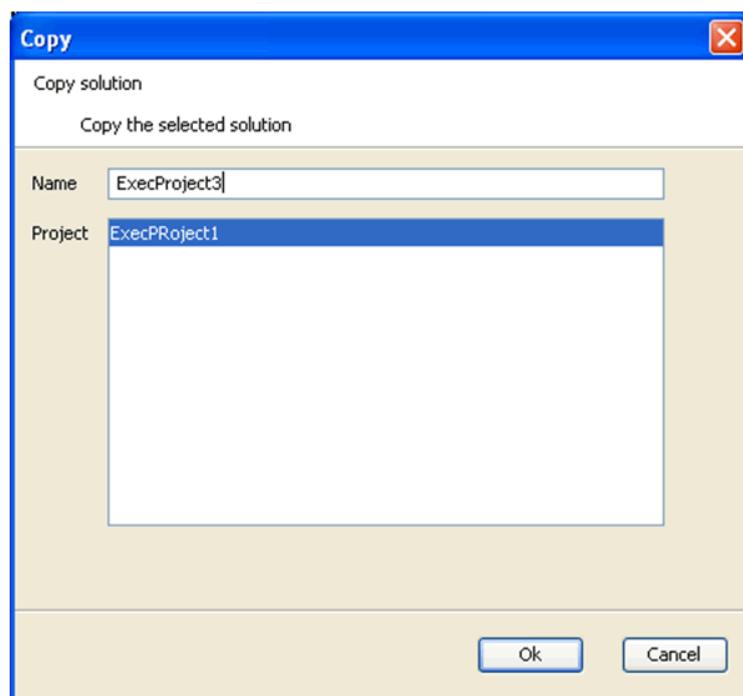
New Window

1. In the **Name** field, enter the name of the solution.

2. In the **Project** field, select the project in which the solution is to belong. There will be multiple Projects listed if multiple Projects are currently open.
3. Click **OK** to save any changes and close the window.

Copy a Solution

1. Select the solution to be copied.
2. Right-click in the Configuration Manager, and select **Copy**. The Copy solution dialog box appears.



Copy Solution Dialog Box

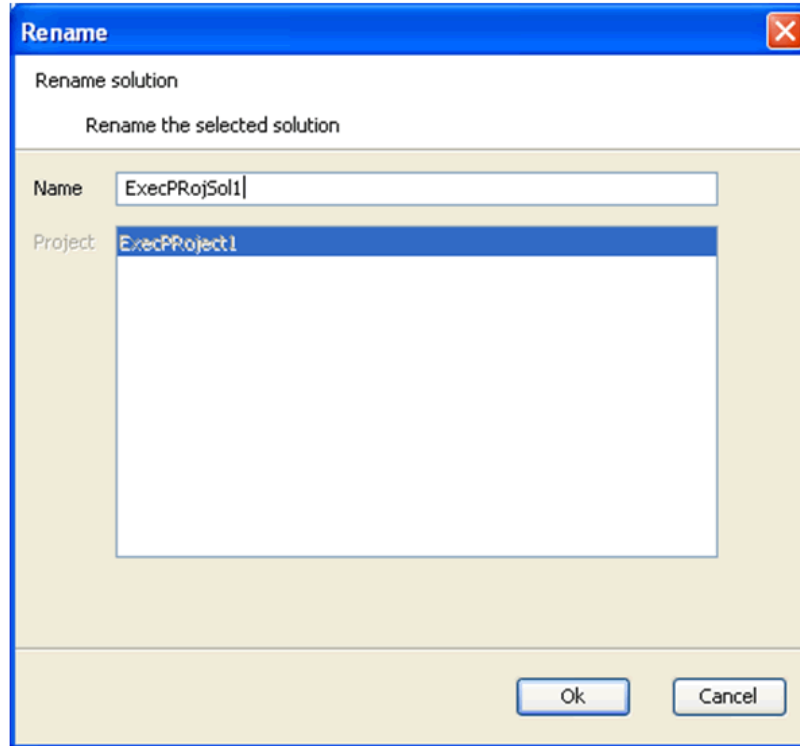
3. Type the new name for the solution in the text box.

Note: This is the name that the solution is called after it has been copied.

4. Select the project where the solution is to be copied.
5. Click **Finish** to save the copied solution in the specified project.

Rename a Solution

1. Select the Solution to be renamed.
2. Right-click in the Configuration Manager, and select **Rename**. The Rename dialog displays.



Rename Dialog Box

3. Delete the old name from the resulting field, and type the new name.
4. Click **OK** to save the new name.

Move a Solution

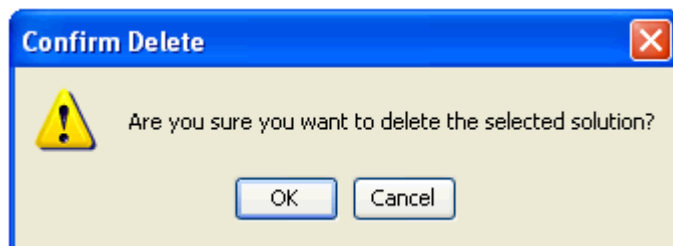
1. Select the solution that you want to move.
2. Right-click in the Configuration Manager, and select **Move**.
3. Type the name of the solution in the text box.

Note: This is what the solution is named after it has been moved.

4. Select the project where you want to move the solution.
5. Click **Finish** to move the solution to the specified project.

Delete a Solution

1. Select the Solution to be deleted.
2. Right-click in the Configuration Manager. The Confirm Delete dialog box appears.



3. Click **OK** to complete the deletion.

Measures and Components

Overview

The measure definition tool allows you to define components of measures and to specify default properties for each component. You can also specify properties for individual measures, which will override the component default properties for those measures.

A major component is the highest level in the component inheritance hierarchy. Properties defined at this level are inherited by all minor components created under a major component.

Within each major component, you create one or more minor components. You can also create a minor component underneath a minor component. You can also modify properties at the minor component level.

Once you have defined your major and minor components, the measure definition tool generates 'prototype measures' that are based on the combination of components that you select. These prototype measures cannot be used elsewhere in the configuration tools until you have realized them. See *Realizing and Unrealizing measures* for more information.

Component Process
Create major components, from which measures are composed.
Create minor components, which are sub-groupings or specific items in a major component.
Define measure properties at the major component level. The minor components will inherit the properties associated with the major component they belong within.
If necessary, modify the measure properties at the minor component level.

Note: Components have no structural impact on the built solutions, and they are not exposed to end users. Components are intended to be a convenience to aid you in easily grouping measures together and setting measure properties at higher levels.

Alerts

Alerts are an exception management tool. An alert is a measure of type Boolean (returning a value of true or false) that is the result of the evaluation of a business rule. RPAS then notifies you of the “true” conditions and it allows you to build workbooks to resolve the scenario that drove the alert.

Example:

A store’s inventory on a particular item is low, so an alert will be triggered (Boolean expression = true).

A summary of the process for defining and finding an alert is as follows:

1. Create an alert measure. This must be a Boolean measure (true-false, yes-no) and must be defined in the domain using the RPAS Configuration Tools.
2. Create the alert (the expression) for which the alert should be evaluated using the Configuration Tools. This flags the registered measure as an alert so that it is recognized when the “alert finder” is run.
3. Run the “alert finder” on the domain to evaluate the number of instances when one or more alert expressions are true. This operation is completed using the RPAS utility `alertmgr`.

For more information on alerts, see the *RPAS Administration Guide*.

Measure Properties

Inheritance

Measure properties are inherited at the component level. The properties defined for a component are inherited by the minor components that belong to that component and to the measures that are associated with that component, unless it is overridden at a lower level.

When a measure can inherit a property from more than one of the components that construct it, the measure inherits from the component that belongs to the highest major component. In other words, the major component that was created first, and is sequenced nearest to the top of the component tree in the measure manager tool. For many properties, it is a good practice to set the properties for just one major component or for minor components in just one major component branch.

Overriding

Measure property inheritance can be overridden at the minor component or at the measure level. Once a property is set at a lower level, changes made to that property at a higher level will no longer be inherited by that measure.

Measure Validation within the Measure Tool

The Measure Tool performs large amounts of validation on the properties of the measures that are created within it. Much of this validation involves dependencies of one property of a measure upon another property of the measure. Some validation involves dependencies of a property of a measure upon the hierarchies that are defined in the Hierarchy Tool. These forms of validation are performed automatically as edits are made in the Measure Tool.

In addition; the validity of rules, rule groups, and workbooks heavily depends on the properties of the measures that they contain. Validation of the measure content of the rules, rule groups, and workbooks of a large Solution can take significant amounts of time.

In order to facilitate the configuration process, this second form of validation does not occur as edits are made in the Measure Tool. Instead, this validation is deferred until you leave the Measure Tool by selecting a different tool from the Workspace Manager. At this time, a dialog appears that informs you of the validation in process. When the full suite of Solution level measure content validations is complete, the dialog goes away and the tool that you selected is activated.

In some cases, you may desire to disable this full validation (for instance, when you perform a number of changes to measure properties that require switching back and forth between multiple tools). In such cases, it is possible to disable the automatic validation. Validation can then be manually initiated in the Rule Tool by hitting the toolbar button, "Perform measure content validation."

To turn off real-time validation, uncheck the "Measure Content Validation" option in the Tools Preferences of the File menu.

Working with Components

Measure Component Design

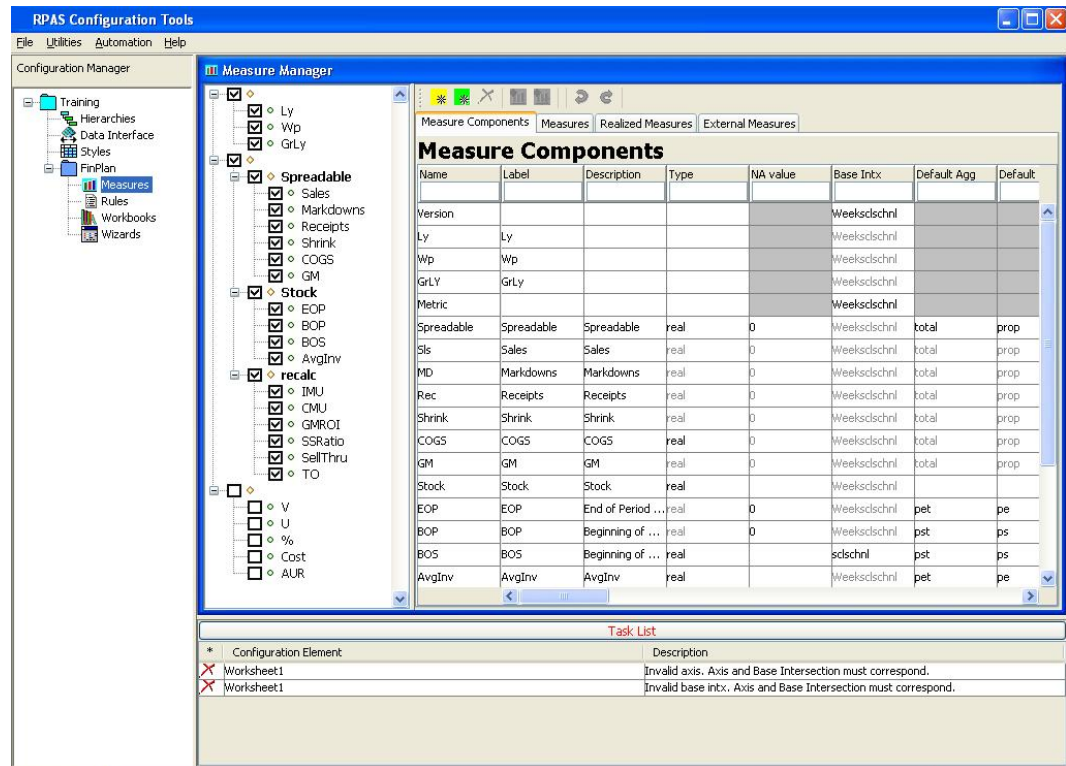
The following two basic principles should be kept in mind to make the measure tool as powerful as possible.

1. Major and minor components should be designed with the idea of maximizing the inheritance of properties, and minimizing the amount of property overriding.
2. Use minor components to make measure definition manageable.

For example, consider a configuration that has 2000 measures and 1500 of the measures are of data type real. Avoid grouping all 1500 measures into a single minor component because you cannot edit smaller subgroups of 1500 measures very easily. Minor components can also have minor components, so within the 1500 measures, you could break them out further. This could be based on the aggregation method, such as total, max, recalc, and base intersections. Ideally, filtering by the "checking" of a lowest level minor component should allow you to easily view and manage every resulting measure for that minor component.

Create a Major Component

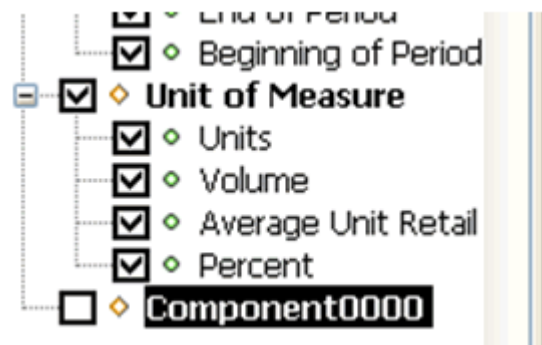
Navigate: In the Configuration Manager, select **Project – Solution – Measures – Measure Components** tab. The Measure Manager window opens in the workspace.



Measure Manager Window

Right click in the left hand pane of the Measure Definition window and select **Add Major Component**, or click the **Add Major Component** button.

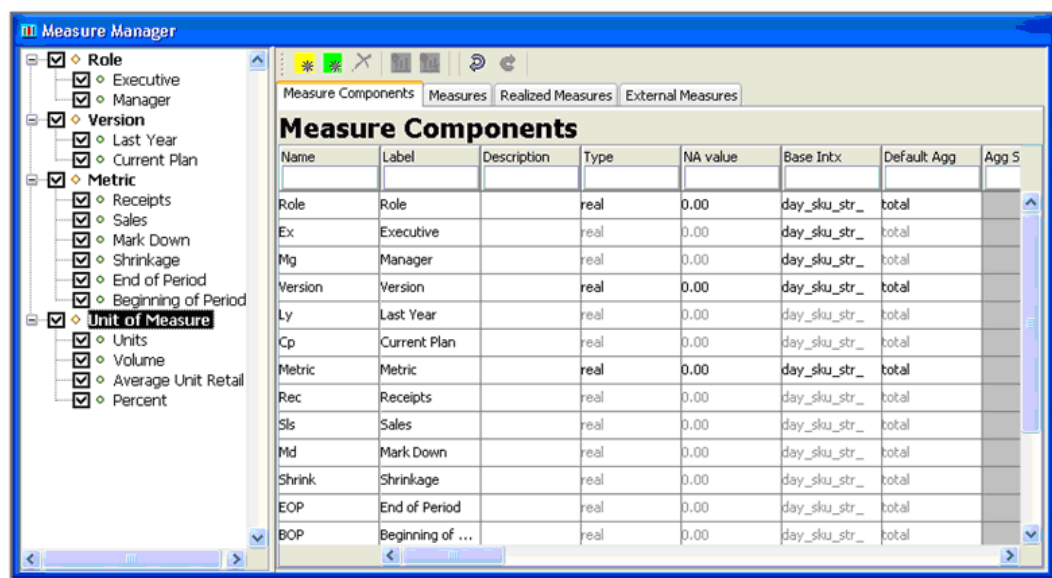
The major component is displayed in the left-hand pane.



Example of New Major Component

Create a Minor Component

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Measures** – **Measure Components** tab. The Measure Manager window opens in the workspace.



Example of Major Measure “Unit of Measure” Selected

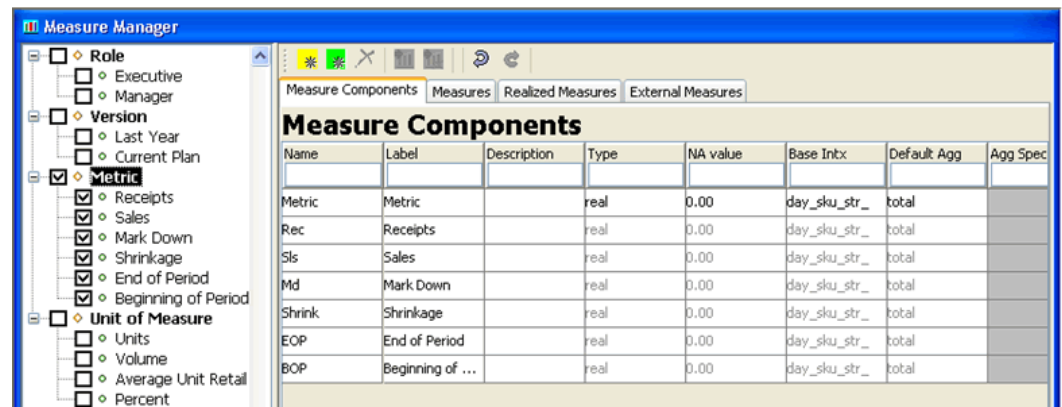
1. Select the major or minor component that the new minor component is to be added beneath.
2. Choose one of the following methods:
 - Right-click in the left hand pane of the Measure Definition window, and select **Add Minor Component**.
 - Click the **Add Minor Component**  button.
 - Press the **Insert** key.

The minor component is displayed in the left hand pane with a default name. The configuration administrator may change the name (using Rename in right-click menu) or change it in the component properties pane.

Specify Component Information

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Measures** – **Measure Components** tab. The Measure Manager window opens in the workspace.

1. In the left hand pane, select the major or minor components to view in the component pane. In the following example, Metric is selected. Only the properties for “Metric” and its minor components are displayed.



Example of Major Measure "Metric" and selected Minor Components

2. Specify the following information, which will apply to the measures that inherit property values from the component:

Note: Not all properties need to be entered for all components. Where the same property is inherited from two or more components, the property used for a measure is that from the component that is highest up the list of components.

Note: Properties that are grayed out in the component properties table cannot have a value in the current context. Typically, this is based on the data type of the measure. For example, only components of Boolean data type can have an alert category or an alert expression.

Component Properties

Name

The name (identifier) of a component is used to identify the component within the configuration tools, but it is not used in the configured solution. Measure names, which are used in the configured solution, are built by concatenating the names of the components that the measure is built from. They are concatenated in the sequence that the components appear in the list of components.

Label

The label of the component is used to generate measure labels in a similar way that measure names are generated. These are displayed to RPAS users. There is no maximum size limit, but you should keep grid display limitations in mind when creating a measure label.

Description

A description of the component is used to generate measure descriptions in the same way that measure labels are generated. This is used for information within the configuration tools only and can be a long description.

Type

The Data type. Select one of the following:

- **Real** – Floating point numeric values. Most measures are of this type.
- **Int** – Numeric integer values. There are no special spreading algorithms for integer measures, which should normally be used only for measures that are calculated “bottoms up.” Formatting can be used to display real measures as integer value in the RPAS client.
- **Date** – Date and time. This can easily be converted to position names using standard RPAS functions.
- **String** – Variable length strings, which are typically used for notes and names.
- **Boolean** – True or false values, which are typically used for flags and indicators.

NA Value

This is a value (typically zero for numeric measures) that is not physically stored, but is inferred. It is used to help with storage and calculation efficiency, and it may be changed by RPAS (in full-evaluation mode) if better efficiencies can be obtained with a different value. See “Appendix B – Calculation Engine Users Guide” and “Appendix C – Rules Function Reference Guide” for more information.

Base Intx

The Base Intersection. The lowest level at which data is stored for a measure. In the domain, the measure is only stored at the base intersection. Inside a workbook (for performance reasons), values for the measure may be stored above the base intersection. Nevertheless, whether stored or not, values for aggregated levels may be viewed in a workbook and used in calculations in workbooks or domains. Double-click in the field to open the Select Intersection window. The hierarchies that were defined using the Hierarchy tool are displayed. One dimension from each hierarchy can be selected, but a dimension is not required for each hierarchy. Alternatively, a measure may be marked as scalar. A scalar measure has only one value at any combination in the positions of dimensions of the domain.

Note: If a measure does not have a common dimension from a hierarchy in its base intersection, it implicitly belongs to the “all” position at the top of that hierarchy.

Default Agg

The default aggregation method should be selected from the valid aggregation methods for the data type. See “Appendix B – Calculation Engine Users Guide” and “Appendix C – Rules Function Reference Guide” for more information.

Default Spread

The default spread method should be selected from the valid spread methods for the data type. See “Appendix B – Calculation Engine Users Guide” and “Appendix C – Rules Function Reference Guide” for more information.

Note: The spread method is the default spread method, which you can override on edit in the RPAS User Interface. For all populated spread methods (ending with “pop”), the spread method is the same as the underlying method (for instance, prop_pop is like prop), except that only cells with a value that is different from the naval are used in the spreading, and cells with a value equal to the naval are ignored.

Base State

The permission attribute of the measure at the base intersection (read or write).

Agg State

The permission attribute of the measure at all intersections above the base intersection (read or write).

Note: Set the Base State to write and the Agg State to read for those measures that need to be modified, but where there is no business requirement to manipulate them other than at their base intersection. Usually there is no sensible way to spread such measures.

Note: The manipulability of measures will change according to “protection processing” principles. Therefore, base state and agg state should only be used to override the result of protection processing (for example, to make a measure non-editable that protection processing would otherwise allow to be modified). See the “Appendix: Calculation Engine User Guide” for more information.

Database

The physical location in the file system of the database that stores the information for this measure. Those measures that contain information that persists beyond the lifetime of a given workbook store their information within a database within the RPAS domain. This field is used to specify the path to the location of the database to use for the measure. All databases are contained within the data directory of the domain. If the specification does not begin with the data directory, “data/” will be appended to the beginning of the entry at the time of installation (for instance, the entry “sales” will be registered as “data/Sales”).

View Type

The View Type field holds properties for two types of measures: those that are calculated when viewed, and those that are synchronized with other measures. If the view type is none, the measure is of neither type.

If the View Type is `view_only`, the measure is not calculated during a normal calculate cycle, and it is calculated on-the-fly when required (for instance, for viewing). Such measures must have an aggregation type of `recalc` and should appear on the left hand side of only one expression in a rule group. They may not appear on the right hand side of any expressions. The measure should not have a database assigned. See “Appendix B — Calculation Engine Users Guide” for more information.

Synchronized measures are, in effect, views of two or more other measures where changes and locks to those other measures are immediately reflected in the synchronized measure (and vice versa). For example, a “closing stock” measure may be synchronized with a “season opening stock” measure and an “opening stock” measure so that a change to “opening stock” in week 3 will immediately cause the same change to be applied to “closing stock” in week 2 (since closing stock in week 2 and opening stock in week 3 are the same). Synchronized measures require a synchronization type in the View Type property, which must be one of `sync_first_lag`, `sync_lead_last`, `sync_first` or `sync_last`, and a list of measures to synchronize with in the Sync With property.

- **none** – The measure is calculated normally.
- **view_only** – The measure is calculated when viewed.
- **sync_first_lag** – Period 1 is from the first measure (no calendar). Periods 2..N are from the second measure 1..N-1 (lag) [for example, bop synchronized with os and eop].
- **sync_lead_last** – Periods 1..N-1 are from the first measure 2..N (lead). Period N is from the second measure (no calendar) [for example, eop synchronized with bop and cs].
- **sync_first** – Period 1 from the measure (similar to pst along calendar dimension) [for example, os synchronized with bop].
- **sync_last** – Period N from the measure (equivalent to pet along calendar dimension) [for example, cs synchronized with eop].

Sync With

A comma-separated list of measures used for synchronization, which depends on the View Type assigned.

Insertable

Indicates whether the measure can be inserted as an extra measure in workbooks built from templates that are not configured to contain the measure. Insertable measures can be added to a workbook during the wizard process on the “Extra Measures” wizard page before a workbook is built, or by inserting the measure in the Show/Hide dialog window in the RPAS client inside a built workbook. Measure security must also be defined for Insertable measures in the RPAS Security Administration workbook template. Possible values are true and false. See the *RPAS Administration Guide* for additional information about measure security.

UI Type

Indicates whether the measure is a picklist or not. If a measure is defined as a picklist, UI users must choose a value for the measure from a list of valid values defined using the range property. Possible values are “picklist” or blank.

Range

Specify an allowed range for the measure at edit time. For numeric values, the syntax is as Lower Bound : Upper Bound. If you attempt to enter a value in the RPAS client outside of this range, the value is rejected.

For numeric or string measures with a UI Type of “picklist,” values are comma separated value/label pairs, where the label is given in brackets, such as a(labela), b(labelb), and c(labelc). If a label is not specified (for example, “a, b, c, d”), the value is also used as the label. The value of the cell is used in calculations; however, labels (if specified) will be displayed in the user interface, both in the grid and in the picklist. If a cell contains a value that is not valid for the picklist, the value is displayed in the grid. When the measure’s range is specified in this manner, all cells in the RPAS client will display these same values as valid options for the picklist.

The valid set of options for a picklist measure can also be defined in such a way that they are ‘context sensitive,’ which means that they vary from position to position. For example, a picklist measure with a base intersection at the SKU dimension could have valid values that vary according to which class the SKU belongs to. You set this up by setting the range property of the picklist measure as “measurerange = measS” where measS is the name of a string measure that holds the valid picklist options (in the valid formats described earlier) in each of its cells. The measure that holds the valid picklist values (in this example, “measS”) can have a base intersection at any of the dimensions in the hierarchies and the values shown in the picklist measure for any intersection are effectively “looked up” using normal ‘nonconforming measure’ handling.

The valid values for a picklist for a cell are referenced from a measure dynamically. If required, it is possible for the valid values of picklists to change during the life of the workbook as a result of calculations or end-user edits. The value used will always be that of the last calculate, so direct or indirect (through calculation) edits to the picklist value measure are ignored when a calculation is pending.

Purge Age

The number of days (without a load) before measure data is purged. See the *RPAS Administration Guide* for details of how this property is used by the LoadMeas utility.

Lower Bound and Upper Bound

If the range of valid values for a numeric (real or integer) measure applies across the whole domain, the range property can be used to specify valid values for data entry validation. If the range of valid numeric values varies according to positions in the hierarchies, the Lower Bound and Upper Bound properties must be used instead. If specified, Lower Bound and Upper Bound must each be the name of a valid, realized numeric measure that provides the bounding values for this measure's data cells. These properties must contain measure names (not numeric values).

Note: If one (but not the other) of Lower Bound and Upper Bound is specified, only one limit is checked. For example, if a Lower Bound is set, but an Upper Bound is not, valid values are greater than or equal to the value held in the Lower Bound measure, but with no upper limit. The Lower Bound and Upper Bound measures can be non-conforming with respect to the measure that has bounds, and the value to be used will be obtained by normal non-conforming processing (that is, “replicated down” from higher levels or “aggregated up” from lower levels).

Sp Value Type and Sp Value

These two properties specify the “special value” type and the “special value” value. They are used to define the manner in which ‘special values’ are handled by RPAS. These two properties can be used together to specify how to display cell values on the User Interface (UI) that have a value equal to the naval of the measure. In particular, it supports solutions that want to interpret cells with the naval as meaning ‘no value’ by displaying a null value to the end user.

The SP Value Type property specifies the type of value that will be shown. Valid values for this property are Null, Cell Value, and User Entered. The default behavior is that such cells will have their cell value shown, which is what the value of Cell Value in the SP Value Type property means. However, these properties can be used to override the default to either show null, which is defined below, or to show a specific value. To configure the values to display null, put the value Null into the SP Value Type property. When null is configured, the cell will be blank in the UI when the value equals the naval for a numeric, a date, or a string measures. For Boolean measures, it will be a grayed-out check box. When a specific value is required, select **User Entered** for the SP Value Type, and enter the value to be displayed in the SP Value.

- For the Sp Value property; the entry in this property must be of the same data type as the measure, and validation is enforced in this field.
- For a Boolean measure; when the Sp Value Type field is set to User Entered, the only valid entry for the Sp Value field is either true or false.
- For a Date measure; when the Sp Value Type field is set to User Entered, you must enter a date in the format of YYYYMMDD.
- By default, each measure will be registered with ‘Cell Value’ as the Sp Value Type. For cases where a Sp Value Type setting other than ‘User Entered’ is used, but a Sp Value entry is provided, the Sp Value entry will be ignored. Instead, the measure will be registered with ‘Cell Value’ as the default behavior.
- When you set the Sp Value Type setting to ‘User Entered,’ but do not provide a Sp Value entry, the Sp Value Type will be ignored. Instead the measure will be registered with ‘Cell Value’ as the default behavior.
- When a domain that includes a measure with a special value setting has already been built, and you remove that special value setting for that measure and patch the domain, the measure will get updated with cellvalue as the default special behavior.
- RPAS allows for the special value measure property to be updated.

Dim Attr Type

A checkbox indicating that the measure should be registered as a dimension attribute. Dimension attributes allow for additional information to be defined for the positions of a given dimension. This is commonly used to define and display an alternate label for a position (other than the loaded position label) or to display supplemental information about a position (such as the status of a given position).

For a measure to be eligible to be a dimension attribute, the following requirements must be met:

- The measure must be realized
- The measure must be 1-dimensional, which means that its base intersection must only have one dimension
- The measure must be stored, which means that it must have a defined database

Dim Attr Name (optional field)

This is only to be used if the measure is set to be a dimension attribute measure. When a dimension attribute is displayed within the RPAS Client, the Dim Attr Name will be used in place of the measure name. If no Dim Attr Name is supplied, the client will display the measure name.

Dim Attr Label (optional field)

This is only to be used if the measure is set to be a dimension attribute measure. When a dimension attribute is displayed within the RPAS Client, the Dim Attr Label will be used in place of the measure label. If no Dim Attr Label is supplied, the client will display the measure label.

Allowed Aggs

The set of the allowable aggregation methods for the measure based on the measure data type. You can add extended measures to UI views that are normal measures, but with aggregations based on different aggregation methods. The aggregation methods that are available for selection for this purpose are based on the Allowed Aggs of the base measure. The same base measure can have multiple extended measures based on different aggregation methods.

Style

One of the styles defined within the style tool may be specified as the default style to be used to display this component inside the UI. See the section on the Style Manager for details on the specification of style information.

Alert Category

Identifies the measure as an alert measure and associates the category with this measure. Only measures of data type Boolean can be alert measures. All defined categories are displayed in the drop-down box. Note that alert categories are defined in the Alert Manager, which can be accessed by right-clicking a measure and selecting Alert Manager. See the *RPAS Users Guide* for a description of alerts.

Alert Expression

The evaluation method for this alert measure. When you double-click on this cell, a dialog box opens that can be used to select a rule or a rule group. Rules displayed will be those whose first expression has the given measure on its left hand side. Rule groups

displayed will be those that have rules whose first expression contains the given measure on its left hand side (see Alerts; also see the *RPAS Administration Guide*).

Single Hier Select

This property is only valid for components of data type “string.” It specifies that the contents of cells are to be entered by users using the Single Select Widget. This Widget on that presents a view of the hierarchy that allows you to select a single position. The value of this property is the dimension from which you select a position. You select this by double-clicking on the property field and selecting the hierarchy and dimension from the resulting dialog.

Filename (read only)

Measures included into the Data Interface Tool have a filename specified. This field displays the value, if one exists, for the filename of the measure. See the section on the Data Interface Tool for details on data interface properties. This property is not editable by the user and is displayed for diagnostic purposes.

Load Intx (read only)

Measures included in the Data Interface Tool have a load intersection [Load Intx] specified. This field displays the value, if one exists, for the load intersection of the measure. See the section on the Data Interface Tool for details on data interface properties. This property is not editable by the user and is displayed for diagnostic purposes.

Start (read only)

Measures included in the Data Interface Tool will have a start position specified. This field displays the value, if one exists, for the start position of the measure. See the section on the Data Interface Tool for details on data interface properties. This property is not editable by the user and is displayed for diagnostic purposes.

Width (read only)

Measures included in the Data Interface Tool will have a width specified. This field displays the value, if one exists, for the width of the measure. See the section on the Data Interface Tool for details on data interface properties. This property is not editable by the user and is displayed for diagnostic purposes.

Load Agg (read only)

Measures included in the Data Interface Tool will have a load aggregation method [Load Agg] specified. This field displays the value, if one exists, for the load aggregation method of the measure. See the section on the Data Interface Tool for details on data interface properties. This property is not editable by the user and is displayed for diagnostic purposes.

Materialized (read only)

Certain measures may be registered as display only measures in order to improve performance within a workbook. This marking is done automatically at the time of installation. If the measure has been marked as display only, that fact will be reflected in this field. This property is not editable by the user and is displayed only for diagnostic purposes. This property is not editable by the user and is displayed for diagnostic purposes.

Creator (read only)

Certain extensions make use of plug-ins to the RPAS Configuration Tools to automatically generate configuration content. This field displays the creator of the given measure. The value “user” represents content generated by a user of the Configuration Tools. A different value represents content generated by a plug-in. This property is not editable by the user and is displayed only for diagnostic purposes.

Signature (read only)

The signature of a measure is used to resolve ambiguity that may result from overriding the name property of a measure. This field contains the value of the measures signature property. The contents of this field are created and maintained automatically by the Measure Tool. This property is not editable by the user and is displayed only for diagnostic purposes.

Agg Spec

This hybrid aggregation mechanism is designed to allow you to specify a complex method to aggregate the values of a measure. It allows you to specify a different aggregation method for each hierarchy in the measure’s intersection. When a measure with the hybrid agg type needs to be aggregated, this is accomplished by separately aggregating each hierarchy of the intersection according to the agg method for that hierarchy.

Example

Measure XYZ is defined at day_sku_str_ and has a hybrid aggregation type. The specifics for the aggregation are:

- Calendar should be aggregated by the first method.
- Location should be aggregated by the total method.
- Product should be aggregate by the total method.

Suppose that XYZ must be aggregated to the level of mnthclssrgn_. The process of generating this new value is accomplished by three successive aggregations:

Note: For now, ignore the ordering of the aggregations. The order to use is also configurable and part of the definition in the tools.

1. day_sku_str_ to day_clsstr_ by total (product)
2. day_clsstr_ to day_clssrgn_ by total (location)
3. day_clssrgn_ to mnthclssrgn_ by first (calendar)

In this example, the hybrid implementation of the pst (period start total) agg type is a hybrid aggregation method that has been hardwired into the RPAS calculation engine for some time. You are now allowed visibility to and control over the mechanism by which pst is performed.

A brief description of the UI functionality/constraints is as follows:

- The hybrid aggregation method now appears in the deff agg drop-down selector.
- When a measure is specified for hybrid agg, the agg spec [aggregation specification] field becomes editable.
- An agg spec can be typed in or built through a dialog (double click the agg spec editor or Ctrl-Space to launch it).

- This dialog looks very similar to the standard wizard popup for workbooks. On the right, the ordering of hierarchies in the intersection of the measure is set by dragging the hierarchies in the list. On the left, a separate aggregation type is selected for each hierarchy. For the most part, these are the aggregation types that are available for the measure based on its type. Exceptions are:
 - You cannot use recalc or hybrid within an agg spec.
 - You may use first and last for the Calendar hierarchy only.
 - An aggregation type must be specified for each hierarchy in the intersection.
 - If you want to type in a value, the syntax and meaning is the same as the arguments used by the aggregate function of the rule engine.
 - A hybrid aggregation measure must be read only in its agg state.
 - A hybrid measure must have a spread type of none.

Note: The hybrid aggregation type is not supported for extended measures (see “Configure Extended Measures”) or for the load aggregation method for a measure (see “

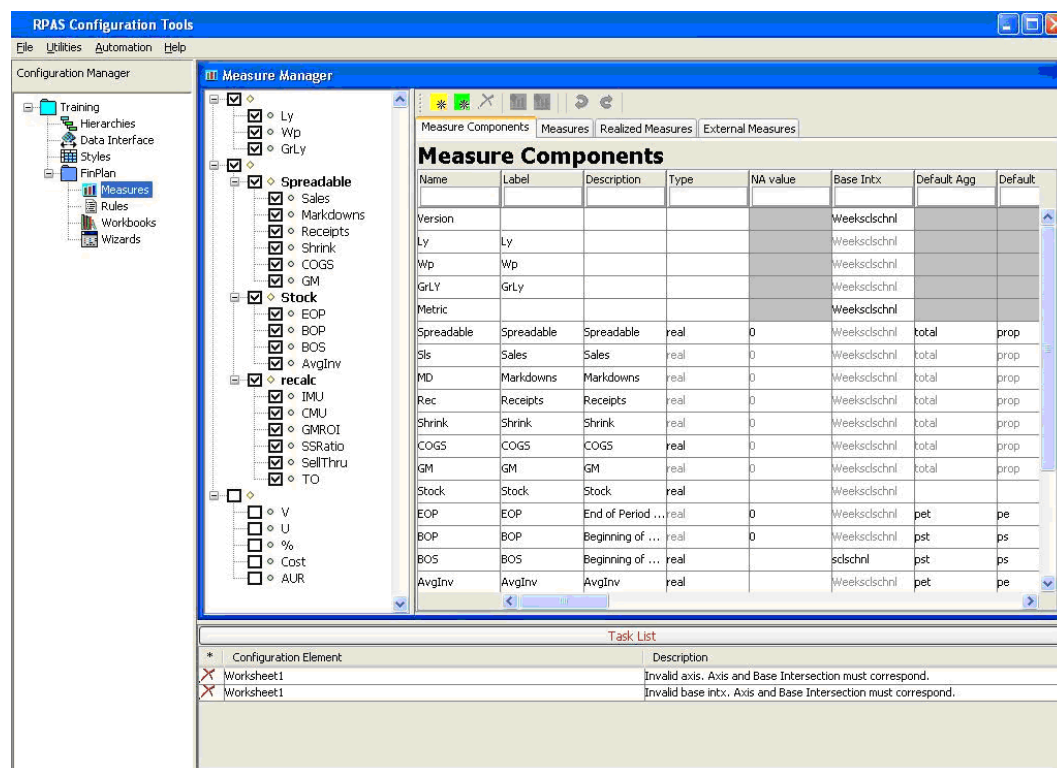
Edit Data Interface Information for a Measure”).

Unlike the Aggregate procedure, the “recalc” aggregation type is not supported for any hierarchy for a measure using the hybrid aggregation type.

Measures that use the hybrid aggregation type cannot be aggregated from a local domain into the global domain.

Edit the Components

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Measures**. The Measure Manager window opens in the workspace.



Push Components Down

1. In the Measure Definition window, select the component that you want to push down.
2. Right-click in the Measure Definition menu, and select **Push Down**.

Note: The component is pushed down one level in the component hierarchy, and a new component is created to take the place of the pushed down component.

Note: You cannot push down a major component.

Pull Components Up

1. In the Measure Definition window, select the component that you want to pull up.
2. Right-click in the Measure Definition window, and select **Pull Up**. The component is pulled up one level in the component hierarchy.

Note: You cannot pull a minor component up to become a major component, nor can you pull up a major component.

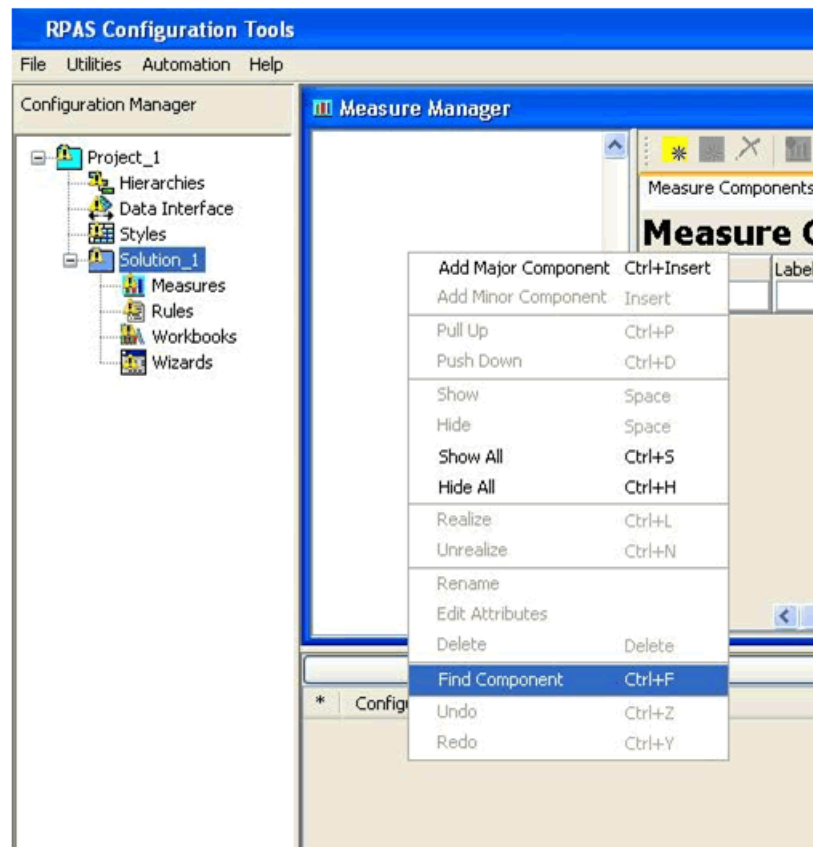
Display or Hide Components

- To display information about a component, select the check box next to the component name.
- To hide information about a component, clear the check box next to the component name.

Note: Selecting or clearing a check box for a major or minor component causes the check boxes for all minor components underneath it to be selected or cleared. This check box is also used to enable a component so it becomes active when measures are generated.

Find a Component

1. Right-click in the left-hand pane of the Measure Manager and select **Find Component**.



2. Type in the name of the desired component. The tree will scroll to bring the specified component into view.
3. Select the desired component.

Note: The full name (case sensitive) of the component is required in order to find it.

Rename a Component

Navigate: In the Measure Definition window, select the component that you want to rename.

Perform one of the following options:

- Right-click in the Measure Definition window, and select **Rename**. Type the new name for the component or double-click the component, and type the new name.
- Select and change the name of the component in the component pane.

Note: Changing the name of a component will result in a change in the name of any measure that inherits from the component unless the measure has overridden the name property.

Remove Components

Navigate: In the Measure Definition window, select the component that you want to remove.

Perform one of the following:

- Select **Remove Component**.
- Press the **Delete** key.
- Right-click in the Measure Definition window and select **Delete**.

The currently selected component will be removed from the Solution.

Note: It is not possible to remove a component that is used (or that has a descendent component that is used) in the specification of a measure.

Move Components

Navigate: In the Measure Definition window, select the component that you want to move.

Drag the component to its new location and release it.

Notes:

You cannot move a minor component to a different major component.

You cannot move a component so that it would be a descendent of another component that is used in the specification of a realized measure.

Working with Measures

Overview

The Measure Tool allows you to create and name measures by selecting major and minor components that are already defined. By default, the measures inherit the properties that are defined for the components. To create a measure, you select the components that you want to use to construct measures. The measure tool will generate prototype measures for all of the combinations that you select. This saves you from manually creating all of the measures.

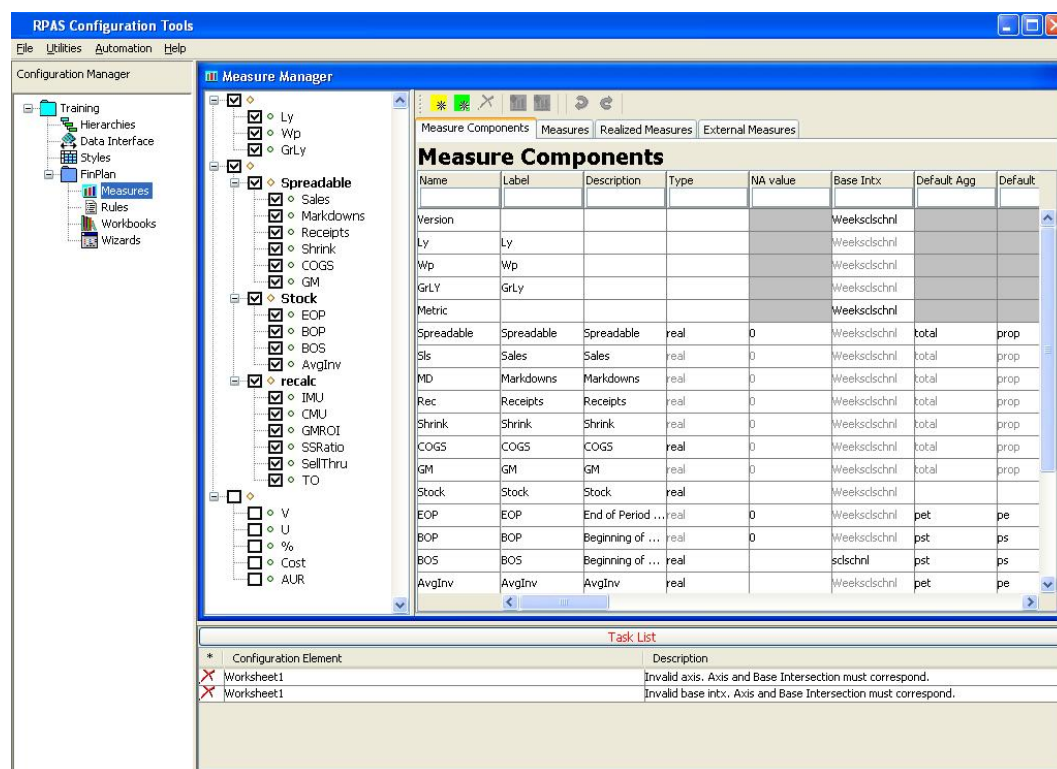
You can then override the properties for individual measures by entering them the same way as on the components tab. Once you override properties at the measure level, changes made at the component level will no longer spread down to that measure as it will retain the overridden value. You may also restore an overridden value to its inherited value or override an inherited value to be unspecified.

When a measure is auto-generated by the measure tool, it is considered a prototype measure and cannot be edited or used in any other configuration component. To enable the measure to be edited and/or used elsewhere in the configuration, it must be realized first. If the measure is not going to be used, it does not need to be realized. Only realized measures are recognized and used in configuration components, such as expressions and workbooks.

The Measures tab in the measure pane shows all “prototype measures” for the selected components. The ‘Realized Measures’ tab shows just those that are realized.

Realize and Unrealize Measures

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Measures**. The Measure Manager window opens in the workspace.



Measure Manager – Measure Components Tab

1. Select the **Measures** tab.
2. In the components tree, select the check boxes for the components that you want to use to filter the measures. The measure tool will show measures using all of the combination of components that you selected. This process filters the list of prototype measures that are shown from all combinations of components to the combinations of components that have been selected. It uses those components to determine which prototype measures to show.

Realize a Measure

1. Select the components that are used in the measures that you want to realize.
2. Select the check box in the Realized column for the prototype measure that you want to realize.

Unrealize a Measure

1. Select the components that are used in the measures that you want to unrealize.
2. Unselect the check box in the Realized column for the prototype measure that you want to unrealize.

Rename a Measure

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Measures**. The Measure Manager window opens in the workspace.

1. Select the components that are used in the measures that you want to rename.
2. In the Measures tab, or the Realized Measures tab, click the name of the measure that you want to rename.
3. Type the new name for the measure.

Note: The measure must be realized before you can rename it.

Show Measures by Component

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Measures**. The Measure Manager window opens in the workspace.

1. In the Measure Manager window, select the **Measures** tab.
2. Perform one of the following:
 - Select the check boxes next to the components whose measures you want to show.
 - Select the components used in the measures that you want to show.
3. Right-click the Measures Definition menu, and select **Show**, or press the space bar.

Show All Measures

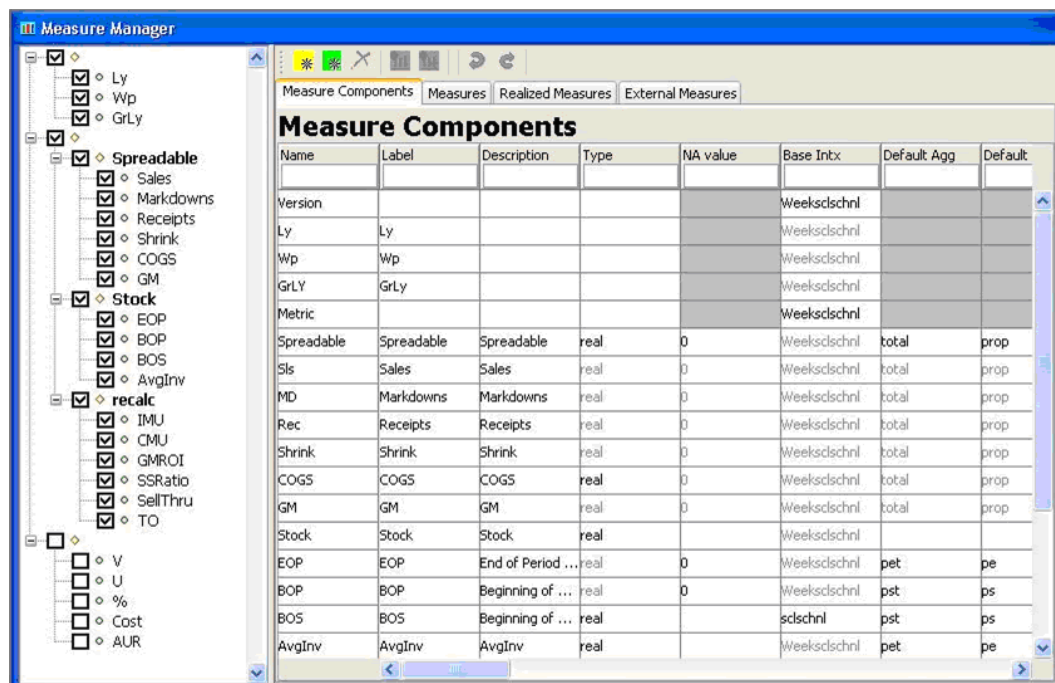
Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Measures**. The Measure Manager window opens in the workspace.

Right-click in the Measure Manager window, and select **Show All**.

Note: Due to memory constraints when working with very large numbers of components, all prototype measures may not be displayed. In this case, you will receive an error message instructing you to deselect some measure components.

Hide Measures by Component

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Measures**. The Measure Manager window opens in the workspace.



Measure Manager – Measure Components Tab

1. In the Measure Manager window, select the **Measures** tab.
2. Perform one of the following:
 - Uncheck the check boxes next to the components whose measures you want to hide.
 - Select the component used in the measures that you want to hide.
3. Right-click in the Measure Manager menu, and select **Hide**, or press the spacebar.

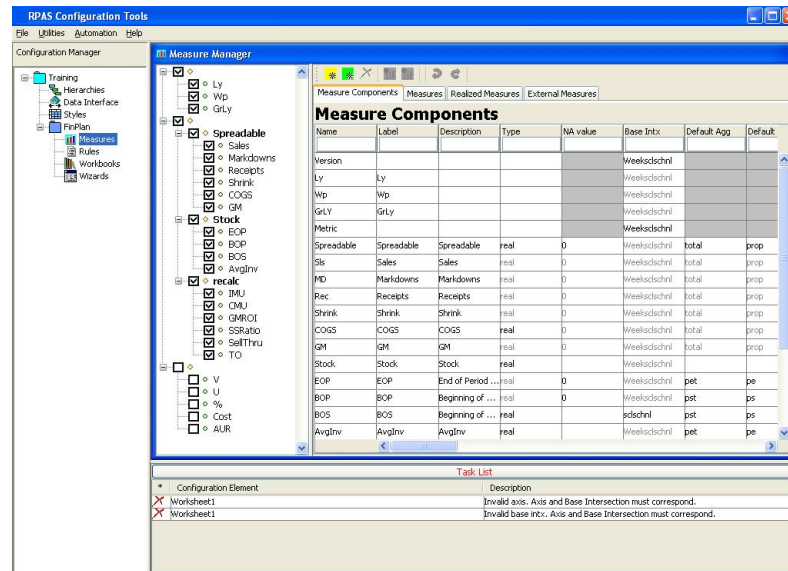
Hide All Measures

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Measures**. The Measure Manager window opens in the workspace.

1. In the Measure Manager window, select the **Measures** tab.
2. Right-click the Measure Manager window, and select **Hide All**.

Sort Measures by Property Value

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Measures**. The Measure Manager window opens in the workspace.



In the Realized Measures tab, measures can be sorted by property value.

1. Select the Realized Measures tab.
2. Hold down the control key (CTRL) and click in the filter field at the top of the table for the property of the measures to be sorted.

The measures are sorted in alphabetical order according to the value of the property.

Filter Measures by Property Value

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Measures**. The Measure Manager window opens in the workspace.

In the Realized Measure tabs, you can filter measures by property value.

1. In the Realized Measures tab, click in the filter field at the top of the table for the property that you want the measures filtered by.
2. Enter the value on which measures are to be filtered. The measures are filtered according to the value of the property entered.

Note: You only get the measures whose value for the selected property is the same as the value entered in the filter box. This field is case sensitive.

Imported Measures

Overview

Under most circumstances, measures exist only within the Solution in which they are defined. When working with a Project with multiple Solutions, it is sometimes desirable to make use of a measure defined in a different Solution. The Measure Tool allows you to “import” a measure defined in a different Solution (but not a different Project) into the current Solution.

These measures (called imported measures or external measures) then become visible in the Imported Measures Tab of the Measure Tool. Within this tab, it is possible to modify certain measure properties so that the use of the measure in the Solution into which it has been imported will differ from its use in the Solution in which it was originally defined. For example, you may want to modify a writeable measure so that it is read only in the Solution into which it is imported.

Within the Measure Selectors present in the Rule and Workbook Tools and the Expression Builder, there is a checkbox marked “Include External Measures.” When this checkbox is selected, the Measure Selector will include those measures that have been imported into the Solution in addition to those that are present due to component selections in the Measure Selector.

Properties that may be overridden in an imported measure:

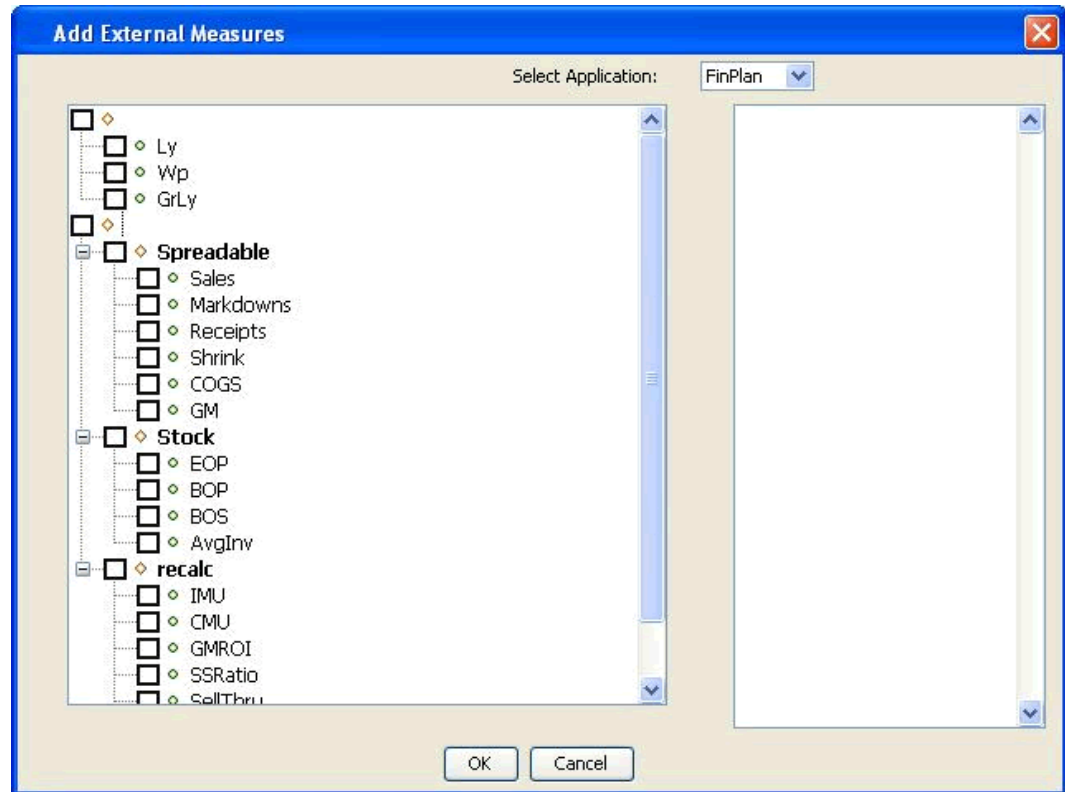
- Label
- Description
- Base State
- Agg State
- UI Type
- Range

Import a Measure

Note: You may only import a measure into a solution in a project that has at least one other solution.

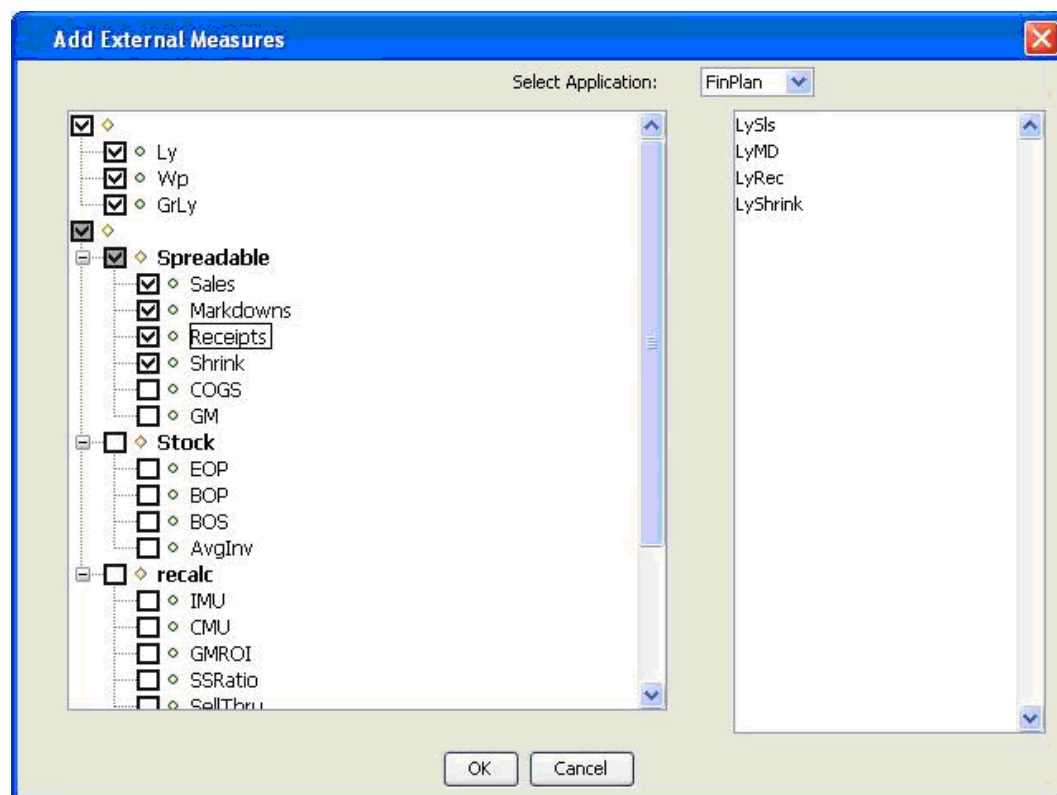
Navigate: In the Configuration Manager, select **Project – Solution – Measures**. The Measure Manager window opens in the workspace.

1. Select the **External Measures** tab and click the **Import Measure** button on the toolbar. The Add External Measures dialog is displayed.



Add External Measures Dialog Box

2. In the Add External Measures dialog, select the Solution in which the desired measure is defined.
3. Use the measure selector (left-hand side) to select the measure(s) to import.



Selected Measures to Be Imported and in Box to Right

4. External measures that are already imported appear in bold. Select **OK** to exit the dialog. The selected measure will appear in the External Measure tab.

Remove an Imported Measure from a Solution

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Measures**. The Measure Manager window opens in the workspace.

1. In the Imported Measures tab, select the external measure to remove.
2. Click the **Remove Imported Measure** button in the toolbar, or right-click and select **Remove Import**. The selected imported measure will be removed from the External Measure table.

Rule Sets

Overview

A rule set is a collection of rule groups. It is used as a placeholder for containing rule groups, which makes the visual display of rules easier. A workbook uses the rule groups specified in one rule set.

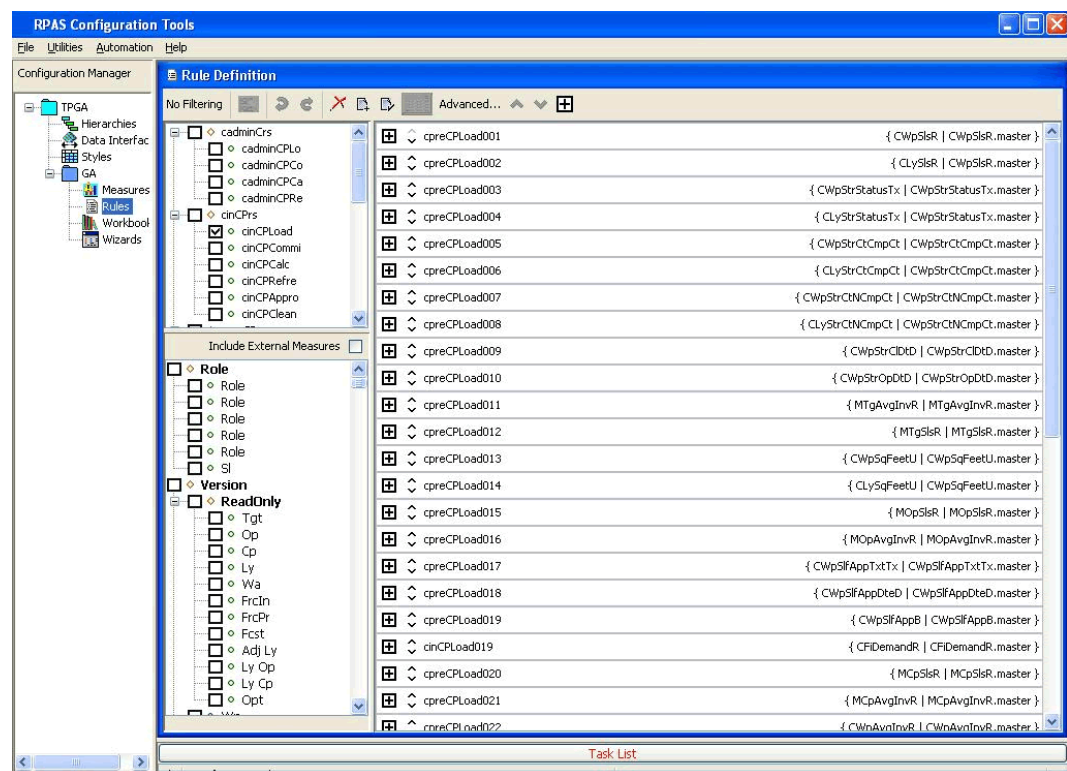
Note: A rule set is a tools concept only. It does not appear in the configured solution.

A rule set is created along with the following default rule groups: load, commit, calc, and refresh.

The rule group names are prefixed with the name of the rule set followed by an underscore. After the default rule groups are created, you can create additional rule groups as necessary. You can also rename the rule groups that were automatically generated, but you cannot delete them.

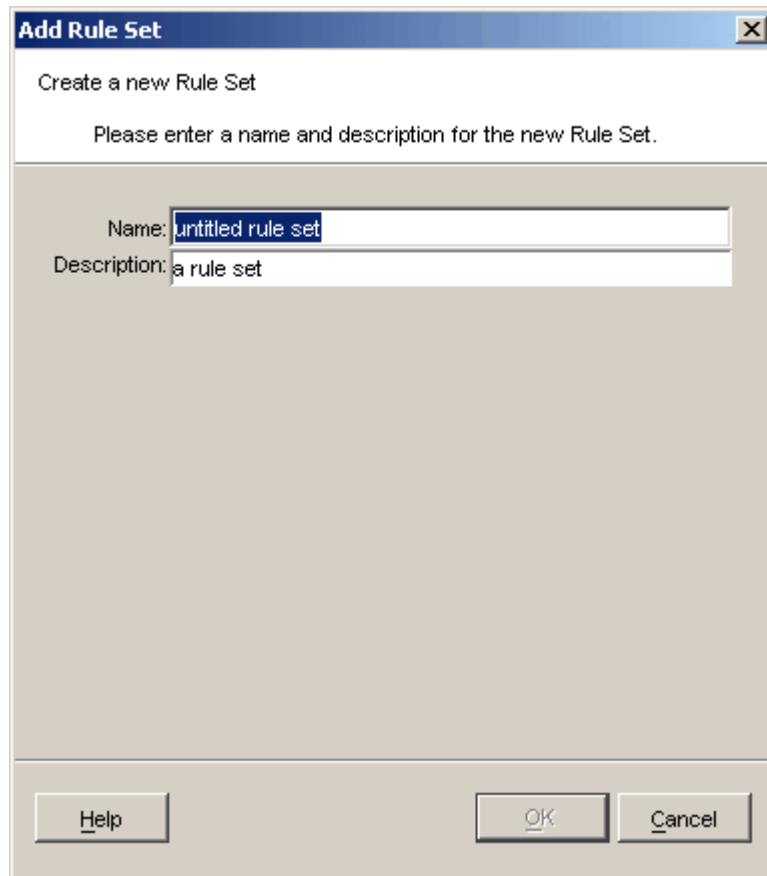
Create a Rule Set

Navigate: In the Configuration Manager, select  Project –  Solution –  Rules. The Rule Definitions window opens in the workspace.



Rule Definition Window

1. From the toolbar, click the **New**  button, and select **Rule Set**, or select **Create/Rule Set** from the right-click menu. The Add Rule Set window appears.



Add Rule Set Dialog Box

2. In the Name field, enter the name of the rule set.

Note: A rule set name can be a maximum of ten alphanumeric or underscore characters. It must not have a name that is the same as any other rule set that exists in the project.

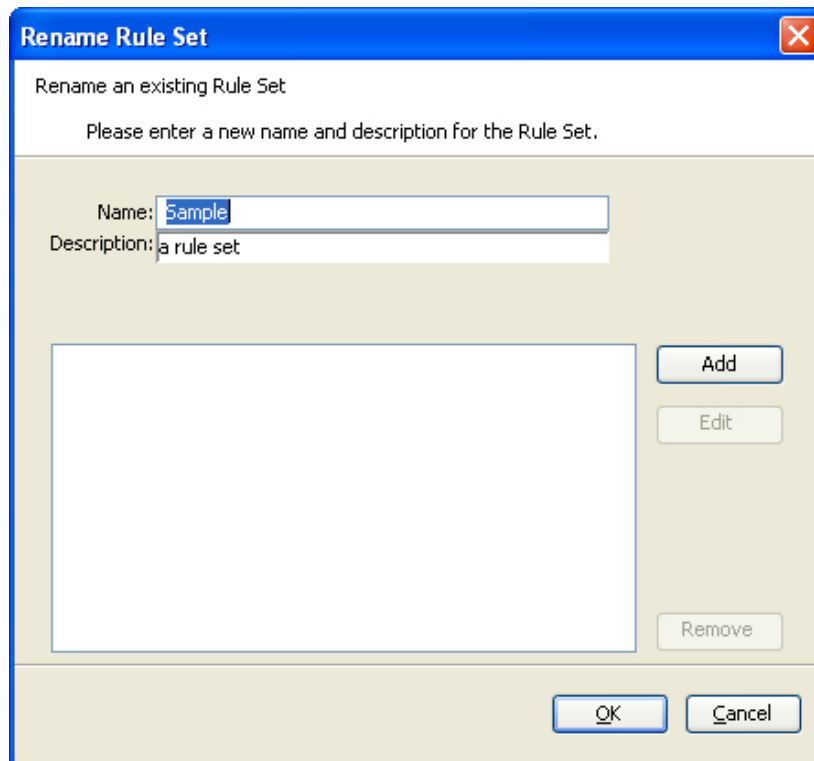
3. In the Description field, enter a description of the rule set.
4. Click **OK** to save any changes and close the window.

Add Attributes to a Rule Set

Attributes are pieces of information about the rule set. This feature allows you to describe or document the information about the configuration.

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Rules**. The Rule Definitions window opens in the workspace.

1. From the toolbar, click the **Edit**  button, and select **Rule Set**, or select **Rename/Rule Set** from the right-click menu. The Rename Rule Set window opens.



Rename Rule Set

Rename an existing Rule Set

Please enter a new name and description for the Rule Set.

Name:

Description:

1. Click **Add**. The Edit Attribute window opens.



Edit Attribute

Attribute: Value:

2. In the Attribute field, enter the appropriate information.
3. In the Value field, enter the appropriate information.
4. Click **OK** to save any changes and close the window.

Delete a Rule Set

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Rules**. The Rule Definitions window opens in the workspace.

1. In the Rule Definition window, select the rule set that you want to delete.
2. Perform one of the following:

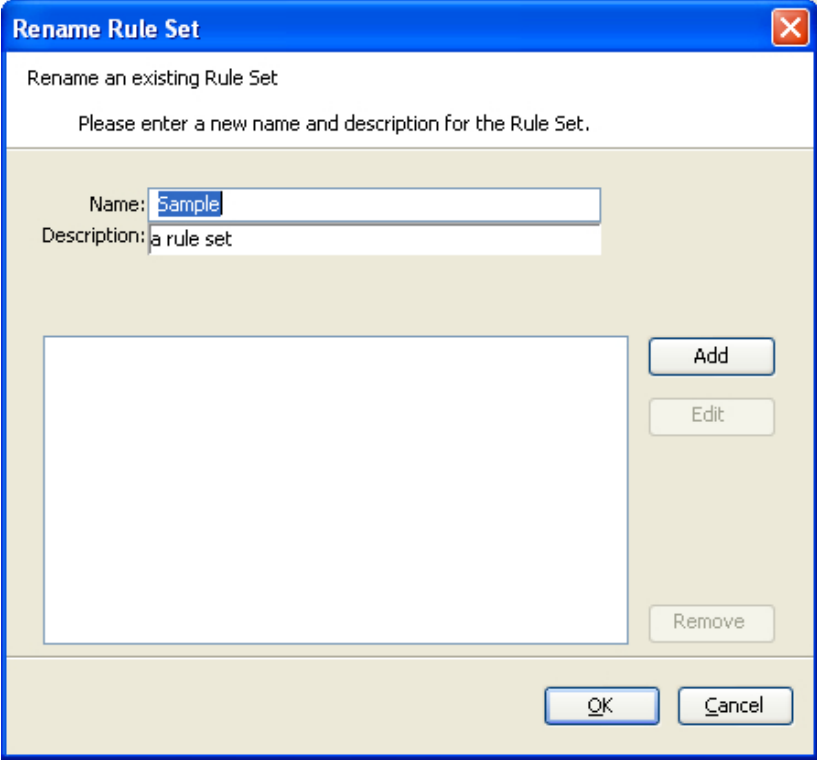
- From the toolbar, click the **Delete**  button, and select **Rule Set**.
- Right-click in the Rule Definition window, select **Delete**, and select **Rule Set** from the list.

Note: When a rule set is deleted, any rules that were just used in rule groups in that rule set will still be in the rule pool, but will be unused. They will be permanently lost when the project is closed unless they are used in another rule group.

Edit Properties of a Rule Set

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Rules**. The Rule Definitions window opens in the workspace.

1. In the Rule Definition window, select the rule group that you want to edit.
2. From the toolbar, click the **Edit**  button, and select **Rule Group**, or select **Rename/Rule Set** from the right-click menu. The Rename Rule Set window opens.



The dialog box titled "Rename Rule Set" has a blue header bar with a close button (X). The main area is light beige. It contains the text "Rename an existing Rule Set" and "Please enter a new name and description for the Rule Set." Below this are two input fields: "Name:" with the text "Sample" and "Description:" with the text "a rule set". To the right of these fields are three buttons: "Add", "Edit", and "Remove". At the bottom of the dialog are "OK" and "Cancel" buttons.

3. In the Name field, enter the name of the rule set.

Note: A rule set name can be a maximum of ten characters.

4. In the Description field, enter a description of the rule set.
5. To edit attributes for a rule set:
 - a. Select the attribute that you want to edit.
 - b. Click **Edit**. The Edit Attribute window opens.
 - c. Update the information as necessary.
 - d. Click **OK** to save any changes and close the window.
6. To remove attributes from a rule set:
 - a. Select the attribute that you want to delete.
 - b. Click **Remove**. The attribute is removed from the display box.
7. Click **OK** to save any changes and close the window.

Rule Groups

Overview

In RPAS, a rule group is an ordered collection of rules that are treated as a unit by the calculation engine with the integrity of all the rules in the rule group being maintained together.

Rules within a rule group are given a priority. The calculation engine uses this to select a calculation path that follows business priorities. It does this by using rule priorities to determine which rule to enforce when there is a choice to be made. Although there may only be one active rule group at any time, you can define multiple rule groups to satisfy different calculation requirements.

Types of Rule Groups

Rule groups may be one of four different types:

- **Load** – The RPAS application automatically uses the load rule group when loading data into a workbook from the domain.
- **Calculate** – The RPAS application uses a calculate rule group to apply the effects of user changes to cells. RPAS supports multiple calculation rule groups. Menu options may be configured to allow the user to transition through different calculation rule groups in order to support special processes, such as authorizations. RPAS ensures a smooth transition from one rule group to another.
- **Refresh** – The RPAS application uses a refresh rule group to refresh the data from the domain (for example, to update “actuals”). Multiple refresh rule groups can be specified and selected by the user.
- **Commit** – The RPAS application automatically uses the commit rule group when committing data from the workbook to the domain.

A measure that does not have data in the domain may be loaded into a workbook by using a rule in the load rule group to calculate it based on other measures that are loaded. Similarly, a measure that exists in a domain, but not a workbook, may be committed by using a rule in the commit rule group that calculates it from other measures that are in the workbook.

Rule Group Validation

Within a solution, there may be many rules defined, and each rule is validated individually. Rules within a rule group are also validated in the context of all the other rules in that rule group. While a rule may be perfectly valid syntactically, it may not be valid within the context of a particular rule group. Rule group validations include:

Each rule in a rule group must represent a completely different measure relationship. This means:

- No two rules in a rule group may use exactly the same collection of measures. If such a condition were allowed, the calculation engine would be unable to calculate either of the rules, because they would be dependent upon each other, so neither could be calculated first. This is explicitly validated for in the rule tool.
- For similar reasons, a rule should not normally use a collection of measures that is a sub-set of the collection of measures in another rule. If the measures that are only in the larger rule were all changed, the situation would be equivalent to the above. There are circumstances; however, where this technique is valuable. For example, in a load rule group where there may be a rule to load a measure from the domain, but other rules that include that measure. In this case, this condition is not explicitly validated for.
- There must be one (and only one) expression that calculates a recalc measure used in a rule group.
- Other than in the special circumstance of a rule constructed from multi-result functions or procedures, a measure may only be on the left-hand side (LHS) of one expression in a rule.

Multiple Refresh Rule Groups

The Refresh Rule Group 'refreshes' (supplies new values for) data, which can generally be thought of as being 'external' to the workbook. An example would be "actuals":

- If you built a workbook in week 5, you would have actuals for weeks 1-4.
- In week 6, you may want to refresh the actuals to get the actuals for week 5.

The Configuration Tools and RPAS support the use of multiple refresh rule groups. Within the rule tool, there is the ability to create multiple refresh rule groups within a rule set. These multiple refresh rule groups can then be assigned to a workbook template using the Workbook Designer, and they will be available for you to select from in the RPAS client.

- A workbook contains all the rule groups in a single rule set, so if multiple refresh rule groups are required in a workbook, they must all be in the same rule set.
- You may consider naming your rule groups so the usage of the refresh rule groups are reflected in their names; such as refresh_all, refresh_actuals, and refresh_manager.

Rule Group Transitions

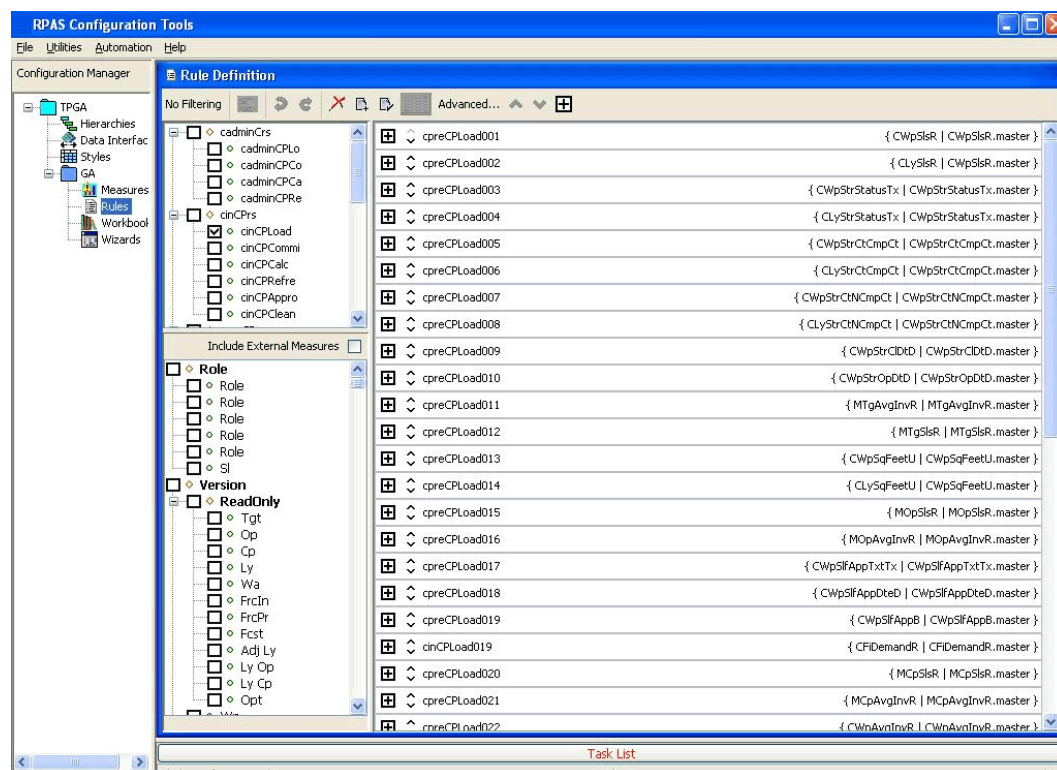
Although only a single rule group may be active at any time, RPAS supports the transition from one rule group to another, so the active rule group may be changed. The calculation engine ensures the integrity of measure relationships at all times, so this transition process is not merely a case of switching from one rule group to another, because there are no guarantees that the integrity of the rules in the rule group being transitioned into would have been maintained. There are different forms of rule group transitions. When designing rule groups, you should consider the impact of anticipated rule group transitions.

Automatic rule group transitions occur:

- **On workbook building** – Data is loaded using the load rule group. This will typically load measures by calculating them from the data values held on the domain using the master modifier, but may also calculate other measures that are not explicitly loaded. When the load is complete, the system will automatically execute a full transition to the calculate rule group.
- **On data refreshing** – Data refreshing causes some measures to be updated from values held in the domain. The measures that are affected by the refreshed measures are treated as affected in the calculate rule group, and a normal calculation of that rule group follows. Effectively, data refreshing causes a calculation using the calculate rule group as if the cells that were refreshed were directly changed by the user.
- **On data committing** – There is a full transition from the current calculate rule group to the commit rule group. This typically commits measures by calculating them on the domain by using the master modifier. There is then a null transition back to the calculation rule group (that is, no transition process is executed since the assumption is that nothing in the workbook has changed), so no transition is required.
- **On executing custom menus** – There is a full transition between each rule group in the custom menu, which is followed by a full transition back to the default calculate rule group.

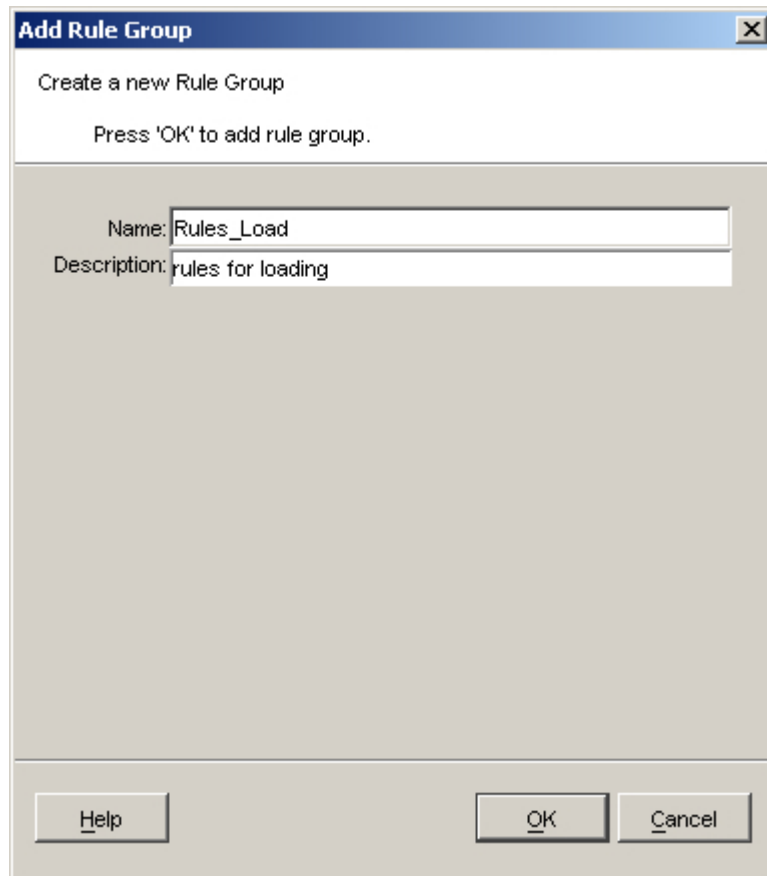
Create a Rule Group

Navigate: In the Configuration Manager, select  **Configuration** –  **Project** –  **Solution** –  **Rules**. The Rule Definitions window opens in the workspace.



Rule Definition Window

1. In the Rule Definition window, select the rule set where you want to create a new rule group.
2. From the toolbar, click the  **New** button and select **Rule Group**, or select **Create/Rule Group** from the right-click menu. The Add Rule Group window is displayed.

A screenshot of a Windows-style dialog box titled "Add Rule Group". The dialog has a blue title bar with a close button (X) in the top right corner. Inside the dialog, the text "Create a new Rule Group" is displayed, followed by the instruction "Press 'OK' to add rule group." Below this, there are two text input fields. The first field is labeled "Name:" and contains the text "Rules_Load". The second field is labeled "Description:" and contains the text "rules for loading". At the bottom of the dialog, there are three buttons: "Help", "OK", and "Cancel". The "OK" button is highlighted with a darker border.

Add Rule Group

Create a new Rule Group

Press 'OK' to add rule group.

Name: Rules_Load

Description: rules for loading

Help OK Cancel

Add Rule Group Window

3. In the Name field, enter the name of the rule group.

Note: A rule group name can be a maximum of 16 alphanumeric or underscore characters. It must not have a name that is the same as any other rule group that exists in the project.

4. In the Description field, enter a description of the rule group.
5. Click **OK** to save any changes and close the window.

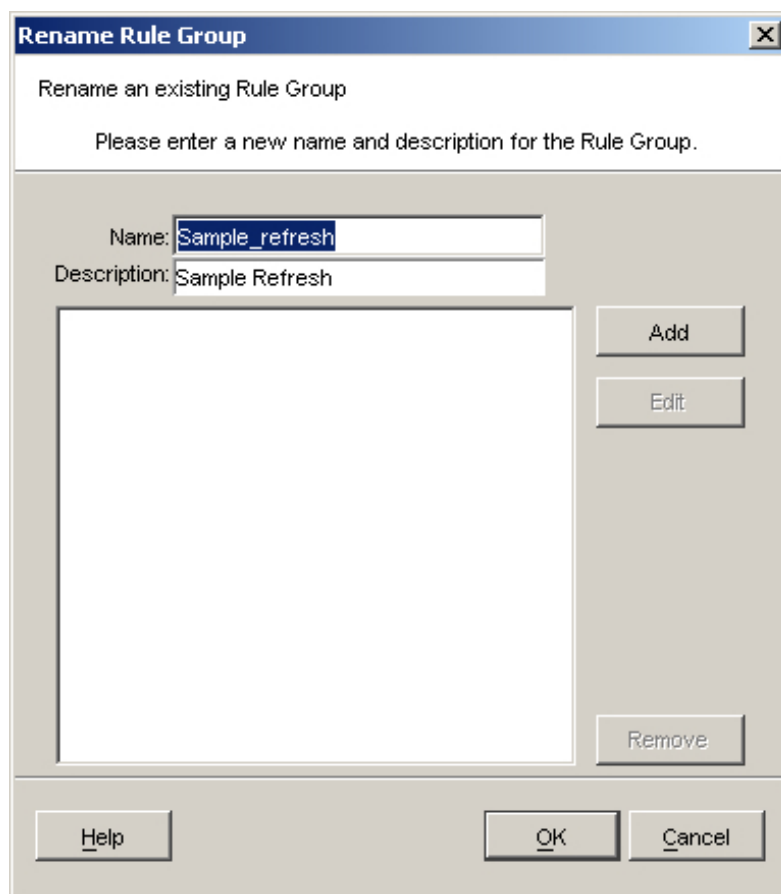
Add Attributes to a Rule Group

Attributes are pieces of information about the rule group. They are a tools feature only (not used in the configured solution) and are provided as a convenience for such purposes as documentation of the configuration.

Navigate: In the Configuration Manager, select  **Configuration** –  **Project** –  **Solution** –  **Rules**. The Rule Definitions window opens in the workspace.

1. In the Rule Definition window, select the rule group whose attributes you want to edit.

1. From the toolbar, click the **Edit**  button and select **Rule Group**, or select **Rename/Rule Group** from the right-click menu. The Rename Rule Group window opens.



The **Rename Rule Group** dialog box is shown. It has a title bar with a close button. The main area contains the text "Rename an existing Rule Group" and "Please enter a new name and description for the Rule Group." Below this are two text boxes: "Name:" with the value "Sample_refresh" and "Description:" with the value "Sample Refresh". To the right of these boxes are three buttons: "Add", "Edit", and "Remove". At the bottom of the dialog are three buttons: "Help", "OK", and "Cancel".

2. Click **Add**. The Edit Attribute window opens.

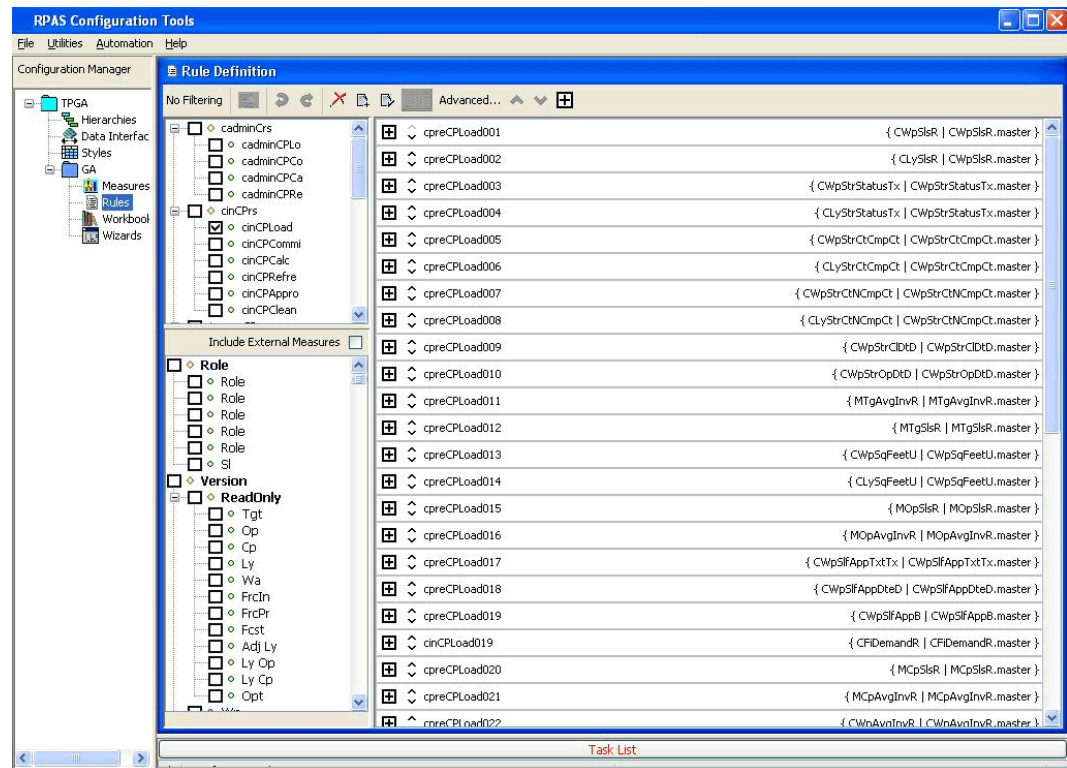


The **Edit Attribute** dialog box is shown. It has a title bar with a close button. The main area contains two text boxes: "Attribute:" with the value "expansionpolicy" and "Value:" with the value "standard". Below these boxes are two buttons: "Cancel" and "OK".

3. In the Attribute field, enter the appropriate information.
4. In the Value field, enter the appropriate information.
5. Click **OK** to save any changes and close the window.

Delete a Rule Group

Navigate: In the Configuration Manager, select  **Configuration** –  **Project** –  **Solution** –  **Rules**. The Rule Definitions window opens in the workspace.



1. In the Rule Definition window, select the rule group that you want to delete.

Note: Only user-created rule groups within a rule set can be deleted. You cannot delete the default load, commit, calc, and refresh rule groups. If you select **Delete** for one of the default rule groups, it will not be deleted, but all of the rules will be removed from the rule group.

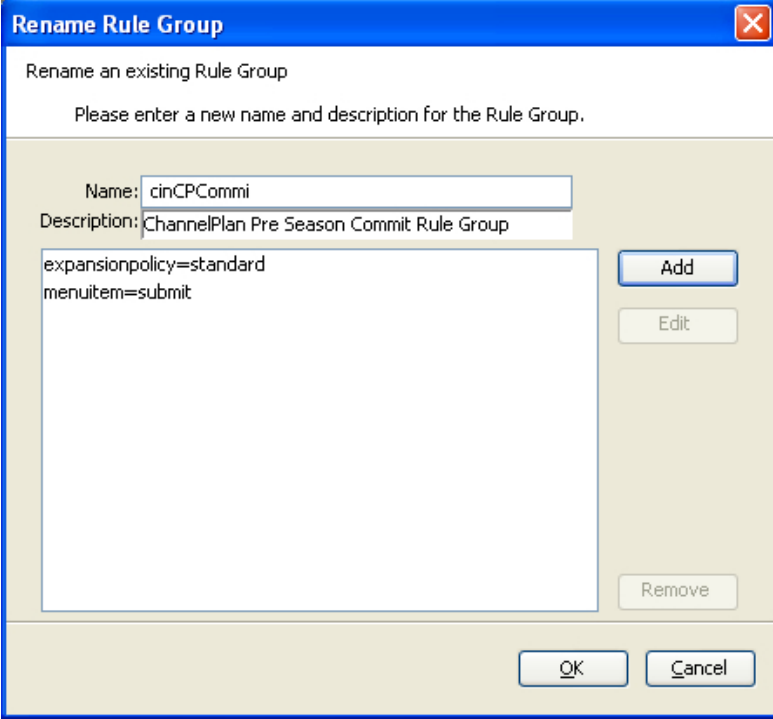
2. From the toolbar, click the **Delete**  button, and select **Rule Group**.

If the rule group was a configured (not a default) rule group, it is deleted. If it was a default rule group, all rules are removed from it. In either case, the rules will still exist within the rule pool, but they will be lost when the project is closed if they are not used in another rule group.

Edit Properties of a Rule Group

Navigate: In the Configuration Manager, select  **Configuration** –  **Project** –  **Solution** –  **Rules**. The Rule Definitions window opens in the workspace.

1. In the Rule Definition window, select the rule group that you want to edit.
2. From the toolbar, click the **Edit**  button and select **Rule Group**, or select **Rename/Rule Group** from the right-click menu. The Rename Rule Group window opens.



Rename Rule Group

Rename an existing Rule Group

Please enter a new name and description for the Rule Group.

Name:

Description:

expansionpolicy=standard
menuitem=submit

3. In the Name field, enter the name of the rule group.

Note: A rule group name can be a maximum of 16 characters.

4. In the Description field, enter a description of the rule group.
5. To edit attributes for a rule group:
 - a. Select the attribute that you want to edit.
 - b. Click **Edit**. The Edit Attribute window opens.



Edit Attribute

Attribute: Value:

- c. Update the information as necessary.
- d. Click **OK** to save any changes and close the window.

6. To remove attributes from a rule group:
 - a. Select the attribute that you want to delete.
 - b. Click **Remove**. The attribute is removed from the display box.
7. Click **OK** to save any changes and close the window.

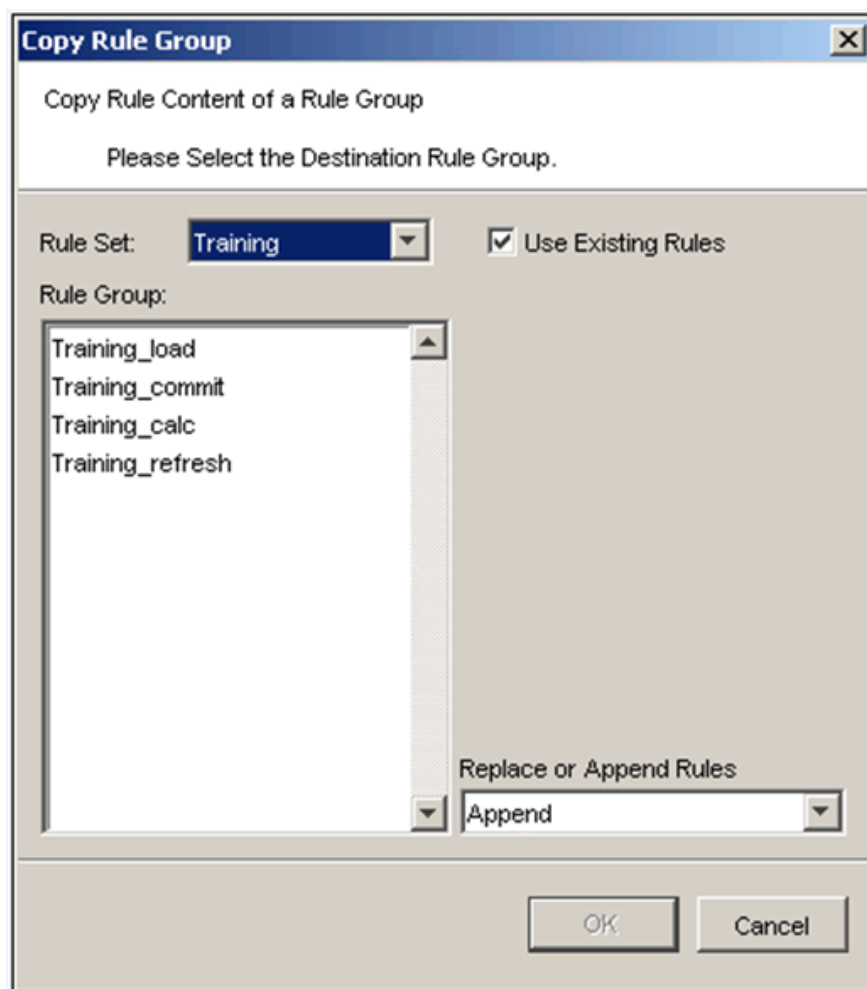
Copy a Rule Group

If two rule groups are similar, it may be beneficial to copy one rule group into the other to prevent having to create a rule group from scratch. When copying the rules of a rule group into another rule group, it is possible to specify whether existing rules will be used or copies of the rules will be created. The Use Existing Rules checkbox defaults to using any existing rules in the rule pool. If this checkbox is checked, the copy rule group operation will use the same rules that the source rule group has. If this checkbox is unchecked, the copy rule group operation will create copies of the rules and use those copies for appending to or replacing rules in the destination rule group.

Note: The rule group that you want to copy into must already exist. You cannot create a new rule group through this process.

Navigate: In the Configuration Manager, select  **Configuration** –  **Project** –  **Solution** –  **Rules**. The Rule Definitions window opens in the workspace.

1. In the Rule Definition window, select the rule group that you want to copy.
2. From the toolbar, click **Advanced**, and select **Copy Rule Group**. The Copy Rule Group window is displayed.



3. In the Rule Set field, select the destination rule set.
4. In the Rule Group area, select the desired destination Rule Group.
5. In the Replace or Append Rules field, select:
 - **Replace** – to overwrite all rules that already exist in the destination rule group.
 - **Append** – to add to the rules already in the destination rule group.
6. Click OK.

Measure Validation in the Rule Tool

If measure content validation has been turned off in the Measure Tool, measure validation must be manually initiated in the rule tool. See “Measure Validation within the Measure Tool.”

Click the perform **Measure Content Validation** button on the toolbar in the Rule Tool.

Rules

Overview

The rule definition tool allows you to define, organize, and manage rule sets, rule groups, and rules. It also allows you to create expressions and add expressions to rules.

Rules are groups of expressions that describe the relationship between measures.

When a rule has multiple expressions, those expressions are given a priority sequence to help the calculation engine select a calculation path that follows business priorities.

When given a choice, the calculation engine will always select the highest priority expression in the rule that is available to be selected. Considerable care should be taken in the design of rules to ensure that appropriate expression priorities are established. The business priority may vary from implementation to implementation, and it may vary from one type of plan to another in the same implementation.

Rules are also given a priority sequence within a rule group to help the calculation engine select a calculation path that follows business priorities. When given a choice, the calculation engine will always select the highest priority rule that needs to be calculated.

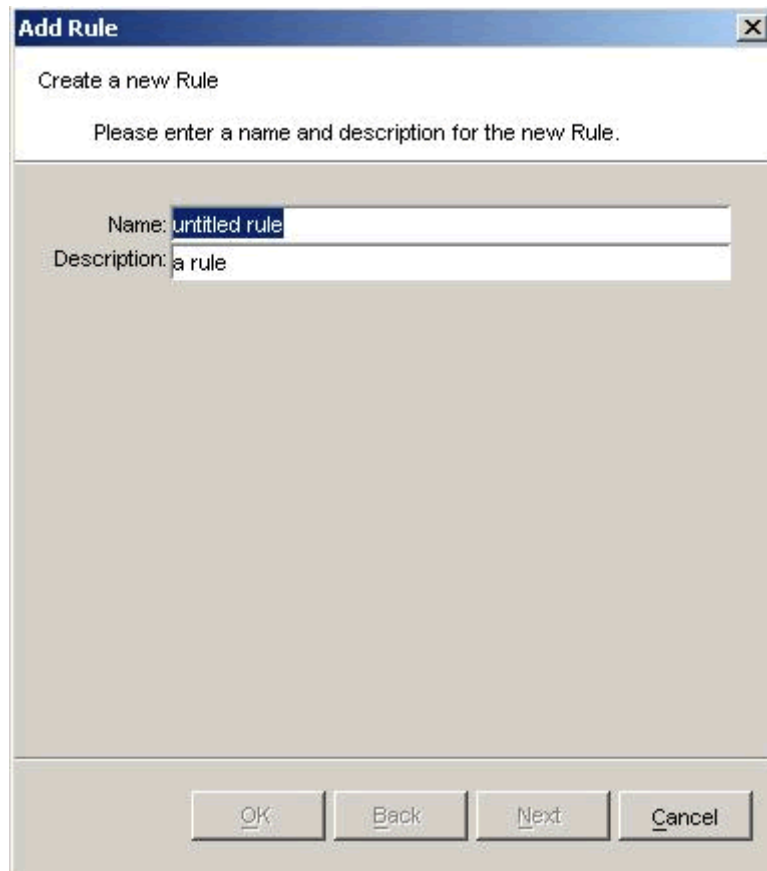
Those who are configuring the calculation requirements of a solution are expected to fully understand the operation of the RPAS calculation engine.

Create a Rule and Add It to a Rule Group

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Rules**. The Rule Definitions window opens in the workspace.

1. In the Rule Definition window, select the rule group where you want to create a rule. If you want the rule to be created in a particular position in the sequence of rules in the rule group, select the rule that you want the new rule to be placed before.
2. Perform one of the following:
 - From the toolbar, click the **New**  button and select **Rule**.
 - Select **Create/Rule** from the right-click menu.
 - Press CTRL-R.

The Add Rule window opens.

A screenshot of a Windows-style dialog box titled "Add Rule". The dialog has a blue title bar with a close button (X) in the top right corner. The main area is light gray. At the top, it says "Create a new Rule" and "Please enter a name and description for the new Rule." Below this, there are two text input fields. The first is labeled "Name:" and contains the text "untitled rule". The second is labeled "Description:" and contains the text "a rule". At the bottom of the dialog, there are four buttons: "OK", "Back", "Next", and "Cancel".**Add Rule Window**

3. In the Name field, enter the name of the rule.

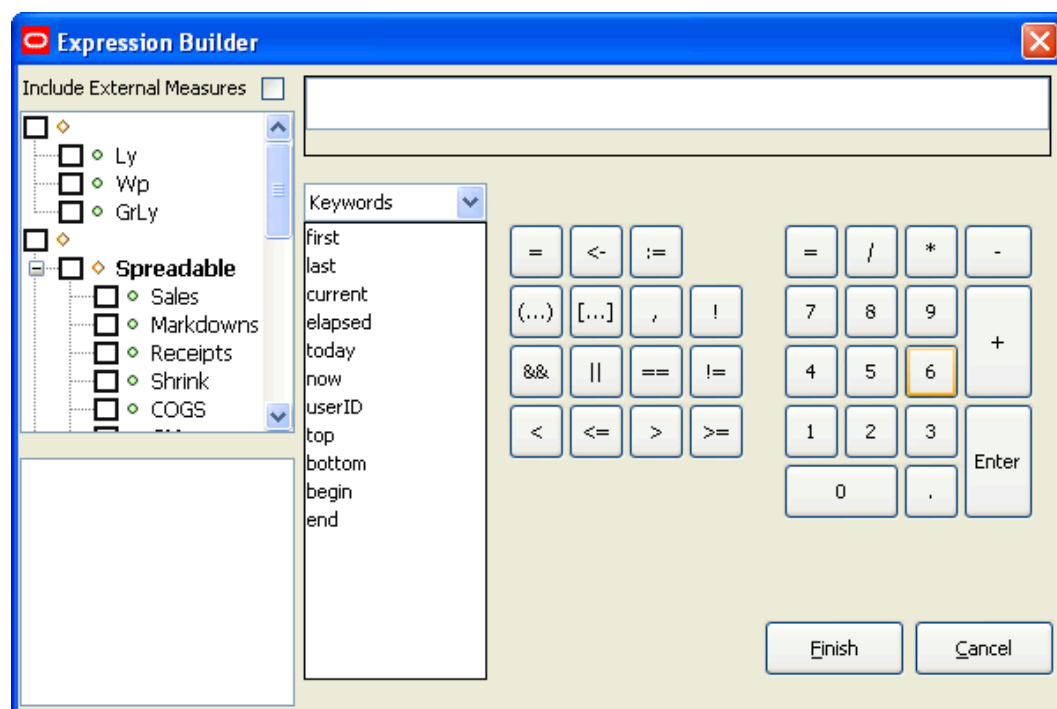
Note: A rule name can be a maximum of 24 alphanumeric or underscore characters. It must not have a name that is the same as any other rule that exists in the project. Rule names may start with a letter or an underscore, but may not start with the letter 'r' or 'R' followed by a number.

4. In the Description field, enter a description of the rule.
5. Click **Next**. The Expression Builder window of the Add Rule window is displayed.

**Specify Expression**

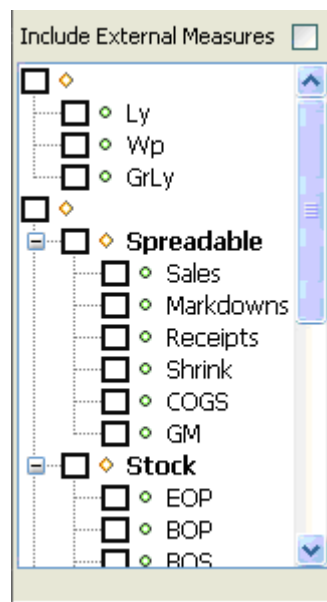
6. You can then type the expression in the input box
OR

- a. Click the expression builder button  to the right of the text box. The Expression Builder window opens.

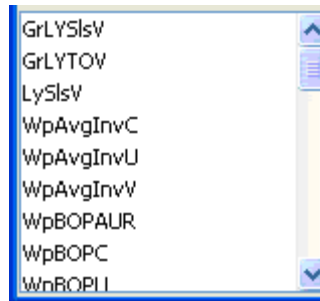


Expression Builder Window

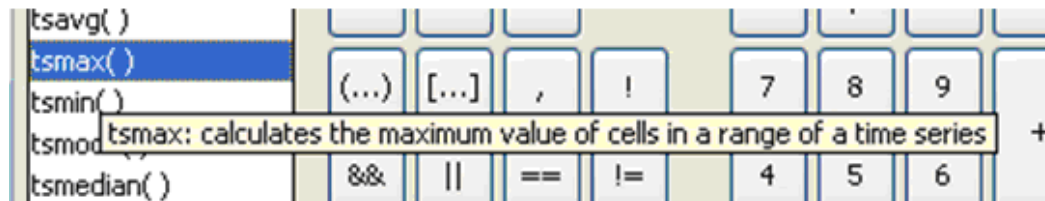
- b. In the upper left-hand pane, select the measure components to filter measures to be visible for pasting into the expression. If External Measures are required in the expression, check the **Include External Measures** checkbox. Realized measures meeting the filtering conditions are displayed in the lower left hand pane.



- c. In the lower left hand pane, double click any measures that you want to use in the expression to get the measure name pasted at the insertion point in the expression.

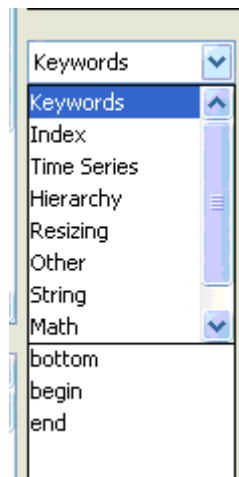


Note: Hold the mouse over the name of a function, procedure, keyword, or modifier to see a 'tooltip' that explains its function.

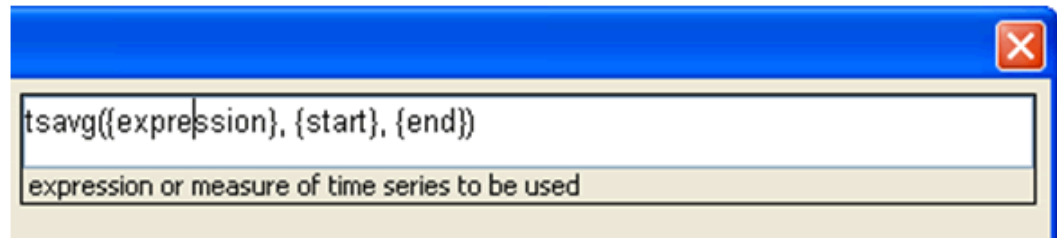


Tooltip Example

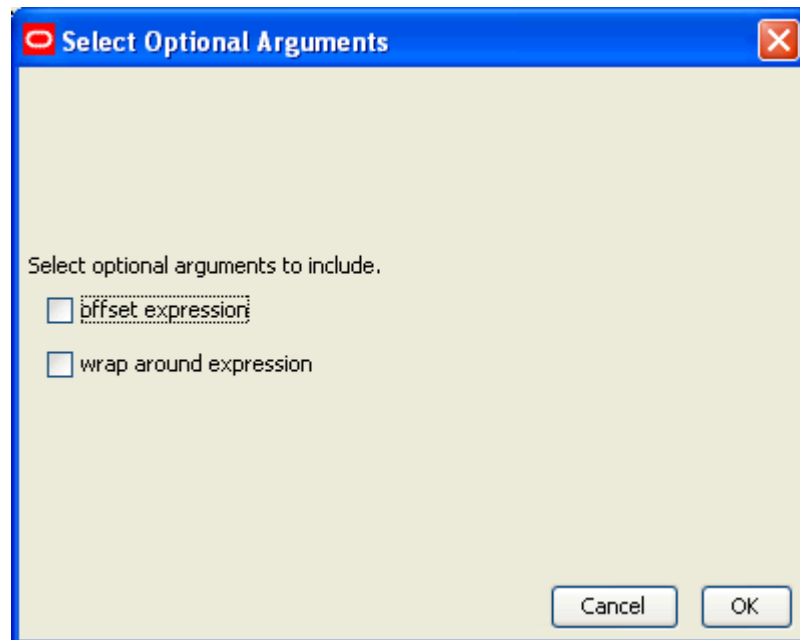
- d. From the drop down list, select a category of functions, procedures, keywords, or modifiers. Double click on a function, procedure, keyword, or modifier that you want to use in the expression to get its name, and (if appropriate) outline syntax pasted at the insertion point in the expression.



The outline syntax of a function, procedure, keyword, or modifier is pasted with components of the syntax separated with braces ('{}'). When the insertion point is in a component of the outline syntax in the expression, you will see a description of the component at the bottom of the expression window. When the insertion point is in a component of the outline syntax, anything that is entered or pasted replaces the whole component.

**Example Expression**

If the function, procedure, keyword, or modifier that is pasted has optional arguments (for example, the cover function has optional arguments for an offset expression and a wrap-around expression), you will be presented with the following dialog box to select which of the optional arguments you want to use in the expression. Note that all arguments are positional, so if you select a later argument, all earlier arguments will be automatically selected, and if you deselect an earlier argument, all later arguments will be automatically deselected.

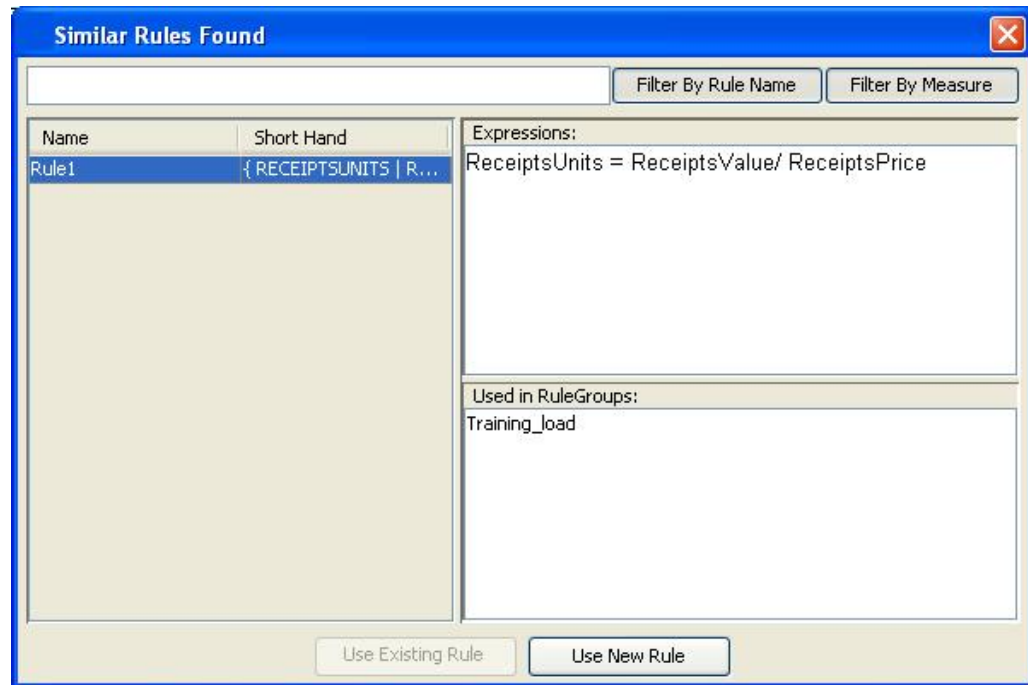
**Select Optional Arguments Dialog Box**

If the function, procedure, keyword, or modifier that is pasted has repeating arguments (for example the min function finds the minimum of a variable number of expressions), you will be presented with a dialog box to select how many of the repeating arguments you want to use in the expression.

- e. Use the keyboard or dialog buttons to add the appropriate mathematical operators and constants to construct the expression.
- f. Click **Finish** to finish entering the expression and close the window.

Note: If you have created an invalid expression, a warning message is displayed.

7. To add further expressions to the rule, click **Add**, and repeat the process of entering an expression.
8. Click **OK** to save any changes and close the window.
9. If the newly defined rule's expressions use exactly the same measures as another rule that already exists in the Rule Pool, the Similar Rules Found window will be displayed. The window shows all rules that use exactly the same measures as the newly defined rule. This gives you an opportunity to use an existing rule from the Rule Pool or to continue with the new rule. The Similar Rules Found window allows you to view rules, associated expressions, and rule groups that contain the rules. The rule table may be filtered based on the rule name or by measure. Click Use Existing Rule or Use New Rule to save changes and close the window.



The new rule is placed above the rule that was selected at the start of the process in the sequence of rules in the rule group or at the end of the rule group if no rule was selected.

Add an Existing Rule to a Rule Group

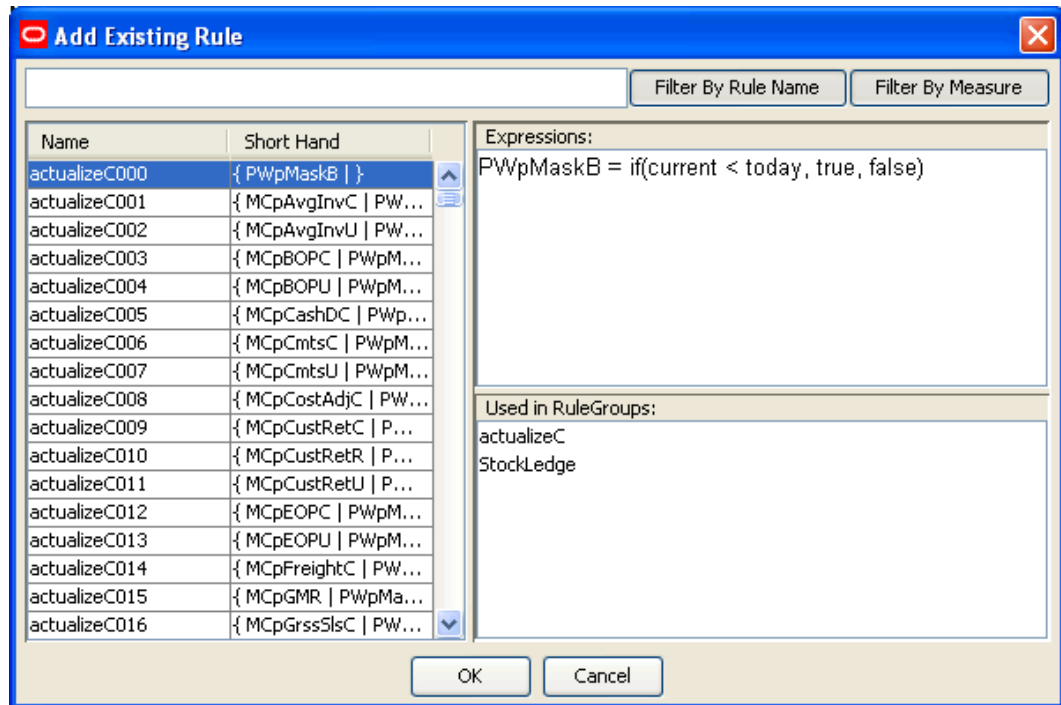
Using the Add Existing Rules dialog, you may select multiple rules to add to a rule group. If at least one of the selected rules is already in the rule group, the OK button will gray out disallowing the operation until that rule is deselected. When multiple rules are selected, the expression and rule group displays will go blank. However, if there is only one selected rule, the rule's expressions and the list of rule groups that use the rule will be displayed.

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Rules**. The Rule Definitions window opens in the workspace.

1. In the Rule Definition window, select the rule group where you want to add an existing rule. If you want the rule to be placed in a particular position in the sequence of rules in the rule group, select the rule that you want the rule to be placed before.
2. Perform one of the following:

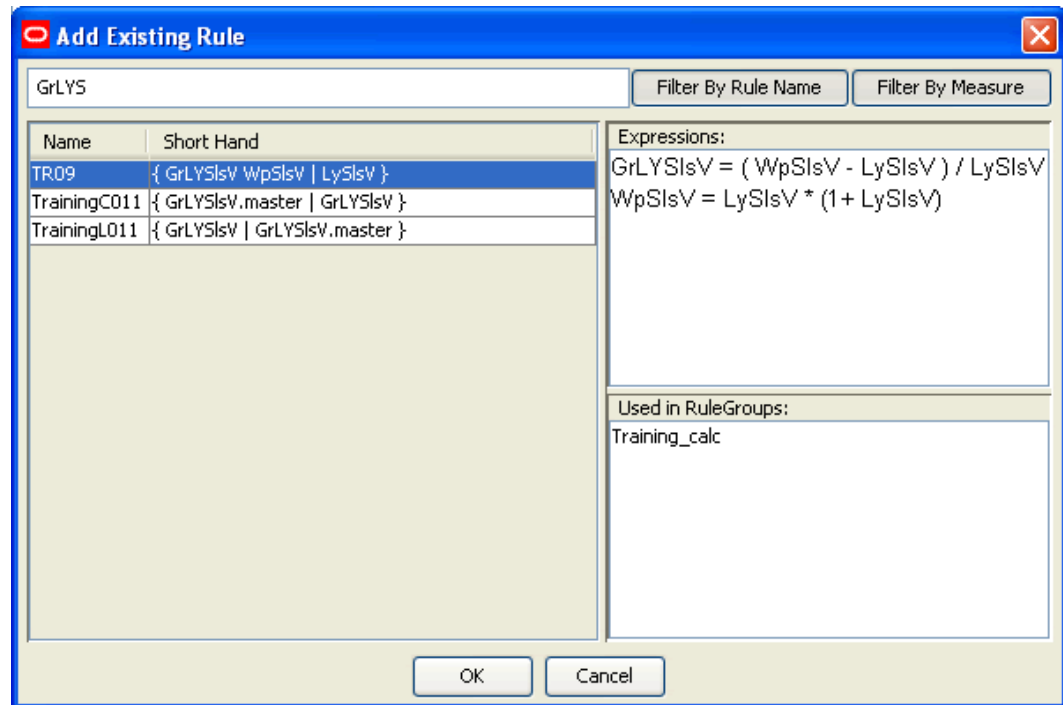
- From the toolbar, click the **New**  button, and select **Using Existing Rule**.
- Select **Create/Using Existing Rule** from the right-click menu.
- Press CTRL-P.

The Add Existing Rule window appears.



Add Existing Rule Window

3. Filter by Rule Name or by Measure and view the associated expressions to find the correct rule. Filtering means that all data are compared, but only matching data are allowed to "pass through." In order to filter by Rule Name or by Measure, you must type a filter string (for instance, 'GrLYS') in the input box directly under the Title Bar. Click **Filter by Measure**, and only those measures with 'GrLYS' will be displayed.



Example of Filter on GrLYS

4. Click on the rule in the table to select it.
5. Click **OK** to save changes and close the window.

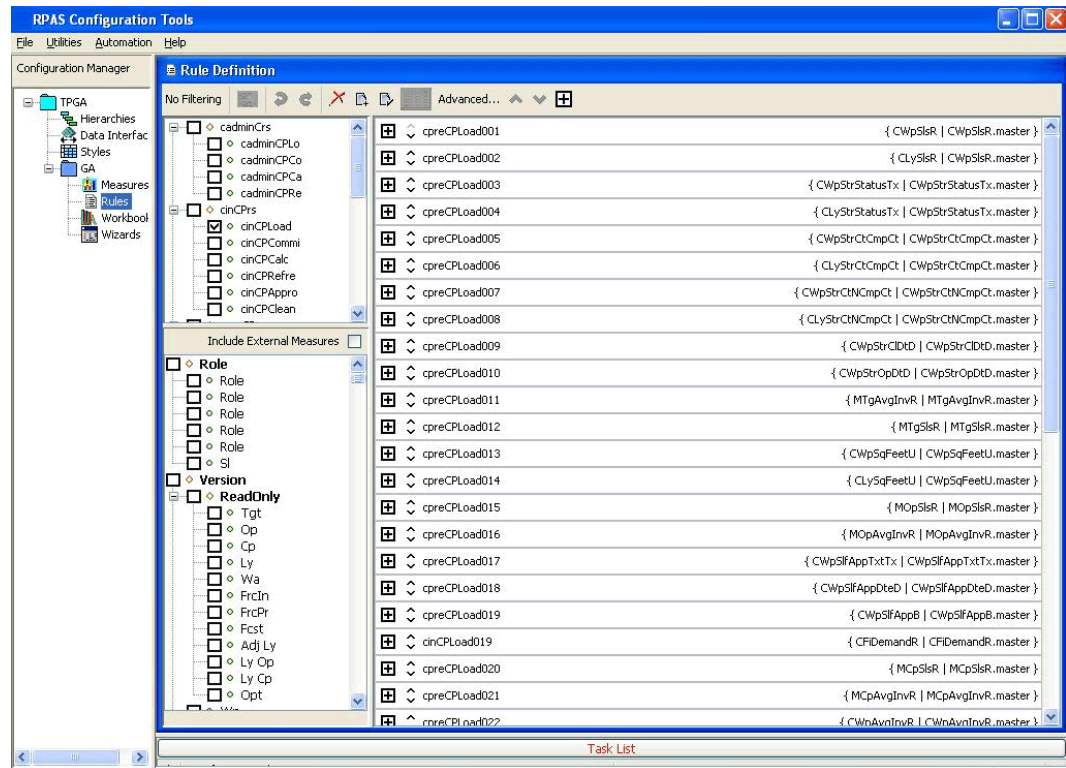
The rule is placed above the rule that was selected at the start of the process in the sequence of rules in the rule group or at the end of the rule group if no rule was selected.

Apply a Rule Pattern to Create New Rules or to Update Existing Rules

The Apply Rule Pattern functionality allows you to create new rules or update existing rules according to a pattern established by a selected 'base' or 'template' rule. The rule tool recognizes inherent similarities or patterns in measure components used in some rules when compared to the base or template rule. Based on these similarities, the rule tool allows for the creation of new rules or update of exiting rules to fit the pattern set by the base or template rule.

When the Apply Pattern capability is enabled, there is a possibility that some of the "New" or "Updated" rules will have the same expressions as a rule that already exists in the rule pool. If this happens, the Similar Rules Found dialog appears and provides the option of using the existing rule or actually creating a new one.

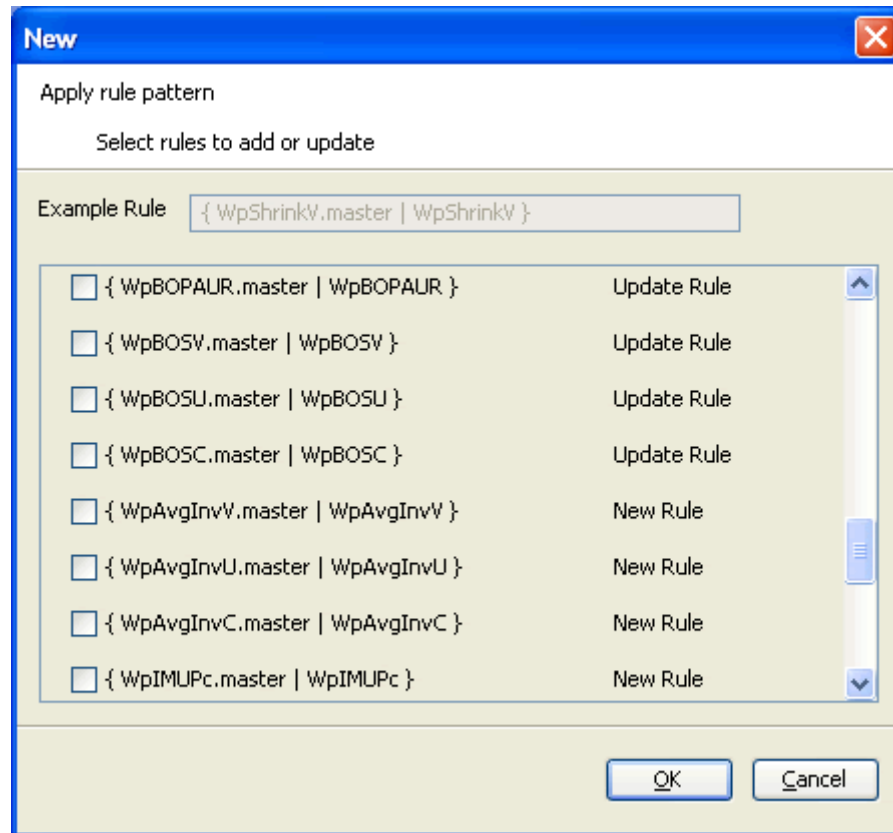
Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Rules**. The Rule Definitions window opens in the workspace.



Rule Definition Window

1. In the Rule Definition window, select any rule group that contains the rule to use as the pattern basis.
2. Select the Rule whose pattern is to be used as a basis for creating or updating rules.
3. From the toolbar, click **Advanced**, or select **Advanced** from the right-click menu.
4. Select **Apply Pattern**.

A New dialog box containing set of rules will be presented for selection. This set is composed of rules whose measures follow the selected rule's pattern in terms of the individual measure components used. The set of rules will be composed of potential new rules or potential updated rules.



New – Apply New Pattern Dialog Box

Example:

Consider a configuration with three components in the measure naming scheme:

- a 'version' (such as 'Wp')
- a metric (such as 'Shrink')
- a unit of measure (such as 'V').

If the selected base rule is:

{ WpShrinkU WpShrinkAUR | WpShrinkV }

then the rules:

Rule1: { WpSlsU WpSlsV WpSlsAUR | }

and

Rule2: { WpRecU WpRecV WpRecC | }

would fit the pattern and be included in the list.

Rule1 fits the pattern because its measures use the same measure components with the exception of the metric component. For the Metric component; the base rule uses Shrink, and Rule1 uses Sls consistently. In this case, the tool will present the rule:

{ WpSlsU WpSlsAUR | WpSlsV }

as a possible update for Rule1.

The update results in the conversion of Rule1 to a rule that uses the same measures as the original Rule1, but it has the expression pattern of the base rule.

Rule2 fits the pattern because it uses the same Version as the base rule. For the Metric component, the base rule uses Shrink, and Rule2 uses Rec consistently. Unlike Rule1, Rule2 uses C as the Unit of Measure in one of its measures. This is not an "exact" fit like

Rule1. In this case, the tool will present the rule { WpRecU WpRecAUR | WpRecV } as a possible new rule. Notice that this rule is “forced” to be an “exact” fit as Rule1 was.

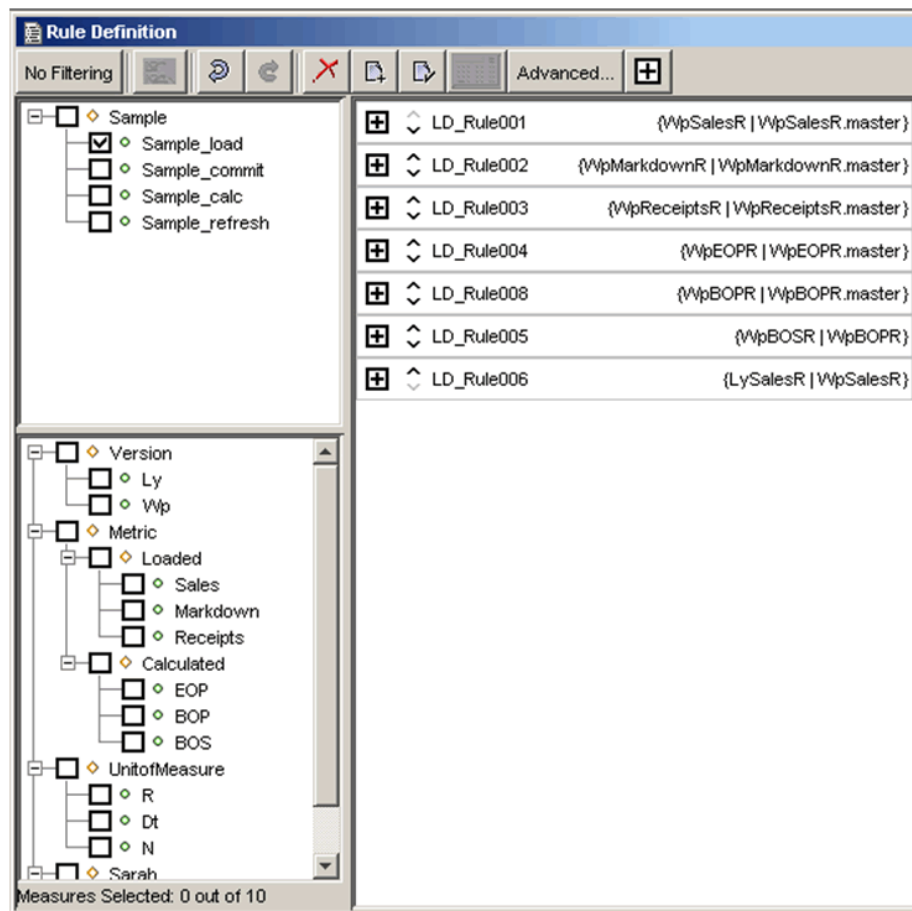
5. From the list of possible new and updated rules, select those that you want to be updated or added to the rule group.
6. Click **OK**.

A selected rule that is labeled “New Rule” will be added to the end of the rule group. The new rule’s name will default to the RuleSet’s name suffixed with a number to keep the name unique. A selected rule that is labeled “Update Rule” is already in the rule group and will be updated. This means that the rule’s measures will be retained, but its expression pattern will be changed to follow the base rule’s expression pattern. In both cases the rule will follow the pattern of the base rule’s expressions.

Delete a Rule from All Rule Groups

Note: This will delete the rule from all rule groups that contain this rule as well as from the Rule Pool. If you just want to remove the rule from a rule group, but retain it in other rule groups, follow the ‘remove a rule from a rule group’ procedure.

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Rules**. The Rule Definitions window opens in the workspace.



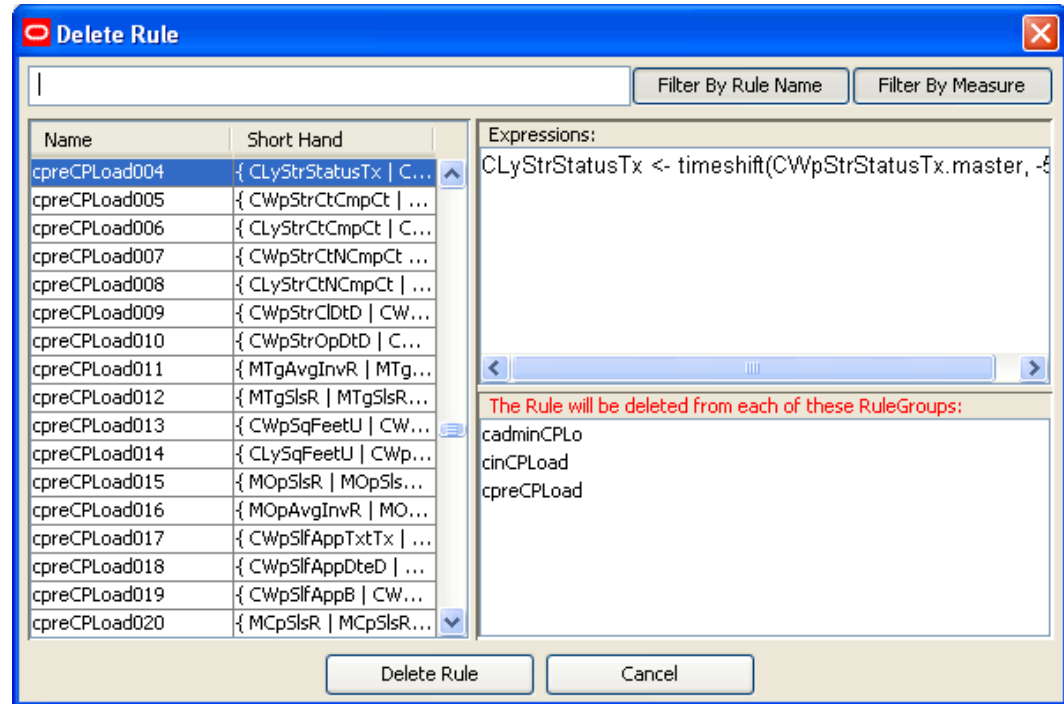
Rule Definition Window

1. In the Rule Definition window, select any rule group that contains the rule that you want to delete.
2. Select the rule that you want to delete.

Note: If you attempt this procedure with multiple rules selected, only the last selected rule gets deleted.

3. From the toolbar, click the **Delete**  button and select **Delete Rule**, or select **Delete – Remove – Delete Rule** from the right-click menu.

The Delete Rule window will be displayed.



Delete Rule Window

4. Verify that the rule selected in the table is the rule to be deleted, or select a different rule to delete by clicking on the rule in the table.
5. Click the **Delete Rule** button to delete the rule and close the window.

Remove a Rule from a Rule Group

Note: This will remove the rule(s) only from the currently selected rule group. If you want to delete the rule(s) from all rule groups and the rule pool, follow the 'delete a rule from all rule groups' procedure.

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Rules**. The Rule Definitions window opens in the workspace.

1. In the Rule Definition window, select the rule group that contains the rule(s) that you want to remove.
2. Select the rule(s) that you want to remove.

Note: You can select multiple rules by holding the CTRL key as you click the rules to select, or by clicking one rule and holding SHIFT as you click on another rule, which selects all rules between the two that were clicked. A selected rule is indicated by a bold rule name.

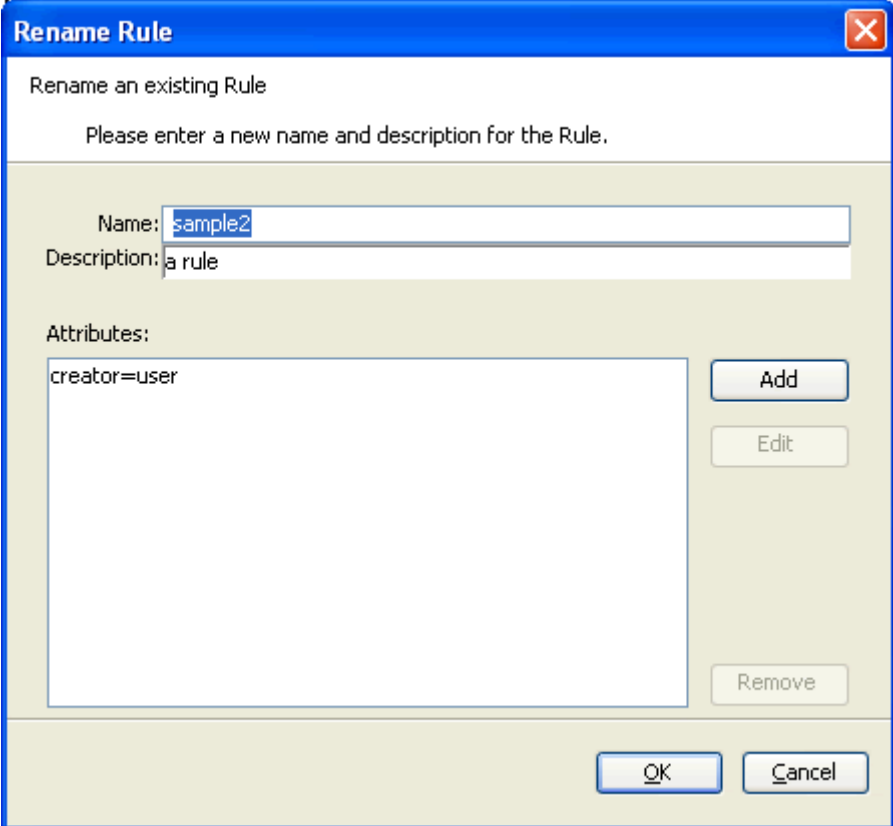
3. From the toolbar, click the **Delete**  button and select **Remove Rule(s)**, or select **Delete/Remove — Remove Rule(s)** from the right-click menu.

The rule(s) are removed from the rule group, but are still in the rule pool. The rules will be permanently lost when the project is closed, unless they are used in another rule group.

Edit Properties of a Rule

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Rules**. The Rule Definitions window opens in the workspace.

1. In the Rule Definition window, select any rule group that contains the rule that you want to edit.
2. From the rule group, select the rule that you want to edit.
3. From the toolbar, click the **Edit**  button and **Select Rule**, or select **Rename/Rule** from the right-click menu. The Rename Rule window opens.



Rename Rule

Rename an existing Rule

Please enter a new name and description for the Rule.

Name:

Description:

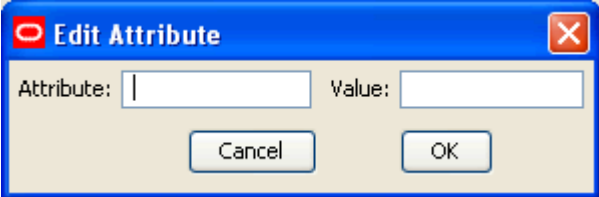
Attributes:

Rename Rule Dialog Box

4. To edit the name of the rule, enter a new name in the **Name** field.

Note: A rule name can be a maximum of 24 alphanumeric or underscore characters. It must not have a name that is the same as any other rule that exists in the project. Rule names may start with a letter or an underscore, but may not start with the letter 'r' or 'R' followed by a number.

5. To edit the description of the rule, enter a new description in the Description field.
6. To edit attributes for a rule:
 - a. Select the attribute that you want to edit.
 - b. Click **Edit**. The Edit Attribute window opens.



Edit Attribute

Attribute:

Value:

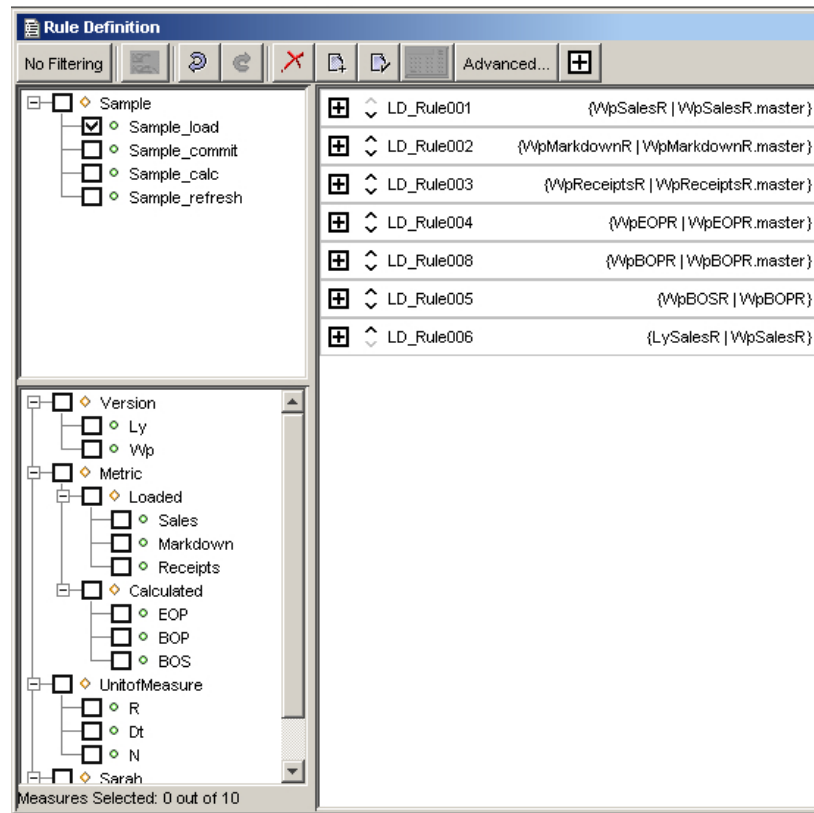
Edit Attribute Dialog Box

- c. Update the information as necessary.
 - d. Click **OK** to save any changes and close the window.
7. To remove attributes from a rule:
 - a. Select the attribute that you want to delete.

- b. Click **Remove**. The attribute is removed from the display box.
8. Click **OK** to save any changes and close the window.

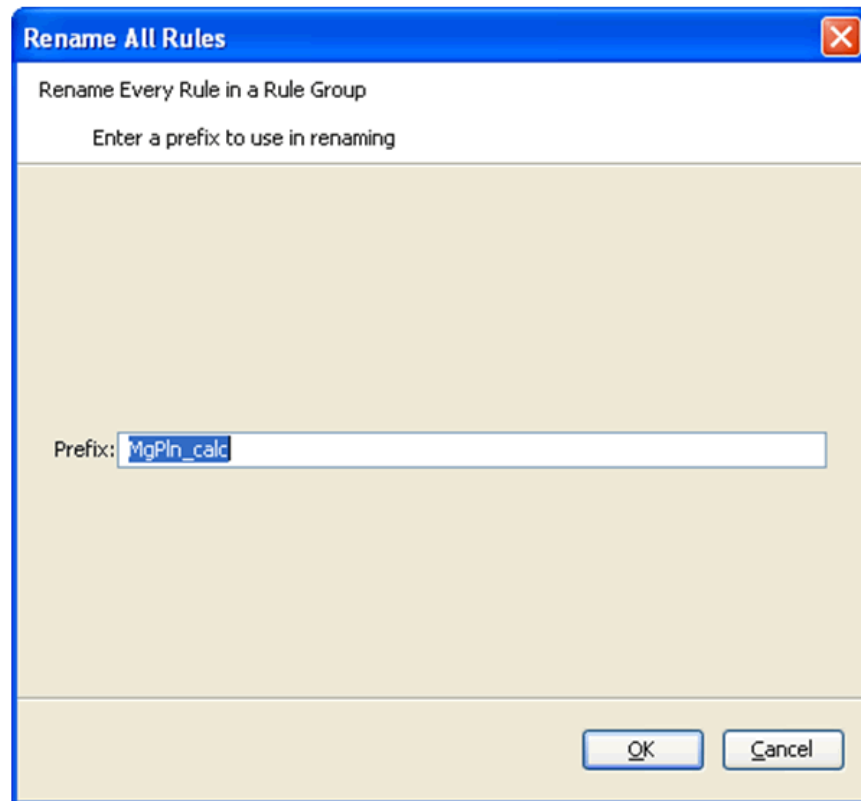
Rename All Rules in a Rule Group

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Rules**. The Rule Definitions window opens in the workspace.



Rule Definition Window

1. In the Rule Definition window, select the rule group that contains the rules that you want to rename.
2. From the toolbar, click **Advanced** and select **Rename All Rules**, or select **Advanced/Rename All Rules** from the right-click menu. The Rename All Rules dialog box appears.

**Rename All Rules Dialog Box**

3. In the Prefix field, enter a prefix up to ten characters long, which will be the start of all rule names.
4. Click **OK**. The process cannot be undone, so you are prompted to confirm your decision.
5. Click **Yes**.

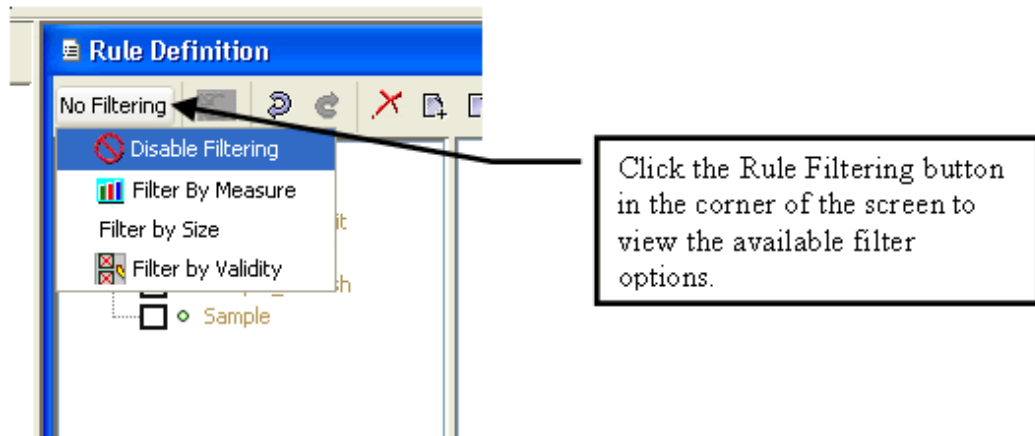
Note: All of the rules in the rule group are renamed with the prefix followed by a 4-digit numeric identifier generated by the rule tool. The rule tool will maintain the order that the rules were in before they were renamed, and it uses that order in generating the numeric identifier.

Note: Since rules may appear in more than one rule group, use of this feature may generate rule names that look out of place in other rule groups, especially if the prefix implies the rule group.

Filter Rules in a Rule Group

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Rules**. The Rule Definitions window opens in the workspace.

1. In the Rule Definition window, select a rule group.
2. In the Rule Definition window, click the **Rule Filtering** button.



Rule Filtering Button in Rule Definition Window

Note: This is a dynamic button, and the text will change depending on the current filter mode.

3. Select one of the following options:
 - **Disable Filtering** – All rules are displayed.
 - **Filter by Measure** – Works in conjunction with the measure components box in the bottom left corner of the screen. Rules are filtered to show those that include any measures that conform to the selected component scheme.
 - **Filter by Size** – Rules are filtered to show just those with more than one expression.
 - **Filter by Validity** – Only invalid rules are displayed.

Note: When rule filtering is active, the buttons to reorder rules in the rule group are disabled.

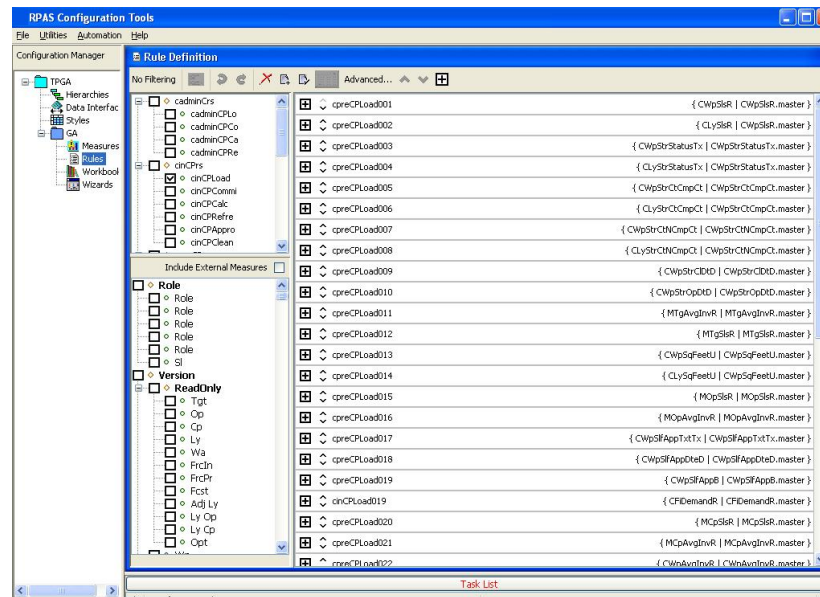
Reordering Rules in a Rule Group

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Rules**. The Rule Definitions window opens in the workspace.

1. In the Rule Definition window, select the rule group that contains the rules that you want to reorder.
2. Select the rule that you want to reorder and perform one of the following options.
 - Use the Up/Down arrows   on the rule definition toolbar to move the rule up or down the list.
 - Click the up/down arrows   on the left hand side of the rule name to move the rule up or down the list.

Auto Generate Load and Commit Rules

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Rules**. The Rule Definitions window opens in the workspace.



Note: Rules for the load and commit rule groups in a rule set can be auto-generated based on the calc rule group. The measures referenced in the calc rule group are assumed to be all of the measures in a workbook (if there are others, manually add their load and/or commit rules). A load or commit rule is generated for all of those measures that have a database allocated (those that are physically stored).

1. Select the rule set that you want to auto generate load or commit rules for, or select any rule group in that rule set.
2. Perform one of the following options:
 - From the toolbar, click **Advanced** and **Select Generate Load Rules** or **Generate Calc Rules**.

- Select **Advanced/Generate Load Rules** or **Advanced/Generate Calc Rules** from the right-click menu.

As this process cannot be undone, you are prompted to confirm your decision.

3. Click **Yes**.

Rules are automatically generated and named for the load or commit rule group. There is one rule with a single expression that is generated for each measure used in the calc rule group for the rule set that has a database assigned. In the load rule group, the rules are named <rulesetname>Lnnn where nnn is a 3-digit order number. The rules in a commit rule group are similarly named <rulesetname>Cnnn. The expression in a generated rule in a load rule group is of the form:

<measurename> = <measurename>.master

and in the generated commit rule group are of the form:

<measurename>.master = <measurename>

Copy Selected Rules to Another Rule Group

When copying selected rules of a rule group into another rule group, it is possible to specify whether existing rules will be used or copies of the rules will be created. The Use Existing Rules checkbox defaults to using any existing rules in the rule pool. If this checkbox is checked, the copy selected rules operation will use the same rules that the source rule group has. If this checkbox is unchecked, the copy selected rules operation will create copies of the rules and use those copies for appending to or replacing rules in the destination rule group.

When using the Find/Replace feature, it is possible that a changed rule will have the same expressions as a rule that already exists in the rule pool. If the Use Existing Rules checkbox is selected, the Similar Rules Found dialog provides the option of using the existing rule or actually creating a new one.

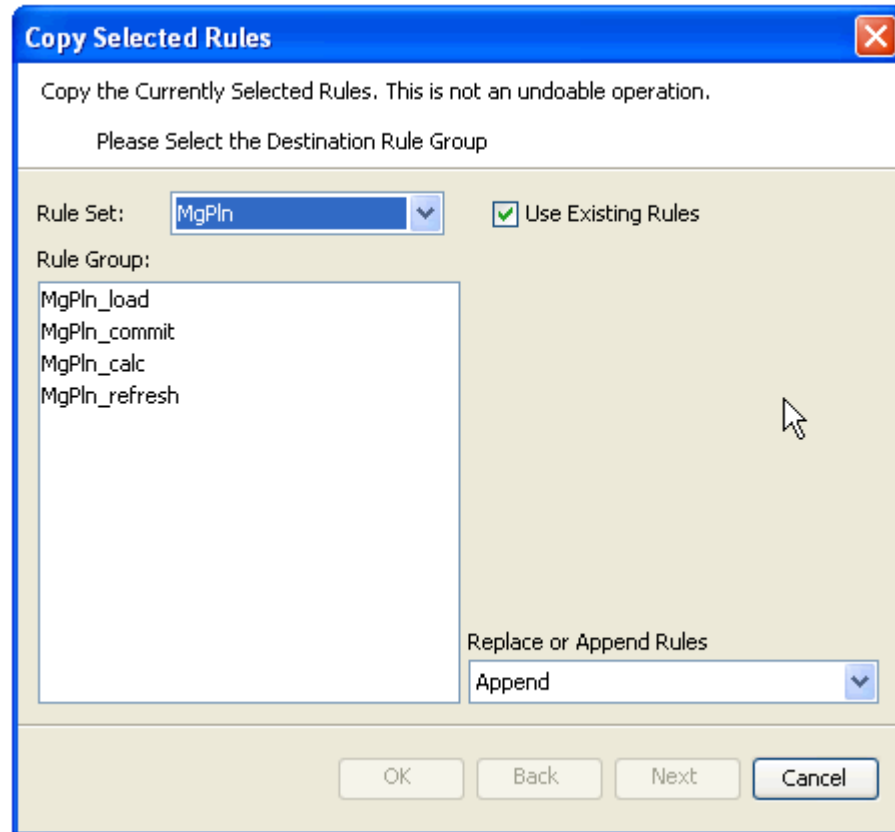
Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Rules**. The Rule Definitions window opens in the workspace.

1. Select the Rule Group that contains the rules you want to copy, and select the individual rules to be copied.

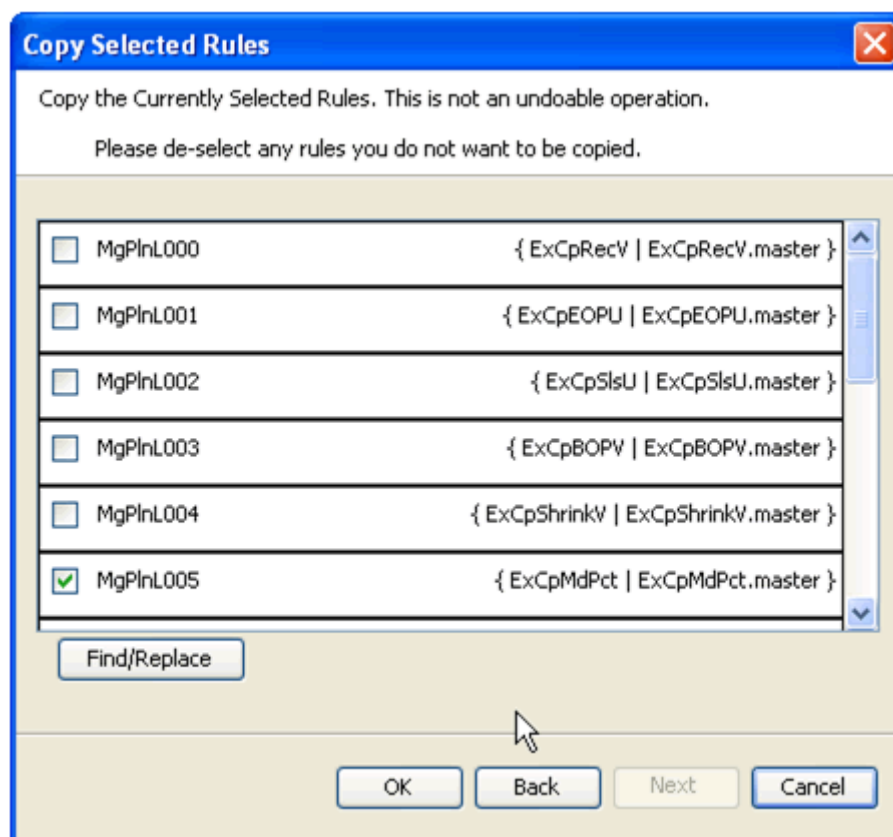
Note: To select multiple rules, hold down the CTRL key and click the rules to select, or click one rule and hold SHIFT as you click on another rule, which selects all rules between the two that you click. A selected rule is indicated by a bold rule name.

2. Perform one of the following options:
 - From the toolbar, click **Advanced**, and select **Copy Selected Rules**.
 - Select **Advanced – Copy Selected Rules** from the right-click menu.
 - Press CTRL-C.

The Copy Selected Rules window opens.

**Copy Selected Rules Dialog Box**

3. In the Rule Set field, select the copy's destination rule set.
4. In the Rule Group area, select the desired copy's destination Rule Group.
5. In the Replace or Append Rules field, select:
 - **Replace** – to remove all rules that already exist in the destination rule group before the copy.
 - **Append** – to add to the rules already in the destination rule group.
6. Click **Next**. The second window of the Copy Selected Rules window opens. This window you to select or deselect rules from those originally selected to copy when you click on the check box beside the rule name.



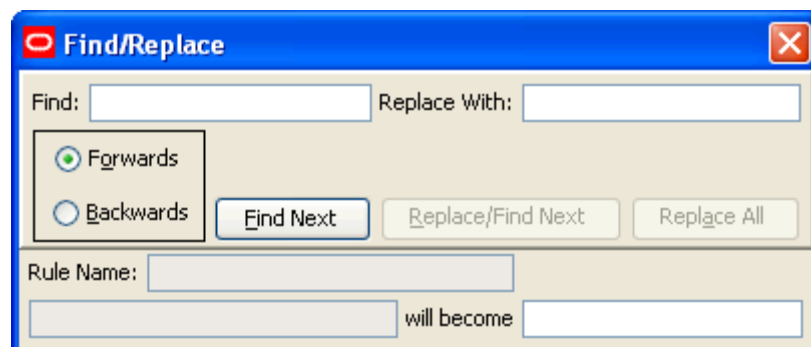
Copy Selected Rules – Rules to be copied appear with check marks

7. Verify the rules to be copied and click **OK**.

Find and Replace Measures in the Copied Rules

Note: The ability to find/replace in the copy rules is a very powerful and useful feature. It allows the builder to build a collection of rules and “clone” them to a very similar collection of rules. For example, a collection of rules that calculate a series of variances with one version can be cloned to produce rules that calculate a series of variances with another version.

1. Click the **Find/Replace** button. The Find/Replace window is displayed.



Find/Replace Dialog Box

2. In the Find field, enter the portion of the measure that you want to replace.

Note: The Find function is case sensitive.

3. In the Replace With field, enter the string to replace the portion of the measure name that you are finding.
4. Perform one of the following options:
 - Select **Forwards** to search the rules in order
 - Select **Backwards** to search the rules in reverse order.

Note: Searching Forwards will proceed from left to right starting with the first measure of the first expression of the first selected Rule and will go through all expressions in all selected Rules. Similarly, searching Backwards will flow in the reverse direction starting with the rightmost measure of the last expression in the last Rule selected.

5. Click **Find Next**. The first candidate measure to be replaced will be displayed in the bottom left field.
6. Perform one of the following:
 - Click **Replace/Find Next** to replace the current candidate measure and display the next candidate measure.
 - Click **Replace All** to replace all instances in all the selected Rules of the current candidate measure. For example, if a search for Wp finds WpRecV, clicking on Replace All will perform a replace on all instances of WpRecV in all the selected Rules.
 - Click **Find Next** to skip over that occurrence of the portion of the measure name, and go onto the next one.
7. Click the **Close**  button to close the Find/Replace box.
8. Click **OK**.
9. Click **Yes** to confirm your decision.

The copies of the rules will be placed in the target rule group. These copies will have names that start with as many characters as possible from the name of the original rule and end with an underscore and number.

Expressions and Rules

An expression describes and solves the relationship between measures in a way that causes a measure to be calculated through the expression. In some cases, there may be business reasons for wanting more than one of the measures in a relationship to be calculable or solvable through that relationship. Expressions are the basis of all calculations of the relationships between measures, and they are evaluated by the calculation engine during a calculation. They are written in a syntax that allows for the calculation of a single measure from other measures, constants, and parameters by using standard arithmetical functions and a rich set of mathematical, technical, and business functions. Expressions have multiple results.

Example:

Expression 1: $\text{ReceiptUnits} = \text{ReceiptsValue} / \text{ReceiptsPrice}$

This expression specifies the way ReceiptUnits are calculated. ReceiptUnits are calculated by the division of ReceiptsValue over ReceiptsPrice.

Note: Measures are not only calculated based on expressions. They are also calculated based on spreading and aggregating.

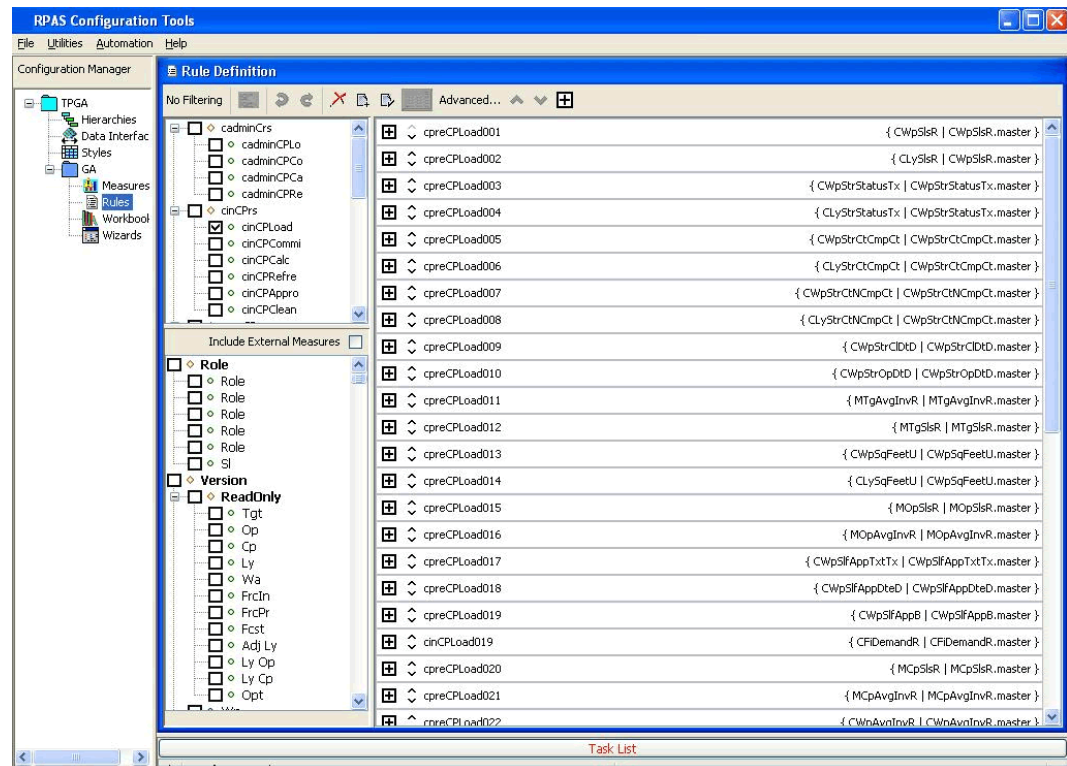
Non-Conforming Recalc Measures

Warning: Under complicated and rare circumstances, having a recalc measure on the right-hand side of an expression that calculates a measure whose base intersection is higher than that of the recalc measure, or using the level modifier on a recalc measure on the right-hand side of an expression can cause incorrect values to be calculated. These incorrect values can then have a knock-on effect onto other measures. Therefore, it is recommended that the expressions be written to replace the recalc measure on the right-hand side of the expression with the 'recalc expression' for the recalc measure.

Explanation: The problem is caused by a complex set of interactions in the way that recalc measures are calculated when the recalc measure is used in an expression above its base intersection. This can occur with the level modifier or with a "non-conforming" calculation. Even when a recalc measure is not calculated through evaluation of an expression, it still must be 'aggregated' (by applying its "recalc rule") at aggregated levels if any of the measures in its recalc rule have changed. This process is not considered in determining the evaluation sequence of expressions, so it is possible that an expression that requires a "reaggregated" value of a recalc measure may be sequenced before the evaluation of the expression in the recalc measure's recalc rule, thus causing incorrect values to be calculated.

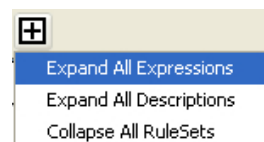
Reorder an Expression in a Rule

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Rules**. The Rule Definitions window opens in the workspace.



Rule Definition Window

1. In the Rule Definition window, select any rule group that contains the rule whose expression you want to reorder.
2. Perform one of the following:
 - Expand the rule to view the expressions associated with the rule. Click on the Toggle button in the toolbar and select **Expand All Expressions**.



- Click the toggle button  for the rule.
3. Use the up and down arrows  to move the expression up or down the list.

Edit an Expression in a Rule

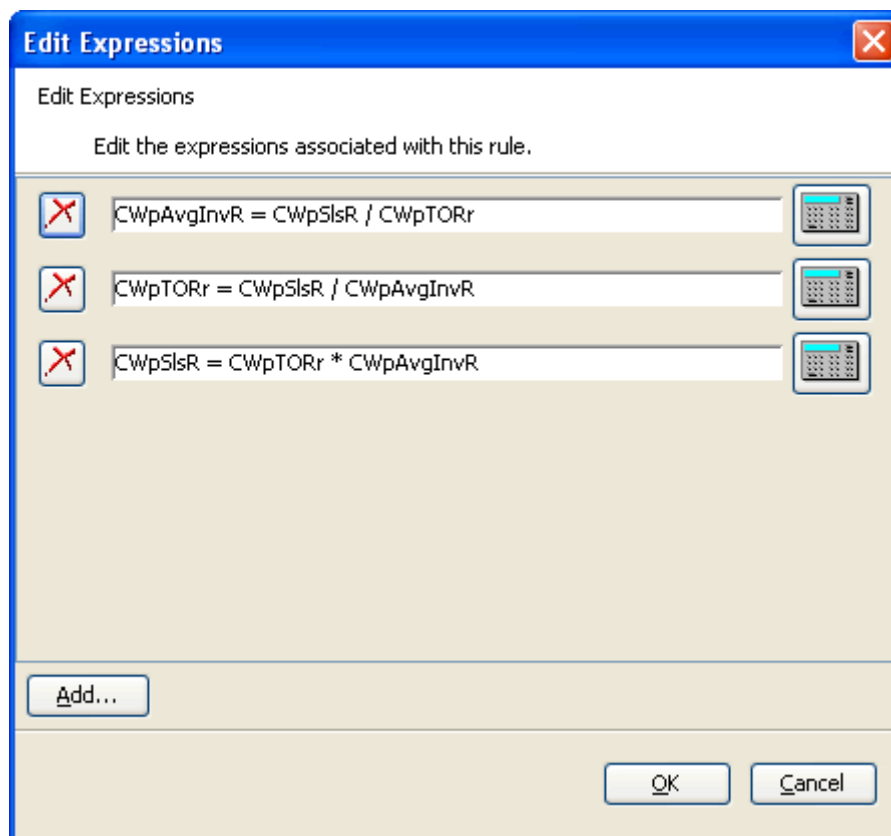
Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Rules**. The Rule Definitions window opens in the workspace.

1. In the Rule Definition window, select any rule group that contains the rule whose expression you want to edit.
2. In the Rule Definition Window, select the rule whose expression you want to edit.

2. Perform one of the following options:

- From the toolbar, click the Expression Builder  button.
- Select **Edit Expressions** from the right click menu.
- Press CTRL-E.

The Edit Expressions window appears.

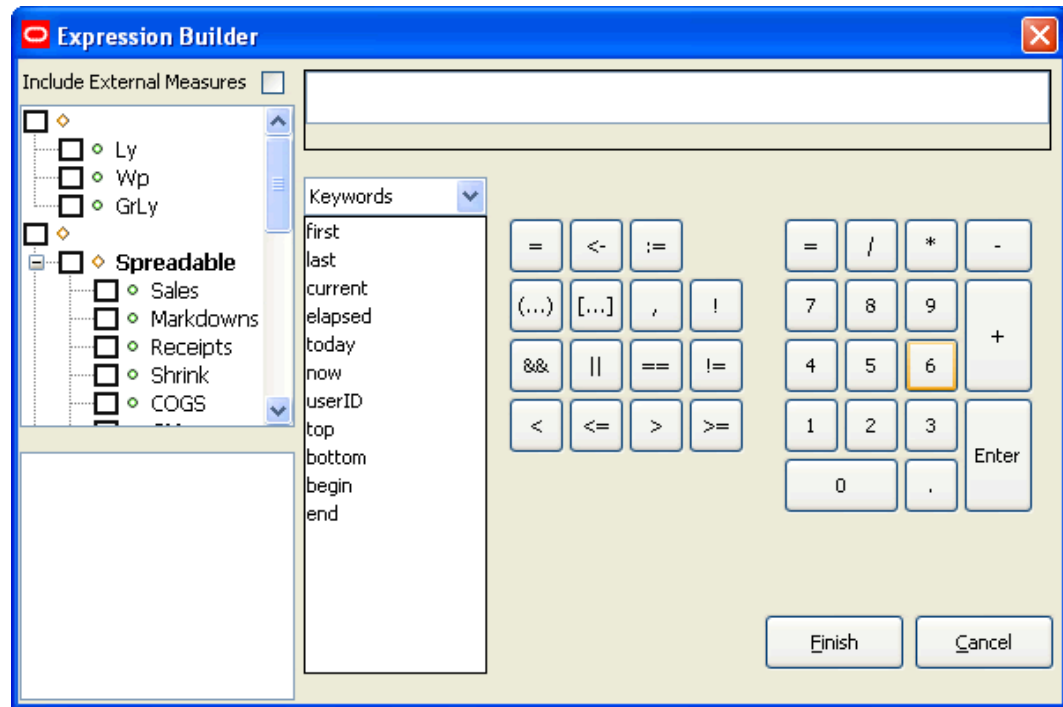


Edit Expressions Window

To Edit an Expression

1. Perform one of the following:

- Edit the expression in its text box.
- Click the Expression Builder  button for the expression you want to edit. The Edit Expressions window is displayed. Use the Expression Builder to make your changes.



Expression Builder Window

2. Click **Finish**.
3. Click **OK** to save changes and close the window.

Delete an Expression from a Rule

Navigate: In the Configuration Manager, select **Project** – **Solution** – **Rules**. The Rule Definitions window opens in the workspace.

1. In the Rule Definition window, select any rule group that contains the rule whose expression you want to delete.
2. In the Rule Definition Window, select the rule whose expression you want to delete.
3. Perform one of the following options:

- From the toolbar, click the **Expression Builder** button.
- Select **Edit Expressions** from the right-click menu.
- Press CTRL-E.

The Edit Expressions window is displayed.

4. Click the **Delete** button to the left of the expression in the Edit Expression box.
5. Click **OK** to delete the expression. Once you click **OK**, the expression will be permanently deleted from the rule.

Note: If you delete the only expression in the rule, the rule will be flagged as being invalid, because it has no expressions.

Add an Expression to a Rule

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Rules**. The Rule Definitions window opens in the workspace.

1. In the Rule Definition window, select any rule group that contains the rule that you want to add an expression to.
2. In the Rule Definition Window, select the rule that you want to add an expression to.
3. Perform one of the following:

- From the toolbar, click the **Expression Builder**  button.
- Select **Edit Expressions** from the right-click menu.
- Press **CTRL-E**.

The Edit Expressions window appears.

4. Click the **Add** button to add a new expression in the Edit Expression box.
5. Enter the expression in its text box, or click the Expression Builder  button for the expression you want to edit. The Edit Expressions window is displayed. Use the Expression Builder to enter the expression.
6. Click **Finish**.
7. Click **OK** to add the expression.

RPAS Functions, Procedures, Keywords, and Modifiers

RPAS functions, procedures, keywords and modifiers are mechanisms for performing operations within an expression that are controlled and executed by the calculation engine. There is a rich collection of available functions, procedures, keywords, and modifiers that can be further extended for an implementation if required.

See “Appendix C – RPAS Rules Function Reference Guide” for details about RPAS functions, procedures, keywords, and modifiers.

Workbooks

Overview

A workbook is the easily viewed, easily manipulated multidimensional framework that is used to perform interactive business functions in the configured solution. To present data, a workbook can contain any number of multidimensional spreadsheets, called worksheets, as well as graphical charts and related reports. All of these components work together to facilitate the viewing and analysis of business functions.

The workbook designer allows you to create, select, and integrate the various components of a workbook template, which is a pre-designed workbook that is formatted for RPAS users to view and manipulate data. It contains workbook tabs, worksheets, rule groups, wizards, and workflow processes.

Take the time to design a well-planned workbook. Workbooks should be laid out in a logical format and should be easy to navigate. When you are building a workbook, think about how the workbook will be used. Understand the business process flow and what end users will need to access most. Most likely, this information should be contained in the first workbook tab and worksheet.

The names of all of the workbook components should be intuitive to an end user.

Note: The internal RPAS names need to be unique across all workbook components in a project. This includes workbook, tab, worksheet, wizard, and custom menu names.

Workbook Tabs

A workbook tab is a major subdivision of a workbook. Each workbook contains at least one workbook tab by default, but additional tabs can be added for organizing workbooks to support business needs. The workbook designer allows you to define and name tabs and to specify their order in the workbook.

Worksheets

Worksheets are multidimensional spreadsheets that are used to display workbook-specific information. Workbooks can include one or many worksheets. Worksheets can present data in the form of numbers in a grid, or the numeric data values can be converted to a graphical chart.

Rule Groups

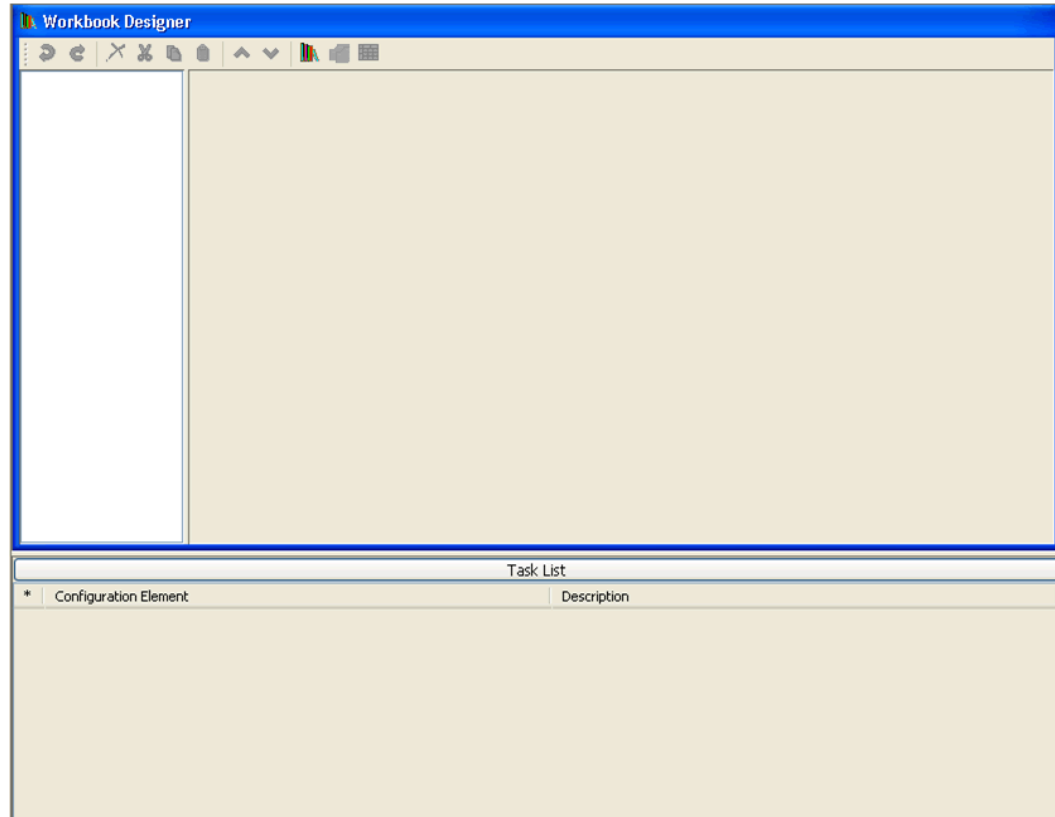
Rule groups are a collection of rules that are treated as a unit by the calculation engine. The rules in the rule group must be considered together in order to satisfy the calculation requirements for a specific business process. The sequence of rules in a rule group determines the calculation sequence unless the sequence is forced. You choose a rule set to select the rule groups to be used in a workbook.

Wizards

RPAS uses a series of wizards to obtain information in order to build a workbook. The workbook contains a subset of the entire data available in the system; so think about the most logical flow for the wizards. The main purpose of a wizard is to allow you to make choices regarding the scope of the workbook. The workbook designer allows you to specify which wizards will be used to build the workbooks.

Create a Workbook

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Workbooks**. The Workbook Designer window opens in the workspace.



Workbook Designer Window

1. Perform one of the following options:

- Click the **New Workbook**  button on the top of the Workbook Designer.
- Select **New Workbook** from the right-click menu.
- Select an existing workbook and press Insert.

A new workbook is created.

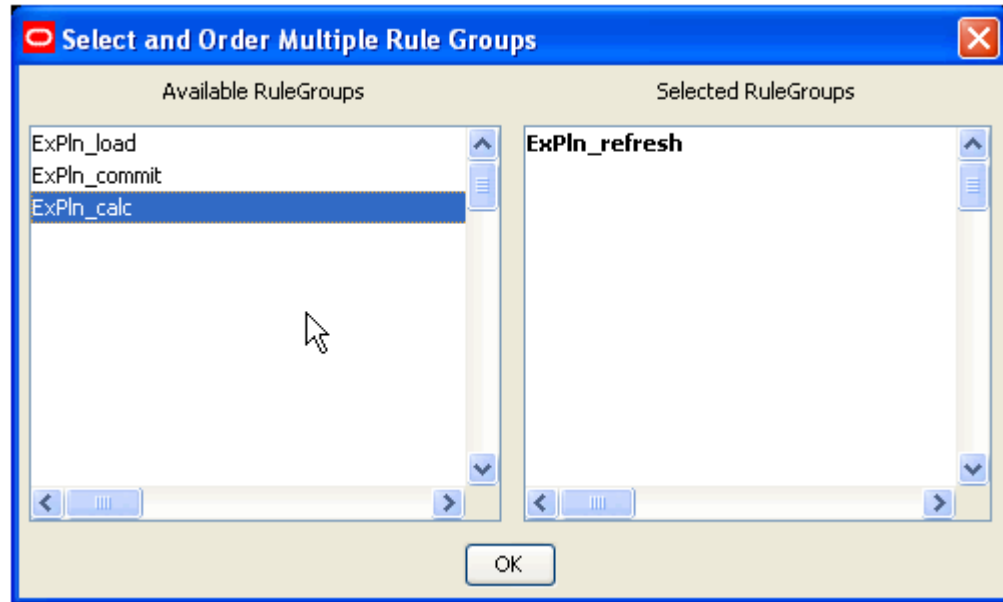
2. Enter information on the following tabs as necessary, which are described in the following topics.

General Tab

- **RPAS Name** – The RPAS internal name of the workbook.
- **User Label** – The label that you will see when selecting which workbook to build.
- **Group** – In the client, workbooks are grouped together under tabs to make it easier for you to find and select the workbook that you want to build in solutions that have multiple workbooks. This is the internal RPAS name of the group that this workbook will belong to.
- **Group Label** – In the client, workbooks are grouped together under tabs to make it easier for you to find and select the workbook they want to build in solutions that have multiple workbooks. This is the label that you will see of the group that this workbook will belong to. If different labels are entered for the same workbook group against different workbooks, the workbook group label shown to you will effectively be arbitrary.
- **Workbook Type** – This property is reserved for use when custom extensions are written. It enables the custom extension to determine the 'type' of the template,

where the template type has a meaning defined by the custom extension writer. By default, when there is no custom extension, this field is set to the value "DynamicTemplate."

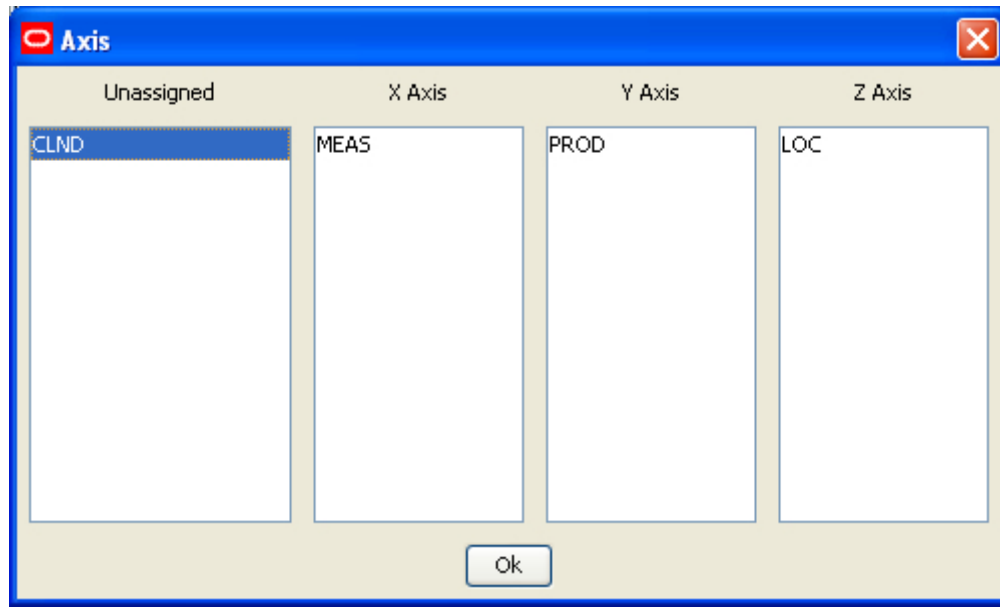
- **Library** – When the workbook type is not "Dynamic Template," it needs to be associated with a relevant custom shared library. This field holds the name of that library. By default, when there are no custom extensions, this field is set to "Template." The name entered here needs to be consistent with the custom extension. For example, if a value of ABCTemplate is entered in this field, the custom library needs to be named ABCTemplateLib and the directory where the custom extension looks for configuration files in the domain will be repos/ABCTemplates.
- **Wizard Only** – The Wizard Only checkbox is only used under limited circumstances when custom code is to be executed in a batch job at the end of the wizard process (which usually uses custom wizards), instead of building and opening a normal workbook. The selections you made in the wizards are passed to the custom code. Therefore, a workbook with the Wizard Only checkbox checked is not really a workbook at all. However, the workbook infrastructure is used so that the process can have a name and label, and be assigned to a workbook group, so the process can be selected by end users in the same manner as workbooks. Therefore, workbooks that have the Wizard Only checkbox checked do not need tabs or worksheets defined, but they do need a name, label, and workbook group.
- **Rule Set** – Select the rule set that you want to use with the workbook. The list of rule sets to select from includes all the rule sets in the same solution as the workbook template.
- **Use Default Rules** – Select the check box to use the default rules (Load, Commit, Calc, and Refresh) associated with the rule set. If this check box is selected, the Load Rules, Commit Rules, Calc Rules, and Refresh Rules properties are disabled. If the check box is not selected; the Load Rules, Commit Rules, Calc Rules, and Refresh Rules properties are enabled.
- **Load Rules, Commit Rules, and Calc Rules** – Select the rule group that you want to apply for each rule group type. Only rule groups from the selected rule set are offered.
- **Refresh Rules** – Select the rule group(s) that you want to use as refresh rule groups. Click once in the field. The Select and Order Multiple Rule Groups window opens:



Select and Order Multiple Rule Groups Dialog

Within this window, you will specify which groups will be available to be used to refresh the workbook. You can refresh part of the data within a workbook by refreshing with a rule group that only updates some of the measures in the workbook. The order that the rule groups appear in is the order that they will appear to the end user when presented with a choice of refresh rule groups within the workbook.

- a. In the Available Rule Groups column, select the rule group to add; and drag it to the **Selected Rule Groups** column.
 - b. Click **OK** to save any changes and close the window.
- **X Axis, Y Axis, Z Axis and Unassigned** –These properties are used to define the default axis layout of the worksheets in the workbook (that is, which hierarchies will appear in each axis).
 - a. Click the **X-Axis**, **Y-Axis**, **Z-Axis**, or **Unassigned** field. The Axis dialog box opens.



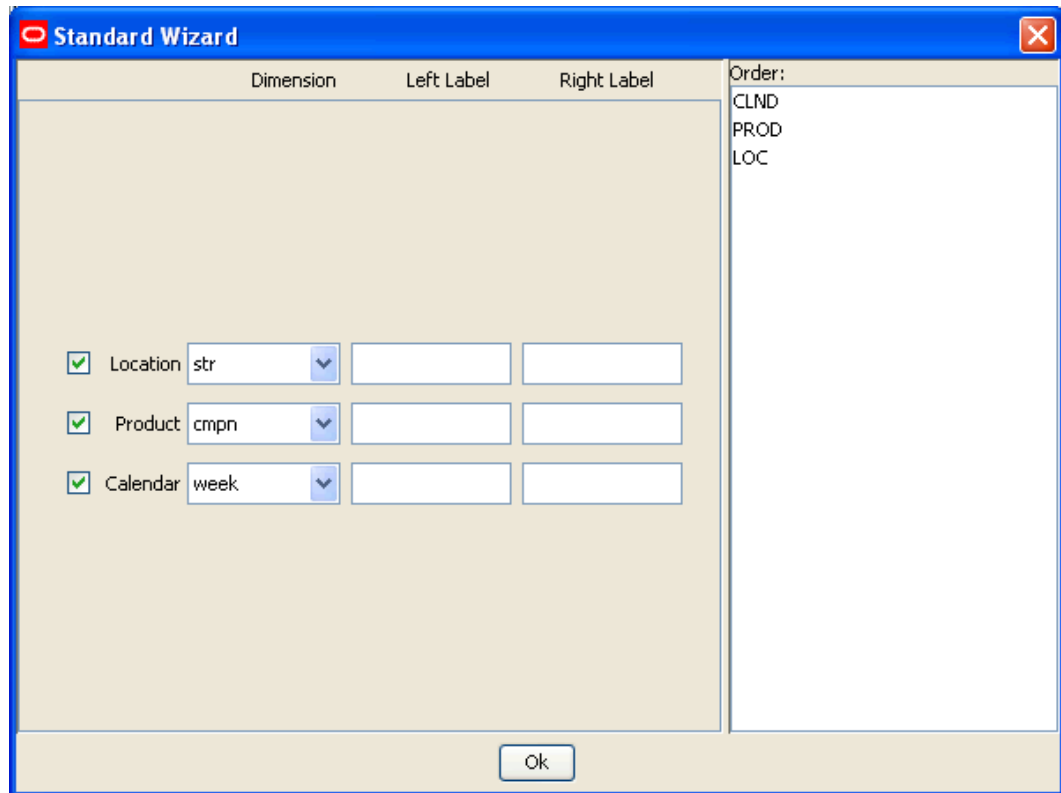
Axis Dialog Box

- b. Drag the hierarchies to the appropriate axis column.
- c. Click **OK** to save any changes and close the window.

Note: The Hierarchies that appear in this process are the hierarchies used by measures placed on worksheets in the workbook. If no worksheets have yet been built, no hierarchies will appear in this process.

- **Use Custom Wizard** – Determines the type of wizard to use for the workbook template. Select the check box to enable the Custom Wizard property and disable the Standard Wizard Property. Unselect the check box to disable the Custom Wizard property and enable the Standard Wizard Property. Custom Wizards are discussed in detail in the Section on Wizards which follows this section on Workbooks.
- **Custom Wizard** – Select the custom wizard that you want to use to build the workbook. This field becomes available when the Use Custom Wizard check box is checked. This field allows you to select a wizard from a list of wizards created in the Wizard Designer.

- **Standard Wizard** – Select the dimensions to be selected by the end-user in the standard wizard, which presents a series of “two tree” selection panes to select the positions in the scope of the workbook to be built.
 - a. Click the **Standard Wizard** field. The Standard Wizard dialog opens.



Standard Wizard Dialog

Note: Hierarchies that are used in the base intersections for the measures used in the rule set assigned to the workbook are displayed. For each hierarchy, there will be a choice of dimensions. The dimensions offered will be the lowest dimension in that hierarchy that is used in the base intersection of a measure in the rule set assigned to the workbook, plus all higher dimensions.

- b. Select the check boxes next to the hierarchy names to enable the hierarchies that the end-user should select positions for.
- c. Select the desired dimension from each of the enabled hierarchies.

Note: The dimension selected will be the lowest dimension offered to the end-user in the scope selection wizard during the workbook build process. However, the workbook requires positions at the lowest dimension offered. Therefore, if the selected dimension is higher than the lowest dimension offered, the scope of the workbook will include all the positions in that lowest dimension that are descended from the positions selected from that higher dimension.

- d. Enter the left and right labels for each hierarchy. These labels will be displayed on the left and right trees of the corresponding 2-tree wizards during the workbook build process. If these fields are left blank, no labels will be displayed over the hierarchy trees.
- e. Adjust the order of the hierarchies as necessary by dragging them in the order pane. This will be the order that the position selection wizards are presented during the workbook build process.
- f. Click **OK** to save any changes and close the window.

Note: If any of the offered hierarchies are not enabled in this Standard Wizard window, the end-user will not be presented with a position selection wizard to define the scope of the workbook being built for that hierarchy. All positions in the lowest dimension offered in that hierarchy that the end user has access rights to will be automatically selected.

Custom Menu Tab

Use of this tab is optional. It is used to create a workbook specific customized menu-driven process within the workbook where the menu options defined execute rule group transitions (which therefore cause a series of calculations to be performed) and external scripts. Custom menus are typically used to define processes such as an approval process.

Create a Custom Menu

1. Click the **Custom Menus** tab.
2. In the Menu Label field, enter the name of the menu that will be displayed in the RPAS client. This menu option will appear as a top level menu option, between the Window and Help menu options.
3. Right-click in the table area, and select **Add**.
4. Enter the following information:
 - **Label** – The label that will be displayed in the menu in the RPAS Client. These labels will appear beneath the top-level menu option named in the Menu Label property, in the order that they are displayed in this window.

Note: Duplicate menu names are not allowed.

- **Function** – Defaults to RuleGroupProcessor and cannot be changed.
- **Arguments** – The processes that are to be executed by the menu option are specified in the arguments property. There may be several processes specified in the order they are to be executed, and separated by commas. If a process starts with an '*', the string that follows the '*' is assumed to be the name of an external script. Otherwise, the string is assumed to be the name of a rule group.

When you select the menu option in the RPAS client, RPAS executes the processes from the arguments property in the specified sequence. RPAS waits until each process has finished before executing the following process. After all of the processes have been executed, RPAS executes a final transition using the 'full' transition type back to the calc rule group for the workbook. This transition does not have to be explicitly specified in the arguments property.

Rule groups are executed with a "full transition" from the previous rule group, and the calculations apply to the whole scope of the workbook (that is, they use "full" (batch) mode rather than "incremental" mode). These terms are explained in

“Appendix B – Calculation Engine User Guide.” The rule group transitions ensure that the integrity of all rules is enforced in the new rule group.

Scripts referenced in the argument’s property should meet the naming conventions for the RPAS server operating system. They must reside in the root directory of the domain and have executable permissions. For the script to execute, the current working directory (./) has to be in the path before the DomainDeamon is started. RPAS passes the name of the current workbook (as RPAS would recognize the workbook to be) to the called script. This variable could be accessed as \$1 if the executable is a shell script, or arg[1] if the executable is a binary. This argument is the internal ID that RPAS recognizes the workbook with, so any RPAS calls that are made in the script (for example, exportData) will readily identify the workbook (if the data needs to be exported from the current workbook).

If the script that you want to call requires different arguments, you should instead call a “wrapper” script, which can call the target script with the appropriate arguments. RPAS waits until the called script has finished before executing the next process in the menu option. If you do not need to wait for the called script to finish, you should call a “wrapper” script that can call the target script, and then return immediately.

Note: Rule groups listed in the arguments for the menu option are not limited to those in the rule set used in the workbook, but they may be any rule groups used in the project.

Note: If any of the processes in a menu option should fail unexpectedly, execution of the menu option stops and displays an error message to the user. This may leave the data in the workbook in an inconsistent state.

Hier Mods Tab

Use of this tab is optional. The Hier Mods (Hierarchy Modifications) function is used to define the workbook hierarchy structure if that is required to differ from the hierarchy structure defined using the hierarchy tool. Hier Mods can effectively be used to “hide” individual dimensions or whole branches of a hierarchy by excluding them from the dimensions that are made available to the workbook. In some limited circumstances, it can also be used to create mappings between dimensions that are not directly related in the hierarchy structure specified in the hierarchy tool, but where you know that the relationship can be accurately deduced.

1. Click the **Hier Mods** tab.
2. All hierarchies that are used in the base intersection of measures that are used in the rule set that are used in the workbook will be shown.
3. For each hierarchy that you want to override, specify the dimension relationships that are required.

Note: Any hierarchies that do not have dimension relationships specified will use the full hierarchy specification from the hierarchy tool in the workbook, using the lowest dimension in that hierarchy used in the base intersection of a measure used in the rule set for the workbook as its root.

4. Dimension relationships are specified as hyphenated child-parent pairs with the child first and the pairs separated by commas. Every parent-child relationship that is

wanted in the workbook must be explicitly specified. Only dimensions that have been defined in the hierarchy tool can be used, and the dimensions must be specified by name. The dimension relationships are validated to ensure that valid dimension names are used, and the use of the same dimension name in both the child and the parent is prohibited. Furthermore, validation prohibits the use of a pair where the child dimension's Aggs attribute is the same as the parent dimension. For example, mnth-week is illegal but week-mnth is legal.

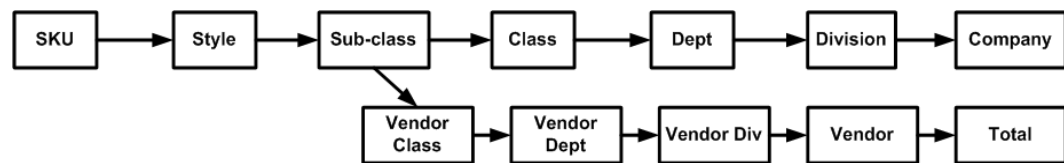
Example 1 – Hiding Dimensions

Consider a product hierarchy where a subclass is only supplied by a single vendor. An example of a product hierarchy that includes a branch for vendor analysis is:

Sku-Style-Subclass-Class-Department-Division-Company

with a branch of:

Subclass-VendorClass-VendorDept-VendorDiv-Vendor-Total



If a workbook was wanted that included measures with a lowest base intersection in the product hierarchy of subclass that did not require the vendor branch or the division dimension, the specification of Hier Mods for the product hierarchy would be:

scls-clss, clss-dept, dept-comp

The resultant workbooks would only contain the subclass, class, department, and company dimensions.

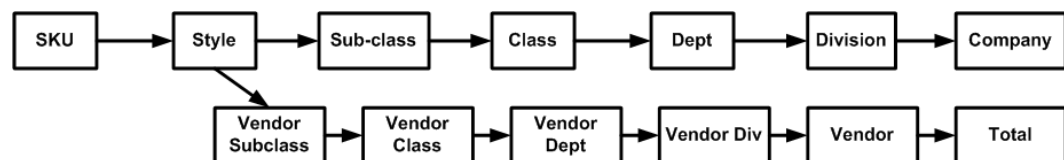
Example 2 – Defining a “Non-Structural” Dimension Relationship

Consider a product hierarchy where a subclass may be supplied by multiple vendors (with a style always supplied by a single vendor), but in some parts of the business, subclasses are only supplied by a single vendor. An example of a product hierarchy that includes a branch for vendor analysis is:

Sku-Style-Subclass-Class-Department-Division-Company

with a branch of:

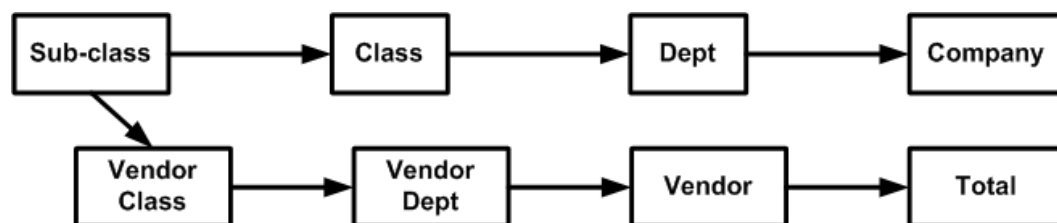
Style-VendorSubclass-VendorClass-VendorDept-VendorDiv-Vendor-Total



If a workbook was wanted that included measures with a lowest base intersection in the product hierarchy of subclass, that did require the vendor branch and did not want the division dimension, the specification of Hier Mods for the product hierarchy would be:

scls-clss, clss-dept, dept-comp, scls-vcls, vcls-vdep, vdep-vend, vend-tot

The resultant workbooks would contain the subclass, class, department, and company dimensions; together with a branch that contains the VendorClass, VendorDept, Vendor, and Total dimensions.



Note: The use of mappings that are non structural should be carefully managed to ensure they are only used where the ‘non structural’ mapping happens to work. In our example, if this workbook is used by an end-user in a part of the business where a subclass happens to only include Styles from a single vendor, the hierarchy built in the workbook will work correctly. However, if it is used in a part of the business where a subclass may include styles with multiple vendors, RPAS will determine (by looking at the VendorClasses that the SKUs in the subclass belong to) that the scls-vcls relationship is ambiguous, because the subclass should belong to multiple VendorClasses. In these circumstances, RPAS will build the hierarchy using one of the valid scls-vcls relationships. As far as the end-user is concerned, the choice of VendorClass for the subclass is likely to be seen as arbitrary, and (in any case) the vendor branch will be of little or no practical value in this case.

Dynamic Hierarchies Tab

Use of this tab is optional. It is used to configure a hierarchical relationship whose parent-child relationships are not defined through the normal loadHier process, but are data driven. The dynamic hierarchical relationships are built using measure data during the workbook build process, and may vary each time a workbook is built; but the relationships within a workbook are constant. For example, the “Cluster” dimension may be an alternate parent of the “Store” dimension in the Location Hierarchy, and the Cluster that a Store belongs to may vary by the “Class” dimension in the Product Hierarchy. In one workbook, a clustering process may determine the Store-Cluster relationships for each Class, and store that information in a measure. A second workbook could then use that relationship to build a dynamic hierarchy. In this example, if the rollup of Store to Cluster is different for each Class, and you bring more than one Class into the workbook, the rollup of Store to Cluster used in the workbook will be based on the data from the first class in the hierarchy. There can only be one dynamic hierarchy defined in a workbook.

The Dynamic Hierarchy process cannot “invent” a new dimension – it can only change the parent-child relationships of the existing dimensions. So in our example, the Cluster dimension must be a normal dimension defined through the hierarchy tool, and maintained through the LoadHier or user defined dimension processes. The dimension is normal, so it may be used in the base intersection of measures.

Note: If the branch of a hierarchy that has parent-child relationships defined by the dynamic hierarchy process only has a business meaning when the dynamic hierarchy process is used, you should use the Hier Mods process to “hide” the dimensions in other workbook templates. For example, in our above Cluster example, if the Store-Cluster relationship only exists in the context of a class, use Hier Mods to hide that relationship in a workbook template that does not include the product hierarchy.

Note: It is your responsibility to ensure that the position names contained in the measure that drives the dynamic hierarchy are real positions that exist in the parent dimension. If not, positions with those names will be present in the workbook, but data for them cannot be committed to the domain, and it will be lost when the workbook is deleted.

Note: The resulting Dynamic hierarchy is created at the end of the wizard selection process and before the actual workbook build. Therefore, the end product is only visible inside the workbook and not in the wizards.

Define a Dynamic Hierarchy

1. Click the **Dynamic Hierarchies** tab.
2. Right-click in the table area and select **Add**.
A row appears with properties to define the dynamic hierarchy.
3. Set the properties as follows:
 - **Name** – The name of the dynamic hierarchy. Duplicate names are not allowed. This is an internal name used as a “handle” to the dynamic hierarchy, and it is not visible to the end-user.
 - **Measure** – This is the name of the measure that holds the name of the parent position. Click Select Measure to get a list of the measures that are used in the solution. Type the first few characters of the measure name in the box at the top of the list to go to the required measure, or scroll to find it. Double-click to select the desired measure. The Measure should have a base intersection of the dimension(s) that the parent-child relationship is dependant on (Class in our example), and the dimension that is the child in the parent-child relationship (Store in our example). The content of the Measure is the name of the parent position in the relationship (in our example, this is the name of the Cluster that the Store belongs to for the Class). This Measure may or may not be included in the workbook. If the Measure is included in the workbook, changes to the Measure within the workbook do not change the parent-child relationships within the workbook, which are static once the workbook is built.
 - **Label Measure** – This is the name of the measure that holds the label of the parent position. The process to select the Label Measure is the same as the process to select the Measure. The base intersection of the Label Measure should be the same as the Measure. The contents of the Label Measure will be the label of the parent position in the relationship.

Note: If a given parent position name has different labels specified for different child positions, the label for the parent position used in the built workbook will be one of the different labels, which are arbitrarily selected.

- **Measure Hier** – This is the name of the hierarchy that the parent-child relationship is dependant on. In our example this is Product.
- **Measure Dim** – This is the name of the dimension that the parent-child relationship is dependant on. In our example this is Class.
- **Hier** – This is the name of the hierarchy that the parent-child relationship belongs to. In our example this is Location.
- **Dim** – This is the name of the dimension that is the child in the parent-child relationship. In our example this is Store.
- **Modified Dim** – This is the name of the dimension that is the parent in the parent-child relationship. In our example this is Cluster.

Note: There can be only one dynamic hierarchy per workbook.

Measures Tab

The use of this tab is optional. It allows you to override certain properties of measures at the workbook level. Use this tab when you configure multiple workbooks for a solution and the processes implied by those workbooks require different measure behavior. For example, a measure may need to be writeable in one workbook in a solution, but read only in all other workbooks. By using this tab, you can override the following standard measure properties at the workbook level: Label, Description, Base State, Agg State, UI Type, Single Hier Select, and Range. In addition, there are two properties, LoadRange and LoadRangeMeas that are not standard measure properties that may only be set through the measures tab.

Specify that a Measure May Have Properties Overridden for this Workbook

1. Click the **Measures** tab.
2. Right-click in the measure table area, and select **Add**.
 - a. Select the newly added row.
 - b. Click **Select Measure** to get a list of the measures used in the workbook.
 - c. Type the first few characters of the measure name in the box at the top of the list to go to the required measure, or scroll to find it. Double-click to select the desired measure. Measures that already have an entry in the measures tab are listed but disabled, and cannot be selected.

Change Measure Properties for a Measure That May Have Properties Overridden for this Workbook

1. Click the **Measures** tab.
2. Specify the desired property information for the measure.

Remove a Measure from Having Properties Overridden for this Workbook

1. Click the **Measures** tab.
2. Select the row for the measure.
3. Right-click in the measure table area and select **Remove**.

Apply Values to the LoadRange and LoadRangeMeas Properties

The LoadRange and LoadRangeMeas properties can only be set in the measure tab. They are used to specify “dynamic picklists.” Dynamic picklists are picklists whose valid values do not vary within a workbook, but can be set dynamically during the workbook build process.

The LoadRange and LoadRangeMeas properties are retained for upwards compatibility purposes. In most cases “context sensitive picklists” (that is, picklists using the “measurerange = measS” syntax in the range property) and “single select wizards” will be used instead.

Dynamic Picklists are of two forms:

1. The first form has values that are set according to the data values in cells for another measure (using the same format as for the Range property of static picklists). As with static picklists, the value shown to the user in the UI is the label for the value. It is more usual to use “context sensitive picklists,” where picklist values can vary according to context in the workbook.
2. The second form is to have values that are positions in a branch of a hierarchy. The value shown to the user in the UI is the label of the position, but the content of the cell is the name of the position. The single select wizard provides an alternative method for selecting a position, which is more commonly used.

Specify that a Measure Uses a Dynamic Picklist in this Workbook

1. Select the row for the measure.
2. Right-click in the row for the measure and select **Set Dynamic Picklist**.
3. If the picklist measure is to show hierarchy positions, an entry is required in the LoadRange property (and not in the LoadRangeMeas property). Click in the **LoadRange** field. The LoadRange dialog box opens:

Range	Hierarchy	Dimension	Sort By Label
AA	CLND	Day	☒
			☐
			☐
			☐
			☐

Load Range Dialog Box

- a. In the Range column, enter a name for the range. This name is used internally by RPAS and needs to be unique across the domain.
- b. In the Hierarchy column, select the hierarchy from which the user is to select a position.
- c. In the Dimension column, select the dimension from which the user is to select a position. The positions along this dimension, which are brought into the workbook, will be the available choices in the picklist in RPAS client.

- d. Select the **Sort by Label** check box if you want the positions in the picklist to be sorted alphabetically by their label. If this is not selected, the positions will be shown in their internal order, which will approximate to the order in which they were defined.
 - e. Click **OK** to save any changes and close the dialog box.
- 4. If the picklist measure is to show values based on the data values for a cell, an entry is required in the LoadRangeMeas property and in the LoadRange property. Click in the LoadRangeMeas field, and click the **Select Measure** button. Type the first few characters of the measure name in the box at the top of the list to go to the required measure, or scroll to find it. Double-click the measure whose contents are the valid picklist values. In addition, there should be an entry in the LoadRange field for each hierarchy in the base intersection of the selected LoadRangeMeasure. Each of the entries in the LoadRange field needs a name, hierarchy, and dimension as described above; but the value for sort label is ignored. If the scope of the workbook is such that it covers multiple cells of the LoadRangeMeasure, the available picklist options in the workbook will be constructed from the content of the first cell of the LoadRangeMeasure when the dimensions are ranged to the positions selected during the workbook build process.

Extended Measures Tab

Use of this tab is optional. It allows you to configure “extended measures” that represent different usages of the underlying base measure. Extended measures support using different aggregation methods for the same base measure and participation measures, such as absolute and relative percent-to-parent measures. Once defined, these extended measures can be added onto worksheet profiles to be viewed in the RPAS client.

Note: Hybrid is not supported for extended measures.

Overview of Participation Measures

A “percent-to-parent measure” or “participation measure” is a measure that contains the value of the current positions as a proportion of the value at a Parent level (for example, sales as a percent of the class sales). These measures can be viewed and edited, and they may be preconfigured through the RPAS Configuration Tools or dynamically defined in the RPAS client in a worksheet.

Typical uses of this functionality are to define measures that are percentage participations of sales measures. Typically, these are either to a fixed level (such as class) so the participation of each item to the class can be viewed and manipulated, or they are to the ‘next level up’ in the product hierarchy.

The following examples will use the sample product hierarchy structure: SKU-style-subclass-class-company-all.

Note the following important points when using this feature:

- Changing the percentage of the percent-to-parent measure will cause the values of the underlying measure to change to reflect the newly set percentage.
- Multiple percent-to-parent measures can be defined for the same underlying measure; however, only one percent to parent measure or the underlying measure can be edited before calculation occurs. All other versions will be protected.
- The value of a percent-to-parent measure is a fraction between zero and one. You must format the measure to be displayed as a percentage if desired.

Configure Extended Measures

To specify an extended measure for a workbook:

1. Click the **Extended Measures** tab.
2. Right-click in the table area, and select **Add**.
 - a. Select the newly added row.
 - b. Click **Select Measure** to get a list of the measures used in the workbook. Type the first few characters of the measure name in the box at the top of the list to go to the required measure, or scroll to find it.
 - c. Double-click to select the required measure.
3. Specify the appropriate property information for the extended measure.

To change properties for an extended measure for a workbook:

1. Click the **Extended Measures** tab.
2. Specify the desired property information for the extended measure.

To remove an extended measure from a workbook:

1. Click the **Extended Measures** tab.
2. Select the row for the extended measure.
3. Right-click the row for the extended measure, and select **Remove**.

The Usage and Args Properties

The Usage property describes the type of extended measure that is being defined. The choices offered will include all the "Allowed Aggs" for the measure, plus "Relative % To Parent" and "Absolute % to Parent."

1. To define the extended measure to use an alternative aggregation type from the list of Allowed Aggs, select the aggregation type in the Usage property.
2. To define the extended measure to be a "Relative % to Parent," select that Option in the Usage property. Select the hierarchy that the parent should be in from the Args property. The extended measure will contain the value of the measure as a proportion of the value of the measure at the next higher dimension in the chosen hierarchy that is visible in the window.

The "relative" percent-to-parent measure type calculates the value for a given level that is the percentage of that level and its immediate parent (meaning one level higher). This type can only be set for a single hierarchy.

Using the previously defined sample hierarchy structure, the percentage displayed at the "SKU" level is the "SKU" as a percent of the "style"; at the "style" level, the "style" as a percent of the "subclass"; and at the "subclass" level, the "subclass" as a percent of the "class."

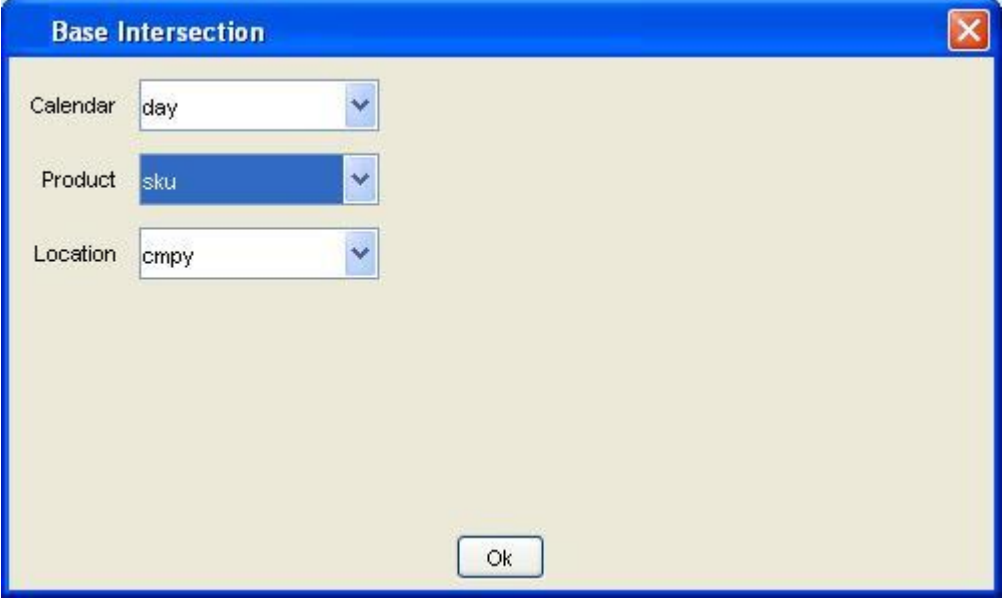
Note that only a single hierarchy is possible with the "relative" percent to parent measure type.

In the case of multiple branching hierarchies, where certain dimensions will have multiple parents, for relative percent-to-parent measures, the percentage will always be calculated based on the active roll-up in the current window.

Note that the calculated value is based on the actual next level up in the hierarchy (based on the hierarchy structure), not necessarily the one that is being displayed. In the previous examples, imagine that SKU-subclass-class is displayed in the client, but the underlying structure is SKU-style-subclass-class. When viewing the value at SKU, the percentage will be based on style (not subclass) even though style is not displayed.

Cells at the top of the hierarchy will be hashed out because those values cannot be calculated.

3. To define the extended measure to be an "Absolute % to Parent," select that Option in the Usage property. Click the Args property to open the base intersection dialog box to specify the intersection to be used in the % participation. A dimension should be selected from each hierarchy in the base intersection of the measure. If a dimension is not selected along one of the hierarchies, the top of that hierarchy is effectively selected.



The image shows a dialog box titled "Base Intersection" with a blue header bar and a close button (X) in the top right corner. The dialog has a light beige background. It contains three labels on the left: "Calendar", "Product", and "Location". Each label is followed by a text input field and a small blue downward-pointing arrow button. The "Calendar" field contains the text "day", the "Product" field contains "sku", and the "Location" field contains "cmpy". At the bottom center of the dialog is an "Ok" button.

The "absolute" type of percent-to-parent measures allows you to explicitly define the parent level(s) that are used to calculate the percentage at all child levels.

Using the previously defined sample hierarchy structure, if you set the absolute parent level to the "class" dimension in the product hierarchy, the percent-to-parent measure will show the "SKU" as a percent of the "class" at the "SKU" level, the "style" as a percent of the "class" at the "style" level, and the "subclass" as a percent of the "class" at the "subclass" level.

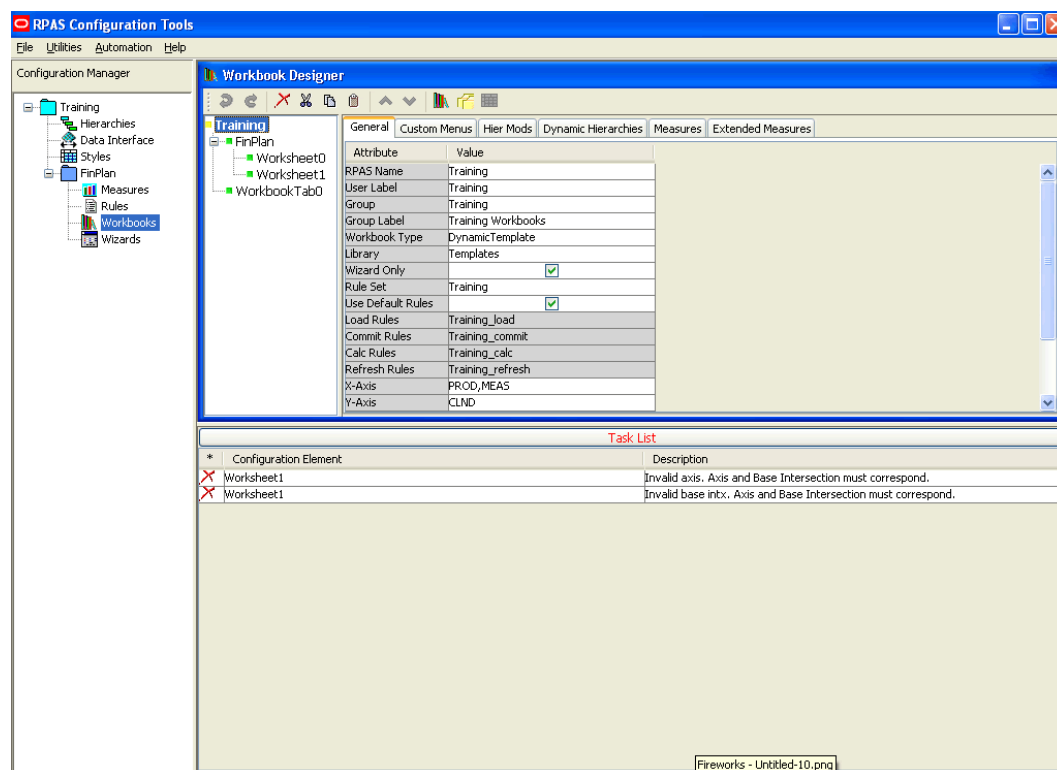
For the "absolute" relationships, cells at or above the explicitly set dimension/level will be hashed out.

Note: You may define multiple different extended measures based on the same measure, but it is an error to define identical extended measures based on the same measure.

Note: Percent-to-parent measures should be defined only on measures that have "total" as their default aggregate methods.

Edit Workbook Properties

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Workbooks**. The Workbook Designer window opens in the workspace.



Workbook Designer Window

1. Select the workbook, and click on the tab you want to edit.
2. Update the information as appropriate.
3. To remove information from any of the tables:
 - a. Select the row.
 - b. Right-click and select **Remove**.

Remove a Workbook

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Workbooks**. The Workbook Designer window opens in the workspace.

1. Select the workbook that you want to remove.
2. From the toolbar, click the **Delete**  button, or select **Remove** from the right-click menu. You are prompted to confirm your decision.
3. Click **Yes**. Associated worksheets and tabs are removed.

Working with the Rule Group Simulator

Overview

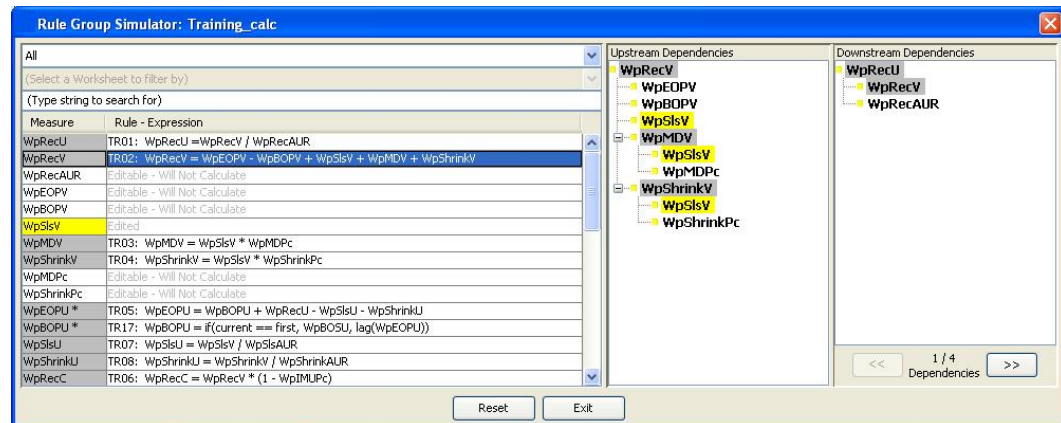
The RPAS calculation engine is powerful and complex. The rule group approach means that there are very many potential calculation paths. However, during any configuration exercise, there is a significant design verification cost to ensure that the behavior is ‘as would be expected’ by an end-user. The rule group simulator enables the verification of the interaction between measures from within the configuration tools. It cannot; however, enable the verification of the calculations themselves because that requires a full domain to be built.

The Rule Group Simulator is integrated into the workbook tool, and it uses all of the measures used in the rule set in the workbook, which may be more than those mentioned in the rule group being simulated. Users of the rule group simulator are expected to understand the calculation cycle, especially with respect to measure protection processing and the process that determines which expressions will be evaluated. See “Appendix B - Calculation Engine Users Guide” for more information.

Note: The rule group simulator is not able to simulate the expressions that will be evaluated as the result of a rule group transition, nor simulate the calculations that will follow if a rule group is evaluated in ‘full’ mode, such as when evaluated from the mace utility, or the evaluation of the load rule group when a workbook is built.

About the Rule Group Simulator

The Rule Group Simulator feature is provided in a separate window with two areas: a measure table, and a “tree window” with upstream and downstream panes.



Rule Group Simulator Window

The measure table shows all of the measures in the scope of the simulation. The first column of the table gives the measure name and (by color coding and a 'tool tip') the status of the measure. All measures can be shown, or the list of measures can be filtered. Measures that are currently editable can have their status toggled (to or from "edited"), and the simulator immediately updates all statuses, calculations, and trees. The color coding is as follows:

Color	Meaning
Yellow	Edited. If it is a recalc measure, it will be calculated by indirect spreading of another measure through a mapping rule and recalculation at aggregated levels. If the measure has another aggregation type, it will be calculated by spreading and aggregation.
Pale Gray	Editable. Although the measure is not forced, and thus is still editable, it will be calculated through the calculation engine having to select an expression in an affected rule.
White	Editable. Will not be calculated, so it will not change at all.
Pale Blue	Protected by protection processing. Although the measure is protected (usually this will be because it is the measure on the left-hand side of the only expression in a rule), it is not 'forced' because none of the right-hand side measures are changed, so it does not need to be calculated, and it will not change at all.
Mid-Blue	Protected by protection processing. Is "forced," so it will be calculated.
Dark Blue	Read Only. The measure is set as being read only in the measure properties, so it will not change at all. A measure that is read only, but is going to be calculated will be shown as mid-blue. That status takes priority over read only.

Note: The status of a measure encapsulates two concepts that are not as closely linked as may appear at first sight:

Whether-or-not the measure is editable (shades of blue = no, white/gray/yellow = yes)

Whether-or-not the measure will be calculated.

It is perfectly possible for a measure to currently be editable, but it would be calculated if a calculate command were issued. Similarly, it is perfectly possible for a measure to be protected by protection processing that would not be calculated if a calculate command were issued.

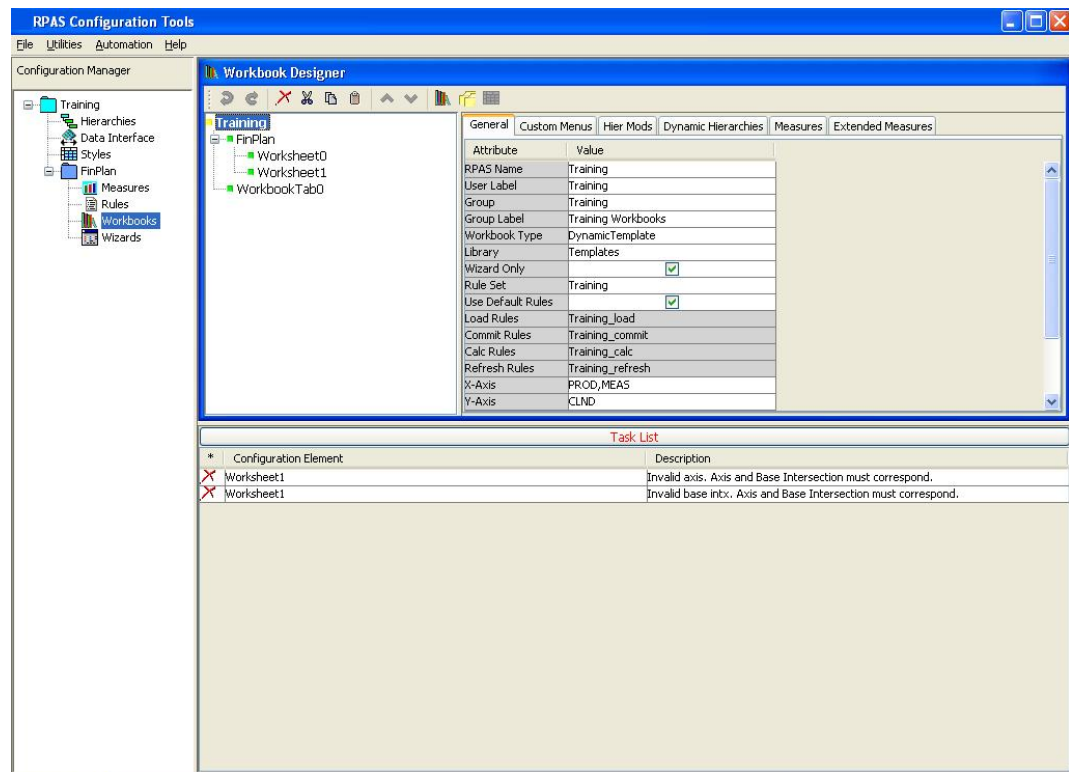
The second column of the table shows the "calculation" for each measure. For those measures that would be calculated if you issued a 'calculate' with the current collection of edited measures, the rule and expression that would be used to calculate the measure is shown. For measures that would not be calculated, this column displays the status of the measure.

The tree window shows (in separate panes) the upstream and downstream measure relationships (that is, the expressions that will be evaluated) for the measure with focus.

Measures in the panes are also color coded. If the measure with focus would be calculated, the upstream pane shows the expression to calculate it, and, recursively, all measures that it is dependent upon (calculated from) with their expressions, if appropriate. The downstream pane similarly shows measures that are dependent upon (calculated from) the measure with focus, if there are any. If the measure with focus is on the right-hand side of several expressions that will be calculated, you can view each of the expressions by scrolling through a “VCR” control.

Invoking the Rule Group Simulator

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Workbooks**. The Workbook Designer window opens in the workspace.



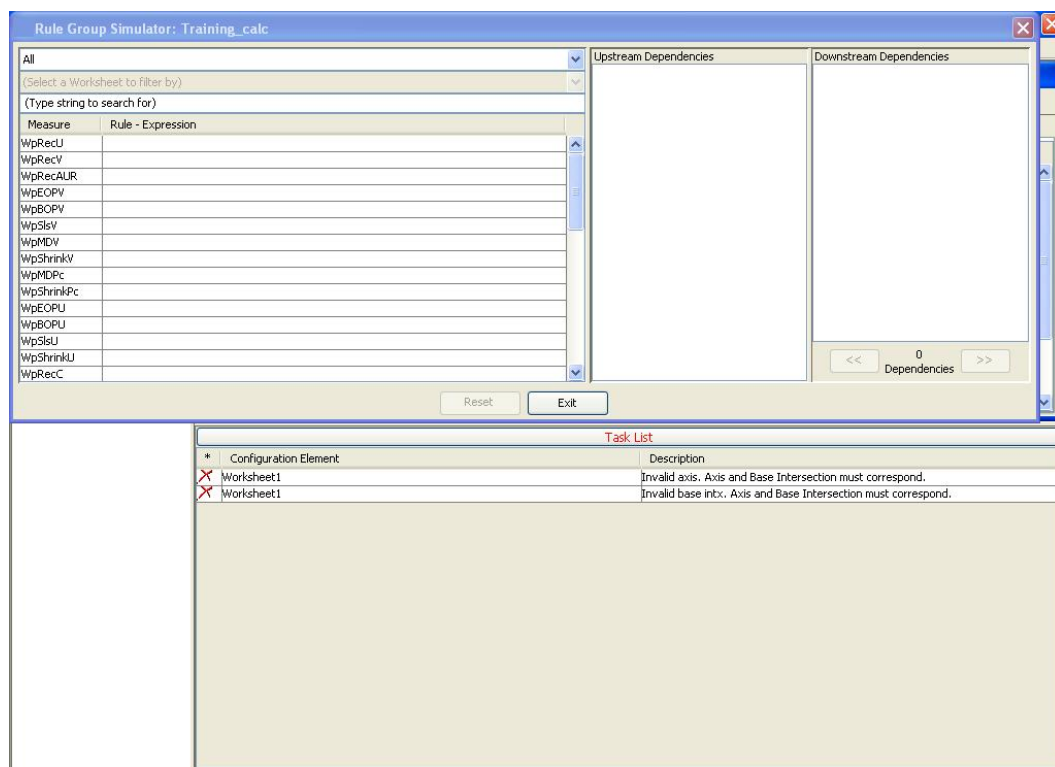
Workbook Designer Window

1. Select one of the workbooks in the Workbook Designer tree display.
2. Select the **General** tab of the Workbook properties table.
3. Right-click on **Calc Rules** and select **Simulate Rule Group**.



Simulate Rule Group Menu Option

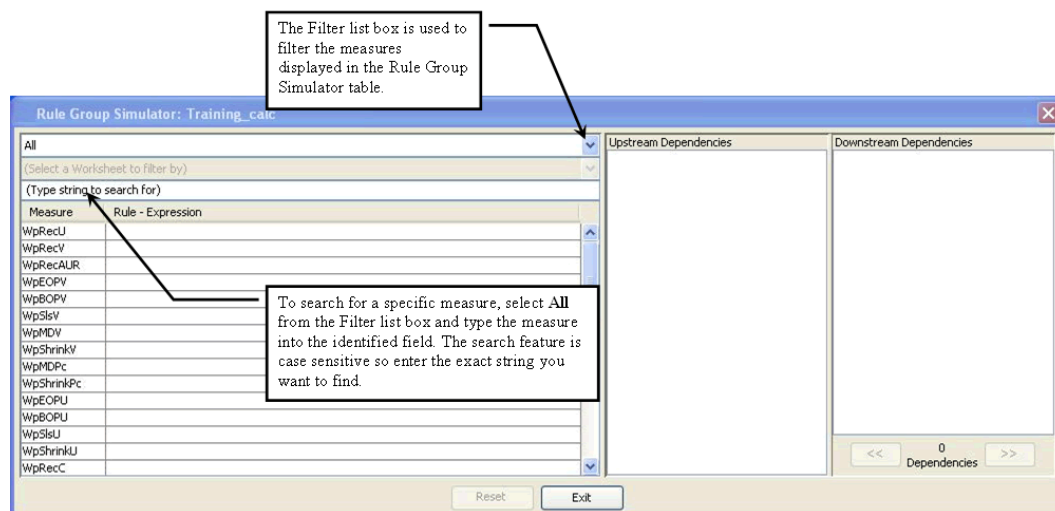
The Rule Group Simulator appears.



Rule Group Simulator Window

Filtering the Measures Table

1. Open the Rule Group Simulator.



Rule Group Simulator Showing Filter and Search Features

2. To filter the table, use the drop down box above the table to select the filter to use.
 - a. If filtering by **All**, all measures in the workbook will be shown.
 - b. If filtering by **Will Calculate (According to Calculation Order)**, only those measures that will be calculated will be shown. The sequence of the measures will be the sequence in which they will be calculated. Measures that are edited are not shown.

- c. If filtering by **Will Not Calculate**, only those measures that will not be calculated will be shown. Measures that are edited are not shown.
 - d. If filtering by **Read Only**, only those measures whose status is read only (that is, have a Base State and Agg State of 'read') will be shown.
 - e. If filtering by **Contains String**, type a case sensitive string to filter by in the text box below the filter drop down box. Only measures that include the string will be shown.
 - f. If filtering **By Worksheet**, select a worksheet from the current workbook using the drop down box below the filter drop down box. Only the default measures from that worksheet are shown.
3. If searching for a specific measure, set the filter to **All**, and type a search string (case sensitive) in the box below the filter list. The first measure that includes the string will be shown and will become the measure with focus.

Changing the Edited Status of Measures

Note: Only measures that are editable (colors gray, white, and yellow) may have their status changed.

1. Click on the name of the measure in the measure table.
 - a. If the status was previously editable (gray or white), the status of the measure changes to edited (yellow).
 - b. If the status was previously edited (yellow), the status of the measure changes to editable (either gray or white, depending on the rule group and the other edits currently applied).
2. After any change in status, the simulator updates all necessary statuses, calculations, and trees.
3. To change the status of all edited measures back to editable (gray or white), click the **Reset** button.

Note: Measures that are calculated in a 'cycle,' which typically includes BOP and EOP inventory values, are indicated with an "*" next to their names in the measure table and upstream and downstream panes.

Using the Upstream and Downstream Panes

1. To change the measure with focus for the upstream and downstream panes:
 - a. Enter a search string when filtering in the **All** state, and the first measure that contains the search string will get focus.
 - b. Click in the **Calculation** column of the measure table for the measure.
 - c. Click on any occurrence of the measure in the Upstream or Downstream Dependency panes.

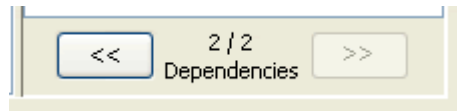
When the focus changes, the tree panes are refreshed, and the measure table scrolls so the measure with focus is shown.

The measure with focus is always shown at the top of the Upstream pane. If it will be calculated, the Upstream pane shows the measures that it is dependent on (calculated from, directly and indirectly). This is shown by a "tree" structure with the measures used to calculate an individual measure showing as "children" of it. If such children are also calculated, they will be shown with their dependent measures, and

so on, recursively. Therefore, the Upstream pane displays all of the measure relationships that affect the measure with focus.

The Downstream pane shows measures that are dependent upon (calculated from) the measure with focus, if there are any. Measure relationships (expressions) are shown in a parent-child “tree” structure. If the measure with focus is on the right-hand side of several expressions that will be calculated, the relationships cannot all be shown at the same time in a simple tree structure, so a single relationship is shown. The number of such relationships, and the one being shown, is indicated at the bottom of the pane.

2. To collapse the detail of the dependencies for a measure in the upstream or downstream panes, click the (-) next to the measure name. The (-) changes to a (+), and the detail is collapsed. To expand the detail of the dependencies for a measure in the upstream or downstream panes, click the (+) next to the measure name. The (+) changes to a (-) and the detail is expanded.
3. To change which measure relationship for the measure with focus is shown in the Downstream pane, click backward (<<) or forward (>>) buttons at the under the Downstream Dependencies box.



Back and Forward Controls

Note: The Rule-Expression column of the measures table will display multiple result expressions with a note beside the rule name saying that it is “multi-result.” Furthermore, the entire expression will be displayed showing all of the left-hand side measures that comprise the multiple results. If a measure that has focus is one of the multiple result measures, it will be shown in the upstream and downstream panes as MeasA [+MeasB][+MeasC] where MeasA is the measure in focus and MeasB and MeasC are the other multiple result measures.

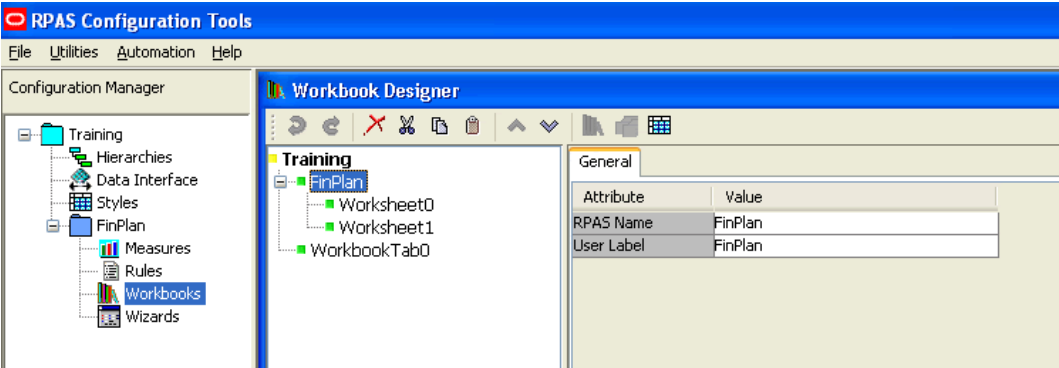
Exiting the Rule Group Simulator

To exit the rule group simulator, click the **Exit** button.

Working with Workbook Tabs

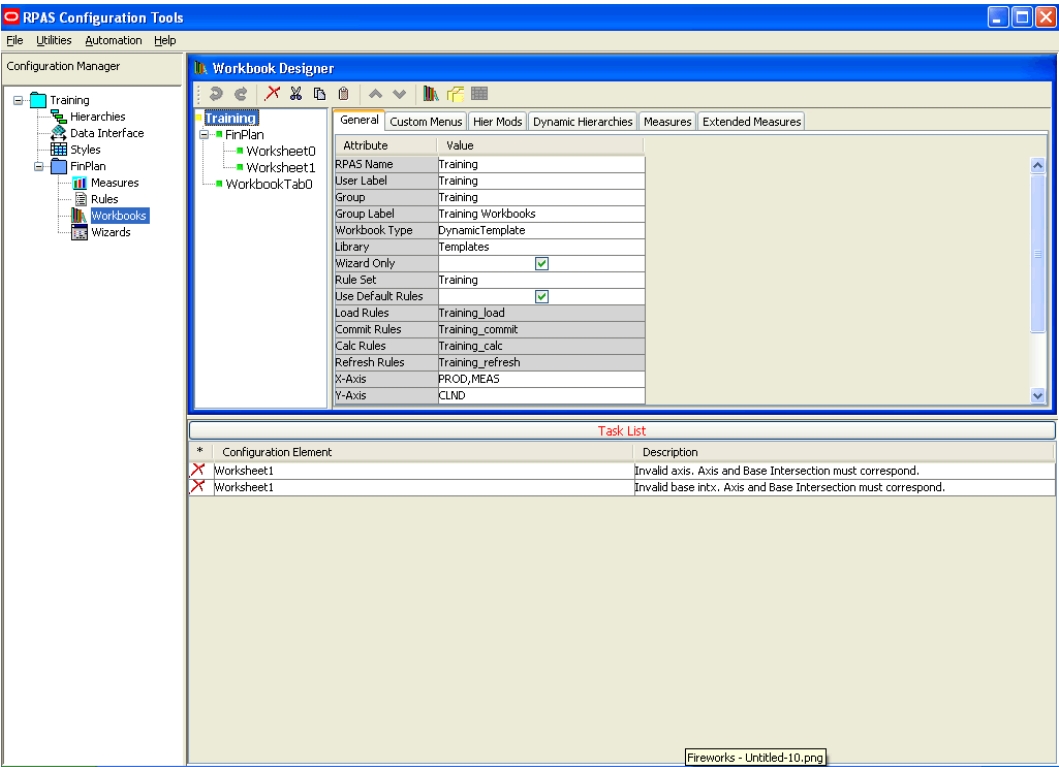
Overview

Tabs are a feature in the RPAS client that enables the workflow to be broken down into steps. Each workbook must have at least one tab, but may have several. You select which tab to use depending on the stage they have reached in the business process. A tab may contain one or more worksheets that enable you to interact with the data in the workbook. The measures available, the orientation of the hierarchies and the base intersection that data is available for may vary by worksheet within the tab.



Create a Workbook Tab

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Workbooks**. The Workbook Designer window opens in the workspace.



Workbook Design Window

1. Select the workbook in which to create a new workbook tab.
2. Perform one of the following:
 - Click the **New Workbook Tab**  button
 - Right-click and select **New Workbook Tab**.
 - Select an existing workbook tab in the workbook, and press **Insert**.A new workbook tab is created.
3. In the RPAS name field, enter RPAS internal name of the workbook tab.
4. In the User Label field, enter a description of the workbook tab that users will see on the tab in the RPAS client.

Edit Workbook Tab Properties

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Workbooks**. The Workbook Designer window opens in the workspace.

1. Select the workbook tab whose properties you want to edit.
2. In the General tab, type the RPAS Name and the User Label.
3. Perform one of the following:
 - To change the order of the tabs, select the up  button or down  button as necessary.
 - Drag and drop the tab in the workbook designer tree display.

Remove a Workbook Tab

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Workbooks**. The Workbook Designer window opens in the workspace.

1. Select the workbook tab that you want to remove.
 2. Perform one of the following:
 - From the toolbar, click the Delete  button.
 - Select **Remove** from the right-click menu.
 - Press **Delete**.You are prompted to confirm your decision.
 3. Click **Yes**.
- The workbook tab and associated worksheets are removed.

Working with Worksheets

Overview

Measures and Worksheets

A worksheet is a specific window into the data in a workbook. Worksheets are placed on workbook tabs. You define the measures on a worksheet, the base intersection the worksheet uses, and the orientation of the hierarchies on the worksheet. The measures that are in the workbook can be categorized in four different categories for a worksheet: default profile, viewable profile, hidden, and extended.

Default Profile Measures

The default profile contains the list of measures that will initially be displayed for this worksheet in the RPAS client. There must be at least one measure on the default profile.

Viewable Profile Measures

The viewable profile contains the full list of measures that the RPAS client user can view in the worksheet by using the Show/Hide functionality within the RPAS client. It must contain all of the measures in the default profile, but it often includes further measures that are not initially displayed.

Hidden Measures

Hidden measures are those that are used in the rule set assigned to a workbook, but that are not assigned to any of the profiles in any of the worksheets contained in that workbook. This might include measures that are used purely for calculation purposes and would have no usefulness to the RPAS client user.

Extended Measures

Extended measures, which represent different usages of the underlying base measures, can be added to the default or viewable worksheet profile. You can add extended measures that are aggregated based on different aggregation methods. The aggregation methods available for selection are based on the Allowed Aggs of the base measure. The same base measure can have multiple extended measures based on different aggregation methods.

You can also add extended measures that represent the relative and absolute percent-to-parent contributions. The same base measure can have multiple extended measures based on different selections for relative and absolute percent-to-parent contributions.

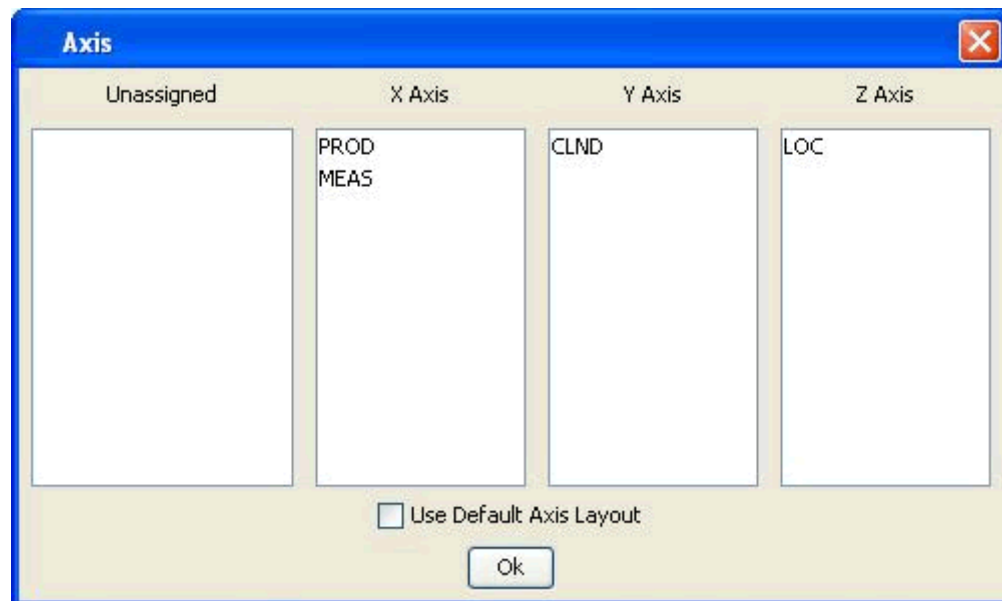
Create a Worksheet

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Workbooks**. The Workbook Designer window opens in the workspace.

1. Perform one of the following options:
 - Select the workbook tab in which to create a new worksheet by clicking the **New Worksheet**  button.
 - Right-clicking and selecting **New Worksheet**.
 - Select another worksheet on the same tab
2. Press **Insert**.
A new worksheet is created.

General Tab

1. In the RPAS Name field, enter the RPAS internal name of the worksheet.
2. In the User Label field, enter a description of the worksheet that users will see.
3. Define the axis layout of the worksheet.
 - a. Click the **X-axis**, **Y-axis**, **Z-axis**, or **Unassigned** field. The Axis dialog box opens.

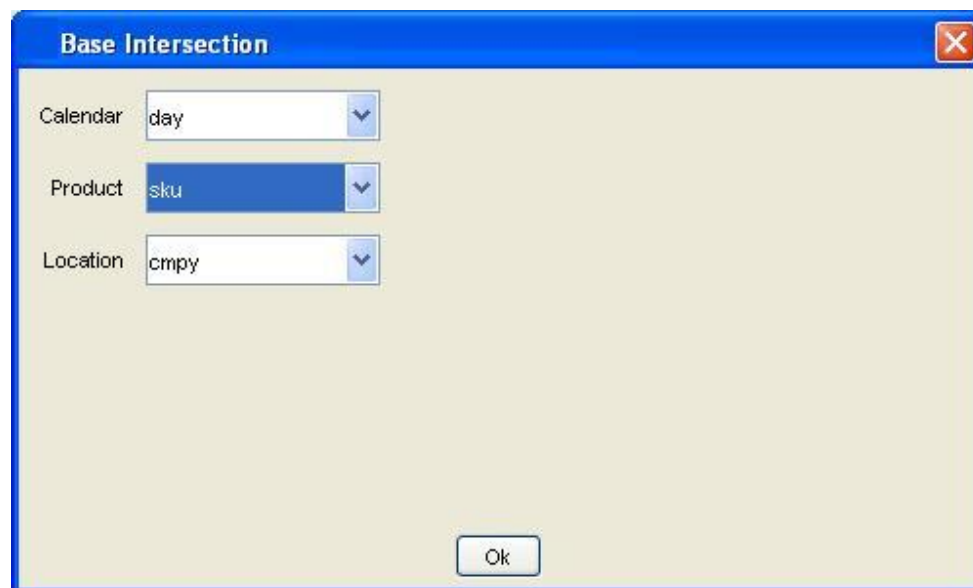


Axis Dialog Box

- b. Drag the hierarchies to the appropriate axis column.
- c. Click **OK** to save any changes and close the window.

Note: The Hierarchies that appear in this process are those used by measures placed on the default and viewable profiles for the worksheet. If no measures have yet been placed in those profiles, no hierarchies will appear in this process.

4. Click in the **Base Intersection** field. The Base Intersection dialog box opens.



Base Intersection Dialog Box

Note: The Hierarchies and dimensions that appear in this process are those used by measures placed on the default and viewable profiles for the worksheet. If no measures have yet been placed in those profiles, no hierarchies will appear in this process.

5. Select the dimension for each hierarchy. The base intersection of the sheet represents the base set of dimensions at which the window could be displayed. Data on the window can be viewed at any dimension/intersection above this. If the base intersection of a measure on the sheet is below the base intersection of the sheet, the measure's values are shown aggregated to the displayed intersection. If measure's base intersection is above the sheet intersection, the measures values are hashed out at all intersections lower than the base intersection of the measure.
6. Click **OK**.

Position Queries Tab

Usage of this tab is optional. It allows you to specify a worksheet where the positions that are shown in a "query" dimension are based on the current position in "driving" dimension(s). The driving dimension(s) must be in the slice area, and the query dimension must be in the X or Y axes. The process uses a position query that is a Boolean measure dimensioned on the query dimension and the driving dimension(s). Only positions in the query dimension that have the value TRUE for the position query measure for the position(s) in the driving dimension(s) are shown in the worksheet. All other positions are automatically hidden. When more than one driving dimensions are present, all of the driving dimensions have to be in Z-axis for the position query to execute. If one or more driving dimensions are taken out of the Z-axis and placed in X or Y axes, associated position queries will not be executed. A given window can have more than one position query, driven by one or more dimensions in the Z-axis and driving different dimensions in the X and Y axes.

1. Click the **Position Queries** tab.
2. Right-click in the table area, and select **Add**.
3. Enter the following information:

- **Name** – The name for the position query used by RPAS. This name has to be unique across the project.
- **Label** – The label for the position query that is used internally by RPAS. This label has to be unique across the project.

Note: Duplicate label names are not allowed.

- **Measure** – This defines the position query measure. Click in the field, and then click Select Measure to get a list of the Boolean measures used in the workbook. Type the first few characters of the measure name in the box at the top of the list to go to the required measure, or scroll to find it. Double-click to select the desired measure.
- **Dimension** – This defines the query dimension. Select a dimension from the list of dimensions for the measure you selected.

Style Overrides Tab

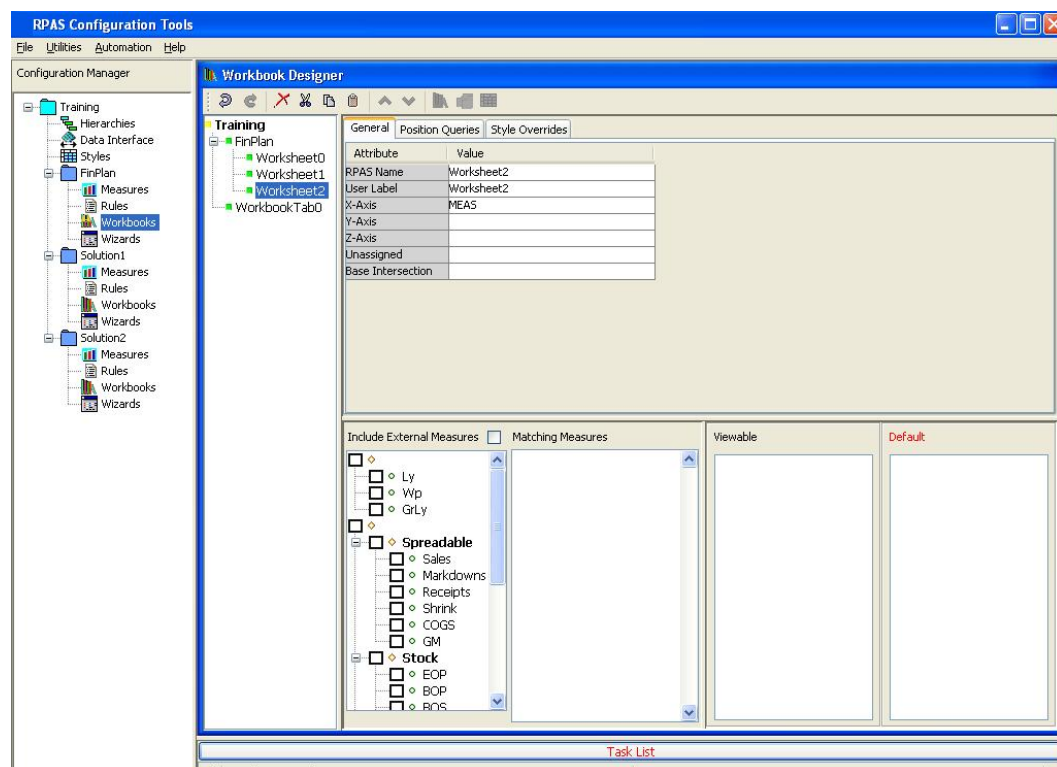
Usage of this tab is optional. It allows you to override the style property for a measure, so that the measure uses a different formatting style on this worksheet.

Note: The measures that appear in this process are those placed on the default and viewable profiles for the worksheet. If no measures have been placed in those profiles, no measures will appear in this process.

1. Click the **Style Overrides** tab.
2. The list of measures appears with their current formatting style.
3. Select an override formatting style for a measure. Measures whose styles have been overridden are shown in black. Those whose styles are defaulting are shown in gray.

Specify Which Measures Are on a Worksheet

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Workbooks**. The Workbook Designer window opens in the workspace.



Workbook Designer Window

1. Select the worksheet that you want to specify measures for.
2. In the column that displays measure components (under Include External Measures), select the check boxes next to the Measure components. The matching measures will appear in the Matching Measures column as the components are selected. Only realized measures that are used in the rule set that is assigned to the selected workbook will be displayed.
3. If you want External Measures to be available to be placed on the worksheet, select the **Include External Measures** checkbox.
4. To add measures from the matching measures column to the Viewable or Default columns, perform one of the following options:
 - Select the measures to add from the Matching Measures column. Drag the measures to the Viewable or Default column
 - Select the measures to add from the Matching Measures column and Press Ctrl-C, and then click in the Viewable or Default column and press Ctrl-V
 - Right-click in the Matching Measures column and select Copy, and then select Paste from the right-click menu in the Viewable or Default column
 - To add ALL measures from the Matching Measures column, right-click in the Viewable or Default column and select Add Matching from the right-click menu.
5. To add measures from the Viewable column to the Default column, perform one of the following options:
 - Select measures in the Viewable column, and drag the measures to the Default column.
 - Select measures in the Viewable column, and press Ctrl-C, and then click in the Default column and Press Ctrl-V.

- Right-click in the Viewable column, and select **Copy**. Right-click in the Default column, and select **Paste**.

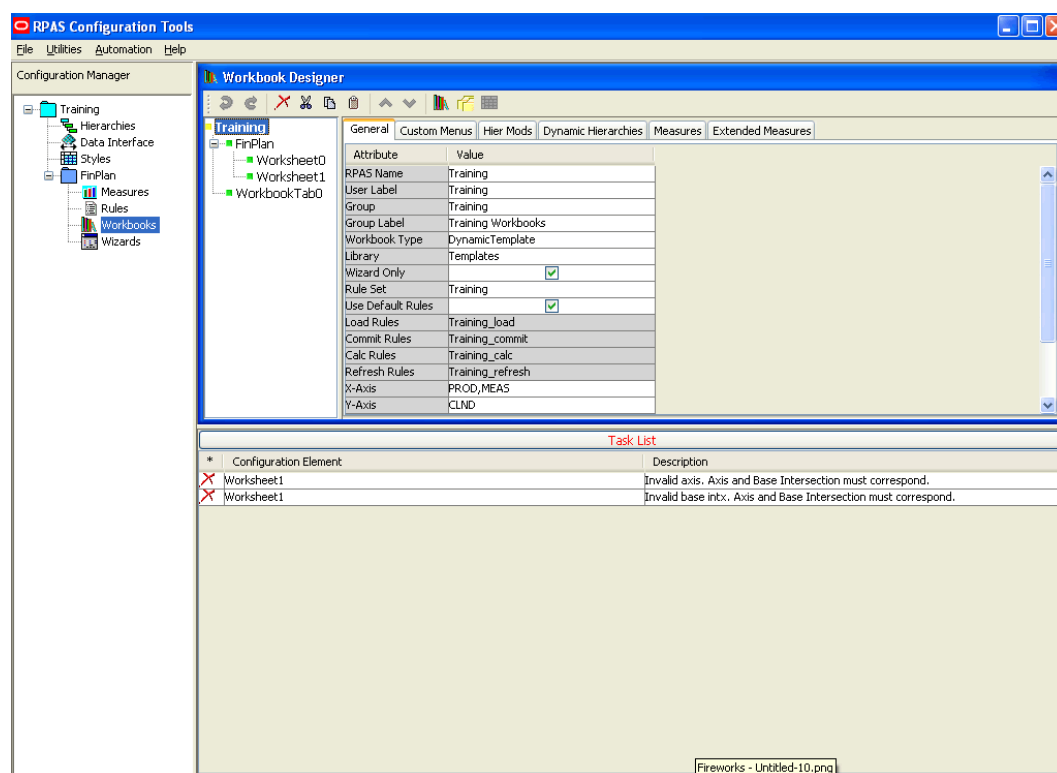
Note: If you add a measure to the Default column, it will also be added to the Viewable column if it is not already in the Viewable column.

6. To remove measures from the Viewable or Default columns, perform one of the following options:
 - Select the measures to remove and press **Delete** or **Ctrl-X**, or right-click and select **Cut**.
 - Right-click and select **Remove Matching** to remove all measures that are also in the Matching Measures column, or select **Remove All** to remove all measures.

Note: If you remove a measure from the Viewable column, it will also be removed from the Default column.

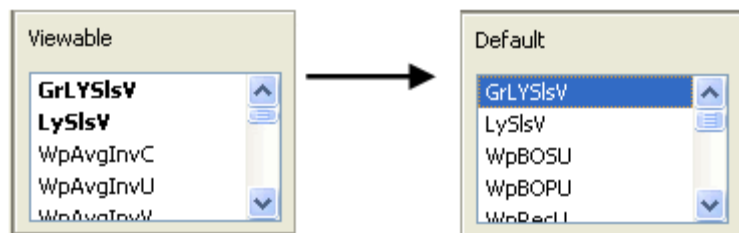
Specify the Sequence of Measures on a Worksheet

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Workbooks**. The Workbook Designer window opens in the workspace.



Workbook Designer Window

1. Select the worksheet that you want to sequence measures for.
2. To sequence measures manually, drag and drop the measures from the Viewable column to the Default column.



3. To sort the measures:
 - a. Right-click and select **Sort** in the Default column. The Sort Measures dialog appears.

Sort Measures Dialog Box

- b. Measures can be sorted based on their major components. Select the sort sequence in which you want the major components to be applied, and whether the sort should be ascending or descending
- c. Press **Apply**.
- d. The measures are sorted into the specified sequence. This is a one time sort, so that if you add further measures to the Default Measures column, or manually sequence measures, the sort sequence you specified will no longer apply.

To be able to see the Measures in this Order in RPAS Client:

1. Right-click in the measure axes of RPAS client and select Sort. The Sort dialog appears.
2. Select **User Specified Order**, and click **OK**.
The measures in the RPAS client for that window will be shown in the same order as that specified in the “Default” column of the window in the Configuration Tools.

Edit Worksheet Properties

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Workbooks**. The Workbook Designer window opens in the workspace.

1. Click on the worksheet that you want to edit.
2. Update the information as appropriate.
3. To remove information from the Position Queries table:
 - a. Select the field.
 - b. Right-click and select **Remove**.

Remove a Worksheet

1. Select the worksheet that you want to remove.
2. From the toolbar, click the **Delete**  button, or right-click and select **Remove**. You are prompted to confirm your decision.
3. Click **Yes**.

Wizards**Overview**

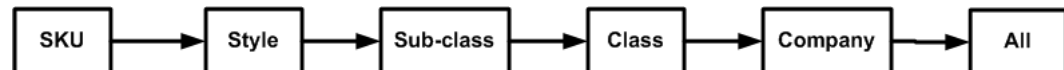
This section describes the tasks that you can perform by using the wizard designer. The Wizard Designer allows you to define the graphical layout for custom wizards.

When the workbook build process only involves selecting the scope of the workbook by selecting positions from the standard hierarchies, you can use standard wizards as described in the “General Tab” section of the “Workbook.” Most of the workbooks can be built using the standard wizards, and no coding is required – just configuration.

If the wizard process needs to do additional processing, you need to use custom wizards. The following two examples will help to clarify the need for custom wizards:

A simple example:

Consider the product hierarchy:



If the workbook builder needs to select just a few SKUs from a domain that contains many SKUs, that process could be tedious if the builder is presented with a wizard with huge numbers of SKUs. You may want to include a two-step process to select the SKUs:

- In the first step, the builder selects a class.
- In the second step, the builder selects SKUs from just the SKUs in that class.

A more complex example:

The workbook build process may need the builder to make choices from some predefined options. The choices that the builder makes could determine what further selections or choices the builder must make. The workbook that is eventually built could therefore be of several different ‘subtypes.’

Neither of the examples above are configurable as “Standard Wizards,” so custom wizards must be used. The custom wizards can be designed in the custom wizard designer, but must be accompanied with code that:

- Describes the sequence of the wizards
- Collects and processes the information from the wizards
- Generates the content of the next wizard
- Describes the content of the workbook that is generated

In the first example, the workbook designer lays out two 2-tree wizards (one for class and the other for SKU). Then to support the wizard process, the designer would write the code to read the selection from the class wizard and range down the available SKUs in the second wizard.

In the second example, the designer lays out the first wizard page as a group of check boxes (or list boxes if there are too many options). The designer would also lay out the other wizard pages that are required. The designer writes the code to collect the selections made in the first wizard, and to display (or not) the other wizard pages that are dependent on the selections made.

In summary, when using custom wizards, the workbook designer is responsible for:

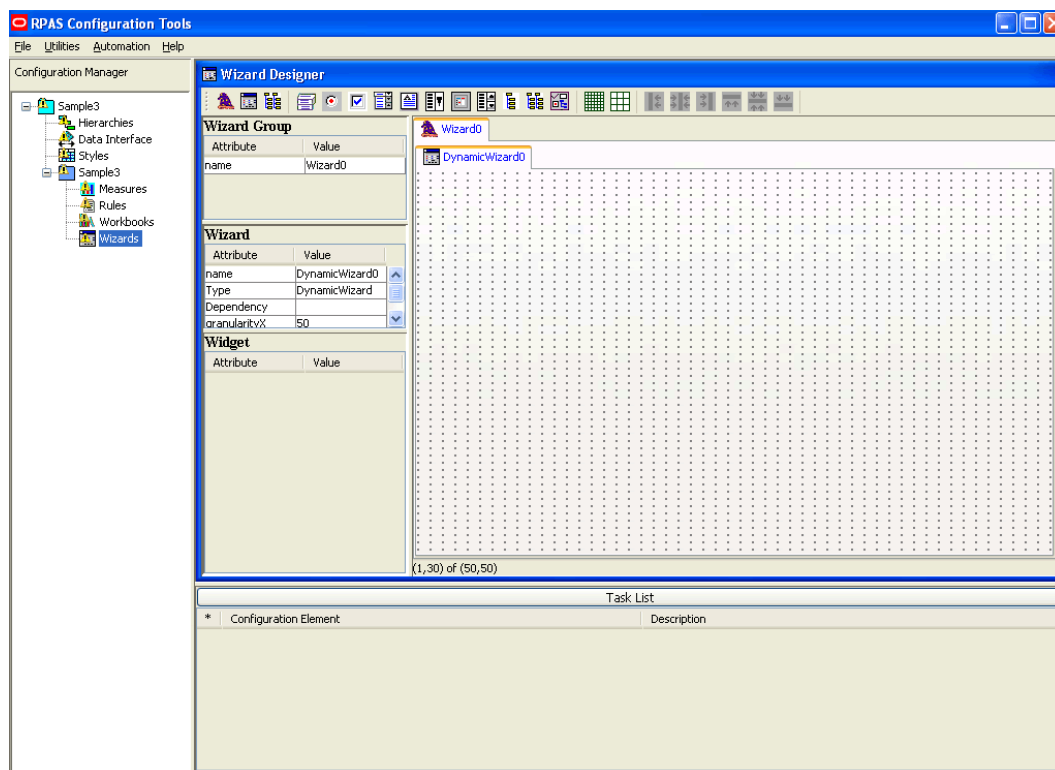
1. Laying out the wizards
2. Writing code to control the transition between the wizards and out of the wizards.
3. Writing code to initiate the building of the workbook.

The Wizard Tool only helps the workbook designer in the first aspect of this work. The workbook designer (the one who establishes the layout of these wizards) is expected to have knowledge of the process of controlling, collecting, and processing the information from these wizards.

The rest of this section describes working with layout of the custom wizards.

Create a Wizard Group

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Wizards**. The Wizard Designer window opens in the workspace.



Wizard Designer Window

1. In the Wizard Designer window, click the **New Wizard**  button.
2. In the Wizard Group area, click in the Value field, and enter the name of the Wizard Group.

Create a Wizard Page

1. Click the button to add the appropriate control. The table below describes the buttons and their function.

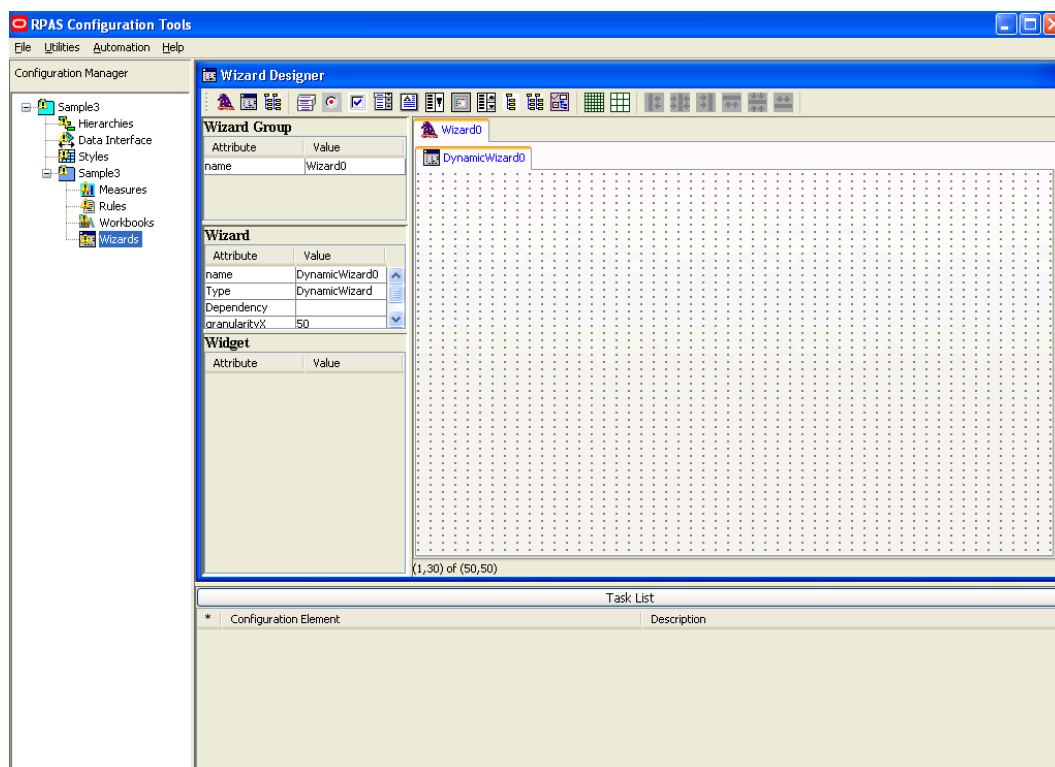
Button	Function
	Allows you to create a new dynamic wizard.
	Allows you to create a two tree page.
	Allows you to create a label field that cannot be edited in the resulting wizard.
	Allows you to create a radio button field from which you can make one choice from several options.
	Allows you to create a checkbox field from which you can make multiple choices from multiple options.
	Allows you to insert a drop-down list box from which you can select the entered choices. The choice you select is displayed in the text box.
	Allows you to insert a field in which you can enter free-form text.
	Allows you to insert a list box that contains a list of items from which you can select.
	Allows you to create a labeled area in the wizard where you can group other wizard elements.
	Allows you to insert a date picker (spinner) that contains independently scrollable date components.
	Allows you to insert a single tree into the wizard.
	Allows you to insert a generic object in the wizard.

2. Click on the wizard page grid to place the selected control on the page.
3. If necessary, drag the control to the appropriate place on the grid to reposition it.
4. In the Widget area, enter the following information:
 - **Name** – The RPAS internal name of the control.
 - **Type** – The control type.
 - **Text** – The text to be displayed on the control label.
 - **Align** – The same as the style attribute. Valid values are left, right, center, multiline, and flip depending on the type of widget you are creating.
 - **Func** – Indicates whether the widget will be dynamic or static. These are the only valid values for this attribute.
 - **Locx** – The x coordinate of the control on the wizard page. The value of this field is automatically changed when you move the control using your mouse.
 - **Locy** – The y coordinate of the control on the wizard page. The value of this field is automatically changed when you move the control using your mouse.

- **Width** – The width of the control in pixels.
- **Height** – The height of the control in pixels.

Edit Control Properties

Navigate: In the Configuration Manager, select  **Project** –  **Solution** –  **Wizards**. The Wizard Designer window opens in the workspace.



Wizard Designer Window

1. Select the wizard group tab that contains the wizard that you want to edit.
2. Select the wizard tab that contains the widget that you want to edit.
3. Select the widget.
4. In the Widget area, update the information as necessary.

System Preferences

Overview

You can set general preferences for the configuration tools at the workbench level, which refers to the entire tool. This includes the domain type and general preferences.

Global Domain

Overview

A Global Domain environment provides the ability to view data from multiple physical domains in a single workbook, and to administer common activities centrally across the RPAS solution.

Domains can be built in one of two methods:

- Simple Domain – This is the traditional, stand-alone domain that has no visibility to other domains.
- Global Domain – This is a domain environment that contains two or more local domains (or sub-domains) and a master domain that has visibility to all local domains that are part of that environment.

There are two primary functional benefits in using a Global Domain environment:

- The ability to have a global view of data in workbooks.
You can build workbooks with data from multiple local domains, refresh global workbook data from local domains, save global workbooks, and commit the data from global workbooks to the individual local domains.
Local domains are typically organized (partitioned) along organizational structures that reflect user responsibilities and roles. Most users will only work within the local domain(s) that contain their area of responsibilities, and they may not need to be aware of the Global Domain environment. For performance and user contention reasons, Global Domain usage should be limited to relatively infrequent processes that require data from multiple local domains.
- Configuration and Administration.
Most of the mechanisms that are required to build and administer a domain are centralized, so they need only be run in the “master” domain, which either propagates data to the local domains or stores it centrally so that the local domains reference it in the master.

Note: For a Global Domain environment to function properly, all local domains must be structurally identical.

Measure Data

In a global domain environment, measure data can be physically stored across the local domains or in the master domain.

Measure data that is stored in local domains is split across the domains based on a pre-determined level of a given hierarchy. This level is defined during the configuration process, and it is referred as the “partition” level.

The base intersection of a measure (the dimensions that a measure contains) determines whether data is stored in the local domains or in the master domain. The data will be stored in the master domain if the base intersection of a measure is above the “partition” level or if it does not contain the hierarchy on which the Global Domain environment is partitioned. This type of measure is referred to as a “Global Domain measure,” or a “Higher Base Intersection measure.”

Consider a global domain environment where the partition-level is based on the Department dimension in the Product hierarchy. In this scenario, data for measures that have a base intersection in the Product hierarchy at or below Department (other hierarchies are irrelevant for this discussion) is stored in the local domain. This is based on the Department that the underlying position in the Product hierarchy belongs to.

Measures that have a higher base intersection in the Product hierarchy than Department (for instance, Division) or measures that do not contain the Product hierarchy (such as a measure based at Store/Week) cannot be split across the local domains. These measures will reside in the master domain, and they will be accessed from there when these measures are required in workbooks.

All measures will be registered in the master domain, and they will be automatically registered in all local domains. RPAS automatically determines where the measure needs to be stored by comparing the base intersection of the measure against the designated partition-level of the Global Domain environment.

The physical location of the measure data will be invisible to the user after the measure has been registered.

Hypersparse

The term Sparsity describes the density of measure data, which indicates the proportion of data points that have populated values. A populated value is a value that is not equal to the NA value, which is most commonly zero for numeric measures. By default, domains are not hypersparse, which means that they only contain dense or sparse data. However, if a domain is expected to be a certain size (based on the number of positions in the hierarchy), the domain must be a hypersparse domain. This is most common for very large environments.

Whether a domain should be hypersparse is a technical decision that should be made with the consultation of Oracle Services.

Note: It is possible to convert a domain to and from a hypersparse setting after the domain has been built by using an RPAS utility. This is described in the *RPAS Administration Guide*.

Multi-Language

RPAS domains are built to be used in English only or in English and other languages. Multi-lingual domains allow for most data elements in an RPAS domain to be translated into another language. The translation process is managed by the Oracle Translation group and is handled as a separate agreement with Oracle.

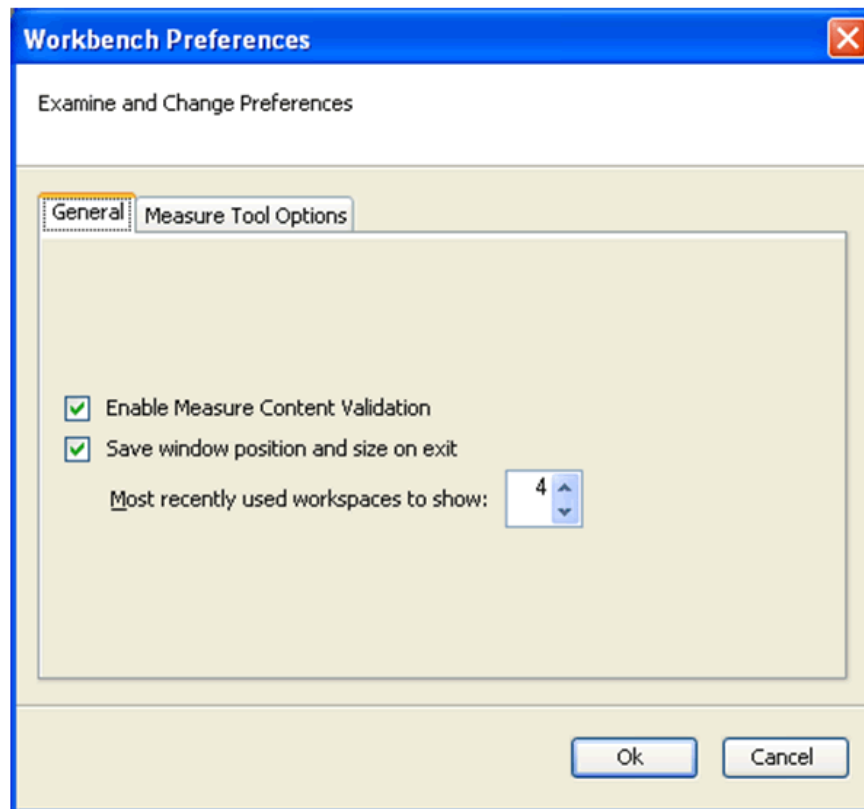
Note: An existing domain cannot be converted to multi-lingual after it has been built.

Setting Tools Preferences

Settings made here will apply to all configuration projects created or viewed with the Configuration Tools.

Navigate: From the File menu, select **Tools Preferences**. The Workbench Preferences window opens.

1. Select the **General** tab.

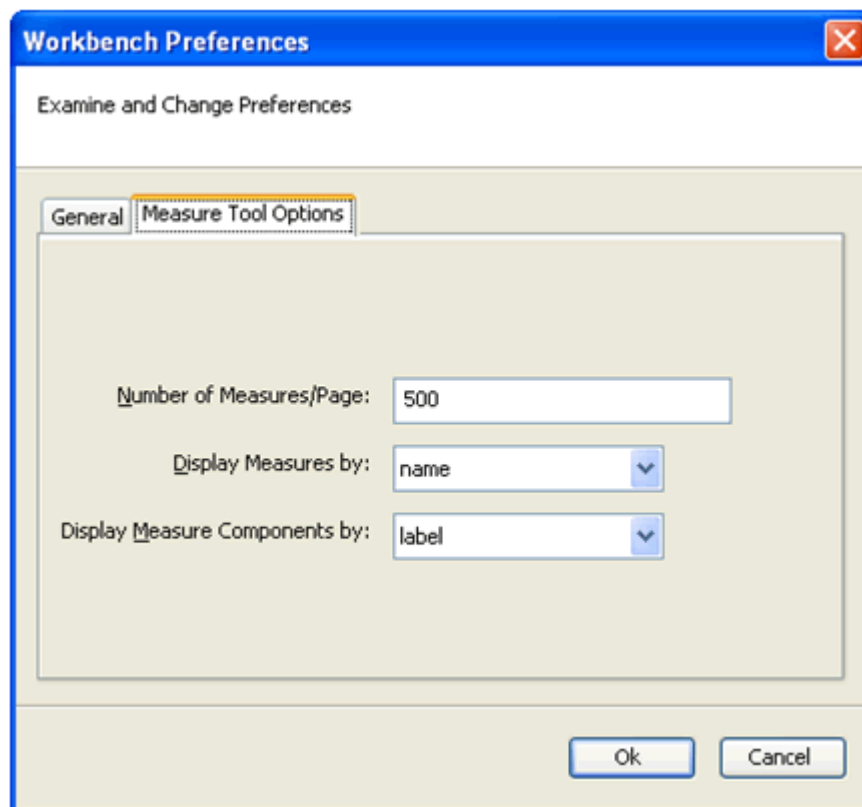


Workbench Preferences Window

- **Enable Measure Content Validation** – Activating this checkbox enables the immediate validation of measure properties when configuration measure information is created or modified. This process can impact the performance of the RPAS Configuration Tools. If this box is not checked, the manual Measure Content Validation icon is enabled on the Rule Definition toolbar. See the Rule Definition Tool for details.

- **Save window position and size on exit** – Activating this checkbox enables the RPAS Configuration Tools to be launched in either full or minimized view based on the status in which the configuration administrator last exited the application. If the application is exited in a minimized view, the size of the window is also maintained when the RPAS Configuration Tools is re-launched.
- **Most Recently used workspaces to show** – Use the up and down arrows to specify the number of configurations to be displayed in the Most Recently Used list displayed in the File menu dialog. This list allows the configuration administrator easy selection of a recently viewed configuration.

2. Select the **Measure Manager Options** tab.

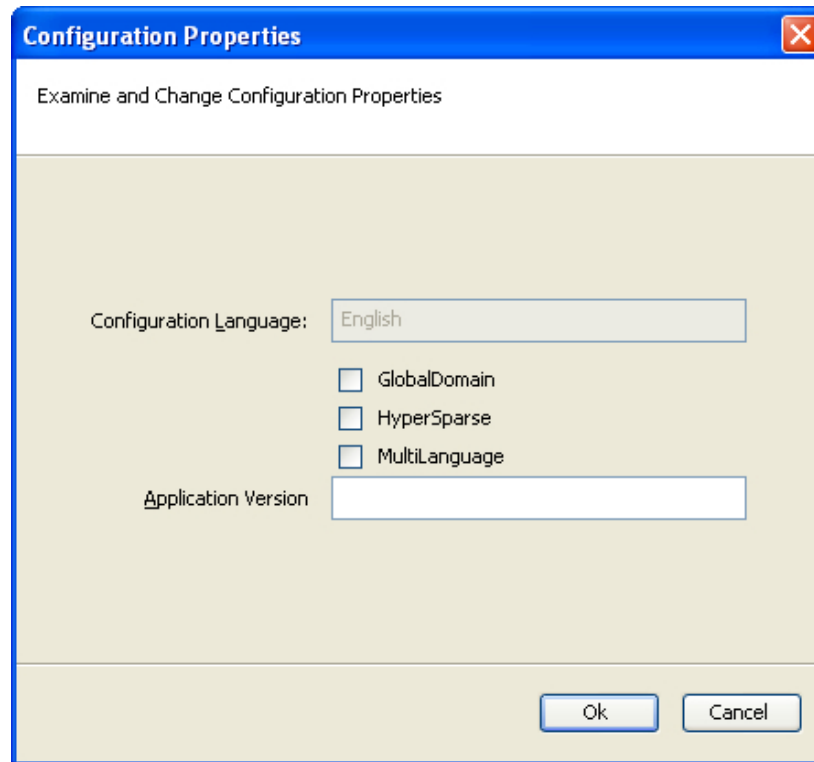


Workbook Preferences - Measure Tool Options

- **Number of Measures/Page** – Select the number of measures to display per page in the Measure Manager Tool. The default is 500.
 - **Display Measures by** – Displays measures either by their name or label in various locations of the Tools. The default setting is “name.”
 - **Display Measure Components by** – Display measure components (in the Measure Manager tool) either by name or by label. The default setting is “label.”
3. Click **OK** to save any changes and close the window.

Set Configuration Properties

Navigate: From the File menu, select **Workspace Properties**. The Configuration Properties dialog box opens.



Configuration Properties Dialog Box

Configuration Language field displays English by default and cannot be changed. Currently configurations can only be in English.

1. Select the following options as necessary:
 - **Global Domain** – The configuration uses a global domain environment. This environment enables you to create workbooks from multiple domains and to administer and update multiple domains from a single master domain.
 - **Hypersparse**– The configuration uses a hypersparse domain. Hypersparse is a format for storing data where up to 5 quintillion cells can be handled.
 - **MultiLanguage** – The configuration supports multiple languages in the domain.
2. Click **OK** to save any changes, and close the dialog box.

Configuration Utilities

Overview

The utilities in this section are standalone utilities that can be run externally or they can be launched from the utilities menu of the Configuration Tools. The utilities provided include the Configuration Converter and the Function Library Manager; which are described in detail.

Configuration Converter

Overview

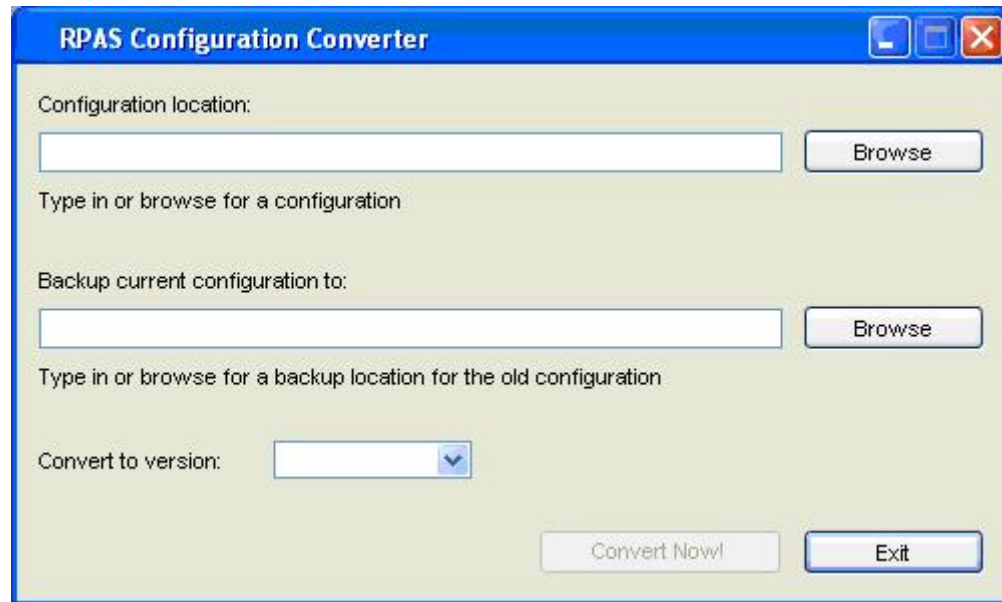
Note: The functionality for converting a configuration is provided directly through the Configuration Tools. See the section, “Open an Existing Project from an Older Version of the Configuration Tools” in Chapter 3.

The Configuration Converter is a standalone utility that converts a configuration that was originally created and saved in a prior release of the Configuration Tools. Only configurations created in a prior major release need to be converted. Configurations saved in previous versions of the same major release, but in different minor releases, do not need to be converted.

Launching the Converter

- The Converter can be accessed in following ways:
- Open the Utilities menu in the Configuration Tools, and select **Configuration Converter**.
- From the Windows Start menu, select **Oracle–RPAS–Utilities**, and click the Configuration Converter icon.
- Run the RpasConverter.exe file in the "utilities" directory where the Configuration Tools were installed.

The Configuration Converter window appears.



Configuration Converter Window

Converting a Configuration

1. In the **Configuration location** field, enter the configuration that you want to convert. This will be the file in your configuration's directory that has the configuration name with an ".xml" extension (for example, "C:\Configs\MyConfig\MyConfig.xml"). Type the path in, or click the **Browse** button to graphically browse for the file.
2. Enter a directory in the **Backup current configuration to** field where you want a copy of your original configuration to be stored. Either type in the directory, or click the **Browse** button and search for a directory.

Note: The directory that you enter must not already contain a directory whose name is the name of your configuration. For example, to put a backup of a configuration named "MyConfig" in a directory "C:\Backups," "C:\Backups\MyConfig" must not exist.

3. Select the version to convert to using the **Convert to version** list. This should always be the current version of the Configuration Tools unless there is a good reason to convert an older version.
4. Click **Convert Now**.
If the conversion was successful, it may now be opened in the Configuration Tools. If there was an error while converting, an error message will be displayed, and the original configuration will remain untouched.

Functional Library Manager

Overview

The RPAS calculation engine is designed to be extensible with support for custom functions or procedures that can be used in normal expressions. For validation purposes, the configuration tools are only aware of the standard RPAS functions and procedures, so they will generate an error for any expressions that use custom functions or procedures. The Function Library Manager is used to allow custom functions or procedures to be validated within the Configuration Tools. The custom functions or procedures must exist in the /applib directory of the RPAS_HOME directory. If necessary, you can also use this utility to remove custom function libraries from being validated. There is no validation that the function libraries exist in RPAS_HOME/applib. When a function library is removed from the manager, it is removed only from the list of external libraries used for validation, and the contents of RPAS_HOME/applib are left intact. The function libraries mentioned in this list are loaded by the configuration tools and will be used for the validation of rules.

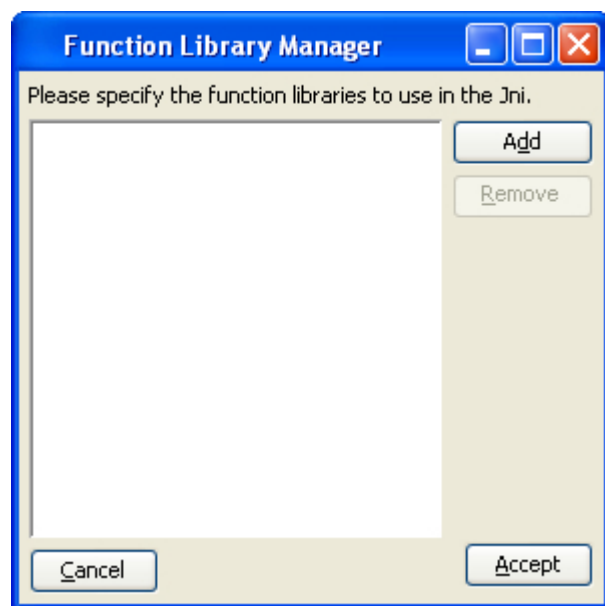
Speak to an Oracle Retail Services representative for additional information about custom functions and procedures.

Launching the Functional Library Manager

Navigate: Perform one of the following options:

- From the Utilities menu in the Configuration Tools, select **Function Library Manager**. The Function Library Manager window opens.
- Run the Functional Library Manager.bat file from the "utilities" directory where Configuration Tools is installed.

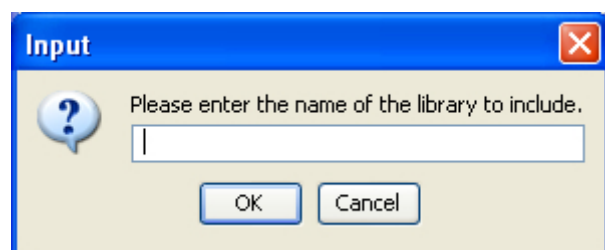
The Functional Library Manager window opens.



Function Library Manager Window

Adding a Function Library to Be Validated in the Configuration Tools

1. Launch the Function Library Manager.
2. Click **Add**. The Input Window opens.



Input Dialog Box

3. Enter the name of the library to include.

Note: Enter the name without the *.dll or *.so extension.

4. Click **OK**.
5. Click **Accept** to save any changes and close the window.

Removing a Function Library from being Validated in the Configuration Tools

1. Launch the Function Library Manager.
2. Select the Function Library to remove from being validated.
3. Click **Remove**. The Function Library is removed from the list.
4. Click **Accept** to save any changes and close the window.

Report Generator

Overview

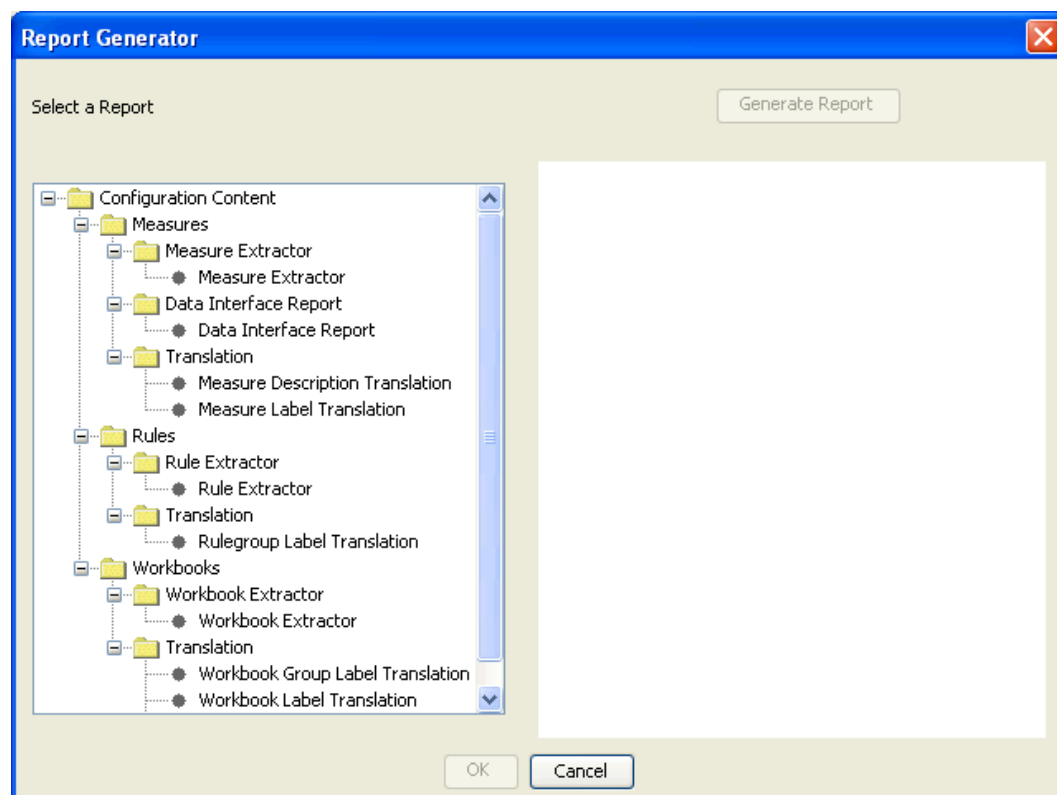
The Report Generator is a utility that may be used to extract information about a configuration for external use. The information is generated in a structured text document that is much easier to manipulate than the XML format configuration files that are saved and loaded by the workbench. Many of the reports correspond to files generated as a part of the installation process.

Available Reports

- **Measure Extractor** – This report generates a text file that lists the measure content of a Solution.
- **Data Interface Report** – This report generates a text file that lists the properties of all of the measures in the Project that have been added to the Data Interface Tool.
- **Measure Description Translation** – This report generates a translation file similar to the file generated as part of the installation process. It allows the extraction of measure descriptions for a Project without the need to first build the Domain.
- **Measure Label Translation** – This report generates a translation file similar to the file generated as part of the installation process. It allows the extraction of measure labels for a Project without the need to first build the Domain.
- **Measure Patch Report** – This report examines a previous version of a configuration to determine which measure properties have changed between the two versions. It is used to determine which measures will be added, removed, or updated during a patch installation.
- **Rule Extractor** – This report generates a text file that lists the rule content of a Solution.
- **Rule Group Label Translation** – This report generates a translation file that is similar to the file generated as part of the installation process. It allows the extraction of rule group labels for a Project without the need to first build the Domain.
- **Workbook Extractor** – This report generates a text file that lists the workbook content of a Solution.
- **Workbook Group Label Translation** – This report generates a translation file that is similar to the file generated as part of the installation process. It allows the extraction of workbook group labels for a Project without the need to first build the Domain.
- **Workbook Label Translation** – This report generates a translation file that is similar to the file generated as part of the installation process. It allows the extraction of workbook labels for a Project without the need to first build the Domain.
- **Messages Translation** – This report generates a translation file that is similar to the file generated as part of the installation process. It allows the extraction of messages issued by the RPAS client for a Project without the need to first build the Domain.

Generate a Report

1. Select the project that requires the report.
2. Select **Generate Reports** from the Utilities Menu. The Select a Report dialog opens.



Select a Report Dialog Box

3. Select the desired report from the list in the left pane of the generator dialog. The right pane displays a short description of the currently selected report.
4. Select **Generate Report** to begin the report generation process.
5. Depending upon the report in question, there may be a number of options to specify in further dialogs. These options commonly include the location of the generated file or the selection of a single Solution from the Project.
6. Once all options have been specified, click **OK** to generate the report. The **OK** button will not be enabled until all options have been specified.

Building and Patching Domains

Build a Domain

Overview

After you create a fully defined configuration, you can build an RPAS domain. Since building an RPAS domain is a manual process, you should be skilled in administering UNIX if installing the domain on a UNIX platform. If the domain is being installed on NT servers, it can be built using the GUI installer.

Prerequisites

The following are the prerequisites for building a domain:

- Installation of RPAS on your server. Refer to the *RPAS Installation Guide* for more information.
- Installation of the Configuration Tools on your server. Refer to the *RPAS Installation Guide* for more information.
- Configuration built using the Configuration Tools.
- A collection of hierarchy input files that contain positions for the domain. A hierarchy data file (name.dat) is required for each defined hierarchy.

Client-Side Procedures

Export the Configuration Project from the Configuration Tools

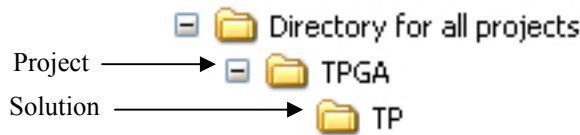
Note: If you are building the domain directly from the GUI Installer, this section may be omitted.

To begin the domain build process, you need to start with a configuration project that was built using the Configuration Tools. You can use a purchased configuration or create a new one with your specific hierarchies, measures, and workbooks. If you are creating a new configuration, be sure to note the path where your configuration is saved on your local disk.

Note: The remainder of the section assumes that the domains are being built on a different server than NT while the configuration is created on the NT platform. If the domain is being built on the same server as the configuration, the steps regarding adding the configuration to the zip file and transferring to a different server can be eliminated.

Zip up the Configuration Project Folder

1. Find the location where you saved your configuration project.
2. Using Windows Explorer, go to the path of the configuration project.
3. Right-click on the Configuration folder, and select Add to Zip. You should zip the entire contents of the project beginning with the configuration project root folder such that the zip file will include all Solution(s). It is important to zip the entire configuration project for the entire directory structure and not just the specific .xml files. Do not change the name of the configuration project folder or alter the contents of the folder in any way.
4. In the following example, you should click on TPGA to highlight the folder. Right-click and select Add to TPGA.zip.



5. Using FTP, transfer the .zip file over to the server in binary mode. You can place it in your home directory for now.

Load Measure Data

Note: The following is optional. If you have already purchased a solution template that contains a script to load data, you can skip this section and proceed with the installation process of your solution template.

During the initial domain build process, all of the hierarchy and measure data files in your input home directory will be copied to the input directory domain. If necessary, move or copy any new measure data files to the input directory of the domain. The measure data file must have a valid measure filename, such as xxx.ovr, xxx.rpl, xxx.inc, or xxx.clr, where xxx is the measure name.

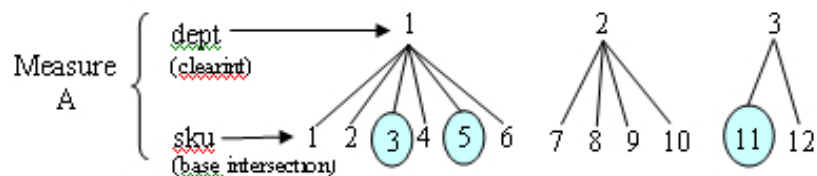
The meanings of the extensions are as follows:

- **ovr** – The data from the incoming file will overlay the data that is already present at the base intersection for the measure. If a particular cell at the base intersection does not receive a new record from the incoming file, that cell's current value will be preserved.
- **inc** – The data from the incoming file will be added to any existing data at the base intersection of the measure.
- **rpl** – All data at the base intersection for the measure will be removed before cells are populated with the data from the incoming file.
- **clr** – Data at the base intersection of a measure will be partially cleared based on incoming data and the clearint attribute for the measure. The clearint attribute defines an intersection above the base intersection. All cells at the base intersection that are descended from a given position at the clearint level will be removed if there is data in the incoming file for at least one of those descending positions.

For example:

Measure “A” has a base intersection of “sku” and a clearint of “dept.”

If there is data for only a few skus (3, 5, and 11) in the incoming file, and skus 3 and 5 roll up to dept1 while sku11 rolls up to dept3, the data in all of the skus that rolls into dept1 and dept3 will be cleared.



- The base intersection is sku.
- The clearint is dept.
- Data is present for skus 3, 5, and 11; which fall under dept 1 and dept3.
- Data will be cleared for dept1 (skus 1, 2, 3, 4, 5, 6) and dept3 (skus 11, and 12).
- Data for skus in dept2 (7, 8, 9, and 10) will be untouched.

Refer to the *RPAS Administration Guide* for the following procedures:

- Loading data using the loadmeasure utility
- Disabling the timestamp header

Also refer to the *RPAS Administration Guide* for information on measure loading properties.

Server-Side Procedures

Import the Configuration to RPAS

Note: If you are building the domain directly from the GUI Installer, this section may be omitted.

After you have exported the configuration project, you can import it to the server and begin building the domain. You can build a new domain, or you can add fixes to an existing domain by applying patches. Once you have made all of the changes to your domain, you should validate the domain build results.

Note: Though the RPAS Configuration Tools are supported only on the Windows platforms, the installation tools are supported on all platforms (Sun, HP-UX, AIX, and NT). However, they require Java v1.4 or higher. Make sure that the server you will be using has this version of Java.

Unzip the Configuration File

1. Find the location where you want to save your configuration project file.

Note: Always put a newly changed configuration project in a new directory path. Do not overwrite an existing configuration project.

2. Move the <Configuration Project>.zip file to this location.
3. Unzip the <Configuration Project>.zip file using the UNIX command:
`unzip a <Configuration Project>.zip`

NOTE: Do not change the directory name for the configuration project or alter the contents in any way.

Verify the Environment Variable Settings

Prior to this step, you should have installed RPAS and the Configuration Tools on your server. During that process, you should have set up the necessary environment variables for RPAS and the Configuration Tools. Refer to the *RPAS Installation Guide* if the environment variables below have not been set up.

Log in to the server and use the commands below to verify your environment settings:

- `echo $RPAS_HOME`
- `echo $RIDE_HOME`
- `echo $JAVA_HOME`
- `echo $PATH`

Note: The path for the `RPAS_HOME` variable changes with each new RPAS release.

If you make any changes to the environment variable settings, remember to exit and restart your UNIX session in order to execute your `.profile` and make the changes effective. This step is very important before you continue to the remaining steps.

Get the Input Files Ready

1. Designate a directory for the location of your input files, and move your files into this directory.

Note: As a recommendation, use the directory name `to_rpas` as a standard for the location of input files. At a minimum, the hierarchy files (product, location, and calendar hierarchy files) are needed to build the domain. At this time, a calendar file must be loaded.

2. If necessary, rename the hierarchy files to `prod.dat`, `loc.dat`, and `clnd.dat`. These file names are required. The format of these files must match the hierarchy configuration specified using the configuration tools.

Manually Building the Domain Using a Command-Line Interface

Run the `rpasInstall` script to build the domain. This executable is located in the path to `$RIDE_HOME/bin`. The syntax for the `rpasInstall` script is as follows:

```
rpasInstall [-version | -testinstall | -fullinstall | -patchinstall] cn
<config_name> ch <config_home> dh <domain_home> in <input_directory> [-rf
<function_name>] log <logfile_name>
```

Note: A full explanation of the RPAS Install options is available in the *RPAS Installation Guide*.

Required:

`[-fullinstall | -patchinstall | -testinstall]`– The type of installation to be performed.

where:

- fullinstall – Builds a full domain and loads the hierarchy data files.
- patchinstall – Patches an existing domain. Updates or unregisters/registers measures that have changed (as necessary).
- testinstall – Used for testing only. Only generates configuration files.

`ch <config_home>` – Where `<config_home>` is path to the directory that contains the configuration.

`-cn <config_name>` – Where `<config_name>` is the name of the configuration.

`-in <input_home>` – Where `<input_home>` is the directory that contains the input files for the domain to be created.

`-log <log_name>` – Where `<log_name>` is the name of the log file to be created.

Optional:

`-v-version` – Provides the version number of the Tool build as it relates to the release or patch number.

`-rf <function_name>` – Where `<function_name>` is the filename of the function to be registered. This pairing may be repeated for multiple functions.

Conditional:

`-configdir <config_directory>` – Where `<config_directory>` is the path to the directory that contains the xml files that are used by RPAS. This is a required argument if you want to supply `globaldomainconfig.xml` or `calendar.xml`.

`-dh <domain_home>` – Where `<domain_home>` is the path to the directory where the domain will be created. This does not include the root of the domain. This is a required option if `globaldomainconfig.xml` is not used.

`-p <dim_name>` – Where `<dim_name>` is the partitioning dimension. Use this if the global domain is being implemented without the use of `globaldomainconfig.xml`.

Note: When building a domain for the first time, an installs directory is created inside the domain. The installs directory is essential for the patch process, so you should not move or rename this directory.

When sending in an issue to Oracle Retail Customer Care, if asked to provide a domain, be sure to provide the installs directory with it. For completeness, you should provide the following items in order to help Customer Care better diagnose the issue:

The configuration

The script used to run the rpaInstall script

The domain

The log output file

Validate Domain Build Results

Once the domain build process is completed, you should check the logfile to verify that the process executed successfully. Search for the words **ERROR**, **FAILURE**, and **exception** inside the logfile. The end of the logfile should look similar to the output below.

Time: 58.451

COMPLETE

Building the Domain using the GUI Installer

Domains can either be built directly from the RPAS Configuration Installer or by running a command line script generated by the installer. This tool can be used in one of two ways.

- It will install a domain and show the output from the install process in real-time.
- It can generate a script that can be run at a later time.

When first run, the RPAS Configuration Installer window appears.

RPAS Configuration Installer

Fill in this screen to build the configuration. The Installer uses the configuration to build the domain.

1. Select the type of install desired.
This can be a full install, a patch install, or a test install.
 - Full install creates a new domain.
 - Patch install patches an existing domain.
 - Test install produces all of the required files for a full install, but it does not install the domain.

Note: For a detailed description of the required fields, see “Build a Domain” under “Server Side Procedures.”

2. Perform one of the following:
 - Specify the configuration to install from by typing in a path to a configuration that was created using the RPAS Configuration Tools
 - Click the **Browse** button next to the text box and select the configuration's top-level .xml file (the same file that would be opened when opening the configuration in the Configuration Tools).

3. Perform one of the following options:

- Specify the “Input files” directory by typing a path to the directory that contains the input files.
- Click the **Browse** button next to the text box and select the directory that contains the input files.

The directory must contain hierarchy load files even for a patch install.

Note: See the example under “Specify Dimension Information” for the syntax of these files and how their layout is configured.

4. Perform one of the following options:

- Specify a “Log file” by typing in the path that contains the Log file
- Click the **Browse** button next to the text box and select the directory that contains the Log file.

5. Perform one of the following options:

- If the configuration is using a global domain, specify the **RPAS Config Directory** by typing in the path that contains the directory.
- Click the **Browse** button next to the text box and select the directory that contains the RPAS Config directory.

The RPAS Config directory is the directory that contains the hierarchy.xml and the GlobalDomainConfig.xml files.

Note: See “Manually Building the Domain Using a Command-line Interface.”

6. Perform one of the following options:

- Specify the **Domain home** directory by typing in the path that contains the directory.
- Click the **Browse** button next to the text box and select the directory that contains the Domain home directory.

The Domain home directory is the directory beneath which the domain will be created. For a patch install, it is the directory above the domain.

Note: This parameter is not required if the RPAS Config Directory contains a valid GlobalDomainConfig.xml file.

7. Select the **Global Domain** option to specify a partitioning dimension.

8. If you have selected **Global Domain**, type the partitioning dimension in the **Specify the Partitioning dimension** field. For example, if the domain is to be partitioned at “dept” level, type “dept” in the input box. This means for each dept in the hierarchy, a different local domain will be created.

Note: This parameter is not required if the RPAS Config directory contains a valid GlobalDomainConfig.xml

9. To update style, select the **Update Styles** option.

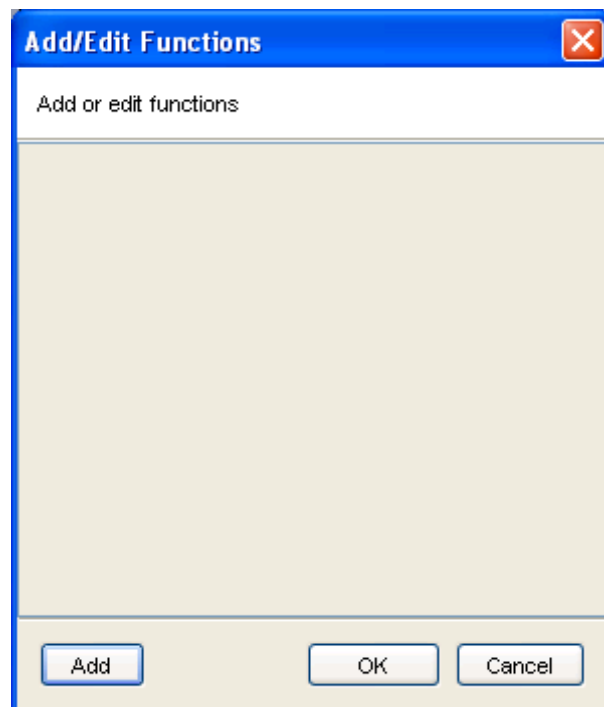
When patching an existing domain, styles on the domain will not be changed if this option is unchecked, which protects any formatting defined through the RPAS UI. If it is checked, all styles will be updated, and any styles defined through the RPAS UI will be lost.

Note: This option only applies to patches. It does not apply to creating new domains.

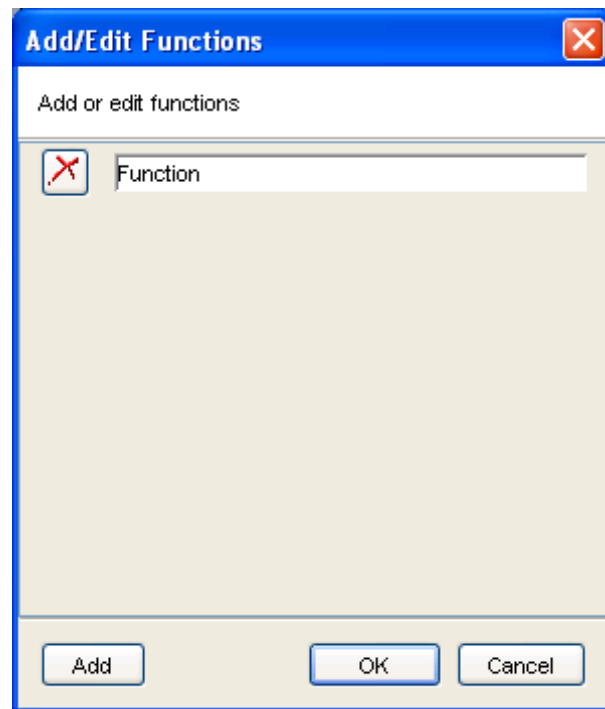
As fields are filled in, a status bar near the bottom of the screen will indicate the next field that should be entered, or it will indicate a problem if one is found.

Note: Fields may become grayed-out and unable to be edited depending on data that is entered.

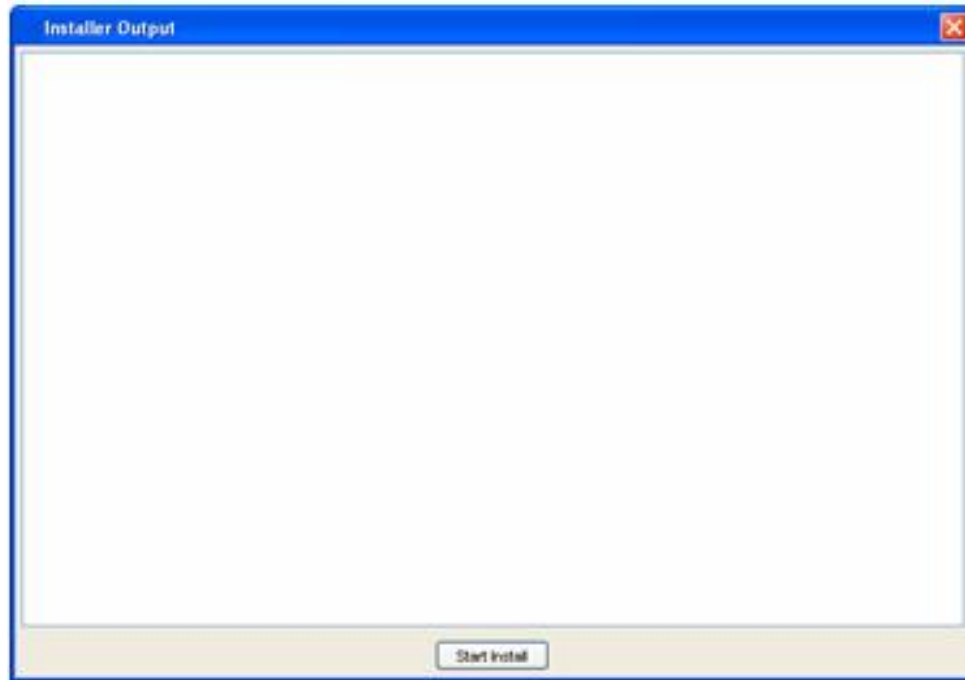
10. To disable checking of input data for required fields and errors, uncheck the **Validate Input** box, which is located in the lower left of the main Installer window. This will enable input to all fields and will disable validation on fields that have been entered. This is useful if you are generating an install script to run on a different machine, because the paths to the input data will likely be different than on the machine where the Installer is being run.
11. To add functions, click **Add/Edit Functions**. The Add/Edit Functions dialog box appears.



- a. Click **Add**. A function input box appears in the dialog box.



- b. Type a function in the text field.
 - c. To delete functions that have been added, click the red **X** button next to the field containing the function to delete.
 - d. Click **OK** to save changes and go back to the main Installer screen.
12. Perform one of the following options:
- After all required fields have been filled in (or, if validation is turned off, after all desired fields have been filled in), click the **Save** button to generate a script that can be run at a later time
 - Click **Run** to run the installer.
- The Installer Output window appears.



- Click the **Start Install** button to begin the installation. The installer output appears in the window.

After Building the Domain

After building the domain and loading measure data:

1. Use the RPAS EConfigure utility to specify the location of the domain on the server and designate a port number where the RPAS server (DomainDaemon) is running.
2. Begin the RPAS server process (DomainDaemon) with the port number specified in the EConfigure utility. Refer to the *RPAS Administration Guide* for instructions on executing the DomainDaemon.
3. Log in to the domain using the RPAS client.
4. Set up the appropriate permissions on the measures in the domain using the Measure Administration workbook, and view the measures by building a Measure Analysis workbook. Refer to the *RPAS Administration Guide* for instructions on using these workbooks.

Patching a Domain

In order to patch your RPAS domain with changes to your configuration project:

- Use the `-patchinstall` option when running the command-line installer.

OR

- Select the **Patch Install** option if using the GUI Installer.

Before beginning the patch installation process, it is recommended that you:

- Save all workbooks.
- Create a backup copy of the domain.
- Verify that you are using the `patchinstall` option in the `rpasInstall` command instead of the `-fullinstall` option.

Note: You should only change the `fullinstall` option to `patchinstall` inside your call to `rpasInstall`. You should not change any other options in the `rpasInstall` call when running a `patchinstall`.

Patching a domain does not update any hierarchy or dimension changes in your configuration with the exception of user-defined dimensions. It does not patch changes to the hierarchy purge age, the security dimension, or change the multi-language setting for a domain.

Note: If you need to update the hierarchy purge age inside an existing domain, you can use the `loadHier` RPAS utility in batch mode to update the current settings. Refer to the *RPAS Administration Guide* for further details regarding the usage of the `loadHier` utility.

Patching a domain allows for changes to some, but not all, measure properties. Patching a domain also allows for changes to rule sets, rule groups, and workbook templates. During the process; the rule sets, rule groups, rules, and workbook templates are completely rebuilt.

Applying a New Tools Build Patch Release

When there is a new release of the RPAS Configuration Tools, you will need to replace the entire Tools directory with the new release. You should make a backup copy of your existing configuration before using it with the new Tools release. You should also create a new directory for the new Tools release rather than overlaying the old release. Remember to update your `RIDE_HOME` environment variable as needed to point to the new Tools release. Refer to the *RPAS Installation Guide* for more details.

Patching Measure Properties

Certain measure properties cannot be updated without unregistering and reregistering measures, which results in the loss of measure data. Updateable measure properties are those that can be modified, and not updateable measures are those cannot be modified. If all of the measure properties that are changed are “updateable,” simply update the measure properties, and measure data will not be lost in the domain during the patch process. If even one of the measure properties that change are not updateable, the patch process results in unregistration and re-registration of the measure. Therefore, the data that was in the domain for that measure before the patch process will not be there after the patch process is complete.

Appendix: Global Domain Technical Information

About Domains

A domain can be implemented as a simple domain when:

- the data size for individual measures is small.
- the number of users working on the domain at any given period of time is small.

The domain can be implemented as a global domain when:

- the data size is increasing due to the hierarchy size.
- there are several people using the same domain .

In the global domain environment, the global domain is accompanied by two or more sub-domains that contain a subset of the data that would have been in a simple domain. Each of the sub-domains contains a subset of one of the hierarchies of the global domain. More specifically, the sub-domains contain a subset of positions along the partition dimension. All of the measures that are defined to be at or below this dimension will be stored in the local domains. Measures above this dimension are stored in the global domain. You can directly access the local domains, and a subset of users will be dealing with each local domain. This way, there is less contention between users. A domain can only be partitioned along one hierarchy of the domain.

Note the following points when configuring and setting up a global domain environment:

- When creating a new global domain configuration file, `globaldomainconfig.xml`, reference the RPAS example of the configuration file, `globaldomainconfig_example.xml`, which is located in the `%RPAS_HOME%/domain/config_examples` directory of the RPAS installation. Reference this to use as a guide in building this file. Once this file is created, put it in the `configdir` directory path as specified by using the `-configdir` option in the Installer.
- In the `globaldomainconfig.xml` file, you must specify (in the path) the entire path to the master domain and the sub-domains including the root name of the domains. The path leading up to the root of the master domain must exist, but the master domain root directory must not exist at the beginning of the domain build process. The path to the sub-domains must also exist unless the sub-domains are located inside the root of the master domain.
- The master global domain will contain a directory called `config`, which will house the `globaldomainconfig.xml` file. You should not delete this directory or file.
- You should not delete the `tmp` directory under the domain home directory while the domain build process is taking place.
- To configure the global domain functionality, you can provide a `globaldomainconfig.xml` file that specifies how each of the local domains should be partitioned based on position groupings. If a `globaldomainconfig.xml` file is not provided and the partitioning dimension is provided, a local domain will be built for each position within the partitioning dimension. For example, if a company has five

departments, and "department" is the partitioning dimension, there will be five local domains and one master global domain.

- Conditional parameters that are used by the Installer for configuring and setting up a global domain environment are as follows:
 - `-dh <domain_home>`: where `<domain_home>` is the path to the directory in which the domain will be created
 - Non-Global Domain – required
 - Global Domain (with `globaldomainconfig.xml`) - required
 - Global Domain (with `globaldomainconfig.xml`) – throws usage error
 - `-configdir <config_directory>`: where `<config_name>` is the path to the configuration xml files, including `globaldomainconfig.xml`, `hierarchy.xml`, and `calendar.xml`
 - Non-Global Domain – optional, but required if using a `calendar.xml` file
 - Global Domain (with `globaldomainconfig.xml`) - required
 - Global Domain (without `globaldomainconfig.xml`) - optional, but required if using a `calendar.xml` file
 - `-p <dim_name>`: where `<dim_name>` is the partitioning dimension. Only valid if the configuration has been marked as a global domain configuration with the Properties dialog box. If the `configdir` option is specified and a `globaldomainconfig.xml` file is found there, the `-p` option will be ignored and the partitioning dimension that is specified within the `globaldomainconfig.xml` file will be used instead.
 - Non-Global Domain – throws usage error
 - Global Domain (with `globaldomainconfig.xml`) - throws usage error
 - Global Domain (without `globaldomainconfig.xml`) - required
- For patching a Global Domain implementation; you can change measures, rules, and workbook templates; and the master domain and local domains will be patched accordingly. You cannot turn a non-global domain implementation into a global-domain implementation and vice versa. The Global Domain flag in the configuration is ignored during the patch process, so the construction of the implementation will not change even if you change its status.
- When patching either a Global Domain implementation or a regular, standalone, or single domain; in the call to the Installer (`rpasInstall`); it is imperative that you do not change any of the parameters that were used during the original domain build. The only parameter that you should change is `fullinstall` to `patchinstall`. The other parameters need to remain unchanged. We recommend using a script for the `rpasInstall` call so that it is easier to change the `fullinstall` parameter to `patchinstall` while leaving the other parameters in their original state.

Appendix: Calculation Engine User Guide

Overview

The RPAS calculation engine is a very powerful and flexible engine that is built to support OLAP type calculations against a multi-dimensional model. At first sight, the engine is very complex. However, when the “building blocks” of the calculation engine are properly understood, much of this apparent complexity goes away. This overview of the calculation engine processes will therefore start by describing the three fundamental processes of aggregation, spreading, and expression evaluation before explaining how the various processes integrate into a comprehensive whole.

RPAS supports an OLAP type model. In this model, individual pieces of data (called cells) apply to a single position in one or more hierarchies or dimensions. These will typically include a ‘measures’ dimension, a calendar or time hierarchy, and other hierarchies such as for products and locations. The measures dimension is fundamentally different to the other hierarchies because measures (in other systems measures may be referred to as facts, performance indicators, or variables) represent the fundamental events or measurements that are being recorded, whereas the positions in the other hierarchies provide a context for the measurement (for instance; where, when, or what). Measures relate to one another through rules and expressions. Positions in all the other hierarchies relate to each other through hierarchical relationships.

RPAS supports two different forms of relationships between cells:

- Hierarchical relationships that require ‘aggregation’ and ‘spreading’
- Measure relationships that require rules and expressions

Hierarchical relationships, such as weeks rolling up to months, or stores rolling up to regions, require the aggregation of data values from lower levels in a hierarchy to higher levels. This is done by using a variety of methods as appropriate to the measure. To enable such data to be manipulated at higher levels, RPAS supports “spreading” the changes, which is also done using a variety of methods. Aggregation and spreading are basic capabilities of the engine that require no coding by the implementer other than the selection of aggregation and spreading types to use for a measure.

The inherent relationships between measures can be modeled through a rich rule and expression syntax. Most of the effort in configuring an application model is in modeling these relationships.

The RPAS calculation engine is designed to be robust and extensible, but in complete control of the calculation process. It enforces integrity of the data by ensuring that all known “relationships” between cells are always enforced whenever possible. Much of the logic of the processing of rules and rule groups depends on this basic principal.

Measure Definition and Base Intersections

Certain characteristics of a measure determine how the calculation engine should handle it with regard to calculation, aggregation and spreading, and the dimensions in the hierarchies at which the measure is calculated. Since this information applies across all rules and rule groups, it is set up as part of the definition of a measure.

Data Types

RPAS supports the following data types:

- **Real**
Floating point numeric values. Most measures are of this type.
- **Integer**
Numeric integer values. There are no special “spreading” algorithms for integer measures, which should normally be used only for measures that are calculated “bottoms up.”
- **Date**
Date and time. Can easily be converted to position names by standard functions.
- **String**
Variable length strings. Typically used for notes and names.
- **Boolean**
True or false values. Typically used for flags and indicators.

Base Intersection

The base intersection for a measure is a list of dimensions (such as Class/Store/Week), one per appropriate hierarchy, which defines the lowest level at which data is held for the measure. Data is assumed to apply to the “All” position in any hierarchy, which is not explicitly referenced in the base intersection (see [Non-Conforming Expressions](#) for more information). Through aggregation, data will logically exist (though there may not be a value) for all levels higher than the base intersection up all alternative rollups.

Aggregation and Spreading Types

The aggregation type defines the aggregation method to be used for the measure (see [Aggregation](#) on the following page) to produce values at higher levels from values at the base intersection. There is a ‘normal’ spreading method associated with an aggregation type, which defines the method to be used to spread changes from higher levels (see [Spreading](#)) to the base intersection. Depending upon the desired characteristics of the measure, there may be several valid allowed spreading types.

Aggregation

Overview

By definition, an OLAP type model has hierarchical relationships between positions in hierarchies. The values of measures above their base intersections for these hierarchical relationships are automatically maintained through a process referred to as aggregation.

Different types of measures need to be aggregated in different ways. Many measures, such as sales, receipts and markdowns, record the events that actually occurred or are planned to occur during a period of time. Simple totaling can produce aggregate values for these: the value for a region is the sum of the stores in the region; the value for a month is the sum of the weeks in the month; and so on. But this technique does not work for all types of measures. For example, with stock, the values record a 'snapshot' at a point in time rather than a total of events over a period of time. The value of stock for a region is the sum of the stock in the stores in the region, but the value of stock for a month is certainly not the sum of the stocks for the weeks in the month. It is usually either the value for the first week or the last week in the month. Similarly, there are measures where the appropriate aggregation type may be to calculate an average, or a minimum, and so on. For some calculation purposes, only cells that are 'populated' (have a value other than their default value, which is typically zero) should participate in aggregations. RPAS supports a wide variety of aggregation types to support all of these requirements.

There is also another class of measures where no aggregation technique would produce the correct result. These measures are typically prices, ratios, variances, and similar performance indicators. The average price of sales for a class cannot be calculated by summing the prices of items in the class. Averaging the prices of items in the class produces a better result, but it is still not accurate because it fails to take account of the weighting of the sales of the items in the class. One item with a very large volume of sales at a low price would pull down the average price attained for the class as a whole, but this would not be reflected in an average aggregation. The way to get a correct result is to redo the price calculation at the required level. By dividing the sales value for the class by the sales units for the class (both of which will have been aggregated by summing), a correctly weighted result will be produced. The type of measure that requires this type of "aggregation" is referred to as a "recalc" measure, as "aggregation" is by recalculation of the expression used to calculate the measure. In planning applications it is not unusual for 40% or more of the measures to be of recalc type.

Aggregation Types

The RPAS calculation engine supports the following aggregation types:

Aggregation Type	Measure Type	Description
Hybrid	For any measure type	The measure is aggregated using a specific aggregation type for each hierarchy. This is selected from the valid aggregation types for the measure type. At intersections that are aggregated in more than one hierarchy, the aggregation type used is that for the highest priority hierarchy.
recalc	For any measure type	The measure is not aggregated, but is recalculated at all aggregated levels through a recalc expression.
none	For any measure type	The measure is not aggregated. The value for all cells above the base intersection is the naval.
total	For numeric measures only	The measure is aggregated by taking the total (numeric sum) of all child values at the base intersection.
average	For numeric measures only	The measure is aggregated by taking the numeric average of all child values at the base intersection.
min	For numeric and date measures only	The measure is aggregated by taking the minimum of all child values at the base intersection. Note: For most purposes, the min_pop aggregation type will be more appropriate because the minimum value of all child values will typically be the naval, which is usually zero.
max	For numeric and date measures only	The measure is aggregated by taking the maximum of all child values at the base intersection.
median	For numeric measures only	The measure is aggregated as the median value (the middle value when sorted from lowest to highest) of all child values.

Aggregation Type	Measure Type	Description
pst [period start total]	For numeric measures only	For cells at the base intersection in the time hierarchy, the measure is aggregated by taking the total (numeric sum) of all child values. For cells at aggregated levels in the time hierarchy, the measure is aggregated by taking the value of the first child time period.
pet [period end total]	For numeric measures only	For cells at the base intersection in the time hierarchy, the measure is aggregated by taking the total (numeric sum) of all child values. For cells at aggregated levels in the time hierarchy, the measure is aggregated by taking the value of the last child time period.
and	For Boolean measures only	The measure is aggregated by performing a Boolean and of all child values.
or	For Boolean measures only	The measure is aggregated by performing a Boolean or of all child values.
ambig	For string type measures only	The measure is aggregated by considering the values of all child cells. If all child cells have the same value, the aggregated value is the same as the child cells. Otherwise it is ambig.
popcount	For any measure type	The measure is aggregated by counting the number of child cells that are populated (have a value different to the naval for the measure).

There are also 'pop' (that is, "populated") versions of several aggregation types. These aggregate in the same manner as the aggregation type above, but only consider cells that are populated, which means that they have a value different to the naval for the measure. This may not necessarily mean a value that an end-user thinks of as being "populated".

These aggregation types are as follows:

- ambig_pop
ambig of all populated values
- average_pop
Average of populated values
- min_pop
Minimum of populated values

- max_pop
Maximum of populated values
- median_pop
Median of populated values
- total_pop
Total of populated values

Spreading

Introduction

By definition, an OLAP type model has hierarchical relationships between positions in hierarchies. Measures are calculated in dimensions above the base intersection by aggregation by using the parent-child relationships between the positions. RPAS allows such measures to be manipulated not only at the bottom levels, but also at aggregated levels. In order to preserve the integrity of the data with such a change, RPAS needs to change the underlying data values at the base intersection for the measure, so that when they are aggregated again, they result in the changed value at the aggregated level. The method of changing the base intersection values to achieve this is known as spreading.

Spreading always applies to cells at the base intersection of the measure. At all aggregated levels above the base intersection, the effect of any change is applied by considering all cells at the base intersection that are descended from the changed cell (for instance, children and grandchildren). These calls are described as 'child cells' in this description. Spreading does not operate from level to level to level down a hierarchical roll-up, which would not only be less efficient, but would also generate different (and generally less acceptable) results when there are changes or locks at levels between the change being spread and the base intersection.

The RPAS engine allows changes to be made to a measure for positions at multiple levels, and the effect of all such changes are actioned in a single calculation step. The basic technique for managing this spreading is the same for all spreading methods, and it is described in "Multi-Level Spreading"

For calculation purposes, a lock to a cell for a spreadable measure is treated as a change to that cell that re-imposes the previous value. If none of the child cells of the locked cell have changed, the lock has no effect, and all child cell values remain unchanged.

Locks and Spreading around Locked and Changed Cells

Other than in the special case where there are no cells that are free to be changed, spreading only affects cells that are free to be changed. All child cells are free to be changed except for those that are elapsed (see Chapter 8), locked by the user, explicitly changed by the user, or that have already been recalculated as the result of spreading another (lower level) change. Spreading always attempts to spread around locked or changed cells without changing their values. Where none of the child cells are free to be changed, spreading applies to all child cells that are not elapsed by using the changed or recalculated values as the base values to spread upon. For spreading purposes, when something has to give, elapsed cells are considered to be 'more important' than locked or changed cells.

Locked cells for recalc type measures are treated in an analogous manner: the mapping expression (see [The Spreading of Recalc Type Measures](#)) is reapplied (using recalculated values of other measures on the right hand side of the mapping expression if necessary) to recalculate the mapped measure. It is then spread normally.

Note: The effect of spreading where there are no child cells free to be changed is that the result for some lower level locked or changed cells will be different to the locked value or the change made. Effectively, higher level locks or changes are deemed to be 'more important' than lower level ones. Causing the circumstance where there are no free child cells can be a very useful technique when initializing data. For example, in a single calculation, a 'shape' can be applied to child cells, and then a 'total' to the parent cell. The result is that the parent total is spread across the children using the appropriate spreading technique, but according to the supplied shape. This is because the higher level change takes precedence.

Spreading Methods

Just as different types of measures require different aggregation techniques, different types of measures require different spreading techniques. Measures that cannot be aggregated (that is, are of "recalc" type) are not usually spread at all (see [The Spreading of Recalc Type Measures](#)), but they may employ the replicate spreading technique. The default spreading method for a measure is set up as part of the definition of the measure. This is the spreading technique that is used for all changes to the measure unless explicitly overridden on edit by the user.

The spreading methods that are supported by RPAS are listed here and described in the following sections:

- Proportional Spreading
- Replicate Spreading
- Even Spreading
- Delta Spreading
- PET and PST Spreading

Proportional Spreading

Proportional spreading is the most commonly used spreading technique once data has been initialized, and it is the default spreading method for most spreadable measures. In proportional spreading, all 'children' that are free to be changed are changed in the same proportion so that their existing ratios to each other are maintained, and the required value for the parent is achieved. If proportional spreading is used for a measure that is not initialized (that is, its children all have the "naïve"), the children are assumed to all have the same weight, so the effect of the spreading is the same as the even spreading method.

Example:

Starting values – ChildA 10, ChildB 20, ChildC 30, ChildD 40, Parent 100.

Changes – Parent changed to 145, ChildA changed to 20, ChildB locked.

Resulting values – ChildA 20, ChildB 20, ChildC 45, ChildD 60, Parent 145.

Spreading Process – ChildA and ChildB are not free to be changed by spreading because ChildA was explicitly changed and ChildB was locked. The required parent value of 145 must include 40 from ChildA and ChildB, and thus ChildC and ChildD must total 105. The previous total for ChildC and ChildD was 70, so their values must be changed by applying the multiplier of 105/70. Thus the new values for ChildC is 45, and for ChildD is 60. After aggregation, the result is:

- The parent has the value 145, as required
- ChildA has the required 20
- ChildB did not change
- The ratio of ChildC being 75% of ChildD is maintained

This spreading method is not legal for measures with a recalc aggregation type.

Replicate Spreading

Replicate spreading is sometimes used when initializing data, especially for recalc type measures, and for measures with aggregation type such as average, minimum, and maximum. It is unusual for it to be the default spreading method for any measure, but may be used by overriding the spread method on data entry. In replicate spreading, all child cells that are free to be changed are changed to the value of the parent cell. With replicate spreading, there is no guarantee that after aggregation the value of the parent cell will be the value that was replicated. In fact, it usually will not be. Replicate spreading should be considered to be an indirect way of entering the same value into multiple child cells.

Example:

Starting values – ChildA 10, ChildB 20, ChildC 30, ChildD 40, Parent 100.

Changes – Parent changed to 145, ChildA changed to 20, ChildB locked.

Resulting values – ChildA 20, ChildB 20, ChildC 145, ChildD 145, Parent 330.

Spreading Process – ChildA and ChildB are not free to be changed by spreading because ChildA was explicitly changed and ChildB was locked. The parent value of 145 is replicated to ChildC and ChildD. After aggregation, the result is that the parent has the value 330.

This spreading method is legal for measures with a recalc aggregation type.

Even Spreading

Even spreading is sometimes used when initializing data. It is unusual for it to be the default spreading method for any measure, but it may be used by overriding the spread method on data entry. In even spreading, all child cells that are free to be changed are changed to the same value, which is the total for the parent cell for the free child cells divided by the number of free child cells.

Example:

Starting values – ChildA 10, ChildB 20, ChildC 30, ChildD 40, Parent 100.

Changes – Parent changed to 145, ChildA changed to 20, ChildB locked.

Resulting values – ChildA 20, ChildB 20, ChildC 52.5, ChildD 52.5, Parent 145.

Spreading Process – ChildA and ChildB are not free to be changed by spreading because ChildA was explicitly changed and ChildB was locked. The required parent value of 145 must include 40 from ChildA and ChildB, and thus ChildC and ChildD must total 105. This is spread evenly, thus the new values for ChildC and ChildD are both 52.5. After aggregation, the result is:

- The parent has the value 145, as required
- ChildA has the required 20
- ChildB did not change
- The remainder has been spread to ChildC and ChildD evenly

This spreading method is not legal for measures with a recalc aggregation type.

Delta Spreading

Delta spreading is sometimes used when data is fully initialized. If it is used when the measure is not initialized, the effect will be the same as even spreading. It is unusual for it to be the default spreading method for any measure, but it may be used by overriding the spread method on data entry. In delta spreading, all child cells that are free to be changed are changed such that the delta to the parent cell is spread evenly across those child cells.

Example:

Starting values – ChildA 10, ChildB 20, ChildC 30, ChildD 40, Parent 100.

Changes – Parent changed to 145, ChildA changed to 20, ChildB locked.

Resulting values – ChildA 20, ChildB 20, ChildC 47.5, ChildD 57.5, Parent 145.

Spreading Process – ChildA and ChildB are not free to be changed by spreading because ChildA was explicitly changed and ChildB was locked. The required parent value of 145 must include 40 from ChildA and ChildB, and thus ChildC and ChildD must total 105. The previous total for ChildC and ChildD was 70, so the delta to the parent is 35. This delta is spread evenly across the children, so ChildC and ChildD are both increased by 17.5. Thus the new values for ChildC is 47.5, and for ChildD is 57.5. After aggregation, the result is:

- The parent has the value 145, as required
- ChildA has the required 20
- ChildB did not change
- The increase to the parent has been evenly divided between ChildC and ChildD

This spreading method is not legal for measures with a recalc aggregation type.

PET and PST Spreading

PET (period end total) and PST (period start total) are special spreading types to support measures with the PET or PST aggregation types where the values of cells represent “snapshots” at a period of time rather than a total of events. Opening and closing stock (inventory) are typical examples of such measures, where the value for a month will be the value for the first (opening stock) or last (closing stock) week in the month, but values up non-time hierarchies will be produced by total aggregation.

PET and PST measures require special spreading. We anticipate a future enhancement to support spreading changes to such measures at aggregated time positions by spreading the effect of the change across all children of the time period. At present, the PET and PST spread types change the first or last child only. At present, a change to closing stock for a month has exactly the same effect as a change to closing stock for the last week in the month.

Multi-Level Spreading

The RPAS engine allows changes to be made to a measure at multiple levels, all of which are dealt with in a single calculation. Because spreading requires parent-child relationships, and spreading is effected between the intersection that is changed and the base intersection for the measure, there is a requirement that all changes to be effected by a single calculation must fall on a single hierarchical roll-up. This is controlled by [Hierarchical Protection Processing](#), which is described in the next section.

When there are changes at multiple levels, the spreading process fundamentally works 'bottoms-up.' That means lower level changes are actioned before higher level changes. The spreading algorithm starts with the lowest level in the hierarchical roll-up that has changes, and it spreads each change at that level in turn.

The result of this process is that every child cell of a changed cell is no longer free to be changed. If it was previously free to be changed, it has now been recalculated by spreading. When all changes at a level have been actioned, the algorithm moves on to the next lowest level in the hierarchical roll-up that has changes, and it continues in this manner until all changes have been actioned. If a higher level change overlaps a lower level change, the lower level changes are unaffected because all child cells of the lower level change will not be free to be changed.

Example (using proportional spreading):

Starting values – jan 10, feb 15, mar 20, apr 25, may 30, jun 35, jul 40, aug 45, sep 50, oct 55, nov 60, dec 65. firsthalf 135, secondhalf 315, year 450.

Changes – year changed to 500, firsthalf changed to 150, jan changed to 15, feb changed to 20, mar locked, jul and aug changed to 50, sep locked.

Resulting values – jan 15, feb 20, mar 20, apr 26.39, may 31.67, jun 36.94, jul 50, aug 50, sep 50, oct 61.11, nov 66.67, dec 72.22. firsthalf 150, secondhalf 350, year 500.

Spreading process – The first change to be spread is the change to the first half to be 150. jan, feb and mar now total 55, so apr, may and jun must total 95. By proportional spreading the results are 26.39, 31.67 and 36.94. The second change to be spread is the 500 for the year. Only the months oct-dec are now free to be changed. The other months total 300, so oct-dec must total 200. By proportional spreading, the results are 61.11, 66.67, 72.22.

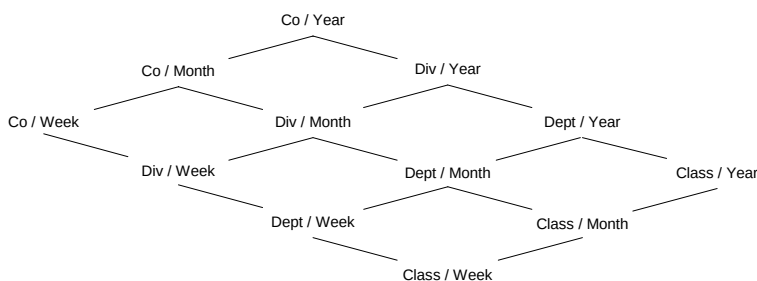
Hierarchical Protection Processing

Hierarchical Protection processing is a process that ensures that all changes made at aggregated levels fall on a single hierarchical roll-up, which is a prerequisite for the spreading process to function correctly. Hierarchical protection processing operates by protecting (preventing direct manipulation) cells for intersections for combinations of dimensions that cannot lie on a single hierarchical roll-up with the changes already made.

In theory, since hierarchical protection processing is necessary to ensure the integrity of the spreading process and each measure is individually spread, hierarchical protection processing could operate independently for each measure. Having the editable measures varying from intersection to intersection would probably cause considerable confusion to the users and would make implementing a consistent methodology difficult. For simplicity, hierarchical protection processing operates on all measures.

An OLAP type model has multiple hierarchies and spreading operates from the cell that has been changed to all child cells at the base intersection, so hierarchical protection processing must operate across multiple hierarchies. A single hierarchy may have multiple roll-ups, which are also considered. Whenever a change or a lock is made to an intersection for a new combination of dimensions, the calculation engine checks all other combinations of dimensions, and it protects those that cannot be on the same hierarchical roll-up as changes already made. It does this by considering a 'cross multiplication' of hierarchical roll-ups across all the hierarchies.

A simple example will clarify the process. Consider the matrix of 'cross multiplied' dimension combinations that result when there is a 2-dimensional product by time model with the dimensions Co/Div/Dept/Class and Year/Month/Week. This is shown below schematically with parent-child relationships:



Other than at the top of a hierarchical roll-up, each combination of dimensions has two parent combinations: one per hierarchy with the next highest dimension, so class/week has parents of class/month and department/week.

Note: For the sake of simplicity, the picture does display the roll-up of the "all" dimension (all products, all time periods).

All spreading is from the changed level to the base intersection (class/week in this example). Consider a change at an intersection of Div/Month. We know that the spreading hierarchical roll-up must be a path from the top (Co/Year) to the bottom (Class/Week) that passes through Div/Month. There are six such paths, none of which go through the combinations Co/Week, Dept/Year or Class/Year, so those combinations of dimensions are all protected. If the next change is at an intersection of Class/Month, then Div/Week and Dept/Week are similarly protected.

Note: Hierarchical protection processing always reflects the current set of locks and changes.

RPAS allows cells that have been changed or locked to be unchanged or unlocked before the calculation is initiated. If an unchange or unlock removes the last change or lock for a combination of dimensions, some other combinations of dimensions that were previously protected could become unprotected. In our example above, if the first change to a Div/Month were now unchanged so that the only change outstanding is at Class/Month, then Dept/Year and Class/Year would now become editable again, but Co/Week, Div/Week and Dept/Week would still be protected.

Note: Non-conforming measures (see [Non-Conforming Expressions](#)) may lead to hierarchical protection processing that may appear to be over protective.

When considering hierarchical protection processing, all measures have their “scope” expanded to include the “All” level of all hierarchies that they are not dimensioned on. For example, the implications of this are that a change to a measure with a base intersection of “class,” which is interpreted as meaning “class/all,” would prevent the manipulation of a measure with a base intersection of ‘year,’ which is interpreted as ‘all/year.’

The Spreading of Recalc Type Measures

Measures that are of recalc type are not usually spread by any spreading technique. Spreading techniques typically rely on the existing relationships between a parent and its children in the spreading process. In a recalc measure, those relationships cannot be relied upon because they are not ‘weighted.’ Spreading of changes to a recalc measure is therefore indirect, and applying a ‘mapping rule’ effects the change.

A mapping rule is a rule (with two or more expressions) that calculate a spreadable measure from the changed value of a recalc measure and other measures. The selected expression for the rule is evaluated at the level of the change to the recalc measure. It results in a changed value of a spreadable measure, and if this is above the base intersection for that measure, it is spread normally using its default spreading method. Therefore, a change to a recalc measure should be considered to be an indirect change to the spreadable measure that it is mapped to.

The only constraint on the manipulability of a normally spreadable measure is through protection processing, which prevents the manipulation of measures that will be calculated. For a recalc measure, the measure must have a mapping rule. Without a mapping rule, the measure cannot be manipulated.

Note: A recalc measures can only appear in a single rule in a rule group. The RPAS calculation engine therefore knows that rule contains the recalc expression for the recalc measure. If there are other expressions in the rule, they may be used as mapping expressions, which allows the recalc measure to be editable. If there is just a single expression in the rule that calculates the recalc measures, the recalc measure is non-editable through normal protection processing.

Recalc measures must not be inserted into a workbook because recalc measures require recalc expressions. If such an expression is active, the measure would already be there.

Expressions, Rules, and Rule Groups

Introduction

Measures are related together through algorithmic relationships. For example, the sales value may be the sales units multiplied by the selling price. In RPAS, these relationships are specified through expressions, which are grouped for usage into rules and rule groups.

It is a fundamental principle in RPAS that the calculation engine maintains and guarantees the integrity of all the active relationships between cells at all times. Hierarchical relationships are maintained through the processes of spreading (see [Spreading](#)) and aggregation (see [Aggregation](#)). Relationships between measures are maintained by the evaluation of expressions. One of the great strengths of RPAS is that both of these types of relationships are automatically maintained in a non-procedural manner. You do not have to write code to determine what is calculated, how it is calculated, or in what sequence it is calculated. All that is required is the definition of the relationships themselves, although you can provide prioritization information to guide the calculation engine when there is a choice of calculation paths.

In an RPAS model, all cell values at aggregated level can be determined by aggregation from the cells at the base intersection. Although the description that follows is a simplification, a basic understanding of the working of the calculation process, and the importance of expressions, can be gained by understanding the interconnection between three fundamental processes: spreading, 'bottom level' expression evaluation, and aggregation. A more detailed and precise description is given in [The Calculation Cycle](#). Changes to measures at aggregated levels are spread down to their base intersections. Here, the calculation engine enforces all measure relationships that are no longer guaranteed to be true by evaluating an expression. This is because the cell values of one or more of the measures in the relationship have changed directly, through spreading, or by prior evaluation of an expression. When base intersection calculation is complete, all measures at the base intersection that have been changed are aggregated to re-impose cell integrity.

Expressions

Expressions are the basis of all calculations of the relationships between measures. They are evaluated by the calculation engine during a calculation. Expressions are written in a syntax that allows for the calculation of one or more measures from other measures, constants and parameters, using standard arithmetical functions and a rich set of mathematical, technical, and business functions. Expressions are therefore an algorithmic statement of a relationship between measures. Details of the allowable syntax for expressions are given in a separate document.

Rules

An expression describes the relationship between measures in a way that causes a measure to be calculated through the expression. An expression may be said to 'solve' the relationship for the measure that is calculated through the expression. In some cases, there may be business methodology reasons for wanting more than one of the measures in a relationship to be calculable or solvable through that relationship.

To support this requirement, RPAS has the concept of a rule, which consists of one or more expressions that describe the same relationship between measures, but that solve for different measures. All of the expressions in a rule should use the same measures, and must have a different target measure. The target measure is the measure on the left-hand side (LHS) of the expression that is calculated by the expression.

Where a rule has multiple expressions, those expressions are given a priority sequence to help the calculation engine select a calculation path that follows business priorities. Consider the rule that relates together sales value, sales units, and sales price. Let us assume that there are three expressions in this rule. Each of the measures involved in the rule may be 'solved' through the rule. For instance, if you make a change to sales value, it should be clear that the calculation engine could enforce the mutual integrity of all the cells by holding the sales price constant and recalculating a new sales units. It could also achieve the same end by keeping the sales units constant and recalculating the sales price. Both approaches are mathematically valid, and produce a consistent result with complete data integrity. However, it is likely that one approach makes more 'business sense' than the other. In this case, most businesses in most circumstances would want the price to remain constant and have the units recalculated. The prioritization of the expressions in the rule provides this information to the calculation engine. Considerable care should be taken in the design of models to ensure that appropriate expression priorities are established.

When given a choice, the calculation engine will always select the highest priority expression in the rule that is available to be selected. In this example, the expression that calculates sales units would have a higher priority than the expression that calculates sales price. Similar consideration of the desired effect of a change to sales units will probably lead to a conclusion that the expression that calculates sales value would also have a higher priority than the expression that calculates sales price.

What of the relative priority of the expressions to calculate sales value and sales units, and the 'business priority' for those expressions? That may vary from implementation to implementation. It may even vary from one type of plan to another in the same implementation. For a financial merchandise plan, the preferred behavior may be that a change to the sales price only causes a recalculation of the sales units, whereas in a unit-oriented lower level plan, the preferred behavior may be that a change to the sales price causes a recalculation of sales value.

The same measure may appear in multiple rules. This will often be necessary because the same measure can be involved in many different relationships with other measures. For example, there may be a relationship between sales value, sales units, and sales price. Sales value may also be involved in another relationship with closing stock and a cover value, and yet another with opening stock, receipts, markdowns and closing stock.

Rule Groups

It is most unusual for a model to only require a single rule. In most cases, there will be a collection of relationships between measures that must be maintained. In RPAS, a Rule Group is a collection of rules that are treated as a unit by the calculation engine with the integrity of all the rules in the rule group being maintained together. The calculation engine always has one (and only one) active rule group. Even if all that is required is a single expression, that single expression will be in a rule, and that single rule will be in a rule group. The process by which the integrity of all the rules in a rule group is maintained is quite complex. It is described in detail on [The Calculation Cycle](#).

Rules within a rule group are given a priority. The calculation engine uses this to select a calculation path that follows business priorities by using rule priorities to determine which rule to enforce when there is a choice to be made. This is described in more detail in [The Calculation Cycle](#).

There may be many rules defined within a system as a whole. The validation of rules is performed in isolation, but rules within a rule group are also validated in the context of all the other rules in the rule group. This can mean that a rule that is perfectly valid syntactically, but it is not valid within a particular rule group. Rule group validations include:

- Each rule in a rule group must represent a completely different measure relationship. Therefore no two rules in a rule group may use exactly the same collection of measures, and neither may one rule group use a collection of measures that is a subset of the collection of measures in another rule.
- There must be an expression that calculates each recalc measure.
- Any measure that is on the LHS of the only expression in a rule may not be on the LHS of any other expression.

Although there may only be one active rule group at any time, RPAS allows for the definition of multiple rule groups to satisfy different calculation requirements. Rule groups may be one of four different types:

- **load** – The RPAS application automatically uses the load rule group when loading data into the workbook.
- **calculate** – RPAS supports multiple calculation rule groups. Menu options may be configured to allow the user to select a different calculation rule group. RPAS ensures a smooth transition from one calc rule group to another.
- **refresh** – The RPAS application automatically uses the refresh rule group to refresh data.
- **commit** – The RPAS application automatically uses the commit rule group when committing data to the domain.

These rule groups are perfectly ‘normal,’ so although they will typically include many rules that use the master modifier to load or commit data, they may also have other rules. For example, it is perfectly possible to commit data to the domain for a measure that does not exist in the workbook merely by including the appropriate rule to calculate the measure (with the master modifier) in the commit rule group. Similarly, a measure may be loaded into a workbook that does not exist in the domain by including an appropriate rule to calculate the measure in the load rule group.

Rule Group Transitions

Although only a single rule group may be active at any time, RPAS supports the transition from one rule group to another. The calculation engine ensures the integrity of measure relationships at all times so this process is not merely a case of switching from one rule group to another. There is no guarantee that the integrity of the rules in the rule group being transitioned has been maintained.

RPAS makes a worst case assumption when transitioning rule groups. Any rule that is in both the old and new rule groups is assumed to have its integrity maintained. Any other rule is assumed to be potentially wrong, and so is flagged as “affected.” A normal calculation is then initiated. Expressions to be evaluated are determined by the usual process (see [The Calculation Cycle](#)). All affected rules will therefore have their integrity imposed by the evaluation of an expression, and ‘knock-on’ effects may cause some rules that occur in both the old and new rule groups to also be evaluated. Since all base intersections must be calculated during rule group transition, a large or complex rule group transition is likely to take longer than a normal calculate.

There are circumstances when automatic rule group transitions occur:

- **On data loading**
Data is loaded using the load rule group. This will typically load measures by calculating them from the data values held on the domain using the master modifier, but it may also calculate other measures that are not explicitly loaded. When the load is complete, the system will automatically transition to the calculate rule group.
- **On data refreshing**
Data refreshing causes some measures to be updated from values held on the domain. Refreshing uses the refresh rule group, but there is no real transition. The measures that are affected by the refreshed measures are treated as affected in the calculate rule group, and a normal calculate of that rule group follows. Effectively, data refreshing causes a calculation by using the calculate rule group as if the cells that were refreshed were directly changed by the user.
- **On data committing**
There is a normal transition from the current calculate rule group to the commit rule group. This will typically commit measures by calculating them on the domain by using the master modifier. When transitioning back from the commit rule group to the calculate rule group, there is an assumption that only measures with a “master” modifier have changed and therefore no transition is required.

The Calculation Cycle

Introduction

The calculation cycle always uses the current active rule group. It is a comprehensive process that uses non-procedural hierarchical cell relationships and expression-driven measure relationships from the rule group. These relationships are used together with details of the locks and changes to individual cells to determine and then execute the required actions to apply the effect of the changes and locks. This section describes how the calculation engine determines what to calculate, how to calculate it, and in what order to perform the calculations. Refer to the *RPAS User Guide* and the *RPAS Administration Guide* for details of processes; such as spreading, aggregation, and the evaluation of expressions.

There are four distinct stages of the calculation cycle.

1. In the first stage, protection processing occurs while the user is making changes to cell values, and it protects those measures that the user cannot change either because they are never changeable or because changes already made force them to be calculated.
2. In the second stage, the engine decides what expressions will be evaluated
3. In the third stage, the sequence of calculation is determined.
4. The final stage is the physical process of doing the calculation.

Note: The calculation cycle can operate in one of two modes: “full” and “incremental.” In “full” mode, it is assumed that all of the cells for the measures being evaluated need to be calculated. This mode is used when calculating in batch, and in all rule group transitions. “Incremental” mode is used when manipulating cells in an online session, and only those cells that are directly or indirectly affected by user edits are calculated.

Protection Processing

Overview

Other than in exceptional circumstances, the calculation engine guarantees the integrity of all relationships and ensures that the value for a cell changed by a user after calculation is the value entered by the user. In order to ensure this, the calculation engine must prevent the user from making changes to any cells where it would be unable to guarantee that integrity. The process that achieves this is called protection processing.

A measure may only be manipulated when the calculation engine is able to change other cells by spreading and/or evaluation of an expression to enforce the integrity of relationships. A measure that is not used in any rules may only be manipulated if it has a spreading technique other than recalc.

It is a basic principle of the calculation engine that a measure that is changed (or locked) cannot also be recalculated by evaluating an expression. It will be aggregated, which in the case of a recalc measure, does involve the evaluation of an expression. A measure that is to be evaluated can only be evaluated using one expression because there is no guarantee that the same result would be produced from two expressions that represent different measure relationships. It is also a basic principle that any measure relationship (rule) must be evaluated when one or more of the measures in that relationship have been changed because this is the only way to enforce the integrity of the rule relationship. Therefore, a rule where there is just a single expression means that the measure calculated by that expression cannot be changed by the user because there is no expression to evaluate to effect that change for that measure relationship. Such measures can never be manipulated in any rule group that uses the rule and are protected.

Where a rule has two expressions, the two measures that are calculated by those expressions are available to be manipulated. However, as soon as one measure is manipulated by the user, we know that the expression that calculates the other measure must be evaluated, as one of the expressions in the rule has to be evaluated, and we cannot evaluate the expression that calculates the measure that was changed. The expression that must be calculated is said to be *forced*, and the measure that it calculates is protected to prevent the user from changing it. That measure may be involved in more than one rule, and in the other rules in which it is used it must be treated as if the user changed it. This so-called *knock-on* effect may force further measures to be forced and protected. Evaluating these effects is the basic technique of protection processing.

Protection processing occurs continuously while the user is editing cells. Each time the 'changed state' of a measure changes, protection processing evaluates the measures that should now be protected. The 'changed state' of a measure means the measure goes from not having changes or locks to having them. Protection processing always reflects the current set of locks and changes. RPAS allows cells that have been changed or locked to be unchanged or unlocked before the calculation is initiated. If an unchange or unlock removes the last change or lock for the measure so that the measure is no longer affected, protection processing is quite likely to find the other measures that were previously forced, but are no longer forced. These measures are free to be manipulated, so they must be unprotected.

Protection Processing Details

The following terms are used in this description:

- An affected measure is a measure that has been changed by the user, is locked by the user, or is forced.
- An affected rule is a rule that contains one or more affected measures.
- A free measure is a measure that is not affected.
- A free expression is an expression for an affected rule that calculates a free measure.
- A forced rule is an affected rule that has only one free expression.
- A forced measure is the measure calculated by the free expression in a forced rule.

Any measure that is the measure on the LHS of the only expression in a rule is protected.

Protection processing considers each affected rule in turn. Each affected rule will be in one of three conditions:

- Affected rules that have previously been forced are ignored
- If the affected rule has two or more free expressions, it is ignored because nothing is forced.
- If the affected rule has just a single free expression, it becomes a forced rule, and the measure calculated by the free expression is forced and becomes an affected measure. The forced measure is protected. All rules that use the forced measure become affected.

When a new measure becomes forced, checking of affected rules begins again. When all affected rules have been considered without any further measures becoming forced, the first stage of protection processing is complete.

The second stage of protection processing is to perform 'look ahead' protection processing. Look ahead protection processing ensures that all measures that are visible in windows (and still unprotected) can be manipulated. It does this by performing the protection processing that would occur if the measure were changed. This includes ensuring that there is a solution to the processes of determining what to calculate and ordering the calculation. If these processes fail to find a solution, the process that determines what to calculate will repeatedly back up the decision tree and select a different expression that is looking for a solution. If there is no such solution, the measure that was being checked is protected. In this manner, the calculation engine ensures that there will always be a method to calculate the effects of all changes that it allows the user to make.

Note: This is a somewhat simplified description of protection processing, as it ignores the implications of "cycle groups" (see [Cycle Groups](#)) and "synchronized measures" (see [Synchronized Measures](#)).

Determining What to Calculate

The protection processing process has established which measures are forced given the current set of changes and locks. When a calculate command is issued, those forced measures will be calculated (using the forced expressions). However, there may be affected rules that are not forced. For those affected rules, we know that an expression must be evaluated, and the calculation engine must select one of the expressions. Otherwise the integrity of the rule is compromised.

When there are one or more affected rules that are not forced, the highest priority affected rule is selected. From this selected rule, the highest priority free expression is selected, and it will be evaluated. These are the only uses to which the rule and expression priorities are put. The measure that is calculated by the selected expression is then treated as forced, and *knock-on* effects considered, which are likely to cause other rules and measures to become forced. At the end of this process, if there are still affected rules that are not forced, the process is repeated until there are no affected rules that are not forced. At this point, any rule that is not affected does not need to be evaluated, and an expression has been forced or selected for all rules that need to be evaluated to ensure the integrity of all measures.

Determining the Calculation Sequence

The previous section has established which expressions to evaluate, but not the sequence in which they are evaluated. The sequence of evaluation of expressions is driven by the status of the right-hand side (RHS) measures. All normally spreadable (not of recalc type) measures that are changed can be spread and then aggregated at the start of the calculation cycle. Normally, spreadable measures that have been changed and those measures that will not change during the calculate are considered to be 'complete.' Any expression whose RHS measures are all complete may be evaluated. If the expression is a mapping rule for a recalc measure, the changed values for the mapped spreadable measure will be calculated for all changed cells. That measure may then be spread and aggregated normally. If the expression is for normal 'base intersection' evaluation, the measure will be calculated, and may then be aggregated. In either case, the calculated measure is now 'complete,' which may make further expressions available to be evaluated. The process continues until all expressions have been sequenced.

When determining the sequence of calculation, the evaluation of expressions is intermingled with spreading and aggregation. In very trivial cases, where all changed measures are spreadable, there will be:

- a phase where a number of measures are spread
- a second phase where a number of measures are calculated at the base intersection
- a third phase where a number of measures are aggregated.

However, if any recalc measures have been changed at aggregated levels, the 'mapping rule' cannot be applied until any affected measures on the RHS of the expression have been spread or calculated and then aggregated.

Note: This is a simplified description of the calculation sequence. For efficiency purposes; groups of measures that must be spread, aggregated, or evaluated are batched together, so that an individual measure is not necessarily spread, aggregated, or evaluated as soon as it is available for that action. However, it is always spread, aggregated, or evaluated before the results of that action are required for another step. Also, expressions are not evaluated for all cells, but only for those cells where one or more of the measures on the RHS of the expression have changed. There are similar efficiencies in aggregation to avoid the redundant re-aggregation of cells that will not have changed.

Cycle Groups

Introduction

This section describes the cycle group feature of the RPAS calculation engine. This feature enables relationships between measures that have cyclic dependencies from the measure perspective (there appears to be a 'deadly embrace' where each measure depends upon the other), but are actually acyclic when the time dimension of these measures is considered. Without this feature, such relationships could not be set up because the calculation engine would be unable to find a calculation sequence that enabled both measures to be calculated.

A common application of cycle groups can be found in inventory calculations that involve measures, such as beginning of period (BOP) and end of period (EOP). It is typical that EOP is calculated in some way from BOP for the same period. Other than in the very first period, the BOP of a period is equal to the EOP of the previous period. Since BOP is dependent on EOP and EOP is dependent on BOP, a cycle exists from a measure perspective. However, when the time dimension is considered, calculations can be performed in an acyclic fashion. In this example, if EOP for the first period is calculated first, then BOP for the second period can be calculated. This allows EOP for the second period to be calculated, and so on.

Cycle Breaking Functions

Some of the functions supported by the calculation engine have special cycle breaking logic associated with them. These include functions that reference previous time periods and functions that reference future time periods. When these functions are used, the calculation engine automatically determines when measure dependencies that appear to be cyclic are in fact acyclic when the calculations are performed one period at a time. The lag and lead functions are examples of cycle breaking functions.

Cycle Group Evaluation

A cycle group is a group of expressions that the calculation engine must calculate together in order to avoid cyclic dependencies. If the apparent cycle is broken by a function that looks backwards in the time dimension (such as lag), the calculation continues with the first time period of each expression in sequence. This is followed by the second time period of each expression in sequence, and it continues until all time periods have been calculated. If the apparent cycle is broken by a function that looks forwards in the time dimension (such as lead), calculation proceeds in reverse order starting with the last time period.

Note: Since the acyclic calculation of expressions in a cycle group is a "base level calculation," all measures being calculated in the cycle group must share the same base intersection. That is, the cycle group evaluation process cannot aggregate measures calculated in the cycle group during the cycle group evaluation

Cycle Group Example

Consider the following measures:

BOP: beginning of period inventory

EOP: end of period inventory

OS: opening stock (that is, the opening inventory for the first period in the plan horizon)

SLS: sales

RCP: receipts

And consider the following rules:

R1: $BOP = \text{if}(\text{current} == \text{first}, OS, \text{lag}(EOP))$

R2: $EOP = BOP - SLS + RCP$

$RCP = EOP - BOP + SLS$

When the measure RCP is edited, R2 is affected and the EOP expression in this rule is forced. Then rule R1 is affected and the BOP expression in this rule is forced.

Since the calculation of EOP requires BOP and the calculation of BOP requires EOP, a cycle is detected that contains both of the selected expressions. This is a valid cycle group because the calculation of BOP is dependent on the lag of EOP. Therefore, the cycle can be broken and the intra-cycle ordering results in the BOP expression being evaluated first and EOP expression second.

The evaluation of this cycle group involves the calculation of the first time period of BOP, followed by the first time period of EOP, followed by the second time period of BOP, followed by the second time period of EOP, and continues until all time periods have been calculated.

Synchronized Measures

Overview

Measure synchronization is an RPAS user interface and calculation engine feature. It enables measures that are very closely related to be represented in the user interface (UI) in a more intuitive manner. It can give the appearance of a cell edit or lock affecting two different measures. From a calculation perspective, the cell edit or lock is only applied to one of these measures. A common application of synchronized measures is to allow BOP and EOP to be synchronized. From a business logic perspective, the BOP in one period and the EOP in the previous period are the same thing, and measure synchronization means that even before calculation, an edit or lock of BOP in one period also appears on the UI as an edit or lock of EOP for the previous period, and vice versa.

To accomplish measure synchronization, a measure is defined with a synchronized view type and a list of synchronized source measures. The measure defined with these attributes is called the synchronized target measure. Synchronized target measures may be edited, but any such edits are actually treated as edits to the underlying synchronized source measures. Protection processing is performed on the synchronized source measures. The protection state of the target measure is then derived from that of the source measures. An edit to one of the source measures is also reflected in the display of the target measure.

The synchronized view types that can be used to define synchronized target measures are as follows:

- **sync_first_lag** – The first period of the target measure is synchronized with the first source measure, and periods 2..N of the target measure are synchronized with periods 1..N-1 of the second source measure, where N represents the last period. The first source measure will not have a time dimension. This view type is particularly useful for defining BOP target measures. Here the first source measure would be an opening inventory, and the second source measure would be the EOP.
- **sync_lead_last** – Periods 1..N-1 of the target measure are synchronized with periods 2..N of the first source measure and period N of the target measure is synchronized with the second source measure, where N represents the last period. The second source measure will not have a time dimension. This view type is particularly useful for defining EOP target measures. Here the first source measure would be BOP, and the second source measure would be a closing inventory.
- **sync_first** – The target measure is synchronized with the first period of source measure. The target measure will not have a time dimension. This view type is particularly useful when defining OS target measures.
- **sync_last** – The target measure is synchronized with last period of the source measure. The target measure will not have a time dimension. This view type is particularly useful when defining CS target measures.

Note: In order for a synchronised measure to be editable, all of the measures that it is synchronized with must be viewable on the worksheet, but they do not need to be visible.

Synchronized Inventory Examples

Consider the following measures:

BOP: beginning of period inventory

EOP: end of period inventory

OS: opening stock (that is, the opening inventory for the first period in the plan horizon)

CS: closing stock (that is, the closing inventory for the last period in the plan horizon)

SLS: sales

RCP: receipts

And consider the following rules:

R1: $BOP = \text{if}(\text{current} == \text{first}, OS, \text{lag}(EOP))$

R2: $EOP = BOP - SLS + RCP$

$RCP = EOP - BOP + SLS$

BOP can be defined as a synchronized measure constructed from the OS and EOP measures with the *sync_first_lag* type. Only one expression in the rule group may have BOP on the LHS. This expression is used to construct views of BOP, and it is merged with expressions that require BOP on the RHS.

When edits or locks are made to BOP, it is the underlying values of OS or EOP that are actually changed or locked. Thus, even though rule R1 has only one expression and this expression calculates BOP, the BOP measure is not protected by protection processing because of the measure synchronization. The BOP measure is only protected when the underlying OS or EOP measures are protected, so the first period is protected when OS is protected and the remaining periods are protected when EOP is protected.

In this example, a CS measure is not required for calculation purposes, but it may be desired for viewing and editing purposes. For example, a window that contains only OS and CS but not BOP nor EOP may be wanted. In this case, the CS measure should be defined as a synchronized measure with type *sync_last* and the synchronized source measure would be EOP. As a result, an edit to CS becomes an edit to the last period of EOP.

Elapsed Periods

Many RPAS models will cover a time horizon where some of the time periods are in the past, and other periods are in the future. There is special logic for handling such models. In particular, the manner in which time periods that are in the past (referred to as elapsed periods) affects spreading and the manipulability of measures. The bottom level time period that is the last elapsed period is set through a rule.

RPAS assumes that time periods that are elapsed contain “actuals,” and that these actuals should not be editable. Therefore, all measures are non-editable during elapsed periods. For positions at aggregated levels in a time hierarchy, the position is only elapsed when the last bottom level time period descended from it is elapsed.

Measures that represent BOP data have special handling. From a business perspective, the BOP in a period is the same as the EOP in the previous period. Therefore, when an EOP value is elapsed, the following BOP value must also be elapsed. In RPAS, all measures with a default spread method of PST (see [Spreading Methods](#)) are assumed to be “BOP type” measures, and are protected for all elapsed periods, and for the first non-elapased period. There is also special handling of these measures for aggregated time positions. These are treated as elapsed, and are therefore protected when the first bottom level time period descended from it is elapsed.

Elapsed Periods and Spreading

You may make changes to measures at high levels of aggregation whose scope covers elapsed periods. When such changes are spread, values for elapsed periods are never changed. They may be considered to be of higher priority than normal cell locks.

Non-Conforming Expressions

Introduction

One of the strengths of the RPAS calculation engine is that a workbook may contain measures with different 'scopes.' The size and shape of the 'multidimensional cube' of data may vary by measure. Any two given measures in a workbook may have scopes that align exactly (for example, both measures have a base intersection of SKU/Store/Week), or where one is a subset of the other (for example, one has a base intersection of SKU/Store/Week and the other is at Class/Week). There can also be circumstances where each measure includes a hierarchy in its base intersection that the other dimension does not use (for example, one has a base intersection of Class/Week and the other is Store/Week). In extreme circumstances, the scopes of two measures may have no point of overlap at all (for example, one has a base intersection of Class and the other Store).

It is the scope of the measure on the LHS of an expression that determines the cells that must be calculated by the expression, even though that scope may be changed by the use of a modifier such as level. Where one or more measures on the RHS of an expression have a scope that is different (in any way) to the scope of the LHS measure, the expression is deemed to be 'non-conforming.' There is special logic to handle the calculation of non-conforming expressions, which depends on the type of nonconformity.

Although not explicitly declared, there is a single logical "All" position at the top of every hierarchy. When considering non-conformity, any measure that is not explicitly dimensioned on a hierarchy is implicitly assumed to be dimensioned on the "All" dimension of that hierarchy, so all data values are assumed to be for the "All" position. This concept is the key to understanding the handling of non-conforming expressions.

Handling of Non-conforming Expressions

When the concept of the "All" position is understood, all expressions can be considered to contain measures that use exactly the same hierarchies. The only potential differences between them are the 'bottom levels' (dimensions in the base intersection). Thus for handling non-conformity, only three cases need to be considered, for each hierarchy:

- **RHS same**
In this case the RHS measure has the same bottom level as the LHS measure. The RHS measure is "conforming" for that hierarchy, and values for the RHS measure are taken from the same position as the position being calculated for the LHS measure.
- **RHS higher**
In this case the RHS measure has a higher bottom level than the LHS measure. The RHS measure is "non-conforming" for that hierarchy. The values for the RHS measure for the position being calculated are assumed to be the same as the value of the RHS measure for the position in its bottom dimension that is the parent (ancestor) of the position being calculated. Effectively, it can be considered that the value of the measure has been 'replicated' down the hierarchy to the required level.
- **RHS lower**
In this case the RHS measure has a lower bottom level than the LHS measure. The RHS measure is "non-conforming" for that hierarchy, but because the scope of the RHS measure includes the bottom level for the LHS measure, values for the RHS measure are taken from the same position as the position being calculated for the LHS measure.

The conceptual case where the measures have scopes that do not overlap, because they have base intersections in a hierarchy that are for dimensions that are up different “branches” of the hierarchy, fails rule validation.

Examples

These examples all use the simple expression $a = b + c$

Example 1

- a has a base intersection of SKU/Store/Week
- b has a base intersection of SKU/Week
- c has a base intersection of SKU/Region/Week

For each SKU/Store/Week, a is calculated from the value of b at SKU/Week (that is, it is assumed that the value of b is the same for all positions in the location hierarchy) and the value of c at SKU/Region/Week, for the Region the Store belongs in. If ‘replication’ of c from the Region level is not appropriate, the rule writer can simulate other ‘spreading’ techniques by the use of functions and modifiers such as **count** and **level**. For example, the **count** function may be used to determine the number of Stores in the Region, and so dividing the measure c by that count will simulate ‘even’ spreading.

Example 2

- a has a base intersection of SKU/Store/Week
- b has a base intersection of SKU/Week
- c has a base intersection of SKU

For each SKU/Store/Week, a is calculated from the value of b at SKU/Week (that is, it is assumed that the value of b is the same for all positions in the location hierarchy) and the value of c at SKU (that is, it is assumed that the value of b is the same for all positions in the location hierarchy and time hierarchy). Note that an alternative approach, if required, would be to use a **level** modifier on the measure a, so that it is calculated at, say, SKU/Week, and then spread down to SKU/Store/Week, using the existing store participations to the measure a.

Example 3

- a has a base intersection of SKU/Week
- b has a base intersection of SKU/Store/Week
- c has a base intersection of SKU/Region/Week

For each SKU/Week, a is calculated from the value of b and c at SKU/“All”/Week.

Appendix: Rules Function Reference Guide

Overview

Introduction

The purpose of this section is to provide the syntax and design of functions, procedures, and modifiers that are used in expressions in the RPAS calculation engine.

The following terminology will be used throughout this document:

Note: There are important distinctions between each of these definitions.

Functions (single result) – Mechanisms for performing operations within an expression that are controlled and executed by the calculation engine.

- Functions are most commonly used in RPAS
- Most functions in base RPAS return only a single measure
- Calculation engine controls and executes the evaluation of a function
- Functions may be used in expressions with other functions and keywords

Functions (multiple result) – Similar to the features and behavior of single result functions, but they have semantic and syntactic differences.

- There can be more than one left-hand side (result) measure that can be specified implicitly by position in the expression or explicitly by label
- Left-hand side measures have to be at same intersection; however, the calendar hierarchy can be dropped or added
- The result(s) from a multi-result function cannot be used as arguments to another function, nor can the result(s) be chained with other operations to form long expression
- Expressions can be used as arguments to multi-result functions
- Multi-result functions cannot be part of a cycle group

Procedures – Mechanisms for performing operations in an expression where the calculation engine controls the execution, which is performed by the procedure itself.

- Procedures can only use measures or scalars
- Procedure executes the evaluation (instead of RPAS/calculation engine), but the calculation engine still controls protection processing, sequence of calculation, when the procedure is called, and so on.
- Can have multiple arguments on the left and right hand sides
- Cannot be used with functions, other procedures, keywords, and certain modifiers
- Because of their flexibility and the control that the developer has, procedures can be used for a wide variety of special calculations and activities
- Require a different syntax: "<-" instead of "="

Modifiers – Used to directly modify the source or destination of measures, to override the level, aggregation type, position, and so on.

- Syntax is <measure>.<modifier>

Keywords – Used standalone in expressions or as arguments inside functions to return specific data values.

Syntactical Approach

The syntax is as quick and straightforward to implement as possible. Function names, keywords, and so on are currently in lowercase.

Keywords are allowed, but they are kept to a minimum. Function parameters are comma separated and may be optional; but they are positional, so that the absence of a parameter needs to be specified by commas if a subsequent parameter is supplied.

The following is the functional syntax used in this document:

- Large square brackets [] are used to indicate an optional parameter.
- Small square brackets [] are part of the expression syntax and are used to specify a hierarchy, dimension, and/or position.
- Large braces { } indicate a choice where one of the items (which will be separated by a pipe sign |) must be selected.
- Small braces {} are part of the expression syntax, and are used to specify a measure set for functions that accept a variable number of arguments (that is, {<measureset>}).
- Parameters of a specific type (such as expressions or dimension names) are shown in angle brackets <>.
- A plus “+” sign is used to specify an intersection, which is done by connecting two or more dimension specifications.
- Keywords, modifiers, function names, and procedure names are shown in **bold**.

Specification of Hierarchy, Dimension, or Position

Many functions in RPAS require the specification of a hierarchy, dimension, or a combination thereof, to define the level at which an expression is evaluated. When defining the hierarchy and dimension names in expressions square brackets [] must be used.

In the document, the following syntax will be used to designate a hierarchy and dimension:

Note: That position is noted as optional because it can only be specified in a limited number of functions:

[<hierarchy>].[<dimension>].[[<position>]]

For simplicity of parsing and clarity of rule writing, the <hierarchy> must be supplied in all cases, even when, as in calendar index functions, it might be implied from the context. Functions that require a hierarchy and dimension specification will have standard validation rules whereby [<hierarchy>] must be a valid hierarchy name and [<dimension>] must be a valid dimension in [<hierarchy>]; in some functions or procedures one of the hierarchical keywords **top**, **bottom**, or **current** (used conditionally based on context) can be used to specify the dimension. Should this validation fail, an **error** will be generated.

Function Inverses

Some functions (such as **cover**) have what are referred to as “inverse” functions. This is required, as all expressions in a rule group must be algorithmic inverses of each other. Each function states whether it has an inverse, and, if so, what the syntax of the inverse is.

An inverse function is only relevant when the function encompasses the whole of the expression. Functions embedded in longer expressions do not have inverses, though the expression itself may have an inverse as long as the measure being ‘solved’ for is not an input into the function. Functions that have inverses usually have enough scope in their syntax to cover the eventualities that would typically cause them to be embedded in longer expressions (such as code to prevent an error result).

Functions with Multiple Results

The following special syntax should be used for functions with multiple results.

The left-hand side measures in a multi-result expression are comma-separated and can be identified by a labeling mechanism. The syntax for labels is as follows:

<measure>:<label>

Valid label names are specified by the multi-result function syntax. If a multi-result function specifies valid labels, the function can be used in an expression without specifying all possible results. The multi-result function itself is aware of which results are being stored and may be able to run faster by skipping the computation of unneeded results.

Special Handling for Functions

There are several keywords and functions that have special control flow over the evaluation of the expression.

Error Handling

RPAS has no facility for holding an “error” value for a cell. Should the evaluation of any expression, or clause in an expression, result in an error, the value for the cell or clause will be the “**naval**.”

Note: It is good programming practice to check for any clauses that may return an error, and the **prefer** function provides a way to specify the behavior under these circumstances. Some functions have their own implicit error handling.

if

Used for handling conditional logic and masking updates within expressions.

Syntax: **if**(<condition>, <use-expression>, <else-expression>)

where <condition> is any valid Boolean expression. <use-expression> and <else-expression> are any valid expressions that are evaluated based on the result of <condition>; one (and only one) of these expressions can contain the keyword **ignore**. <use-expression> is evaluated when the result of <condition> is true; <else-expression> is evaluated when the result of <condition> is not true.

<expression> is any valid expression. **ignore** is a keyword used to indicate that the entire expression is not to be evaluated, i.e. masking the update to the entire expression.

Note: **ignore** can ONLY be used in either the <use-expression> or <else-expression>, but not both.

The use of **ignore** always flags the expression as a masked update – this will always prevent the expression from being evaluated or involved with aggregations when the condition is not met. To reiterate, note that the entire expression is not evaluated, not just the sub-expression that uses the **if** clause. When **ignore** is used in expression where the LHS measure is modified with the **master** keyword (typically in a commit rule group), then the <condition> must be a Boolean measure (in other words, not an expression). This syntactical restriction is validated when the expression is parsed.

if clauses can be nested without restrictions but must be enclosed with parentheses when used more than once within an expression.

Examples:

Conditional logic:

- `BOP = if(current == first, SeasOP, lag(EOP))`
- `OTB = if(ProjEOP > PlanEOP, 0, PlanRecpt - OnOrder)`

Masked update with a single expression:

- `SalesOP = if(Approved, SalesWP, ignore)`
This example updates Sales for the Original Plan version to the value in the Working Plan version when the Boolean measure Approved is set to true. **ignore** designates that no update is made to SalesOP if the Approved measure is false. This is functionally equivalent to the next example.
- `SalesOP = if(NotApproved, ignore, SalesWP)`
This example does not update the measure SalesOP with the values from the measure SalesWP when the Boolean measure NotApproved is true.

Note the distinctly different behavior between the following similar expressions:

- `a = b + (if(<condition>, c, ignore))`
This is an example of a masked update where no update is made to measure a if the condition is not met, i.e. the entire expression is not evaluated.
- `a = b + (if(<condition>,c, 0))`
This is an example of conditional logic where an else clause is provided and the expression is always evaluated, thus “a” is always updated to either “b” or “b+c”

prefer

Returns the first non-error value from a series of expressions.

The primary use is to enable the capture and appropriate calculation of error conditions.

Syntax: **prefer**(<expression1>, <expression2> [, <expression3> ... <expressionn>])

Where < expression1-n> are expressions which return values of the appropriate data type. The function returns the value of the first of the expressions that does not generate an **error** when it is evaluated. It is good coding practice to use a **prefer** function around any clause of an expression, which could potentially generate an **error**.

Inverse: The **prefer** function does not have an inverse.

Examples:

- `prefer(A/B, 100)`
Returns the value of A divided by B, unless that generates an **error** (as it would if B is zero), when it returns 100.
- `prefer(lag(A), B)`
Returns the value the **lag** of A, unless that generates an **error** (as it would when evaluating the first period of the plan horizon), when it returns the value of B. The **prefer** function in this example is thus the functional equivalent of the expression:
`if(current == first, B, lag(A)).`

Non-Conforming Measures

Definition

One of the strengths of the RPAS engine is that a workbook may contain measures with different “scopes”: the size and shape of the “multidimensional cube” of data may vary by measure. Any two given measures in a workbook may have scopes that align exactly (for instance, both measures have a base intersection of SKU/Store/Week), or where one is a subset of the other (for instance, one has a base intersection of SKU/Store/Week, and the other is at Class/Week). There can also be circumstances where each measure includes a hierarchy in its base intersection that the other dimension does not use (for instance, one has a base intersection of Class/Week and the other is Store/Week). In extreme circumstances, the scopes of two measures may have no point of overlap at all (for instance, one has a base intersection of Class and the other Store).

It is the scope of the measure on the left hand side of an expression (referred to as the LHS measure) that determines the cells that must be calculated by the expression, though that scope may be modified by the use of a modifier such as **level**. Where one or more measures on the right-hand side (RHS) of an expression have a scope that is different (in any way) to the left-hand side (LHS) measure, the expression is deemed to be “non-conforming.” There is special logic to handle the calculation of non-conforming expressions, which depends on the type of nonconformity.

Although not explicitly declared, there is a single logical “All” position at the top of every hierarchy. When considering non-conformity, any measure that is not dimensioned on a hierarchy, is implicitly assumed to be dimensioned on the “All” level of that hierarchy, and thus all data values are assumed to be for the “All” position. This concept is the key to understanding the handling of non-conforming expressions.

When the concept of the “All” position is understood, all expressions can be considered to contain measures that use exactly the same hierarchies. The only potential differences between them are the “bottom levels” (dimensions in the base intersection). Thus, for handling non-conformity, only three cases need to be considered, for each hierarchy:

1. RHS same

In this case, the RHS measure has the same bottom level as the LHS measure. The RHS measure is “conforming” for that hierarchy, and values for the RHS measure are taken from the same position as the position being calculated for the LHS measure.

2. RHS higher

In this case; the RHS measure has a higher bottom level than the LHS measure. The RHS measure is “non-conforming” for that hierarchy. The values for the RHS measure for the position being calculated are assumed to be the same as the value of the RHS measure for the position in its bottom dimension that is the parent (ancestor) of the position being calculated. Effectively, it can be considered that the value of the measure has been “replicated” down the hierarchy to the required level.

3. RHS lower

In this case; the RHS measure has a lower bottom level than the LHS measure. The RHS measure is “non-conforming” for that hierarchy, but because the scope of the RHS measure includes the bottom level for the LHS measure, values for the RHS measure are taken from the same position as the position being calculated for the LHS measure. RHS measure is aggregated using the default aggregation method.

The conceptual case where the measures have scopes that do not overlap, because they have base intersections in a hierarchy that are for dimensions that are up different “branches” of the hierarchy, fails rule validation.

Examples

Note: These examples all use the simple expression $a = b + c$.

Example 1

- “a” has a base intersection of SKU/Store/Week
- “b” has a base intersection of SKU/Week
- “c” has a base intersection of SKU/Region/Week

For each SKU/Store/Week, “a” is calculated from the value of b at SKU/Week (it is assumed that the value of “b” is the same for all positions in the location hierarchy) and the value of “c” at SKU/Region/Week, for the Region the Store belongs in. If “replication” from the Region level is not appropriate, the rule writer can simulate other “spreading” techniques by the use of functions and modifiers such as count and level.

For example, the count function may be used to determine the number of Stores in the Region, and so dividing the measure c by that count will simulate “even” spreading. Additionally level could be used to force the calculation of a at Region instead of a’s base intersection, Store ($a.\text{level}([\text{loc}].[\text{reg}])=b+c$). In this scenario edits to “b” or “c” would calculate “a” at Region and would then spread those values down to Store for measure a using the default spread method.

Example 2

- “a” has a base intersection of SKU/Store/Week
- “b” has a base intersection of SKU/Week
- “c” has a base intersection of SKU

For each SKU/Store/Week, “a” is calculated from the value of b at SKU/Week (it is assumed that the value of b is the same for all positions in the location hierarchy) and the value of “c” at SKU (it is assumed that the value of “b” is the same for all positions in the location hierarchy and time hierarchy).

Note: An alternative approach, if required, would be to use a level modifier on the measure a, so that it is calculated at, say, SKU/Week, and then spread down to SKU/Store/Week, using the existing store participations to the measure a.

Example 3

- “a” has a base intersection of SKU/Week
- “b” has a base intersection of SKU/Store/Week
- “c” has a base intersection of SKU/Region/Week

For each SKU/Week combination, “a” is calculated from the value of “b” and “c” at SKU/“All”/Week. Otherwise stated, b and c are aggregated up the location hierarchy, and then added to “a” for each position in SKU/Week.

Functional Keywords

Overview

Functional Keywords are keywords that may be used in expressions that return specific data values. There are a group of keywords that provide information (in the form of index numbers) about the calendar hierarchy, and a further group of keywords that provide information of the current session.

Calendar Index Functional Keywords

Certain calendar index functional keywords are supported in the syntax, as described below. In this context, a calendar index number is an ordinal position counter of the position in a dimension within the scope of the calendar horizon, where the dimension is as for the cell being evaluated. For example, in a plan whose scope is a year, the first week will have an index of 0, week 26 will have an index of 25, and week 52 will have an index of 51. Similarly, if an expression is being evaluated at the quarter level, the first quarter will have an index of 0, and the last one an index of 3. Calendar index functional keywords may be included in any numeric expression.

first

Returns the index number of the first calendar position.

This keyword is provided for completeness and clarity of rule function writing, since the value will always be zero!

last

Returns the index number of the last calendar position.

last + 1 will therefore always be the number of positions in the calendar horizon in the current dimension.

current

Returns the index number of the period being evaluated.

current can be used as a standalone keyword only under the context of time.

Note: It can also be used in the syntax of a function as a hierarchical keyword (for specifying the current level in a hierarchy) and is allowed for any hierarchy (but must follow the syntax <hierarchy>.**current**).

today

Returns the index number of the period that contains the current time as given by the system clock.

The index number that **today** returns is determined by the base intersection of the measure that is being evaluated (on the left hand side of the expression). For example, if the base intersection of the measure being evaluated is week, **today** will return the index number of the current week.

Note: The effect of this keyword may be overridden by providing the environment variable RPAS_TODAY. If this is present, the time in the RPAS_TODAY environment variable is used instead of the system clock time.

Note: The difference between the keywords **today** and **now** in that **today** returns an index number; **now** returns the value of the current date and time. An **error** is generated when the current period is not included in the workbook.

elapsed

- Returns the index number of the period that is the last elapsed period.
- **elapsed** is interpreted as the last period for which actuals have been posted. If there is no elapsed period, this keyword returns -1.
- **elapsed** must be assigned (a measure on the left hand side of an expression evaluated in a load rule group) before it can be used in calculations (on the right hand side of other rule groups). Use the following syntax for assigning the index number in the base calendar dimension as the elapsed value.
- Syntax: **elapsed** = <expression>
Where <expression> is any valid expression that returns a numeric value, of which only the integer portion is used. **elapsed** can only be used on the left hand side of an expression in a load rule group. The elapsed keyword can be used on the right hand side of other rule groups only after the elapsed value has been assigned.

Session Keywords

now

Returns the current date and time from the system clock.

now is stored with date and time information.

Note: The difference between the keywords **today** and **now** in that **today** returns an index number; **now** returns the value of the current date and time.

The displayed format of **now** is based on the measure type.

Note: The RPAS 11.0 client cannot display time.

This keyword can be used to hold information about when data was changed (for instance, the beginning date and time of a batch run). The value returned by **now** can be overridden by RPAS_TODAY environment variable.

userid

Returns a string that contains the id of the current user.

This keyword can be used to hold information about the user who made a specific change.

Calendar Hierarchical Date Keywords

begin

Returns a date type value for the first index in the root calendar dimension.

Because it returns a date type value, this keyword is not context sensitive (meaning it does not depend on where it is being used) and can be compared with the **now** keyword.

Note: The root calendar dimension is defined as the unique dimension that is at the root of the calendar hierarchy.

end

Returns a date type value for the last index in the root calendar dimension.

Because it returns a date type value, this keyword is not context sensitive (meaning it does not depend on where it is being used) and can be compared with the **now** keyword.

Modifiers

Overview

Modifiers are used to directly modify the source or destination of measures. Modifiers must be used in conjunction with a measure in the following manner:

Syntax: <measure>[.<modifier>[.<modifier>]...]

The following modifiers can be used with measures in a variety of ways. Note the acceptable uses for each modifier as there are restrictions regarding use on the left hand side and if they can be used in conjunction with other modifiers.

master

References the domain-version of a measure.

master is used as a modifier to a measure to reference the version of the measure that resides in the domain. It can only be used in load and commit rule groups. It cannot be used in calculation rule groups.

Syntax: <measure>.**master**

Where <measure> is any valid measure. **master** can be used on both the left hand side and right hand side of expressions and can be used with functions. When used with other modifiers **master** must be the first modifier.

On the right hand side of an expression, **master** can be used with both **level** and **aggtype**. On the left hand side, **master** must be used by itself.

Examples

- Sales=Sales.**master**
Used in load rule group to retrieve Sales from the domain into a workbook.
- Sales.**master**=Sales
Used in commit rule group to commit the updated Sales measure to the domain from the version in the workbook.

aggtype

References to alternative aggregation types.

When a measure is referenced just by name in an expression, or as a parameter in a rule function, the value used is for the default aggregation type for the measure. Values from alternative aggregation types are also available by using the syntax:

Syntax: *<measure>.<aggtype>*

Where **<aggtype>** is a supported aggregation type as listed in an appendix of this document. Every function parameter that requires a measure will also accept this extended form.

Note: If alternate aggregation types are required for a measure in rules, this approach is more efficient than defining another measure with the alternate aggregation type, as data values at the base intersection are not duplicated.

The **aggtype** modifier can only be used on the right hand side of an expression, but can be used with functions and other modifiers. When used with **level** and/or **master** modifiers, **aggtype** must be the last specified.

level

Returns the value of an expression for a specific intersection of 'parent' positions, or forces the calculation at a specific intersection.

The parents specified may be in one or more hierarchies.

Syntax: *<measure>.level(<dimspec1>[+<dimspec2>... +<dimspecn>])*

Where *<dimspec1-n>* is [*<hierarchy>*].{[*<dimension>*] | **top** | **current**} and each dimension specification is separated by a plus "+" sign.

<measure> is the measure to be specified. *<hierarchy>* is the name of a valid hierarchy. **top** and **current** are keywords referring to the highest, and current (that is, being evaluated if on the RHS, or base intersection in the hierarchy if on the LHS) dimensions in the hierarchy. If a hierarchy is not specified, the *<dimension>* for that hierarchy is assumed to be **current**. If the *<dimension>* for a hierarchy is lower than the base intersection for the measure (when used on the LHS), or the *<dimension>* is not a valid dimension in the specified hierarchy, an **error** is generated.

This modifier can be used on both the LHS and RHS of a rule expression. It can only be used by itself on the LHS, but it can be combined with other functions and modifiers on the RHS.

When this modifier is on the LHS of a rule expression, the rule is evaluated at the specified intersection. The newly calculated value at an aggregated intersection is then spread down the hierarchies to the base intersection for the measure, using the default spread-type for the measure. A typical usage of this modifier on the LHS of a rule expression is to calculate a "non-conforming" measure, where the scope of the measure includes hierarchies not present in the measures on the RHS of the expression. The calculation would usually be at the base intersection of the common hierarchies, but at the 'top' of the additional hierarchies, and spread to their base intersections.

When this modifier is on the RHS of a rule expression, the measure being modified is evaluated at the specified intersection

Note: Just the measure, not the rule, is evaluated at the designated level.

Under normal circumstances a measure is always calculated at the base intersection, or the intersection at which the LHS is being evaluated if it is higher. Use of the modifier will evaluate the measure at the designated (higher) level using the measure's default aggregation type, which can be overridden by the `aggtype` modifier. An example of its use on the RHS could be calculating a ratio of sales for each SKU with respect to its parent department.

Examples

- `sales.level([loc].top)`
Returns the value for the measure sales for the position at the top of the location hierarchy and for the current position in all other hierarchies.
- `sales.level([loc].[area])`
Returns the value for the measure sales for the position in the area dimension that is the parent of the position being evaluated and the current position in all other hierarchies (that is, 'the total sales in my area').
- `sales.level([loc].[area]+[prod].[div])`
Returns the value for the measure sales for the position in the area and division dimensions that is the parent of the position being evaluated and current position in all other hierarchies (that is, 'the total sales in my area for my division').
- `recpts.level([rec].top) = <expression>`
The measure "recpts" is calculated at the base intersection of all hierarchies except the "rec" hierarchy, where it is calculated at the top. This value is spread down to the base intersection for the measure.

old

References the value of a measure as of the previous calculate.

Syntax: `<measure>.old`

Any measure modified with **old** will use the value that was available at the start of the calculation process, which means that these modified measures can be ignored for such things as protection processing. Most importantly, this means that a measure can effectively be calculated from itself, as the **.old** modifier breaks the cycle.

Assumptions/restrictions:

- Can only be used in a rule group of type "calculation"
- Can only be used on the right-hand side of an expression
- Cannot be used in combination with **.master**, **level**, or **.aggtype** modifiers
- Cannot be used with (cannot modify) non-materialized measures

Use of the **old** modifier has no effect on calculation sequence or protection processing, as the values of measures modified with **old** are known before the calculation starts.

Note: The old modifier is not designed to operate with measures whose aggregation type is *recalc*. In particular, expressions that attempt to use the old modifier on a measure with an aggregation type of *recalc*, such as

$a = b + c.\text{old}$

where *c* is a measure with an aggregation type of *recalc*, are illegal. Similarly, expressions that attempt to calculate a measure with an aggregation type of *recalc*, but which use the old modifier, such as

$c = a + b.\text{old}$

where *c* is a measure with an aggregation type of *recalc*, are also illegal.

Example

The **old** modifier can be used in conjunction with the **propspread** function to implement a hierarchical relationship among measures. In the following example, Total sales (TotalSls) is the “parent” measure and regular sales (RegSls), promotional sales (PromSls), and markdown sales (MkdSales) are the “child” measures. Using **old** and **propspread** to configure this relationship allows the manipulation of any combination of these measures before calculating, except for all of them.

In the following example and in other such hierarchical measure relationships, the order of the expressions within a rule is critical for the measures to be correctly calculated.

TotalSls = RegSls + PromoSls + MkdSls

RegSls, PromoSls, MkdSls = **propspread**(TotalSls, RegSls.**old**, PromoSls.**old**, MkdSls.**old**)

PromoSls, MkdSls = **propspread**(TotalSls - RegSls, PromoSls.**old**, MkdSls.**old**)

RegSls, MkdSls = **propspread**(TotalSls - PromoSls, RegSls.**old**, MkdSls.**old**)

RegSls, PromoSls = **propspread**(TotalSls - MkdSls, RegSls.**old**, PromoSls.**old**)

RegSls = TotalSls - PromoSls - MkdSls

PromoSls = TotalSls - RegSls - MkdSls

MkdSls = TotalSls - RegSls - PromoSls

Description of Functions

Calendar Index Functions

These are functions that return the calendar index numbers of positions that are specified relative to the current position through hierarchical relationships, or by date. Support is in place for functions to find the first and last children of a parent at a given dimension (for instance, the first week of the current quarter, the last week of the current month). These are to support relative time series functions, such as month to date totals. These may be constrained by setting a condition under which the expression is evaluated.

indexfirst

Returns the calendar index number of the first position in the current dimension that is descended from the parent of the current position at the specified dimension.

See the **tssum** function for an example of typical usage. The function may be constrained by setting a condition for the evaluation.

Syntax: **indexfirst**([<clndhierarchy>].{[<dimension>] | **top**}[, <boolexpr>])

Where <clndhierarchy> is the name of the calendar (time) hierarchy, and <dimension> is the name of a dimension in the calendar hierarchy. **top** is a keyword that implies the top dimension in the calendar hierarchy. If <dimension> is not a valid dimension in the calendar hierarchy, or it is not a dimension that is equal to or higher than the current (being evaluated) dimension in any alternate hierarchy, an **error** is generated.

<boolexpr> is optional and is any valid Boolean expression used to set a condition for the evaluation of the function. If <boolexpr> is not specified, the function returns the index number of the first position of the dimension descended from the parent of the current position of the specified dimension. When <boolexpr> is specified, the function returns the index number of the first position of the dimension descended from the parent of the current position at the specified dimension where the <boolexpr> evaluates to true.

Inverse: The **indexfirst** function does not have an inverse.

Examples:

- **indexfirst**([clnd].[qtr])
If the cell being evaluated is a week, this returns the calendar index number of the first week in the quarter that the week of the cell being evaluated belongs to (that is, 'the first week in the current quarter').
- **indexfirst**([clnd].[week], Receipts != 0)
If the cell being evaluated is a day, this returns the calendar index number of the first day of the current week when that has a value for Receipts that is not equal zero (that is, 'the first day in the current week with recorded Receipts').
- **indexfirst**([clnd].**top**)
If the cell being evaluated is a week, this returns the calendar index number of the first week in the calendar horizon. This keyword is included for consistency with other functions, as it will always return the value **first** (that is, zero).

indexlast

Returns the calendar index number of the last position in the current dimension that is descended from the parent of the current position at the specified dimension.

The function may be constrained by setting a condition for the evaluation.

Syntax: **indexlast**([<clndhierarchy>].{[<dimension>] | **top**}[, <boolexpr>])

Where <clndhierarchy> is the name of the calendar (time) hierarchy. <dimension> is the name of a dimension in the calendar hierarchy. **top** is a keyword that implies the top dimension in the calendar hierarchy. If <dimension> is not a valid dimension in the calendar hierarchy, or is not a dimension that is equal to or higher than the current (being evaluated) dimension (in any alternate hierarchy), an **error** is generated.

<boolexpr> is optional and is any valid Boolean expression used to set a condition for the evaluation of the function. If <boolexpr> is not specified, the function returns the index number of the last position of the dimension descended from the parent of the current position at the specified dimension. When <boolexpr> is specified, the function returns the index number of the last position of the dimension descended from the parent of the current position at the specified dimension where the <boolexpr> evaluates to true.

Inverse: The **indexlast** function does not have an inverse.

Examples:

- **indexlast**([clnd].[qtr])
If the cell being evaluated is a week, this returns the calendar index number of the last week in the quarter that the week for the cell being evaluated belongs to (that is, “the last week in the current quarter”).
- **indexlast**([clnd].[week], Receipts != 0)
If the cell being evaluated is a day, this returns the calendar index number of the last day of the current week that has a value for Receipts that is not equal to zero (that is, “the last day of the current week with recorded Receipts”).
- **indexlast**([clnd].top)
If the cell being evaluated is a week, this returns the calendar index number of the last week in the calendar horizon. This keyword is included for consistency with other functions, as it will always return the value **last**.

indextostartdate

Returns the start date of the period whose index number is supplied.

Syntax: **indextostartdate**(<index>[, [<clndhierarchy>].{[<dimension>] | **current**}]])

Where <clndhierarchy> is the name of the calendar (time) hierarchy, and <dimension> is the name of a dimension in the calendar hierarchy. **current** is a keyword that implies the current dimension in the calendar hierarchy. If <dimension> is not a valid dimension in the calendar hierarchy, an **error** is generated. If the calendar hierarchy and dimension are not supplied, the default is the current calendar dimension.

Note: This function requires that the day dimension of the calendar hierarchy be included in the workbook. If the lowest dimension of the calendar hierarchy is above the day dimension, the function will not be able to return a valid date.

<index> is an expression that returns an index number in the indicated calendar dimension. If *<index>* is non-integer, only the integer portion is used. If *<index>* is not a valid index number for the specified dimension, an **error** is generated. If the measure being evaluated does not have a base intersection in the calendar hierarchy, and the **current** option is used, an **error** is generated.

The function returns a date that is the start date of the period indicated by the dimension and index number. If the period being evaluated is at or below the day level, the start date is the date of the whole of the period. If the period being evaluated is above the day level, the start date is the date of the first child position at the day level of the period being evaluated.

Inverse: The *indextostartdate* function does not have an inverse.

Examples:

- **indextostartdate(current)**
Returns the start date of the current time period
- **indextostartdate (indexfirst([cldn].[qtr]))**
Returns the start date of the first period in the current time dimension in the current quarter
- **indextostartdate (index([cldn].[week], openweek), [cldn].[week])**
Returns the start date of the period at the week level whose name is held in the openweek measure

indextoenddate

Returns the end date of the period whose index number is supplied.

Syntax: **indextoenddate**(*<index>*[, [*<cldnhierarchy>*].[*<dimension>*] | **current**])

Where *<cldnhierarchy>* is the name of the calendar (time) hierarchy, and *<dimension>* is the name of a dimension in the calendar hierarchy. **current** is a keyword that implies the current dimension in the calendar hierarchy. If *<dimension>* is not a valid dimension in the calendar hierarchy, an **error** is generated. If the calendar hierarchy and dimension are not supplied, the default is the current calendar dimension.

Note: This function requires that the day dimension of the calendar hierarchy be included in the workbook. If the lowest dimension of the calendar hierarchy is above the day dimension, the function will not be able to return a valid date.

<index> is an expression that returns an index number in the indicated calendar dimension. If *<index>* is non-integer, only the integer portion is used. If *<index>* is not a valid index number for the specified dimension, an **error** is generated. If the measure being evaluated does not have a base intersection in the calendar hierarchy, and the **current** option is used, an **error** is generated.

The function returns a date that is the end date of the period indicated by the dimension and index number. If the period being evaluated is at or below the day level, the end date is the date of the whole of the period. If the period being evaluated is above the day level, the end date is the date of the last child position at the day level of the period being evaluated.

Inverse: The *indextoenddate* function does not have an inverse.

Examples:

- **indextoenddate(current)**
Returns the end date of the current time period.
- **indextoenddate (indexfirst([clnd].[qtr]))**
Returns the end date of the last period in the current time dimension in the current quarter.
- **indextoenddate (index([clnd].[week], openweek), [clnd].[week])**
Returns the end date of the period at the week level whose name is held in the openweek measure.

Index and Position Functions

This is a class of general functions that may be used for any hierarchy that enables reference to positions in a generic manner. In most cases, the functions do not generate results that are useful in themselves, but that are used as parameters that are embedded into other functions.

An “index” is an internal reference to a position in a dimension. For dimensions in the calendar hierarchy, the index reflects an ordering of positions because there is a well-defined sequence (oldest to newest, based on the start and end dates) of periods. There are special calendar index functions (see above) that exploit this property. For other dimensions, there is no such ordering, and the index number can be considered to be ‘random.’

Note: Index numbers (including calendar index numbers) should not be saved and reused between planning sessions, as there is no guarantee that the same index numbers will apply in subsequent sessions, as the positions or relationships in a hierarchy may change.

These general index functions may be used for any hierarchy, including the calendar hierarchy.

index

Returns the index number of the specified position in the specified dimension of the specified hierarchy.

Syntax: **index**([<hierarchy>].[{<dimension> | **current**}] [, { <stringexpr> | <dateexpr> }])

Where *<hierarchy>* is the name of a valid hierarchy, and *<dimension>* is the name of a valid dimension in that hierarchy. **current** is a keyword that returns the current dimension in *<hierarchy>*. If *<hierarchy>* is not a valid hierarchy or *<dimension>* is not a valid dimension in that hierarchy, an **error** is generated.

<stringexpr> and *<dateexpr>* are optional expressions that can be used to specify a position. If neither *<stringexpr>* nor *<dateexpr>* are specified the function returns the index number of the current position of the dimension being evaluated. *<stringexpr>* is a string expression that results in a position name. If the result of *<stringexpr>* is not a valid position name in the dimension being evaluated, an **error** is generated. *<dateexpr>* is a numeric expression that results in a date type value and can only be used if *<hierarchy>* is the calendar hierarchy. If the result of *<dateexpr>* is not a date type value, or the result is returned when evaluating a dimension that is not in the calendar hierarchy, an **error** is generated.

The function returns the index number of the indicated position in the specified dimension of the specified hierarchy. When used with dates, the indicated position is the position that contains the date specified.

Inverse: The **index** function does not have an inverse.

Examples:

- **index**([prod].[item], likeitem)
This returns the index number of the string position in the item dimension referenced in the likeitem measure.
- **index**([prod].[cls], "cls123")
This returns the index number of the class cls123.
- **index**([clnd].[mnth], opendate)
This returns the index number of the month that contains the date that results from the opendate measure.

position

Returns the position name of the position in the specified dimension of the specified hierarchy with the supplied index number. The returned string is in uppercase.

Syntax: **position**([<hierarchy>].[<dimension>] | **current**)[, <indexexpression>]

<hierarchy> must be the name of a valid hierarchy. If specified, *<dimension>* must be the name of a valid dimension in that hierarchy. **current** is a keyword that returns the current dimension in *<hierarchy>*. If *<hierarchy>* is not a valid hierarchy or *<dimension>* is not a valid dimension in that hierarchy, an **error** is generated.

<indexexpression> is an optional parameter to specify the index of the position to be evaluated. If *<indexexpression>* is not specified, the current position is assumed. The expression must be a valid expression that results in a numeric measure. The integers of the resulting values of the expression are used as the index numbers to determine the position to be evaluated. If *<indexexpression>* does not return a valid index number for the specified dimension an **error** is generated.

The function returns an uppercase string that is the position name of the position with the specified index number for the specified dimension of the specified hierarchy.

Inverse: The **position** function does not have an inverse.

Examples:

- **position**([prod].[item], 3)
This returns the position name of the item with index number =3.
- **position**([prod].[item], likeindex)
This returns the position name of the item with the index number in the measure likeindex.
- **position**([prod].current)
This returns the position name of the current position of the current dimension in the product hierarchy.

attribute

Returns the value of the specified attribute for the current position, or the position with the supplied index number.

Syntax: **attribute**(<attribute>, [<hierarchy>].{[<dimension>] | **current**}[<indexexpression>])

Where <attribute> is a valid attribute for the dimension to be used, otherwise an **error** is generated. <hierarchy> must be the name of a valid hierarchy. If specified, <dimension> must be the name of a valid dimension in that hierarchy. **current** is a keyword that returns the current dimension in <hierarchy>. If <hierarchy> is not a valid hierarchy or <dimension> is not a valid dimension in that hierarchy, an **error** is generated.

<indexexpression> is an optional parameter to specify the index of the position to be evaluated. If <indexexpression> is not specified, the current position is assumed. The expression must be a valid expression that results in a numeric measure. The integers of the resulting values of the expression are used as the index numbers to determine the position to be evaluated. If <expression> does not return a valid index number for the specified dimension an **error** is generated.

Valid values for <attribute> for all non-measure dimensions include the following:

"label" – the label (description) for the position; this value must be specified using quotes.

The function returns the value of the specified attribute for the specified position.

Inverse: The **attribute** function does not have an inverse.

Examples:

- **attribute**(label, [prod].current)
This returns the value of the label attribute for the current position of the current dimension in the product hierarchy.
- **attribute**(label, [prod].[item], likeindex)
This returns the value of the label attribute for the item with the index number in the measure likeindex (that is, "the label for my like item").

Time Series Functions

This is a collection of very similar functions to perform typical calculation tasks over a range of cells in one or more time series. A <start> and an <end> position, defined using calendar index numbers, specify the range of cells to be used. Typically, there may be some arithmetic performed to calculate the start and/or end positions.

Note: By using the **indexdate** or **index** functions to provide calendar index numbers, the <start> and <end> positions to be used in the time series can be specified effectively by position name or by date.

Note: If the level modifier is used, the **current** keyword only has a value when the level used is higher than the level being evaluated (since, for example, the concept of “the current week” is ambiguous when evaluating a month, so an **error** is generated).

Single Time Series Functions

tssum

Produces a sum of the cells in the time series for the measure defined by the start and end positions.

This is used for such calculations as ‘season to date,’ ‘balance to achieve,’ ‘4 week moving sum,’ and so on. The function produces a sum of the cells in the time series for the positions implied by the <start> and <end> for the specified dimension.

Syntax: **tssum**(<expression>[, <start>[, <end>]])

Where <expression> is an expression or measure whose time series is to be used, and <start> and <end> are expressions that calculate numbers. The current calendar dimension is assumed, and if the cell being evaluated does not have a calendar dimension, the bottom calendar dimension is assumed. If the values of <start> or <end> are numeric but non-integer, only the integer portion will be used. If <end> is less than <start>, or either parameter is non-numeric or outside the scope of the calendar index numbers for the specified dimension, an **error** is generated.

<expression> is mandatory, the other parameters are optional. If <start> is not specified, the default value is **first** (that is, 0). If <end> is not specified, the default value is **current**.

Use the **level** modifier to specify a dimension in the calendar hierarchy when calculating above or below the current calendar dimension.

Inverse: The **tssum** function does not have an inverse.

Examples:

- **tssum**(PlanSales)
This is a plan-to-date or running total value for sales.
- **tssum**(PlanSales, **current**, **last**)
This provides a 'balance to achieve' (that is, a sum from the current period to the end of the horizon).
- **tssum**(PlanSales.level([clnd].[week]), **current** – 3, **current**)
This provides a 4 week moving total for sales.
- **tssum**(PlanSales, **indexfirst**([clnd].[qtr]))
This provides a 'quarter to date' running total (see the **indexfirst** function).

tsavg

The average (mean) value of the cells in the range.

The function produces an average of the cells in the time series for the positions implied by the *<start>* and *<end>* for the specified dimension.

Syntax: **tsavg**(*<expression>*[, *<start>*[, *<end>*]])

Where *<expression>* is an expression or measure whose time series is to be used, and *<start>* and *<end>* are expressions that calculate numbers. The current calendar dimension is assumed, and if the cell being evaluated does not have a calendar dimension, the bottom calendar dimension is assumed. If the values of *<start>* or *<end>* are numeric but non-integer, only the integer portion will be used. If *<end>* is less than *<start>*, or either parameter is non-numeric or outside the scope of the calendar index numbers for the specified dimension, an **error** is generated.

<expression> is mandatory, the other parameters are optional. If *<start>* is not specified, the default value is **first** (for instance, 0). If *<end>* is not specified, the default value is **current**.

Use the **level** modifier to specify a dimension in the calendar hierarchy when calculating above or below the current calendar dimension.

Inverse: The **tsavg** function does not have an inverse.

tsmax

The maximum value of any cell in the range.

The function returns the maximum value of the cells in the time series for the positions implied by the *<start>* and *<end>* for the specified dimension.

Syntax: **tsmax**(*<expression>*[, *<start>*[, *<end>*]])

Where *<expression>* is an expression or measure whose time series is to be used, and *<start>* and *<end>* are expressions that calculate numbers. The current calendar dimension is assumed, and if the cell being evaluated does not have a calendar dimension, the bottom calendar dimension is assumed. If the values of *<start>* or *<end>* are numeric but non-integer, only the integer portion will be used. If *<end>* is less than *<start>*, or either parameter is non-numeric or outside the scope of the calendar index numbers for the specified dimension, an **error** is generated.

<expression> is mandatory, the other parameters are optional. If *<start>* is not specified, the default value is **first** (for instance, 0). If *<end>* is not specified, the default value is **current**.

Use the **level** modifier to specify a dimension in the calendar hierarchy when calculating above or below the current calendar dimension.

Inverse: The **tsmax** function does not have an inverse.

tsmin

The minimum value of any cell in the range.

The function returns the minimum value of the cells in the time series for the positions implied by the *<start>* and *<end>* for the specified dimension.

Syntax: **tsmin**(*<expression>*[, *<start>*[, *<end>*]])

Where *<expression>* is an expression or measure whose time series is to be used, and *<start>* and *<end>* are expressions that calculate numbers. The current calendar dimension is assumed, and if the cell being evaluated does not have a calendar dimension, the bottom calendar dimension is assumed. If the values of *<start>* or *<end>* are numeric but non-integer, only the integer portion will be used. If *<end>* is less than *<start>*, or either parameter is non-numeric or outside the scope of the calendar index numbers for the specified dimension, an **error** is generated.

<expression> is mandatory, the other parameters are optional. If *<start>* is not specified, the default value is **first** (for instance, 0). If *<end>* is not specified, the default value is **current**.

Use the **level** modifier to specify a dimension in the calendar hierarchy when calculating above or below the current calendar dimension.

Inverse: The **tsmin** function does not have an inverse.

tsmode

The modal value of the cells in the range.

The function returns the modal value of the cells in the time series for the positions implied by the *<start>* and *<end>* for the specified dimension.

Syntax: **tsmode**(*<expression>*[, *<start>*[, *<end>*]])

Where *<expression>* is an expression or measure whose time series is to be used, and *<start>* and *<end>* are expressions that calculate numbers. The current calendar dimension is assumed, and if the cell being evaluated does not have a calendar dimension, the bottom calendar dimension is assumed. If the values of *<start>* or *<end>* are numeric but non-integer, only the integer portion will be used. If *<end>* is less than *<start>*, or either parameter is non-numeric or outside the scope of the calendar index numbers for the specified dimension, an **error** is generated.

<expression> is mandatory, the other parameters are optional. If *<start>* is not specified, the default value is **first** (i.e. 0). If *<end>* is not specified, the default value is **current**.

If there is more than one value for the mode the function, it returns the first value that is calculated.

Use the **level** modifier to specify a dimension in the calendar hierarchy when calculating above or below the current calendar dimension.

Inverse: The **tsmode** function does not have an inverse.

tsmedian

The median value of the cells in the range.

The function returns the median value of the cells in the time series for the positions implied by the *<start>* and *<end>* for the specified dimension.

Syntax: **tsmedian**(*<expression>*[, *<start>*[, *<end>*]])

Where *<expression>* is an expression or measure whose time series is to be used, and *<start>* and *<end>* are expressions that calculate numbers. The current calendar dimension is assumed, and if the cell being evaluated does not have a calendar dimension, the bottom calendar dimension is assumed. If the values of *<start>* or *<end>* are numeric but non-integer, only the integer portion will be used. If *<end>* is less than *<start>*, or either parameter is non-numeric or outside the scope of the calendar index numbers for the specified dimension, an **error** is generated.

<expression> is mandatory, the other parameters are optional. If *<start>* is not specified, the default value is **first** (for instance, 0). If *<end>* is not specified, the default value is **current**.

If there is no middle number, the function returns the average of the middle two numbers.

Use the **level** modifier to specify a dimension in the calendar hierarchy when calculating above or below the current calendar dimension.

Inverse: The **tsmedian** function does not have an inverse.

tsstd

The standard deviation of the cells in the range.

The function returns the standard deviation of the cells in the time series for the positions implied by the *<start>* and *<end>* for the specified dimension.

Syntax: **tsstd**(*<expression>*[, *<start>*[, *<end>*]])

Where *<expression>* is an expression or measure whose time series is to be used, and *<start>* and *<end>* are expressions that calculate numbers. The current calendar dimension is assumed, and if the cell being evaluated does not have a calendar dimension, the bottom calendar dimension is assumed. If the values of *<start>* or *<end>* are numeric but non-integer, only the integer portion will be used. If *<end>* is less than *<start>*, or either parameter is non-numeric or outside the scope of the calendar index numbers for the specified dimension, an **error** is generated.

<expression> is mandatory, the other parameters are optional. If *<start>* is not specified, the default value is **first** (for instance, 0). If *<end>* is not specified, the default value is **current**.

Use the **level** modifier to specify a dimension in the calendar hierarchy when calculating above or below the current calendar dimension.

Inverse: The **tsstd** function does not have an inverse.

tsvar

The variance of the cells in the range.

The function returns the variance of the cells in the time series for the positions implied by the *<start>* and *<end>* for the specified dimension.

Syntax: **tsvar**(*<expression>*[, *<start>*[, *<end>*]])

Where *<expression>* is an expression or measure whose time series is to be used, and *<start>* and *<end>* are expressions that calculate numbers. The current calendar dimension is assumed, and if the cell being evaluated does not have a calendar dimension, the bottom calendar dimension is assumed. If the values of *<start>* or *<end>* are numeric but non-integer, only the integer portion will be used. If *<end>* is less than *<start>*, or either parameter is non-numeric or outside the scope of the calendar index numbers for the specified dimension, an **error** is generated.

<expression> is mandatory, the other parameters are optional. If *<start>* is not specified, the default value is **first** (for instance, 0). If *<end>* is not specified, the default value is **current**.

Use the **level** modifier to specify a dimension in the calendar hierarchy when calculating above or below the current calendar dimension.

Inverse: The **tsvar** function does not have an inverse.

Double Time Series (Statistical Error) Functions**tsme**

Produces the Mean Error of an 'estimate' time series compared to an 'actuals' time series.

Syntax: **tsme**(*<x>*, *<y>*[, *<start>*[, *<end>*]])

Where *<x>* is an expression or measure that represents the "estimate" and *<y>* is an expression or measure that represents the "actuals," and *<start>* and *<end>* are expressions that calculate numbers. The current calendar dimension is assumed, and if the cell being evaluated does not have a calendar dimension, the bottom calendar dimension is assumed. If the values of *<start>* or *<end>* are numeric but non-integer, only the integer portion will be used. If *<end>* is less than *<start>*, or either parameter is non-numeric or outside the scope of the calendar index numbers for the specified dimension, an **error** is generated.

<x> and *<y>* are mandatory, the other parameters are optional. If *<start>* is not specified, the default value is **first** (that is, 0). If *<end>* is not specified, the default value is **current**. The Mean error is calculated using the formula below.

$$\frac{\sum_{i=0}^{n-1} (x_i - y_i)}{n}$$

Inverse: The **tsme** function does not have an inverse.

Examples:

- `tsme(FcstSales, ActSales)`
This calculates the Mean Error of the FcstSales measure from the start of the calendar horizon until the current time period.
- `tsme(FcstSales, ActSales, first, elapsed)`
This calculates the Mean Error of the FcstSales measure from the start of the calendar horizon until the last time period with actuals loaded.
- `tsme(FcstSales, ActSales, first, min(elapsed, current))`
This calculates the Mean Error of the FcstSales measure from the start of the calendar horizon until the first of the period being evaluated or the last time period with actuals loaded.

tsmae

Mean Absolute Error.

Syntax: **tsmae**(*<x>*, *<y>*[, *<start>*[, *<end>*]])

Where *<x>* is an expression or measure that represents the “estimate” and *<y>* is an expression or measure that represents the “actuals,” and *<start>* and *<end>* are expressions that calculate numbers. The current calendar dimension is assumed, and if the cell being evaluated does not have a calendar dimension, the bottom calendar dimension is assumed. If the values of *<start>* or *<end>* are numeric but non-integer, only the integer portion will be used. If *<end>* is less than *<start>*, or either parameter is non-numeric or outside the scope of the calendar index numbers for the specified dimension, an **error** is generated.

<x> and *<y>* are mandatory, the other parameters are optional. If *<start>* is not specified, the default value is **first** (i.e. 0). If *<end>* is not specified, the default value is **current**. The Mean Absolute error is calculated using the formula below.

$$\frac{\sum_{i=0}^{n-1} |x_i - y_i|}{n}$$

Inverse: The **tsmae** function does not have an inverse.

tsmape

Mean Absolute Percentage Error.

Syntax: **tsmape**(<x>, <y>[, <start>[, <end>]])

Where <x> is an expression or measure that represents the “estimate” and <y> is an expression or measure that represents the “actuals,” and <start> and <end> are expressions that calculate numbers. The current calendar dimension is assumed, and if the cell being evaluated does not have a calendar dimension, the bottom calendar dimension is assumed. If the values of <start> or <end> are numeric but non-integer, only the integer portion will be used. If <end> is less than <start>, or either parameter is non-numeric or outside the scope of the calendar index numbers for the specified dimension, an **error** is generated.

<x> and <y> are mandatory, the other parameters are optional. If <start> is not specified, the default value is **first** (i.e. 0). If <end> is not specified, the default value is **current**. The Mean Absolute Percentage error is calculated using the formula below.

$$\frac{\sum_{i \in \{0 \leq i < n \cap y_i \neq 0\}} \frac{|x_i - y_i|}{y_i}}{\sum_{i \in \{0 \leq i < n \cap y_i \neq 0\}} 1}$$

Inverse: The **tsmape** function does not have an inverse.

tsrmse

Root Mean Square Error.

Syntax: **tsrmse**(<x>, <y>[, <start>[, <end>]])

Where <x> is an expression or measure that represents the ‘estimate’ and <y> is an expression or measure that represents the “actuals,” and <start> and <end> are expressions that calculate numbers. The current calendar dimension is assumed, and if the cell being evaluated does not have a calendar dimension, the bottom calendar dimension is assumed. If the values of <start> or <end> are numeric but non-integer, only the integer portion will be used. If <end> is less than <start>, or either parameter is non-numeric or outside the scope of the calendar index numbers for the specified dimension, an **error** is generated.

<x> and <y> are mandatory, the other parameters are optional. If <start> is not specified, the default value is **first** (i.e. 0). If <end> is not specified, the default value is **current**. The Root Mean Square error is calculated using the formula:

$$\sqrt{\frac{\sum_{i \in \{0 \leq i < n\}} (x_i - y_i)^2}{n}}$$

Inverse: The **tsrmse** function does not have an inverse.

tspae

Percentage Absolute Error.

Syntax: **tspae**(*<x>*, *<y>*[, *<start>*[, *<end>*]])

Where *<x>* is an expression or measure that represents the “estimate” and *<y>* is an expression or measure that represents the “actuals,” and *<start>* and *<end>* are expressions that calculate numbers. The current calendar dimension is assumed, and if the cell being evaluated does not have a calendar dimension, the bottom calendar dimension is assumed. If the values of *<start>* or *<end>* are numeric but non-integer, only the integer portion will be used. If *<end>* is less than *<start>*, or either parameter is non-numeric or outside the scope of the calendar index numbers for the specified dimension, an **error** is generated.

<x> and *<y>* are mandatory, the other parameters are optional. If *<start>* is not specified, the default value is **first** (i.e. 0). If *<end>* is not specified, the default value is **current**. The Percentage Absolute error is calculated using the formula below.

$$\frac{\sum_{i=0}^{n-1} |x_i - y_i|}{\sum_{i=0}^{n-1} |y_i|}$$

Inverse: The **tspae** function does not have an inverse.

Hierarchical Functions and Procedures

This is a collection of functions and procedures that provide some knowledge of hierarchical structures, and the how the current position fits in, or uses knowledge of hierarchical structures.

count

Returns the count of children at a specified level that belong to a parent at a higher level.

Syntax: **count**([*<hierarchy>*][, *<childdimspec>*[, *<parentdimspec>*]])

Where *<childdimspec>* is { [*<childdimension>*] | **bottom** | **current** }

and *<parentdimspec>* is [*<hierarchy>*].{ [*<parentdimension>*] | **top** | **current** }

<hierarchy> is the name of a valid hierarchy (same hierarchy must be referenced throughout the function). *<childdimension>* and *<parentdimension>* must be valid dimensions in the specified hierarchy. If both are specified, then *<childdimension>* must be lower than *<parentdimension>* in a roll-up, or an **error** is generated. **bottom**, **top** and **current** are keywords referring to the lowest, highest, and current (being evaluated) dimensions in the hierarchy. If *<childdimspec>* is not specified, the default is **bottom**. If *<parentdimension>* is not specified, the default is **current**.

The function returns the number of children in the dimension *<childdimension>* that are descended from the implied position (the current position or the ancestor of the current position at the specified level) in the dimension *<parentdimension>*. If the *<childdimension>* and the *<parentdimension>* are the same, the function returns the value of 1.

Inverse: The **count** function does not have an inverse.

Examples:

- **count([loc].bottom)**
Returns the number of children in the bottom dimension in the location hierarchy for the current position in the location hierarchy
- **count([loc].str)**
Returns the number of children in the store dimension (str) in the location hierarchy for the current position in the location hierarchy (that is, 'how many stores do I own')
- **count([loc].str, [loc].area)**
Returns the number of children in the store dimension in the location hierarchy for the position in the dimension "area" that is the ancestor of the current position in the location hierarchy (that is, 'how many stores in my area')

lookup

Procedure that returns the value of an expression for a specific intersection.

The positions to be "looked up" may be in one or more hierarchies. This procedure has special uses and restrictions in RPAS 11.0:

- lookup is a procedure and thus cannot be combined with functions and other procedures in any manner
- Used for history mapping and like SKU/sister store functionality
- The base intersection of the output measure must be the same as the input measure and/or one or more of the mapping measures

Syntax: <output> <- **lookup**(<input>, <dimspec1> [, <dimspec2> ... , <dimspecn>])

Where <dimspec1-n> is [*<hierarchy>*].[*<dimension>*] | **bottom** | **current** | **top**], <map>

<output> is the measure being updated. <input> is the measure to be evaluated. Each <dimspec> is used to specify the hierarchy and dimension to be used in the mapping process and the measure that contains the mapping values.

For each <dimspec> that is specified, the <hierarchy> must be the name of a valid hierarchy and the <dimension> must be the name of a valid dimension in that hierarchy. **top** is a keyword that refers to the highest dimension in the hierarchy, **bottom** is a keyword that refers to the lowest dimension in the hierarchy, and **current** is a keyword that refers to the current dimension (that is, the dimension of the cell being evaluated).

<map> is either a measure or an explicitly stated position used to designate how positions in <input> are mapped to determine the resulting values in <output>. The output, input, and mapping measures used in the **lookup** procedure must conform in a certain manner. Specifically, the resulting measure <output> must have the same base intersection as <input>, <map>, or both measures so that all conform.

When <map> is a measure, it must result in either index numbers or position names, either of which must be valid index numbers or position names from the related dimension specification. When <map> contains index numbers that do not map to valid positions, an *error* is generated and the *na value* for <output> is returned. There is no special "cycle breaking" logic for the **lookup** procedure. This means that a measure may never be calculated from the **lookup** of the same measure.

Note: `lookup` is a procedure so it cannot be combined with functions, modifiers, or other procedures in any manner. As a procedure, it requires a different syntax: “<-” instead of “=” when being assigned.

Inverse: The `lookup` procedure does not have an inverse.

Examples:

- **`output <- lookup(input, [presentationstyle].[ presentationstyle], map)`**
Where output is at sku-week, input is at sku-presentationstyle, and map is at sku-week with position names or index numbers from the presentation style dimension. The output and mapping measures have the same base intersection. This expression calculates the output measure from the mapping of presentation styles that vary for each sku-week combination.
- **`output <- lookup(input, [prod].[sku], map)`**
Where output and input are at sku-week and map is at sku with position names or index numbers from the sku dimension. This expression calculates the output measure from the like sku of the input measure.

tablelookup

Procedure that returns the value (interpolated if necessary) from the entry in a “table” of information held in measures that matches with supplied “keys.”

Syntax: **`tablelookup(<expression>, <matching technique>, <keymeasure> [
,<resultmeasure>])`**

Where *<matching technique>* is {exactmatch, *<nomatchvalue>* | average | nearest | high | low | interpolate}

The `tablelookup` procedure requires that a “table” be available that may be the target of the lookup. This table will be formed from normal measures with a base intersection of “normal” dimensions. Nevertheless, the most usual usage will be where the table measures are dimensioned on a dimension built for the purpose, plus other dimensions as required.

Example:

Imagine a requirement to look up valid price points that may be applied as prices for an *item*. The collection of valid price points will be different for each *class*. To satisfy this requirement, a table is built. A ‘table’ hierarchy is defined with a “tableentry” dimension with a number of positions, which are named e01, e02, e03, ... e99 to allow for 99 entries in the table, with the order of the positions being the same as their natural sort sequence, and the order of the hierarchy being the highest (innermost) non-time hierarchy. A measure named ‘pp’ is defined with a base intersection of tableentry/class. The ‘pp’ measure is populated with valid pricepoints for each class, with the lowest valid pricepoint in position e01, the next lowest in e02, and so on. This ‘table’ can now be used to ‘look up’ valid pricepoints. The procedure call (indirectly) provides a class and (directly) provides a target price as arguments, and a valid price point is returned based on the selected matching technique. See the following examples for an example that uses this ‘table.’

<expression> is any valid expression that results in a value of the same data type as the *<keymeasure>*. In the description that follows, this value is referred to as the key value. *<keymeasure>* is the name of the measure to be used as a key when matching the key value against the table. *<resultmeasure>* is an optional measure that holds the return

value. If *<resultmeasure>* is not specified, *<keymeasure>* is used for the values of the result as in the price point example, below.

The procedure attempts to match the key value against an entry in the table. The innermost non-time dimension in the base intersection of the *<keymeasure>* is assumed to be the dimension along which entries in the table are indexed. For all other dimensions in the base intersection of the *<keymeasure>*, the procedure will match against the parent at that dimension of the cell being evaluated.

Note: The values in *<keymeasure>* must be in ascending order and must not contain any repeated values. A value that is either out of sequence or repeated designates that the previous value is the last entry in the 'table.' In other words, only the sorted elements in the key measure will be considered in the lookup process.

The *<matching technique>* specifies the matching technique to be used when an exact match of the *<keymeasure>* against the key value is not found. If the matching technique is **exactmatch**, *<nomatchvalue>* is a numeric value that must be specified to indicate the value to use in cells when there is no exact match. Otherwise, if the key value is higher than the highest value in the "table," or lower than the lowest value in the table, it is assumed to match against the highest or lowest value accordingly. If the matching technique is **high** and no match against the key value is found, the procedure returns the value of the *<resultmeasure>* for the entry immediately higher than the key value. If the matching technique is **low** and no match against the key value is found, the procedure returns the value of the *<resultmeasure>* for the entry immediately lower than the key value. If the matching technique is **nearest** and no match against the key value is found, the procedure returns the value of the *<resultmeasure>* for the entry immediately lower than the key value or immediately higher than the key value, depending upon which entry is nearest (this is like rounding to the nearest value). If the matching technique is **average** and no match against the key value is found, the procedure returns the numeric average of the value of the *<resultmeasure>* for the entry immediately lower and immediately higher than the key value, or it generates an error if the *<resultmeasure>* is not of numeric data type.

If the matching technique is **interpolate** and no match against the key value is found, the procedure returns an interpolated value between the value of the *<resultmeasure>* for the entry immediately lower and immediately higher than the key value, or it generates an error if the *<resultmeasure>* is not of numeric data type. The interpolation is calculated as:

$$\text{lowresult} + \frac{(\text{highresult} - \text{lowresult}) * (\text{keyvalue} - \text{lowvalue})}{(\text{highvalue} - \text{lowvalue})}$$

Inverse: The **tablelookup** procedure does not have an inverse.

Examples:

- **tablelookup**(tgtpr, nearest, pp)
Returns the nearest valid value of the pp measure to the supplied target price (tgtpr).
- **tablelookup**(perc, interpolate, epct, elast)
Looks up the percentage markdown (perc) of the current position against a percentage change elasticity table (epct). Returns the matching elasticity value (elast). If the percentage markdown is not found in the table, the procedure will interpolate the elasticity value from the nearest values above and below the percentage markdown.

flookup

The fixed lookup function. It returns the value of a measure for an explicitly named fixed intersection.

Syntax: **flookup**(<measure>, <posspec1> [, <posspec2> ... , <posspecn>])

Where <posspec1-n> is: [<hierarchy>]. [<dimension>]. [<positionname>]

<measure> is the measure to be looked up. This <measure> must conform with the measure being calculated as follows. Some hierarchies may be present in the base intersection of both measures, and these are handled by normal “non-conforming” logic. For any hierarchies that are only in the base intersection of the measure being calculated (output measure), all positions will lookup the same value. For any hierarchies that are only in the base intersection of the <measure> (input measure), the position to be used **must** be **explicitly** named through a position specification (<posspec>).

Note: If the position to be used can only be specified indirectly (for example, if it is held in a measure), the **flookup** function cannot be used, and the more powerful **lookup** procedure should be used instead.

flookup can be used to return a constant or a slice. In case of a constant, the NA value of the **flookup** function will be the value of the constant. In case of a slice, the NA value of the **flookup** function will be the NA value of <measure>.

For each <posspec> that is specified, the <hierarchy> must be the name of a valid hierarchy, the <dimension> must be the name of a valid dimension in that hierarchy, and the <positionname> must be the name of a valid position in that dimension. If the position name includes special characters, it can be enclosed in quotes (" ") in addition to the standard requirement for square brackets ([]). If <hierarchy> is not a valid hierarchy or <dimension> is not a valid dimension in that hierarchy, or <positionname> is not a valid position in that dimension an **error** is generated.

Additionally, <dimension> must be a dimension in the base intersection of <measure>. To use dimensions not in the base intersection, the <measure> must have a **level** modifier to explicitly raise it to the desired dimension.

There is no special “cycle breaking” logic for the **flookup** function. This means that a measure may never be calculated from the **flookup** of the same measure. The **flookup** function returns the value of the expression from the specified fixed intersection.

Inverse: The **flookup** function does not have an inverse.

Examples:

- **flookup**(perc, [flvl].[flvl].[flvla])
Returns the value for the measure perc for the position 'flvla' in the flvl dimension of the flvl hierarchy.
- **flookup**(leadtime, [prod].[cls].[class1], [loc].[whse].[whseA])
Returns the value for the measure *leadtime* for the class 'class1' for the warehouse 'whseA.'

aggregate

Note: The aggregate procedure provides similar functionality to the hybrid aggregation type. Measures that use the hybrid aggregation type cannot be manipulated above their base intersection (as there is no mechanism to spread changes), but since the aggregate procedure is used on recalc measures, they can be changed with the change being applied through normal mapping rules. In addition, the Aggregate procedure has a "recalc" aggregation type that is not available in the hybrid aggregation method.

This procedure returns the value of a measure aggregated from the base intersection to the current level using the supplied aggregation type.

Syntax: **aggregate** (<cachemeasure>, <hierspec1> [, <hierspec2> ... , <hierspecn>])

Where <hierspec1-n> is [<hierarchy>].<aggtype>

The rule writer specifies a <cachemeasure> that holds the base intersection values to be aggregated, and it is also the source of values for 'recalc' aggregation.

The rule writer also specifies the aggregation type for each hierarchy and the priority sequence to be used. The priority sequence is required because at levels that are aggregated in more than one hierarchy (for instance, Department/Region/Month for a measure with a base intersection of Class/Store/Week), different results would usually be obtained by aggregating up each of the hierarchies. For example, if the requirement is to aggregate up the product hierarchy by using the total aggregation type, up the location hierarchy by using the average aggregation type, and up the calendar hierarchy by using the first aggregation type; there are three potential ways to calculate a value at Department/Region/Month. We could total from Class/Region/Month, average from Department/Store/Month, or first from Department/Region/Week. These would almost certainly generate three completely different values. By providing a priority sequence, the rule writer explicitly determines which of these values are required. See the worked example, below.

Note: The effect of a series of aggregations of the same type up a single hierarchy may return different results from those of a measure with the same aggregation type.

“Normal” aggregation for a measure driven by its aggregation type is performed from all base intersection cells descended from the cell being evaluated. For example, for a measure with a base intersection of Class/Week and an ‘average’ aggregation type, the value calculated for a cell at Department/Month is the average of all values for all Class/Week cells for the Department/Month. If the measure is a recalc measure, calculated at aggregated levels from a rule with an aggregate function, such as **aggregate**(x, [prod].average, [clnd].average), the value for the Department/Month will be the average of all the Class/Months (not Class/Weeks) that belong to the Department/Month. Other than coincidentally, this would generate a different value.

<cachemeasure> is the measure to be aggregated, and the value of <cachemeasure> is also the value used for cells that are at the base intersection (bottom levels), and at aggregated levels when the required aggregation type is *recalc*. <hierarchy> is the name of a valid hierarchy. Each hierarchy may only be specified once in the procedure, but hierarchies may appear in any order. The sequence that the hierarchies are specified in is used to determine which hierarchy to aggregate up if the cell being evaluated is at an aggregated level in more than one hierarchy. In this circumstance, aggregation is performed up the first specified hierarchy that the cell is at an aggregated level in, and the other hierarchies are ignored. <aggtype> specifies the aggregation type to be used. The <aggtype> must be one of the standard aggregation types. If any hierarchy that is in the scope of the measure being calculated is not explicitly specified, the aggregation type of that hierarchy is assumed to be total. Such hierarchies are assumed to be sequenced after all hierarchies that are explicitly referenced, and they are ordered from innermost to outermost.

Note: The value of the <cachemeasure> is used at the base intersection of the measure being calculated. If, for a given cell, the aggregation type to be used is *recalc*, the value is also obtained directly from the <cachemeasure> at that level, which will normally have an aggregation type of *recalc*.

Note: **aggregate** is a procedure so it cannot be combined with functions, modifiers, or other procedures in any manner. As a procedure it requires a different syntax: “<-” instead of “=” when being assigned.

Inverse: The **aggregate** procedure does not have an inverse.

Examples:

- `result <- aggregate(x, [cldn].recalc)`
For cells at the base intersection, the value is calculated from the measure x. For cells at an aggregated level in the calendar hierarchy, the value is also obtained from the measure x, which we can assume has an aggregation type of recalc, and thus the result of the procedure is as if the aggregation type were recalc, using the usual expression to calculate measure x. If the cell is not at an aggregated level in the time hierarchy, and assuming in this example that the other hierarchies are product and location; in that priority, the value for a cell at an aggregated product level is calculated as the total of all cells for products descended from that product for the same location and time. Otherwise, the value for the cell is calculated as the total of all cells for locations descended from the cell's location for the same product and time.
- `result <- aggregate(x, [loc].average)`
In a similar manner to the previous example, cells at aggregated levels in the location hierarchy will be calculated by averaging the values of cells for all descendent locations. Otherwise, the value will be totaled up the product or time hierarchy as appropriate.
- `result <- aggregate(x, [cldn].average)`
Totals up all hierarchies except time, which uses an average aggregation type.
- `result <- aggregate(x, [prod].average, [cldn].first)`
This example averages up the product hierarchy if possible. Otherwise, takes the first child value up the calendar hierarchy. Otherwise, totals up the other hierarchies.
- `result <- aggregate(x, [prod].average, [cldn].last)`
This example averages up the product hierarchy, if possible. Otherwise, it takes the last child value up the calendar hierarchy. Otherwise, totals up the other hierarchies.

Multi-level Calculation Example:

Consider a measure calculated from the expression `aggregate(x, [prod].total, [loc].avg, [cldn].first)`. The measure is assumed to have a base intersection of Class/Store/Week. Examples of the calculations that would be applied at various levels are:

- Class/Store/Month: first from Class/Store/Week
- Class/Region/Month: avg from Class/Store/Month
- Department/Region/Month: total from Class/Region/Month

Normalization and Resizing Functions

resize

Uses the “shape” of a time series to produce another time series of a different length, but with the same shape.

Syntax: **resize**(<expression>, <start>, <fromlength>, <tolength>)

Where <expression> is a measure or expression whose time series is to be used, and <start>, <fromlength> and <tolength> are expressions that calculate numbers. <start> is assumed to be a calendar index number: if its value is numeric but non-integer, only the integer portion will be used. If <fromlength> or <tolength> are less than 0, or either parameter is non-numeric or when added to <start>-1 is outside the scope of the calendar index numbers for the dimension being calculated, an **error** is generated. If <fromlength> or <tolength> are non-integer, only the integer portion will be used.

The function returns a time series that is resized such that the overall shape of the values is retained, but the number of time periods is stretched or shrunk from <fromlength> or <tolength>. For time periods outside the horizon covered by <start> and <start> -1 + <tolength> (if there are any) the function will return zero – if values other than this are required, or if no update to those periods is required, the function should be wrapped in an if function that can set the appropriate value or use the **ignore** clause, as appropriate.

The function stretches or shrinks the section of the time series by interpolation or decimation. The algorithm uses upsampling, convolution, and then downsizing. The filter used in convolution is a finite impulse response (FIR) lowpass filter, using a hamming window with cut-off frequency and length determined from greatest common denominator of the source and destination time series lengths.

The values generated for individual cells through this process are not normalized (for a similar function that normalizes the result, see the **resizenorm** function), and will be of similar magnitude to the cell values for the source cells.

Inverse: The **resize** function does not have an inverse.

Examples:

- **resize**(profile, first, 10, 17)

The first 17 periods of the result time series will have values with a shape the same as the first 10 periods of the measure *profile*. All other periods will be zero.

- **resize**(lag(profile,startweek), startweek, profilelength, numweeks)

This example should be compared with the similar example of the **normalize** function. It uses a profile to generate a sales plan for an item for a specified length of time from a specified period of time. The profile is not necessarily the same length as the period for which sales are to be generated. The measure *profile* is assumed to have a profile (shape) for the sales of an item, starting in the first period with values for a number of periods given by the measure *profilelength*. *startweek* is an index number of the period from which sales should be generated for the item. *numweeks* has the length of the sales profile to be generated. Periods before the *startweek* or after the *startweek-1+numweeks* will have a result of zero. The periods from *startweek* to *startweek-1+numweeks* will have the result of the first *profilelength* weeks of the *profile* measure, stretched or shrunk to fit the appropriate number of periods.

resizenorm

Uses the “shape” of a time series to produce another time series of a different length, but with the same shape, normalized to a specific total.

Syntax: **resizenorm**(<expression>, <start>, <fromlength>, <tolength>[, <total>])

<total> is an expression that returns a numeric value. If <total> is not specified, it is assumed to be the sum of the cells of <expression> from startweek to startweek-1+fromlength. See the **resize** function for an explanation of the other parameters.

This function is identical to the **resize** function, except that the calculation engine automatically normalizes the resized values to the specified <total>.

Inverse: The **resizenorm** function does not have an inverse.

Examples:

- **resizenorm**(profile, first, 10, 17)
The first 17 periods of the result time series will have values with a shape the same as the first 10 periods of the measure *profile*. All other periods will be zero. The values of the cells will be such that sum of the 17 generated periods of the result time series will be the same as the first 10 periods of the measure *profile*.
- **resizenorm**(lag(profile,startweek), startweek, profilelength, numweeks, targetsales)
This example should be compared with the similar example of the **resize** function. The generated sales will be normalized so that their sum is the value of the *targetsales* measure.

Other Functions and Procedures

cover

This function returns the number of future periods for which “stock” covers” sales.

Alternately phrased, that is a “forwards weeks of supply,” or the number of future periods of “sales” that could be satisfied from the “stock” with no further receipts.

The **cover** function allows for two “sales” expressions, where the second is a ‘wrap around’ expression to provide a well defined cover for periods at or near the end of the calendar horizon that would otherwise “run out” of forward sales. An offset is also specified to allow the **cover** function to behave appropriately for both opening and closing stock.

Syntax: **cover**(<stockexpression>, <salesexpression>[, <offsetexpression>, [<wraparoundsalesexpression>]])

Where <stockexpression> is an expression or measure that represents the ‘stock.’

<salesexpression> is an expression or measure that represents the “sales.”

<offsetexpression> is an expression that calculates a number that represents the offset to apply. If the value is non-integer, only the integer portion is used. If the value is non-numeric, an **error** is generated. If <offsetexpression> is not provided, the default value will be 1. <wraparoundsalesexpression> is an expression or measure that represents the “wrap around sales.” If <wraparoundsalesexpression> is not provided, there will be no wraparound, and the function will generate an **error** if there is insufficient ‘forward sales’ to calculate the cover.

The *<salesexpression>* can be considered to define a time series of sales data values, starting at the current period offset by the *<offsetexpression>*, and stretching until the end of the calendar horizon. If this time series is too short to evaluate the cover value, it can be considered to be extended by one or more copies of the time series implied by the *<wraparoundsalesexpression>*, if specified, from the start until the end of the calendar horizon. The cover value is calculated by summing down the time series until a sum is reached that is equal to or greater than the value of the *<stockexpression>*. If the sum is equal to the *<stockexpression>*, the number of periods used is returned. If the sum is greater than the *<stockexpression>*, the value returned is the number of periods used minus 1, plus the proportion of the last period reached that is required to exactly reach the value of the *<stockexpression>*. If the *<offsetexpression>* causes the start of the time series to be before the start of the calendar horizon, or no *<wraparoundsalesexpression>* is specified, and there is insufficient 'forward sales' to determine the cover, an **error** is generated.

Inverse: The **cover** function has an inverse function, the **uncover** function. This function returns the amount of "stock" that is required to give a specified number of 'forward periods cover.' There is no inverse function that solves this relationship for sales (which is used as a time series, rather than a single value).

Note: The inverse can only apply if the *<stockexpression>* is a single measure, rather than an expression.

Examples:

- **cover**(EOP, Sales)
This provides an EOP based forward cover. There is no "wraparound" sales expression, so this function will generate **errors** towards the end of the plan horizon.
- **cover**(BOP, Sales + MD, 0)
This provides a BOP based forward cover, using Sales plus markdowns as the expression to be covered. There is no "wraparound" sales expression, so this function will probably generate **errors** towards the end of the plan horizon.
- **cover**(EOP, Sales, 1, Sales)
This provides an EOP based forward cover. Sales itself is used as the 'wraparound sales expression' (this is typical where the plan horizon is a year, since the Sales measure has the appropriate seasonality; where this is not the case, another measure, such as 'next season sales' would be used) so this function will return 'reasonable' values towards the end of the plan horizon when the cover is greater than the number of weeks remaining.

Notes:

The **cover** function is always calculated at the current time dimension. For example, in a plan where the bottom time dimension is week, a measure with an aggregation type of "recalc" that is calculated from a **cover** function at the month level will calculate "forward months of supply." If "forward weeks of supply" are required to be calculated for the month dimension, it would be more appropriate to specify the measure with an aggregation type of first or last, so that aggregation, rather than calculation through the rule, is used to generate the values at the month dimension.

It is the responsibility of the rule writer to ensure that the wrap around expression, if used, is seeded with appropriate values.

Both the “stock” and the “sales” used in the **cover** function are expressions. This supports various business needs, such as using covers based on “sales plus markdowns.” If the “stock” is provided as an expression, rather than just a single measure, the function will not have an inverse.

The offset expression is used to define the offset: which period to start using the sales expression from. It is assumed to be an offset from the current period, so that a value of zero means that the sales for the current period should be used in evaluating the cover (which is appropriate for an “opening stock” based cover), and an offset of 1 means start in the period following the current period (which is appropriate for a “closing stock” based cover). Values other than 0 and 1 may be used.

uncover

This function returns the amount of “stock” required to cover “sales” for the specified number of forward periods.

The **uncover** function allows for two “sales” expressions where the second is a ‘wrap around’ expression that provides a well defined cover for periods at or near the end of the calendar horizon that would otherwise ‘run out’ of forward sales. An offset is also specified to allow the **uncover** function to behave appropriately for both opening and closing stock.

Syntax: **uncover**(<coverexpression>, <salesexpression>[, <offsetexpression>, [<wraparoundsalesexpression>]])

Where <coverexpression> is an expression or measure that represents the “cover value”. <salesexpression> is an expression or measure that represents the ‘sales.’ <offsetexpression> is an expression that calculates a number that represents the offset to apply. If the value is non-integer, only the integer portion is used. If the value is non-numeric, an **error** is generated. If <offsetexpression> is not provided, the default value will be 1. <wraparoundsalesexpression> is an expression or measure that represents the ‘wrap around sales.’ If <wraparoundsalesexpression> is not provided, there will be no wraparound, and the function will generate **errors** if there is insufficient ‘forward sales’ to calculate the stock.

The <salesexpression> can be considered to define a time series of sales data values, starting at the current period offset by the <offsetexpression>, and stretching until the end of the calendar horizon. If this time series is too short to evaluate the stock value, it can be considered to be extended by one or more copies of the time series implied by the <wraparoundsalesexpression>, if specified, from the start until the end of the calendar horizon. The stock value is calculated by summing down the time series for a number of periods equal to the integer portion of the <coverexpression> and adding the value of the next period, multiplied by the fractional portion of the <coverexpression>. If the <offsetexpression> causes the start of the time series to be before the start of the calendar horizon, or no <wraparoundsalesexpression>, is specified, and there is insufficient ‘forward sales’ to determine the stock, an **error** is generated.

Inverse: The **uncover** function has an inverse function, the **cover** function. This function returns the number of forward periods of cover implicit in the specified stock. There is no inverse function that solves this relationship for “sales” (which is used as a time series, rather than a single value).

Note: The inverse can only apply if the *<coverexpression>* is a single measure, rather than an expression.

Examples:

- **uncover**(WOS, Sales)
This provides an EOP stock value that gives the specified weeks of supply. There is no “wraparound” sales expression, so this function will generate *errors* towards the end of the plan horizon.
- **uncover**(WOS, Sales + MD, 0)
This provides a BOP stock value that gives the specified weeks of supply, using Sales plus markdowns as the expression to be covered. There is no “wraparound” sales expression, so unless the value of WOS is less than 1, this function will generate *errors* towards the end of the plan horizon.
- **uncover**(WOS, Sales, 1, Sales)
This provides an EOP stock value that gives the specified weeks of supply. Sales itself is used as the ‘wraparound sales expression’ (this is typical where the plan horizon is a year, since the Sales measure has the appropriate seasonality; where this is not the case, another measure, such as ‘next season sales’ would be used) so this function will return ‘reasonable’ values towards the end of the plan horizon when the cover is greater than the number of weeks remaining.

Notes:

The **uncover** function is always calculated at the current time dimension. For example, in a plan where the bottom time dimension is week, a rule or mapping rule that uses an **uncover** function at the month level will calculate the “stock” on the assumption that the *<coverexpression>* provides a ‘forward months of supply.’

It is the responsibility of the rule writer to ensure that the wrap around expression, if used, is seeded with appropriate values.

Both the “cover” and the “sales” used in the **uncover** function are expressions. This supports various business needs, such as using covers based on “sales plus markdowns.” If the “cover” is provided as an expression, rather than just a measure, the function will not have an inverse.

The offset expression is used to define the offset: which period to start using the sales expression from. It is assumed to be an offset from the current period, so that a value of zero means that the sales for the current period should be used in evaluating the cover (which is appropriate for an “opening stock” based cover), and an offset of 1 means start in the period following the current period (which is appropriate for a ‘closing stock’ based cover). Values other than 0 and 1 may be used.

min

This function returns the minimum value from a series of expressions or set of measures.

Syntax: **min**(<expression1>, <expression2> [, <expression3> ... <expressionn>])

Where < *expression1-n* > are expressions or a set of measures (denoted by {<*measureset*>}), which return numeric values. The function returns the minimum value of the expressions.

Inverse: The **min** function does not have an inverse.

Example:

- **min** (A, B, C)
Returns the minimum of the measures A, B and C

max

This function returns the maximum value from a series of expressions or set of measures.

Syntax: **max**(<expression1>, <expression2> [, <expression3> ... <expressionn>])

Where < *expression1-n* > are expressions or a set of measures (denoted by {<*measureset*>}) which return numeric values. The function returns the maximum value of the expressions.

Inverse: The **max** function does not have an inverse.

Example:

- **max** (A, B, C)
Returns the maximum of the measures A, B and C

sum

This function returns the sum of a series of expressions or measure set.

Syntax: **sum**(<expression1>, <expression2> [, <expression3> ... <expressionn>])

Where < *expression1-n* > are expressions or a set of measures (denoted by {<*measureset*>}), which return numeric values. The function returns the summed value of the expressions or measure set.

Inverse: The **sum** function does not have an inverse.

Example:

- **sum** (A, B, C)
Returns the sum of the measures A, B, and C

lag

This function returns the value of an expression from the previous time period in the dimension being evaluated.

Syntax: **lag**(<expression>)

Where <expression> is any valid expression. The function returns the value of the expression in the previous period. If the current period being evaluated is the first period in the calendar horizon (so that there is no previous period), an **error** is generated. For that reason, **lag** functions are usually embedded in **if** functions or **prefer** functions to check for that case.

Inverse: The **lag** function does not have an inverse.

Example:

- **lag**(EOP)

Returns the value of the measure EOP from the next period

Notes:

lag is deliberately intended as a “simple” version of the **timeshift** procedure for one of the most frequently used cases, which is that the offset is one period in the past. Use the **timeshift** procedure for lagging with a variable offset.

The **lag** function has special “cycle breaking” logic that enables a series of expressions to be calculated in a manner that allows them to be evaluated “period wise.” This allows an apparent “deadly embrace” to be broken. Thus, the two expressions:

$$\text{EOP} = \text{BOP} + \text{Rec} - \text{SLs} - \text{MD} \quad \text{and}$$

$$\text{BOP} = \text{lag}(\text{EOP})$$

are legal, and can be calculated in the same rule group, even though EOP appears to depend on BOP, which appears to depend on EOP. Note however, that the cycle breaking logic does not support the measure being calculated being lagged on the RHS of the expression. Thus the following expression is illegal:

$$\text{AccumSls} = \text{Sls} + \text{lag}(\text{AccumSls})$$

lead

This function returns the value of an expression from the next (following) time period in the dimension being evaluated.

Syntax: **lead**(<expression>)

Where <expression> is any valid expression. The function returns the value of the expression in the following period. If the current period being evaluated is the last period in the calendar horizon (so that there is no following period), an **error** is generated. For that reason, **lead** functions are usually embedded in **if** functions or **prefer** functions to check for that case.

Inverse: The **lead** function does not have an inverse.

Examples:▪ **lead(BOP)**

Returns the value of the measure BOP from the next period.

Notes:

lead is deliberately intended as a “simple” version of the **timeshift** procedure for one of the most frequently used cases, which is that the offset is one period in the future. Use the **timeshift** procedure for leading with a variable offset.

In a similar manner to the **lag** function, the **lead** function has special “cycle breaking” logic that enables a series of expressions to be calculated in a manner that allows them to be evaluated ‘period wise.’ This allows an apparent ‘deadly embrace’ to be broken.

Even when an error is generated because the current period is the last period in the calendar horizon (see Syntax,) the **lead** function itself, if not guarded by **if** or **prefer** functions, returns the re-evaluated NA value of the measure. For example, for the following expression group:

A = **lead**(B)

B = A + 1

Assume that the NA value for both A and B is 0. The system first re-evaluates B's NA value to be A's NA value + 1 = 1 based on the second expression. The system will attempt to retrieve the time period after B's last time period when A = **lag**(B) is evaluated. Because that time period does not exist, the **lead** function will return B's re-evaluated NA value instead, which is 1.

timeshift

Procedure that returns the value of a measure from a time period in the dimension being evaluated that is lagged by a designated number of periods.

This procedure is used as follows:

- Measures used in this procedure can be modified with the **master** modifier
- Currently **timeshift** cannot be used in calculation rule groups
- Used for lagging the values of a measure by more than one period
- Used for retrieving values from time periods outside the scope of the workbook
- Used for addressing 52-53 week year differences

Syntax: <output> <- **timeshift**(<input>, {<lagvalue> | <lagmeas> | <lagmap>})

<input> is the measure that is being lagged and must have the same base intersection as <output> or must be forced to evaluate at the base intersection of <output> by using the **level** modifier. <input> must include a dimension in the calendar hierarchy and must be the same data type as <output>.

<lagvalue> is a scalar value that designates the number of periods each position in <input> is shifted. A negative value refers to shift forward in calendar dimension (lead), and a positive value refers to shift backward in calendar dimension (lag).

<lagmeas> is a numeric measure that contains values that determine how each position is shifted. *<lagmeas>* cannot have a calendar dimension and all non-calendar dimensions must be identical to *<input>*.

Note: This implies that if either *<input>* or *<lagmeas>* measure is modified with the **master** modifier, the other measure must also be modified with the **master** modifier.

<lagmap> is a string measure used for sophisticated mappings. The measure contains position names that indicate how each time period is mapped, and it must only contain positions from the dimension from the calendar hierarchy. Multiple positions can be specified by separating them using a space. In other words, *<lagmap>* defines a mapping of positions from the input measure to the destination measure along time. Entries in *<lagmap>* that are not the names of valid positions in the dimension from the calendar hierarchy are ignored.

Note: This implies that if either *<input>* or *<lagmap>* measure is modified with the **master** modifier, the other measure must also be modified with the **master** modifier.

This mapping technique is primarily used when lagging measures between 52 and 53-week years. When mapping multiple positions to a single position (such as mapping the last 2 weeks in a 53-week year to the last week in a 52-week year), the resulting value is the sum of the source values (that is, the sum of the last 2 weeks of the 53-week year). When mapping a single position to multiple positions (such as mapping the last week in a 52-week year to the last 2 weeks in a 53-week year), the source value is replicated to the resulting values (that is, weeks 52 and 53 in the 53-week year are updated to week 52 in the 52-week year).

Note: **timeshift** is a procedure so it cannot be combined with functions, modifiers, or other procedures in any manner. As a procedure, it requires a different syntax: “<-” instead of “=” when being assigned.

Inverse: The **timeshift** procedure does not have an inverse.

Examples:

- `salesly <- timeshift(sales.master, -52)`
Updates the positions in the workbook measure for last year’s sales with the values from the domain measure sales where each position is lagged by 52 periods
- `salesly <- timeshift(sales.master, saleslag)`
Where sales and salesly have a base intersection of SKU-week, the numeric measure saleslag contains a value for each SKU that indicates the number of periods to lag by SKU
- `salesly <- timeshift(sales.master, salesmap)`
Where salesmap is a string measure that contains position names indicating which position in sales to use for each position in Salesly; the current year in the workbook contains 52 weeks, the previous year that is not in the workbook contains 53 weeks; the 52nd position in SalesLY contains the position names “week52” and “week53” and results in the sum of the 2 positions

round

This function returns the value of an expression rounded up or down to the nearest multiple.

Syntax: **round**(<expression>[, <multipleexpression>])

Where <expression> is any valid expression, which specifies the value to be rounded, <multipleexpression> is an expression that calculates a number that represents the multiplier to use. If the value is not specified, it is assumed to be 1. The value may be non-integer. If the value of either expression is non-numeric, an **error** is generated. The rounding is up or down to the nearest multiple of the multiplier. If there are 2 multiples equally near to the value (for instance, rounding 1.5 to the nearest integer), then rounding is up (away from zero).

Inverse: The **round** function does not have an inverse.

Examples:

- **round(qty)**
Returns the value of the measure qty, rounded up or down to the nearest integer. If the qty is 14.324, this returns the result of 14, if the qty is 14.824, this returns the result of 15.
- **round(qty, packsize)**
Returns the value of the measure qty, rounded up or down to the nearest multiple of the packsize. If the qty is 14.324 and the packsize is 6, this returns the result of 12. If the qty is 16.824 and the packsize is 6, this returns the result of 18.

roundup

Returns the value of an expression rounded up to the nearest multiple.

Syntax: **roundup**(<expression>[, <multipleexpression>])

Where <expression> is any valid expression, which specifies the value to be rounded. <multipleexpression> is an expression that calculates a number that represents the multiplier to use. If the value is not specified, it is assumed to be 1. The value may be non-integer. If the value of either expression is non-numeric, an **error** is generated. Rounding is always up (to the nearest multiple of the multiplier further away from zero).

Inverse: The **roundup** function does not have an inverse

Examples:

- **roundup(qty)**
Returns the value of the measure qty, rounded up to the nearest integer. If the qty is 14.324 or 14.824, this returns the result of 15.
- **roundup(qty, packsize)**
Returns the value of the measure qty, rounded up to the nearest multiple of the packsize. If the packsize is 6 and the qty is 14.324 or 16.824, this returns the result of 18.

rounddown

Returns the value of an expression rounded down to the nearest multiple.

Syntax: **rounddown**(<expression>[, <multipleexpression>])

Where <expression> is any valid expression, which specifies the value to be rounded, <multipleexpression> is an expression that calculates a number that represents the multiplier to use. If the value is not specified, it is assumed to be 1. The value may be non-integer. If the value of either expression is non-numeric, an *error* is generated. Rounding is always down (to the nearest multiple of the multiplier closer to zero).

Inverse: The **rounddown** function does not have an inverse.

Examples:

- **rounddown(qty)**
Returns the value of the measure qty, rounded down to the nearest integer. If the qty is 14.324 or 14.824, this returns the result of 14.
- **rounddown(qty, packsize)**
Returns the value of the measure qty, rounded down to the nearest multiple of the packsize. If the packsize is 6 and the qty is 14.324 or 16.824, this returns the result of 12.

Notes on Round Functions:

The round functions have no inverses. Great care should be used in designing rule groups that use these functions, and the preferred technique for rounding is often to not round during calculation, but to round values on display only.

The round functions cause problems because they can compromise the integrity of rule and expression relationships. Consider a typical relationship between value, units and price. If the units are calculated through a round function (on the apparently reasonable assumption that units should be integers) after a change to, say, the value, then the integrity of the rule relationships is immediately compromised because the price is no longer the value divided by the units.

navalue

Returns the NA value of the specified expression.

Syntax: **navalue**(<expression>)

<expression> can be a constant, a measure, or an expression.

The **navalue** function does not directly generate *errors*, but it can propagate errors generated by <expression>.

Inverse: The **navalue** function does not have an inverse.

Examples:

- **navalue**(<meas>)
This returns the NA value of <meas>.
- **navalue**(<meas1> + <meas2>)
This returns the NA value of the expression "<meas1> + <meas2>." In this example, if the NA value of <meas1> is 2 and the NA value of <meas2> is 5, the result of the **navalue** function will be 7.

proppspread

Multiple result function that spreads a value across a collection of measures while retaining their relative proportions. The multiple results are not named, and are therefore positional only. The typical usage of this function is to allow spreading of 'hierarchical measures.'

Syntax: **proppspread**(<totalexpression>, <childexp1>, ... <childexpn>)

Where <totalexpression> is an expression that returns a numeric value to which to balance the results of the function, <childexp1> - <childexpn> are expressions that provide the 'shape' of the results. They will typically be the same measures as those assigned to the result of the function, but using the **old** modifier. A measure defined as a result cannot be used on the right hand side without **old**.

The function generates n positional results, where n is the number of 'child expressions.' The results will sum to the <totalexpression>, using the 'shapes' of the child expressions in the order of the child expressions.

The number of results should be equal to the number of child expressions, which means that there should be one more argument on the right hand side than output measures on the left hand side. Additional child expressions are ignored. If too few child expressions are defined, the function will fail. Currently, there is no validation to warn an individual when this condition is met.

If the sum of the child expressions is zero, the spread will be even.

Inverse: The **proppspread** function does not have an inverse.

Example:

The **old** modifier can be used in conjunction with the **proppspread** function to implement a hierarchical relationship among measures. In the following example, Total sales (TotalSls) is the "parent" measure and regular sales (RegSls), promotional sales (PromSls), and markdown sales (MkdSales) are the "child" measures. Using **old** and **proppspread** to configure this relationship allows the manipulation of any combination of these measures before calculating, except for all of them.

In the following example and in other such hierarchical measure relationships, the order of the expressions within a rule is critical for the measures to be correctly calculated.

TotalSls = RegSls + PromoSls + MkdSls

RegSls, PromoSls, MkdSls = **proppspread**(TotalSls, RegSls.**old**, PromoSls.**old**, MkdSls.**old**)

PromoSls, MkdSls = **proppspread**(TotalSls - RegSls, PromoSls.OLD, MkdSls.OLD)
 RegSls, MkdSls = **proppspread**(TotalSls - PromoSls, RegSls.OLD, MkdSls.OLD)
 RegSls, PromoSls = **proppspread**(TotalSls - MkdSls, RegSls.OLD, PromoSls.OLD)
 RegSls = TotalSls - PromoSls - MkdSls
 PromoSls = TotalSls - RegSls - MkdSls
 MkdSls = TotalSls - RegSls - PromoSls

passthrough

This multiple result function is used to encapsulate any number of normal computations into a single expression.

Syntax: **passthrough**(<exp1>, <exp2>, ..., <exp-n>)

Where <exp1> - <exp-n> are normal expressions used to calculate the resulting measures.

All measures on the left hand side must be computed at the same base intersection. The number of results should be less than or equal to the number of calculation expressions (additional calculation expressions are ignored). If too few calculation expressions are defined then function will fail. Currently, there is no validation to warn an individual when this condition is met.

There are two main reasons for using this function:

1. Use **passthrough** in an expression for a rule when you need to compute values for multiple measures without having to write (develop) a multiple-result function or procedure.
2. For performance reasons: If many measures are computed using the same or similar set of RHS measures, combining those calculations using 'passthrough' may be faster because there is less physical input/output with the data.

Inverse: the **passthrough** function does not have an inverse.

Examples:

- A, B = **passthrough**(C + D, C - D)
Computes the sum and difference of two measures simultaneously
- SalesA, SalesB = **passthrough**(SalesA.OLD * TotalSales / TotalSales.OLD, SalesB.OLD * TotalSales / TotalSales.OLD)
Proportionately spread TotalSales down to its components, SalesA and SalesB

String Functions

uppercase

Converts a string to upper case.

Syntax: **uppercase**(<expression>)

Where the value of <expression> is returned as a string with upper case characters.

Useful in making string comparisons.

lowercase

Converts a string to lower case.

Syntax: **lowercase**(<expression>)

Where the value of <expression> is returned as a string with lower case characters.

Useful in making string comparisons.

Math Functions

pow

Returns the value of a number raised to the power of another number (x to the power of y).

Syntax: **pow**(<x>, <y>)

<x> and <y> are expressions that return real numbers. <y> designates the exponent to which <x> is raised.

exp

Returns the value of the transcendental number “e” raised to the power of a number (e to the power of x).

“e” is the base of natural logarithms.

Syntax: **exp**(<x>)

<x> is an expression that returns a real number to which the number “e” (value 2.71828183) is raised.

sqrt

Returns the square root of a number.

Syntax: **sqrt**(<x>)

<x> is an expression that returns a real number. This function returns the equivalent of “**pow**(x, 0.5)”.

log

Returns the logarithm of a number.

This function returns the exponent that indicates the power to which a number is raised to produce a given number.

Syntax: **log**(<x>, [<base>])

Where <x> and <base> are expressions that return real numbers. If <base> is not specified the default value is 10.

Examples

- **log**(100)
The logarithm of 100 to the base 10 is 2.
- **log**(125, 5)
The logarithm of 125 to the base 5 is 3.

ln

Returns the natural logarithm of a number.

This function returns the logarithm of <x> to base “e” (2.71828183).

Syntax: **ln**(<x>)

<x> is an expression that returns a real number.

mod

Returns the remainder as the result of the division of 2 numbers.

The result of this function is the remainder of <x> divided by <y>.

Syntax: **mod**(<x>, <y>)

<x> and <y> are expressions that return real numbers.

Example

- **mod**(5, 2)

The remainder of 5 divided by 2 is 1.

abs

Returns the absolute value of a number.

Syntax: **abs**(<x>)

<x> is an expression that returns a real number.

Appendix: Aggregation and Spread Types

This section describes the aggregation and spread types that are supported in RPAS.

Supported Aggregation Types

The table below displays the supported aggregation types.

Aggregation Type	Description	Valid Data Types	Recommended Spread Types
recalc	recalculate measure at each level, via recalc expression	numeric, string, date, Boolean	none
total	sum of all values	numeric	prop
total_pop	sum of all populated values	numeric	prop_pop
average	average of all values	numeric	prop
average_pop	average of all populated values	numeric	prop_pop
min	minimum of all values	numeric, date	repl
min_pop	minimum of populated values	numeric, date	repl_pop
max	maximum of all values	numeric, date	repl
max_pop	maximum of populated values	numeric, date	repl_pop
median	median of all values	numeric	repl
median_pop	median of populated values	numeric	repl_pop
pst	first value in innermost hierarchy, total in others hierarchies Note: "First" only has a meaning in the calendar hierarchy. Therefore, this agg type should only be used for measures whose innermost hierarchy is the calendar hierarchy."	numeric	ps

Aggregation Type	Description	Valid Data Types	Recommended Spread Types
pet	last value in innermost hierarchy, total in other hierarchies Note: "First" only has a meaning in the calendar hierarchy. Therefore, this agg type should only be used for measures whose innermost hierarchy is the calendar hierarchy."	numeric	pe
and	and of all values	Boolean	repl
or	or of all values	Boolean	repl
ambig	ambig of all values (all values equal, otherwise ambig)	string	none
ambig_pop	ambig of all populated values	string	none
popcount	count of populated values in base	numeric, string, date, Boolean	none

Supported Spread Types

The table below displays the supported spread types.

Spread Type	Description	Valid Data Types
none	values are not spread	numeric, string, date, Boolean
prop	spread value proportionally (previous total non-zero) or evenly (previous total zero)	numeric
prop_pop	spread value proportionally (previous total non-zero) or evenly (previous total zero) to all populated cells	numeric
even	spread value evenly	numeric
even_pop	spread value evenly to all populated cells	numeric
delta	increment/decrement each cell evenly. Effectively the 'even' spreading of the change ("delta").	numeric
delta_pop	increment/decrement each cell evenly over the populated cells. Effectively the 'even' spreading of the change ("delta") over the populated cells.	numeric
repl	replicate the value to each cell (this is not possible with recalc measures as a default spread type in RPAS 11.1)	numeric, string, date, Boolean
repl_pop	Replicate the value to each cell that is populated (not possible as a default spread type with recalc measures in RPAS 11.1)	numeric, string, date, Boolean
ps	apply delta to starting period	numeric
pe	apply delta to ending period	numeric

Arithmetic Operators

The following arithmetic operators are supported:

Unary Operators

Symbol	Type	Function
-	Real	negation
!	Boolean	compliment

Binary Operators

Symbol	Type	Function
=	real, Boolean, string, date	assignment
+	real	addition
-	real	subtraction
*	real	multiplication
/	real	division
&&	Boolean	Boolean and
	Boolean	Boolean or
==	real, Boolean, string, date	equality
!=	real, Boolean, string, date	inequality
<	real, Boolean, string, date	less than
<=	real, Boolean, string, date	less than or equal to
>	real, Boolean, string, date	greater than
>=	real, Boolean, string, date	greater than or equal to