Oracle® Retail Predictive Application Server

Configuration Tools
Release 14.1

December 2014

ORACLE

Oracle® Retail Predictive Application Server Configuration Tools, Release 14.1

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Primary Author:

Contributors:

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Value-Added Reseller (VAR) Language

Oracle Retail VAR Applications

The following restrictions and provisions only apply to the programs referred to in this section and licensed to you. You acknowledge that the programs may contain third party software (VAR applications) licensed to Oracle. Depending upon your product and its version number, the VAR applications may include:

(i) 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.

(ii) the **Wavelink** component developed and licensed by Wavelink Corporation (Wavelink) of Kirkland, Washington, to Oracle and imbedded in Oracle Retail Mobile Store Inventory Management.

(iii) 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.

(iv) 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.

You acknowledge and confirm that Oracle grants you use of only the object code of the VAR Applications. Oracle will not deliver source code to the VAR Applications to you. Notwithstanding any other term or condition of the agreement and this ordering document, you shall not cause or permit alteration of any VAR Applications. For purposes of this section, "alteration" refers to all alterations, translations, upgrades, enhancements, customizations or modifications of all or any portion of the VAR Applications including all reconfigurations, reassembly or reverse assembly, re-engineering or reverse engineering and recompilations or reverse compilations of the VAR Applications or any derivatives of the VAR Applications. You acknowledge that it shall be a breach of the agreement to utilize the relationship, and/or confidential information of the VAR Applications for purposes of competitive discovery.

The VAR Applications contain trade secrets of Oracle and Oracle's licensors and Customer shall not attempt, cause, or permit the alteration, decompilation, reverse engineering, disassembly or other reduction of the VAR Applications to a human perceivable form. Oracle reserves the right to replace, with functional equivalent software, any of the VAR Applications in future releases of the applicable program.

Contents

Send Us Your Comments xiii

Preface 1

Audience 1
Related Documents 1
Customer Support 1
Review Patch Documentation 1
Improved Process for Oracle Retail Documentation Corrections 2
Oracle Retail Documentation on the Oracle Technology Network 2
Conventions 2

1 Introduction 1

Overview 1
Configuration Tools Business Process 1
Sample Configurations 2
Using the Configuration Tools Online Help 2
About the Online Help 2
Formatting Conventions 2
Navigate the Online Help 2
Using Links 3
Navigating the Configuration Tools 4
Starting the Program 4
About the Configuration Tools Windows 4
A Note about RPAS Configurability and Extensibility 5

2 Configuration Components Pane 7

Know the Configuration Components 7
Configuration Components Pane Overview 8
How RPAS Uses Solution Configurations 11
Right-Click Menus in the Configuration Components Pane 12
Setting Tools Preferences 13

3 Projects 15

Working with Projects 15
Overview 15
Create a Project 15
Save Changes to a Project 17
Using the Save As Option to Save a Project using a Different Name 17
Open an Existing Project 18
Open an Existing Project from an Older Version of the Configuration Tools 18
Close a Project 20
Hierarchies 21
Overview 21

The Hierarchy Definition Window 22
Working with Hierarchies 25
Create a New Hierarchy 25
Specify Hierarchy Properties 26
Delete a Hierarchy 28
Copy (Clone) Hierarchies 28
Working with Position Formats 29
Specifying the Position Format 29
Working with Dimensions 31
Overview 31
Create a Dimension 31
Defining Dimension Properties 32
Delete a Dimension 36
Edit a Dimension 36
Create a Branch in a Hierarchy 37
Labeled Intersections 37
Data Interface Manager 40
Overview 40
Specify the Data Interface for a Measure 40
Add/Edit Data Interface Properties for a Measure 41
Delete Data Interface Information for a Measure 44
Working with Styles 44
Overview 44
Create a Style 46
Remove a Style 47
Edit a Style 47
Working with Taskflows 48
Overview 48
Create a Taskflow 53
Adding a Task to the Taskflow 55
Add a Step to the Taskflow 57
Add a Tab to the Taskflow 58
Delete Items from the Taskflow 59
Edit Items from the Taskflow 59
Hyperdynamic Tasks, Steps, and Tabs 61
Creating a Multi-Solution Taskflow 65
4 Solutions 69
Working with Solutions 69
Overview 69
Create a Solution 69
Copy a Solution 69
Rename a Solution 70

Move a Solution 71
Delete a Solution 71
Measures and Components 72
Measure Manager 72
Measure Component Design 73
Create a Major Component 74
Create a Minor Component 75
Defining Measure Component Properties 76
Edit Components 87
Alerts 89
Measure Validation within the Measure Manager 89
Working with Measures 91
Overview 91
Realize and Unrealize Measures 93
Rename a Measure 94
Show all Measures 94
Hide Measures by Component 94
Hide All Measures 94
Sort Measures by Property Value 94
Filter Measures by Property Value 95
External Measures 95
Overview 95
Import a Measure 96
Remove an Imported Measure from a Solution 97
Rule Sets 97
Overview 97
Create a Rule Set 98
Delete a Rule Set 99
Rule Groups 99
Overview 99
Create a Rule Group 101
Delete a Rule Group 103
Copy a Rule Group 103
Measure Validation in the Rule Definition Window 104
Rule Definition 105
Create a Rule and Add It to a Rule Group 105
Add an Existing Rule to a Rule Group 110
Apply a Rule Pattern to Create New Rules or to Update Existing Rules 112
Delete a Rule from All Rule Groups 114
Remove a Rule from a Rule Group 115
Edit Properties of a Rule 116
Rename All Rules in a Rule Group 117
Filter Rules in a Rule Group 118

Reordering Rules in a Rule Group 119
Auto Generate Load and Commit Rules 119
Copy Selected Rules to Another Rule Group....120
Find and Replace Measures in the Copied Rules 122
Expressions and Rules....124
Overview....124
Reorder an Expression in a Rule....124
Edit an Expression in a Rule....125
Delete an Expression from a Rule....126
Add an Expression to a Rule 127
RPAS Functions, Procedures, Keywords, and Modifiers 127
Overview....127
Workbooks....128
Overview....128
Overview of Participation Measures....129
Create a Workbook....130
Edit Workbook Properties 130
Defining Workbook Properties 131
Remove a Workbook 167
Working with the Rule Group Simulator 168
Overview....168
About the Rule Group Simulator....168
Invoking the Rule Group Simulator....170
Filtering the Measures Table 170
Changing the Edited Status of Measures....171
Using the Upstream and Downstream Panes 172
Exiting the Rule Group Simulator 173
Working with Workbook Tabs....173
Overview....173
Create a Workbook Tab 173
Edit Workbook Tab Properties....174
Remove a Workbook Tab....174
Comprehensive Workbook Validation 175
Working with Worksheets....175
Overview....175
Create a Worksheet....177
Defining Worksheet Properties for Pivot/Chart Worksheets 177
Defining Worksheet Properties for Detail Popup Worksheets....184
Specify Which Measures Appear in a Worksheet 188
Specify the Sequence of Measures on a Worksheet....189
Edit Worksheet Properties....190
Remove a Worksheet....191
Wizards 191

Overview 191
Create a Wizard Group 192
Create a Wizard Page 192
Edit Wizard Control Properties 193
5 System Preferences 195
Overview 195
Global Domain 195
Overview 195
Setting Workbench Preferences 197
Setting Configuration Properties 198
6 Configuration Utilities 201
Overview 201
Configuration Converter 201
Overview 201
Launching the Configuration Converter 201
Converting a Configuration 202
Functional Library Manager 203
Overview 203
Launching the Functional Library Manager 203
Adding a Function Library to Be Validated in the Configuration Tools 203
Removing a Function Library from Being Validated in the Configuration Tools 204
Report Generator 204
Overview 204
Generate a Report 205
7 Integration Tool 207
Overview 207
RPAS Data Mart 207
Overview 207
Integration Configuration Components 208
Overview 208
Shared Hierarchies and Dimensions 208
Shared Facts 209
Integration Map 210
Domain Information 211
Integration Tool 211
Overview 211
Working Integration Configurations 212
Working with Domain Information 214
Working with Shared Hierarchies 215
Working with Shared Facts 219
Working with the Integration Map 222

Fact Grouping Best Practices 224
A Appendix : Global Domain Technical Information 229
Appendix A: Global Domain Technical Information 229
B Appendix: Calculation Engine User Guide 231
Overview 231
Measure Definition and Base Intersections 231
Data Types 232
Base Intersection 233
Aggregation and Spreading Types 233
Aggregation 233
Spreading 234
Locks and Spreading Around Locked and Changed cells 234
Spreading Methods 235
Hierarchical Protection Processing 238
The Spreading of Recalc Type Measures 240
Expressions, Rules, and Rule Groups 240
Introduction 240
Expressions 241
Rules 241
Rule Groups 242
The Calculation Cycle 244
Introduction 244
Protection Processing 244
Cycle Groups 248
Synchronized Measures 250
Elapsed Period Locking 251
Non-Conforming Expressions 252
Introduction 252
Handling of Non-conforming Expressions 253
C Appendix: Rules Function Reference Guide 255
Overview 255
Functions 255
Procedures 255
Modifiers 256
Keywords 256
Syntax Conventions 256
Specification of Hierarchy, Dimension, or Position 257
Function Inverses 257
Functions with Multiple Results 257
Special Handling for Functions 258
Error Handling 258
Non-Conforming Measures 259

Definition 259
Functional Keywords 261
Overview 261
Calendar Index Functional Keywords 261
Session Keywords 263
Calendar Hierarchical Date Keywords 264
Modifiers 264
Overview 264
master 264
aggtype 265
level 265
old 266
Description of Functions 267
Calendar Index Functions 267
Calendar Calculation Functions 270
Index and Position Functions 272
Forecast Procedure 275
Time Series Functions 279
Hierarchical Functions and Procedures 288
Other Functions and Procedures 306

D Appendix: Aggregation and Spread Types 327
Aggregation Types 327
Spread Types 329
Arithmetic Operators 329
Unary Operators 329
Binary Operators 330

E Appendix: Configuration of RPAS Extensions 331
Configuration of RPAS Extensions 331
About the RPAS Solutions Extension Framework 331
Launch from Navigation Tree 332
Module Tasks 332
Module Steps 333
Launch on Home Page 334
Launch In-Context of a Worksheet 334

F Appendix: RPAS Configuration Manager and rpasConfigMgr 337
Using the rpasConfigMgr 337
rpasConfigMgr Process 337
Diff Process 338
Merge Process 339
diffAndMerge Process 340
rpasConfigMgr Usage 341
RPAS Configuration Manager 342

Merge Functionality....344
Conflict Resolution Functionality....344
RPAS Configuration Manager Application....346
Merge Operation....346
A Note on Saving and Loading Merge Operations....348
Change Report Operation....349
G Appendix: Dynamic Hierarchies....351
Dynamic Hierarchies Overview....351
Domain Modified Dimensions....351
Multiple Domain Modified Dimensions in Single Workbook....353
Multiple Domain Modified Dimensions in Single Workbook....353
Domain Modified Dimensions Dependent on Multiple Dimensions....353
Multiple Dimension Notes....354
H Appendix: RPAS Rule Writing Tips....355
RPAS Rule Writing Tips Overview....355
Basic RPAS Rules Information....355
Full and Incremental Evaluation Modes....355
Rule Group Transitions....356
NA Values and Iterators....356
Principles for Writing Efficient Rules....357
Expensive Functions, Modifiers, and Procedures....357
Caching Intermediate Results....357
Automatic Caching of Expression Phrases....358
Tips....358
Rule Groups....359
Non-materialized Measures....359
Display-Only Non-materialized Measures....359
The If Statement....359
Expression Iteration Examples....360
Tips to Design Efficient RPAS Expressions....362

Send Us Your Comments

Oracle Retail Predictive Application Server, Configuration Tools, Release 14.1

Oracle welcomes customers' comments and suggestions on the quality and usefulness of this document.

Your feedback is important, and helps us to best meet your needs as a user of our products. For example:

- Are the implementation steps correct and complete?
- Did you understand the context of the procedures?
- Did you find any errors in the information?
- Does the structure of the information help you with your tasks?
- Do you need different information or graphics? If so, where, and in what format?
- Are the examples correct? Do you need more examples?

If you find any errors or have any other suggestions for improvement, then please tell us your name, the name of the company who has licensed our products, the title and part number of the documentation and the chapter, section, and page number (if available).

> **Note:** Before sending us your comments, you might like to check that you have the latest version of the document and if any concerns are already addressed. To do this, access the Online Documentation available on the Oracle Technology Network Web site. It contains the most current Documentation Library plus all documents revised or released recently.

Send your comments to us using the electronic mail address: retail-doc_us@oracle.com

Please give your name, address, electronic mail address, and telephone number (optional).

If you need assistance with Oracle software, then please contact your support representative or Oracle Support Services.

If you require training or instruction in using Oracle software, then please contact your Oracle local office and inquire about our Oracle University offerings. A list of Oracle offices is available on our Web site at www.oracle.com.

Preface

This guide describes the Predictive Application Server user interface. It provides step-by-step instructions to complete most tasks that can be performed through the user interface.

Audience

This User Guide is for users and administrators of Oracle Retail Predictive Application Server. This includes merchandisers, buyers, business analysts, and administrative personnel.

Related Documents

For more information, see the following documents in the Oracle Retail Predictive Application Server Release 14.1 documentation set:

- *Oracle Retail Predictive Application Server Installation Guide*
- *Oracle Retail Predictive Application Server Release Notes*
- *Oracle Retail Predictive Application Server Operations Guide*
- *Oracle Retail Predictive Application Server Online Help*

Customer Support

To contact Oracle Customer Support, access My Oracle Support at the following URL:

https://support.oracle.com

When contacting Customer Support, please provide the following:

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

Review Patch Documentation

When you install the application for the first time, you install either a base release (for example, 14.1) or a later patch release (for example, 14.1.1). If you are installing the base release or additional patch releases, read the documentation for all releases that have occurred since the base release before you begin installation. Documentation for patch releases can contain critical information related to the base release, as well as information about code changes since the base release.

Improved Process for Oracle Retail Documentation Corrections

To more quickly address critical corrections to Oracle Retail documentation content, Oracle Retail documentation may be republished whenever a critical correction is needed. For critical corrections, the republication of an Oracle Retail document may at times **not** be attached to a numbered software release; instead, the Oracle Retail document will simply be replaced on the Oracle Technology Network Web site, or, in the case of Data Models, to the applicable My Oracle Support Documentation container where they reside.

This process will prevent delays in making critical corrections available to customers. For the customer, it means that before you begin installation, you must verify that you have the most recent version of the Oracle Retail documentation set. Oracle Retail documentation is available on the Oracle Technology Network at the following URL:

`http://www.oracle.com/technetwork/documentation/oracle-retail-100266.html`

An updated version of the applicable Oracle Retail document is indicated by Oracle part number, as well as print date (month and year). An updated version uses the same part number, with a higher-numbered suffix. For example, part number E123456-**02** is an updated version of a document with part number E123456-**01**.

If a more recent version of a document is available, that version supersedes all previous versions.

Oracle Retail Documentation on the Oracle Technology Network

Documentation is packaged with each Oracle Retail product release. Oracle Retail product documentation is also available on the following Web site:

`http://www.oracle.com/technetwork/documentation/oracle-retail-100266.html`

(Data Model documents are not available through Oracle Technology Network. These documents are packaged with released code, or you can obtain them through My Oracle Support.)

Documentation should be available on this Web site within a month after a product release.

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."

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

1
Introduction

Overview

The Oracle Retail Predictive Application Server (RPAS) Configuration Tools provide a flexible means to configure and build RPAS-based applications with retailer-specific business parameters. The configuration tools provide a streamlined, user-friendly interface to leverage 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 through 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 accessed represents 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 *Oracle Retail Predictive Application Server Administration Guide for the Fusion Client*, the *Oracle Retail Predictive Application Server Administration Guide for the Classic Client*, and the *Oracle Retail Predictive Application Server Installation Guide*.

Configuration Tools Business Process

1. Set up system properties
2. Create a project
3. Create solutions
4. Configure hierarchies and dimensions
5. Configure measures and measure components
6. Configure rules sets, rule groups, and rules
7. Configure workbooks, workbook tabs, and worksheets
8. Configure wizards
9. Define interfaces used to import data
10. Build an RPAS domain
11. Configure domain integration with an RPAS Data Mart (Optional)

Sample Configurations

Some examples in this document use the sample configuration, which is delivered with the RPAS platform and can be installed along with the RPAS software and Configuration Tools. The sample configuration may not match every illustration because RPAS software and Configuration Tools software versions might vary between users. The examples are meant to provide a context to the reader. For information about the sample configuration provided with the RPAS platform, refer to the *Oracle Retail Predictive Application Server Installation Guide*.

Using the Configuration Tools Online Help

This Help site provides step-by-step procedures and other information about using RPAS Configuration Tools. We have implemented some tools to assist your navigation of this Help site. This page explains these tools.

About the Online Help

The online Help system uses JavaScript for some of its functionality. Make sure you have enabled JavaScript for your Web browser. Refer to the online Help in your Web browser for instructions on enabling JavaScript.

Formatting Conventions

This section provides information about the documentation conventions used in the online Help.

> **Note:** Notes are displayed using this convention. Notes contain additional information about the process or procedure that you are performing.

Navigate: The navigation sections of a procedure provide information about how to access the window that is the starting point of a procedure.

Navigate the Online Help

This Help site provides several ways for you to navigate to your topic.

Use the Table of Contents

The table of contents is the most common way that you will navigate to your topic.

1. Select the **Table of Contents** tab to display the table of contents on the left side of your screen.
2. Select the **+** sign in front of a book to expand it and view the topics.
3. Select a topic from the table of contents to view it.

Using the Search Feature

Use the search feature to explore the contents of your topics and find matches to queries that you define. There are some basic rules for making queries in full-text searches.

- You can type your search in uppercase or lowercase characters. Searches are not case sensitive.
- You can search for any combination of letters (a-z) and numbers (0-9).
- Punctuation marks such as the period, colon, semicolon, comma, and hyphen are ignored during a search.

- Group the elements of your search using double quotes or parentheses to set apart each element.
- You cannot search for quotation marks.

Use the following procedure to search the online Help:

1. Select the **Search** tab to display the search feature on the left side of your screen.
2. In the Search field, enter the word or words that you want to find.
3. Press the **Enter** key. Topics that match your search criteria display in the left pane.
4. Select a topic to view it.

Using the Business Process

The business process typically provides links to procedures that you need to perform to complete a task. You can select any link in the business process to view that topic.

Using the Index

Some Help sites may have an index. The index provides another way for you to navigate to information. There are two ways to use the index to search.

Browse the Index Entries

1. Select the **Index** tab. Words and phrases that are listed in the index display in alphabetical order.
2. Scroll up or down to find a word or phrase.
3. Select the word or phrase to view additional information.

Search the Index

1. Select the **Index** tab. Words and phrases that are listed in the index display in alphabetical order.
2. In the keyword field, type the word or phrase. Words and phrases that match your entry are displayed.
3. Select the word or phrase to view additional information.

Using Links

There may be two different types of links in this online Help. Select the link type below to learn more about it.

- Some topics contain hyperlinks that open a new page.
- Many topics have information that appears using a drop-down text link.

Drop-down text typically provides additional steps or sub-steps for a process or procedure and displays under the linked word or phrase.

- Select the link once to view the text.
- Select the link again to hide the text.

Using Hyperlinks

Hyperlinks bring you to another page in the online Help or to a Web page on the Internet. There are two things to remember when using hyperlinks:

- Hyperlinks always display a brief description of where the hyperlink takes you.
- If your browser controls are turned off, follow these steps to return to the previous page:

1. Display the shortcut menu by perform one of the following actions:

- Right-click with your mouse.
- Press the Application key.

2. From the shortcut menu, select **Back**. The previous page appears.

Navigating the Configuration Tools

Starting the Program

Once the Configuration Tools are installed, it can be accessed from the following default location by selecting **Start – Program Files – Oracle – RPAS – Configuration Tools**.

The following is an example of a location for the executable file:

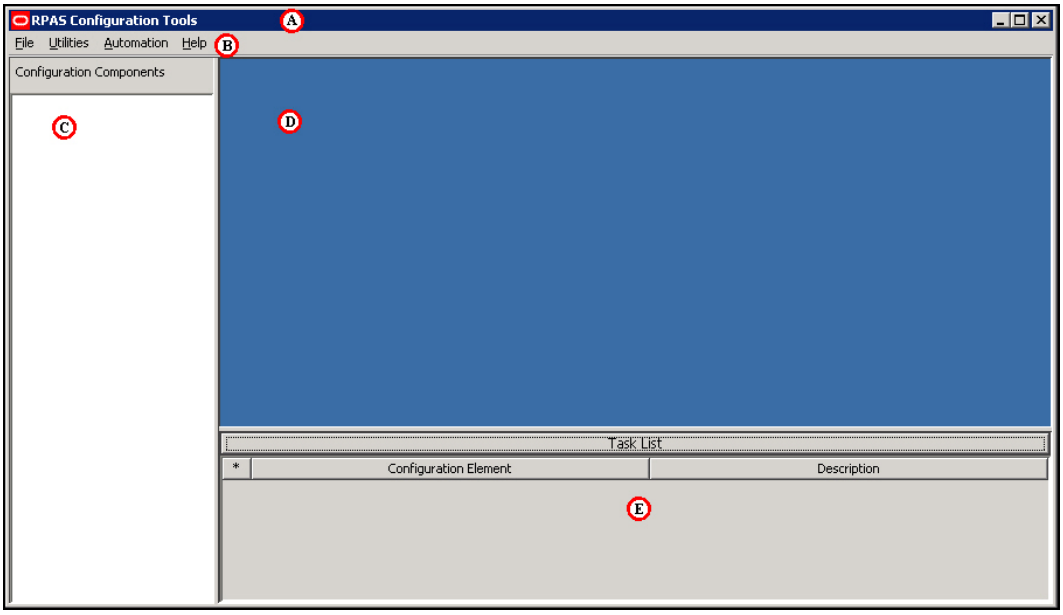
c:\Oracle\RPAS\ConfigTools\bin\ConfigTools.exe

Shortcuts may be created here and placed wherever they provide convenient access.

About the Configuration Tools Windows

All tasks are performed through the RPAS Configuration Tools window, which provides the following features:

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



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

Element	Purpose
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 tool
Configuration Components (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 Components pane, the related windows are displayed in the workspace.
Task List (E)	▪ Displays errors and warnings within the configuration

A Note about RPAS Configurability and Extensibility

RPAS is a configurable and extensible platform providing a significant amount of flexibility in design and implementation. Lack of specific verbiage around a platform component does not guarantee a commitment to support its use in a non-documented way. In case of any ambiguity in the documentation, the customer is expected to ask for clarification with Oracle Customer Support.

2

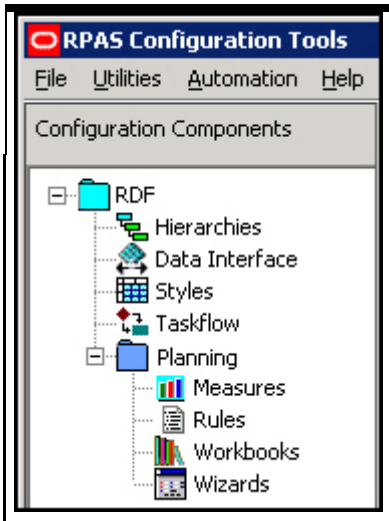
Configuration Components Pane

Know the Configuration Components

The Configuration Components pane 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 Components pane 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.

As a new configuration is created the configuration administrator assigns a name to the configuration, the project, and the solution. The configuration administrator can drill down through the configuration to work in specific areas. The icons in the Configuration Components assist the configuration administrator to intuitively navigate features within the Configuration Tools. If an area of the configuration has been modified, its icon will contain a modification flag icon. The configuration must be saved if the modifications are to be retained.



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

	Wizards	Wizard Designer window

Configuration Components Pane Overview

Projects

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 the project. The RPAS Configuration Tools allows for multiple projects to be viewed and modified (the limit is 3 projects).

Solution

Each solution represents a grouping of measures, rules, and workbooks to support a business process as defined by the retailer. A project may have multiple solutions and a solution may use a subset of the hierarchies and dimensions defined within the project.

Hierarchy

The user may access the Hierarchy Definitions window by selecting the **Hierarchy** icon in the Configuration Components pane. 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 selecting the **Data Interface** icon in the Configuration Components pane. 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 `loadmeasure` utility is used to load data for a measure.

Styles

The user may access the Style Definition window by selecting the **Styles** icon in the Configuration Components pane. 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.

Taskflow

The user can access the Taskflow Manager window by selecting the **Taskflow** icon in the Configuration Components pane. Users can use the Taskflow tool to configure an activity taskflow for use within the RPAS Fusion Client. This Taskflow directs users through various steps that have been setup to help them accomplish their business process. Taskflows contain activities that the user needs to complete. Each activity contains one or more tasks, which are mapped to certain workbooks. Within each task are one or more steps that are mapped to worksheets within the task's workbook. The activities can also be organized in an activity group. An activity group is a single, integrated taskflow that represents a business process and can include activities from multiple solutions.

Users can use the preconfigured taskflow that is delivered with the RPAS solution, modify that preconfigured taskflow, or configure a customized one.

> **Note**: The taskflow is used only for the RPAS Fusion Client. It does not affect the RPAS Classic Client.

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 retailer's business process.

Rule Sets, Rule Groups, and Rules

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: ReceiptUnits = ReceiptValue / ReceiptPrice
- Expression 2: ReceiptValue = ReceiptUnits * ReceiptPrice
- Expression 3: ReceiptPrice = ReceiptValue / 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 set is a collection of rule groups that is used for organizational purposes by the Configuration 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 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 will ask the user to select the SKUs to include in the workbook. The second will ask the user to select the stores to include in the workbook, and the third wizard will 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.

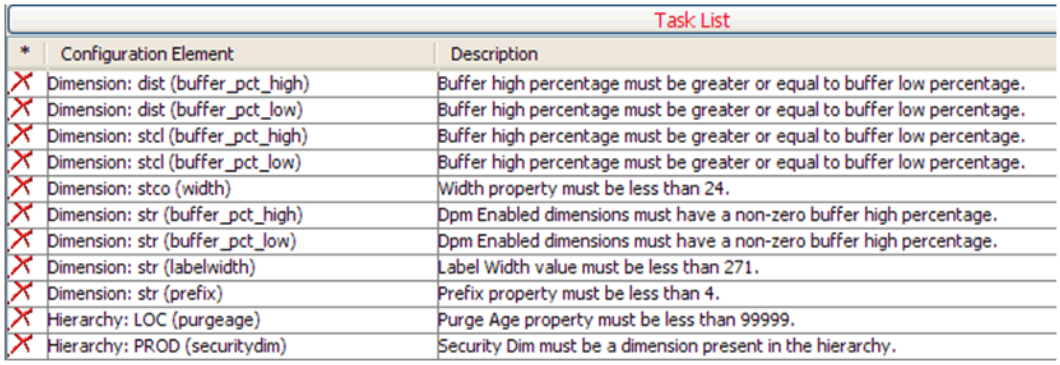
The Task List title bar serves as a status indicator. If the title “Task List” is displayed in red, the Task List contains items that need the user’s attention. If there are no errors or warnings, the title will be displayed in black.

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:

- The first column indicates the nature of the Task List item.
  - Indicates an error.
  - Indicates a warning.
- The second column identifies the configuration element involved.
- The third column provides a detailed description of the issue.

Below is a sample task list.



Task List

*	Configuration Element	Description
✗	Dimension: dist (buffer_pct_high)	Buffer high percentage must be greater or equal to buffer low percentage.
✗	Dimension: dist (buffer_pct_low)	Buffer high percentage must be greater or equal to buffer low percentage.
✗	Dimension: stcl (buffer_pct_high)	Buffer high percentage must be greater or equal to buffer low percentage.
✗	Dimension: stcl (buffer_pct_low)	Buffer high percentage must be greater or equal to buffer low percentage.
✗	Dimension: stco (width)	Width property must be less than 24.
✗	Dimension: str (buffer_pct_high)	Dpm Enabled dimensions must have a non-zero buffer high percentage.
✗	Dimension: str (buffer_pct_low)	Dpm Enabled dimensions must have a non-zero buffer high percentage.
✗	Dimension: str (labelwidth)	Label Width value must be less than 271.
✗	Dimension: str (prefix)	Prefix property must be less than 4.
✗	Hierarchy: LOC (purgeage)	Purge Age property must be less than 99999.
✗	Hierarchy: PROD (securitydim)	Security Dim must be a dimension present in the hierarchy.

Sample Task List

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 through a right-click menu.

How RPAS Uses Solution Configurations

RPAS uses the following components for 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) correspond 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 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:

- Hierarchical relationships that require aggregation and spreading
- 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.

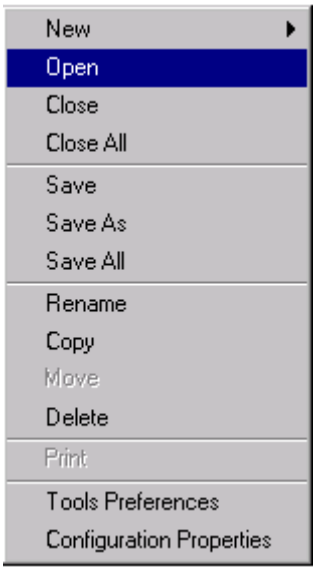
Note:

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

See the Appendix: Rules Function Reference Guide for details about standard RPAS functions .

Right-Click Menus in the Configuration Components Pane

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

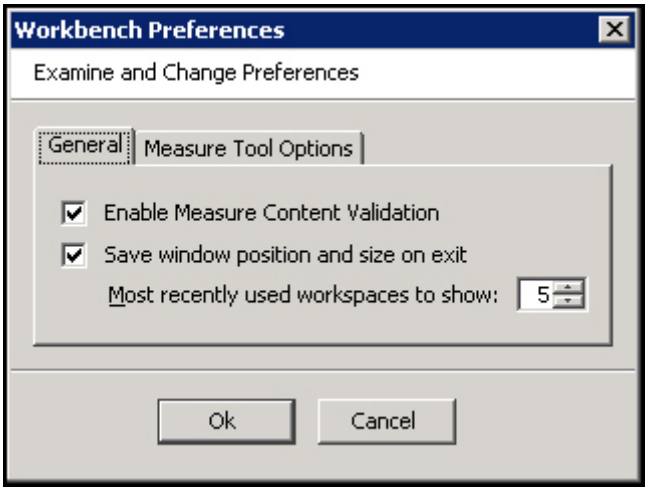


Example: Right-click Menu

Setting Tools Preferences

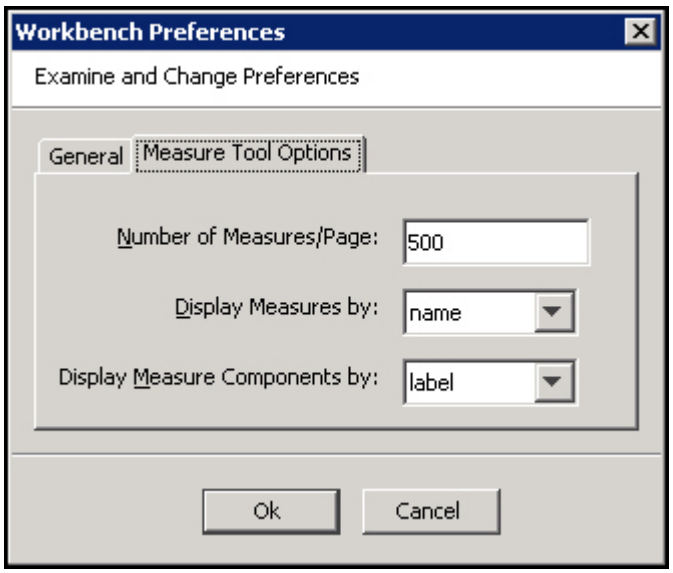
Settings made here will apply to all configuration projects created or viewed with the tool.

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.

3

Projects

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, styles, and taskflow are defined within a project and are available for use within all solutions in a given project.

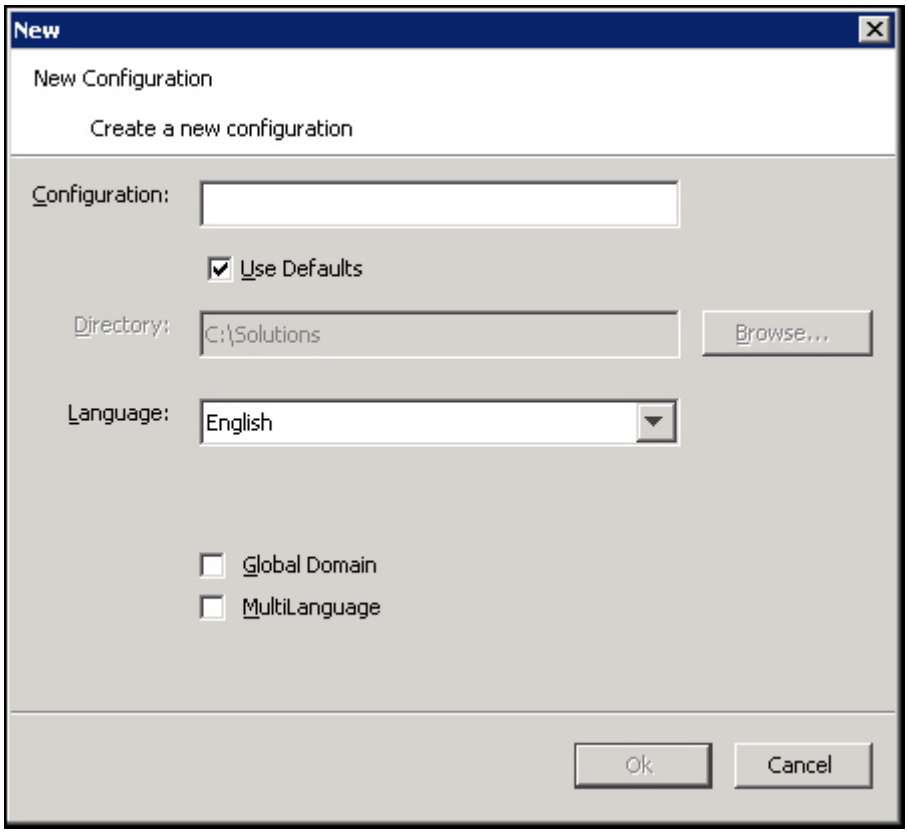
> **Note:** A solution can use a subset of the hierarchies and dimensions defined within the project. Within a project, you can additionally define certain properties (which are part of the Data Interface Tool) that describe how measures will be loaded into the domain.

Hierarchies are "domain-specific," which means that they are defined at the project (domain) level and can be used by all solutions that are defined within that project (domain). There is no requirement that each solution use all of the hierarchies defined in the project.

For example, a project may contain five hierarchies and three solutions, but each solution might only use four of those hierarchies in the base intersections of its measures, so even though five hierarchies exist, each solution may not use all of them.

Create a Project

Navigate: From the File menu, select **New** – **Project**, or right-click in the Configuration Manager and select **New** – **Project**. The New dialog box appears.



New Dialog Box

1. In the **Configuration** field, enter the name of the configuration/project.

   The Use Defaults option under the Configuration field points the user to a default path for storing the configuration. Deselect the Use Defaults option to enable the user to navigate to the appropriate directory using the Browse button.
2. Select the default language for the domain in which this configuration will be used to create. The default is English.
3. Select the options for Global Domain and MultiLanguage as necessary. The possible settings for these boxes are as follows:
   - **Global Domain** – Selecting this option overrides the default setting of "Simple Domain." A Global Domain allows the user to create workbooks from multiple domains and to administer and update multiple domains from a single master domain. Whether a domain should be Global is a technical decision that must be made with the consultation of Oracle Services. This setting cannot be changed after the domain is built.
   - **MultiLanguage** – If this option is selected the resulting domain will be enabled to support multiple languages. Multi-lingual domains allow for most data elements (measures, labels, and so on) in an RPAS domain to be translated into other languages. More information on Multi Language support can be found in the "Translation Administration" chapter of the *Oracle Retail Predictive Application Server Administration Guide*.

Note: The Global Domain and Multi-Language settings must be defined before the domain is built. Changes to the Global Domain and MultiLanguage properties are ignored when modifying the configuration of an existing domain.

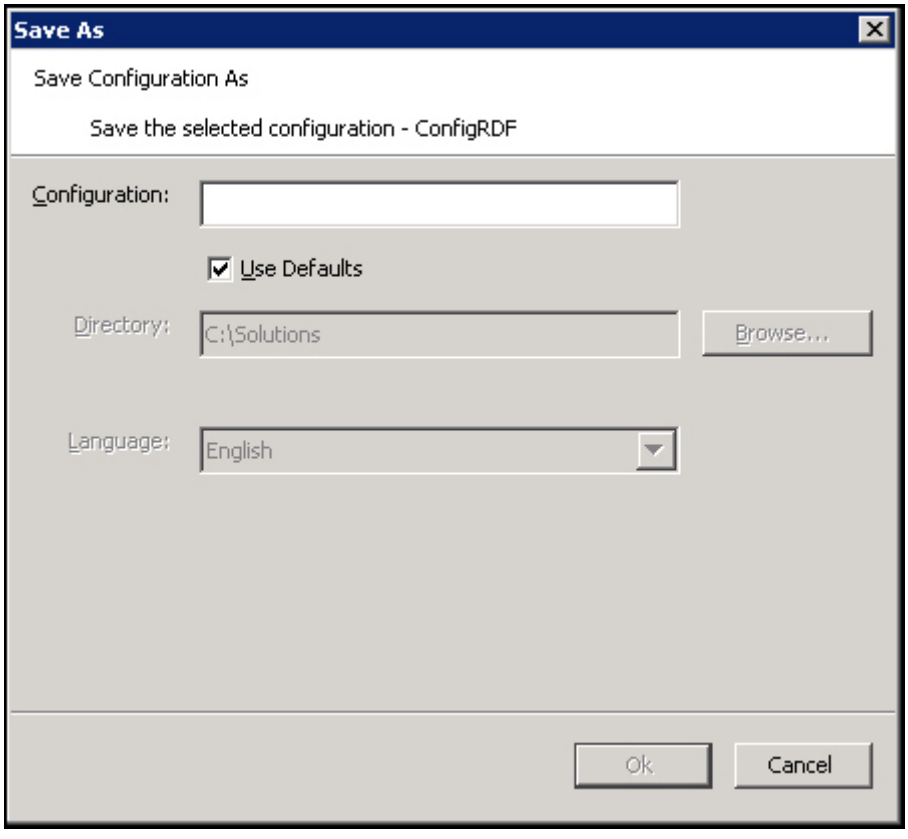
4. Click **OK** to save any changes and close the window.

Save Changes to a Project

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

Using the Save As Option to Save a Project using a Different Name

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



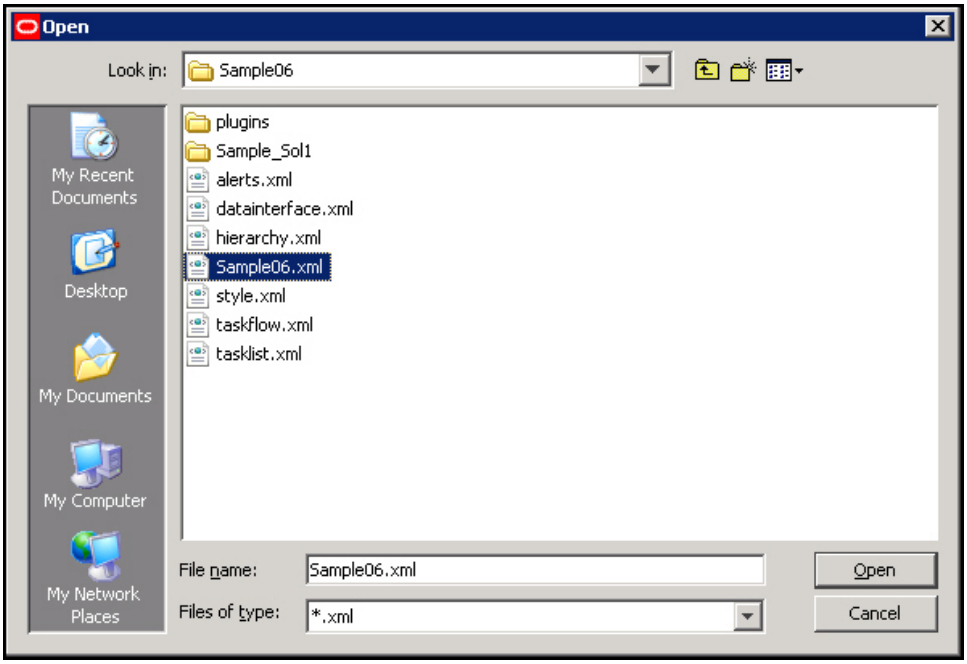
Save As Dialog Box

1. In the **Configuration** field, enter the new project name.
2. If **Use Defaults** is selected, click **OK** to save the project to the path displayed in the **Directory** field. If **Use Defaults** is selected and you want to save the project to a different location, deselect **Use Defaults** and either enter the appropriate path in the **Directory** field, or click **Browse** to navigate to the appropriate location where you want the project saved.
3. Click **OK** to save the project.

Open an Existing Project

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

1. Choose one of the following methods:
   - 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 appears in the Configuration Components pane.



Open Dialog Box

   - To open a project that was recently opened, select the project from the recently used projects list in the File menu. These projects appear as a numbered list where the most recently used project is first in the list. Select the project to open in the Configuration Components pane. The number of projects that appears in the most recently used list may be changed from the Workbench Preferences dialog box, which is accessed by selecting **Tools Preferences** from the **File** menu.

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

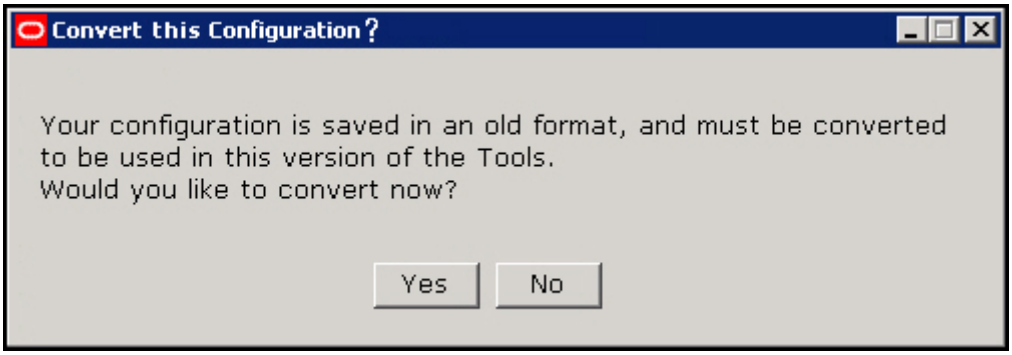
If you attempt to open a project saved in a previous version of the RPAS Configuration Tools, a dialog box may appear which allows you to convert the configuration to the new version.

Navigate: From the File menu or right-click from the Configuration Components pane, 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 opens in the Configuration Components pane.
   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

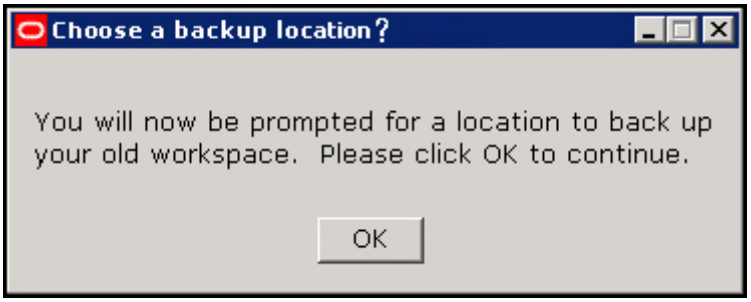
opens in the configuration manager. The number of projects that appears in the most recently used list may be changed by using the **Tools Preferences** option of the File menu.

A message box appears:



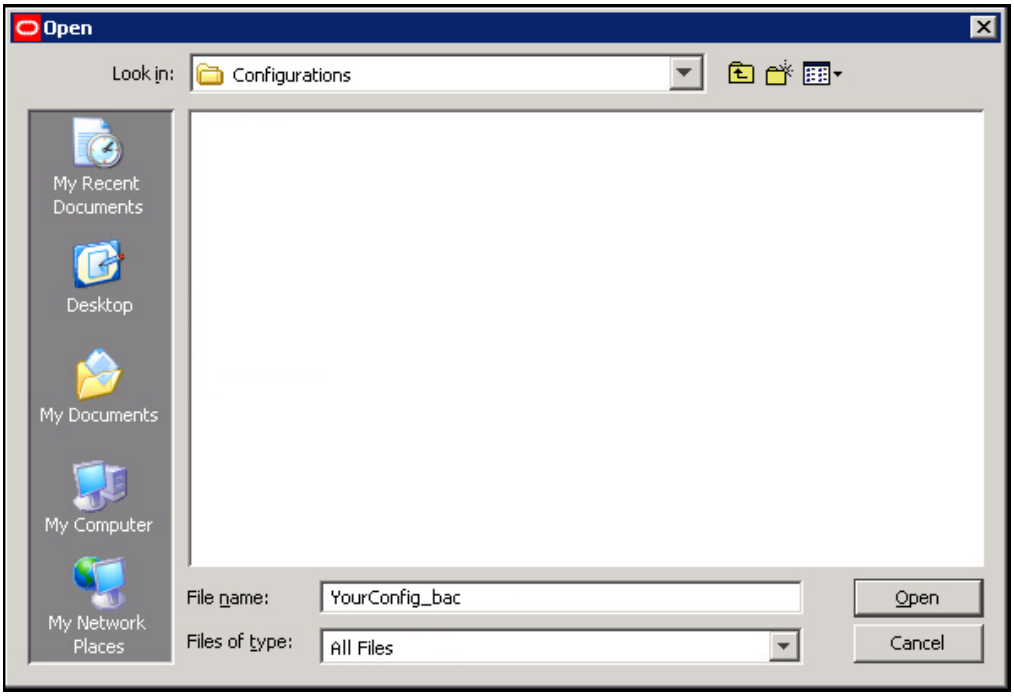
Convert This Configuration Message Box

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 **Yes** is selected, the Choose a backup location message box appears.



Choose a Backup Location Message Box

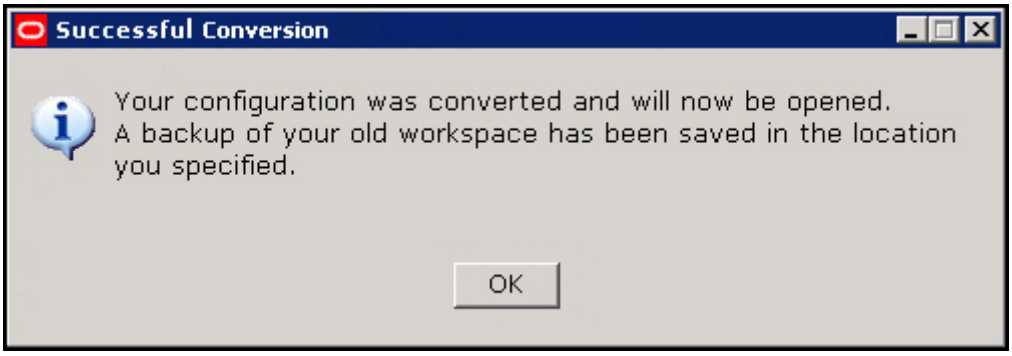
3. Click **OK**. The Open dialog box appears and prompts you for a location to store a backup of the project that is to be converted.



Open Dialog for Saving Configuration Backup

Note: You may either rename the original configuration that is to be backed up or specify a new directory to store the original.

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

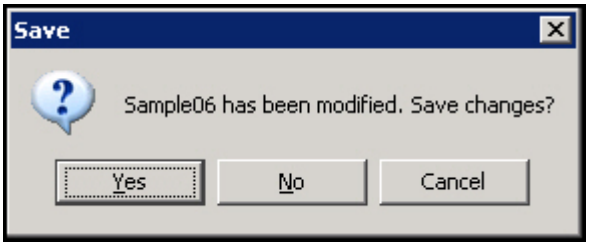


Successful Conversion Dialog Box

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 Components pane, select **Close**. If no changes were made to the Project, the project will be closed. If changes were made, the Save dialog box appears.



Close Project

Perform one of the following options:

- 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 elements of the same type. Hierarchies provide a means to define relationships between dimensions (aggregates, roll-ups, and alternate roll-ups) and groups belonging to the same entity (for example, Time = years, months, weeks, and days).

The following hierarchies are automatically created and cannot be deleted within the Configuration Tools:

- CLND (Calendar)
- PROD (Product)
- LOC (Location)
- ADMU (User)
- LNGS (Languages)

These hierarchies are required by RPAS-based solutions and cannot be removed, but additional hierarchies can be added to support the required business process.

Hierarchies define the path of data aggregation and spreading. In a workbook, the configuration administrator can view data at any required level of detail by drilling down or rolling up through dimensions in the hierarchy.

> **Note:** ADMU and LNGS are not configurable hierarchies; therefore, they cannot be created or modified. ADMU and LNGS are built by RPAS, and the RPAS Configuration Tools makes them available for use in configurations. ADMU is the user hierarchy, and it exists to allow a measure to use the user dimension as part of its base intersection. LNGS is the language hierarchy, and it exists to support translation in multi-language domains. It is also available so that you can create a measure with the language dimension as part of its base intersection.

You can create and define dimensions for each of these hierarchies and for any additional hierarchies that are added to the project.

Note: The names for the automatically generated hierarchies (CLND, PROD, LOC, ADMU, and LNGS) cannot be changed, but the default user labels for CLND, PROD, and LOC (Calendar, Product, and Location) can be changed. The user labels of ADMU and LNGS cannot be changed.

The CLND, ADMU, and LNGS hierarchies must exist in all domains, but PROD and LOC are not mandatory. If there are no dimensions created for the PROD and LOC hierarchies, the hierarchies are not created in the resulting domain.

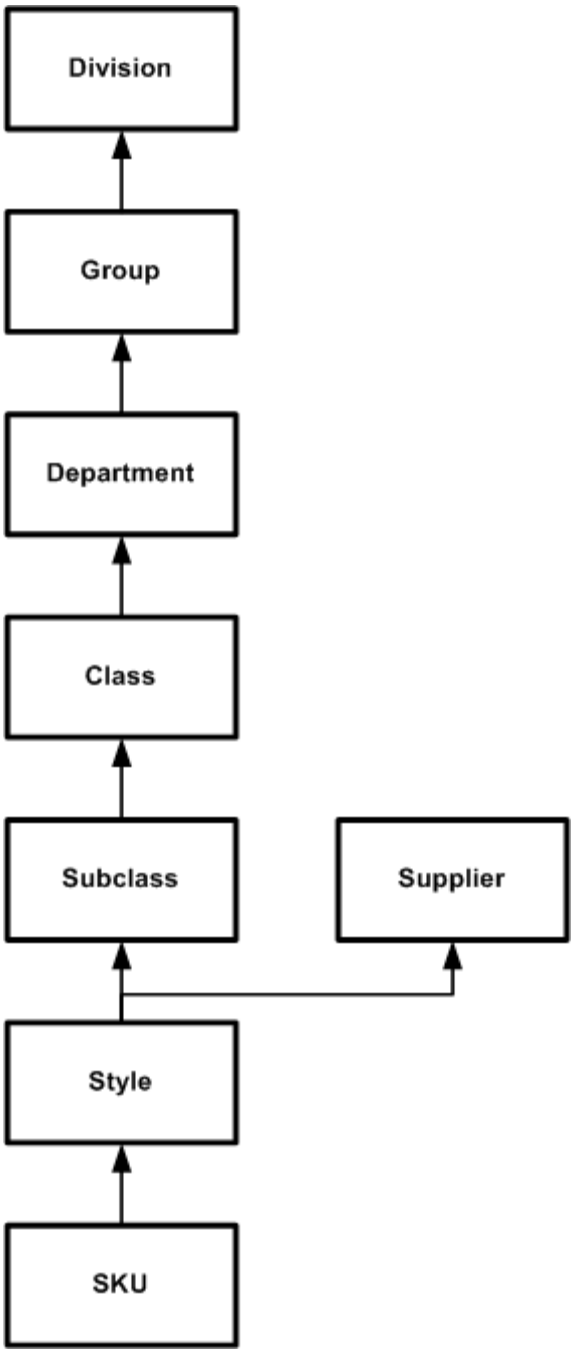
RPAS does not impose any limit on the number of hierarchies that can be configured in a project.

The Hierarchy Definition Window

The Hierarchy Definition window allows you to define and construct hierarchies, dimensions for each hierarchy, and the relationships between dimensions. It also offers the following features:

- 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.

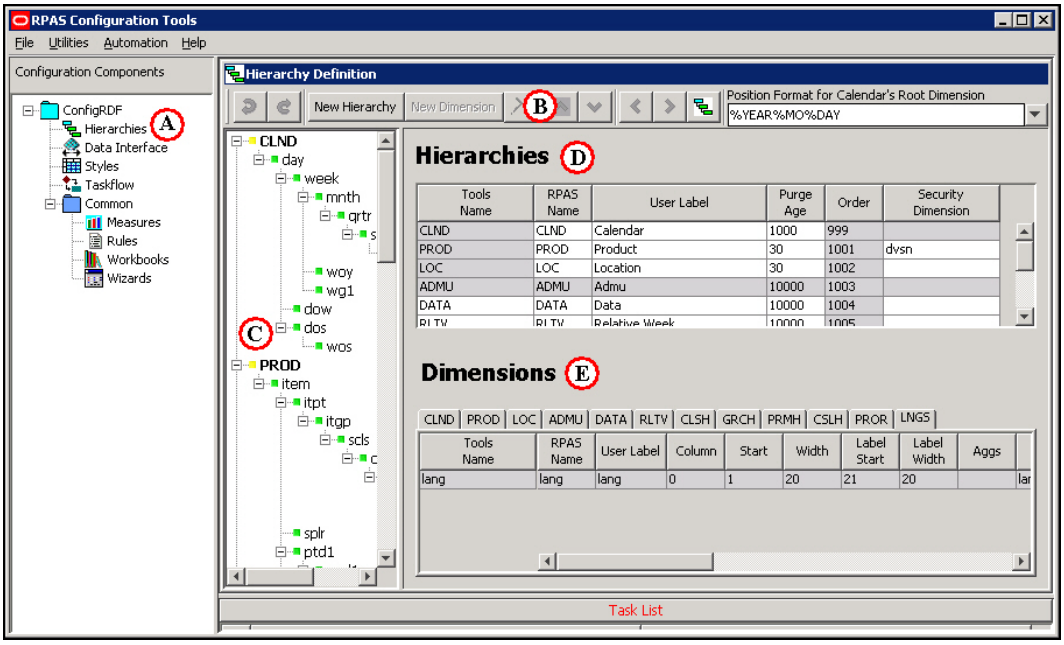


Example of Product Hierarchy

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.

About the Hierarchy Definition Window

To access the Hierarchy Definition window, select **Hierarchies** (A) from the Configuration Components pane. The Hierarchy Definition window appears in the workspace.



Example of Hierarchy Definition Window

The Hierarchy Definition window contains the following elements:

- The Hierarchy Definition toolbar (B) - This toolbar displays options that can be performed. Buttons are enabled or disabled based on the item selected on screen.
- The Hierarchy navigation tree (C) - The navigation tree provides a visual representation of your hierarchies. Bold elements at the top of the tree structure represent the hierarchies. The items listed below each bolded hierarchy are the dimensions defined in that hierarchy. Click the plus sign (+) or minus sign (-) to expand the tree. The Hierarchy tree is also used to select a hierarchy or hierarchy dimension. Once an item is selected, you can modify its properties from the **Dimension** region in the Hierarchy Definition window. The Hierarchy navigation tree also provides a context menu when you right-click a tree item. The available options in the context menu depend on whether a hierarchy or dimension is selected. This context menu can be used to create a new hierarchy or dimension at the selected level. It also allows you to rename the selected item. When an item is renamed from the tree, it is the **Tools Name** that is being modified, which appears in the **Dimensions** region of the window.
- The Hierarchies region (D) - This area displays the defined hierarchies and their properties.
- The Dimensions region (E) - This area contains hierarchy tabs and allows you to define the dimension properties for your hierarchies. The tabs represent the hierarchies defined. Select the appropriate hierarchy tab to display its dimensions and modify dimension properties.

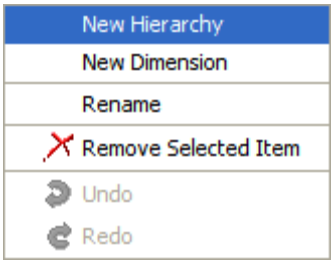
Gray fields in the Hierarchy Definition window indicate fields that cannot be modified. Any elements that appear in red indicate problems or issues must also appear in the Task List pane along with a brief description of the issues identified.

Working with Hierarchies

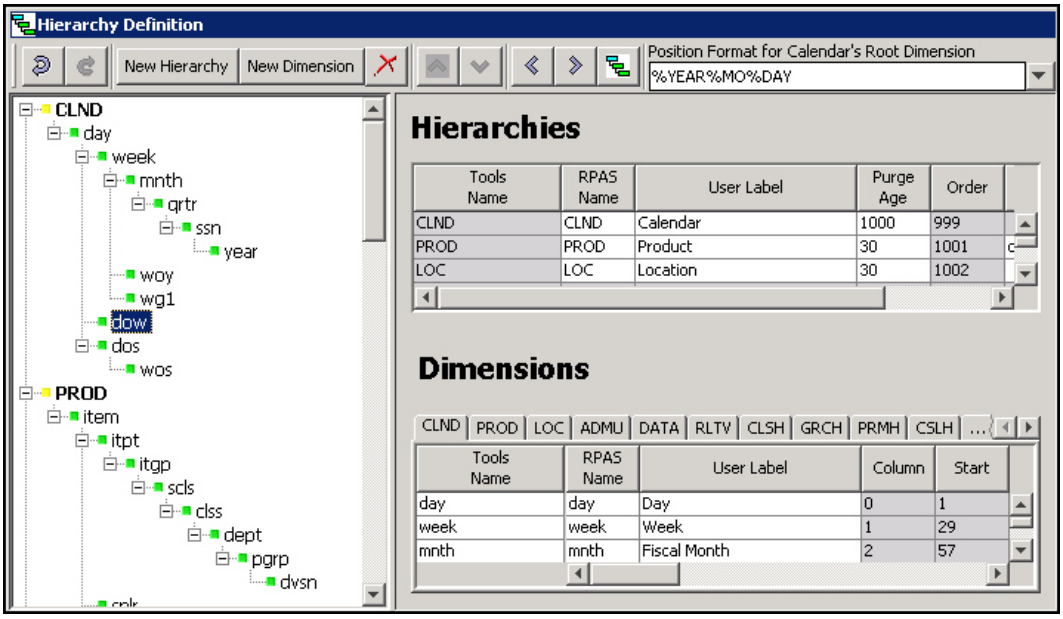
Create a New Hierarchy

When a new project is created, following default hierarchies are automatically created: Calendar (CLND), Product (PROD), Location (LOC), User (ADMU), and Languages (LNGS). Additional hierarchies and dimensions can be created to meet your business needs.

Navigate: Select **New Hierarchy** from Hierarchy Definition toolbar, or from the Hierarchy Definition tree, right-click and select **New Hierarchy** from the menu.



Note: If multiple projects are open, make sure you are working from the desired project before adding a new hierarchy.



Hierarchy Definition Window

1. To change the Tools Name of the newly created hierarchy in the Hierarchy navigation tree of the Hierarchy Definition window, choose one of the following methods:
   - Right-click on the hierarchy name, and select **Rename**.
   - Double-click the hierarchy name.
2. Enter the new name.

Note: The RPAS name can only be up to four (4) characters long.

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

Specify Hierarchy Properties

Hierarchy properties are defined from the Hierarchies region on the Hierarchy Definition window.

Tools Name	RPAS Name	User Label	Purge Age	Order	Security Dimension
CLND	CLND	Calendar	10000	999	
PROD	PROD	Product	10000	1001	
LOC	LOC	Location	10000	1002	
ADMU	ADMU	Admu	10000	1003	
LNGS	LNGS	Lngs	10000	1004	

Hierarchy Properties Window

From this location you can modify the following hierarchy properties:

- **Tools Name** – The name of the hierarchy that appears within the RPAS Configuration Tools. This field is less restrictive than the RPAS Name field, allowing you to view and select a meaningful label for hierarchies and dimensions while working with the configuration rather than using the RPAS Name.
- **RPAS Name** – The RPAS internal name of the hierarchy. This hierarchy name is used only by RPAS (not the user) within the domain.

  Note: The RPAS Name of a hierarchy cannot be edited if it is shaded gray; however, you can change other properties, such as User Label.

  CLND is always the innermost dimension. The order of the other hierarchies (PROD, LOC, and so on) can be changed.

- **User Label** – The hierarchy label that is displayed to RPAS users within the domain.
- **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 during a batch run (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. See the *Oracle Retail Predictive Application Server Administration Guides* for more information on the `loadHier` utility.

  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.

- To change the order of a hierarchy, select the hierarchy from the Hierarchies region or from the Hierarchy navigation tree and use the up/down buttons located on the Hierarchy Definition toolbar to move the hierarchy to the desired location.

  The hierarchy can also be arranged by dragging and dropping in the Hierarchy navigation tree. The order numbers are automatically changed and generated regardless of the utilized reordering technique.

  For performance reasons, the Calendar hierarchy (and therefore all of its dimensions) is always the "innermost" dimension and defaults to an unedited number of 999. 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. This is followed by all calendar positions for the second product and first location, and so on.

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

  Note: Certain RPAS-based solutions (such as Advanced Inventory Planning and Demand Forecasting) have additional hierarchies that are in a pre-defined order that should not be changed.

  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.

CLND is always the innermost dimension and ADMU is always the outermost dimension.

The order of the other hierarchies (PROD, LOC, and others created by the configuration administrator) can be changed.

- 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 Oracle Retail Predictive Application Server Administration Guide for additional information about position-level security.

Note: The Security Dimension hierarchy can be changed and patched. It will not adversely affect the results if changed.

Delete a Hierarchy

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

1. Select the hierarchy to delete.
2. Choose one of the following methods:
   - Click the **Delete** button. The hierarchy is removed.
   - Press the **Delete** key from your keyboard.
   - Use the right-click menu to select **Remove Selected Item**.

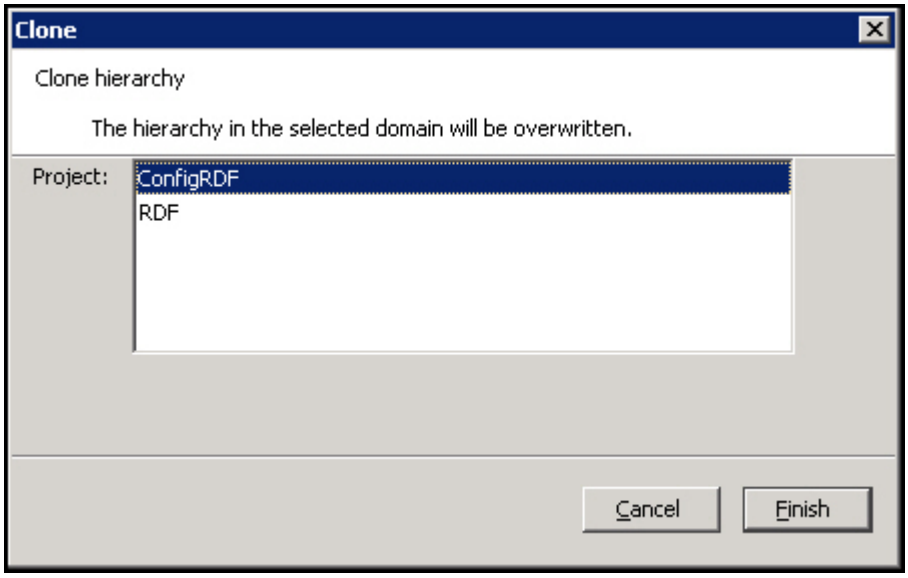
Note: The CLND, PROD, LOC, ADMU, and LNGS hierarchies CANNOT be deleted from the configuration.

Copy (Clone) Hierarchies

The RPAS Configuration Tools allows for the hierarchies of an existing project to be copied into a new or existing project.

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

1. Right-click **Hierarchies** in the Configuration Components pane, and select **Copy**. The Clone dialog box appears.



Clone Dialog Box

2. Select the destination project for the hierarchies to be copied.
3. Click **Finish**. The hierarchies in the selected project are overwritten.

> **Note:** Each project has a single set of hierarchies. Hierarchies can only be copied from one project to another, thus multiple projects must be open in the RPAS Configuration Tools before the copy process is initiated.

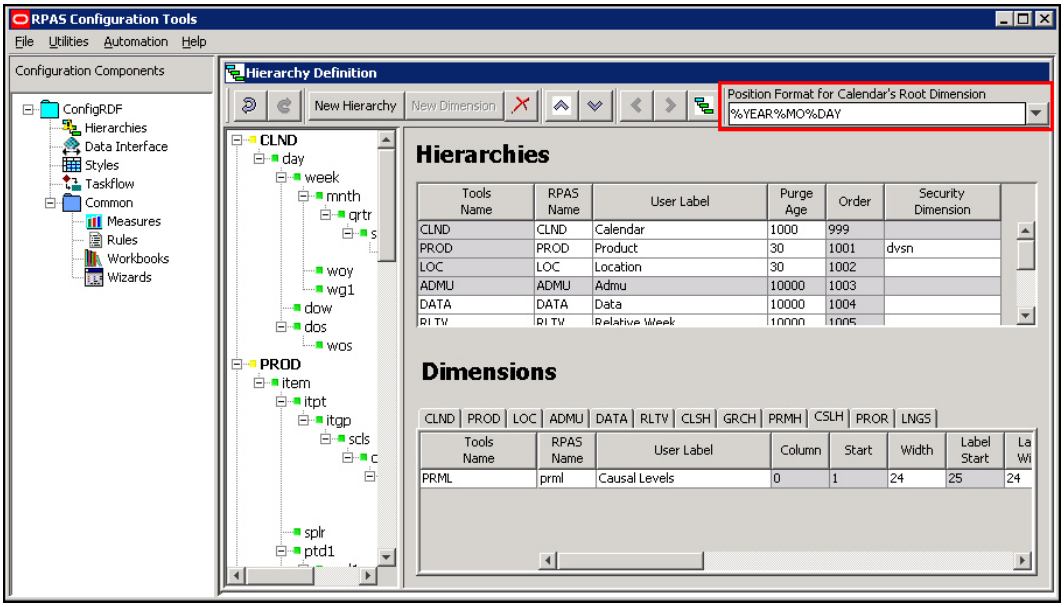
Working with Position Formats

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.

> **Note:** See "Appendix B - Calculation Engine Users Guide" and "Appendix C - Rules Function Reference Guide" for more information.

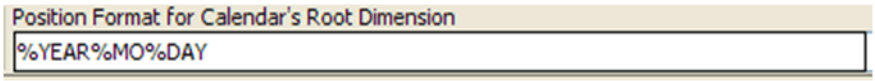
Specifying the Position Format

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



Example of Position Format in Hierarchy Definition Window

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



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 appropriate syntax. Refer to "Position Format Syntax" for more information.

Position Format Syntax

The Position Format field uses the following syntax conventions:

- %YEAR – Four digit Gregorian calendar year.
- %YR – Two east significant digits of the year (for example, 15 is 2015).
- %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, displays 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.

The length of the position name must 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 %YEAR%MONTH%DAY evaluates to a total of 15 characters (4 for the year, 9 for month, and 2 for day).

Examples:

- Format: %YEAR%MO%DAY
- A position that represents the January 31, 2013 would have the name 20130131
- Format: %YR%MON%DAY
- A position that represents the January 31, 2013 would have the name 13Jan31

Working with Dimensions

Overview

Dimensions are the components within a hierarchy that define the structure and roll up 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 Components pane, 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" level dimensions or "over" and "under," these terms are to be interpreted visually. The bottom level is at the top of the hierarchy, and the top levels are at the end of the hierarchy branches. 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 is the bottom level of a Calendar hierarchy, but it falls directly beneath CLND.

2. Choose one of the following methods:
   - From the Hierarchy Definition right-click menu, select **New Dimension**.
   - Click the **New Dimension** button on the toolbar.

   Create the first dimension, which becomes the root dimension, for a new hierarchy when positioned on the hierarchy. Once the root dimension is defined, new dimensions cannot be defined directly under the hierarchy. New dimensions are added under other dimensions. For example, after "Day" is added to the CLND hierarchy, CLND cannot be selected again to add "Hour." However, "Week" can be added under the "Day" dimension. There can only be one root dimension created 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 represents the day level. Such a dimension must be defined, although the user label can be changed from "day" if needed for localization purposes.

There is no limit on the number of dimensions that may be created for a hierarchy.

3. Define the dimension as necessary. Refer to "Defining Dimension Properties" for more information.

Defining Dimension Properties

Navigate: In the Configuration Components pane, select **Project – Hierarchies.**

1. Select a dimension using one of the following methods:
   - From the Hierarchy tree, select the dimension you want to modify.
   - From the **Dimensions** region of the Hierarchy Definition window, select the hierarchy tab that contains the dimension you want to define or modify.

Dimensions

CLND | PROD | LOC | ADMU | DATA | RLTV | CLSH | GRCH | PRMH | CSLH | PROR | LNGS

Tools Name	RPAS Name	User Label	Column	Start	Width	Label Start	
day	day	Day	0	1	8	9	2
week	week	Week	1	29	8	37	2
mnth	mnth	Fiscal Month	2	57	8	65	2
qrtr	qrtr	Fiscal Quarter	3	85	7	92	2
ssn	ssn	Fiscal Half	4	112	7	119	2
year	year	Fiscal Year	5	139	5	144	2
dow	dow	Day of Week	6	164	3	167	2
dos	dos	Day of Season	7	187	6	193	2

Example Dimensions Properties Window – CLND Tab Selected

Dimensions

CLND | PROD | LOC | ADMU | DATA | RLTV | CLSH | GRCH | PRMH | CSLH | PROR | LNGS

Tools Name	RPAS Name	User Label	Column	Start	Width	Label Start	
item	item	Item	0	1	20	21	2
itpt	itpt	Parent	1	271	20	291	2
itgp	itgp	Grandparent	2	541	20	561	2
scls	scls	Subclass	3	811	20	831	1
clss	clss	Class	4	951	20	971	1
dept	dept	Department	5	1091	20	1111	1
pgrp	pgrp	Group	6	1231	20	1251	1
dvsn	dvsn	Division	7	1371	20	1391	1

Example Dimensions Properties Window – PROD Tab Selected

2. In the Dimensions properties region, select or double-click in the field to edit. Scroll to the right to see all of the fields. You can resize the columns by placing the cursor over the column until the double-sided arrow appears and then drag the column to the desired width.

   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.

   You can specify the following dimension information:

   > **Note**: Only four of the dimension properties are patchable in an existing domain. Once the domain is built, changes made to any other dimension properties are ignored.
   >
   > If a dimension's property is patchable, it will be stated in a note in its description below.

   - **Tools Name** – The name of the dimension that is displayed within the RPAS Configuration Tools. This field allows you to assign meaningful labels while working with a configuration in the Configuration Tools. For example, the Tools Name appears in Select Intersection dialog box, making it easier for you to assign the appropriate intersections.
   - **RPAS Name** – The RPAS internal name of the dimension. This dimension name is used only by RPAS (not the user) within the domain.
   - **User Label** – The dimension description that is displayed to RPAS users in the RPAS Client.

     > **Note:** Any alpha-numeric characters are allowed. Single or double quotes are not 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.

- **Start** – This is a read-only, calculated field. This field identifies the start position of the position names for this dimension in the hierarchy load file.
- **Width** – This field identifies the width of position names for this dimension in the hierarchy load file.
- **Label Start** – This is a read-only, calculated field. This field identifies the start position of the position label for this dimension in the hierarchy load file. The sum of the dimension Start and Width fields determines the value of the Label Start.
- **Label Width** – This field identifies the width of position labels for this dimension in the hierarchy load file.

> **Note:** When using comma separated value (CSV) files to load data into RPAS, the following dimension are ignored: **Column**, **Start**, **Width**, **Label Start**, and **Label Width**. See the *Oracle Retail Predictive Application Server Administration Guides* for more information on Comma Separated Value (CSV) flat file format data load and export.

- **Aggs** – This field establishes the relationship of the dimension to the other dimensions in the same hierarchy. Specifically, this field references the child dimension that aggregates up to this dimension (the parent dimension). It can be edited by using the drop-down list, or you can drag and drop dimensions in the left hand side hierarchy pane.
- **Database** – This field identifies 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 dimensional positions.

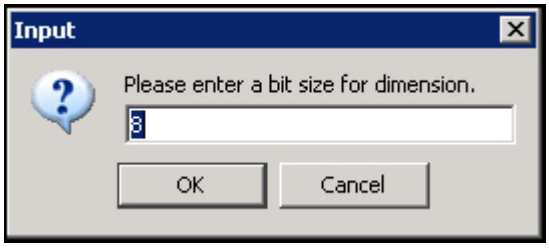
> **Note:** The database property for a dimension cannot be patched once the domain is built. It is therefore important to establish the databases to hold dimension information correctly prior to the creation of the domain.
>
> **Note:** The size of any RPAS database should be limited to ~2 GB for contention and performance purposes.

- **User Dimension** –When selected (checked), this field indicates that the dimension is user maintained. User-defined definitions (UDD) cannot have another dimension as a parent. 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. See the Oracle Retail Predictive Application Server Administration Guide for detailed instructions for enabling position label translation.

> **Note:** Enabling the translation of a dimension is patchable and will not adversely affect the expected results if altered. However, disabling translation is not patchable and may result in failure.

- **Cardinality** – Use this field to specify the approximate size of a dimension in terms of the number of positions the dimension is expected to contain. Based on the range you select, RPAS allocates a number of bits within its internal representation of the dimension and all measures that contain that dimension in their intersection. This ability to configure the amount of space necessary to represent the positions of a dimension will result in smaller domains. The range options are:
- Very small: between 1 and 100 positions (8 bits)

- Small: between 100 and 800 positions (10 bits)
- Medium: between 800 and 12,000 positions (14 bits)
- Large: between 12,000 and 100,000 positions (17 bits)
- Very Large: between 100,000 and 500,000 positions (20 bits)
- Extremely Large: between 500,000 and 2,000,000 positions (22 bits)
- Ultra Large: between 2,000,000 and 4,000,000 positions (23 bits)
- Custom: selecting this option allows you to enter a specify bit number. An input dialog box appears. Enter the bit number and click **OK**.



Input Dialog for Custom Cardinality Option

> **Note**: The cardinality property of a dimension is patchable. However, changes to the cardinality of a dimension will not take effect until the reindexDomain utility is executed on the domain after the dimension is patched.

- **Re-indexing Threshold** – Use this field to specify the reindexing threshold. As a dimension exhausts its supply of unused indices, RPAS recycles the deactivated positions and compacts the indices of a dimension. This operation is known as re-indexing. When reindexDomain is run, every dimension is analyzed to determine how many indices are still available to be assigned. If this amount falls below a certain number (the re-indexing threshold), that dimension is reindexed. By default, the threshold is set to 10% of the available positions in the dimension. This means that when more than 90% of a dimension's positions have been allocated, the dimension is reindexed. For more information about reindexing and the reindex utility, see the "Hierarchy Management" chapter of the *Oracle Retail Predictive Application Server Administration Guide for the Fusion Client* or the *Oracle Retail Predictive Application Server Administration Guide for the Classic Client*.

> **Note**: The re-indexing threshold property of a dimension is patchable. However, changes to the re-indexing threshold of a dimension will not take effect until the reindexDomain utility is executed on the domain after the dimension is patched.

- **Enable DPM** – Dynamic Position Maintenance (DPM) allows informal positions to be added to a dimension on-the-fly from the RPAS Client. Select the **Enable DPM** option for the dimensions that will be enabled to support DPM. After Enable DPM is defined for the dimension, you must also specify workbooks and the dimensions in each workbook that will use DPM (see the Workbook Designer window for more details).

Note: DPM can be enabled for all hierarchy dimensions except for CLND, ADMU, and LNGS.

When **Enable DPM** is selected for a specified dimension, it is also selected for all dimensions that roll up to it.

For more information on DPM, see the *Oracle Retail Predictive Application Server Administration Guides* and *Oracle Retail Predictive Application Server User Guides*.

Enabling DPM is patchable and will not adversely affect the expected results if altered. However, disabling DPM is not patchable. DPM may be disabled for the templates, but not for the dimension.

- **Enable Images** – Select this option to enable the association of images (image paths) to positions along the specified dimension. To disable this feature, deselect the option for the appropriate dimensions. This option is available for all hierarchy dimensions, except the calendar hierarchy. For the calendar hierarchy, the Enable Images column is disabled or grayed. RPAS supports GIF, BMP, and JPEG image formats. Once Enable Images is defined for a dimension, you must also specify the workbook that will use this feature (see the Workbook Designer window for more details). See the Oracle Retail Predictive Application Server Administration Guide for more information on loading image paths.

Note: Enabling and disabling images is patchable and will not adversely affect the expected results if altered.

Delete a Dimension

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

1. From the Dimensions region or from the Hierarchy navigation tree, select the dimension to be deleted.
2. Perform one of the following options:
   - Click the **Delete** icon.
   - Press the Delete key.
   - Select **Delete** from the right-click menu in navigating in the Hierarchy navigation tree.

Note: Deleting a dimension causes all of the dimensions that are structurally dependant on it (its parents, grandparents, and so on) to also be deleted.

Note: The user dimension contained in the ADMU hierarchy cannot be deleted.

Edit a Dimension

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

1. Select the Hierarchy in which a dimension will be edited.

2. From the Dimensions region, select the dimension to edit.
3. Update the dimension property as necessary. Refer to "Defining Dimension Properties" for more information. property to update.
4. To change the order of the dimension click the left button or right button to change the order of the dimensions. Re-ordering dimensions only affects the file, not the parent/child relationship of data.

> **Note:** The up button and the down button are used to change the order of hierarchies only.

Create a Branch in a Hierarchy

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

1. From the Dimensions region or from the hierarchy navigation tree, select the dimension that will be the base of the branched hierarchy. The base of the branched hierarchy is the root dimension.

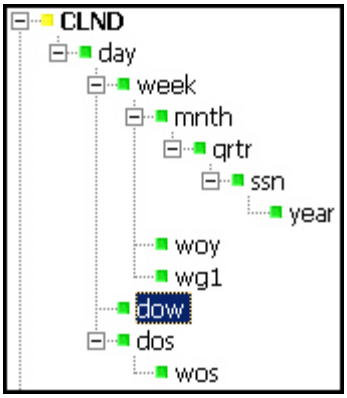
Example Create a Branch in a Hierarchy

> **Note:** Ensure that the root dimension will have more than one parent (that is, where the branch starts), and create another parent dimension. Branches can never join together (for example, both style-subclass-class and style-supplier-class roll-ups in the same hierarchy are invalid).

Choose one of the following methods:

- From the Hierarchy Definition right-click menu, select **New Dimension**.
- Click the **New Dimension** button on the toolbar.
- Press the **Insert** key on the keyboard.

  The new dimension will be created below the selected dimension.



Example of Branch Hierarchy

Labeled Intersections

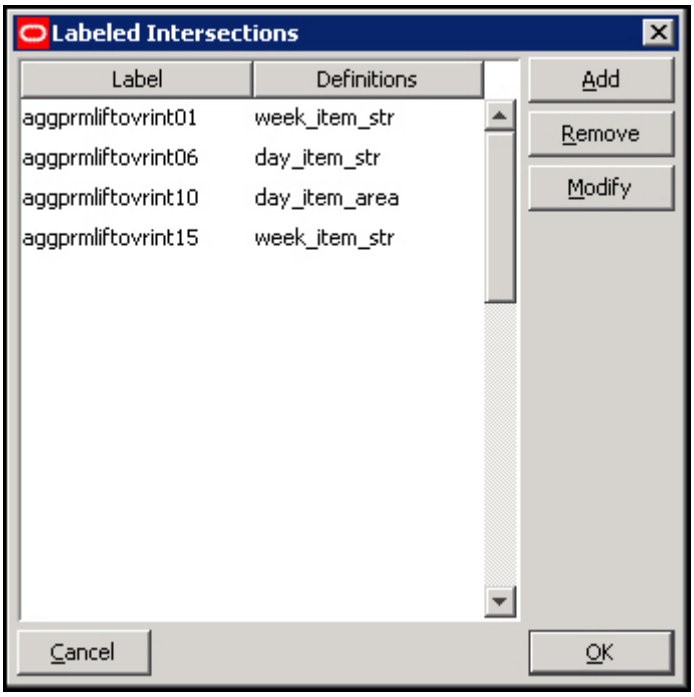
The Labeled Intersections window supports the addition, removal and modification of hierarchy intersections. A hierarchy intersection defines the dimensionality at which data

is defined. An intersection may be defined as using no dimension (scalar), using a single dimension from a hierarchy, or multiple dimensions from different hierarchies.

> **Note:** See the section on "Measures and Base Intersections" for more information on defining intersections for data.

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

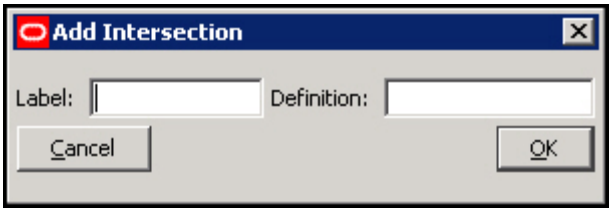
1. From the Hierarchy Definition toolbar, select the **Labeled Intersection** icon . The Labeled Intersections dialog box appears. This dialog box allows you to add new intersections or remove or modify existing intersections.



Labeled Intersections Dialog Box

Adding a Labeled Intersection

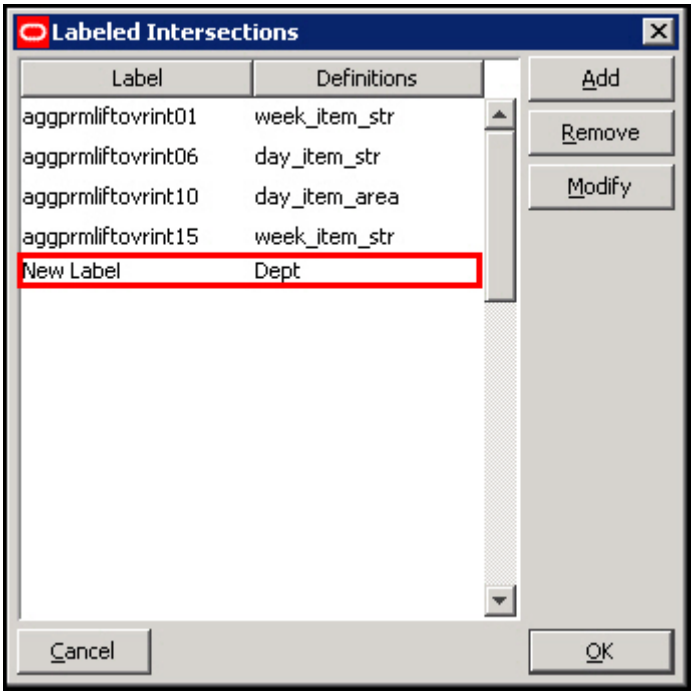
1. Click Add from the Labeled Intersection dialog box. The Add Intersection dialog box appears.



Add Intersections Dialog Box

2. Perform the following:
   a. In the **Label** field, enter a label for the Labeled Intersection.
   b. In the **Definition** field, enter the dimension name(s) for the intersection.
   c. Click **OK**. The dialog box closes and the new entry appears in the Labeled Intersection dialog box.

Note: If the **Definition** field is left empty, the measure is assumed to be Scalar. If the level is non-scalar, dimension names are used to define the intersection. If multiple dimension names are to be specified, each dimension name must be separated by an underscore (_). The last dimension specified SHOULD NOT have an underscore following the dimension name. As well, there is no required order of dimensions.



Example of New Labeled Intersections

3. Click **OK**.
4. Once the labeled intersection is added, you can perform the following:
   - Define or update the Base Intersection of major or minor measure component using the labeled intersection.
   - Define or update the Load Intersection of a measure using the labeled intersection.
   - Define or update the Base Intersection of a worksheet using the labeled intersection.

Note: RPAS imposes a limit of 5 dimensions that can be defined in a measure or worksheet's base intersection.

Modifying a Labeled Intersection

1. Select a labeled intersection.
2. Click **Modify** from the Labeled Intersection menu.

   Only the **Definition** field can be modified.
3. Click **OK**. If the change is undesired, select **Cancel**.

   When the **Definition** of an existing labeled intersection is modified, the base intersections of measures and worksheets, and load intersections of measures that are

currently assigned the labeled intersection are automatically updated. No action is required.

Removing a Labeled Intersection

1. Highlight a labeled intersection.
2. Click **Remove** from the Labeled Intersection dialog box.
3. Click OK.

When an existing labeled intersection is removed, the base intersections of measures and worksheets, and load intersections of measures that are currently assigned the labeled intersection will be displayed as invalid (red). Warning messages will also appear in the Task List, indicating the intersections that must be updated. These intersections must be corrected prior to installing or patching a domain.

Data Interface Manager

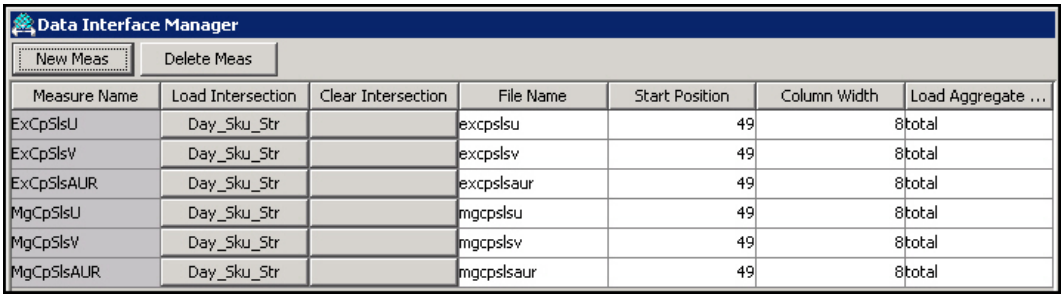
Overview

The Data Interface Manager tool is used to specify information about how data will be loaded into the domain. This includes properties of the file to be loaded and the intersection at which data will be loaded into the domain.

Data can only be loaded into stored, realized measures in the domain. Therefore, only such measures can be used in the Data Interface Manager. Refer to the "Working with Measures" section of this document.

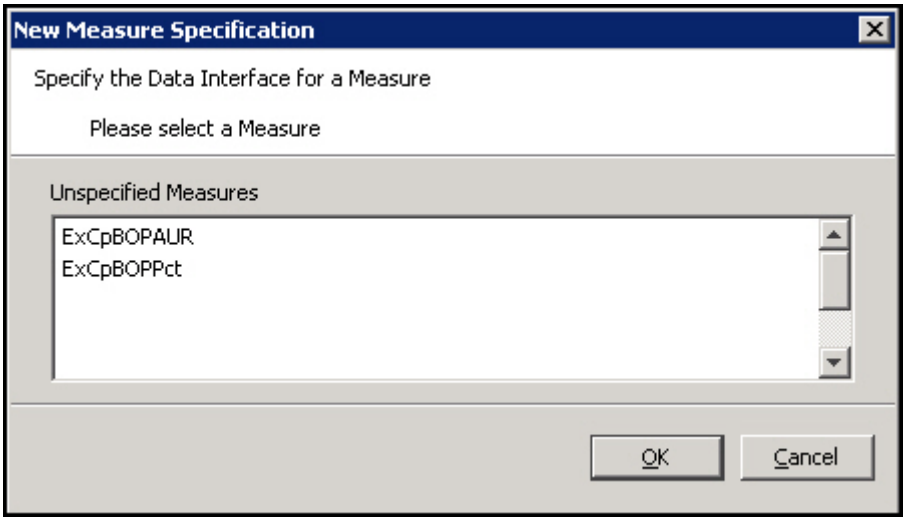
Specify the Data Interface for a Measure

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



Example of Data Interface Manager Window

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



Example of New Measure Specification Window

> **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.

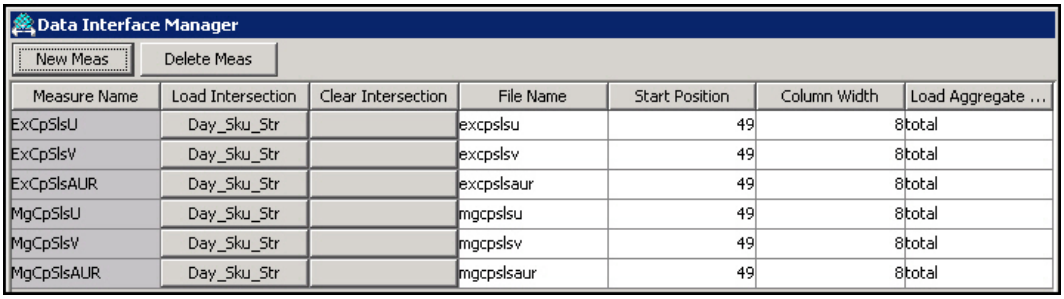
2. Select the measure requiring a data interface definition.
3. Click **OK**. The measure appears in the Data Interface Manager window.

Add/Edit Data Interface Properties for a Measure

Navigate: In the Configuration Components pane, 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.

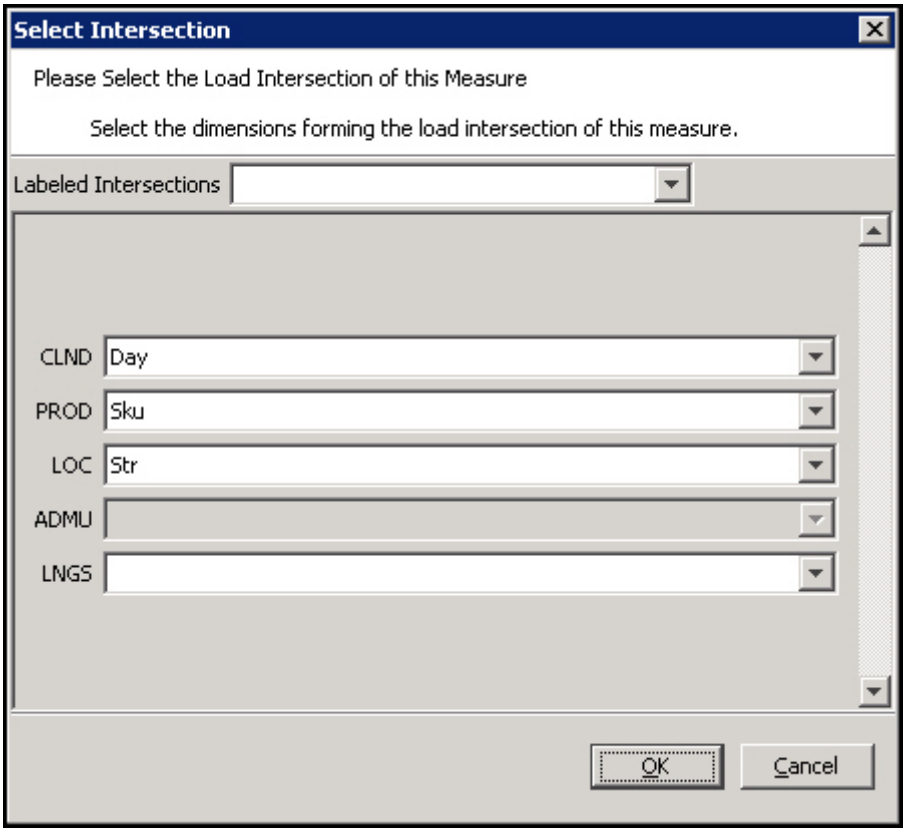
> **Note:** Data can be loaded at the same intersection or lower than the base intersection of a measure. Data cannot be loaded at a higher intersection than the base intersection.



Measure Name	Load Intersection	Clear Intersection	File Name	Start Position	Column Width	Load Aggregate ...
ExCpSlsU	Day_Sku_Str		excpslsu	49	8	total
ExCpSlsV	Day_Sku_Str		excpslsv	49	8	total
ExCpSlsAUR	Day_Sku_Str		excpslsaur	49	8	total
MgCpSlsU	Day_Sku_Str		mgcpslsu	49	8	total
MgCpSlsV	Day_Sku_Str		mgcpslsv	49	8	total
MgCpSlsAUR	Day_Sku_Str		mgcpslsaur	49	8	total

Data Interface Manager Window – Add/Edit

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



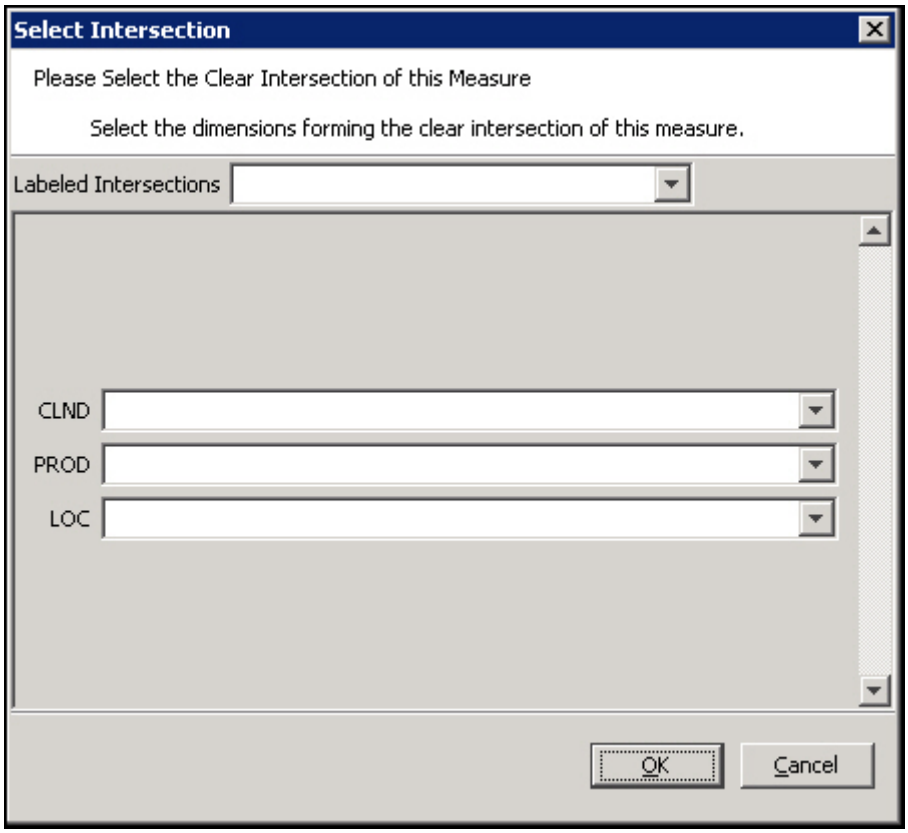
Select Intersection Window

2. To specify the load intersection:
   a. Using the list options, select the appropriate dimensions or Labeled Intersection.

      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. Click the **Clear Intersection** field to change its value. The **Select Intersection** window opens.

   Note: The clear intersection allows support of .clr measure data files. Using .clr files with loadMeasure allows the clearing of selected portions of a measures data arrays.



Select Intersection – Clearing the Intersection

4. To specify the clear intersection:
   a. Using the list options, select the appropriate dimensions or Labeled Intersection.

      Note: Only those dimensions that are at the same level as the base intersection or above are displayed for the clear intersection.

   b. Click **OK** to save any changes. Close the window.
5. In the **File Name** field, enter the file name from which data for the measure will be loaded.
6. 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.
7. 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 it can be changed.

8. 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. Click the **Load Aggregation Method** field and select the appropriate aggregation method from the option list.

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

Delete Data Interface Information for a Measure

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

1. Select the measure you want to remove from the Data Interface Manager.
2. Click Delete Meas.
3. Click **Yes**. The measure is removed from the table.

Working with Styles

Overview

It is possible for the RPAS Client user 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 Definition Tool

The Style Definition Tool provides the following functionality:

- 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 is also defined for all of the children of that style unless a style is defined for a child. The attributes of a style that may be specified are as follows:

- **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 U.S. currency values. The prefix can be any character sequence, but cannot exceed 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 suffix can be any character sequence, but cannot exceed 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. If this number is set to 3, the client must always display 3 positions after the decimal. For example, the measure value is 1, with a Precision setting of 3, it will be displayed as 1.000. 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 come from 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. It can also be displayed in other formats depending on the regional settings.
- **Text Font** – Sets the font of cell value in the RPAS Classic Client (Times New Roman, Arial, and so on).
- **Text Style** – Sets the display style of the text value in the RPAS client (Bold, Italic, and so on).

  Note: Text font is not used by the Fusion Client.
- **Text Size** – Sets the font size in which the cell value in the RPAS Classic Client is to be displayed.

  Note: Text size is not used by the Fusion Client.
- **Text Color** – Sets the color of the cell values in the RPAS client.
- **Background** – Sets 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 Configuration 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** – Sets the alignment of values within the cells when viewed in the RPAS client (Left, Center, and Right

- **Border Style** – Sets they style of border of cells. Border style determines the kind of borders (for example, 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).

  Note: Border style is not used by the Fusion Client.

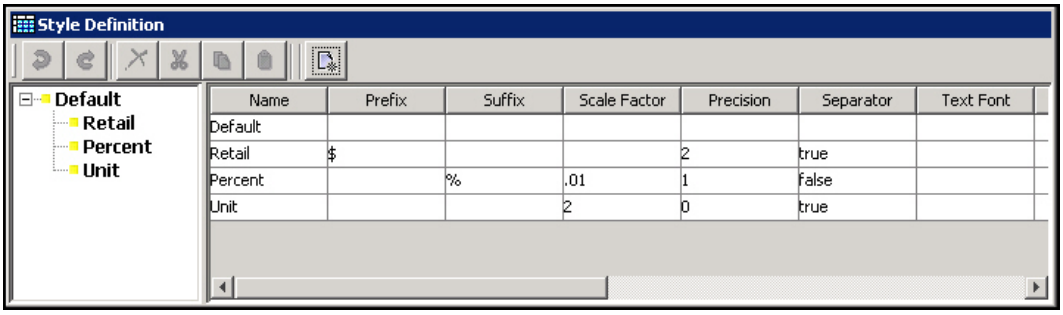
- **Border Color** – Sets the color of the border lines for cell values.

  Note: Border color is not used by the Fusion Client.

- **Time Format** – Sets the time format for styles. Options are No Time, Twenty-Four Hour, and Twelve Hour.

Create a Style

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



Style Definition Window

1. Select or create the Style that will be the parent of the new style. All styles must ultimately be descendents of the Default Style.
2. Choose one of the following methods:

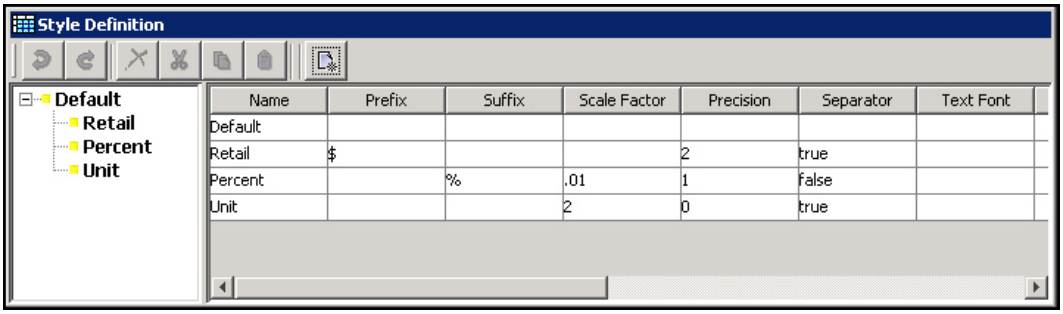
   a. Click the **Create a new style** button in the toolbar.

   b. Select **New Style** from the right-click menu.

   c. Press the **Insert** key.

   If the Style Attributes for Default are populated, all of its descendants will inherit the same attributes unless you specify 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.

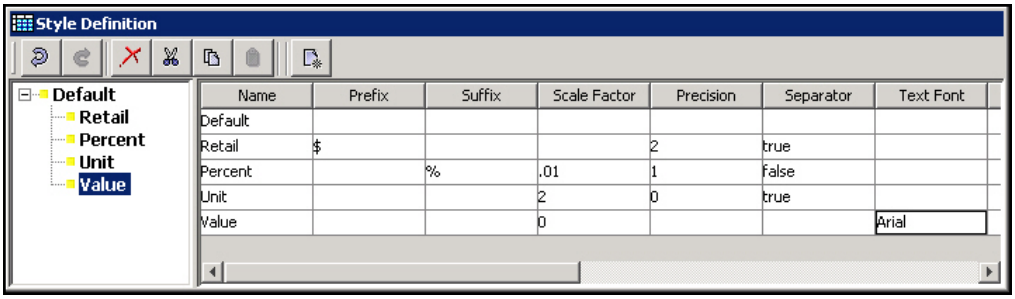


Style Definition Window

Remove a Style

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

1. Select the style to be removed.

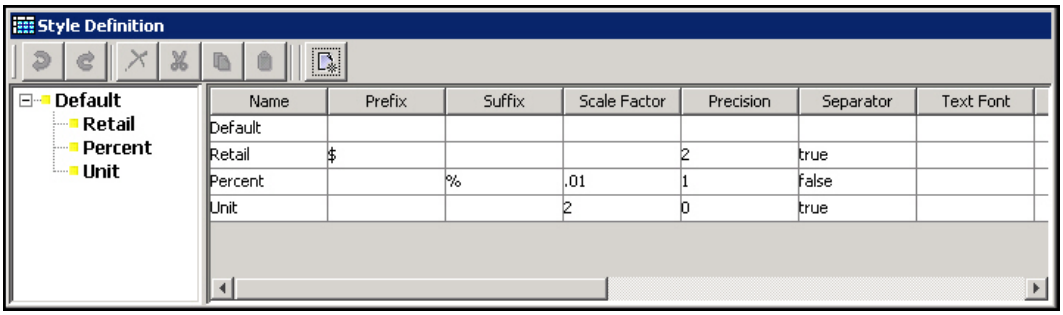


Style Definition Dialog Box

Choose one of the following methods:

- Click the **Delete Style** button in the toolbar
- Select **Remove** from the right-click menu.
- Press the **Delete** key.

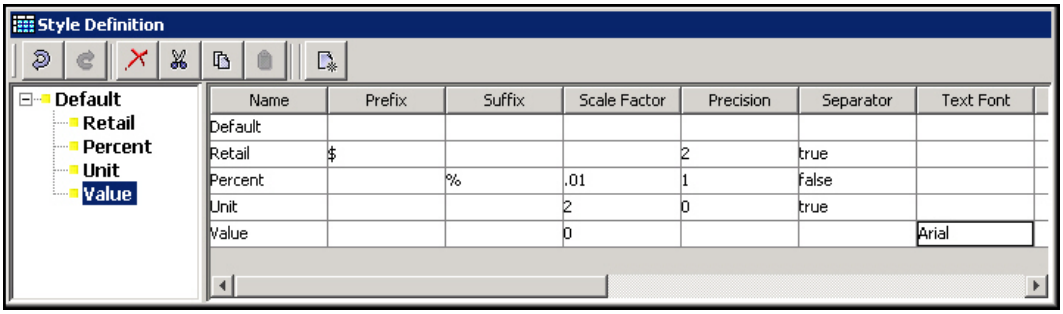
The selected style and all of its child styles will be removed from the style description tool. Any measures using a deleted style will be displayed as invalid.



Style Definition Dialog Box

Edit a Style

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



Style Definition Dialog Box

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 (does not contain any value), select the property, and press the **Delete** key.

Working with Taskflows

Overview

The RPAS Fusion Client provides a more flexible approach than the RPAS Classic Client to user interaction with the workbooks configured for an RPAS domain. This flexibility allows users to focus less on the structural elements that make up the workbook configuration and more on the tasks that they use those structural elements to perform.

> **Note:** Taskflows can be created only for the RPAS Fusion Client, not the RPAS Classic Client. This entire taskflow section only applies to the Fusion Client.

This flexibility is found in the Fusion Client taskflow. The taskflow allows configurators to more closely describe and model their business practices within the client. Configurators can create taskflow elements (activity groups, activities, tasks, and steps) that can be associated with structural elements of the workbook configuration. These taskflow elements then provide a more intuitive and business practice-oriented view of the structural elements of the RPAS domain.

The activity group / activity / task / step can be organized however the user wants. The task refers to a template and specific solution. Inside a Configuration Tools configuration, they all refer to the same solution. Names are all qualified by the Solution ID, so that they do not conflict later when they are manually combined. A single solution taskflow can have multiple activity groups, and a combined taskflow could conceivably have just one

(for example, put all the activities from the various solutions under a single activity group).

For multiple solutions, activity groups can be used to integrate activities from the solutions into a unified taskflow configuration that spans those solutions. The activity group provides an integrated workflow that represents your business process across multiple solutions, that is, it organizes activities from multiple solutions so the activities can be presented together under a single organizing entity.

When creating an activity, the activity group name, label, and description can be specified. When the domain is built, the activities are combined into an activity group based on the specified activity group properties. For users upgrading to 14.0, the defaults are used to specify activity group information so that no additional configuration work is required if you are not implementing a taskflow across multiple solutions.

Each task is specific to a workbook template and therefore a specific solution. If there is a restriction needed for the domain type, it can be specified for each task by selecting the task type when specifying the task properties:

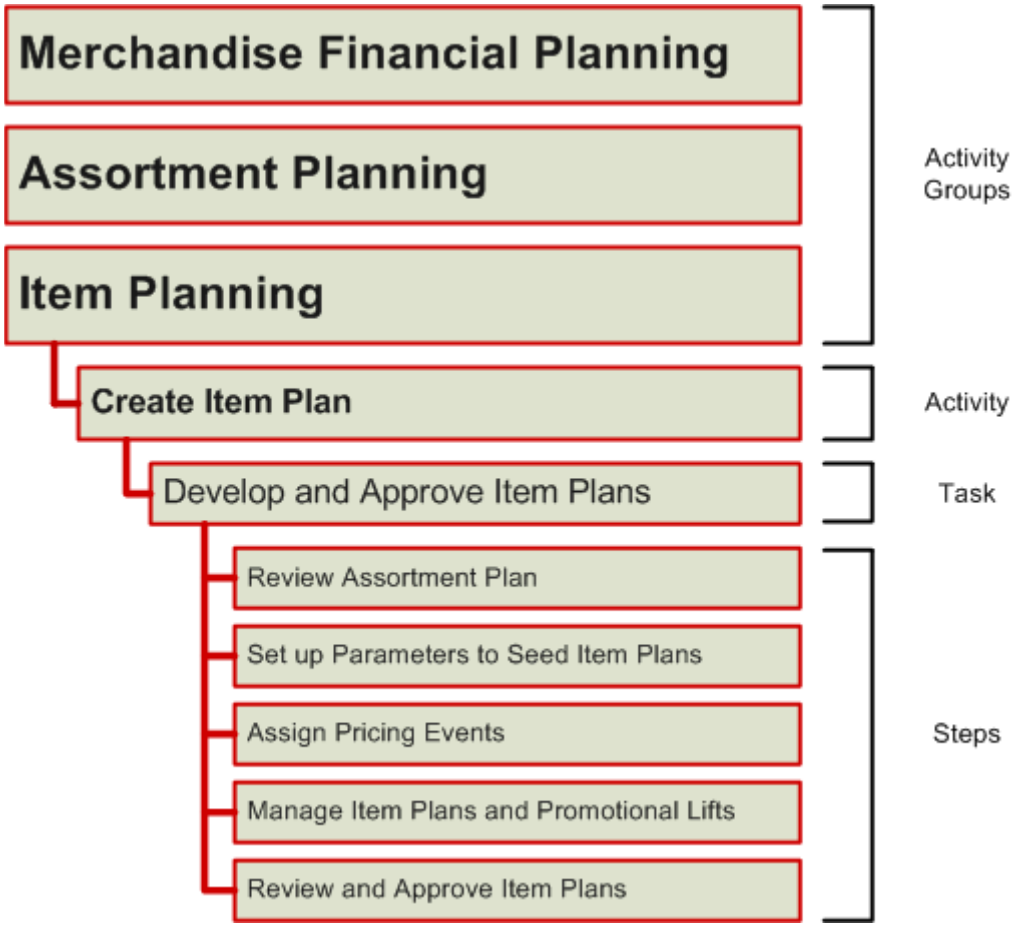
- Master – create the workbook in the master domain only.
- Locals – create the workbook in the local domains only.
- Both – create the workbook in either the master or local domains. This is the default.

> **Note**: For clients upgrading to 14.0, the defaults are used to specify activity group information so that no additional configuration work is required if you are not implementing a taskflow across multiple solutions. However, new taskflow files need to be generated to conform to the new structure.

For users upgrading to 14.0 or later, the default task type of Both is used so that no additional configuration work is required if you are not implementing a taskflow across multiple solutions.

The following examples illustrate single and combined activity groups. The first example shows an activity group that includes multiple solutions. Fashion Planning: Pre-Season Planning is the activity group. Within this activity group, the Develop and Approve Item Plans task is from Item Planning and the Financial Review of Item Plans activity is from Merchandise Financial Planning.

The following shows the activity group at the solution level. Merchandise Financial Planning, Assortment Planning, and Item Planning each contain the taskflow for the individual solution.

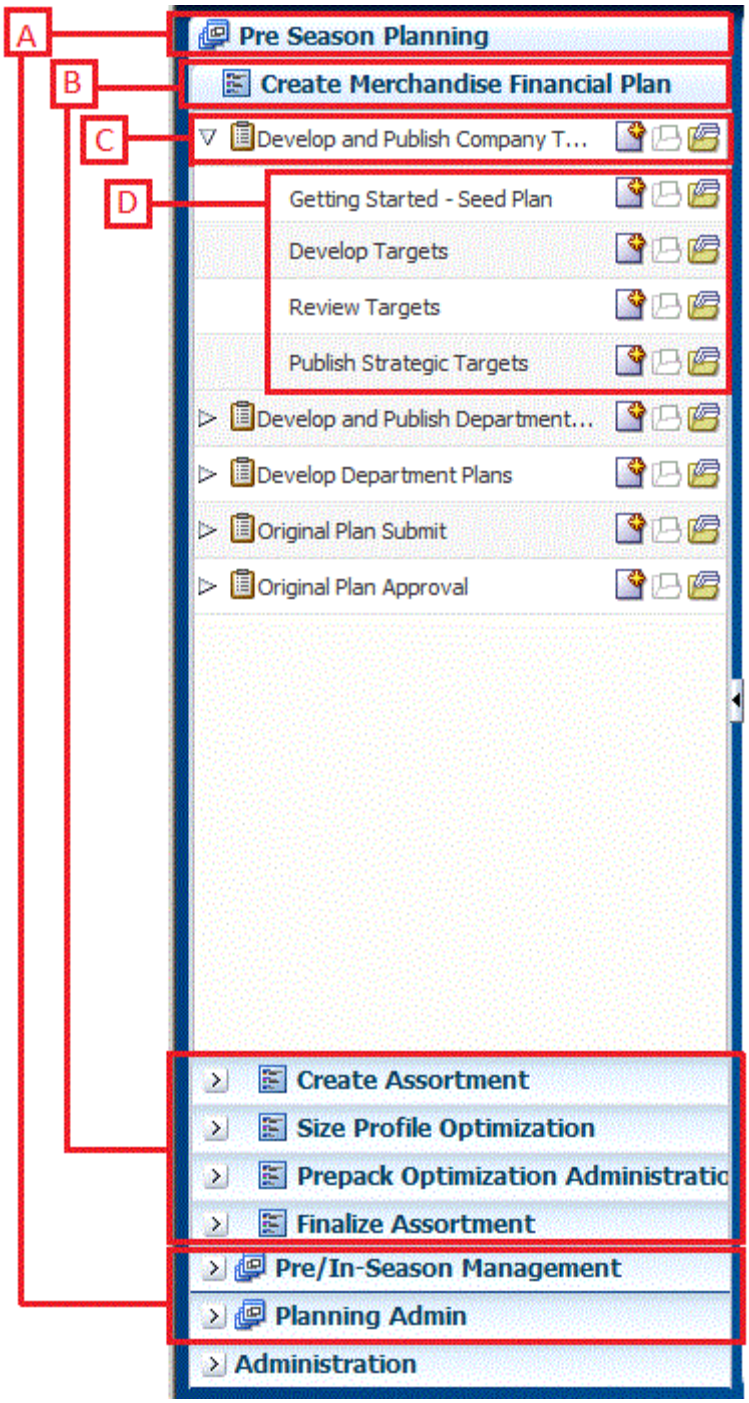


Using the RPAS Configuration Tool, you have the ability to create a customized activity taskflow for the RPAS Fusion Client. This activity taskflow helps users of the RPAS Fusion Client understand the tasks they must complete in order to meet their planning goals.

The RPAS Configuration Tool works on a taskflow for a single solution (configuration). Everything within it will be qualified by Solution ID so as not to create conflicts with other solution's taskflows. For a multi-solution configuration, these need to be combined into a single multi-solution taskflow. This is a manual process of combining the taskflow XML and resource properties files. For information on moving the taskflow and resources into place, see Creating a Multi-solution Taskflow.

Some RPAS configured solutions are delivered with preconfigured taskflows. These preconfigured taskflows can be customized using the steps below. Or, you can create new taskflow.

Below is an example of a taskflow in the RPAS Fusion Client.



Taskflow Within the RPAS Fusion Client

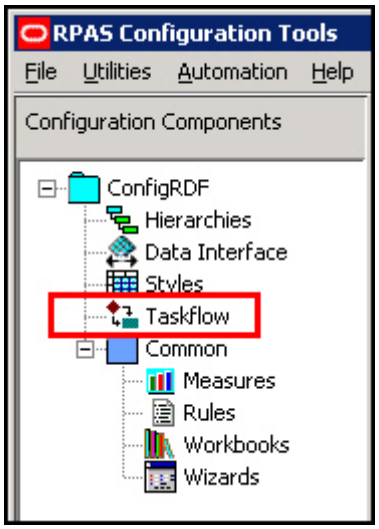
The following table describes the icons that appear with all the entries in the activity taskflow.

Legend	Icon Name	Description
A	Activity Groups	These tabs represent the grouping of activities

Legend	Icon Name	Description
B	Activities	These tabs represent the predefined activities of the application
C	Tasks	These are individual tasks within an activity
D	Steps	One or more steps make up each task

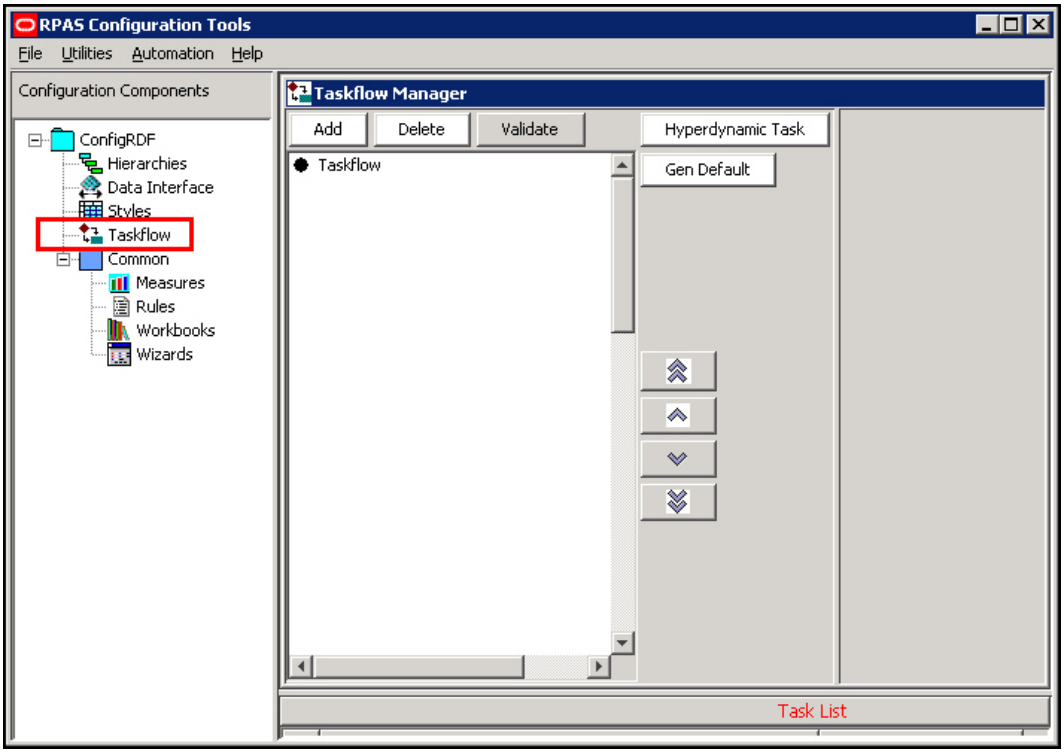
Create a Taskflow

Navigate: In the Configuration Components pane, select Taskflow within the project for which you want to create a taskflow.



Taskflow Icon in the Configuration Components Pane

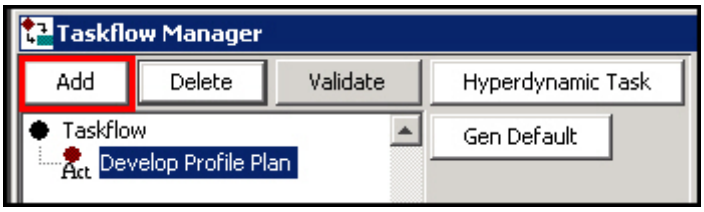
The Taskflow Manager window appears. The first time that you use the Taskflow Manager, it contains one bullet called Taskflow.



Taskflow Manager

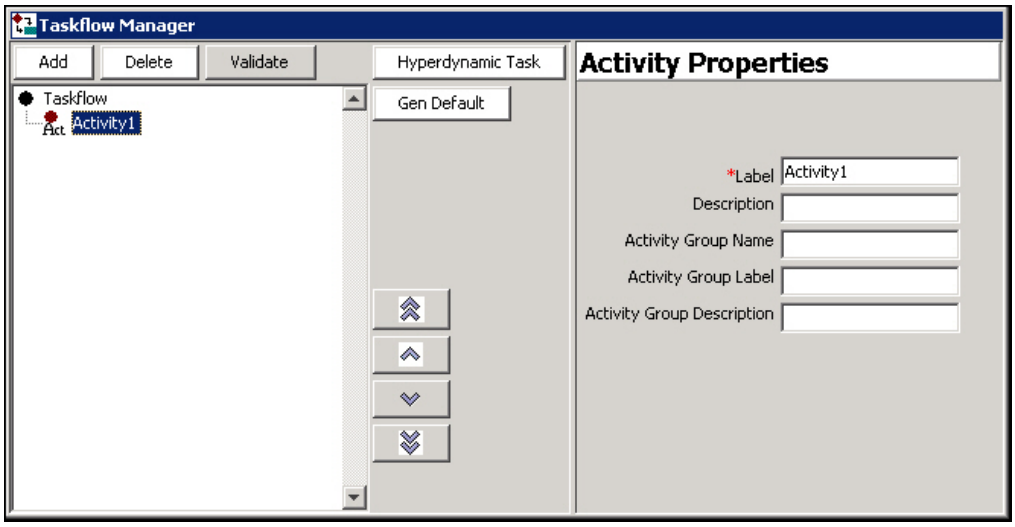
Adding an Activity to the Taskflow

1. Select the **Taskflow** bullet inside the navigation pane of the Taskflow Manager. Click **Add**.



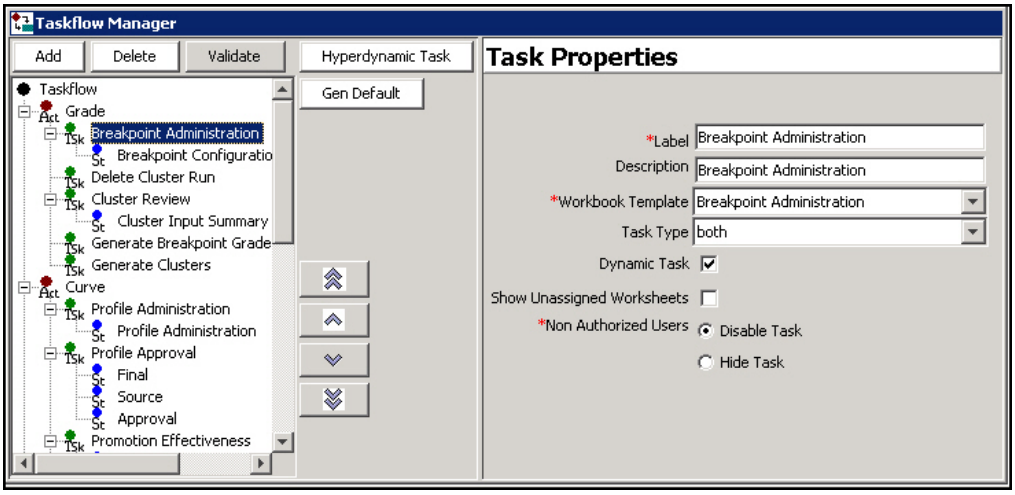
Adding an Activity to the Taskflow

2. An activity called Activity1 appears in the navigation pane. Select **Activity1**. The Activity Properties appear in the detail pane.



Activity1 in the Taskflow

3. Enter the name of the activity in the **Label** field. This is the name of the task as it appears in the taskflow of the RPAS Fusion Client. The red asterisk denotes that this step is required. Note that as you type the name in the Label field, the name is updated in the navigation pane.



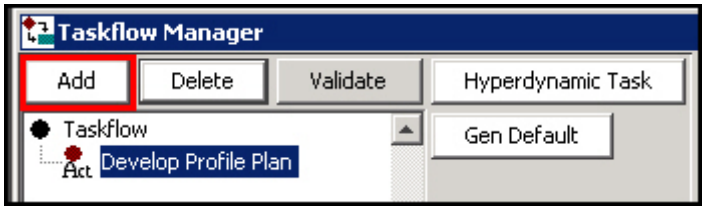
Activity Properties

4. Enter a description for the activity in the **Description** field. The description appears when the user rolls the cursor over the activity name in the RPAS Fusion Client. This step is optional.
5. Enter the activity group name in the **Activity Group Name** field, group label in the **Activity Group Label** field, and group description in the **Activity Group Description** field. This step is optional.

After you've entered the activity properties, the new activity is created.

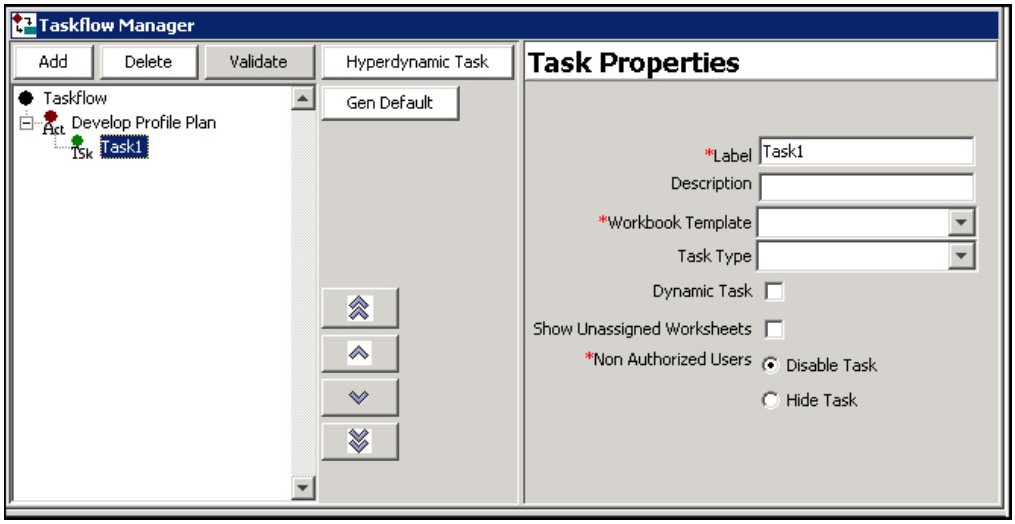
Adding a Task to the Taskflow

1. To create a task within an activity you created, select the activity in the navigation pane and click **Add**.



Adding a Task to an Activity

1. A task called Task1 appears in the navigation pane. Select **Task1**. The Task Properties appear in the details pane.



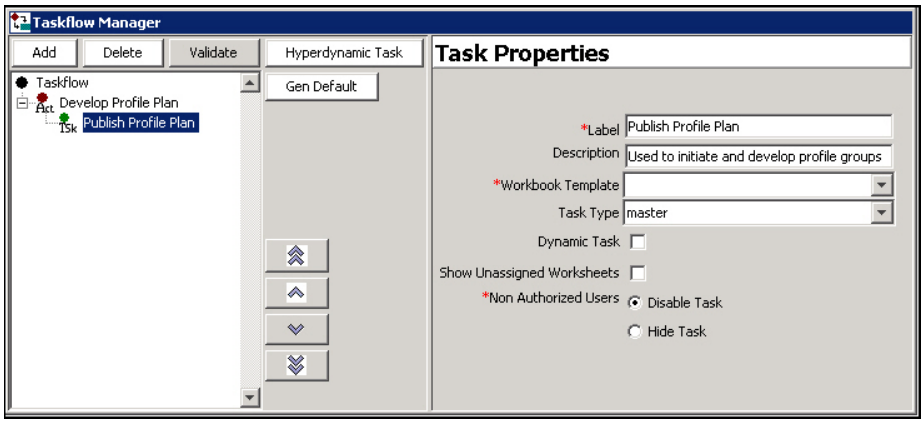
Task Properties

2. Enter the following information in the Task Properties.
   - **Label:** enter the name of the task as you want it to appear in the Oracle Retail Predictive Application Server Fusion Client. This step is required.
   - **Description:** enter the description of the task. This description appears when the user rolls the cursor over the task name in the Oracle Retail Predictive Application Server Fusion Client.
   - **Workbook Template:** choose the workbook template that will be utilized in this task. This step is required.
   - **Task Type:** Choose whether the task can be performed over the master domain, local domains, or both. The default is both.
   - **Dynamic Task:** Select this option if the steps for the task are dynamic based on the user selections made during the workbook wizard. When this option is selected, the user will not see any steps under a task in the Oracle Retail Predictive Application Server Fusion Client until a workbook is open.

     Note: Custom workbook code is still required to provide the ability to filter steps based on the wizard selections

   - **Show Unassigned Worksheets** – By checking the unassigned flag, applications that include the potential for non-configured worksheets can allow those worksheets to be displayed within the Fusion Client while continuing to maintain the ability to hide worksheets irrelevant to the task at hand for workbooks that do not contain non-configured worksheets.

- **Non Authorized Users:** Choose whether unauthorized users are able to see the task but are unable to edit it (Disable Task) or are not able to see it at all (Hide Task).

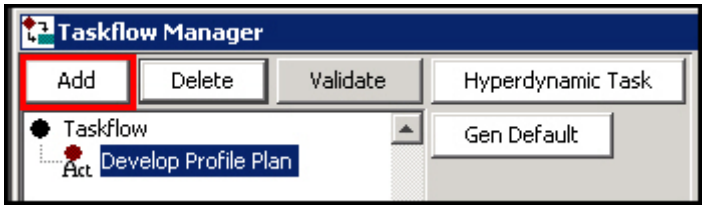


Task Properties

Once you've entered the task properties, the new task is created.

Add a Step to the Taskflow

1. To add a step to a task you created, select the task and click **Add**.

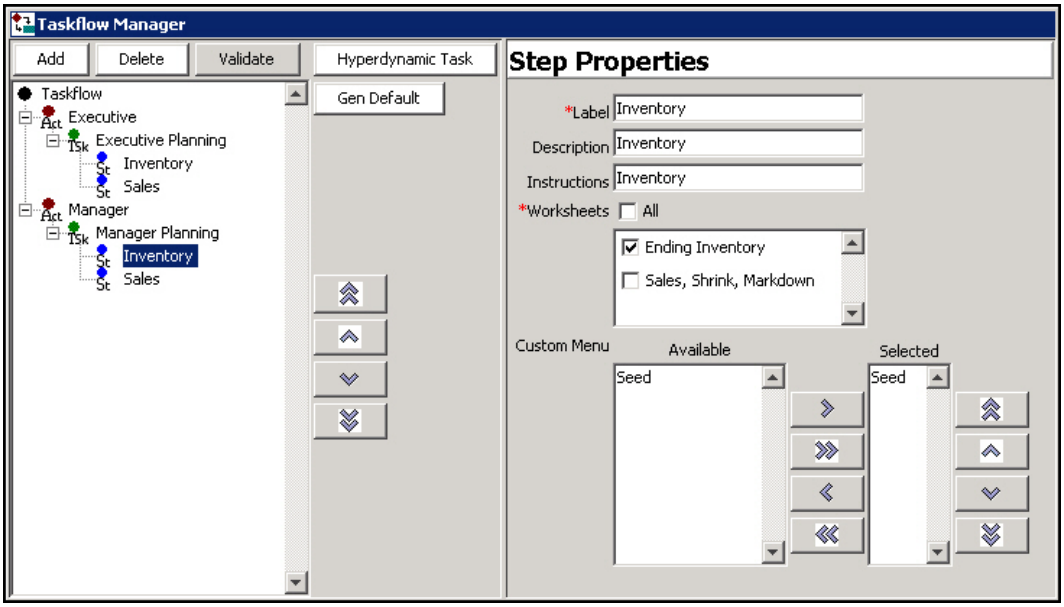


Adding a Step to a Task

2. A step called Step1 appears in the navigation pane. Select **Step1**. The Step Properties appear in the details pane.
3. Enter the following information in the Step Properties:
   - **Label:** enter the name of the step as you want it to appear in the Oracle Retail Predictive Application Server Fusion Client. This is required.
   - **Description:** enter the description of the step. This description appears when the user rolls the cursor over the step name in the Oracle Retail Predictive Application Server Fusion Client.
   - **Instructions:** enter instructions that explain what to do in the step. These instructions appear below the task pane.

Instructions in the RPAS Fusion Client

   - **Worksheets:** select the worksheets that will be utilized in this step. You can select all worksheets to be included by checking the **All** checkbox. Or, you can select a subset of worksheets from that workbook template. This is required.
   - **Custom Menu:** if you want to include a custom menu in this step, select it in the **Available** area and then use the arrow to move it to the **Selected** area. You can change the order of the custom steps by using the up and down arrows.

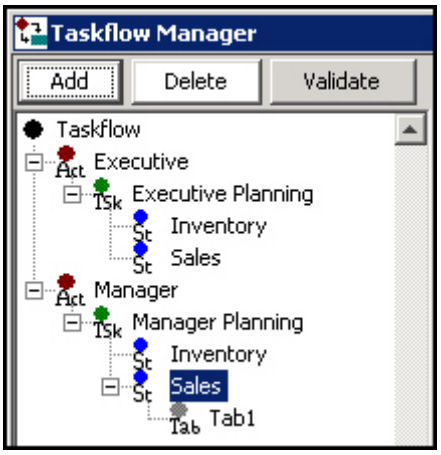


Step Properties

Once you've entered the step properties, the new step is created.

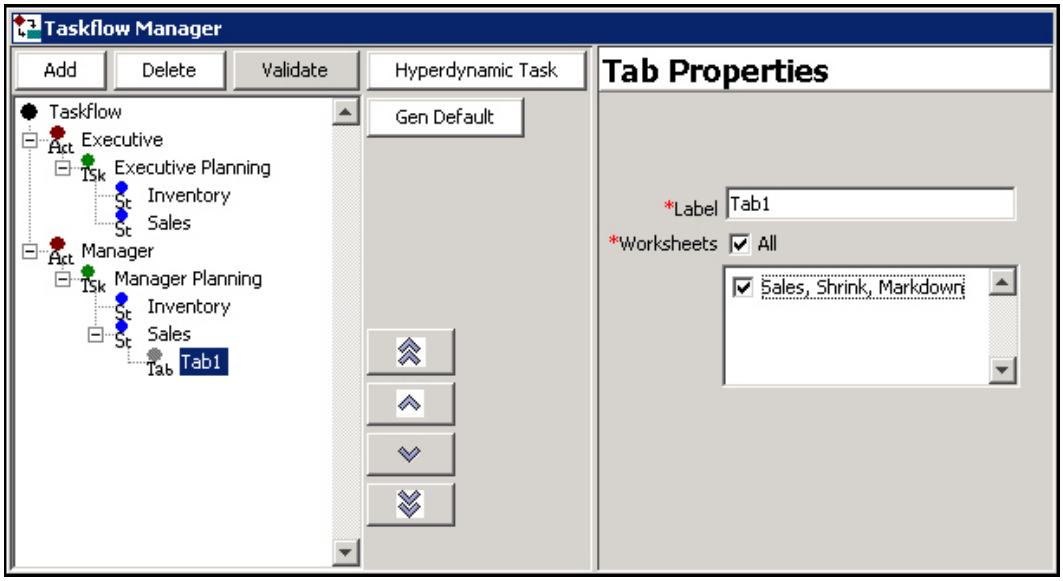
Add a Tab to the Taskflow

1. To add a tab to a step you created, select the step and click **Add**.



Adding a Tab to a Step

2. A tab called Tab1 appears in the navigation pane. Select **Tab1**. The Tab Properties appear in the details pane.
3. Enter the following information for the tab:
   - **Label:** enter the name of the tab as you want it to appear in the Oracle Retail Predictive Application Server Fusion Client. This is required.
   - **Worksheets:** select the worksheets that will be utilized in this tab. You can select all worksheets to be included by checking the All checkbox. Or, you can select a subset of worksheets. This is required.

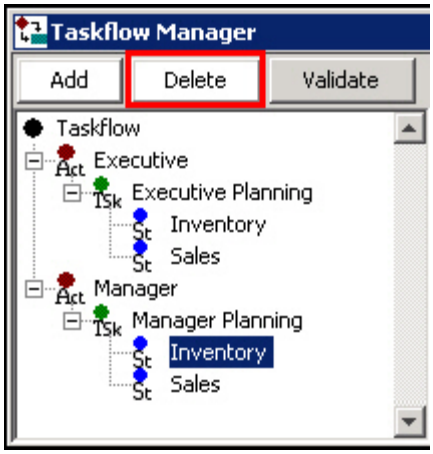


Tab Properties

Once you've entered the tab properties, the new tab is created.

Delete Items from the Taskflow

To delete any activity, task, step, or tab in the taskflow, select it and click Delete.



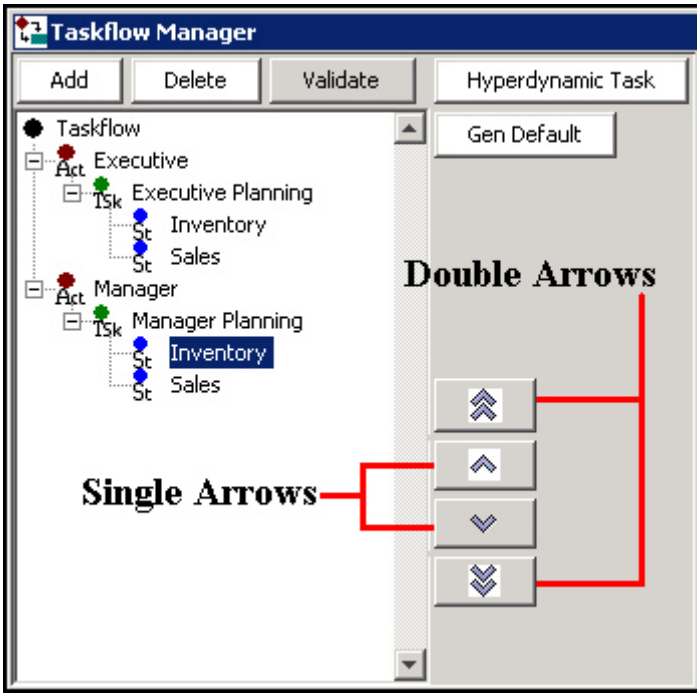
Deleting Items in the Taskflow

Edit Items from the Taskflow

To edit an activity, task, step, or tab that you have already created, select it in the navigation pane. It properties appear in the details pane. Then edit the properties you want.

Order Items in the Taskflow

To change the order of activities, tasks, steps, or tabs, select the item that you want to move and use the arrow buttons to the right of the navigation pane. Click a single arrow to move it one level in the list. Click the double arrow to move it to the top or bottom of the list.



Ordering Items in the Taskflow

Validate the Taskflow

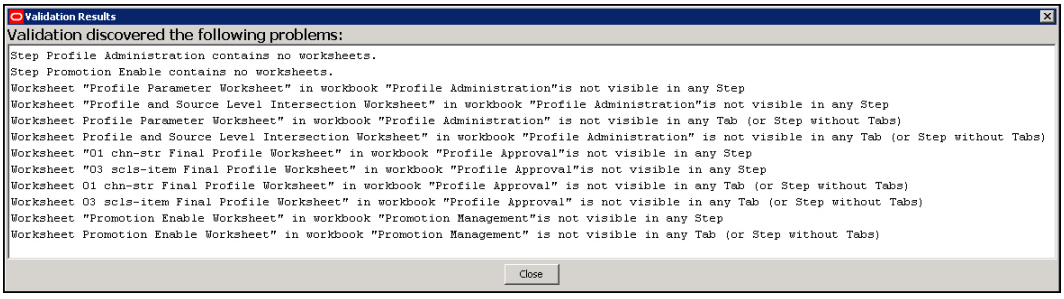
As you create the taskflow, the Task List below the Taskflow Manager displays configuration elements that may cause validation errors. In addition, the Validation Results tool shows you if you have configured elements that are unavailable to users. For instance, if a configured taskflow does not have a task associated with a given workbook template, then the RPAS Fusion Client user is not able to build or open any workbooks built from that template.

> **Note:** The presence of validation problems does not prevent you from building a domain.

To validate your taskflow, click **Validate** at any time. The Validation Results window appears. The Validation Results tool checks for the following:

- Workbook templates that are not associated with a task
- Worksheets that are not a part of any step
- Custom menu items that are not available in any step

When conditions such as these exist, they appear in the Validation Results window. Once you have reviewed the results, you can choose to resolve any issues if necessary.

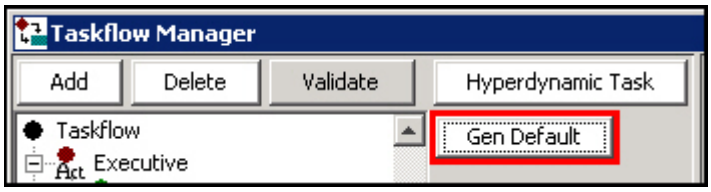


Validation Results

The RPAS Configuration Tools also performs validation within the Task List below the Taskflow Manager. As you edit within the Taskflow Manager, the Task List reports potential issues and errors within the configuration. For more information, see the Task List section.

Generate Default Mapping

Click **Gen Default** at any time to revert the taskflow to the default mapping.



Generate Default Mapping

The default mapping provides a basic structure to taskflow elements and can serve as either a simple configuration of workbook elements or as a starting point for a customized configuration of the taskflow elements.

The default mapping is setup as follows:

- There is one activity generated for every workbook group in the domain. The label of the workbook group is used as the activity's label and description.
- Each activity contains a single task for each workbook present in the workbook group associated with that activity. The label of that workbook is used as the task's label and description. By default, workbooks are disabled for unauthorized users in all tasks.
- Each task contains a single step for each workbook tab present in the workbook associated with that task. The label of the worksheet is used as the step's label, description, and instructions. Each step includes the worksheets contained within the workbook tab associated with that step and does not include worksheets contained within other workbook tabs present in the workbook. Each step contains all Custom Menus present in the workbook.
- The default mapping contains no taskflow tab elements.

Hyperdynamic Tasks, Steps, and Tabs

Some RPAS solutions use custom workbook template libraries to extend the workbook creation functionality of RPAS. For some of these solutions, some or all of the content of a workbook is determined, not by the configuration, but by the custom template library at the time the workbook is built.

Because the worksheet and tab content of these workbooks is dynamic, it is not possible to configure the taskflow in the same manner that a taskflow is configured for standard workbooks. Instead, these custom workbooks use special taskflow objects named hyperdynamic tasks, step, and tabs. Hyperdynamic tasks, steps, and tabs differ from their standard counterparts in the following ways:

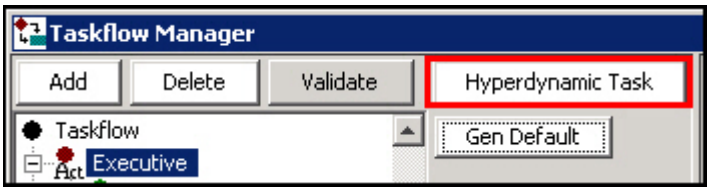
- A hyperdynamic task can contain only hyperdynamic steps.
- A hyperdynamic step can contain only hyperdynamic tabs.
- Hyperdynamic steps and tabs do not select visible worksheets based on the worksheets of the workbook assigned to the task. Instead, they are populated by providing the names of worksheets that may not exist within the configured workbook at the time the taskflow is configured.

- At the time of workbook creation, the Fusion Client uses the configured list of worksheet names in coordination with the set of worksheets in the built workbook to determine which worksheets will be visible in the Fusion Client and how they will be assigned to tabs.
- Because the worksheet membership of hyperdynamic tasks, steps, and tabs is not determined until workbook build time, they cannot be validated using the taskflow validation functionality.

Creating a Hyperdynamic Task

To create a hyperdynamic task to an activity, perform the following steps:

1. Select the activity and click **Hyperdynamic Task**.



Hyperdynamic Task Button

2. The new hyperdynamic task appears in the navigation pane. Select the new task. The new task properties appear in the detail panel.
3. Enter the following information in the Task Properties.
   - **Label**: Enter the name of the task as you want it to appear in the RPAS Fusion Client. This step is required.
   - **Description**: Enter the description of the task. This description appears when the user rolls the cursor over the task name in the RPAS Fusion Client.
   - **Workbook Template**: Choose the workbook template to be utilized in this task. This step is required.
   - **Dynamic Task**: Select this option if the steps for the task are dynamic based on the user selections made during the workbook wizard. When this option is selected, the user does not see any steps under a task in the RPAS Fusion Client until a workbook is open.

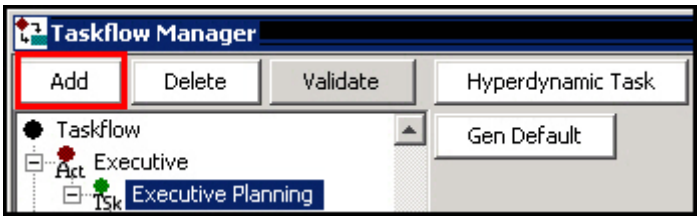
     Note: Custom workbook code is still required to provide the ability to filter steps based on the wizard selections.

   - **Show Unassigned Worksheets** – By checking the unassigned flag, applications that include the potential for non-configured worksheets can allow those worksheets to be displayed within the Fusion Client while continuing to maintain the ability to hide worksheets irrelevant to the task at hand for workbooks that do not contain non-configured worksheets.
   - **Non Authorized Users** – Choose **Disable Task** to allow unauthorized users to see the task but not access it. Choose **Hide Task** if you do not want unauthorized users to see the task at all.

Adding a Hyperdynamic Step to the Taskflow

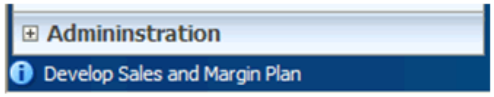
To add a hyperdynamic step to a hyperdynamic task you have created, perform the following steps:

1. Select the hyperdynamic task and click Add.



Adding a Step to a Task

2. A step named Step1 appears in the navigation pane. Click **Step1**. The Step Properties appear in the details pane.
3. Enter the following information in the Step Properties:
   - **Label**: Enter the name of the step as you want it to appear in the RPAS Fusion Client. This is required.
   - **Description**: Enter the description of the step. This description appears when the user rolls the cursor over the step name in the RPAS Fusion Client.
   - **Instructions**: Enter instructions that explain what to do in the step. These instructions appear below the task pane.



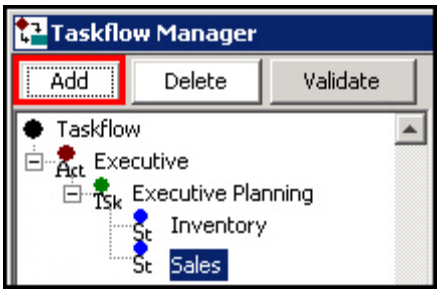
Instructions in the RPAS Fusion Client

   - **Worksheets**: Enter the names of the worksheets that can be visible within the step.
   - **Custom Menu**: If you want to include a custom menu in this step, select it in the Available area and then use the arrow to move it to the Selected area. You can change the order of the custom steps by using the up and down arrows.

Adding a Hyperdynamic Tab to the Taskflow

To add a hyperdynamic tab to a hyperdynamic step you created, perform the following steps:

1. Select the hyperdynamic step and click **Add**.



Adding a Tab to a Step

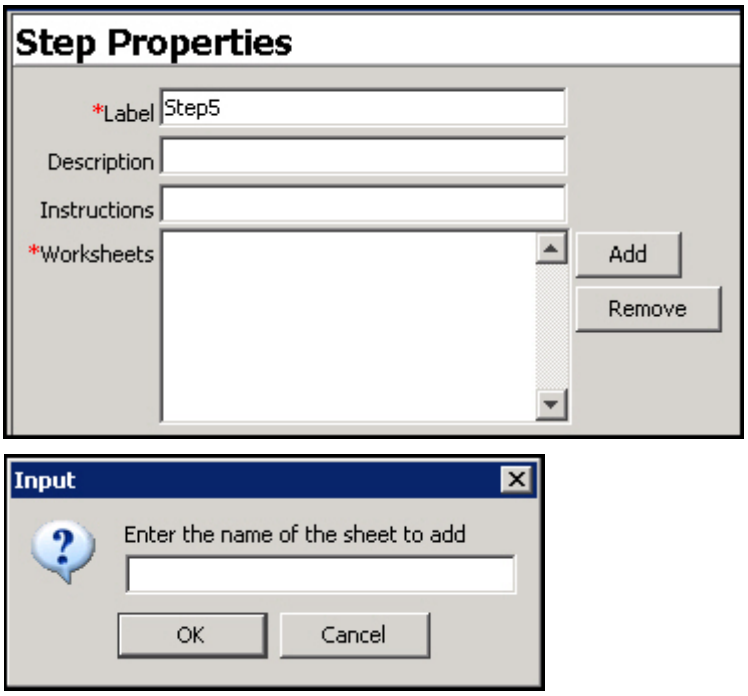
2. Tab1 appears in the navigation pane. Click **Tab1**. The Tab Properties appear in the details pane.
3. Enter the following information for the tab:
   - **Label**: Enter the name of the tab as you want it to appear in the RPAS Fusion Client. This is required.
   - **Worksheets**: Select the worksheets that will be utilized in this tab.

After you have entered the tab properties, the new tab is created.

Adding Worksheets to a Hyperdynamic Step or Tab

To add a worksheet to a hyperdynamic step or tab, perform the following steps:

1. Select the hyperdynamic step or tab. It appears in the detail pane.
2. To add a worksheet, click the **Add** button and specify the name of the worksheet in the pop-up dialog.



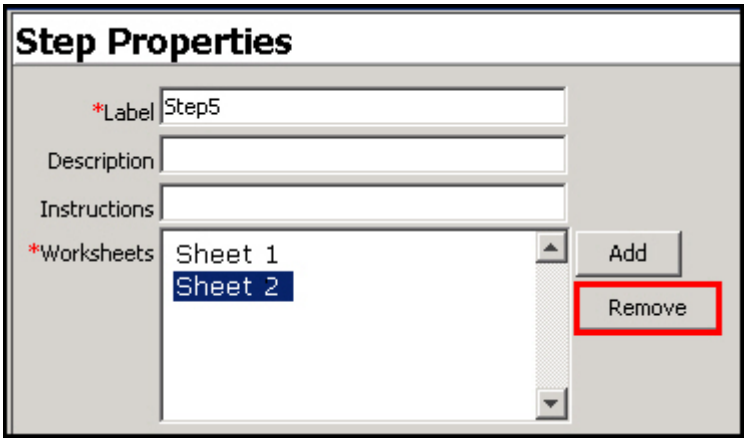
Adding a Worksheet to a Hyperdynamic Step or Tab

3. When finished, click **OK**.

Removing a Worksheet from a Hyperdynamic Step or Tab

To remove a worksheet from a hyperdynamic step or tab, perform the following steps:

1. Select the step or tab that contains the worksheet you want to remove. The details of the step or tab appear in the detail pane.
2. Select the worksheet to remove and click **Remove**.



Removing a Worksheet from a Hyperdynamic Step or Tab

The selected worksheet is removed from the step or task.

Creating a Multi-Solution Taskflow

There are two goals:

- The taskflow is usable without further modification in a single solution environment.
- The taskflows can be easily combined, manually, to form a multiple solution environment:
  - In the simplest form, the activity groups from each solution's taskflow can be put into one taskflow, and the resources concatenated, to produce a simple, usable, multi-solution taskflow.
  - This must also ease the process of producing a more thoughtfully integrated taskflow, for example, with activity groups like "Pre-Season Planning" that comprise tasks from each solution.

Keep the following in mind:

- Solution ID – The solution ID should be standard for the application, for example, "mfprtl". This will be used to make the taskflow definitions and resources unique to this application. This avoids conflicts when combined with other applications into an integrated taskflow.
- Name and description of the top-level activity group – Activity groups are a higher-level grouping of activities in the taskflow. For a single solution taskflow, a single activity group containing the entire application is produced.
- Each task can specify domain type restrictions:
- master – create the workbook in the master domain only.
- locals – create workbooks in local domains only.
- both – workbooks may be created in either master or local domains. This is the default if not specified in the taskflow definition.

A simple domain always qualifies, regardless of this setting. The workbook template may have restrictions, as well, which are also applied.

When combining taskflows, the taskflow xml and resource file need to be updated:

- All resource IDs must be prefixed by the solution ID, both in the taskflow xml and the .properties resource file. This makes each definition specific to this application, so there will be no conflicts when combined with taskflows of other solutions.
- A top-level activity group needs to be created. The <activity> tags need to be inside of the <activity_group> definitions. This provides a higher level of grouping:

```
<activity_group>
  <name>mrprtl.ActivityGroup1</name>
  <description>mfprtl.ActivityGroup1.Desc</description>
  <order_num>1</order_num>
    ... all activity definitions ...
</activity_group>
```

- Resource definition for the solution needs to be created:

```
{solutionId}.Solution.label={solution label}
```

You need to replace {solutionId} and {solution label} with the actual values.

- Resources definitions for the activity group need to be created:

```
mfprtl.ActivityGroup1=MFPRetail
mfprtl.ActivityGroup1.Desc=MFP Retail
```

- Each and every task needs to specify the solution ID via a <solution> tag:

```
<task>
  <name>mfprtl.Activity1.Task1</name>
```

```
<description>mfprtl.Activity1.Task1.Desc</description>
<solution>mfprtl</solution>
    ...
```

- Any task that has a domain type restriction must include the <domain_types> tag:
  ```
  <domain_types>locals</domain_types>
  ```

If you are transforming old taskflows manually, each <task> needs a <solution> tag before this.

- Note that the current generated taskflows include generated resource IDs which include each level of grouping, that is, Activity1.Task1.Step1, etc. In 14.0, the Solution ID is included. It is up to you whether you want to also include ActivityGroup. For single solution taskflows, there are activity group (name, label, description) properties on each activity, and you can produce multiple activity groups by assigning activities to different ones. Activity groups are an additional level of grouping that can be useful when taskflows are combined (that is, useful in that situation, but not exclusively), which is a manual process.
- There still may be value in using tools to manipulate the existing XML and resource files. For example, you might want to rename the solution ID to provide a test bed solution, or apply a solution ID to a previously translated resource file.

  Note: You use whatever tools you are comfortable with to transform XML and text files.

- Not directly related to the taskflow, but the resource file is used to provide "friendly names" for RPAS domains, as they are not available from the server. This is more an installation/administration issue:
  ```
  mfprtl.Domain.simple.label=MFP Retail Simple
  mfprtl.Domain.master.label=MFP Retail Master
  mfpftl.Domain.domainidx_0.label=MFP Retail Local 0
  mfprtl.Domain.domainidx_1.label=MFP Retail Local 1
  mfprtl.Domain.domainidx_2.label=MFP Retail Local 2
  ```
  etc.

Refer to the *Oracle Retail Predictive Application Server Administration Guide for the Fusion Client* or the *Oracle Retail Predictive Application Server Installation Guide* for additional information about domain partitioning.

Taskflow Naming

The following items are helpful for the names used in your taskflow. Understanding these can help you avoid a lot of unnecessary work combining taskflows and manipulating multi-solution taskflows.

- <name> and <description> for <activity_group>, <activity>, <task>, and <step> are all IDs for resource strings that are defined in MultiSolutionBundle.properties. They should not conflict or the display will be wrong.
- <name> for <activity_group>, <activity>, <task>, and <step> are all IDs that need to be unique within their parent.
- <solution> within <task> refers to the Solution ID where the workbook template resides (see <wkbk_template>). This Solution ID must have a connection mapping in Foundation.xml (<solution>'s <name> must match) or this task will not be used.

As long as you pick different Solution IDs for each Project in ConfigTools, the automatic naming is such that there should be no conflicts. This is true even if you move activities, tasks, and etc. around the taskflow XML and the fully qualified names no longer reflect

their new location. For example, Sol1.ActivityGroup1.Activity2.Task.4 can be moved inside Sol2.ActivityGroup2.Activity1 without the IDs all having to be fixed up. They are still unique as resource IDs and the task name does not conflict with any other tasks inside its new parent Sol2.ActivityGroup2.Activity1.

Creating the Taskflow for 14.0 from a Taskflow Created in a Release Earlier than 13.3.1

If you have a taskflow created in a release earlier than 13.3.1 that you want to use in 14.0 (without running through ConfigTools for some reason), at minimum you need to do the following:

- Add a single <activity_group> around the existing <activity> tags to group them.
- Add a <solution> tag with the solution ID to each task in the taskflow.
- Add resources to the property file for the new activity group, for example:
- solution1.ActivityGroup1=sample activity group label
- solution1.ActivityGroup1.Desc=sample activity group description
- Add a resource to the property file for the solution's label:

```
solution1.Solution.label=sample solution label
```

- You probably want to qualify all the <name> and <description> IDs by putting the "solutionId." in front of them. Then do the same in the properties file so they match up.

Upgrading retailers may have multiple taskflows on the same server, where the users who switch between them use profiles. In this case, you probably want to construct a multi-solution taskflow that includes each solution. The simplest way to do this is to combine taskflows from Configuration Tools or construct as above by putting each <activity_group> in the multi-solution taskflow xml, one after another, and then concatenating all the .properties resource files.

4

Solutions

Working with Solutions

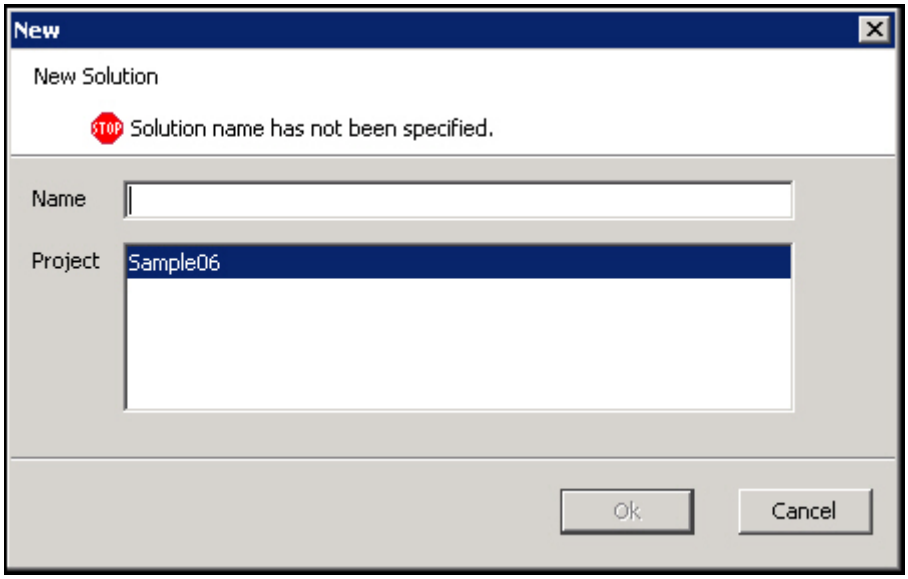
Overview

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, measures, rules, workbooks, and wizards are defined. Once a solution is created, it can be moved from one project to another.

> **Note:** Some solutions, such as Curve, Grade, and RDF, have configuration steps that are specific to those solutions. For more information, see the corresponding configuration guide for the solution.

Create a Solution

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



New Window

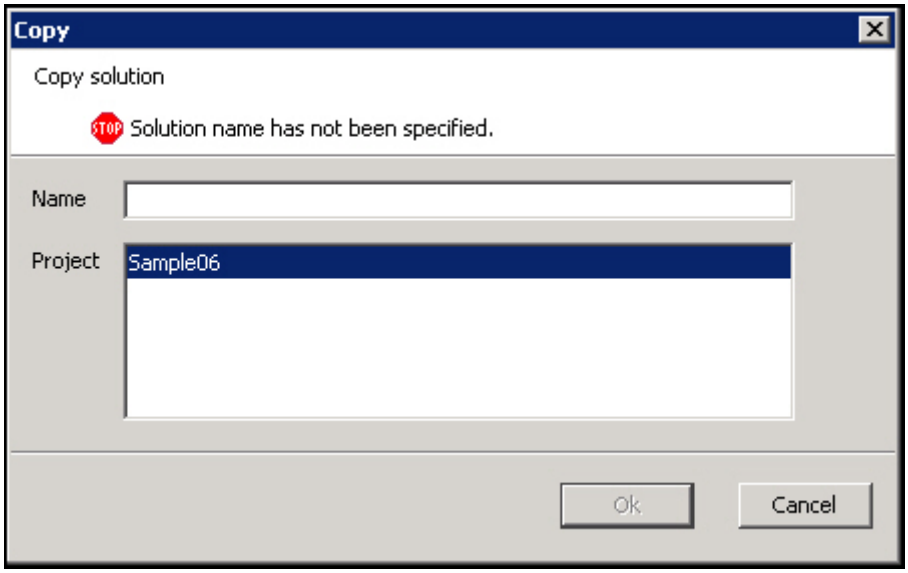
1. In the **Name** field, enter the name of the solution.
2. In the **Project** field, select the project in which the solution is to belong. There will be multiple projects listed if multiple projects are currently open.
3. Click **OK** to save any changes and close the window.

Copy a Solution

Perform the following procedure to copy a solution:

1. Select the solution to be copied.

2. Right-click in the Configuration Components pane, and select **Copy**. The Copy solution dialog box appears.



Example of Copy Solution Dialog Box

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

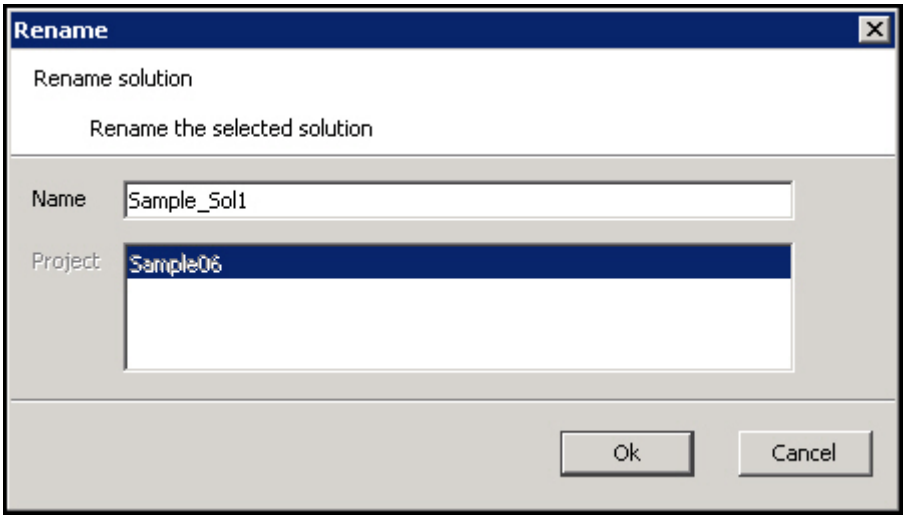
> **Note:** This is the name that the solution is called after it has been copied.

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

Perform the following procedure to rename a solution:

1. Select the solution to be renamed.
2. Right-click in the Configuration Components pane, and select **Rename**. The Rename dialog box appears.



Rename Dialog Box

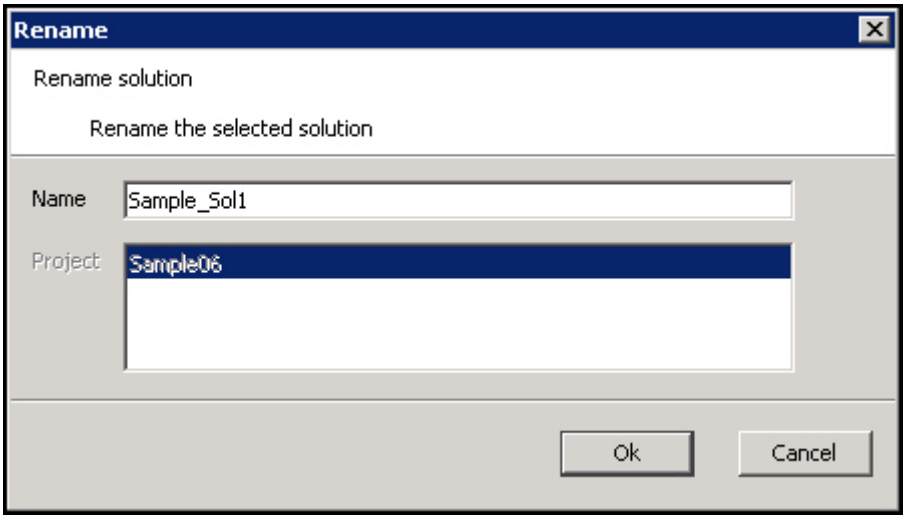
3. Delete the old name from the resulting field, and type the new name.
4. Click **OK** to save the new name.

Move a Solution

Perform the following procedure to move a solution:

> **Note**: The Move a Solution operation is only available when multiple projects are open.

1. Select the solution to be moved.
2. Right-click in the Configuration Components pane, and select **Move**. The Move dialog box appears.



Example of Move Dialog Box

3. Type the name of the solution in the text box.

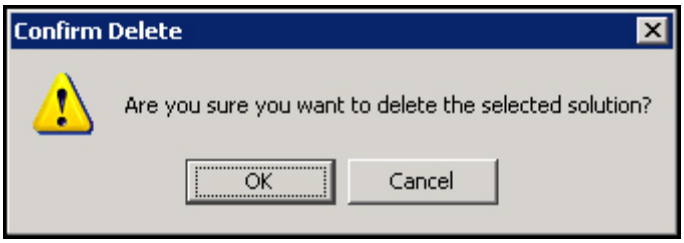
> **Note:** This is the name that the solution is called after it has been moved.

4. Select the destination project for the solution from the resulting **Project** list.
5. Click **OK** to move the solution to the specified project.

Delete a Solution

Perform the following procedure to delete a solution:

1. Select the solution to be deleted.
2. Right-click in the Configuration Components pane. The Confirm Delete dialog box appears.



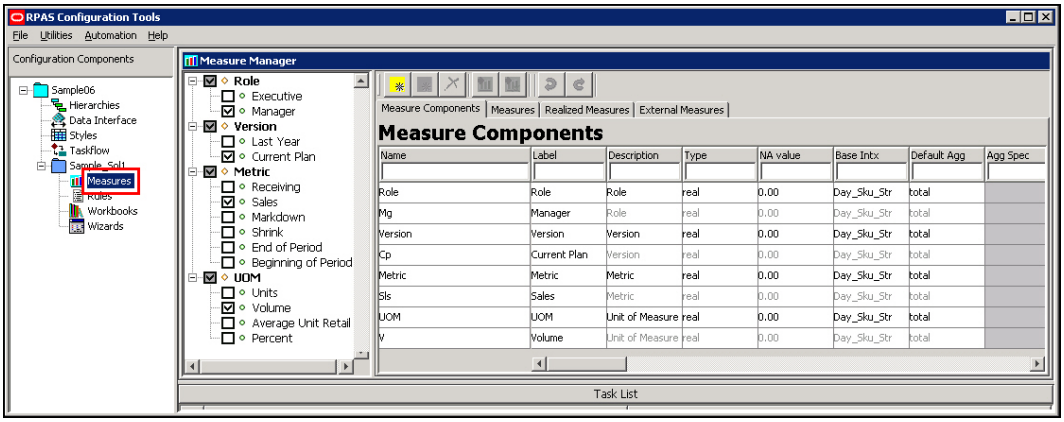
3. Click **OK** to complete the deletion.

Measures and Components

Measure Manager

Overview

The Measure Manager window allows you to define major and minor components of measures and to specify properties for each component. Once the component structure is defined, the Measure Manager generates measures by combining the components that are selected. You may then select (realize) the valid measures and further update the properties for individual measures.



Example of Measure Manager Window

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 the component tree. 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 at that lower level.

Measure Components

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

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

Once major and minor components are defined, the Measure Manager generates measures that are based on the combination of selected components. These measures cannot be used elsewhere in the configuration tools until the valid prototype measures are Realized (see the following section on Realizing and Unrealizing measures for more information).

> **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 the configuration administrator to easily group measures together and to set measure properties at higher levels.

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.

Measure Naming Conventions

All components used in RPAS configurations must adhere to the following naming convention:

Characters allowed:

- Capital and lowercase letters (A, b…Z)
- Numerals (1, 2, 3…)
- Underscore (_)

With the exception of underscores, no non-alphanumeric characters are allowed. Measure component names must start with a letter. Spaces are not allowed.

> **Note:** Since the names of realized measures are limited to 30 characters, and those names are constructed by concatenating the names of the components from which the measure is built, it is usually good practice to abbreviate the names of components where necessary.
>
> However, there is no limit on the number of characters for measure labels.

Measure Component Design

The following two basic principles must be kept in mind to make the Measure Manager as powerful as possible.

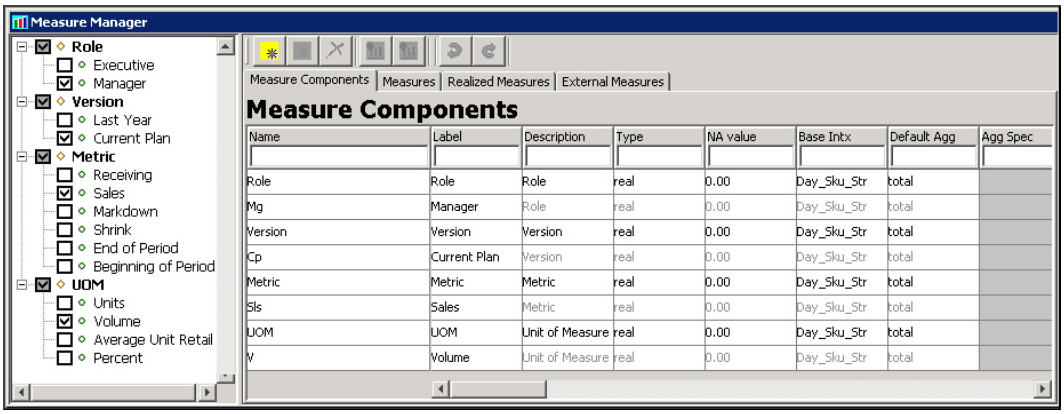
1. Major and minor components should be designed with the idea of maximizing the inheritance of properties, and minimizing the amount of property overriding.
2. Use minor components to make measure definition manageable.

For example, consider a configuration that has 2000 measures and 1500 of the measures is of data type real. Avoid grouping all 1500 measures into a single minor component because smaller subgroups of 1500 measures cannot be easily edited. Minor components

can also have minor components, so within the 1500 measures, the configuration administrator may 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 the configuration administrator to easily view and manage every resulting measure for that minor component.

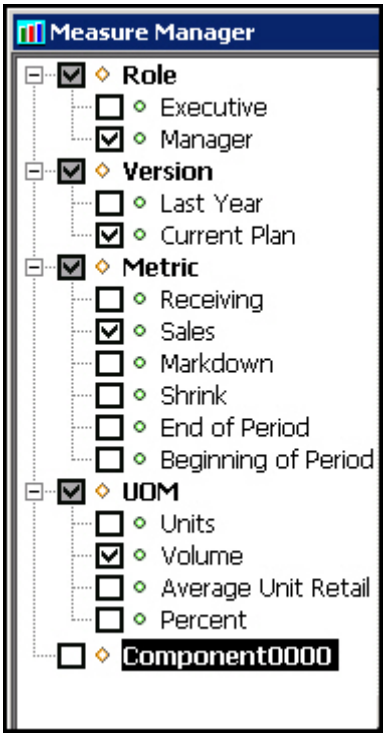
Create a Major Component

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



Example of Measure Manager Window - Measure Components Tab

1. Right-click in the left-hand pane of the Measure Manager window and select **Add Major Component**, or click the **Add Major Component** button. The major component is displayed in the Measure Manager navigation tree with a default name.

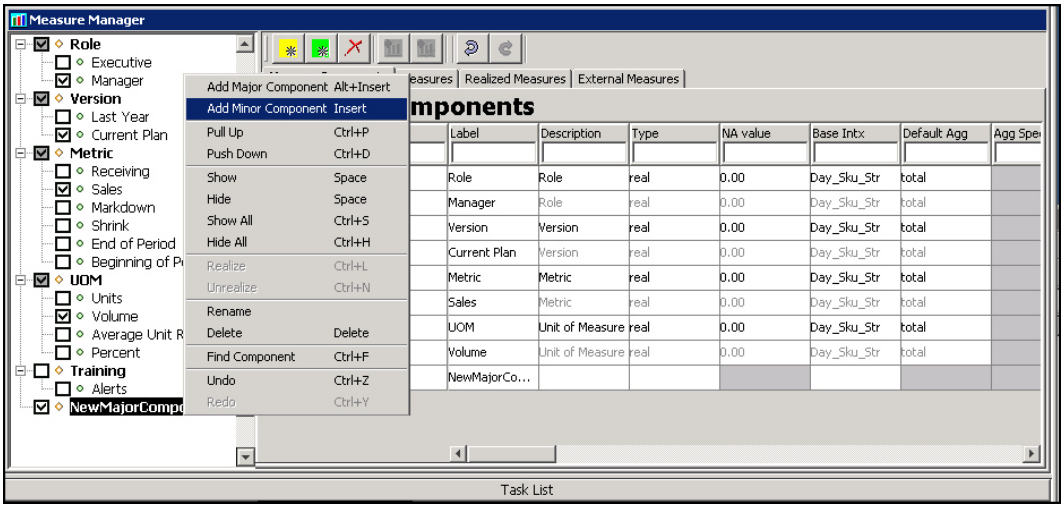


Example of New Major Component

2. To change the component name, select right-click on the component and select **Rename**, or select the component from the navigation tree and then modify its **Label** field from the **Measure Components** tab.

Create a Minor Component

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

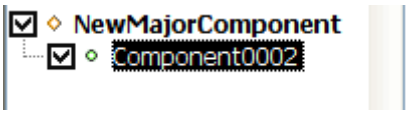


Example of Major Measure "NewMajorComponent" Selected

1. Select the major or minor component that the new minor component is to be added beneath. In the example above, **NewMajorComponent** is selected.

2. Choose one of the following methods:
   - Right-click the Measure Definition navigation tree, and select **Add Minor Component**.
   - Click the **Add Minor Component** button.
   - Press the **Insert** key.

   The minor component appears in the Measure Manager navigation tree with a default name.



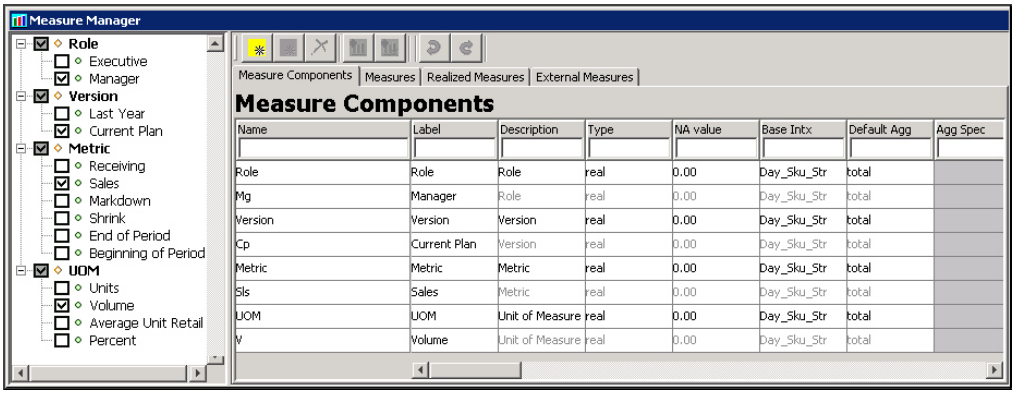
Example of New Minor Component

3. To change the name, right-click on the component from the navigation tree and select **Rename**. You can also select the component from the navigation tree and modify the **Label** field from the **Measure Components** tab.

Defining Measure Component Properties

Perform the following procedure to define the Measure Component properties:

1. Once components have been created, open the Measure Manager and select the **Measure Components** tab.
2. Select the major or minor components you want to view from the Measure Manager navigation tree. The selected components appear in the Measure Components tab.



Example of Measure Manager Window - Measure Components Tab

3. Specify the information for the component properties (for example, Name, Label, Description, etc.), which will apply to the measures that are inheriting property values from the component.

> **Note:** The values that are entered for major components are inherited by the child minor components and the auto-generated measures. Not all properties need to be entered for all components. 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.

Measure Component Properties

This section describes the fields displayed in the Measure Components tab.

Name

The name (identifier) of a component is used to identify the component within the Configuration Tools. Measure names are built by concatenating the names of the components from which the measure is built. They are concatenated in the order (from top to bottom) of the sequence that the components appear in the list of components. You can override the measure name by manually entering a new name in the Name field. Measure names can be up to 30 characters in length. They can include letters and numbers, but must start with a letter.

Label

The label of the component is used to generate measure labels in a similar way that measure names are generated. Labels are displayed to RPAS end users. There is no maximum size limit, but keep the 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. The configuration administrator can enter any text to provide more information beyond the measure label to the end user. The description can be viewed by the end user in the RPAS user interface.

Type

Select one of the following data types:

- **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.
- **Boolean** – True or false values, which are typically used for flags and indicators.
- **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.

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" of this document 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 this 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. A Labeled Intersection may also be selected as the base intersection of a measure. The Labeled Intersection field is populated based on the Labeled Intersections defined through the Labeled Intersection dialog accessed in the Hierarchy Definition manager.

> **Note:** RPAS imposes a limit of 5 dimensions that can be defined in a measure's base intersection.

Default Agg

The default aggregation method should be selected from the valid aggregation methods for the component. The valid aggregation method depends on the data "Type" selected for the component. See Appendix B – Calculation Engine Users Guide, and Appendix C – Rules Function Reference Guide for more information on aggregation and spread methods.

> **Note:** Only measures with an aggregation type of ambig, pst, or pet can be aggregated from below the partition levels to above the partition levels in a global domain.

Agg Spec

This hybrid aggregation mechanism is designed to allow the configuration administrator to specify a complex method to aggregate the values of a measure. It allows a different aggregation method to be specified 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 as follows:

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

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

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

In this example, the user is allowed visibility to and control over the mechanism by which pst is performed.

A brief description of the user interface 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 select Ctrl-Space to launch it).
- This dialog looks very similar to the standard wizard 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 as follows:
- Recalc or hybrid cannot be used within an agg spec.
- First and last can be used only on the Calendar hierarchy only.
- An aggregation type must be specified for each hierarchy in the intersection.
- If a value is to be typed into agg spec, 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. Unlike Aggregate procedure, the "recalc" aggregation type is not supported for any hierarchy for am measure using the hybrid aggregation type. Measures that use the hybrid aggregation type cannot be aggregated from a local domain into the global domain.

Default Spread

The default spread method should be selected from the valid spread methods for the component. The valid default spread method depends on the data “Type” selected for the component. See "Appendix B – Calculation Engine Users Guide" and "Appendix C – Rules Function Reference Guide" of this document for more information on aggregation and spread methods.

> **Note:** The spread method can be overridden 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 ability of the measure at the base level to be modified. The available options are **read** or **write**.

Agg State

The editability of the measure at the aggregate level, which are all intersections above the base intersection (read or write). Set the Base State to write and the Agg State to read for

those measures that need to be manipulable, 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. 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-manipulable that protection processing would otherwise allow to be manipulated). See "Appendix B – Calculation Engine Users Guide" and "Appendix C – Rules Function Reference Guide" of this document for more information on the Agg State of measures.

Database

The physical location in the file system of the database that stores the data for this measure. Those measures that contain data 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 attached to the beginning of the entry at the time of installation (for instance, the entry "Sales" will be registered as "data/Sales")

> **Note**: The presence of multiple measures within a single database can create unnecessary contention when the measures' data are being updated as part of a workbook commit or batch calculation. In order to avoid this, measures that are updated through batch calculations or workbook commits should be configured with unique values for the database property so that they each get a separate database. Measures that are only updated by data load and that are read-only for workbook and batch operations can be grouped together without causing contention.

View Type

The View Type field holds properties for two types of measures:

1. Those that are calculated when viewed
2. 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 must 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 must not have a database assigned. See "Appendix B – Calculation Engine Users Guide" of this document for more information.

Synchronized measures are in effect, views of two or more other measures where changes and lock to those other measures are immediately reflected in the synchronized measure (and vice versa).

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** – Gets Period 1 from the measure (similar to pst along calendar dimension) [for example, os synchronized with bop].
- **sync_last** – Gets 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. Depends on the View Type.

Insertable

This field 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 *Oracle Retail Predictive Application Server Administration Guide* for additional information about measure security.

> **Note:** The Extra Measures wizard is not available by default for every workbook; it must be configured as a custom wizard page.

Non-Translatable

The selected value of true will represent that a measure should be omitted from generated translation resources. The false or unspecified value will represent that a measure should be included in these resources and all legacy configurations will therefore treat all measures as translatable unless further configuration is performed.

UI Type

The UI Type property affects how the user interacts with a measure within the client. There are two supported values for UI Type; by default, UI Type is unspecified, meaning that the measure exhibits no special behavior within the client.

- **Picklist** – When a measure is given a UI Type of picklist, it will no longer accept edits within the client. Instead, the user will be provided with a drop-down box that contains a set of valid values. Selecting an item in this drop-down will set the cell of the measure to the appropriate value. For picklist measures, the contents of the drop-down box are determined by the range attribute.

- Media – When a measure is given a UI Type of media, the Fusion Client will not display the values of the measure's cells. Instead, it will examine the contents of a cell to extract image location information, retrieve the appropriate image resource and display that image in the cell. In order for a measure to have a UI Type of media, that measure must be a string type measure.

> **Note:** For information regarding the values that should be loaded into a media UI Type measure; please consult the Media measures section of the RPAS Administration Guide.

Range

Specify an allowed range for the measure at edit time. For numeric values, the syntax is as Lower Bound : Upper Bound. If the RPAS UI user attempts to enter a value in the RPAS Client outside of this range, the modification is rejected. For string measures, any entry in this field is ignored.

For date measures, the lower and upper bounds specify both the date and the time. The default time for the lower bound is 12:00:00 AM. The default time for the upper bound is 11:59:59 PM. The time portion of the bounds is included regardless of the style setting because users can change the style setting on the client side.

For numeric or string measures with a UI Type of "picklist" (numeric and string), 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.

> **Note:** "," and ":" are reserved characters for picklist and range definitions of a measure with a UI Type of picklist. These characters cannot be used to define the LABEL part of a picklist Value/Label pair.

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. The configuration administrator sets 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 *Oracle Retail Predictive Application Server Administration Guide* for details of how this property is used in the `loadmeasure` 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 must be used instead. If specified by the configuration administrator, the Lower Bound and Upper Bound properties must be a valid, realized measure name 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 in 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 display a specific value. To configure the values to display null, assign Null as the SP Value Type property. When Null is configured, the cell will be blank in the RPAS Client 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, you should select User Entered for the SP Value Type, and enter the value to be displayed in the SP Value field.

For the Special Value field, the entry in this field must be of the same data type as the measure, and validation is enforced in this field.

- For a Boolean measure, when the Special Value Type field is set to User Entered, the only valid entry for the Special Value field is either true or false.
- For a Date measure, when the Special Value Type field is set to User Entered, a date in the format of YYYYMMDD can only be entered by the user.
- By default, each measure will be registered with Cell Value as the default Special Value behavior. For cases where a Special Value Type setting other than 'User Entered' is used, but a Special Value entry is provided, the Special Value entry will

not be used. Instead, the measure will be registered with Cell Value as the default behavior.

- When Special Value Type is set to User Entered, but a Special Value entry is not entered, the Special Value Type will not be used. Instead the measure will be registered with Cell Value as the default behavior.
- When a domain has been built that includes a measure with a special value setting, and that special value setting for that measure is removed. When the domain is patched the measure will get updated with Cell Value as the default special behavior.
- RPAS allows for the special value measure property to be updated.

Dim Attr Type

This checkbox option is used to indicate that the measure must 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).

The following requirements must be met for a measure to be eligible to be a dimension attribute:

- 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 RPAS Client displays 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 RPAS Client displays the measure label.

Allowed Aggs

The set of the allowable aggregation methods for the measure based on the measure data type. You can add so called extended measures to RPAS Client views that are normal measures, but with aggregations based on different aggregation methods. The aggregation methods that are 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.

Style

One of the styles defined within the style tool may be specified as the default style to be used to display measures based on this component inside the User Interface. 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. right click and select Alert Manager to access the alert categories defined in the Alert Manager. See the *Oracle Retail Predictive Application Server User Guide* for a description of alerts.

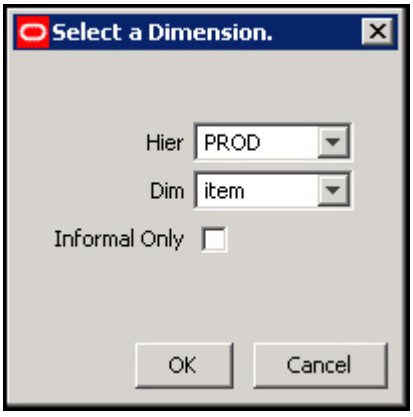
Alert Expression

The evaluation method for this alert measure. Double-click this cell to open the Select Evaluation Method for Alert dialog box. This dialog 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 in which its first expression contains the given measure on its left-hand side. See the *Oracle Retail Predictive Application Server Administration Guide* for more information on Alerts.

Select Evaluation Method for Alert Dialog Box

Single Hier Select

This property is only valid for components that have a Type (data type) of string. It specifies that cell contents are to be entered by users using the "Single Select Widget." This is a widget in the RPAS Client that presents the end user with a view of the positions along the dimension set in the configuration. The user may then select any single position. You must select the hierarchy and dimension whose positions will be displayed to the user in the RPAS Client.



Select a Dimension Dialog Box

Filename (read only)

Measures included into the Data Interface Manager 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 cannot be modified and is displayed for diagnostic purposes.

Load Intx (read only)

Measures included in the Data Interface Manager 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 cannot be modified and is displayed for diagnostic purposes.

Clear Intx (read only)

Measures included in the Data Interface Manager may have a clear intersection [Clear Intx] specified. This field displays the value, if one exists, for the clear intersection of the measure. See the section on the Data Interface Tool for details on data interface properties. This property cannot be modified and is displayed for diagnostic purposes.

Start (read only)

Measures included in the Data Interface Manager 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 cannot be modified and is displayed for diagnostic purposes.

Width (read only)

Measures included in the Data Interface Manager 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 cannot be modified and is displayed for diagnostic purposes.

Load Agg (read only)

Measures included in the Data Interface Manager 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 cannot be modified 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 is marked as display only, that fact will be reflected in this field. This property cannot be modified and is displayed only 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 cannot be modified and is displayed for diagnostic purposes only.

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 Manager. This property cannot be modified and is displayed only for diagnostic purposes.

Edit Components

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

Move Components

1. From the Measure Manager navigation tree, select the component to be moved.
2. Drag the component to the new location and release it.

> **Note:** A minor component cannot be moved to a different major component.
>
> The configuration administrator cannot move a component so that it would be a descendent of another component that is used in the specification of a realized measure.

Push Components Down

1. From the Measure Manager navigation tree, select the component to be pushed 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:** A major component cannot be pushed down.

Pull Components Up

1. From the Measure Manager navigation tree, select the component to be pulled up.
2. Right-click and select **Pull Up**. The component is pulled up one level in the component hierarchy.

> **Note:** A minor component cannot be pulled up to become a major component, nor can a major component be pulled up.

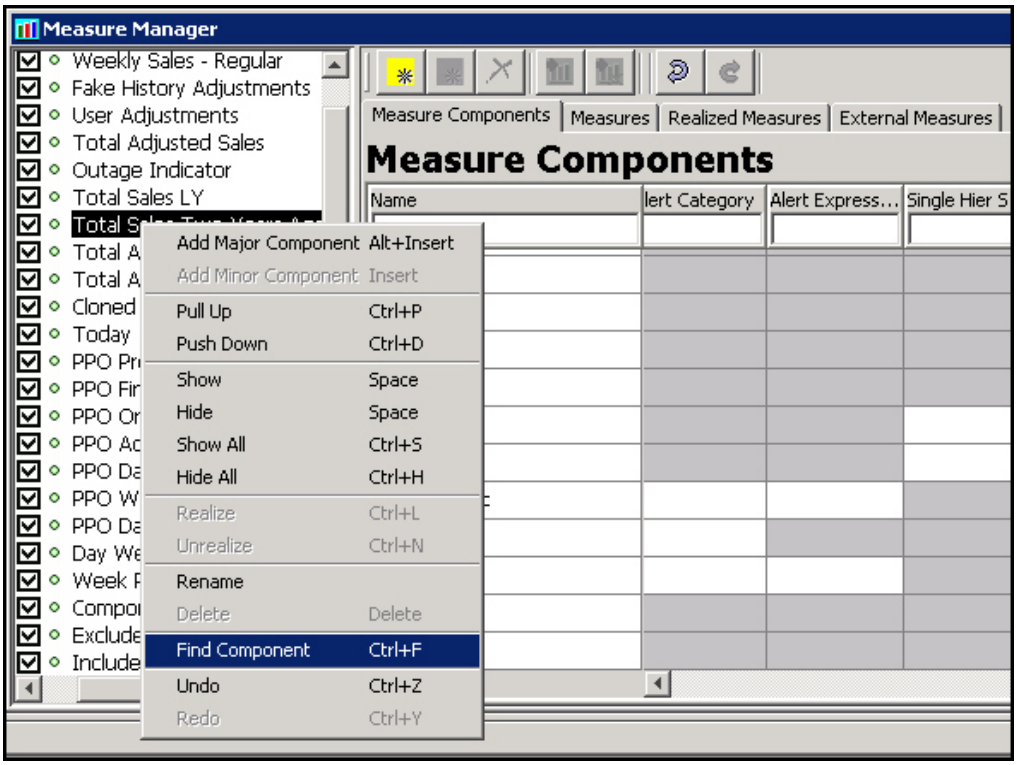
Display or Hide Components

- 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 Measure Manager navigation tree and select **Find Component**. The Input dialog box appears.



Example – Find Component Menu Option

2. Type in the name of the desired component and click **OK**. 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. If there is no component with the exact name entered, the tree will not scroll.

Rename a Component

Navigate: From the Measure Manager navigation tree, select the component to be renamed.

Choose one of the following methods:

- Right-click in the Measure Manager navigation tree and select **Rename**. Type the new name for the component.
- Double-click the component, and type the new name.
- Select and change the name of the component in the Measure Component tab.

> **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: From the Measure Manager navigation tree, select the component to be removed.

Choose one of the following methods:

- Select **Remove Component** .
- Press the **Delete** key.
- Right-click in the Measure Manager navigation tree and select **Delete**.

The component is 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.

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. Oracle Retail Predictive Application Server then notifies the user of the "true" conditions, and it allows workbooks to be built 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 Oracle Retail Predictive Application Server 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 *Oracle Retail Predictive Application Server Administration Guide*.

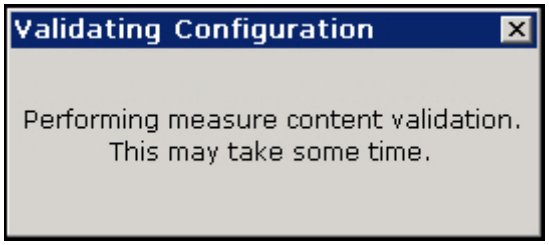
Measure Validation within the Measure Manager

The Measure Manager 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 Manager.

In addition, the validity of rules, rule groups, and workbooks 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 a significant amount of time.

In order to facilitate the configuration process, this second form of validation does not occur as edits are made in the Measure Manager. Instead, this validation is deferred until Measure Manager is exited by selecting a different tool/option from the Configuration

Components pane. At this time, a dialog briefly displays to indicate the validation process is running.



Validation Configuration

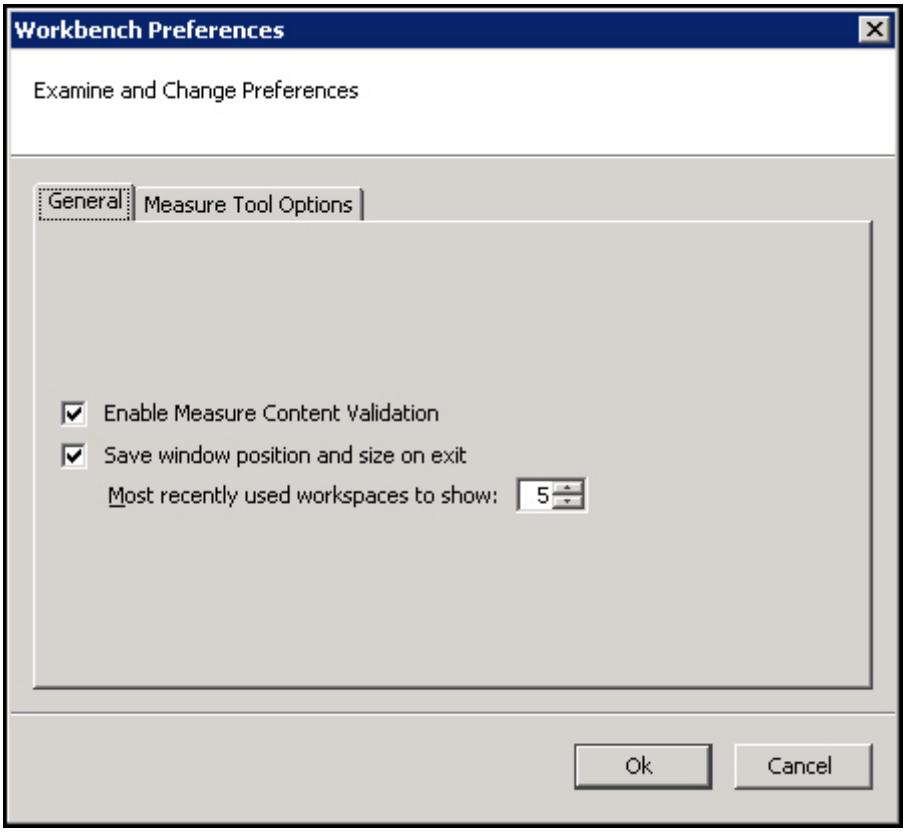
When the full suite of solution level measure content validations is complete, the dialog is no longer displayed and the selected tool is activated.

> **Note**: For fast performing validations, dialog may only be displayed for a few seconds. If the validation process is lengthy, the dialog will be displayed for a considerably longer time.

Disabling Measure Content Validation

For some cases you may disable the full validation (for instance, when performing a number of changes to measure properties that require switching back and forth between multiple tools). Validation can then be manually initiated from the Rule Definition window by selecting **Perform measure content validation** from the Rule Definition toolbar. Perform the following procedure to turn off real-time validation.

1. Select **File** – **Tools Preferences**. The Workbench Preferences dialog box appears.



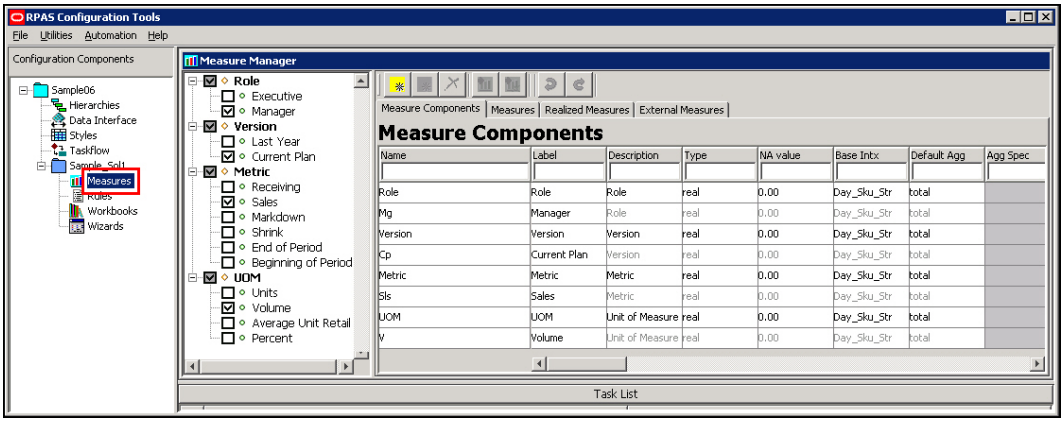
Workbench Preferences – General Tab

2. Deselect Enable Measure Content Validation.
3. Click **OK**.

Working with Measures

Overview

The Measure Manager 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, select the components that will be used to construct measures. The Measure Manager will generate measures for all of the combinations of selected components. This saves you from the tedious task of manually creating all required measures.

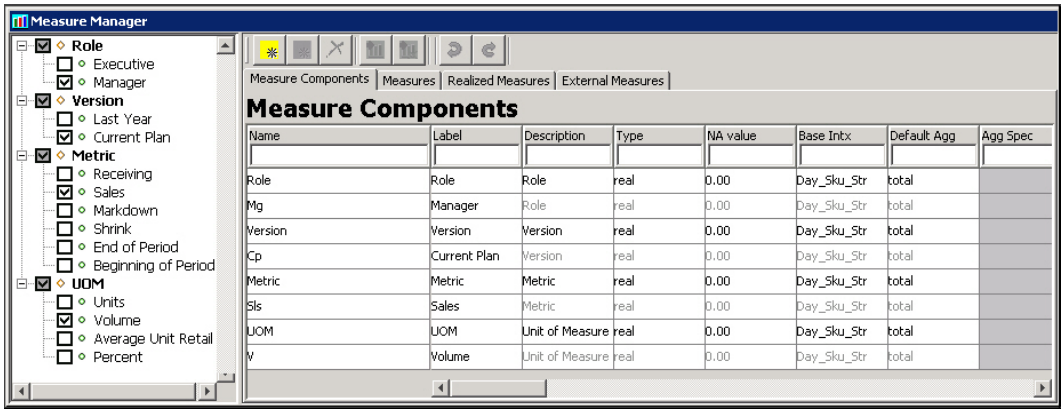


Example of Measure Manager Window

You may then override the properties for individual measures by entering them the same way as on the Measure Components tab. Once properties are overridden 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. An overridden value can also be restored to its inherited value or you may override an inherited value to be unspecified.

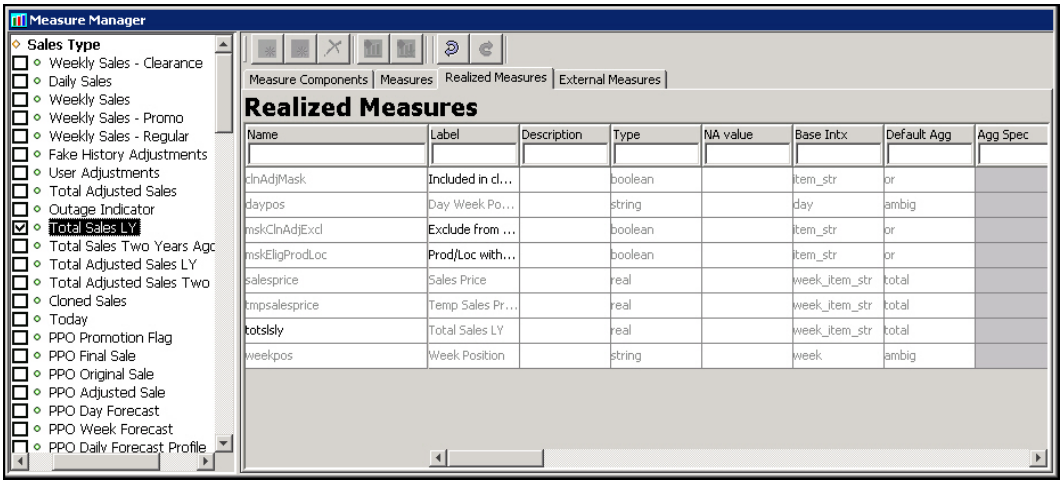
When a measure is auto-generated by the Measure Manager, it cannot be edited or used in any other configuration component such as rules and workbooks until it is realized. A measure does not need to be realized if it is not going to be used.

The **Measures** tab displays all auto-generated measures for the selected components.



Example of Measure Manager - Measures Tab

The **Realized Measures** tab only displays those measures that are realized.

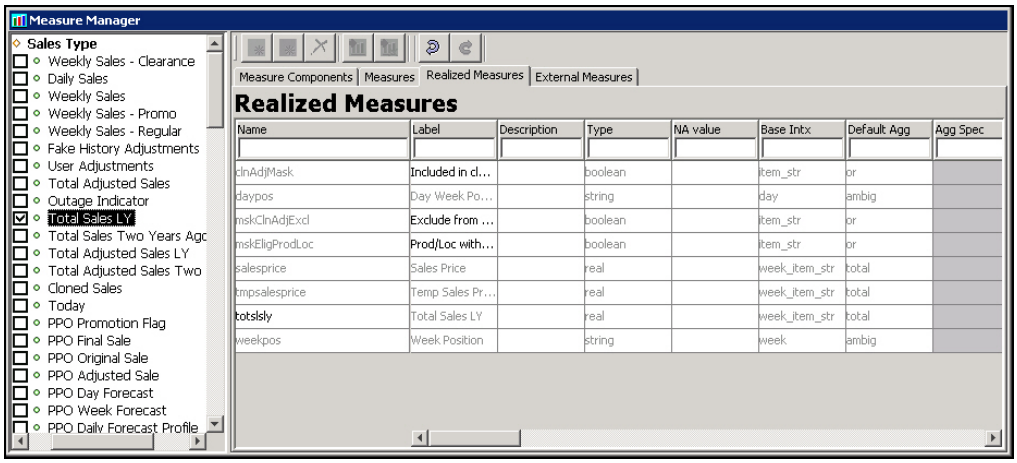


Example of Measure Manager - Realized Measures Tab

Realize and Unrealize Measures

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

1. Select the **Measures** tab.



Example of Measure Manager - Measures Tab

2. In the components tree, select the check boxes for the components that will be filtered. The Measure Manager will show measures using all of the combination of selected components. 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 for the measure(s) to be realized.
2. Select the check box in the **Realized** column for each auto-generated measure to be realized.

Unrealize a Measure

1. Select the components that are used for the measure(s) to be unrealized.

2. Deselect the check box in the **Realized** column for each auto-generated measure to be unrealized.

Rename a Measure

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

1. Select the components that are used in the measures to be renamed.
2. In the **Measures** tab, or the **Realized Measures** tab, click the name of the measure that is to be renamed.
3. Type the new name for the measure.

> **Note:** The measure must be realized before it can be renamed.

Show all Measures

Navigate: In the Configuration Components pane, 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 auto-generated measures may not be displayed. In this case, the configuration administrator will receive an error message indicating that some measure components should be deselected.

Hide Measures by Component

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

1. In the Measure Manager window, select the **Measures** tab.
2. Choose one of the following methods:
   - Deselect the check boxes next to the components to hide.
   - Select the component used in the measures to hide.
3. Right-click in the Measure Manager navigation tree and select **Hide**, or press the spacebar.

Hide All Measures

Navigate: In the Configuration Components pane, 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 navigation tree, and select **Hide All**.

Sort Measures by Property Value

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

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

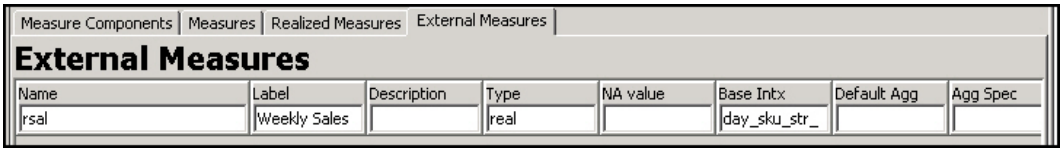
1. In the Realized Measures tab, click in the filter field at the top of the table for the property of the measures to be filtered.
2. Enter the value on which the measures are to be filtered.

> **Note:** This field is case sensitive.

External 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 Manager allows the "import" of a measure defined in a different solution (but not a different project) into the current solution. These measures (called external measures) then become visible in the External Measures tab of the Measure Manager. 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, the configuration administrator may want to modify a writable measure so that it is read-only in the solution into which it is imported.



Example of Measure Manager - External Measure Tab

Within the Measure Selectors present in the Rule and Workbook tools and the Expression Builder, there is a checkbox named **Include External Measures**. When this checkbox is selected, the Measure Selector will include those measures that were imported into the solution in addition to those that are present due to component selections in the Measure Selector.

The following properties may be overridden for an external measure:

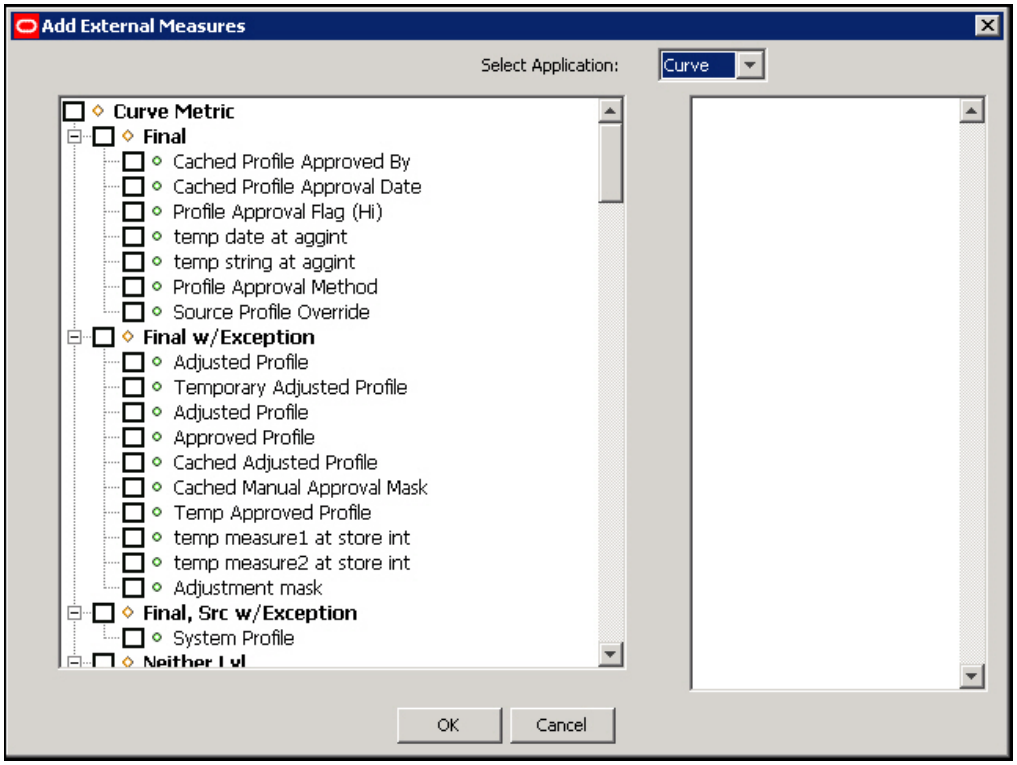
- Label
- Description
- Base State
- Agg State
- UI Type
- Range

Import a Measure

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

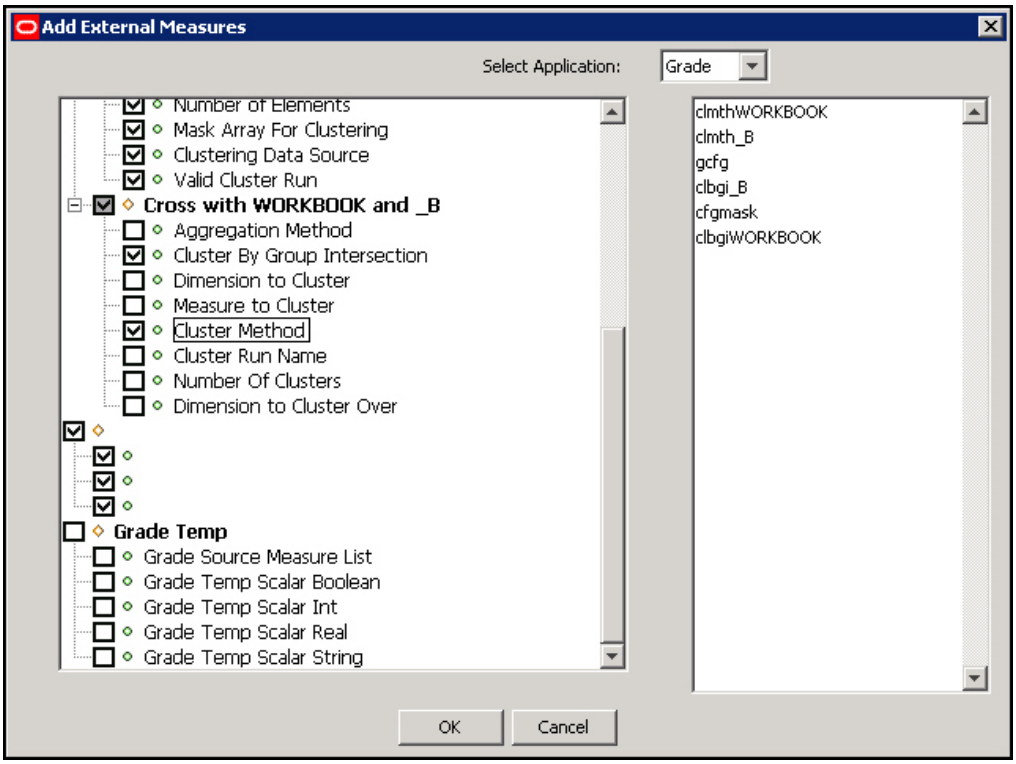
Navigate: In the Configuration Components pane, select **Project** – **Solution** – **Measures**. The Measure Manager window opens in the workspace.

1. Select the **External Measures** tab, and click the **Import Measure** button on the toolbar. The Add External Measures dialog box appears.



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

External measures that are already imported appear in **bold**.

4. From the right-hand list, select the measure to import and click **OK**. The selected measures appear in the External Measure tab.

Remove an Imported Measure from a Solution

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

1. Click the **External Measures** tab, and select the measure to remove.
2. Click the **Remove Imported Measure** button from the Measure Manager toolbar, or right-click and select **Remove Import**. The selected imported measure are removed from the External Measures tabs.

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.

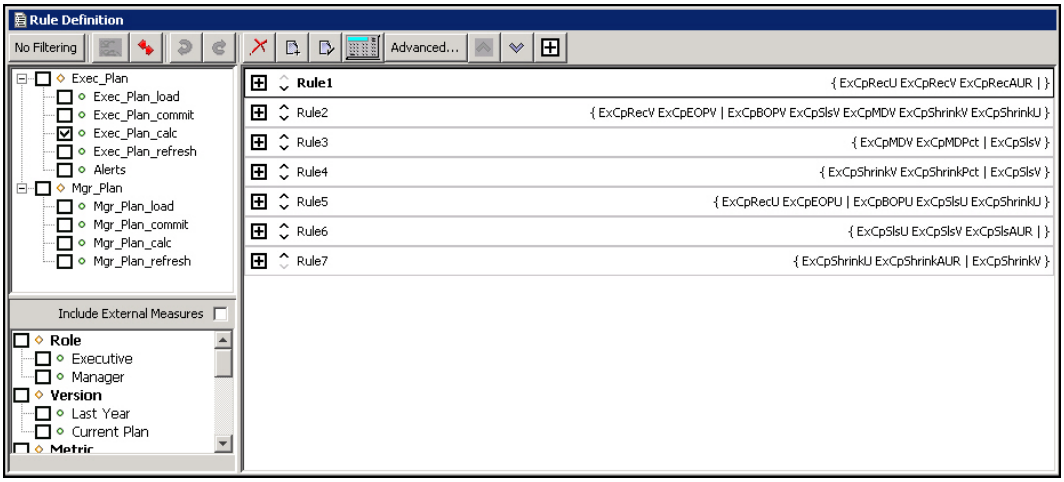
> **Note:** A rule set is a tools concept only. It does not appear in the configured solution.

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, the configuration administrator can create additional rule groups as necessary. The configuration administrator can also rename the rule groups that were automatically generated, but these rule groups cannot be deleted.

Create a Rule Set

Navigate: In the Configuration Components pane, select **Project – Solution – Rules**. The Rule Definition 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. Perform the following:
   a. 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.

b. In the **Description** field, enter a description of the rule set.

c. Click **OK** to save any changes and close the window. The rule set appears in the Rules navigation tree.

Delete a Rule Set

Navigate: In the Configuration Components pane, select **Project – Solution – Rules**. The Rule Definition window opens in the workspace.

1. In the Rule Definition window, select the rule set that to be deleted.
2. Choose one of the following methods:

   a. From the toolbar, click the **Delete** button, and select **Delete Rule Set**.

   b. Right-click in the Rule Definition window, select **Delete/Remove**, and select **Delete Rule Set** from the drop-down list.

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

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, multiple rule groups can be defined 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 for the 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. In Rule group validation, each rule in a rule group must represent a completely different measure relationship, which means that the following restrictions apply:

- 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 in the rule tool.
- For similar reasons, a rule normally does not use a collection of measures that is a subset 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 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.
- 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 multiple result functions or procedures, a measure may only be on the left-hand side of one expression in a rule.

> **Note**: When performing validation on rules in RPAS Configuration Tools, a proxy domain is used that only contains a small portion of information that is present in a real domain. This proxy domain does not contain any information about the positions that exist in various hierarchies and dimensions. Therefore, when performing rules validation, any rule that contains a reference to a position will not be properly validated within Configuration Tools and will be marked as invalid, despite the fact that it will execute in the final domain.

Multiple Refresh Rule Groups

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

- If a workbook is built in week 5, the user would have actuals for weeks 1-4.
- In week 6, the user can 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 selection within the RPAS Client.

- A workbook contains all of 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 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

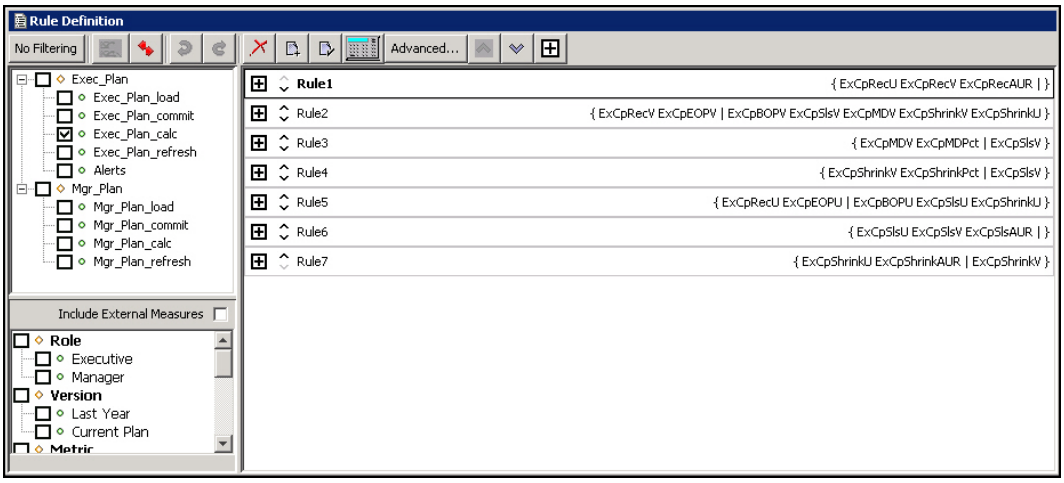
Although only a single rule group may be active at any time, RPAS supports the transition from one rule group to another, so the active rule group may be changed. The calculation engine ensures the integrity of measure relationships at all times, so this transition process is not merely a case of switching 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 must consider the impact of anticipated rule group transitions.

Automatic rule group transitions occur under the following circumstances:

- **On workbook building** – Data is loaded using the load rule group. This typically loads 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 automatically executes 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

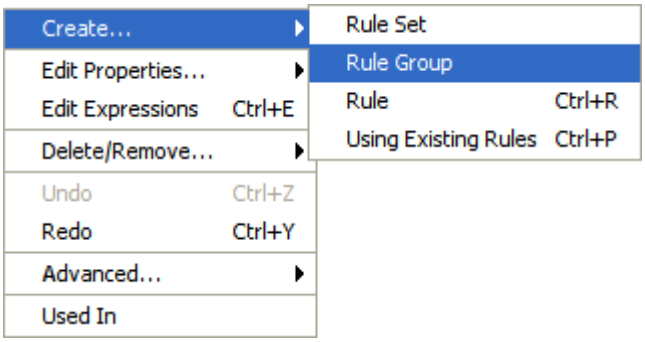
Navigate: In the Configuration Components pane, select **Configuration** – **Project** – **Solution** – **Rules**. The Rule Definition window opens in the workspace.



Example of Rule Definition Window

1. In the Rule Definition window, select the rule set to be used 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.



Example of Create - Rule Group Menu Option

The Add Rule Group window is displayed.

Add Rule Group Window

3. Perform the following:
   a. In the Name field, enter the name of the rule group.

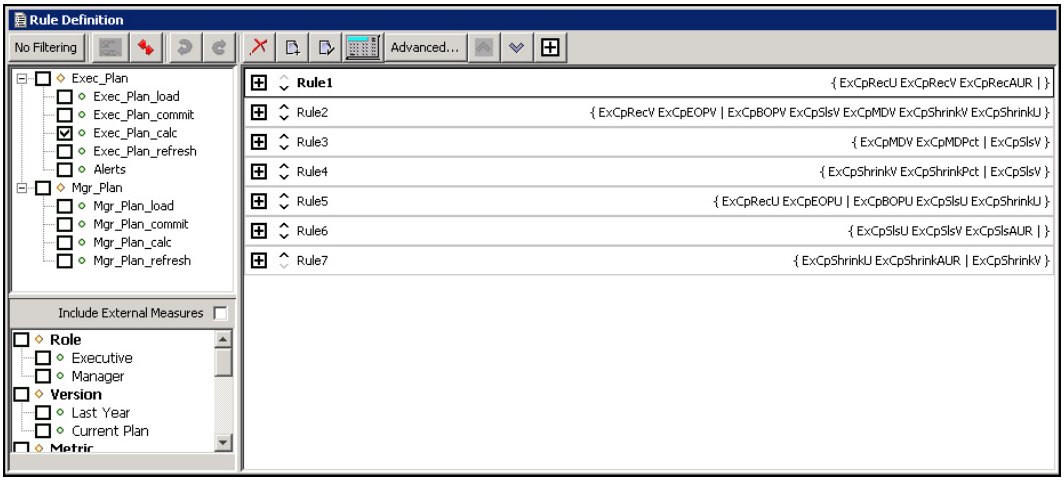
> **Note:** A rule group name can be a maximum of 16 alphanumeric or underscore characters. It must not have a name that is the same as any other rule group that exists in the project.

b. In the **Description** field, enter a description of the rule group.

c. Click **OK** to save any changes and close the window. The new rule group appears in the Rule Definition tree window.

Delete a Rule Group

Navigate: In the Configuration Components pane, select **Configuration** – **Project** – **Solution** – **Rules**. The Rule Definition window opens in the workspace.



Rule Definition

1. In the Rule Definition window, select the rule group to be deleted.

> **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 Delete is selected for one of the default rule groups, it will not be deleted, but all of the rules will be removed from the rule group. 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.

2. From the toolbar, click the **Delete** button, and select **Rule Group**. The group is removed from the Rule navigation tree.

Copy a Rule Group

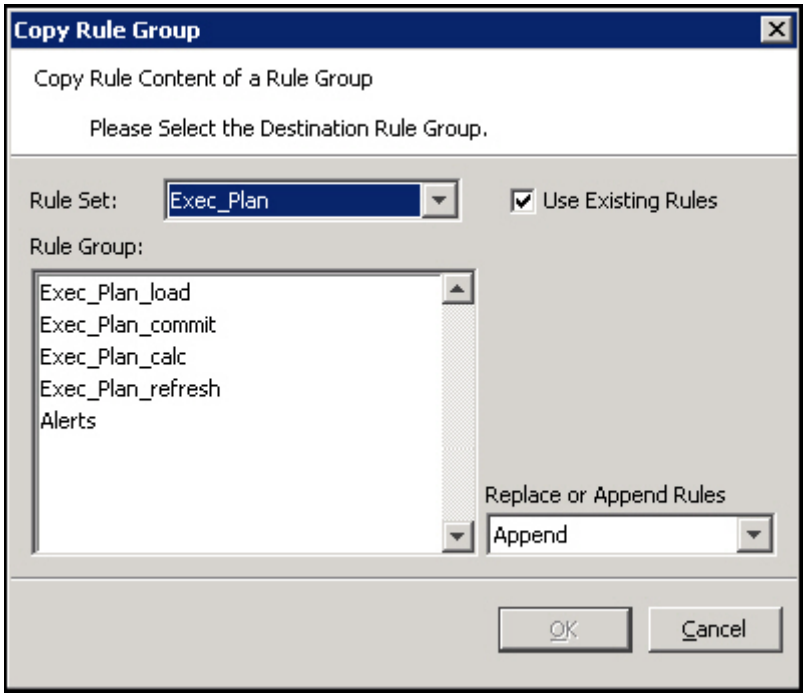
If two rule groups are similar, it may be beneficial to copy one rule group into the other to prevent having to create a rule group from scratch. When copying the rules of a rule group into another rule group, it is possible to specify whether existing rules will be used or copies of the rules will be created. The Use Existing Rules checkbox defaults to using any existing rules in the rule pool. If this checkbox is selected, 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 to be copied into must already exist. A new rule group cannot be created through this process.

Navigate: In the Configuration Components pane, select **Configuration** – **Project** – **Solution** – **Rules**. The Rule Definition window opens in the workspace.

1. In the Rule Definition window, select the rule group to be copied.
2. From the toolbar, click **Advanced**, and select **Copy Rule Group**. The Copy Rule Group window appears.



Example of Copy Rule Group Window

3. Perform the following:
   a. Using the **Rule Set** list, select the destination rule set.
   b. From the **Rule Group** area, select the desired destination rule group.
   c. Using the Replace or Append Rules list, 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.
   d. Click **OK**. The Confirm Operation dialog appears.
4. Click **OK**. The rules are copied to the selected rule group.

Measure Validation in the Rule Definition Window

If **Enable Measure Content Validation** is NOT selected in the Workbench Preferences window, measure content validation is turned off in the Measure Manager. Measure validation must be manually initiated in the rule tool. See "Measure validation within the Measure Manager."

From the Rule Definition window, click the **Perform Measure Content Validation** button on the Rule Definition toolbar.

Rule Definition

The Rule Definition tool allows the configuration administrator to define, organize, and manage rule sets, rule groups, and rules. It also allows for the creation of expressions and addition of 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.

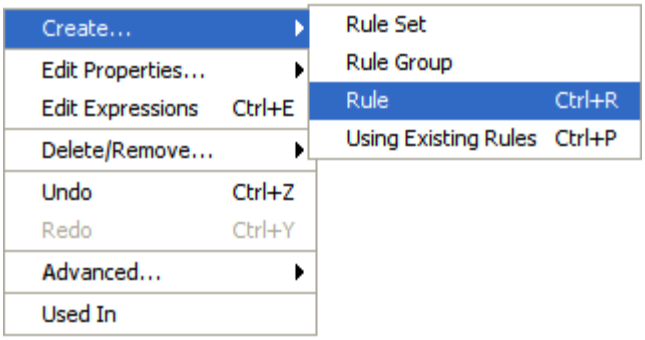
Those who are configuring the calculation requirements of a solution are expected to fully understand the operation of the RPAS calculation engine.

Create a Rule and Add It to a Rule Group

Navigate: In the Configuration Components pane, select **Project** – **Solution** – **Rules**. The Rule Definition window opens in the workspace.

1. In the Rule Definition window, select the rule group in which 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 before which you want the new rule to be placed. The new rule will be created above the selected rule.
2. Choose one of the following methods:

   - From the toolbar, click the **New** button and select **Rule**.
   - Select **Create – Rule** from the right-click menu.



Example of Create - Rule Menu Option

   - Press CTRL+R.

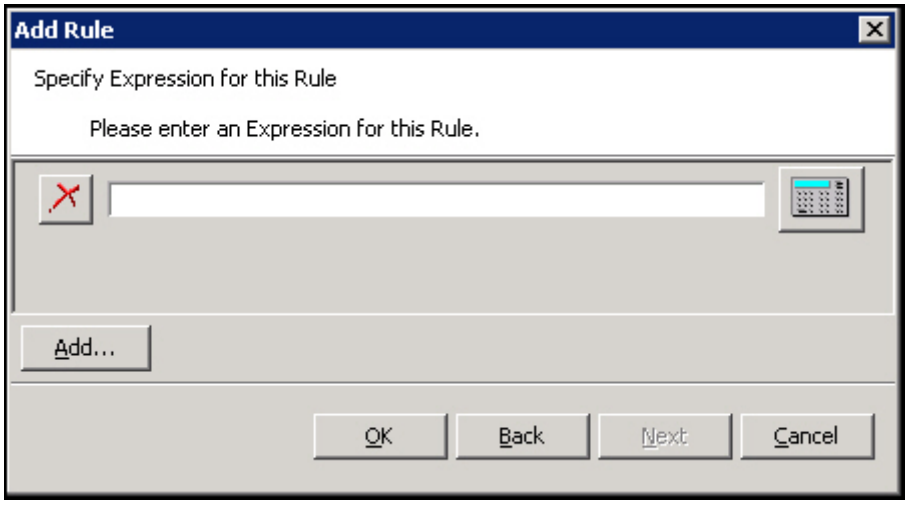
   The Add Rule window opens.

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.

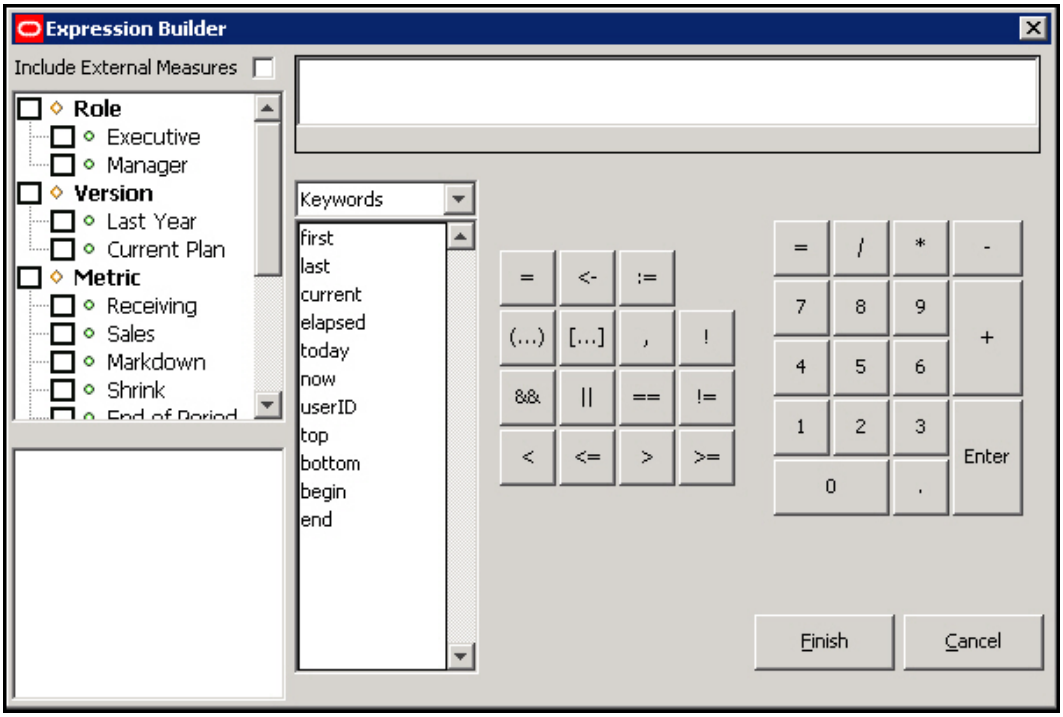
4. In the **Description** field, enter a description of the rule.
5. Click **Next**. The Expression Builder window of the Add Rule window is displayed.



Specify Expression

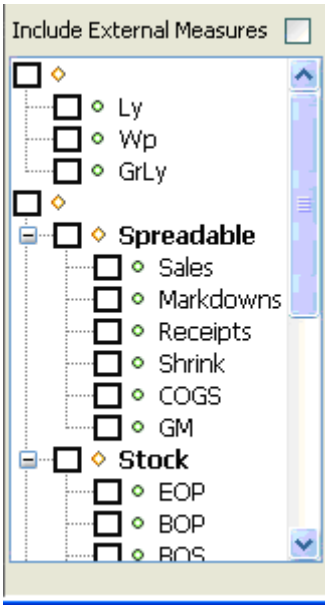
6. Type the expression in the input box or perform the following to use the Expression Builder.

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

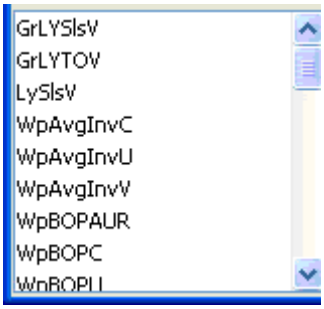


Expression Builder Window

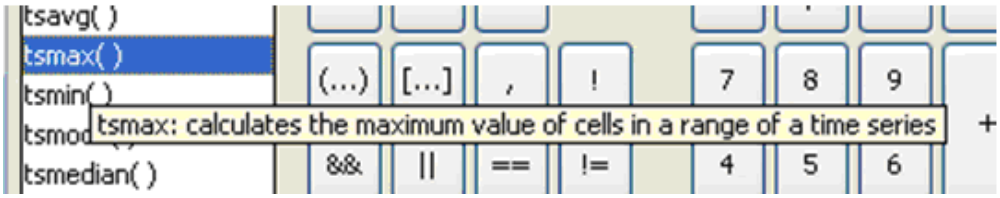
b. In the upper left-hand pane, select the measure components to filter measures to be visible for pasting into the expression. If External Measures are required in the expression, select the **Include External Measures** check box. Realized measures meeting the filtering conditions are displayed in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



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

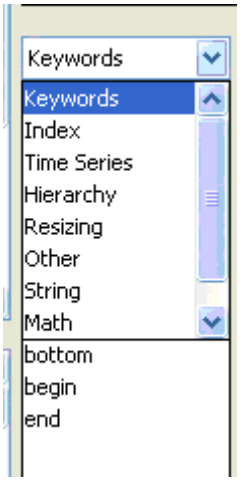


Note: Place the mouse over the name of a function, procedure, keyword, or modifier to see a tooltip that explains its function.



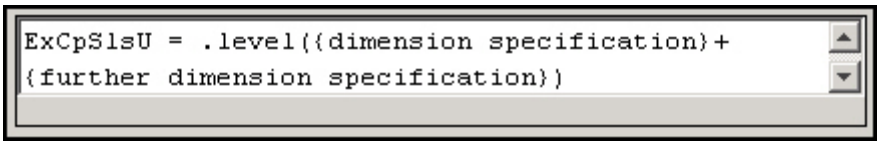
Tooltip Example

d. From the drop-down list, select a category of functions, procedures, keywords, or modifiers. Double-click on a specific function, procedure, keyword, or modifier to be used in the expression, and (if appropriate) outline syntax pasted at the insertion point in the expression.



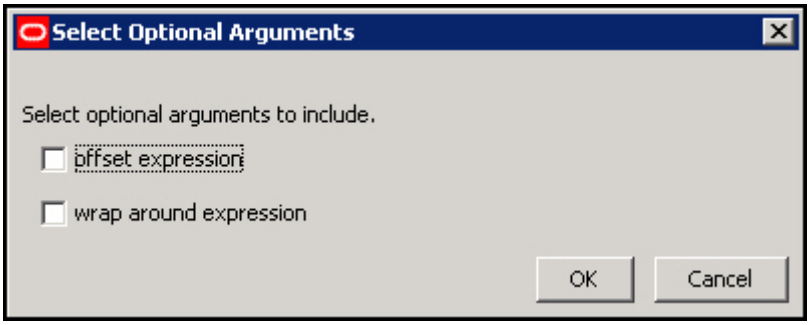
Category Options List

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 to use in the expression. Note that all arguments are positional, so if a later argument is selected, all earlier arguments will be automatically selected. If an earlier argument is deselected, all later arguments will be automatically deselected.



Select Optional Arguments Dialog Box

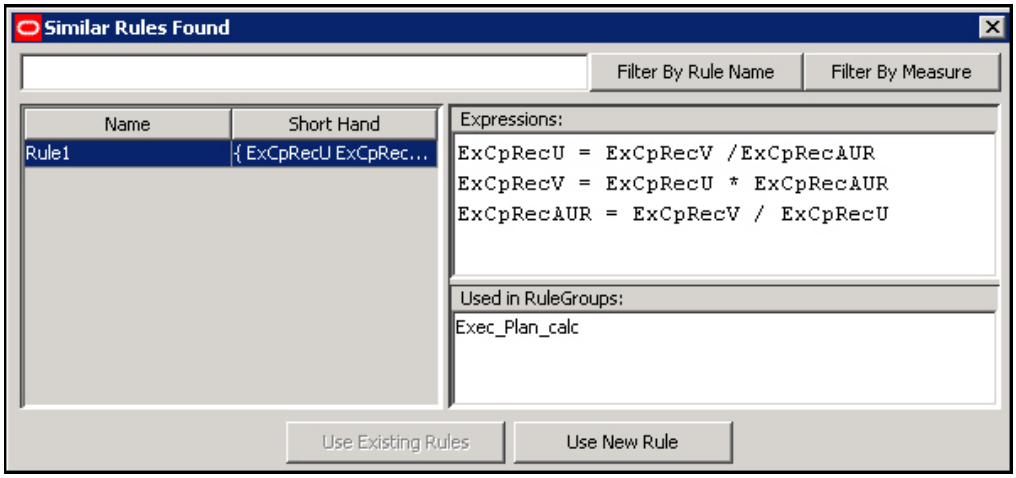
If the function, procedure, keyword, or modifier that is pasted has repeating arguments (for example the min function finds the minimum of a variable number of expressions), a dialog box appears to select how many of the repeating arguments to use in the expression.

e. Use the keyboard or dialog buttons to add the appropriate mathematical operators and constants to construct the expression.

f. When you have defined the expression as needed, click **Finish** and close the window.

> **Note:** If an invalid expression is created, 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 provides you with 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.



Similar Rules Found 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.

Note: If **Use Existing Rule** or **Use New Rule** is not selected and the window is closed manually, a new rule is created.

Add an Existing Rule to a Rule Group

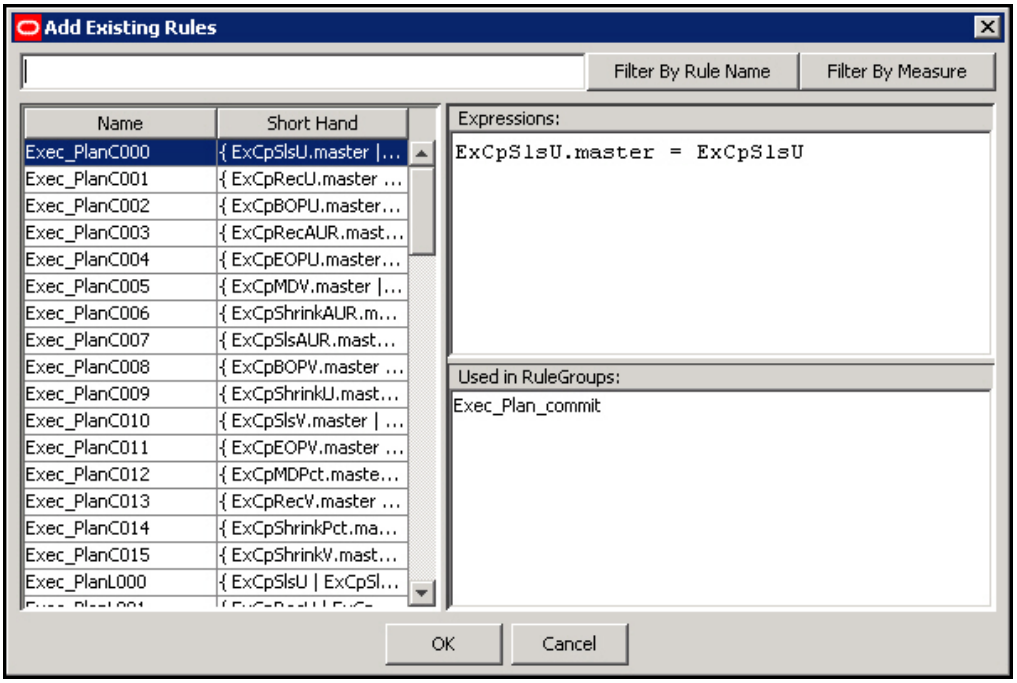
Using the Add Existing Rules dialog, you may select multiple rules to add to a rule group. If at least one of the selected rules is already in the rule group, the **OK** button will gray out disallowing the operation until that rule is deselected. When multiple rules are selected, the expression and 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 Components pane, select **Project** – **Solution** – **Rules**. The Rule Definition window opens in the workspace.

1. In the Rule Definition window, select the rule group to be used to add an existing rule. If the rule is to be placed in a particular position in the sequence of rules in the rule group, select the rule in which the new rule is to precede.
2. Choose one of the following methods:

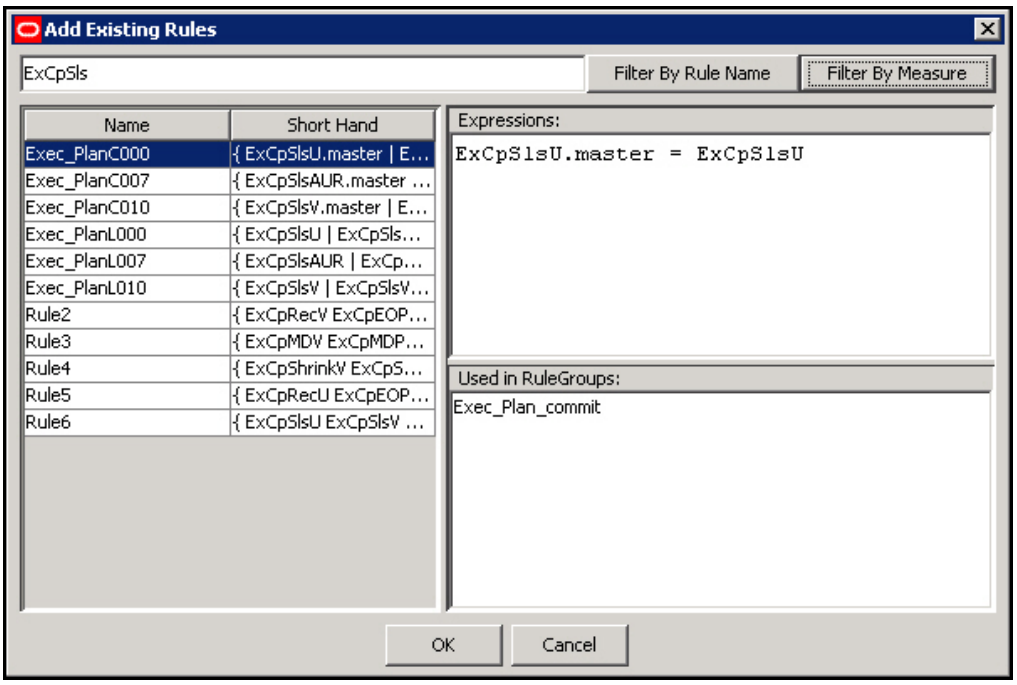
   - From the toolbar, click the **New** button and select **Using Existing Rule**.
   - Select **Create** – **Using Existing Rule** from the right-click menu.
   - Press Ctrl+P.

   The Add Existing Rule dialog appears.



Add Existing Rule Dialog Box

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, type a filter string (for instance, ExCpSls in the input box directly under the Title Bar. Click **Filter by Measure**, and only those measures with ExCpSls will be displayed.



Example of Filter on ExCpSls

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

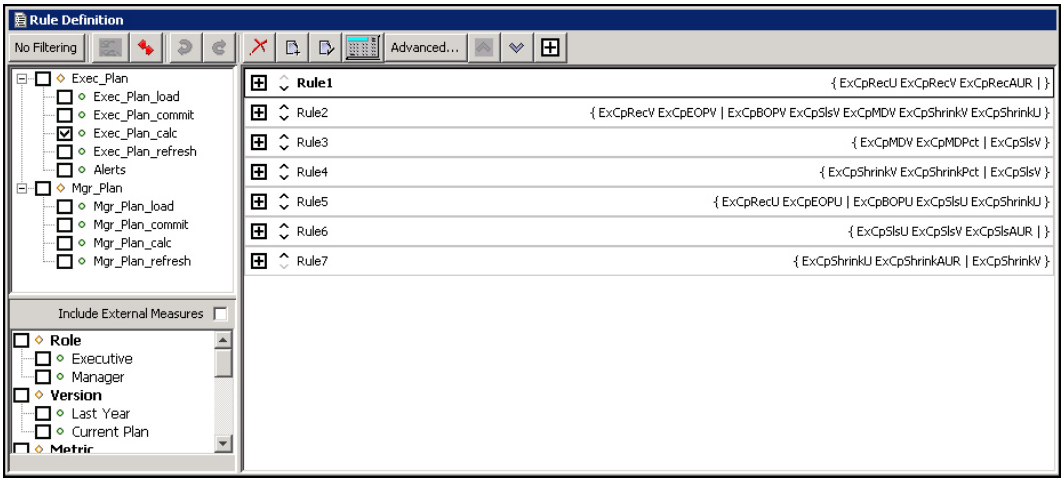
The rule is placed above the rule that was selected at the start of the process in the sequence of rules in the rule group or at the end of the rule group if no rule was selected.

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.

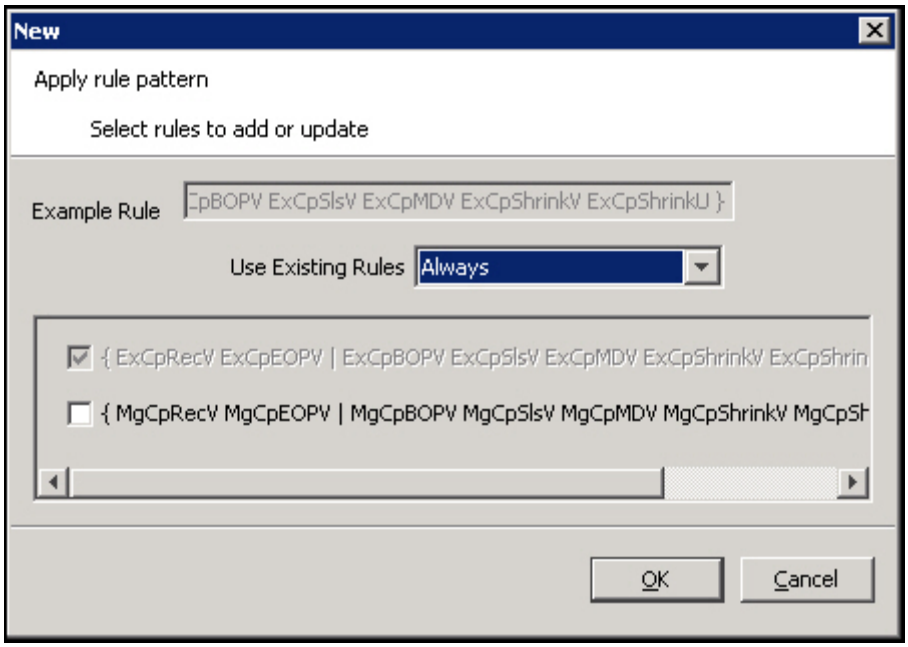
Navigate: In the Configuration Components pane, select **Project** – **Solution** – **Rules**. The Rule Definition window opens in the workspace.



Example of 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** and select **Apply Pattern,** or right-click and select **Advanced - Apply Pattern**.

   A New dialog box containing a 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.

4. Select the rule replacement policy in the Use Existing Rules dropdown menu. This allows the user to select, on a rule-by-rule basis, which rules to reuse and which to re-create. By default, this is set to "Always".

- When "Always" is selected, rules that already exist elsewhere in the configuration will always be used in place of newly generated rules.
- When "Never" is selected, new rules will always be created, even if an identical rule already exists elsewhere in the configuration.
- When "Prompt" is selected, the Rule Locator dialog will be presented for each rule whose expressions already exist elsewhere in the configuration.

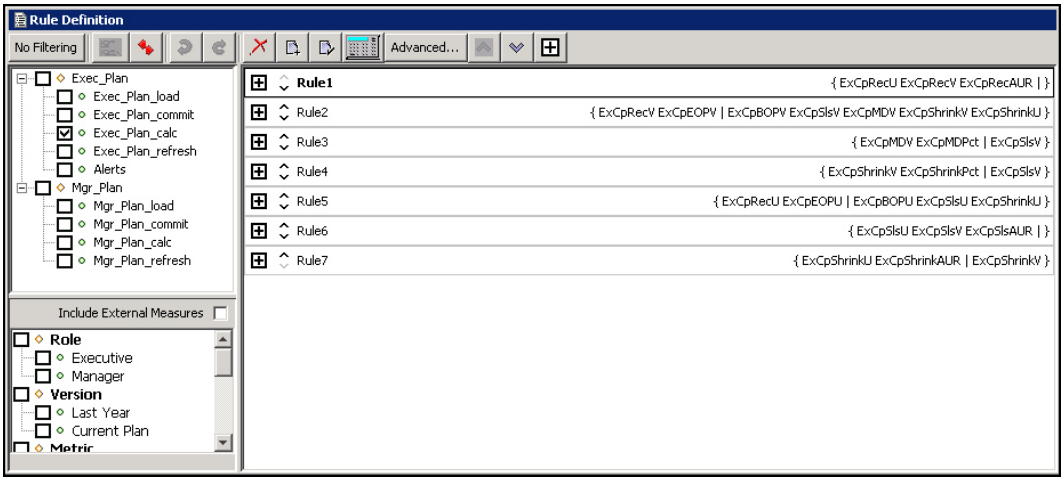
5. From the list of possible new and updated rules, select those to be updated or added to the rule group. In either case, if rule reuse is specified in Step 4, a new rule will not be created, but the rule existing elsewhere in the configuration will be used in its place.
6. Click **OK**.

   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 template rule'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 replaced. This means that the old rule will be removed and replaced with a new rule whose expressions follow the base rule's expression pattern. In both cases, the rule will follow the pattern of the base rule's expression.

Delete a Rule from All Rule Groups

> **Note:** This procedure will delete the rule from all rule groups that contain this rule as well as from the Rule Pool. If the desired action is 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 Components pane, select **Project** – **Solution** – **Rules**. The Rule Definition window opens in the workspace.

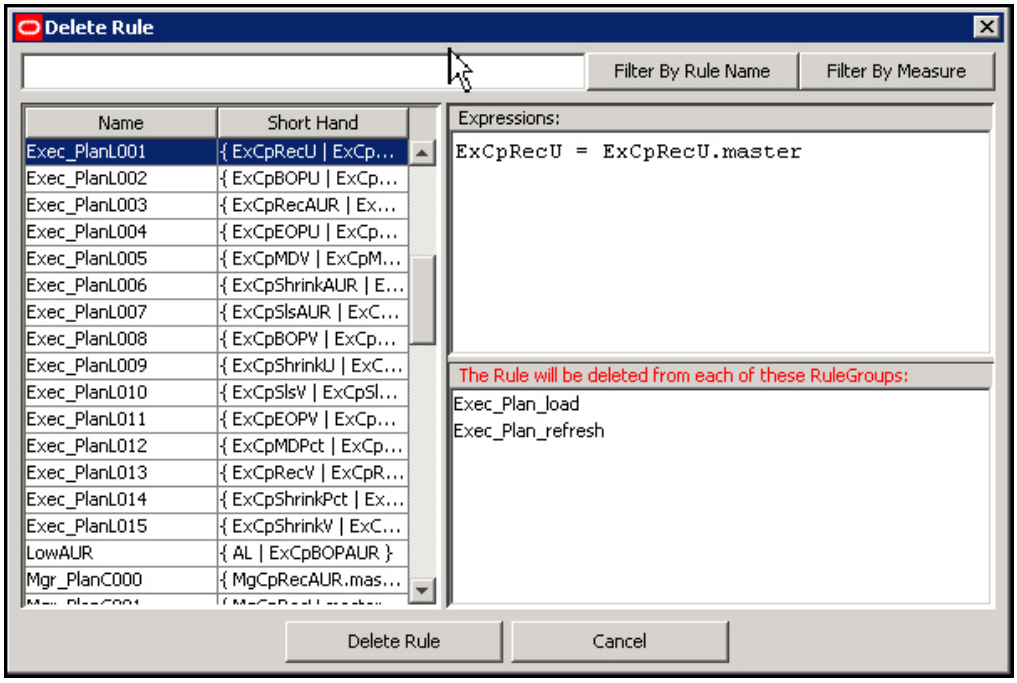


Example of Rule Definition Window

1. In the Rule Definition window, select any rule group that contains the rule to be deleted.
2. Select the rule to be deleted.

Note: If multiple rules are selected, only the last selected rule will be 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 Group window appears.
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.



Example of Delete Rule Window

5. Click **Delete Rule** to delete the rule and close the window.

Remove a Rule from a Rule Group

Note: This procedure will remove the rule(s) only from the currently selected rule group. If the desired action is to delete the rule(s) from all rule groups and the rule pool, see "Delete a Rule from All Rule Groups" in this document.

Navigate: In the Configuration Components pane, select **Project** – **Solution** – **Rules**. The Rule Definition window opens in the workspace.

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

Note: Multiple rules can be selected by holding the Control (Ctrl) key as the individual rules are selected, or by clicking one rule and holding Shift key as another rule is selected, which selects all rules between the two that were clicked. A selected rule is indicated by a bold rule name.

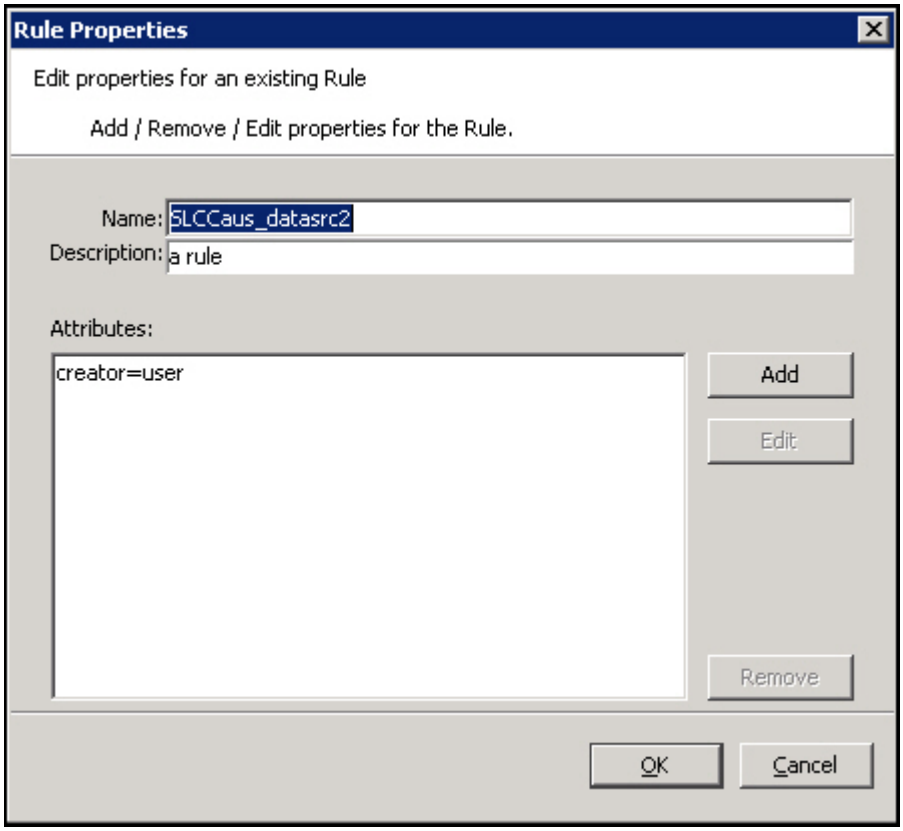
3. From the toolbar, click the **Delete** button and select **Remove Rule(s)**, or select **Delete – Remove**… – **Remove Rule(s)** from the right-click menu.

   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 Components pane, select **Project** – **Solution** – **Rules**. The Rule Definition window opens in the workspace.

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

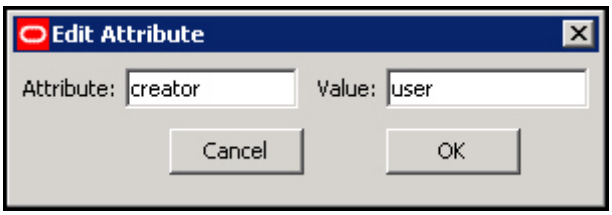


Rule Properties Dialog Box

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 to edit.
   b. Click **Edit**. The Edit Attribute dialog box opens.



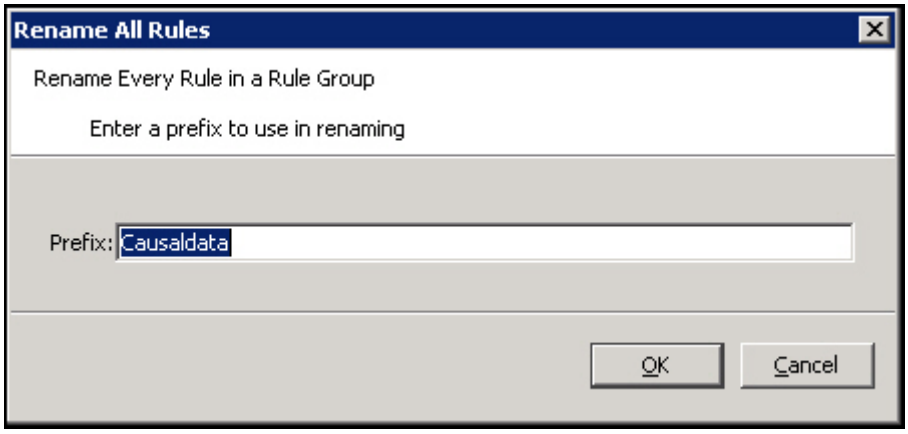
Edit Attribute Dialog Box

   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 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

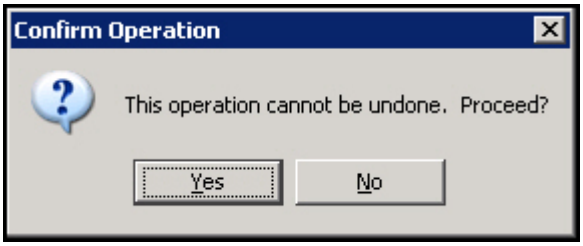
Navigate: In the Configuration Components pane, select **Project** – **Solution** – **Rules**. The Rule Definitions window opens in the workspace.

1. In the Rule Definition window, select the rule group that contains the rules 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.



Example of Rename All Rules Dialog Box

3. In the **Prefix** field, enter a prefix up to ten characters in length, which will be the start of all rule names.
4. Click **OK**. The Confirm Operation dialog box appears.



Confirm Operations

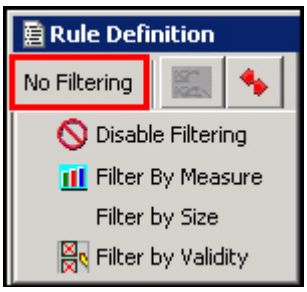
5. Click **Yes**.

> **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 Components pane, 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 **Rule Filtering**.



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 whose measures conform to the selected component scheme.
   - **Filter by Size** – Rules are filtered to show those with more than one expression.
   - **Filter by Validity** – Only invalid rules are displayed.

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

Reordering Rules in a Rule Group

Navigate: In the Configuration Components pane, select **Project** – **Solution** – **Rules**. The Rule Definition window opens in the workspace.

1. In the Rule Definition window, select the rule group that contains the rules to reorder.
2. Select the rule to reorder.
3. Perform the following as needed:
   - Use the Up/Down arrows on the Rule Definition toolbar to move the rule up or down the list.
   - Click the Up/Down arrows to the left of the rule name to move the rule up or down the list.

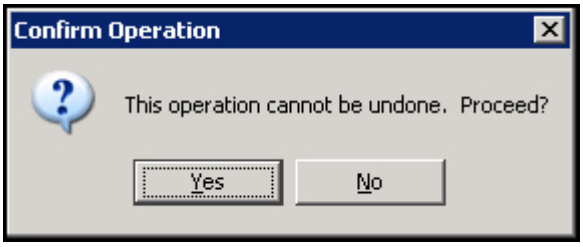
Auto Generate Load and Commit Rules

Navigate: In the Configuration Components pane, select **Project** – **Solution** – **Rules**. The Rule Definition 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 for which load or commit rules are to be auto generated, or select any rule group in that rule set.
2. Perform one of the following methods:
   - 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.

   The Confirm Operation dialog box appears to inform you that this process cannot be undone.



Confirm Operation

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** check box defaults to using any existing rules in the rule pool. If this checkbox is selected, the copy selected rules operation will use the same rules that the source rule group has. If this check box is not selected, 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 the user uses 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** check box is selected, the Similar Rules Found dialog appears. you have the option of using the existing rule or actually creating a new one.

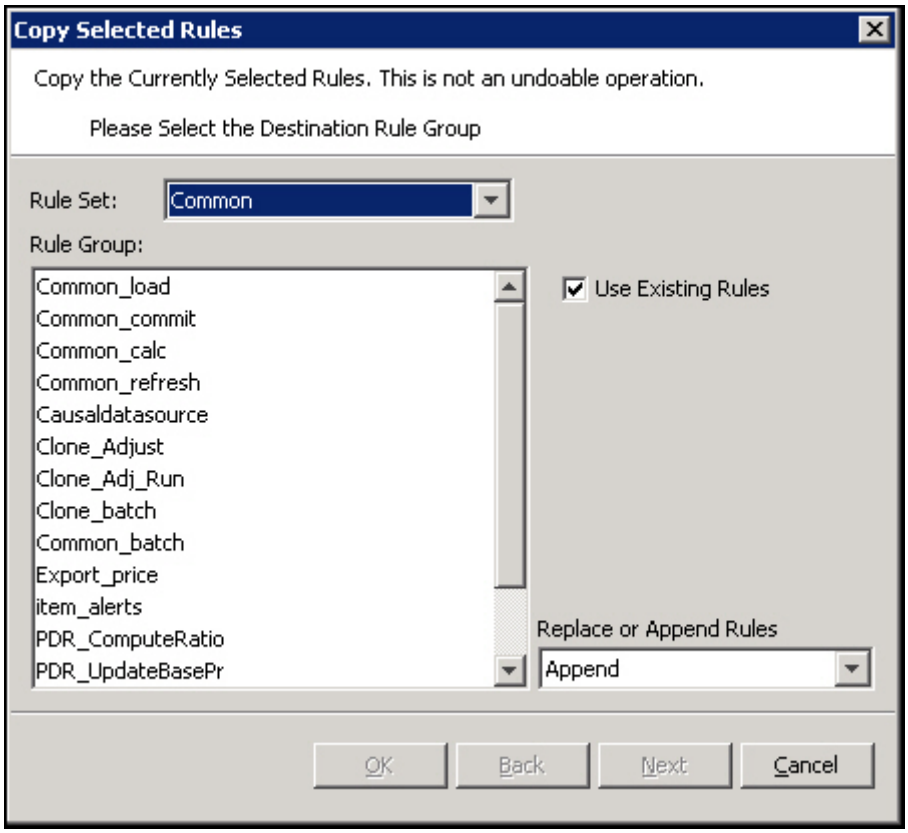
Navigate: In the Configuration Components pane, select **Project** – **Solution** – **Rules**. The Rule Definitions window opens in the workspace.

1. Select the Rule Group that contains the rules 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 key as selecting another rule, which selects all rules between the two that have been selected. A selected rule is indicated by a bold rule name.

2. Perform one of the following methods:
   - 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 dialog box opens.

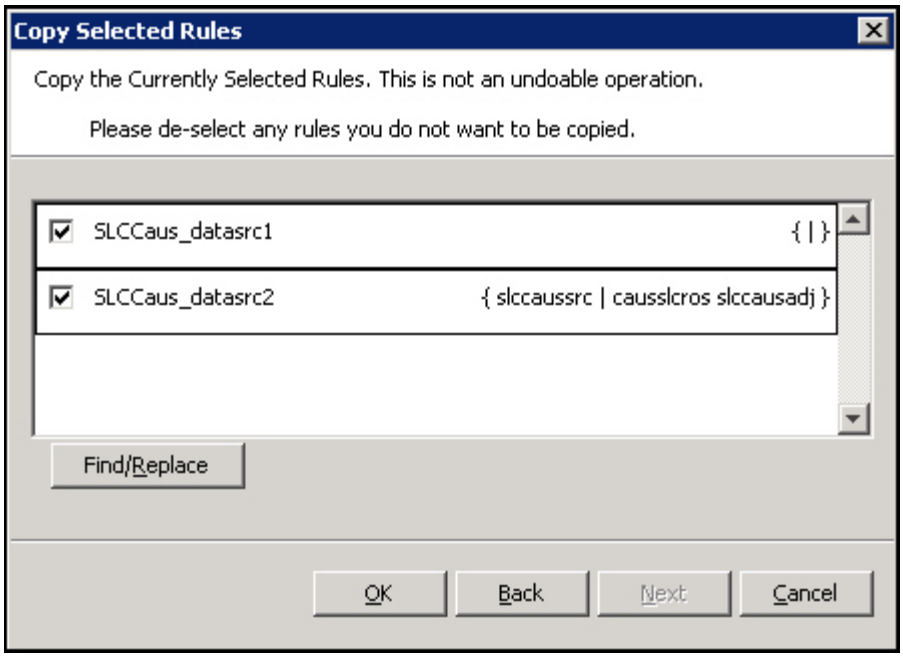


Copy Selected Rules Dialog Box

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. Select the **Use Existing Rules** checkbox if the rules should be added to the destination Rule Group. If the User Existing Rules checkbox **is not selected**, copies of the selected rules will be created for the destination Rule Group instead.

   Note: If the Find/Replace functionality is used, copies will be created for affected rules even if the Use Existing Rules checkbox is selected.

6. 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.
7. Click **Next**. The second window of the Copy Selected Rules window opens. This window allows you to select or deselect rules from those originally selected to copy when the check box beside the rule name is selected.

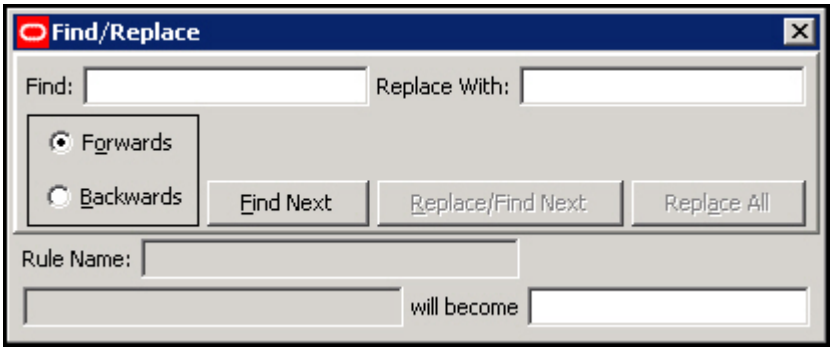


Copy Selected Rules – Rules to be copied appear with check marks

Find and Replace Measures in the Copied Rules

The ability to find/replace in the copy rules is a very powerful and useful feature. This feature can be used 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 dialog box appears.



Find/Replace Dialog Box

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

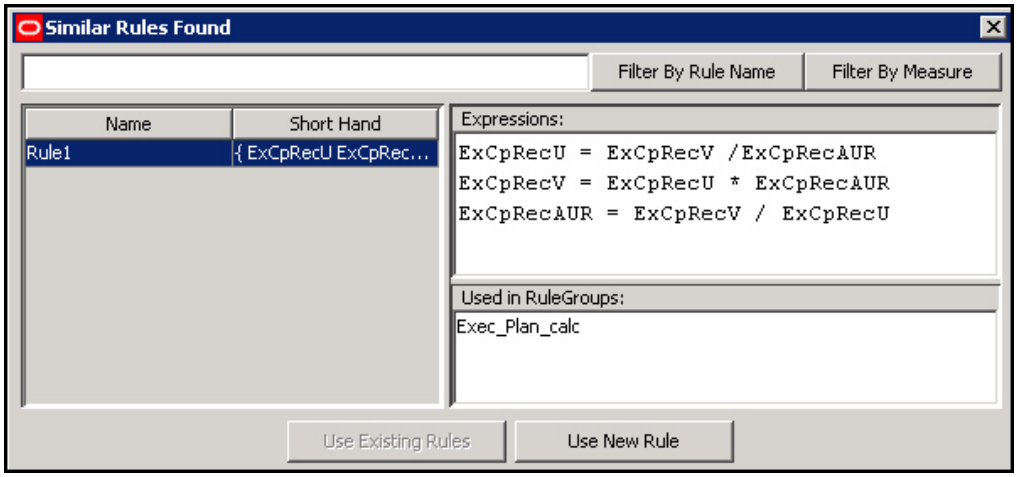
> **Note:** The Find function is case-sensitive.

3. In the Replace With field, enter the string to replace the portion of the measure name.
4. Perform one of the following:
   - 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 the following as needed:
   - 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.

   The Similar Rules Found dialog box appears.



Similar Rules Found Dialog Box

Using this dialog box, you can replace a portion of the rules (for instance a prefix) either for all instances of the rules or only for the new instances (where the old instances are not affected).

Example:

Suppose you create a series of rules for the Executive (Ex) Calc Rule Group, and wants to use these as a model for the Manager (Mg) Calc Rule Group. Each of these original rules contains expressions with the "Ex" prefix. Each such instance needs to be replaced with "Mg."

1. Click **Use Existing Rules** to replace every instance of "Ex."
2. Click **Use New Rule** to replace the new instances of "Ex" without affecting the previous rules that contain "Ex."
3. Click the **Close** button to close the Find/Replace dialog box.
4. Click **OK**.
5. Click **Yes** to confirm.

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

Overview

An expression describes and solves the relationship between measures in a way that causes a measure to be calculated through the expression. They form the basis for all calculations of the relationships between measures, and they are evaluated by the calculation engine during a calculation. 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 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:

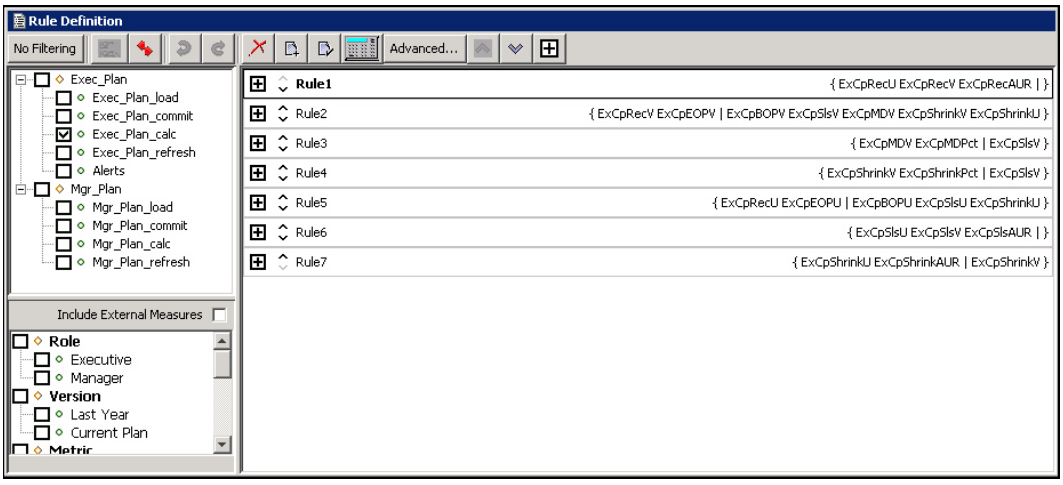
Expression 1: ReceiptUnits = ReceiptsValue / ReceiptsPrice

This expression specifies the way ReceiptUnits are calculated. ReceiptUnits are calculated by dividing ReceiptsValue by ReceiptsPrice.

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

Reorder an Expression in a Rule

Navigate: In the Configuration Components pane, select **Project** – **Solution** – **Rules**. The Rule Definition window opens in the workspace.



Example of Rule Definition Window

1. In the Rule Definition window, select any rule group that contains the rule whose expression is to be reordered.
2. Choose one of the following methods:
   - Expand the rule to view the expressions associated with the rule. Click the Toggle button on the Rules toolbar and select **Expand All Expressions**.

-

Toggle Button

- Click the **Toggle** button for the rule.

3. Use the up and down arrows to move the expression up or down the list.

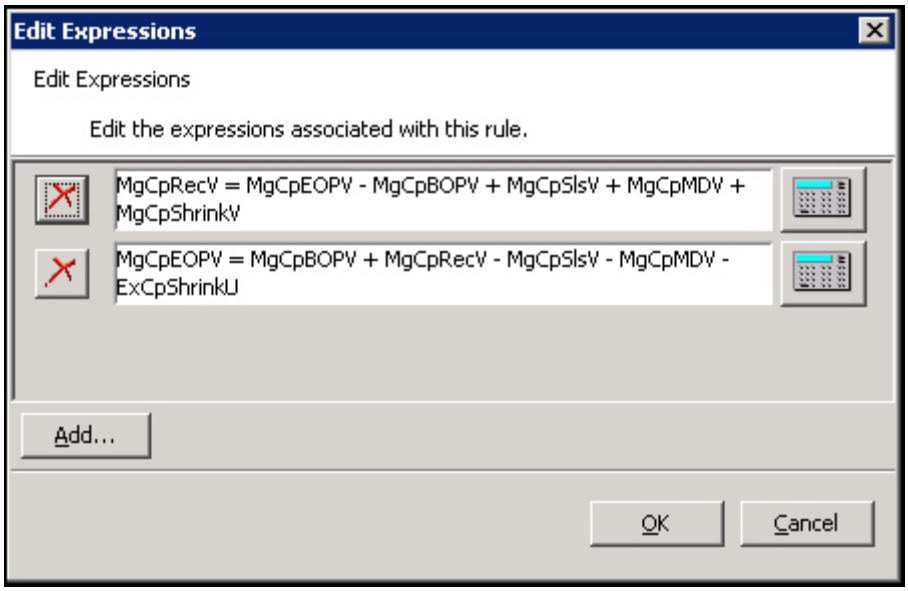
Edit an Expression in a Rule

Navigate: In the Configuration Components pane, select **Project** – **Solution** – **Rules**. The Rule Definition window opens in the workspace.

1. In the Rule Definition window, select any rule group that contains the rule whose expression is to be edited.
2. In the Rule Definition window, select the rule whose expression is to be edited.
3. Choose one of the following methods:

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

   The Edit Expressions window appears.

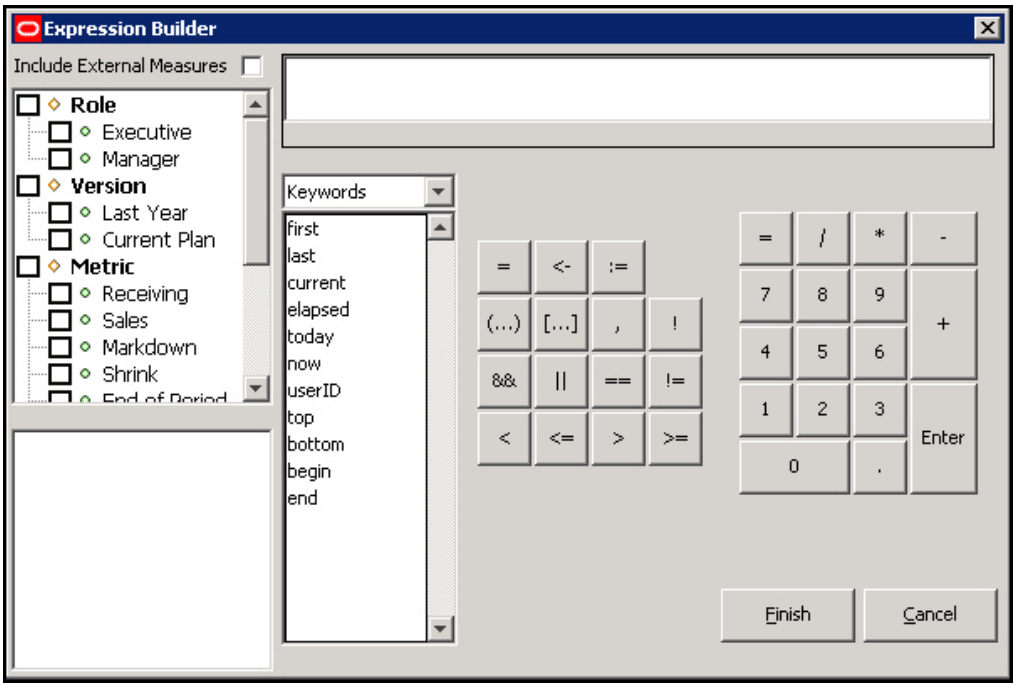


Edit Expressions Window

To Edit an Expression

1. Choose one of the following methods:
   - Edit the expression in its text box.

- Click the **Expression Builder** button for the expression to edit. The Edit Expressions window appears. Use the Expression Builder to make necessary changes and click **Finish** when complete.



Expression Builder Window

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

Delete an Expression from a Rule

Navigate: In the Configuration Components pane, select **Project** – **Solution** – **Rules**. The Rule Definition window opens in the workspace.

1. In the Rule Definition window, select any rule group that contains the rule whose expression is to be deleted.
2. In the Rule Definition window, select the rule whose expression is to be deleted.
3. Choose one of the following methods:

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

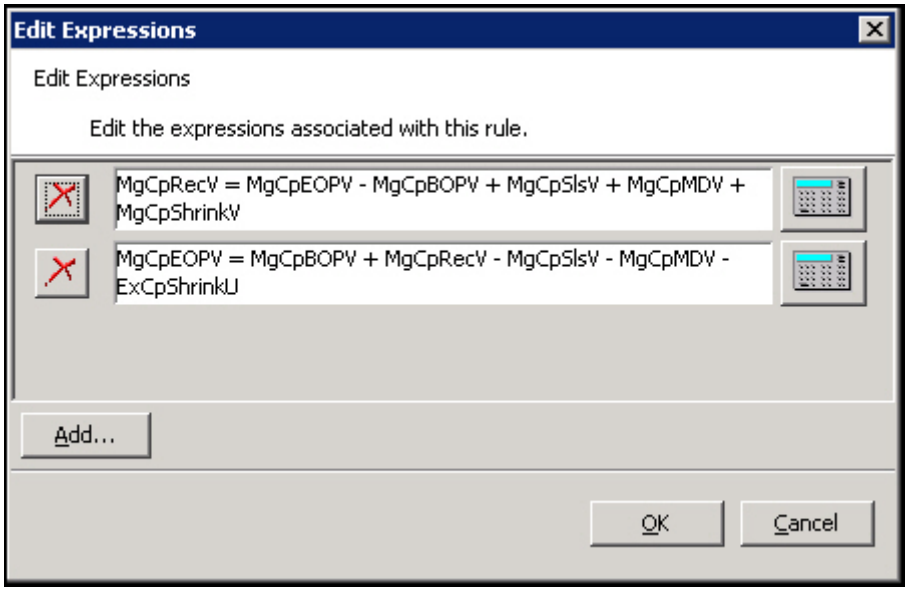
> **Note:** If the only expression in the rule is deleted, the rule will be flagged as being invalid, because it has no expressions.

Add an Expression to a Rule

Navigate: In the Configuration Components pane, select **Project** – **Solution** – **Rules**. The Rule Definition window opens in the workspace.

1. In the Rule Definition window, select the rule group that contains the rule that will have an expression added.
2. In the Rule Definition window, select the rule that will have an expression added.
3. Choose one of the following methods:
   - 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

4. Click **Add** to add a new expression in the Edit Expression box.
5. Choose one of the following methods:
   - Enter the expression in its text box
   - Click the **Expression Builder** button for the expression to edit. The Edit Expressions window appears. Use the Expression Builder to define the rule and click **Finish** when complete.
6. Click **OK** to add the expression.

RPAS Functions, Procedures, Keywords, and Modifiers

Overview

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 an 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 these components work together to allow you to view and analyze business functions.

The Workbook Designer allows for the creation selection, and integration of 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 must be laid out in a logical format and must be easy to navigate. When configuring a workbook, think about how the workbook will be used by the users in the RPAS Client. Understand the business process flow and what end users will need to access most. Most likely, this information must be contained in the first workbook tab and worksheet.

The names of all of the workbook components must 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.

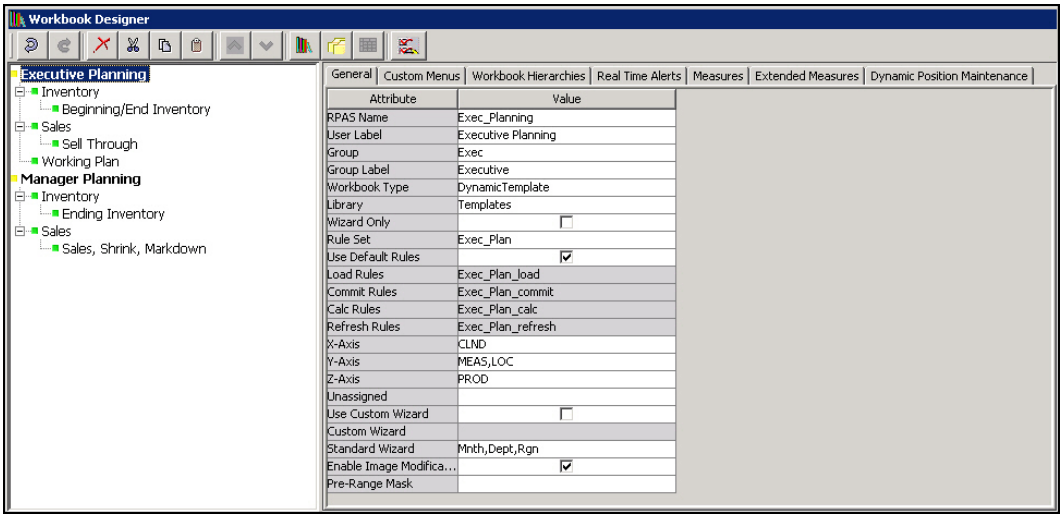
Workbook Tabs

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.

The Workbook Designer provides a visual represent of your workbooks, workbook tabs, and worksheets.



Example of Workbook Designer Window

The Workbook Designer contains the following areas:

Workbook Tree - The workbook tree provides a visual representation of the workbooks, workbook tabs, and worksheets. In the example provided, Executive Planning and Manager Planning are workbooks. Inventory and Sales are workbook tabs. Beginning/Ending Inventory and Sell Through are worksheets, which are contained in the Inventory and Sales workbook tabs.

Workbook Toolbar - This toolbar is used to perform common tasks. The buttons available depend on the item selected in the Workbook Designer window.

Workbook Tabs - The workbook tabs are used to define property at the workbook level. The tabs displayed depend on whether a workbook, workbook tab, or worksheet is selected from the Workbook tree. In the example above, Executive Planning (a workbook) is selected. The seven workbook tabs displayed are available to define specific properties for your workbook. For information on these tabs, refer to the Workbook Tabs section.

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 the end user 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.

Overview of Participation Measures

A "extended 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:

Sample Product Hierarchy Structure

Note the following important points when using this feature:

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

Create a Workbook

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

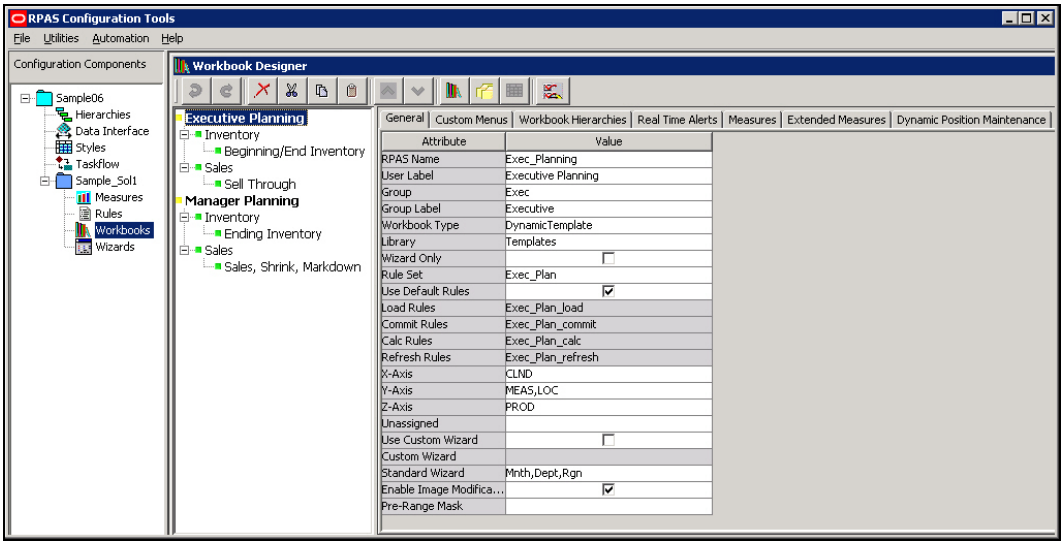
1. Choose one of the following methods:
   - Click the **New Workbook** button from the Workbook Designer toolbar.
   - Right-click in the Workbook tree area, and select **New Workbook**.
   - Select an existing workbook and press **Insert**.

   A new workbook is created.
2. Enter information for the tabs displayed across the top of the Workbook Designer window as necessary. Refer to "Defining Workbook Properties" for more information.

> **Note:** Double-click in the fields in the **Value** column to enter the information.

Edit Workbook Properties

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



Workbook Designer Window

1. Select the workbook, and click on the tab to edit. For information on these tabs, see the Defining Workbook Properties section.
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**.

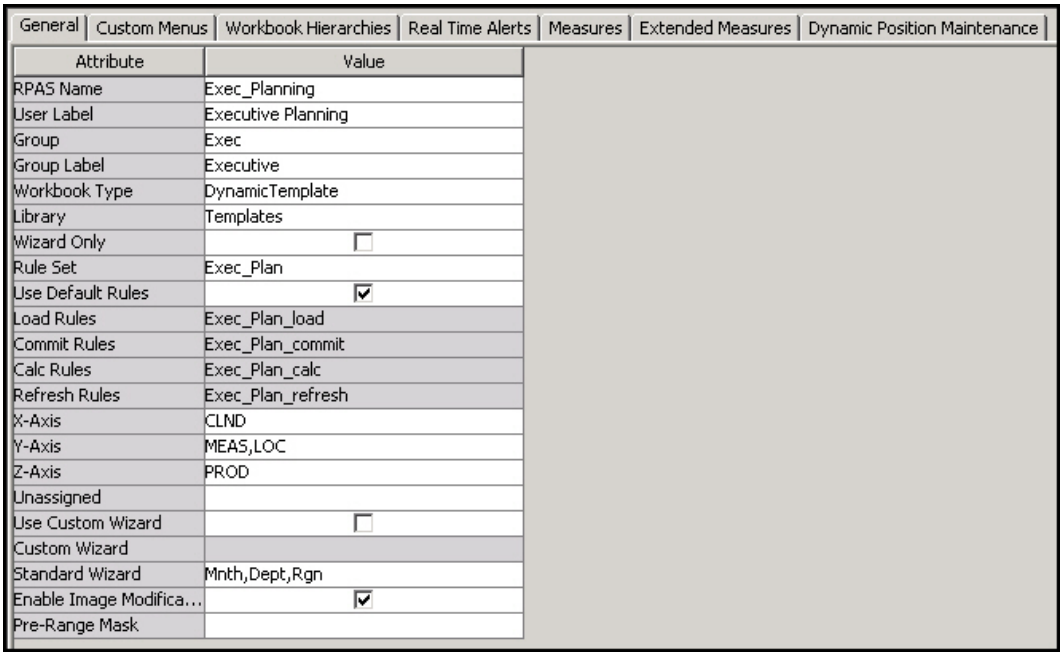
Defining Workbook Properties

When a workbook is selected from the Workbook Designer window, the following tabs appear in the workspace:

- General Tab
- Custom Menus Tab
- Workbook Hierarchies Tab
- Real Time Alerts Tab
- Workbook Transitions Tab
- Measures Tab
- Extended Measures Tab
- Dynamic Position Maintenance Tab

Refer to the topics below of information on using these tabs to define the workbook properties.

General Tab



Attribute	Value
RPAS Name	Exec_Planning
User Label	Executive Planning
Group	Exec
Group Label	Executive
Workbook Type	DynamicTemplate
Library	Templates
Wizard Only	☐
Rule Set	Exec_Plan
Use Default Rules	☑
Load Rules	Exec_Plan_load
Commit Rules	Exec_Plan_commit
Calc Rules	Exec_Plan_calc
Refresh Rules	Exec_Plan_refresh
X-Axis	CLND
Y-Axis	MEAS,LOC
Z-Axis	PROD
Unassigned	
Use Custom Wizard	☐
Custom Wizard	
Standard Wizard	Mnth,Dept,Rgn
Enable Image Modifica...	☑
Pre-Range Mask	

Example of General Tab

The following sections describe the default fields appearing on the General tab.

General Tab Default Fields

RPAS Name

The RPAS internal name of the workbook.

User Label

The label that the end user will see when selecting which workbook to build.

Group

In the RPAS Client, workbooks are grouped together under tabs (workbook template groups) to make it easier for the end user to find and select the needed workbook when solutions have multiple workbooks. This is the internal RPAS name of the group that this workbook will belong.

Group Label

In the RPAS Client, workbooks are grouped together under tabs (workbook template groups) to make it easier for the end user to find and select the needed workbook when solutions have multiple workbooks. This is the label the end user will see of the group to which this workbook will belong. If different labels are entered for the same workbook group against different workbooks, the workbook group label shown to the end user 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. When there is no custom extension, this field is set to the value **DynamicTemplate** by default.

Library

When the Workbook Type is not **DynamicTemplate**, it needs to be associated with a relevant custom shared library. This field holds the name of that library. When there are no custom extensions, this field is set to **Template** by default. 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 option is only used under circumstances when custom code is to be executed in a batch job at the end of the wizard process (which typically uses custom wizards) instead of building and opening a standard workbook. The selections made in the wizards are passed to the custom code. Therefore, a workbook with the Wizard Only option selected is not a workbook. However, the workbook infrastructure is used so that the process can have a name and label, and be assigned to a workbook group. This allows end users to select a Wizard Only template using the same process as workbooks. Workbooks that have the **Wizard Only** option selected do not need tabs or worksheets defined, but they do need a name, label, and workbook group.

Rule Set

Select the rule set 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

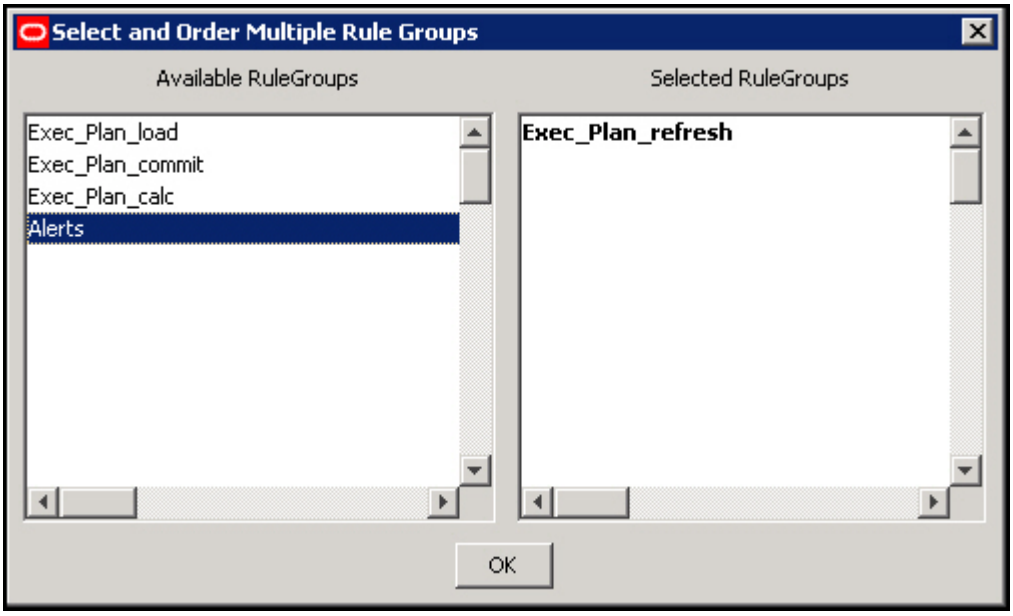
Select this option to use the default rules (Load, Commit, Calc, and Refresh) associated with the rule set. If this option is selected, the Load Rules, Commit Rules, Calc Rules, and Refresh Rules properties are disabled. If the option is not selected, the Load Rules, Commit Rules, Calc Rules, and Refresh Rules properties are enabled.

Load Rules, Commit Rules, and Calc Rules

Select the rule group to apply for each rule group type. Only rule groups from the selected rule set are offered.

Refresh Rules

This is only enabled when the **Use Default Rules** option is not selected. Select the rule group(s) to use as refresh rule groups. When enabled, click in the **Refresh Rules** field. The Select and Order Multiple Rule Groups dialog box opens.



Select and Order Multiple Rule Groups Dialog Box

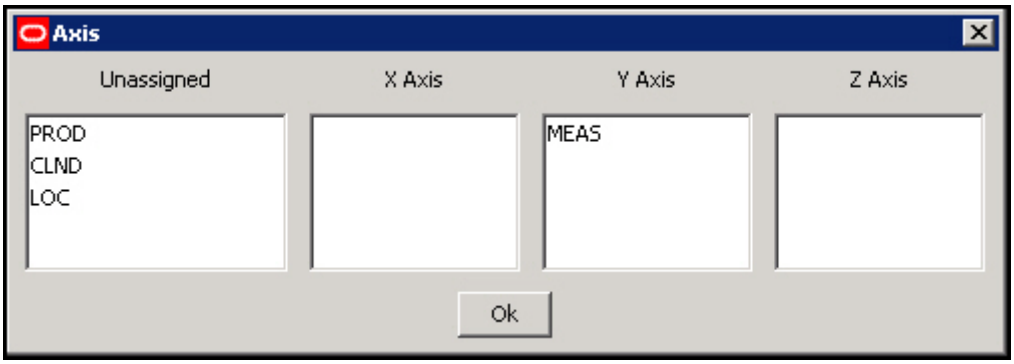
Within this window, specify which groups will be available to be used to refresh the workbook. Partial data in a workbook can be refreshed by refreshing with a rule group that only updates some of the measures in the workbook. The order that the rule groups appear is the order in which they are displayed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

1. In the Available Rule Groups column, select the rule group to add, and drag it to the **Selected Rule Groups** column.
2. 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). Before the hierarchies will appear in the Axis dialog, you must make sure that the database and base intersection have been assigned from the Measure Manager. The measures must also be made viewable. To do this, right-click in **Default** and select **Add Matching**.

1. Click in the **X-Axis**, **Y-Axis**, **Z-Axis**, or **Unassigned** field. The Axis dialog box opens.



Axis Dialog Box

2. Drag the hierarchies to the appropriate axis column.

3. 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. De-select the check box to disable the Custom Wizard property and enable the Standard Wizard Property. See "Wizards" in this document for more information on Custom Wizards.

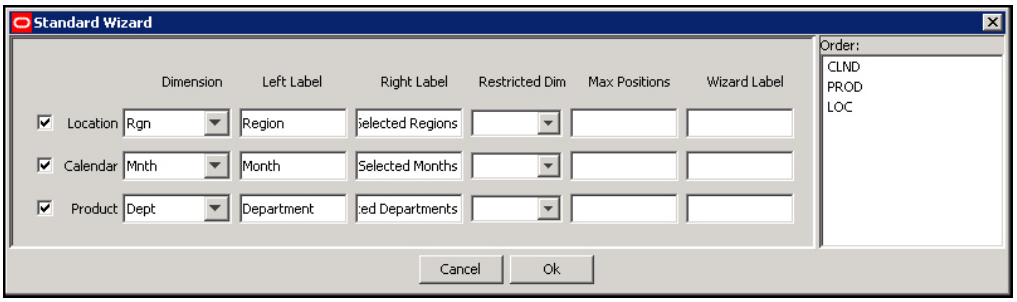
Custom Wizard

Select the custom wizard that to use to build the workbook. This field is only enabled when the **Use Custom Wizard** check box is selected. 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.

1. Click in the **Standard Wizard** field. The Standard Wizard dialog box opens.



Standard Wizard Dialog Box

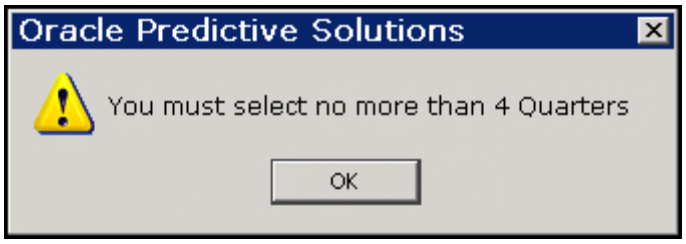
> **Note:** Hierarchies used in the lowest base intersection for the measures used in the rule set assigned to the workbook will be displayed. For each hierarchy, there will be a choice of dimensions. The dimensions offered will be the lowest dimension in that hierarchy that is used in the base intersection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

2. Select the check boxes next to the hierarchy names to enable the hierarchies for which that the end user in the RPAS Client should select positions.
3. **Dimension**: 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 of the positions in that lowest dimension that are descended from the positions selected from that higher dimension.

4. **Left Label** and **Right Label**: (Optional) 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.
5. **Restricted Dim** and **Max Positions**: For applications such as pricing, a retailer might want to restrict the selection of SKUs in a planning workbook to only one category. A planner may be allowed to plan several categories within a department, but only one category per plan. Per the retailers business process, a category would establish a coherent set of SKUs, the cross-item effects of which could be considered meaningful for a price optimization algorithm. Mixing in SKUs from two or more categories could be considered as polluting the cross-item effects, and therefore an undesirable situation. What may also be required for this application is the ability to select SKUs or Classes that comprise a subset of possible SKUs or Classes within the Category, but not the whole category.

   With this latter requirement, a single-select wizard at the category level would not allow the user to filter subsets of SKUs or Classes. What is required is the ability to make multiple selections at the SKU or Class levels in a standard RPAS Two Tree selection wizard, while still ensuring that only one Category is used. Even though this can be achieved through the disciplined selection of SKUs and Classes in a Two Tree Wizard, RPAS allows for setting up a hard constraint so that the wizard itself can keep the user from selecting subsets in more than one Category by displaying an error message and preventing the user from proceeding to the next wizard page until the constraint has been satisfied.



Example Error Message

The constraint can be easily established within the Standard Wizard definition dialog in the Workbook Definition tool of the Configuration Tools. Two new fields are available for every Two Tree selection page in the standard wizard, one where the user selects the level from the hierarchy (Restricted Dim field), and another where the user enters the maximum number of selectable positions from that level (Max Positions field). These fields are optional, and if left empty, there is no limit on the number of positions that may be selected using the wizard. These fields are also available for Two Tree pages used in custom wizards.

Another possible business application of this feature is to constrain the length of the planning period. A retailer may want the planners to never plan more than 12 weeks at a time, and it may not matter whether these weeks belong to the same quarter or not. In such a case, the retailer will want to establish the constraint of 12 at the week

level, the lowest level where the selection is made. The planner may select at the quarter level, thus automatically selecting all the weeks in the Quarter; however, RPAS will ensure that whatever the definition of the Quarter is, it does not contain more than 12 weeks.

Apart from the functional ability to restrict selections to coherent set of positions provided by the Max Positions feature, this feature also allows system designers to constrain the size of workbooks by limiting the maximum number of positions that a user can add to a workbook. In the past, users have been known to add all positions to a workbook because such a selection is easy to make. They may only require 5-10% of those positions, but they still add them all because they can easily work with the desired subset in the workbook. System designers would like to prevent such abuse of the flexibility that workbooks provide, primarily because such abuse leads to wastage of disk space and because it slows down online performance due to the extra work that RPAS has to do with unnecessary positions. System designers may therefore constrain the workbook to, for example, not include more than 500 SKUs at a time. The number 500 may not have any functional meaning, but may be chosen because it does not constrain functionality in any way while still helping to constrain the size of the workbooks.

Note: This is a patchable feature, for example, existing configurations can be enhanced to benefit from this feature.

6. **Wizard Label**: To attach a custom label to a wizard page, enter the desired label in this field. If this field is left blank, the wizard page will be given the default label.
7. **Order**: 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.
8. Click **OK** to save any changes and close the window.

Note: If any of the offered hierarchies are not enabled in the Standard Wizard dialog box, 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.

Enable Image Modification

Select this option to allow users the ability to add, modify, or delete image paths for all image enabled dimensions in this workbook. Refer to "Specify Dimension Properties" for information on enabling images.

Note: A rule will be automatically created to load and commit images for workbooks that are enabled for image modification. If no load or commit rules exist for the workbook, a load or commit rule group will be automatically created to contain this new rule.

Hierarchy Pre-ranging

Hierarchy pre-ranging allows you to filter available positions for selection in two-tree wizards based on relationships established at a specific intersection between the positions of two or more hierarchies. For example, you can set pre-ranging up so that when users select the time period Fall 09 in the wizard, the subsequent SKU selection screen will only display Fall-specific products, such as sweaters and jackets.

Hierarchy pre-ranging can be enabled for both standard wizards and for two-tree wizards in custom wizards; however, in custom wizards the behavior is guaranteed only if two-tree wizards are used as is, i.e., their code is not overridden by the implementation team.

Pre-ranging is achieved by setting up a Pre-range Mask measure for the workbook. This Boolean measure establishes a mapping between the positions of the hierarchies in the intersection of the measure. This mapping is what governs the position availability in the two-tree wizard.

Note when designing pre-ranging the intersection of the Pre-range Mask measure could be different from the base levels of the two-tree wizards, such that the levels of each hierarchy in the base intersection of the measure correspond exactly with the wizard, or are above or below the wizards base level. Additionally, the intersection of the mask measure may not include all hierarchies that are being filtered in the wizards or that are in the workbook. The intersection may even have more hierarchies than those used in the wizard pages. Several combinations are possible; the following such cases detail the behavior of Hierarchy pre-ranging:

- Limits on Dimensionality of the Pre-range Mask

The Pre-range Mask measure must have at least one dimension in its base intersection; i.e., it cannot be a scalar, and it may have up to a maximum of five dimensions. The upper limit is imposed by the limit on dimensionality of a measure in RPAS.

When the mask is one-dimensional, the two-tree page for that hierarchy simply filters the hierarchy independently of selections along any other hierarchy in other pages within the wizard. Since there is only one mask measure per workbook, only one hierarchy may be filtered independently of others; other hierarchies will not be filterable at all unless special provisions are provided in custom wizard implementation.

When the mask is multidimensional, the filter of only the first two-tree wizard is independent of others; available selections in all subsequent wizard pages are reduced to the set of positions mapped to user selections in previous wizard pages. Note that the order of appearance of wizard pages is what is important here and not the order of hierarchies for the domain.

- Constraints on the Pre-range Mask Measures Base Intersection

The base intersection of the pre-range mask measure must be at the same level as or above the lowest base intersection of all measures in the workbook.

The masks base intersection may be below, at the same level as, or above the wizards base intersection, but it cannot be across the wizards intersection. That is, the masks base intersection cannot be such that some of its dimensions are below the base dimensions of the wizard while some are above. All dimensions should be below, at the same level, or above the base dimensions of the wizard.

If the Pre-range Mask measure has one or more dimensions in its base intersection that are higher than the dimensions in the wizard, then pre-ranging will be evaluated by spreading the mask measure value down to the dimensions in the wizard by replication. For example, if the Pre-range Mask is at the Dept/Str level and the standard wizard is at the SKU/Str level, pre-ranging is evaluated by spreading the mask measure down to

SKU/Str using replication. Thus, if Dept/Str is a valid combination, all SKUs in that Department are valid for the Store.

If the mask measure has one or more dimensions that are lower than the dimensions in the standard wizard, pre-ranging will be performed by aggregating the mask up to the wizard level using the measures default aggregation method. That is, if the mask measure has an aggregate type of OR, and if one or more positions in the mask are enabled, the wizard will make the higher level position selectable. It is important to note that when the user selects this higher level position in the wizard, the selection will be replicated down to all positions at the levels below, regardless of whether the lower level position was selectable per the mask or not. To prevent this behavior, the implementer must either make sure that the mask is at the same level as the wizard, or that the mask has a default aggregation type of AND so that the aggregate level position is not available for selection in the wizard if any of the mask level positions that roll up to that position are not selectable.

- Pre-ranging on Hierarchies Not Selectable In the Wizards

For hierarchies that are in the base intersection of the pre-range measure, but are not configured in the standard wizard to be displayed, pre-ranging will be performed based on selections made in the visible hierarchies. The un-displayed hierarchies will then be filtered to only have those positions that correspond to the selected positions for the displayed hierarchies. If none of the hierarchies of the pre-range mask measure are chosen to be displayed in the associated standard wizard, then pre-ranging will be performed based on the mask measure as if all valid positions had been selected.

- Non-Pre-ranged Dimensions In the Wizard

Hierarchies in the standard two-tree wizard pages that are not a part of the base intersection of the pre-range mask measure will not be impacted by pre-ranging; i.e., all positions will be available for selection.

- Non-Populated Pre-range Mask Measure

If at the time of building the workbook the Pre-range Mask measure has no cell with a value of TRUE, an error message will appear and the wizard will exit without building the workbook.

- Non-Boolean Pre-range Mask Measures

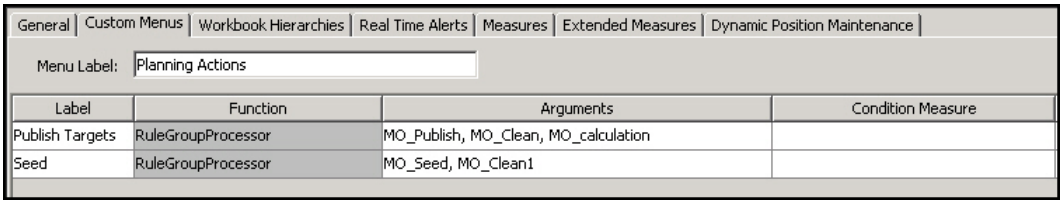
If a non-Boolean measure is used as the mask, when building the workbook an error message will appear and the wizard will exit without building the workbook.

Note that the state of the Pre-range Mask measure is checked when the wizard process starts and not during the wizard process. As a result, changes to the mask measure during a wizard process will not be reflected in the wizard. Furthermore, cached user selections are retained when rebuilding a pre-ranged workbook; however, if pre-ranging removes any of the positions that were in the users previous selection, the workbook build will successfully complete with those positions eliminated (there must be at least one selectable position).

Hierarchy pre-ranging also works with batch workbook builds. Situations that lead to errors when building a pre-ranged workbook manually also cause batch builds to fail.

Custom Menus Tab

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



Example of Custom Menus Tab

Create a Custom Menu

1. Select 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:** This field 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. There should be no gap after the comma between any two adjacent arguments. 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 the end user selects 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 arguments property should meet the naming conventions for the operating system of the RPAS Server. 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 to be called requires different arguments, a "wrapper" script should be called instead, 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 the called script does not need to finish before the next process begins, a "wrapper" script should be implemented that can call the target script, and then return immediately.

- **Condition Measure:** This field is used to specify a scalar, Boolean condition measure in the workbook that will be checked by the custom menu to decide whether it must execute or not.

  In order for a measure to be a candidate for the condition measure of a custom menu item, the measure must meet the following criteria:

- In the workbook. The measure must exist within the rule groups for the workbook.
- Scalar.
- Boolean.

If the value of the measure is TRUE, the custom menu executes, but if the value is FALSE, the menu does not execute and displays a message relating that the custom menu could not execute because the conditions were not met. (For more information on how this message can be customized, see the Return Message Measure bullet.)

If the condition measure is not specified, meaning the field is empty, the custom menu always executes. The table below specifies the behavior of custom menu execution and the display of custom messages based on whether a measure name has been entered (available) in the field and whether the measure's value has been set (TRUE in case of the condition measure and a non-zero length string in case of the message measure).

Condition Measure	Return Message Measure	Behavior
Available & Set	Available & Set	The Custom Menu executes and displays the Custom Menu Response pop-up containing the value of the Return Message measure.
	Available but Not Set	The Custom Menu executes and displays the default message.
	Not Available	The Custom Menu executes and displays the default message.
Available but Not Set	Available & Set	The Custom menu does not execute, but does display the Custom Menu Response pop-up containing the value of the Return Message measure.
	Available but Not Set	The Custom Menu does not execute and displays a Warning pop-up message reading "Conditions for executing the Custom Menu have not been met!".
	Not Available	The Custom Menu does not execute and displays a Warning pop-up message reading "Conditions for executing the Custom Menu have not been met!".
Not Available	Available & Set	The Custom menu executes and displays the Custom Menu Response pop-up containing the value of the Return Message measure.

Condition Measure	Return Message Measure	Behavior
	Available but Not Set	Custom Menu executes and displays the default message.
	Not Available	Custom Menu executes and displays the default message.

- **Return Message Measure:** This field is used to specify the scalar, String measure, the value of which is displayed by the custom menu in a pop-up dialog upon the menus successful or failed execution. If the field is empty, RPAS will display the default message that it has historically displayed. If a measure is specified, but the value is empty, RPAS will again display the default message. If the value is a non zero-length string then the value is displayed.

In order for a measure to be a candidate for the return message measure of a custom menu item, the measure must meet the following criteria:

- In the workbook. The measure must exist within the rule groups for the workbook.
- Scalar.
- String.

To effectively use this feature it is important to understand the execution of a custom menu. When you select a custom menu from the menu, RPAS first checks if there is a condition measure available for controlling the execution. If there is none, it continues to run the rule groups or scripts in the argument of the custom menu. If a condition measure is available, RPAS checks the value of the measure for controlling the execution of the custom menu. If the value is false, it checks the availability of the return message measure. If the message measure is unavailable, RPAS displays a default message informing the user that menu could not be executed because the conditions were not met. However, if the measure is available, RPAS examines its value. If the value is empty, it defaults to the same behavior as when the message measure was not available. If the value is not empty, it displays the custom message.

If the custom menu can execute, either because the condition measure is unavailable or because it is set to true at the time the custom menu is invoked, RPAS executes the rule groups and scripts in the argument of the custom menu specification. RPAS will then look for the value of the message measure, and if the value is empty, it informs the user that the menu was successfully executed using the default message. If the value is not empty, it displays the value of the message measure. It then transitions to the Calc rule group and completes execution.

- Commit ASAP: It is possible to include commit rule groups (rule groups that include commit rules) in the list of rule groups that forms the arguments of a custom menu item. Normally, these commit rule groups execute synchronously, meaning that after submitting a commit request, the user must wait until the commit process ends. It is also possible to mark a custom menu item as Commit ASAP.

When marked as Commit ASAP, a custom menu item runs the commit rule group asynchronously, returning the control to the user immediately. The commit ASAP process creates a temporary copy of the workbook and places a commit request in a queue. The commit rule group is executed as soon as the measure databases become available.

In this manner, it is possible to commit certain measures within the workbook without halting work on the workbook until the commit has been processed and without having to deal with contention issues associated with immediate commits.

Note that when a custom menu item is marked as Commit ASAP, that menu item may only have a single commit rule group in its list of arguments. Furthermore, it is recommended that the commit rule group be the last rule group in the custom menu item's list of arguments, as any subsequent rule groups that execute will not affect the values committed and any side effects of the commit rule group execution will not apply to the workbook.

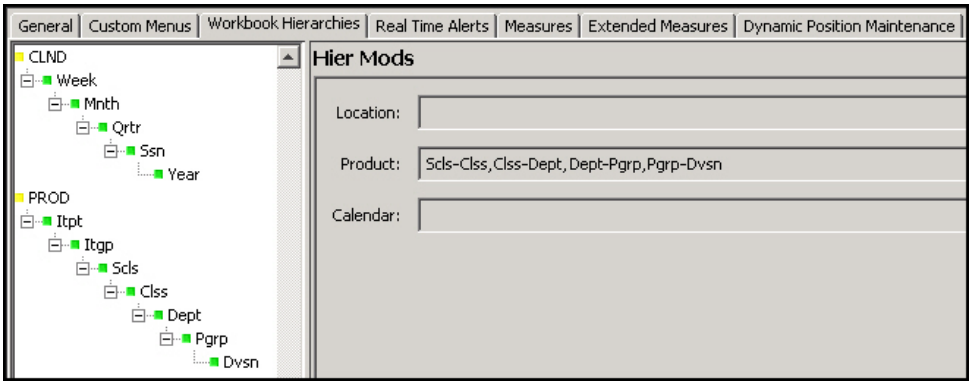
- **Intraday Concurrent**: This field is used to flag the custom menu as one that can run concurrently with any intraday process running on the domain. The default setting is false which means that the custom menu will not run when an intraday process is running. Please see the Oracle Retail Predictive Application Server Administration Guide for more details on intraday processes.

A custom menu that is configured to run intraday concurrent should be something that only accesses workbook data, runs a script that uses the ride utility and/or runs commits via the commit ASAP functionality. Custom menus that updates or reads directly from the domain should not be configured as intraday concurrent as this would conflict with the ride process.

Workbook Hierarchies Tab

This tab is optional. Use this tab to configure hierarchical relationships whose parent-child relationships are not defined in The Hierarchy Definition Window or the loadHier process, but are data driven. For instance, you can:

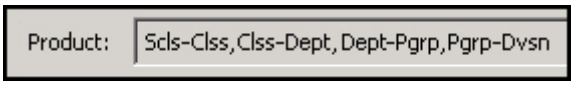
- Hide individual dimensions or whole branches of a hierarchy by excluding them from the dimensions that are available to the workbook. (For more information, see "Remove Dimension.")
- Create mappings between dimensions that are not directly related in the hierarchy structures. (For more information, see "Change Rollup.")
- Create dynamic hierarchical relationships that 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 instructions on creating dynamic hierarchies, see "Make Rollup Dynamic," "Add Workbook Dimension," and "Insert Workbook Dimension." For in-depth examples, see the "Dynamic Hierarchies" appendix.)



Workbook Hierarchies Tab

Any hierarchies that do not have dimension relationships specified will use the full hierarchy specified in the Hierarchy Definition Window, with the lowest dimension in that hierarchy used as the base intersection of a measure used in the rule set for the workbook as its root.

Dimension relationships are specified as hyphenated child-parent pairs. In these pairs, the child is listed first (for example scls-clss). More than one child-parent pairs are separated by commas, as shown below.



Child-Parent Pairs

> **Note**: The hierarchy fields in the Hier Mods section are read-only. The child-parent pairs displayed cannot be edited there.

Every parent-child relationship in the workbook must be explicitly specified. Only dimensions that have been defined in The Hierarchy Definition Window can be used and must be specified by name. The Configuration Tools validates the dimension relationships to ensure that valid dimension names are used. It prohibits the use of the same dimension name in both the child and the parent dimensions. In addition, the Configuration Tools prohibits pairs where the child's Aggs attribute is the same as the parent's. For example, Clss-Scls is not allowed but Scls-Clss is.

To modify a hierarchy in a workbook, perform the following steps:

1. Click the **Workbook Hierarchies** tab. In this tab you can see all hierarchies used in the base intersection of measures used in the rule set in the workbook.
2. Right-click a hierarchy or dimension and choose one of the following options:
   - Remove Dimension
   - Restore Dimension
   - Change Rollup
   - Make Rollup Dynamic
   - Remove Dynamic Rollup
   - Add Workbook Dimension
   - Insert Workbook Dimension

For instructions for each of these options, scroll down.

> **Note**: You cannot use these options on the Calendar hierarchy.

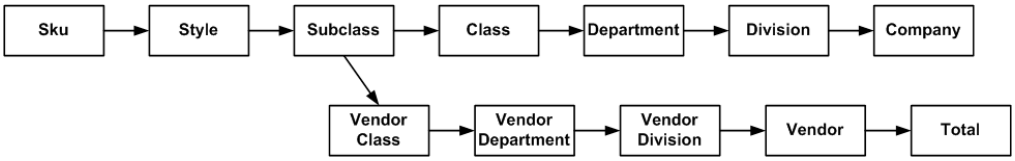
Important Note About Using Hier Mods Options

The use of mappings that are non-structural should be carefully managed to ensure they are only used where the non structural mapping work. For example, 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:

1. **Sku-Style-Subclass-Class-Department-Division-Company**

   with a branch of:

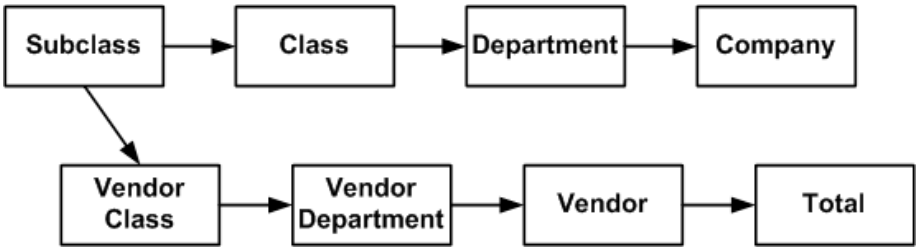
   Subclass-VendorClass-VendorDept-VendorDiv-Vendor-Total



If you want a workbook that includes measures with a lowest base intersection of subclass that required the vendor branch but not the division dimension, you could specify the Hier Mods as follows:

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

The workbook would contain the subclass, class, department, and company dimensions, as well as a branch that contains the VendorClass, VendorDept, Vendor, and Total dimensions.



In the example above, if this workbook is accessed by a user in a part of the business where a subclass only includes 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 includes styles with multiple vendors, RPAS determines (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 builds 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 is of little or no practical value in this case.

Remove Dimension

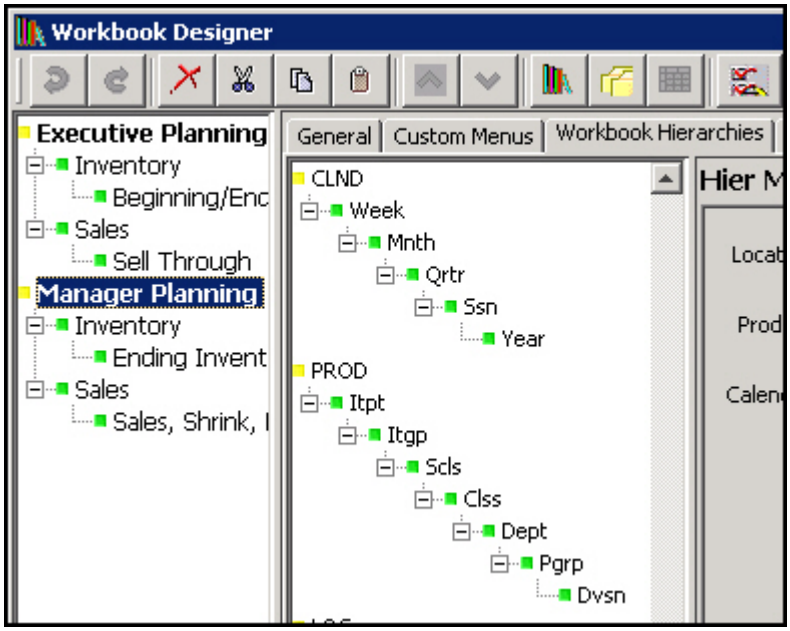
This option allows you to remove existing dimensions from a particular workbook. For instance, if you have a domain that has a product hierarchy with the following dimensions:

subclass > class > department > division > company

But you want a particular workbook to stop at the department dimension and not include the division and company dimensions. You can make that workbook's product hierarchy end at department by performing the following steps:

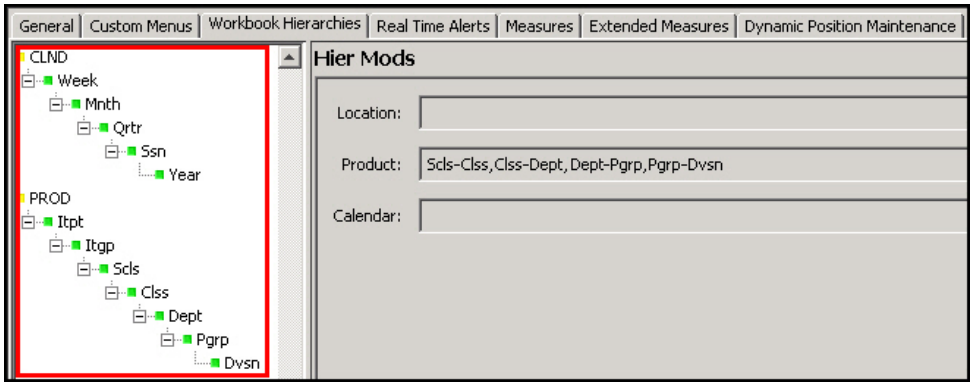
> **Note**: You cannot remove the lowest child-parent relationship (the last two dimensions) of a hierarchy.

1. Select the workbook in the Workbook Designer window.



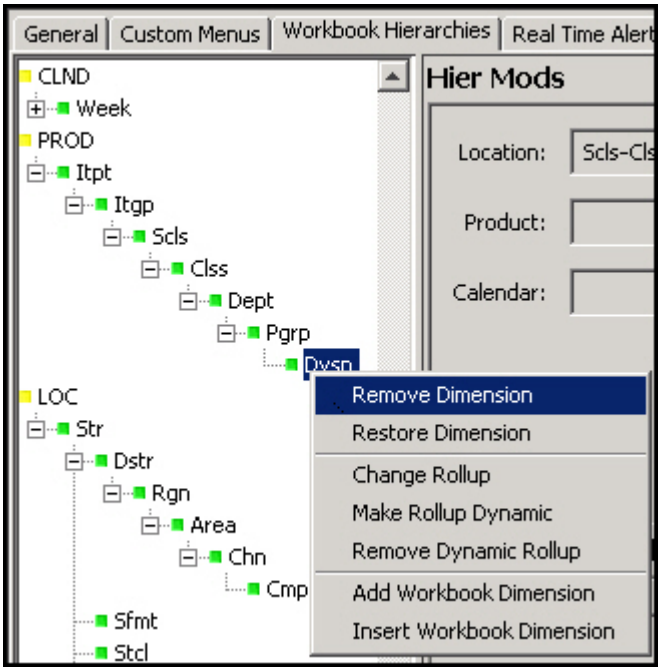
Workbook List in Workbook Designer

2. Navigate to the **Workbook Hierarchies** tab. The available hierarchies in the workbook are displayed.



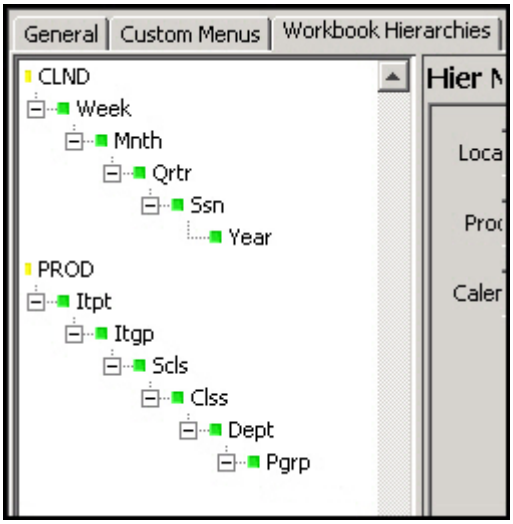
Displayed Hierarchies and Dimensions in the Workbook Hierarchies Tab

3. Right-click the dimension you want to remove. From the right-click menu, select **Remove Dimension**.



Remove Dimension Option

The dimension disappears and the hierarchy name becomes italicized to show that the hierarchy has been modified.



Removed Dimension

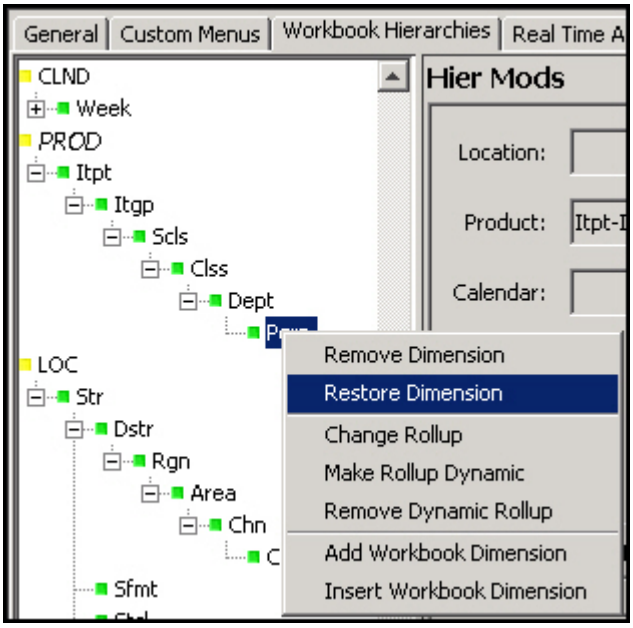
4. Repeat step 3 for other dimensions if desired.

To bring back a removed dimension, use the **Restore Dimension** option described in the next section.

Restore Dimension

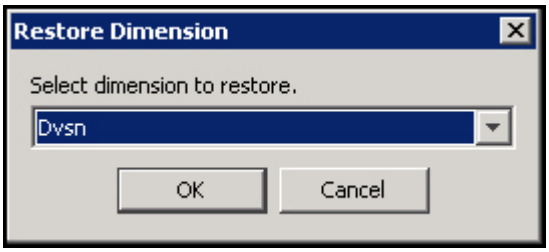
To restore a removed dimension, perform the following steps:

1. Right-click the hierarchy or any dimension within the hierarchy of the removed dimension.
2. In the right-click menu, click **Restore Dimension**.



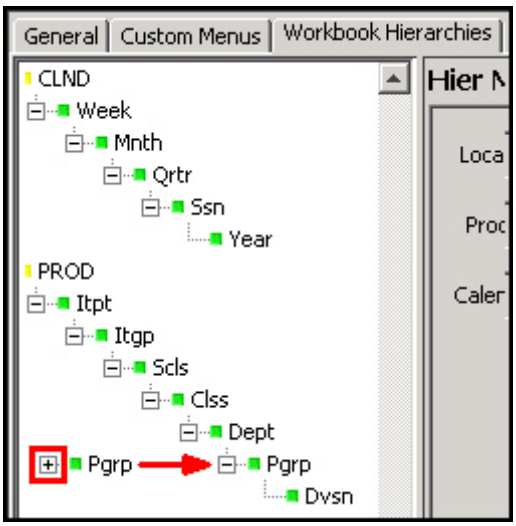
Restore Dimension Option

3. The Restore Dimension dialog box appears. From the dropdown list, select the dimension you want to restore. Click **OK**.



Restore Dimension Dialog

The Restore Dimension dialog box disappears. In the Workbook Hierarchies pane, the dimension above the restored dimension will have an expand icon next to it. Click the icon to see the restored dimension.

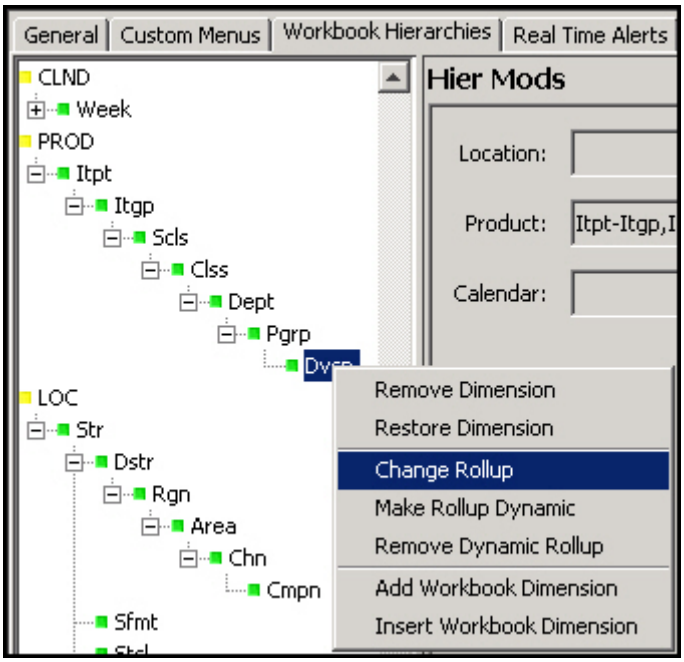


Restored Dimension

Change Rollup

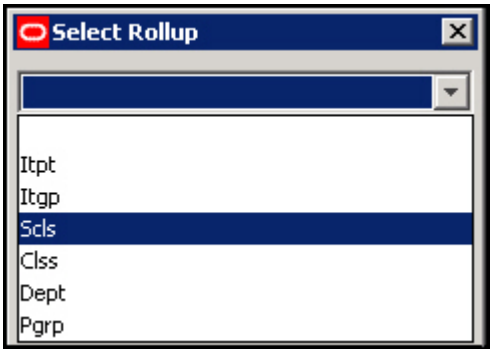
You can make a dimension rollup to a different dimension by using the Change Rollup feature. For instance, if you want the company dimension to rollup from subclass rather than division, you would perform the following steps:

1. Right-click the dimension you want to change.
2. In the right-click menu, click **Change Rollup**.



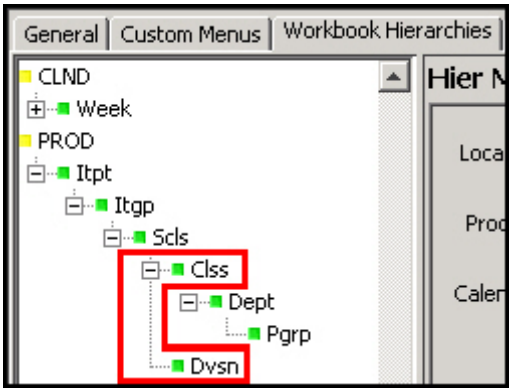
Change Rollup Option

3. The Select Rollup dialog box appears. Select the dimension that you want the original dimension to roll up from. (Only dimensions below the original are available.) Click **OK**.



Select Rollup Dialog

The Select Rollup dialog box disappears. In the Workbook Hierarchies pane, the dimension now rolls up from the new dimension.



New Rollup

Make Rollup Dynamic

Use this option to create a dynamic hierarchy entry for a dimension. Dynamic hierarchical relationships are built using measure data during the workbook build process and may vary each time a workbook is built, but the hierarchical relationships within the workbook remain constant. Dynamic hierarchies can be based on two or more other hierarchies.

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 the user brings 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.

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 Definition window 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.

Important Dynamic Hierarchy Notes:

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 must use the Remove Dimension option to remove 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 the Remove Dimension option to hide that relationship in a workbook template that does not include the product hierarchy.

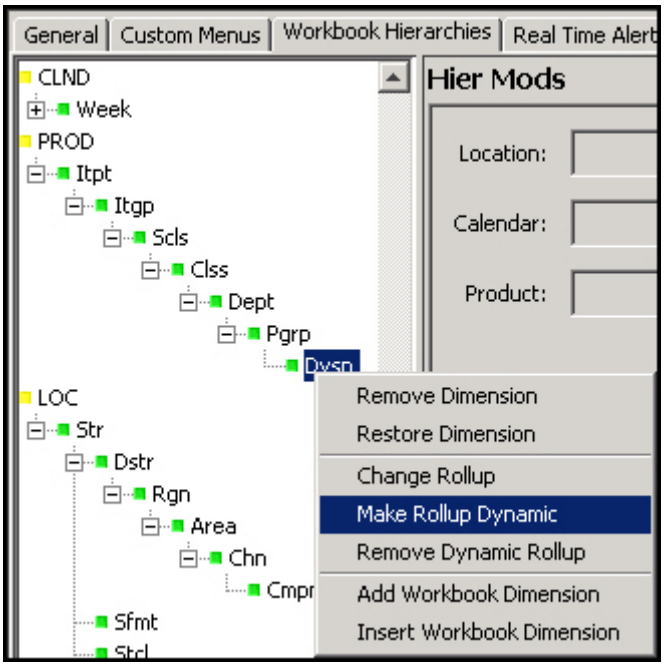
It is your responsibility to ensure that the position names contained in the measure that drives the dynamic hierarchy are real positions that exist in the parent dimension. If not, positions with those names will be present in the workbook, but data for them cannot be committed to the domain, and it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

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.

> **Reference**: For in-depth examples and explanations about dynamic hierarchies, see the "Dynamic Hierarchies" appendix.

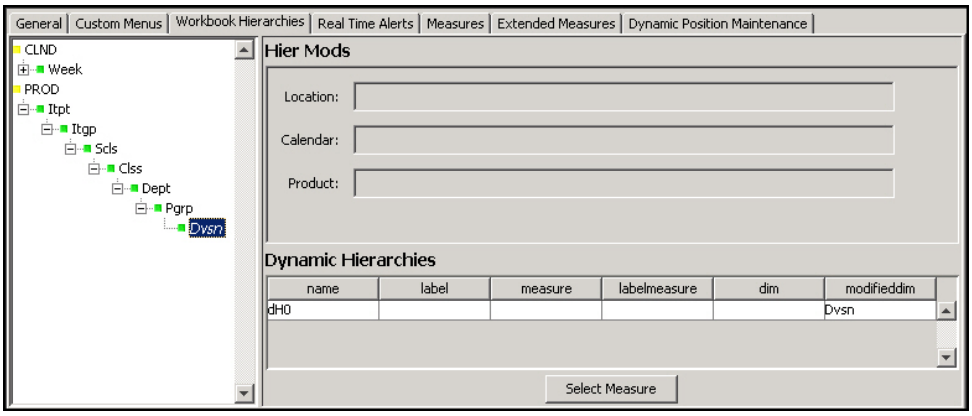
To make a rollup dynamic, perform the following steps:

1. Select the dimension you want to change.
2. In the right-click menu, click **Make Rollup Dynamic**.



Make Rollup Dynamic Option

The selected dimension is now italicized and a dynamic hierarchy appears in the Dynamic Hierarchy section. The dimension you right-clicked in Step 1 is populated in the modifieddim attribute.



New Dynamic Rollup

3. In the Dynamic Hierarchies section, set the following properties:
   - **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. It is not visible to the end user.

- **Label** – The label of the dynamic hierarchy entry that represents the workbook-only dimensions that can be configured with an external label. This external label is displayed within the RPAS Classic Client or RPAS Fusion Client in place of the dimension name. Only dynamic hierarchy entries that represent workbook-only dimensions have editable label properties.
- **Measure** – This is the name of the measure that holds the name of the parent position. Click Select Measure to see the 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's base intersection must contain the dimension that is the child of the child-parent relationship (dvsn in the example) or is below it (dept, clss and scls in the example) and one or more dimensions which the dynamic position assignment is dependent on (such as comp). The content of the measure is the name of the parent position in the relationship (in the example, this would be the name of the cmpp that the dvsn belongs to for this comp). 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 after 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 must 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.

- **Dim** – This is the name of the dimension that is the child in the parent-child relationship. It is the dimension in the modified hierarchy that is used to resolve the modified roll-up relationship represented by the dynamic hier entry. It is stored within the measure and the label measure. The hierarchy that the dynamic hierarchy belongs to is derived from this property.

  Note: A dimension that is the modified dimension of one dynamic hierarchy entry cannot be used as the dim attribute of another dynamic hierarchy.

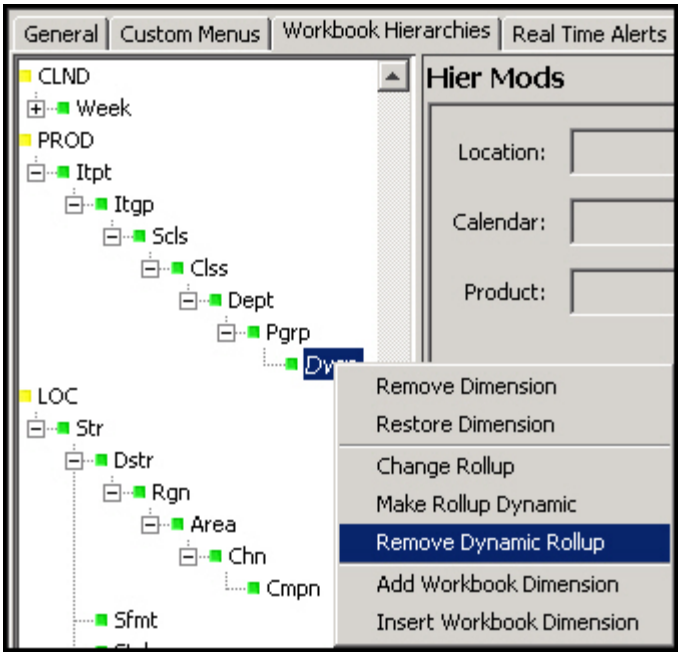
- **Modified Dim** – This is the name of the dimension that is the parent in the parent-child relationship. It is the dimension within the modified hierarchy whose roll-up behavior is being modified by the dynamic hierarchy entry. Note that, for dynamic hier entries that represent workbook-only dimensions, the modified dim attribute will be the name of the dynamic hier entry and the name of the workbook only dimension that this entry represents.

If you want to remove this new hierarchy, use the instructions described in the "Remove Dynamic Rollup" section.

Remove Dynamic Rollup

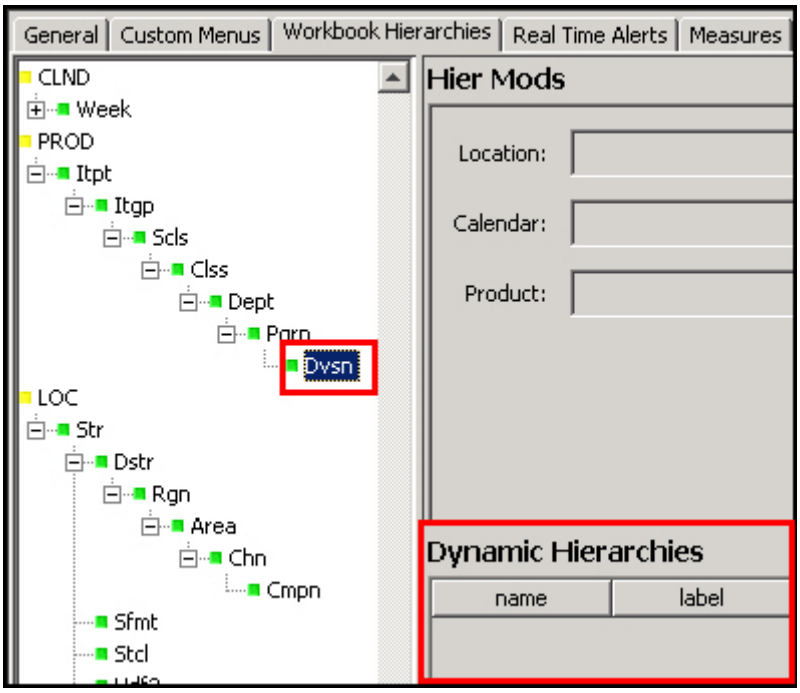
If you have added a dynamic rollup with the Make Rollup Dynamic option, you can remove it with the Remove Dynamic Rollup option. To remove a dynamic rollup, perform the following steps:

1. Right-click the dimension with the dynamic rollup. The dimension's name is in italics
2. In the right-click menu, select **Remove Dynamic Rollup**.



Remove Dynamic Rollup Option

The dynamic rollup is removed from the Dynamic Hierarchies section and the dimension name is no longer in italics.



Removed Dynamic Hierarchy

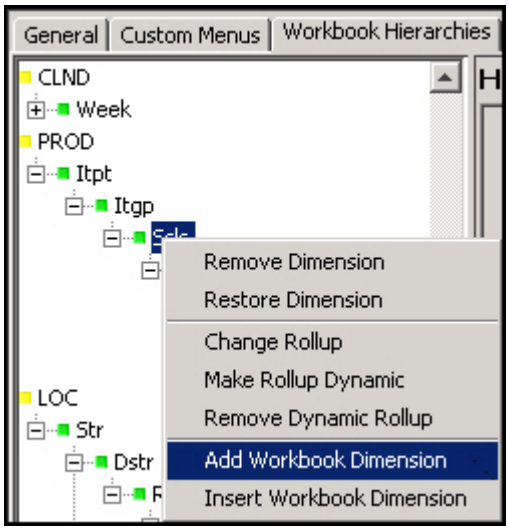
Add Workbook Dimension

Use this option to add a workbook-only dimension to the workbook. Workbook-only dimensions are driven by measures just like regular dynamic dimensions. The difference

is that workbook-only dimensions exist only in the workbook and never in the domain. They can also be used to support a dynamic number of roll-ups and levels. If the workbook dimension does not have data in the measures, it is not displayed as a dimension in the workbook. For more information about workbook-only dimensions, see the Dynamic Display of Dynamic Branches section.

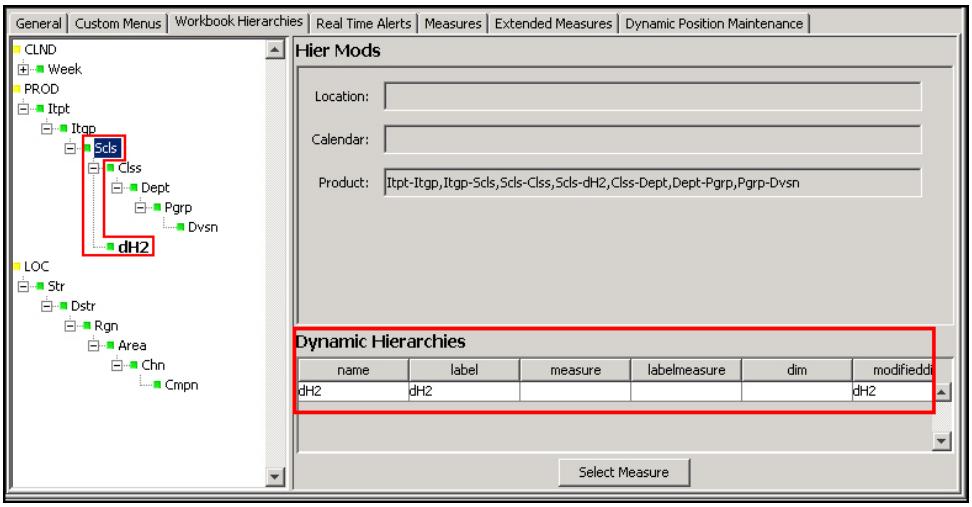
To add a workbook dimension, perform the following steps:

1. Right-click the dimension that you to add a dimension above.
2. In the right-click menu, click **Add Workbook Dimension**.



Add Workbook Dimension Option

The new workbook dimension appears below the selected dimension as well as in the Dynamic Hierarchies table.



New Workbook Dimension

To continue editing the workbook dimension hierarchy, see the instructions in the "Make Rollup Dynamic" section. To remove it, see the "Remove Dimension" section.

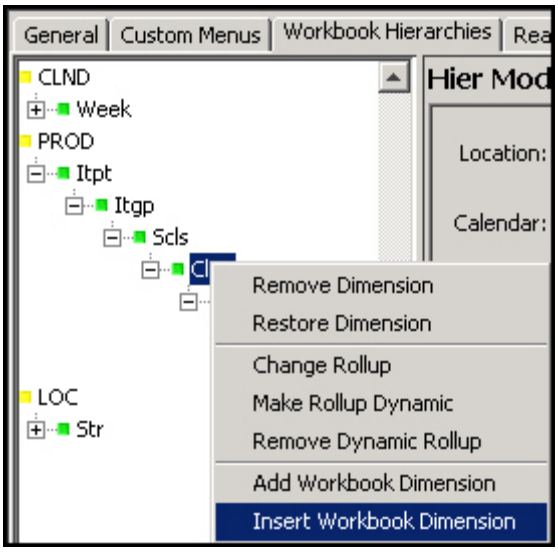
Insert Workbook Dimension

Use this option to insert a workbook-only dimension between existing dimensions in the workbook. Workbook-only dimensions are driven by measures just like regular dynamic

dimensions. The difference is that workbook-only dimensions exist only in the workbook and never in the domain. They can also be used to support a dynamic number of roll-ups and levels. If the workbook dimension does not have data in the measures, it is not displayed as a dimension in the workbook. For more information about workbook-only dimensions, see the Dynamic Display of Dynamic Branches section in the Dynamic Hierarchies appendix.

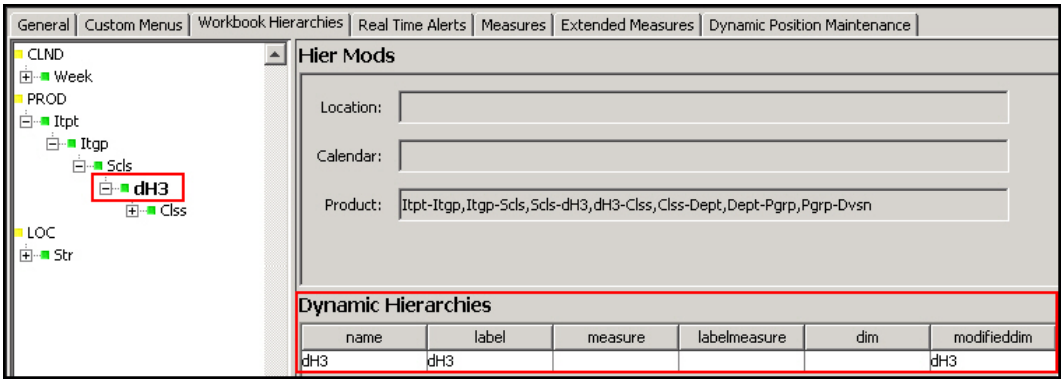
To insert a workbook dimension, perform the following steps:

1. Right-click the dimension that you to insert a dimension below.
2. In the right-click menu, click Insert Workbook Dimension.



Insert Workbook Dimension Option

The new dimension appears above the selected dimension as well as in the Dynamic Hierarchies table.



New Workbook-Only Dimension

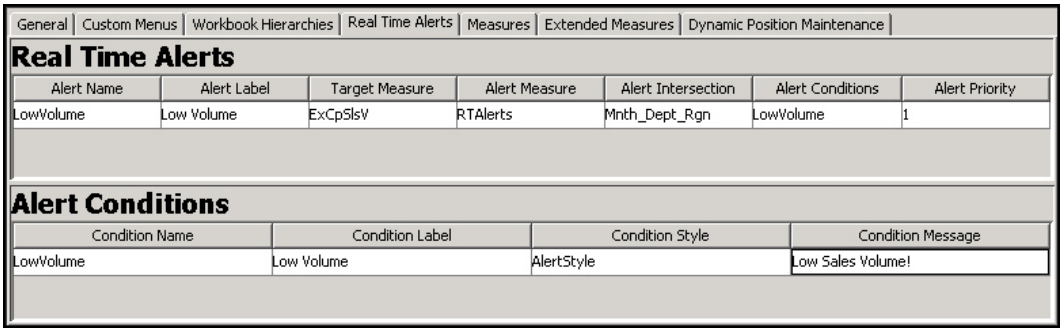
To continue editing the workbook dimension hierarchy, see the instructions in the Make Rollup Dynamic section. To remove it, see the Remove Dimension section.

Real Time Alerts Tab

The use of this tab is optional. It allows you to create real time alerts within a workbook that update dynamically. Unlike batch alerts configured through the Measure Tool, real time alerts will re-evaluate every time the measures upon which they depend are modified. Also unlike batch alerts, Real Time Alerts are not two-state (either a hit or not a hit); any number of discrete conditions can be defined for a Real Time Alert. When the

alert is evaluated, whichever condition applies will be used (e.g. a Real Time Alert defined on an inventory measure could define one condition for low stock and a more severe condition for no stock). The Fusion Client provides the following support for real time alerts:

- A Real Time Alert can be set as the active alert. When an alert is active within the Fusion Client, users can navigate between hits for that alert in the same fashion that the user can navigate batch alerts used to build the workbook.
- A Real Time Alert can define one or more styles for a Real Time Alert. Every cell of the target measure which evaluates as an alert hit for the Real Time Alert will use the alert's style for that cell in place of the measure's default style. As Real Time Alerts evaluate, the cell styles will update to reflect changes in which cells are hits and which are not.
- One or more messages can be defined for a Real Time Alerts. This message will be displayed as a tool tip when the user hovers the cursor over a cell that is a hit for that alert.

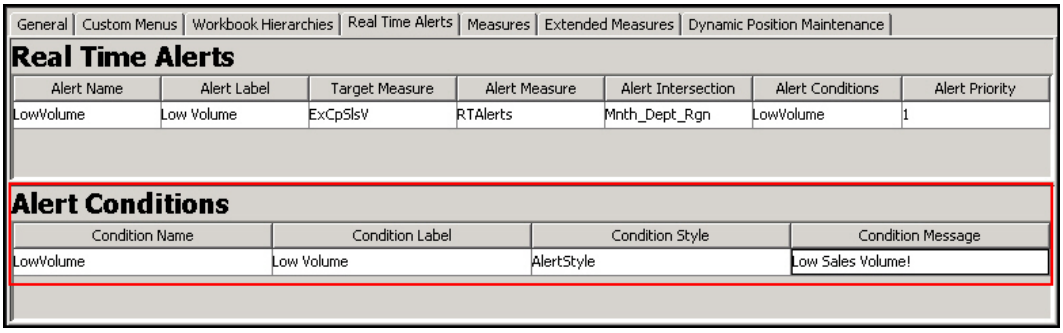


Real Time Alerts Tab

Condition Definition Interface

The Condition table will allow users to create, remove and modify the conditions that are used in alerts for the current workbook. This table will contain columns to allow the specification of the following properties:

- **Condition Name** – The name will be the value populated within the alert measure to identify which condition, if any, has been triggered for a given cell in the target measure. The condition name will also be used to create a key to allow translation of condition messages.
- **Condition Label** – The label will be the externalized label used to represent a condition to users of the client. This label will be used to identify the condition within the client. The condition label will be internationalized so that it can be translated using the RPAS multi-language functionality.
- **Condition Style** – The style will be the name of a style configured within the Style Definition tool that describes the formatting that should be applied to those cells which are evaluated as 'hits' for the condition.
- **Condition Message** – The message will be the text message that will be supplied to describe the condition that has been triggered when the workbook alert evaluates a 'hit' on a given cell of the base measure. This message will be internationalized so that it can be translated using the RPAS multi-language functionality.



Condition Definition Table

Alert Definition Interface

The Alert table will allow users to create, remove and modify workbook alerts used within the selected workbook. Selecting the row corresponding to an alert will cause the alert conditions of that alert (if it has any defined) to be populated within the condition table. The alert table will contain columns to allow the specification of the following properties:

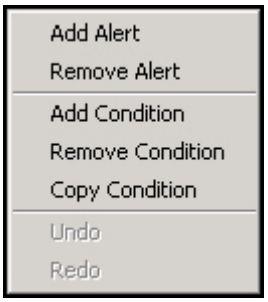
- **Alert Name** – The name is used as an internal key by RPAS.
- **Alert Label** – The label is an external identifier that is used to represent the alert to users of RPAS.
- **Target Measure** – The target measure is the measure over which alert 'hits' will be evaluated and to which the formatting will be applied.
- **Alert Measure** – The alert measure is a String measure that is used in the alert evaluation process. It is this measure that will hold the information about what cells in the target measure are considered 'hits' and which condition a given cell has triggered. The intersection of the alert measure should be identical to the alert intersection property of the workbook alert.
- **Alert Intersection** – The alert intersection is the intersection on which the workbook alert is defined. It is possible to define multiple workbook alerts over a single target measure, each at a different intersection. Because an alert can only register 'hits' at the intersection of the alert measure, it may be desirable to create multiple alerts for a single target measure so that users can evaluate the target measure at different intersections within the client.
- **Alert Conditions** – The alert conditions field contains the list of conditions (of the full set of defined conditions for a workbook) that are associated with a given workbook alert. This list is used to identify which conditions should be grouped with an alert when a workbook is registered or built.
- **Alert Priority** – The alert priority is used when more than one alert has been created for a single target measure. In such cases, some cells of the target measure may trigger alert hits for more than one alert. When this is the case, the alert with the lower priority will be applied to that cell of the target measure. The alert priority is validated to be unique for the same target measure.

Real Time Alert Definition table

Operations within the Real Time Alerts panel

The Real Time Alerts panel supports the following operations through the right-click menu:

- Add Alert
- Remove Alert
- Add Condition
- Remove Condition
- Copy Condition



Real Time Alerts Menu

Add Alert

Selecting the Add Alert action will create a new Real Time Alert and allow specification of properties within the Alert Table.

Remove Alert

Selecting the Remove Alert action will remove the currently selected alert and its conditions from the workbook. If no alert is currently selected, this option will be disabled.

Add Condition

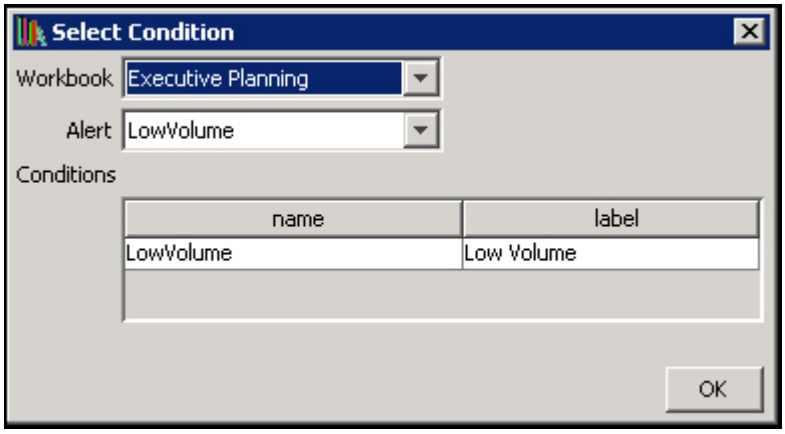
Selecting the Add Condition action will create a new Alert Condition within the currently selected real time alert and allow specification of properties within the Condition Table. If there is no currently selected Real Time Alert, this option will be disabled.

Remove Condition

Selecting the Remove Condition action will remove the currently selected condition from the alert. If no condition is currently selected, this option will be disabled.

Copy Condition

Selecting the Copy Condition action will allow users to copy an alert condition defined in another real time alert so that it can be used in the currently selected real time alert. When using the Copy Condition action, users must specify solution (for multi-solution configurations), workbook and source alert from which the condition will be copied. The user can then select the condition to be copied to the currently selected alert. If there is no currently selected Real Time Alert, this option will be disabled.



The Copy Condition Dialog Box

Example of workbook configuration

The process of configuring a workbook alert requires multiple steps across several functional areas within the Configuration Tools. For this reason, an example is being provided that details the end-to-end process of the workbook alert configuration.

For purposes of this example, assume the existence of an inventory measure. The desire is to create an alert to warn of low stock. Assume further that the process defines two conditions that would be considered as a low stock situation: first, if the inventory count falls below a projected sales figure for the period; second, if the inventory count falls below some static threshold. For our purposes, we will assume that the failure to meet projected sales is considered more important than falling below the static threshold.

Given these assumptions assume that the following three measures already exist in the domain and have processes in place to be calculated or seeded:

- **InvU** – base inventory count
- **ProjSls** – projected sales
- **MinThrsh** – low stock threshold

Assume also that the workbook in which the alert will be defined is already configured to contain all three of the above measure and that processes exist to load and/or calculate the values of the three measures.

Given these inputs, the following steps illustrate the configuration of a workbook alert to evaluate low stock conditions.

Definition of the condition styles

Within the Style Definition tools, styles must be configured to represent the formatting to be applied for each condition within the alert. Given that we have defined two distinct conditions, it would be desirable to define two distinct styles so that it is possible to visually distinguish which condition is triggered for any given alert hit. Note that it is not necessary to create new styles for each condition; styles may be reused between conditions and may be used both for static formatting of measures and for alert hits.

For our example, we will assume that the following new styles are created.

Warning: Text color yellow, Font type: bold

Error: Text color red, Font type: bold

For more information on style configuration, please see the section, ***Styles*** earlier in this document.

Definition of alert measures

Within the Measure Definition tool, an alert measure must be defined for the workbook alert. This measure must be a String-type measure and its base intersection should be identical to the intersection at which the alert is going to be evaluated. In addition, the naValue of the alert measure should always be the empty string.

The following measure will be created within the Measure Definition tool:

LowStkAlrt – type: String, naValue: “”

Note that the new measure can be configured as by identical processes to normal measure configuration. For more information, please see the section, ***Measures and Components*** earlier in this document.

Creation of alert calculation rule

Within the Rule Definition tool, a rule must be defined to evaluate the alert measure. This rule should be added to the calc group for the workbook so that it will be re-evaluated as a part of every calc cycle to allow the workbook alert to update as the data within the workbook changes. Because the alert in question contains two conditions, the rule must be able to evaluate either of the conditions.

The general form of alert calculation rules is one or more if statements that assign the name of an condition to the alert measure if the condition representing the alert is triggered. The assignment of an condition is handled by using a string literal that contains the name of the condition. For our example, the alert rule would be:

LowStkCalc:

LowStkAlrt = if (InvU < ProjSls, “Condition1”, if (InvU < MinThrsh, “Condition2”, “”))

For more information, please see the section, ***Rules*** earlier in this document.

Creation of the workbook alert

Within the Workbook Definition tool, a workbook alert must be created. This new alert should be configured according to the configuration already performed.

Real Time Alerts

Alert Name	Alert Label	Target Measure	Alert Measure	Alert Intersection	Alert Conditions	Alert Priority
Alert1	Low Stock Alert	InvU	Alert1Meas	Mnth_Dept_Rgn	LowStock1,LowStock2	1

Example of Configuration of Workbook Alert

Creation of alert conditions

Within the Workbook Definition tool, conditions must be defined for the workbook alert. These two conditions will correspond to the two triggers that will be a part of the alert. Note that it is not necessary to configure conditions before the creation of the alert rule but since the rule will make use of the condition names, it is suggested that they be created prior to the rule.

Alert Conditions

Condition Name	Condition Label	Condition Style	Condition Message
Condition1	Low Stock	WrnStyle	Stock Below Minimum Level
Condition2	Out of Stock	ErrStyle	Stock cannot meet expected sales

Example of Configuration of Conditions

At this point, the workbook alert has been configured. Upon the execution of the calculation cycle, the alert measure will be populated with the appropriate condition for any cell that is evaluated as an alert hit. This information will then be used by the client to display information about the workbook alert.

Workbook Transitions Tab

The use of this tab is optional. It allows you to define automatic transitions between the worksheets of a workbook within the Fusion Client. This transition is initiated through selecting an item in the context menu. When a transition is selected, the current worksheet will be de-activated and a target worksheet will be activated.

General | Custom Menus | Workbook Hierarchies | Real Time Alerts | Workbook Transitions | Measures | Extended Measures | Dynamic Position Maintenance

Worksheet Transitions

Label	Trigger Type	Trigger Content	Destination	Shared Context Dims
WkshtTransition	Worksheet	WkstTrigger	TargetWkst	PROD,LOC
MeasTransition	Measure	ex_b	TargetWkst	PROD,CLND
DimTransition	Dimension	SKU	TargetWkst	LOC,CLND

Worksheet Transitions Tab in the Workbook Designer

Defining a Worksheet Transition

1. Select the Workbook Transitions tab.
2. Right-click in the transitions table area and select Add.
3. Enter the following information:
   a. **Label** – The label for a worksheet transition will be the text displayed within the right-click menu of the Fusion Client.
   b. **Trigger Type** – The trigger type is used to define the type of content that acts as the trigger for this transition. There are three possible types of triggers: worksheets, measures and dimensions.
   c. **Trigger Content** – The trigger of a worksheet transition determines under which circumstances the menu item for a transition will be available in the context menu. When the trigger is a worksheet, the item will be available from any selection of the specified worksheet. If the trigger is a measure, the item will be available whenever the selection includes the specified measure. If the trigger is a

dimension, the item will be available when a position of the specified dimension is selected.

d. **Destination** – The worksheet that will be activated when a worksheet transition is selected from the context menu of the Fusion Client.

e. **Shared Context Dims** – When making the transition from the source to destination worksheets, the Fusion Client is capable of automatically applying the selection context of the source worksheet to the destination. Hierarchies listed within the Shared Context Dims property will have this automatic filtering applied.

Removing a Worksheet Transition

In order to remove a worksheet definition from the workbook:

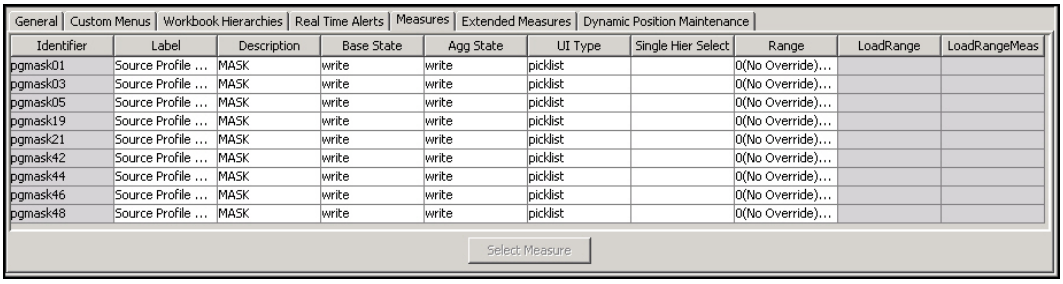
1. Select the Workbook Transitions tab.
2. Select an existing worksheet transition from the worksheet transitions table.
3. Right-click and select Remove.

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 in cases when it is necessary to 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 writable 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
- Range

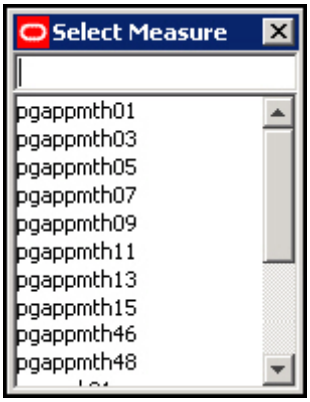
In addition, there are two properties, LoadRange and LoadRangeMeas that are not standard measure properties that may only be set though the Measures tab.



Example of Measures Tab

Defining Measure Properties Override Settings for a 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. The Select Measure window appears.



Select Measure Window

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.

Modifying Measure Properties Override Settings for a Workbook

1. Click the **Measures** tab.
2. Modify the appropriate property information for the measure.

Removing a Measure Override Settings from a Workbook

Perform the following procedure to remove the measure property override settings.

1. Click the **Measures** tab.
2. Right-click on the measure row you want to delete and select **Remove**.

Defining the LoadRange and LoadRangeMeas Properties

The **LoadRange** and **LoadRangeMeas** property fields 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 backwards 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.

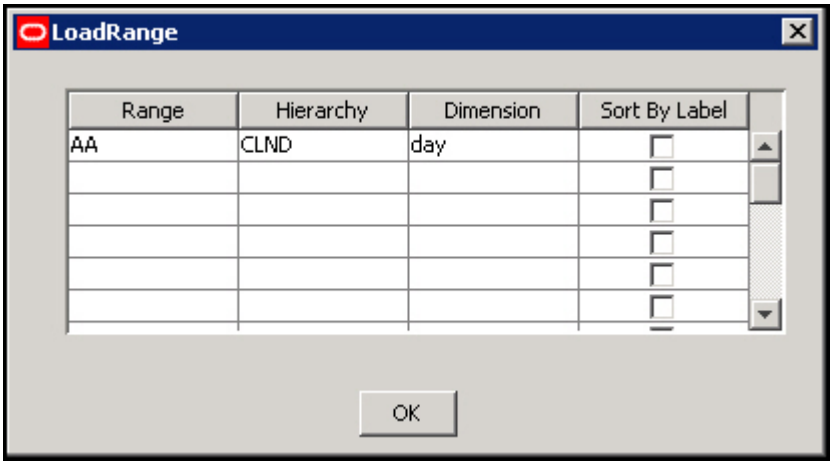
Dynamic picklists are of two forms:

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.

Defining a Measure with a Dynamic Picklist

1. Select the row for the measure.

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



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 the positions in the picklist are to be sorted alphabetically by their label. If this is not selected, the positions will be shown in their internal order, which is the order in which they were defined.
   e. Click **OK** to save your changes and close the window.

> **Note:** When creating a dynamic picklist based on a dimension, you must ensure that the position labels for the source dimension do not contain opening or closing parentheses. Because dynamic picklists cannot parse label names that have parenthesis in them, using them in a label name will cause the system to encounter a run time failure when building or opening workbooks.

4. If the picklist measure is to display values based on the data values for a cell, an entry is required in the **LoadRangeMeas** property and in the **LoadRange** property fields. 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

The use of this tab is optional. It allows for the configuration of extended measures that represent different usages of the underlying base measure. The following usages are supported for extended measures:

- Creation of a measure that aggregates data using an alternate aggregation method from the default aggregation type of the measure. The aggregation types available are those configured as the Allowed Aggs of the measure.
- Creation of participation measures, such as absolute and relative percent-to-parent measures.
- Creation of a ranking measure, which assigns a rank in either ascending or descending order to the values of a measure.
- Creation of a cumulative total measure, which contains a running total of a measures values based upon an ascending or descending ranking.
- Creation of a cumulative percentage measure, which contains a running percent-to-parent contribution total based upon an ascending or descending ranking.

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.

General | Custom Menus | Workbook Hierarchies | Real Time Alerts | Measures | Extended Measures | Dynamic Position Maintenance

Measure	Label	Usage	Arguments
slccaussrcXLXB		Relative	PROD
ap06_B		Absolute	CLND_day_PROD_item_LOC_str
cchdaf01XB		Ranking	Ascending_All_PROD
sf13_B	sf13_CmValue	CumulativeValue	Descending_All_CLND
sf12_B	sf12_CmPercent	CumulativePercent	Ascending_All_LOC

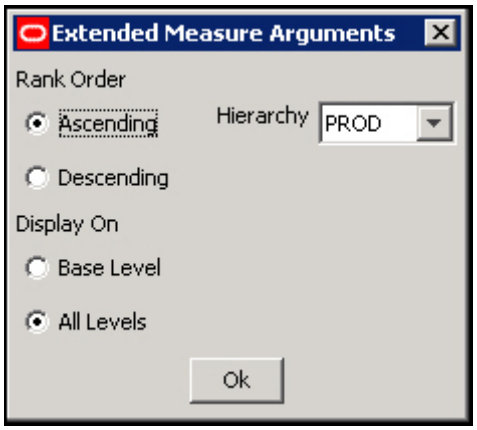
Extended Measures Tab

Adding an Extended Measure

1. To add a measure, right-click in the table area and select **Add**. The row is inserted into the tab and is highlight in red until you define the Measure, Usage, and Argument fields.
2. Click **Select Measure**. The Select Measure window opens.
3. Double-click a measure to select it.
4. For Relative, Absolute, Ranking, Cumulative value and Cumulative percent extended measures, a label can be specified. This label will be used to display the extended measure within the workbook. If no label is entered, the label of the base measure will be used
5. In the **Usage** field, select **Relative**, **Absolute**, **Ranking**, **Cumulative value**, **Cumulative percent**, or an alternate aggregation type.
6. Click in the **Arguments** field and select the appropriate options. If the **Usage** is defined as **Absolute** the Parent Intersection dialog appears, set the appropriate options from the dialog box and click **OK**. If the **Usage** is defined as **Relative**, select appropriate hierarchy from the list displayed. If the Usage is defined as **Ranking**, **Cumulative value** or **Cumulative percent**, the Extended Measure Arguments dialog appears, set the appropriate options from the dialog box and click **OK**.

The Parent Intersection Dialog

Within the Parent Intersection Dialog the dimension that will be used to determine the percent-to-parent contribution of an Absolute extended measure may be set. All dimensions above the dimension of the measure's base intersection for that hierarchy are available as well as the value AllDim, which represents all positions within the workbook.



Extended Measure Arguments

The following arguments may be set for Ranking, Cumulative value and Cumulative percent extended measures:

- **Rank Order** – Whether the ranking must be in ascending or descending order.
- **Display Type** – Whether the ranking must be performed only on the base intersection of the measure, or at all aggregate intersections also.
- **Hierarchy** – The hierarchy along which the values of the measure should be ranked or totaled.

> **Note:** Alternate aggregation extended measures do not require arguments.

Removing an Extended Measure

Right-click on the measure you want to remove and select Remove.

Dynamic Position Maintenance Tab

If dimensions are enabled to support Dynamic Position Maintenance (DPM) in the Dimensions pane within the Hierarchy Definition tool, the configuration administrator will see those dimensions in the Dynamic Position Maintenance tab.

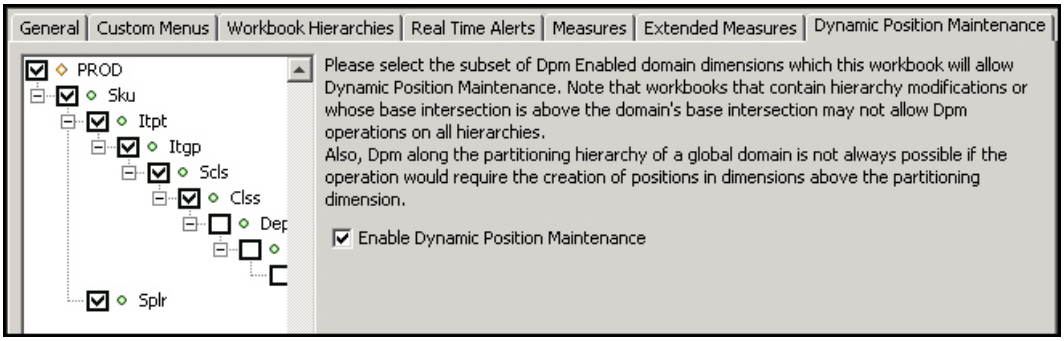
Depending on how the workbook has been configured, some dimensions that support DPM within the domain may not be available for DPM within the workbook. The following conditions will disqualify a dimension for DPM within a workbook:

- Dimensions below the base intersection of the workbook will not be available for DPM. The base intersection of a workbook is the highest intersection that can be aggregated to the intersections of all the worksheets in the workbook.
- Dimensions belonging to a hierarchy that has been modified by the Remove Dimension or Change Rollup operations within the Workbook Hierarchies panel will not support DPM.
- Dimensions more than one level above the modified dimension of a dynamic hierarchy will not support DPM.
- Dimensions that are the modified dimension of a dynamic hierarchy will not support DPM if their immediate parent dimension (where week is the parent of day) does not support DPM.
- Dimensions whose immediate parent dimension is the modified dimension of a dynamic hierarchy will not support DPM if that modified dimension does not support DPM.

See the Workbook Hierarchies section and the Dynamic Hierarchies Appendix for more information on dynamic hierarchies.

In addition, if the workbook has been configured to contain workbook-only dimensions, those workbook-only dimensions support DPM if the dimension they aggregate supports DPM. See the Workbook Hierarchies section and the Dynamic Hierarchies Appendix for more information on workbook-only dimensions.

To enable DPM functionality in the workbook, select **Enable Dynamic Position Maintenance**. The configuration administrator may then select the highest dimension in the hierarchy in which the end user will add positions. See the *Oracle Retail Predictive Application Server Administration Guide* and *Oracle Retail Predictive Application Server User Guide* for more information on Dynamic Position Maintenance.



Example of Dynamic Position Maintenance Tab

> **Note:** Users cannot import positions from alternate hierarchies that are not already in the workbook to be used as parents for new DPM positions.

Remove a Workbook

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

1. Select the workbook to remove.

2. From the toolbar, click the **Delete** button, or select **Remove** from the right-click menu.
3. Click **Yes**. The associated worksheets and tabs are removed.

Working with the Rule Group Simulator

Overview

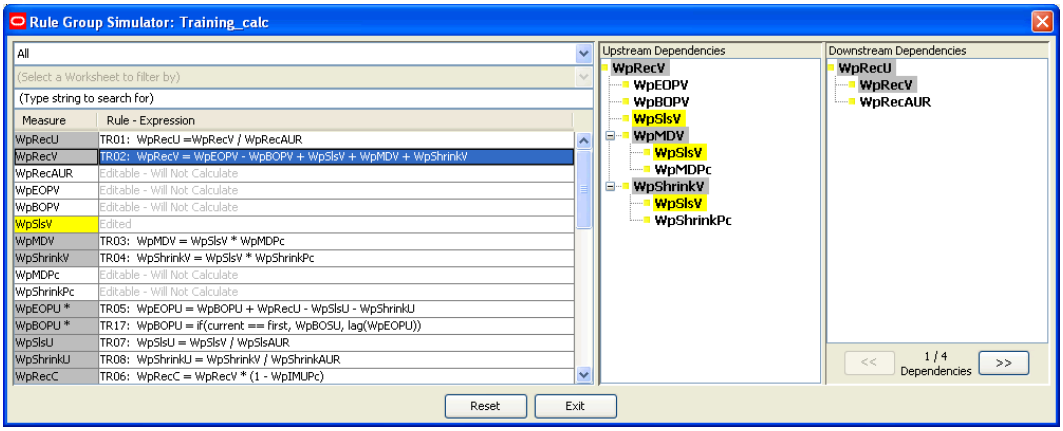
The RPAS calculation engine is powerful and complex. The rule group approach means that there are very many potential calculation paths. However, during any configuration exercise, there is a significant design verification cost to ensure that the behavior is "as would be expected" by an end user. The 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 view with Upstream and Downstream Dependencies panes.



Rule Group Simulator Window

The measure table displays all of the measures in the scope of the simulation. The **Measure** column displays the measure name. The measure status is reflected by color coding. A tooltip also displays the measure status when the mouse is placed over the measure name. All measures can be shown, or the list of measures can be filtered.

Editable measures can have their status toggled (to or from **Edited**), and the simulator immediately updates all statuses, calculations, and trees. The table below explains the meaning of the color coding used in the Rule Group Simulator window.

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 can be edited (shades of blue = no, white/gray/yellow = yes)

Whether or not the measure will be calculated.

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

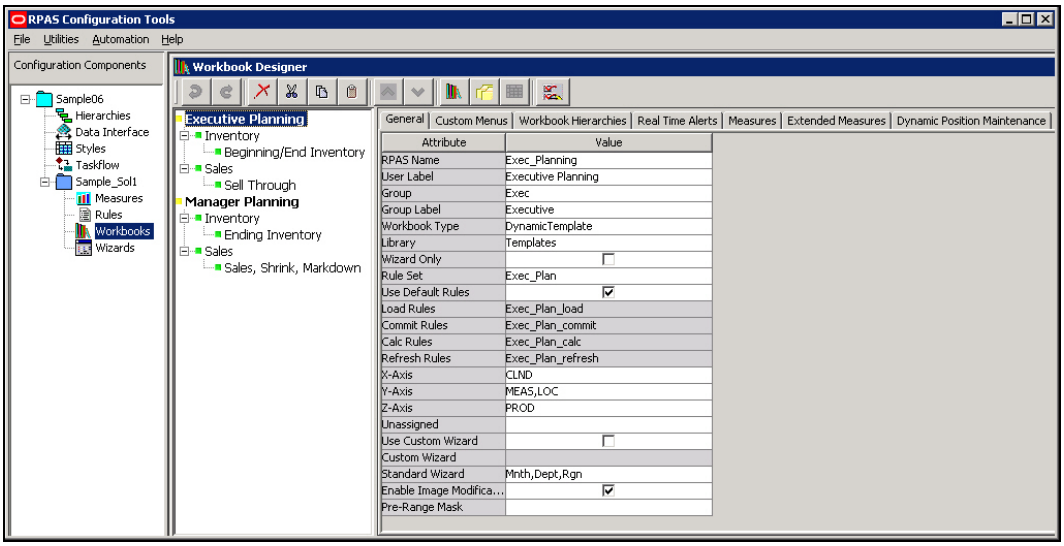
The Rule - Expression column of the table shows the calculation for each measure. For those measures that would be calculated if the end user issued a "calculate" with the current collection of edited measures, the rule and expression that would be used to calculate the measure is shown. For non-calculated measures, this column displays the measure status.

The tree view 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, 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, each of the expressions can be viewed using the forward (>>) and backward (<<) arrows.

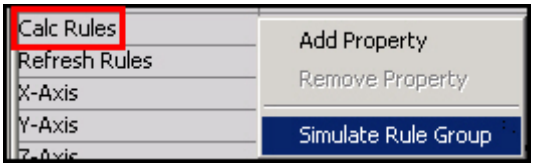
Invoking the Rule Group Simulator

Navigate: In the Configuration Components pane, 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**.



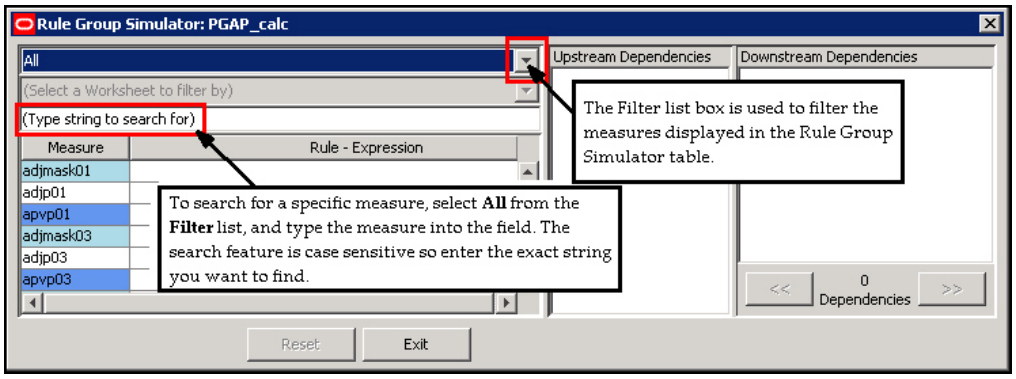
Simulate Rule Group Menu Option

The Rule Group Simulator window appears.

Filtering the Measures Table

Perform the following to filter measures displayed in the Rule Group Simulator:

1. Open the Rule Group Simulator. See "Invoking Rule Group Simulator."



Rule Group Simulator Showing Filter and Search Features

2. To filter the measures table, select the one of the following options from the **Filter** list:
   - Select **All** to display all measures in the workbook.
   - Select **Will Calculate (According to Calculation Order)** to display only those measures that will be calculated. The sequence of the measures displayed is the sequence in which they will be calculated. Measures that are edited are not shown.
   - Select **Will Not Calculate** to display only those measures that will not be calculated will be shown. Measures that are edited are not shown.
   - Select **Read Only** to display only those measures whose status is read-only (that is, have a Base State and Agg State of "read").
   - Select **Contains String** to type a case-sensitive string to filter by in the text box below the **Filter** list option. Only measures that include the string entered are displayed.
   - Select **By Worksheet** to select a worksheet from the current workbook using the list option below the filter list option. Only the default measures from that worksheet are shown. The **Worksheet** list option is disabled until **By Worksheet** is selected from the **Filter** list.
3. If searching for a specific measure, set the filter to **All**, and enter 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 can be edited (colors gray, white, and yellow) may have their status changed.

1. Select 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).

   After any change in status, the simulator updates all necessary statuses, calculations, and tree views.
2. 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 Dependencies panes.

Using the Upstream and Downstream Panes

1. To change the measure with focus for the upstream and downstream panes:
   a. With the **Filter** list option set to **All**, enter a search string in the field under the **Filter** list option. The first measure that contains the search string will get focus.

      Note: Remember, when searching by measures, the text entered in the search text field is case-sensitive.

   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 Dependencies panes.

   When the focus changes, the tree panes are refreshed as appropriate based on the measure which currently has focus, and the measure table scrolls so the measure with focus is shown.

   The measure with focus always appears at the top of the Upstream Dependencies pane. If it will be calculated, the Upstream pane shows the measures that it is dependent upon (calculated from, directly and indirectly). This is displayed using a parent-child tree structure with the measures used to calculate an individual measure showing as "children" of it. If the children are also calculated, they appear with their dependent measures, and so on. Therefore, the expanded Upstream Dependencies tree view displays all of the measure relationships that affect the measure with focus.

   The Downstream Dependencies pane shows measures that are dependent upon (calculated from) the measure with focus, if there are any. Measure relationships (expressions) appear 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 displayed. 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 Dependencies 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 back (**<<**) or forward (**>>**) buttons at the under the **Downstream Dependencies** pane.

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 "multiple 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 Dependencies panes as MeasA [+MeasB][+MeasC] where MeasA is the measure with focus and MeasB and MeasC are the other multiple result measures.

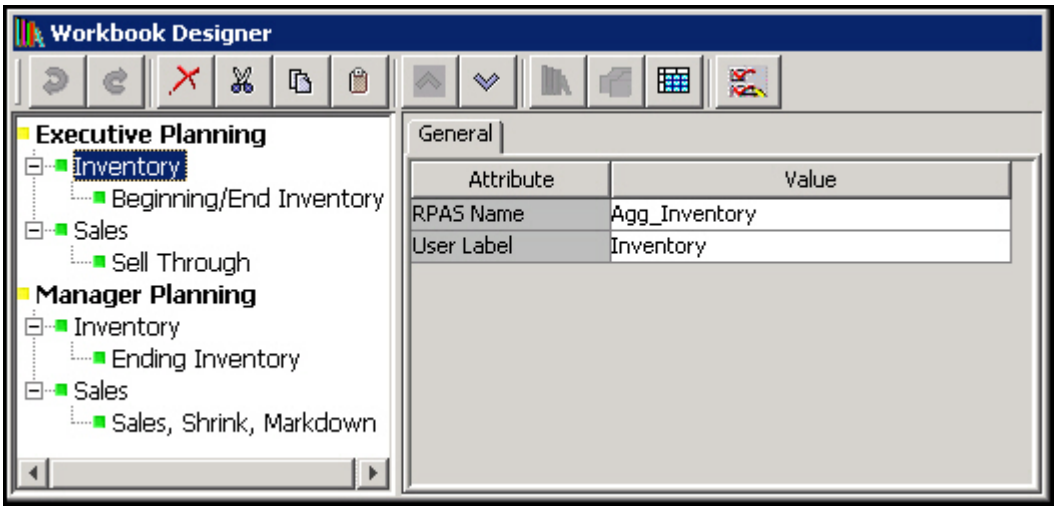
Exiting the Rule Group Simulator

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

Working with Workbook Tabs

Overview

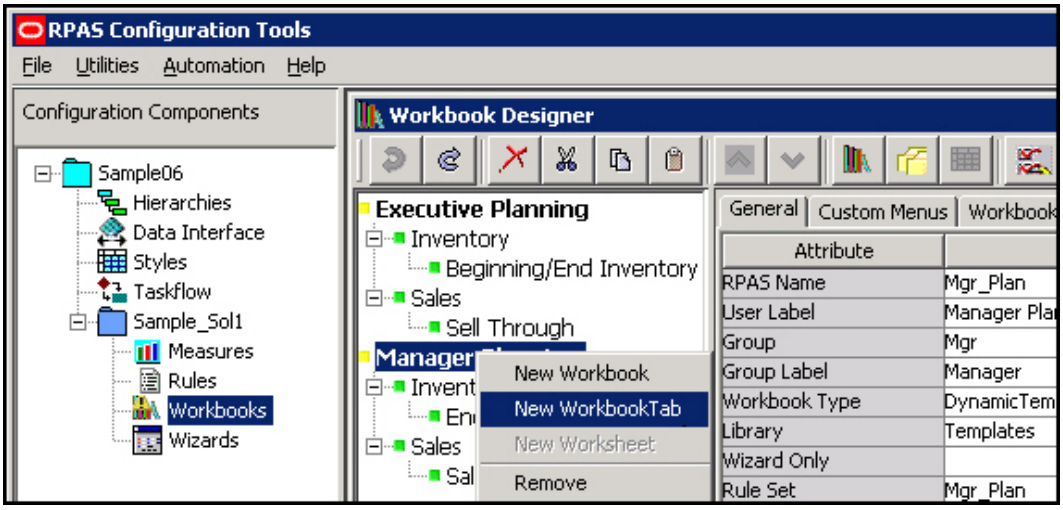
Workbook tabs are a feature in the RPAS Client that enables the workflow to be separated into steps or business processes. Setting up tabs described here is for the Classic Client. The Fusion Client can have tabs, but those are set up using the taskflow. Each workbook must have at least one tab. Users select the appropriate tab to use depending on the stage they have reached in the business process. A tab may contain one or more worksheets that allow the users 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.



Workbook Designer Dialog Box

Create a Workbook Tab

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



Example of Workbook Design Window (Create New Workbook Tab)

1. Select the workbook in which to create a new workbook tab.
2. Choose one of the following methods:
   - 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 Component pane, select **Project** – **Solution** – **Workbooks**. The Workbook Designer window opens in the workspace.

1. Select the workbook tab whose properties are to be edited.
2. In the **General** tab, type the **RPAS Name** and the **User Label**.
3. Perform one of the following to alter the order in which the tab is displayed in the RPAS Client:
   - 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 Component pane, select **Project** – **Solution** – **Workbooks**. The Workbook Designer window opens in the workspace.

1. Select the workbook tab to remove.
2. Choose one of the following methods:
   - From the toolbar, click the **Delete** button.

- Select **Remove** from the right-click menu.
- Press **Delete**.

3. Click **Yes**.

The workbook tab and associated worksheets are removed.

Comprehensive Workbook Validation

In addition to the real-time validation of user inputs, the Workbook Designer has the capability to perform a comprehensive validation of all Workbook content in the solution. Upon performing comprehensive validation, the Task List will be updated with any validation issues present in the Workbooks of the Configuration.

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

1. From the toolbar, click the **Comprehensive Validation** button.
2. All Workbook content for the solution will be validated and any issues detected will be entered into the Task List.

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. Using the Configuration Tools, you define the measures on a worksheet, the base intersection the worksheet uses, and the orientation of the hierarchies on the worksheet. The workbook measures can be organized in following categories for a worksheet:

- Selected profile
- Viewable profile
- Hidden
- Extended

Selected Profile Measures

The selected profile contains the list of measures that are initially displayed for this profile in the RPAS Client. All worksheets have one profile that is marked as the default profile. This default profile is the profile that appears when a workbook is initially built. There must be at least one measure on each of the selectable profiles.

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.

You can add extended measures that rank the values of a measure in either ascending or descending order or that contain a running total of a measure's values based on an ascending or descending ranking.

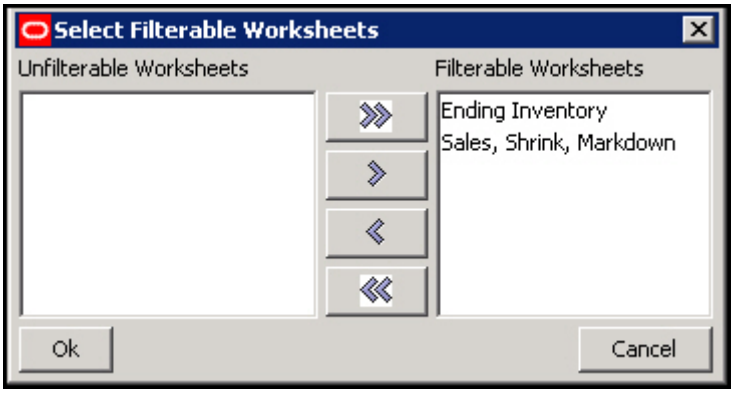
See the Extended Measures section for more information on extended measure types.

Worksheet Types

There are two types of worksheets supported. The **Pivot/Chart** type worksheet type is supported for both Classic Client and Fusion Client. Fusion Client also supports the **Detail Popup** worksheet type. By setting the Worksheet Type to "Detail Popup", you can change the type of view used for a worksheet to a Detail Pop-up worksheet.

Filterable Sheets

The Fusion Client provides the capability to filter the visible positions of a worksheet based upon the selection context of a different worksheet. By default, this filtering is enabled for all worksheets. It is possible to exclude worksheets from the list of filterable worksheets. Selecting this attribute in the table will launch the "Select Filterable Worksheets" dialog.



Select Filterable Worksheets Dialog Box

The Select Filterable Worksheets dialog contains the list of all candidates for filtering from the worksheet being edited. By default all worksheets will be in the Filterable Worksheet column. By selecting one or more worksheets and moving them to the Unfilterable Worksheets column, those worksheets will not be able to be filtered from the worksheet being edited.

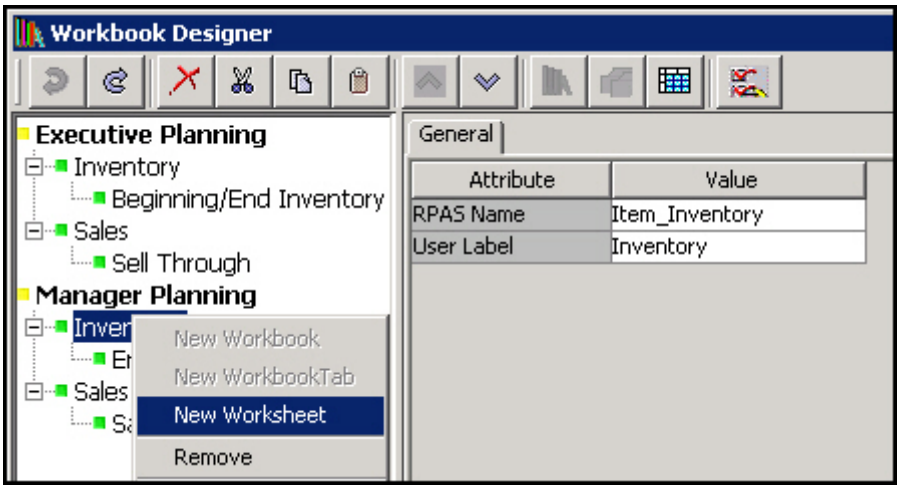
- **Auto PQD** – If a worksheet is configured to be Auto PQD, then Position Query Definitions defined for the worksheet will be automatically applied within the

Fusion Client. This removes the need for the user of the client to enable the Position Query for the worksheet when building a workbook.

- **Lock PQD Dimensions** – If a worksheet is configured to Lock PQD Dimensions, then the user of the Fusion Client will not be able to modify the axis layout of the worksheet through drag and drop of hierarchy tiles if the change to the axis layout would invalidate the Position Query. For example, if a Z-Axis Position Query that modified the LOC hierarchy were defined in the worksheet, the user would not be able to move LOC off the page edge to the X or Y axes.

Create a Worksheet

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



Example of Workbook Designer Window (Create New Worksheet)

1. Choose one of the following methods:
   - Select the workbook tab in which to create a new worksheet by clicking the **New Worksheet** button.
   - Right-click and select **New Worksheet**.
   - Select another worksheet on the same tab.
2. Press **Insert**.

   A new worksheet is created.
3. Assign the appropriate properties using the worksheet tabs (General, Position Queries, Measure Profiles, Style Overrides, Window Formatting). Refer to "Defining Worksheet Properties" for more information.

Defining Worksheet Properties for Pivot/Chart Worksheets

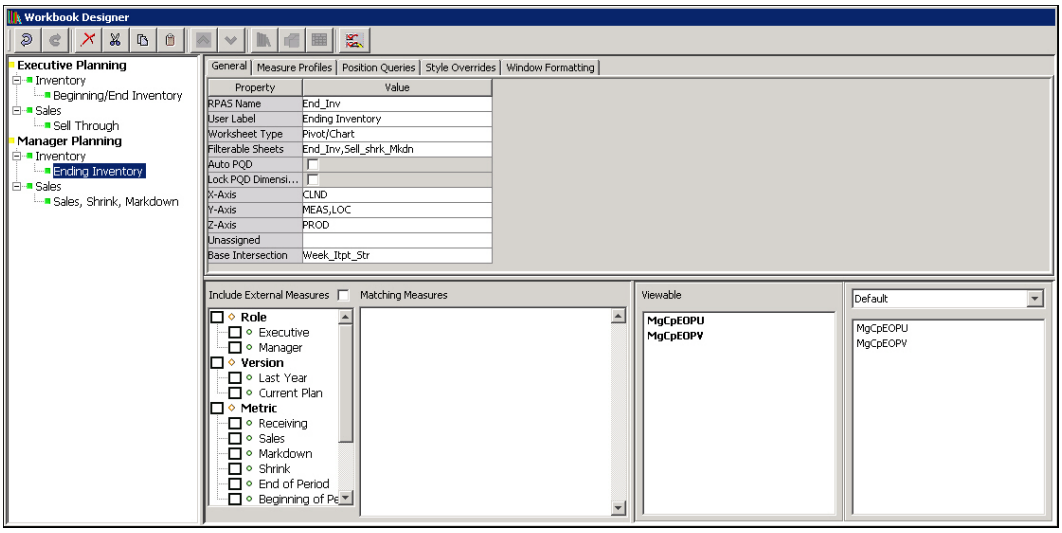
When the Pivot/Chart worksheet type is selected from the Workbook Designer window, the following tabs appear in the workspace:

- General
- Measure Profiles
- Position Queries
- Style Overrides

- Window Formatting

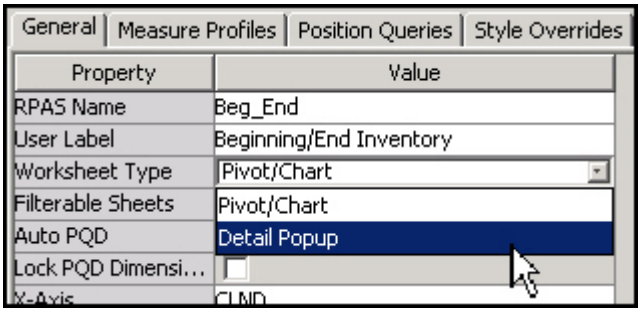
Refer to the topics below of information on using these tabs to define the worksheet properties.

General Tab



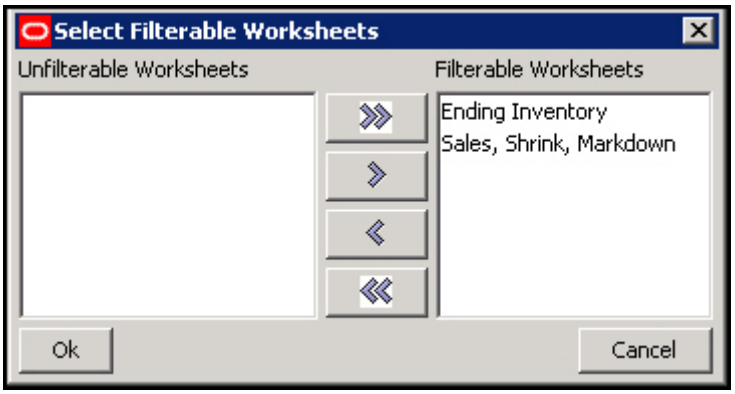
Example of General Tab for a Worksheet

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. In the **Worksheet Type** field, select Pivot/Chart.



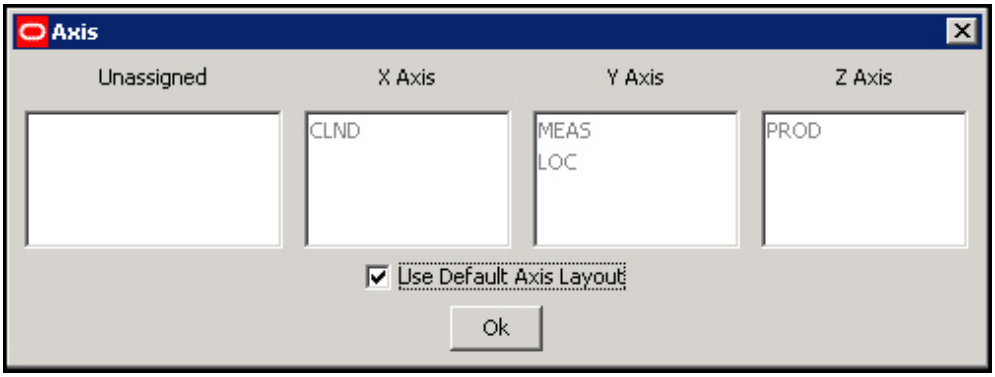
Worksheet Type Pick List

4. Determine candidates for filtering from the worksheet, if any.



Select Filterable Worksheets Dialog Box

5. Define the axis layout of the worksheet.
   a. Click the X-axis, Y-axis, Z-axis, or Unassigned field. The Axis dialog box opens.

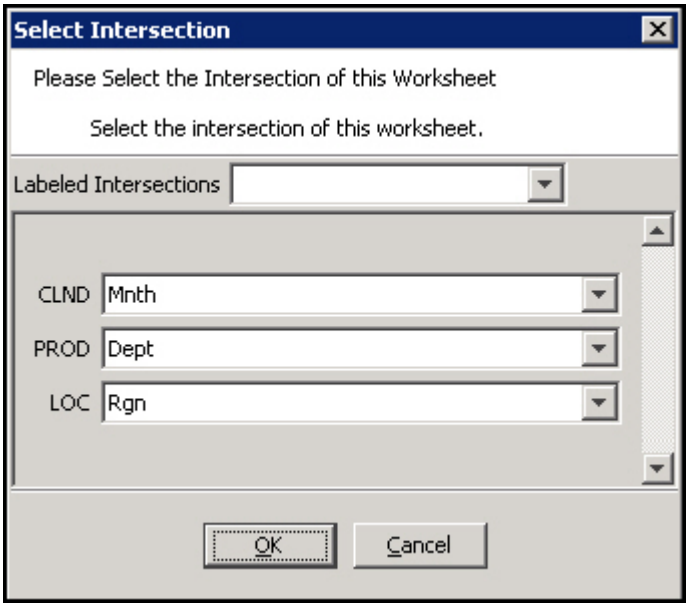


Axis Dialog Box

   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.

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



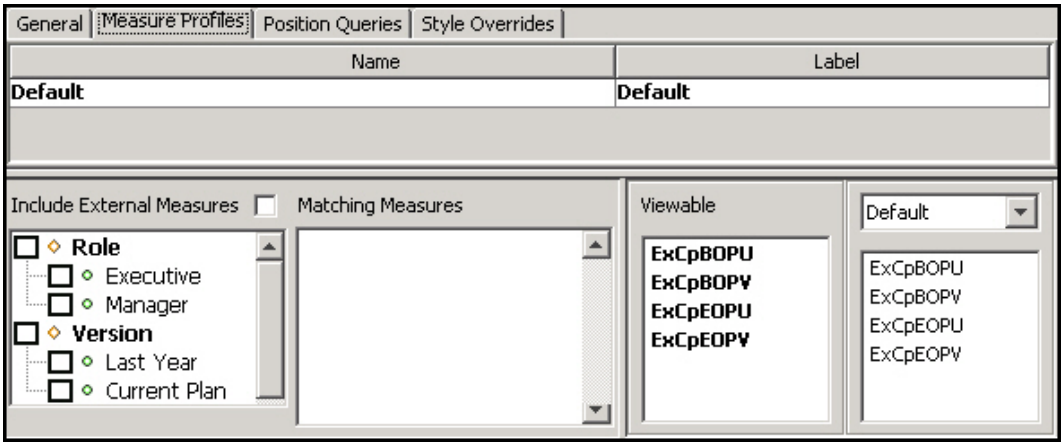
Select Intersection Dialog Box

> **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.

7. 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.
8. Click **OK**.

Measure Profiles Tab

Usage of this tab is optional. This tab allows users to configure additional measure profiles beyond the Default profile for the worksheet. These additional profiles may be populated with measures in the same manner as the default profile. When the workbook is built, these additional profiles will be available through the RPAS Client to provide users a different set of measures or a different measure ordering for the measures of the worksheet.



Measure Profiles Tab

Adding a measure profile

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

- **Name** – The name for the measure profile used by RPAS. This name has to be unique across the worksheet.
- **Label** – The label for the measure profile that is used internally by RPAS. This label has to be unique across the worksheet.

> **Note**: Duplicate label names are not allowed within the scope of the worksheet.

Copying a Measure Profile

1. Click the Measure Profiles tab.
2. Select an existing measure profile.
3. Right-click in the table area and select **Copy**.
4. A new measure profile will be created that contains the same measure membership an ordering as the selected profile.

Note: Multiple measure profiles containing the same measure membership and ordering are not allowed. As such, a measure profile created using the **Copy** operation is marked invalid until the membership or ordering is changed from the membership or ordering of the profile from which it was copied.

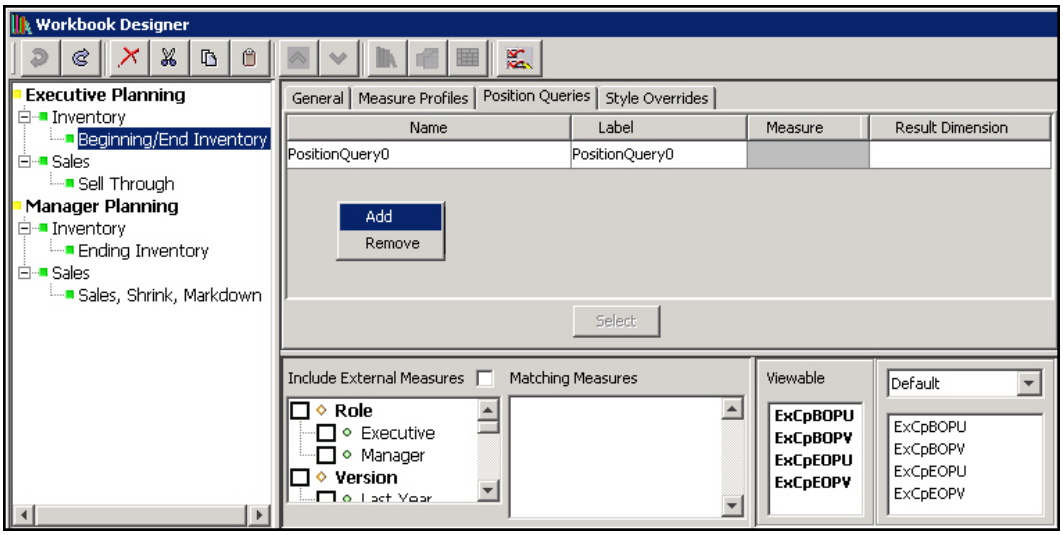
Marking a Measure Profile as Default

1. Click the Measure Profiles tab.
2. Select an existing measure profile that is not the default (the first profile in the table is always the default).
3. Right-click in the table area and select **Make Default**.
4. The selected measure profile becomes the default profile for the worksheet and moves to the first row in the measure profile table.

Removing a Measure Profile

1. Click the Measure Profiles tab.
2. Selected an existing measure profile that is not the default.
3. Right-click in the table area and select **Remove**.
4. The selected measure profile is removed from the list of profiles for the worksheet.

Position Queries Tab



Example of Position Queries Tab for a Worksheet

Usage of this tab is optional. This tab allows you to specify a worksheet where the positions that are shown in a "query" dimension are based on the current position in "driving" dimensions. The driving dimensions must be in the slice area. The process uses a position query that is a Boolean measure dimensioned on the query dimension and the driving dimensions. Only positions in the query dimension that have the value TRUE for the position query measure for the positions in the driving dimensions 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, Y, and Z axes.

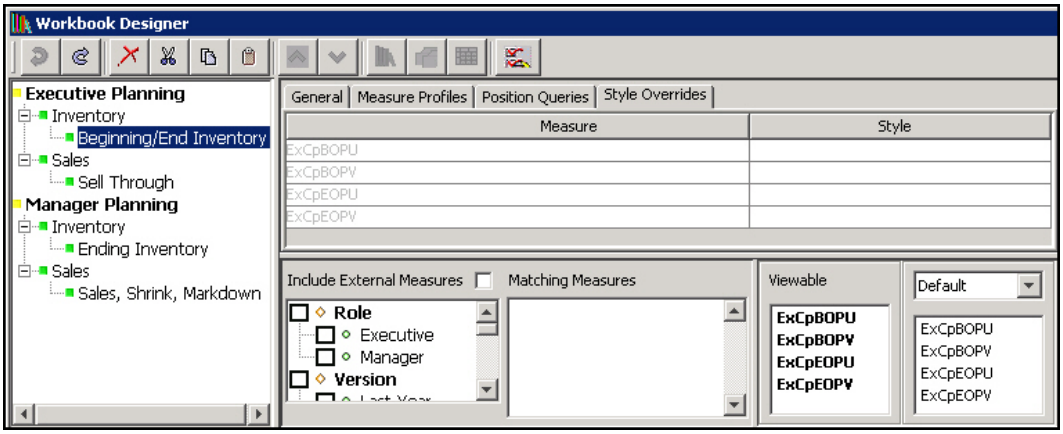
1. Click the Position Queries tab.
2. Right-click in the table area, and select **Add**.
3. Enter the following information:
   - **Name** – The name for the position query used by RPAS. This name has to be unique across the project.
   - **Label** – The label for the position query that is used internally 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. This must be a Boolean type measure. Click in the field, and then click **Select Measure** to view 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 selected measure.

     Note: While configuring position queries, it is important that the Boolean mask measure that drives the position queries be reference in the workbook (by either referencing it in the load rule group or by setting it through the calc rule groups). Currently, there is no validation in Configuration Tools that checks for this, and no error is thrown at configuration time/domain build time or workbook build time.
4. To remove a row from the tab, select the row, right-click, and select **Remove**. The row is deleted from the Position Queries tab.

Style Overrides Tab



Example of Style Overrides tab for a Pivot/Chart Worksheet

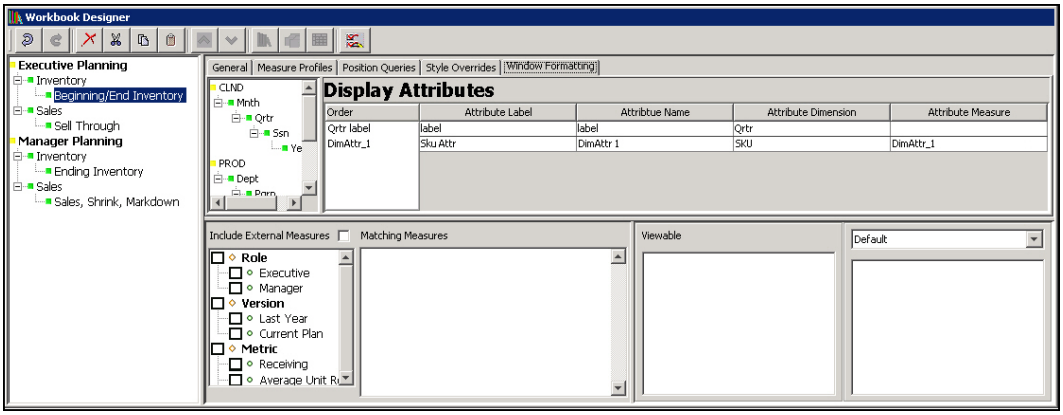
Usage of this tab is optional. This tab allows you to override the style property for a measure, so that measure uses a different formatting style in the worksheet. The Style Overrides tab can also be used to apply a formatting style to an attribute defined on any of the dimensions used in the worksheet. This formatting style will then be applied to the attribute where it appears within the worksheet.

Note: The measures that appear in this process are those placed on the profiles of the worksheet. Attributes that appear are those that are defined at or above the intersection of the worksheet. If no measures have been placed on profiles and the worksheet has not been configured with an intersection, no entries will appear in the Style Overrides tab.

1. Click the **Style Overrides** tab.
2. Select an override formatting style for a measure. Measures whose styles have been overridden appear in black. Those whose styles are defaulting appear in gray.

Window Formatting Tab

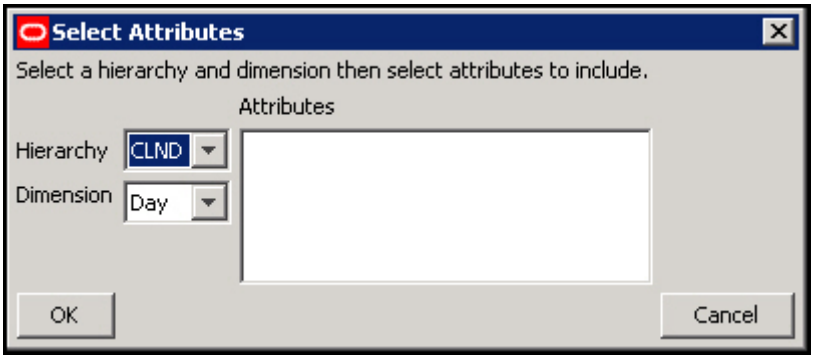
Usage of this tab is optional. This tab allows you to specify the display attributes for the dimensions used within the worksheet. By default, dimensions display the position label within the Fusion Client. If there are any additional attributes defined for a dimension, the client can display those attributes in addition or in place of position labels. Within the Fusion Client, if a media UI Type attribute is added to the set of display attributes, the Fusion Client will display the appropriate image in the position display area.



Example of Window Formatting Tab for Worksheet

Add Additional Display Attributes for a Dimension

1. Click on the Window Formatting tab.
2. Right-click and select Add from the menu.
3. Within the Select Attributes dialog, specify the dimension to modify.
4. Select desired attributes from the list of available attributes for the dimension.
5. Click **OK**.



Select Attributes Dialog Box

Modify Order of Existing Attributes

1. Click the Window Formatting tab.
2. Select the dimension to modify from the hierarchy tree.
3. Within the Order column, use drag-and-drop to modify attribute order.

Remove a Display Attribute

1. Click the Window Formatting tab.
2. Select the dimension to modify from the hierarchy tree.
3. Select an attribute doe the dimension in the Order column or attribute table.
4. Right-click and select Remove from the menu.

> **Note:** All dimensions have the label attribute set by default and all dimensions must retain at least one display attribute.

Defining Worksheet Properties for Detail Popup Worksheets

Working with Detail Popup Worksheets

The Detail Pop-up is an alternate type of worksheet supported by the Fusion Client. It differs from a Pivot View in a number of aspects.

- It is not automatically realized when a workbook is built, instead Detail Pop-ups are accessed by the user of the client through a worksheet transition.
- Detail Pop-ups cannot be minimized, restored and maximized; they are modal dialogs that are created at user need and dismissed after us.
- Detail Pop-ups cannot be minimized, restored and maximized; they are modal dialogs that are created at user need and dismissed after use.
- Detail Pop-ups contain information for a single set of selections, only one position along any hierarchy in the workbook can be displayed within a Detail Pop-up.
- Detail Pop-ups are capable of displaying and allowing the user to interact with data at different dimensionalities; they do not exhibit the hashing out of measures with fewer dimensions than the worksheet is experienced in Pivot Worksheets.
- Detail Pop-ups do not support multiple measure profiles; there is only a single profile for measures and it serves as both the Viewable and Selected Profile.

Detail Pop-ups support a second profile for dimension attributes; these attributes can be defined along any hierarchy present in the Detail-popup and can include both configured dimension attributes and default RPAS attributes such as position label.

Create a Detail Pop-up Worksheet

1. Navigate to the Workbook Designer.
2. Select the Workbook Tab that will contain the new worksheet in the Workbook tree.
3. Click the New Worksheet button or press Insert to create a new worksheet.
4. Change the Worksheet Type of the new worksheet to Detail Pop-up.

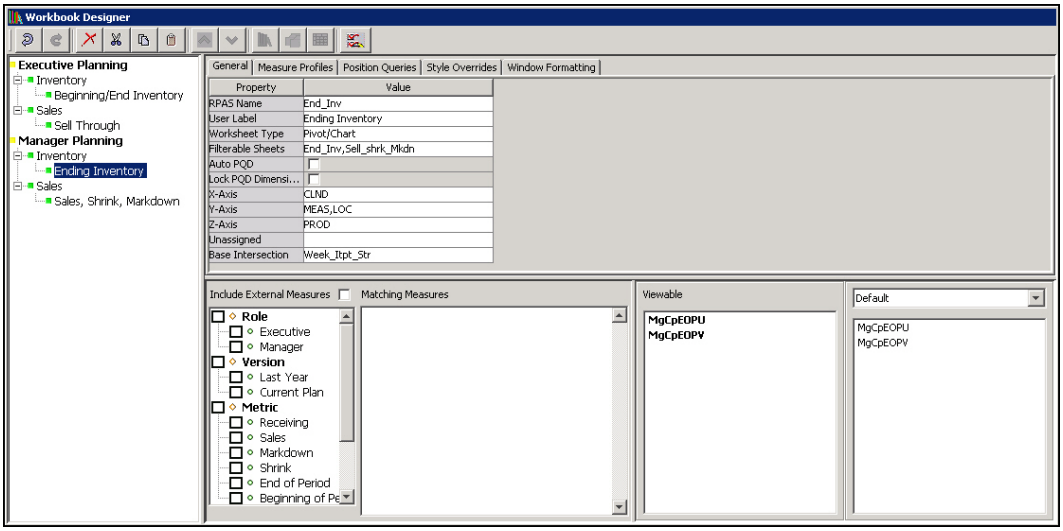
Defining Worksheet Properties

Detail Pop-up worksheets have the following tabs in the workspace:

- General
- Style Overrides
- Attribute Profile

Refer to the topics below for information on using these tabs to define the worksheet properties.

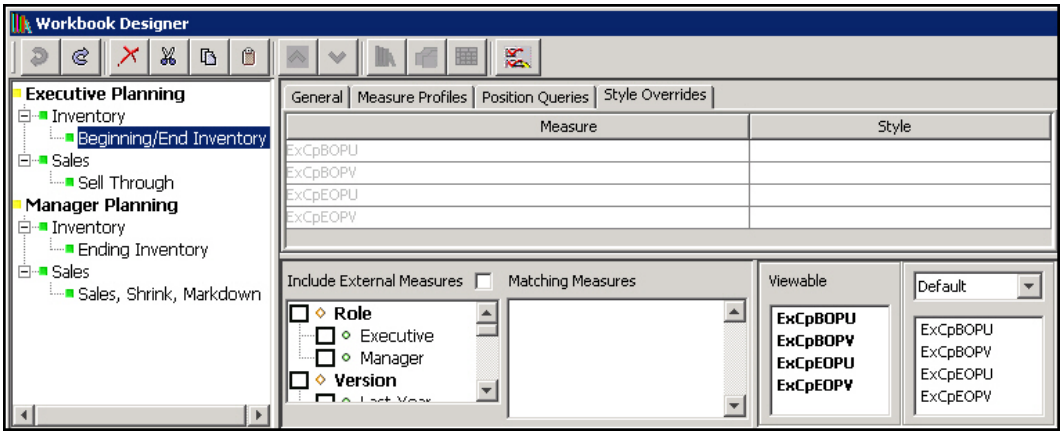
General Tab



Example of General Tab for a Detail Popup Worksheet

1. In the RPAS Name field, enter the RPAS internal name of the worksheet.
2. In the **User Label** field, enter the description of the worksheet that the user will see.
3. The **Worksheet Type** can be used to change the worksheet to another type of worksheet.
4. In the **Base Intersection** field, enter the base intersection for the worksheet.

Style Overrides Tab



Example of Style Overrides Tab for a Detail Popup Worksheet

Usage of this tab is optional. This tab allows you to override the style property for a measure, so that measure uses a different formatting style in the worksheet. The Style Overrides tab can also be used to apply a formatting style to an attribute defined on any of the dimensions used in the worksheet. This formatting style will then be applied to the attribute where it appears within the worksheet.

> **Note:** The measures that appear in this process are those placed on the profiles of the worksheet. Attributes that appear are those that are defined at or above the intersection of the worksheet. If no measures have been placed on profiles and the worksheet has not been configured with an intersection, no entries will appear in the Style Overrides tab.

1. Click the **Style Overrides** tab.

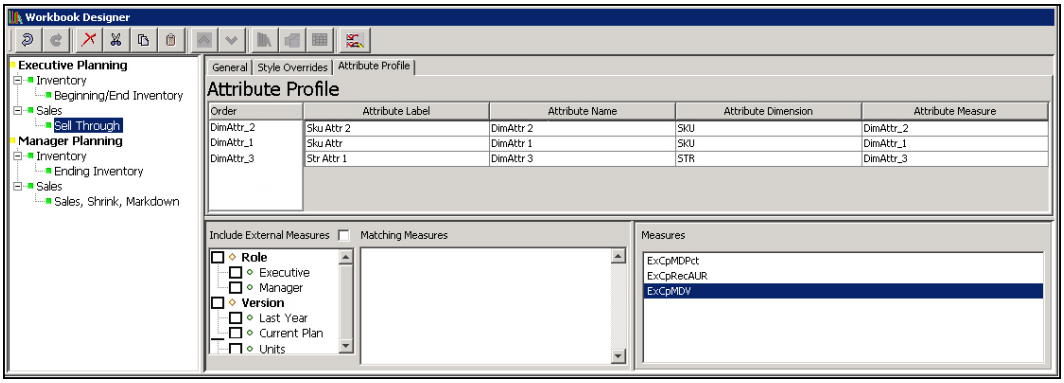
   The list of measures with their current formatting style appears.
2. Select an override formatting style for a measure. Measures whose styles have been overridden appear in black. those whose styles are defaulting appear in gray.

Attribute Profile Tab

Detail Pop-up worksheets support two profiles. The first is a measure profile and is configured as Pivot worksheet measure profiles are configured. The second profile is the attribute profile which is configured through this tab.

In addition to display of measure information, the Detail Pop-up can display dimension attributes describing the positions displayed within the worksheet. Because dimension attributes have their data loaded into a workbook by a different mechanic from the measures found in the load, calc, commit and refresh rule groups, they are not available for selection using the measure selector used for populating measure profiles.

The Attribute Profile tab can be used to select the set of attributes to incorporate in a Detail Pop-up Worksheet.

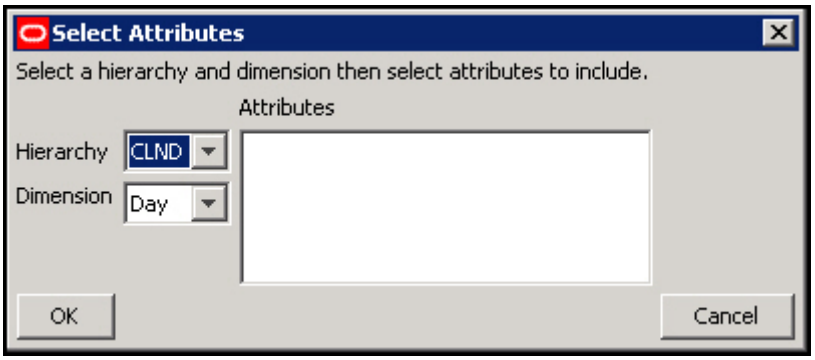


Example of the Attribute Profile Tab in the Worksheet

The Attribute Profile tab contains a table listing all attributes configured for the attribute profile of the selected worksheet. The columns in the table provide information about the name, label, modified dimension, and, if the attribute is a configured attribute, the measure configured to hold the attributes information. These properties are supplied for information purposes and cannot be edited within the Attribute Profile tab.

Adding an Attribute to the Attribute Profile

1. Click on the **Attribute Profile** tab.
2. Right-click within the Attribute Profile display and select **Add** from the menu.
3. Within the **Select Attributes** dialog, select the dimension whose attributes are desired.
4. Select one of more available attributes and click **OK**.



Select Attributes Dialog Box

Remove an Attribute from the Attribute Profile

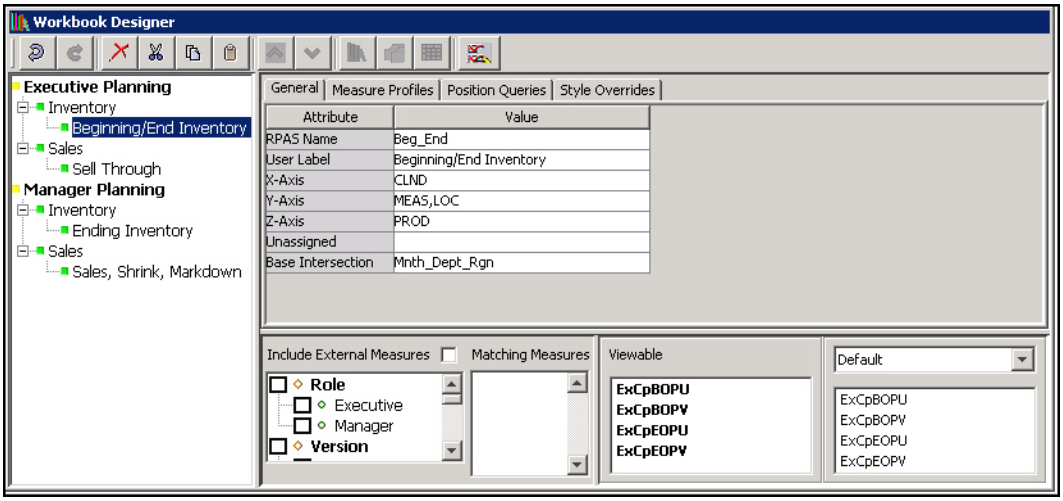
1. Navigate to the Attribute Profile tab.
2. Select an existing attribute within the Attribute Profile table.
3. Right-click within the Attribute Profile tab and select Remove from the menu.

Modify the Order of Attributes Within the Attribute Profile

1. Navigate to the Attribute Profile tab.
2. Use drag-and-drop within the Order column to modify the order of configured attributes.

Specify Which Measures Appear in a Worksheet

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



Workbook Designer Window

1. Select the worksheet to be used to specify measures.
2. Select the profile to be used to specify measures. When the user has specified multiple profiles for a worksheet using the **Measure Profiles** tab, each of these profiles may have a different set of measures defined for it. In cases where there is more than one configured profile, one profile is considered to be the **Selected** profile. It is this profile that will have its measure content modified through interaction within the **Profile Panel**. When there are multiple configured profiles for a worksheet, the profile that is currently **Selected** may be changed using the drop-down list in the upper right of the **Profile Panel**.
3. 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.
4. If External Measures are to be available for placement on the worksheet, select the **Include External Measures** check box.
5. To add measures from the matching measures column to the **Viewable** or **Selected** columns, perform one of the following options:
   - Select the measures to add from the **Matching Measures** column. Drag the measures to the **Viewable** or **Selected** column.
   - Select the measures to add from the **Matching Measures** column and press **Ctrl+C**, and then click in the **Viewable** or **Selected** 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 **Selected** column.
   - To add ALL measures from the **Matching Measures** column, right-click in the **Viewable** or **Selected** column and select **Add Matching** from the menu.
6. To add measures from the **Viewable** column to the **Selected** column, perform one of the following options:

- Select measures in the **Viewable** column, and drag the measures to the **Selected** column.
- Select measures in the **Viewable** column and press **Ctrl+C**, and then click in the **Selected** column and press **Ctrl+V**.
- Right-click in the **Viewable** column, and select **Copy**. Right-click in the **Selected** column, and select **Paste**.

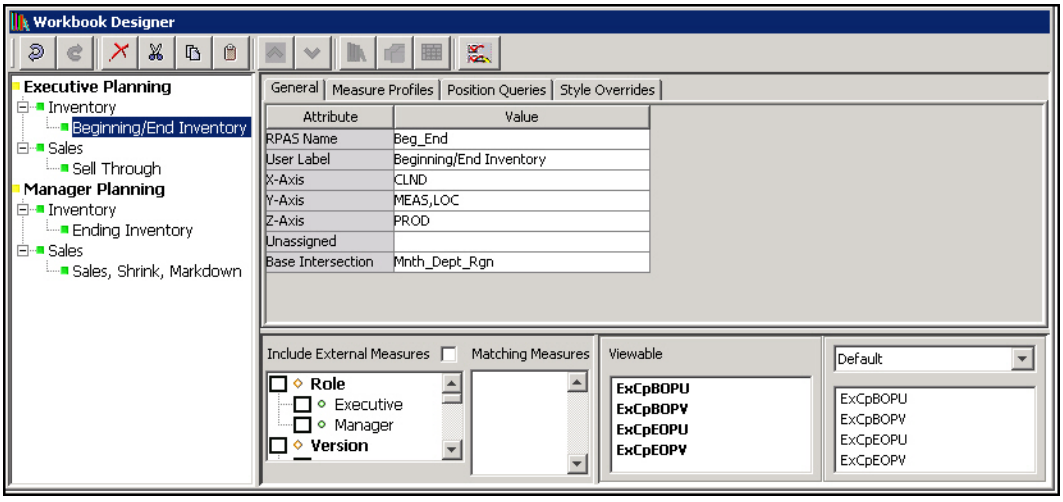
> **Note:** Adding a measure to the **Selected** column also adds it to the **Viewable** column if it is not already in the **Viewable** column.

7. To remove measures from the **Viewable** or **Selected** 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:** Removing a measure from the **Viewable** column also removes it from all profiles in the **Selected** column.

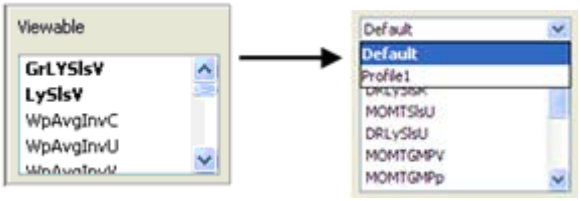
Specify the Sequence of Measures on a Worksheet

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



Workbook Designer Window

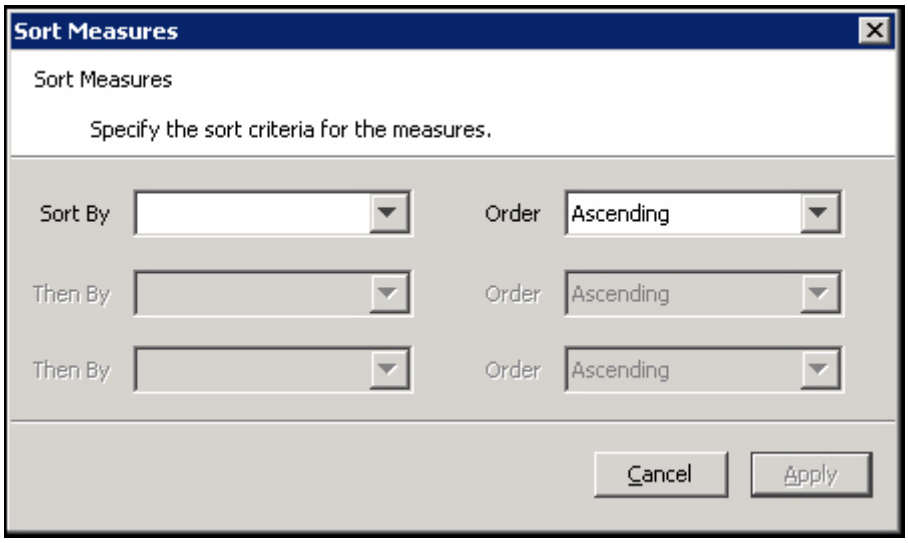
1. Select the worksheet that will be used to sequence measures.
2. To sequence measures manually, drag and drop the measures from the **Viewable** column to the **Selected** column.



Viewable Column to Selected Column

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

> **Note:** Sorting measures are based on the internal component name (not the label).



Sort Measures Dialog Box

   b. Measures can be sorted based on their major components. Select the sort sequence in which the major components are to be applied, and whether the sort should be ascending or descending
   c. Click **Apply**.

The measures are sorted into the specified sequence. This is a "one time" sort. If new measures are added to the Selected column, or measures are manually sequenced, the sort sequence previously specified will no longer apply. You would need to resort the measures again.

Edit Worksheet Properties

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

1. Click the worksheet 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

Perform the following procedure to remove a worksheet:

1. Select the worksheet to remove.
2. From the toolbar, click the **Delete** button, or right-click and select **Remove**.
3. Click **Yes**.

Wizards

Overview

This section describes the tasks you can perform by using the wizard designer. The wizard designer supports 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:

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 designer is responsible for:

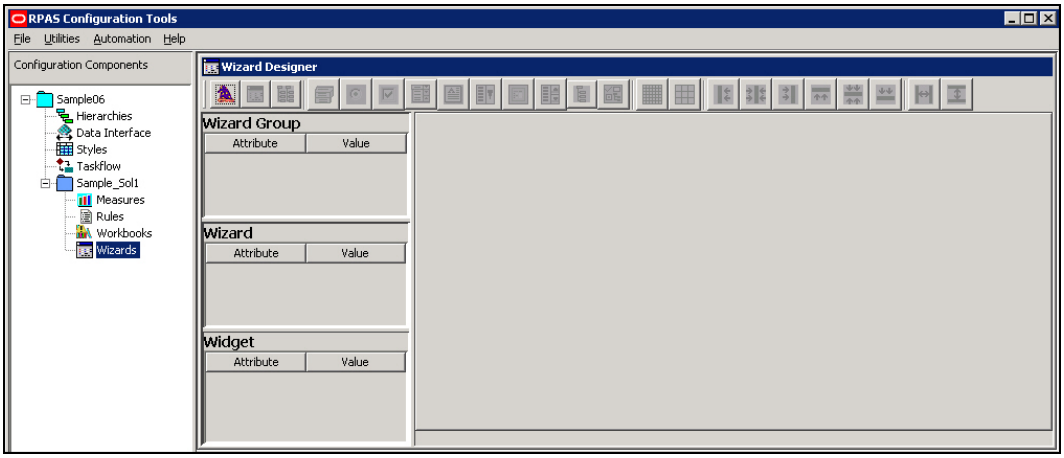
- Laying out the wizards.
- Writing code to control the transition between the wizards and out of the wizards.
- Writing code to initiate the building of the workbook.

The Wizard Tool only helps the designer in the first aspect of this work. The 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 Components pane, 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:

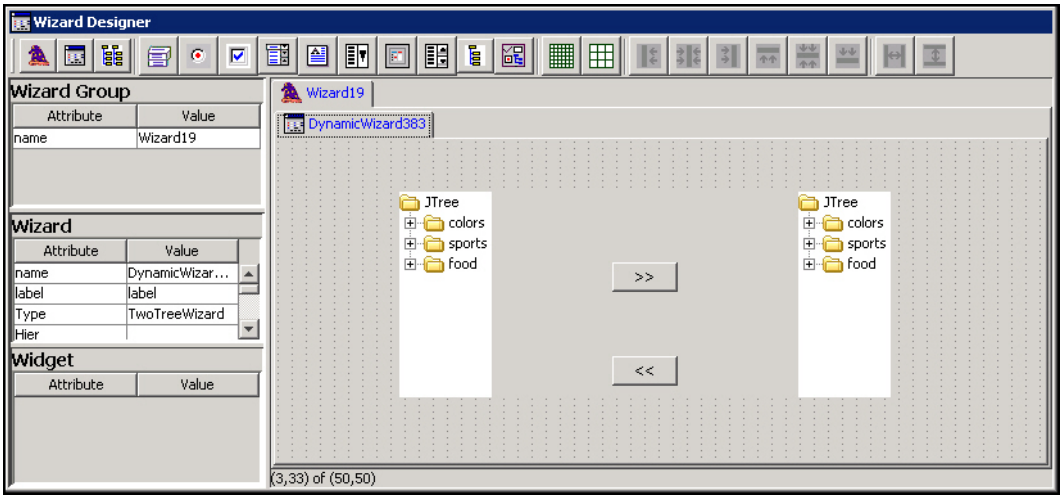
Button	Function
	Supports the creation of a new dynamic wizard.
	Supports the creation of a two tree page.
	Supports the creation of a label field that cannot be edited in the resulting wizard.
	Supports the creation of a radio button field from which the user can make one choice from several options.

Button	Function
	Supports the creation of a checkbox field from which the user can make multiple choices from multiple options.
	Allows the insertion of a drop-down list box from which the user can select the entered choices. The selection will be displayed in the text box.
	Allows the insertion of a field in which the user can enter free-form text.
	Allows the insertion of a list box that contains a list of items from which the user can select.
	Supports the creation of a labeled area in the wizard where other wizard elements can be grouped.
	Allows the insertion of a date picker (spinner) that contains independently scrollable date components.
	Allows the insertion of a single tree into the wizard.
	Allows the insertion of 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.
   - **Time_Format** – The format of the time. There are two valid inputs for this attribute: 12 and 24. Use 12 if you want to use a 12-hour clock (for example, the time appears like 10:58PM). Use 24 for a 24-hour clock (for example, the time appears like 22:58).
   - **Align** – The same as the style attribute. Valid values are left, right, center, multiline, and flip depending on the type of widget being created.
   - **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 the control is moved using your mouse.
   - **Locy** – The y coordinate of the control on the wizard page. The value of this field is automatically changed when the control is moved using your mouse.
   - **Width** – The width of the control in pixels.
   - **Height** – The height of the control in pixels.

Edit Wizard Control Properties

Navigate: In the Configuration Components pane, 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 to edit.
2. Select the wizard tab that contains the widget to edit.
3. Select the widget.
4. In the **Widget** area, update the information as necessary.

5

System Preferences

Overview

General preferences can be set for the Configuration Tools at the workbench level, which refers to the entire tool. This includes the domain type and general preference settings.

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.
- The end user 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 the Fusion Client, position level security is used to guide users into the domains in which they have some access. 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.

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.

Solution ID

A taskflow that includes multiple solutions creates the possibility for information in the taskflow of one configuration to collide with the information in the taskflow of another. For example, two different configurations could each contain an activity named Activity1. Because it is not possible at solution configuration time to know the full set of solutions that might be combined using a multi-solution taskflow, it is not possible to prevent such conflicts through configuration validation.

In order to prevent collisions between the taskflow information of multiple configurations, the solution ID is used. It is a property of the configuration. Each configuration can have a different Solution ID which defaults to the name of the configuration.

When generating taskflow resources, the solution ID for the configuration is used as a prefix to all keys. In addition, every task generated out of the Configuration Tools as part of the taskflow resources used by the Fusion Client has a solution ID attribute used to identify to which solution the task is applicable. Note that multi-solution taskflow

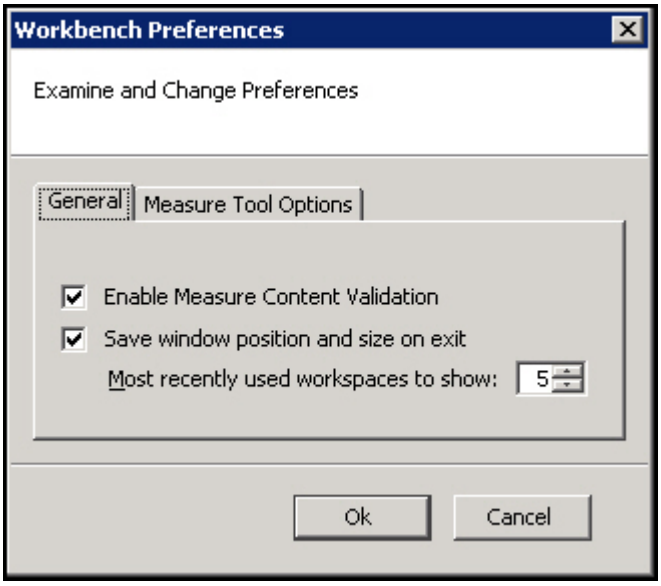
implementations that contain multiple instances of a solution are required to specify unique Solution IDs for each instance of that solution to prevent collisions.

For users upgrading to 14.0, the default value of solution name is used for the solution ID so that no additional configuration work is required for users not implementing a multi-solution taskflow.

Setting Workbench Preferences

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

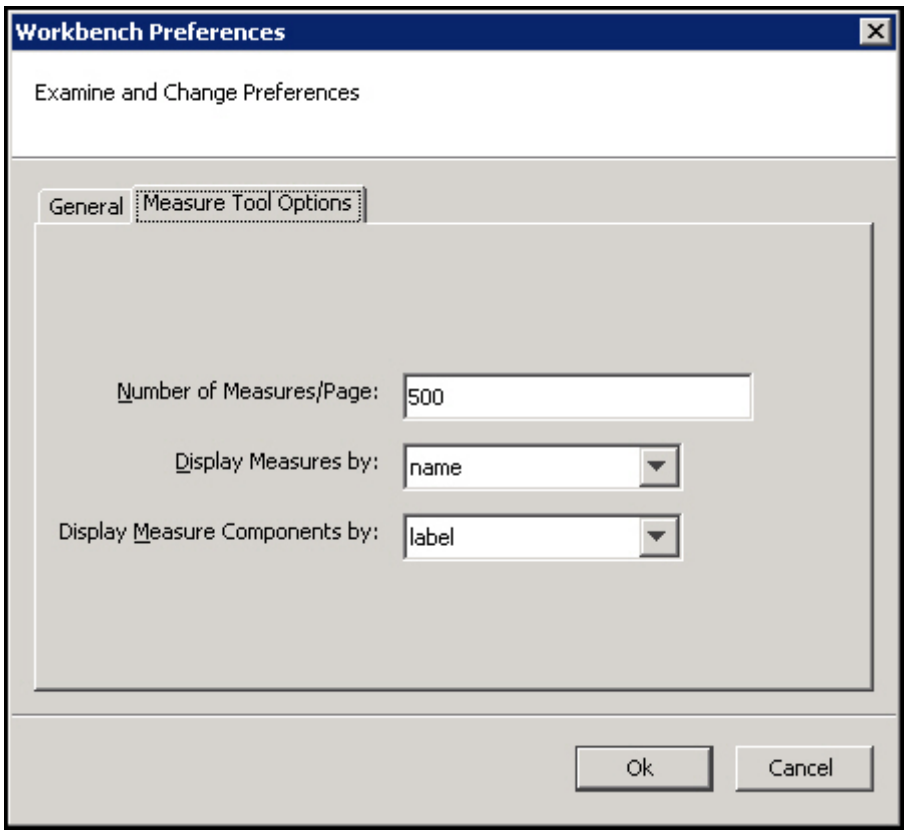
1. Select the **General** tab and select the appropriate options.



Workbench Preferences Dialog – General Tab

- The main file menu lists configurations that were recently opened. The number of configurations is displayed in this menu. To set the number of configurations, set the "Most recently used workspaces to show" field by using the up and down arrows.
- Select the **Measure Content Validation** check box to enable measure content validation. Deselect the check box to disable measure content validation. A change to the properties of a measure can affect the validity of a large number of components both within the Rule Tool and the Workbook Tool. Whenever a measure editing session is completed (for instance, upon exiting the Measure Manager and entering a different tool), the workbench will evaluate the effects of the edits made upon the measures. This process can be time consuming. When the Measure Content Validation option is unchecked, the automatic validation of measure property edits is disabled. This allows for rapid transitioning between tools when working on a large configuration, because it will not be necessary to await the completion of the automatic validity checking. When automatic validation is disabled, a manual check of measure validity can be enabled from the Rule Tool. This allows you to manually update the measure content validation of the Rule and Workbook Tools (see Measure Validation within the Measure Manager for more information).

2. Select the **Measure Manager Options** tab and adjust the fields as needed.



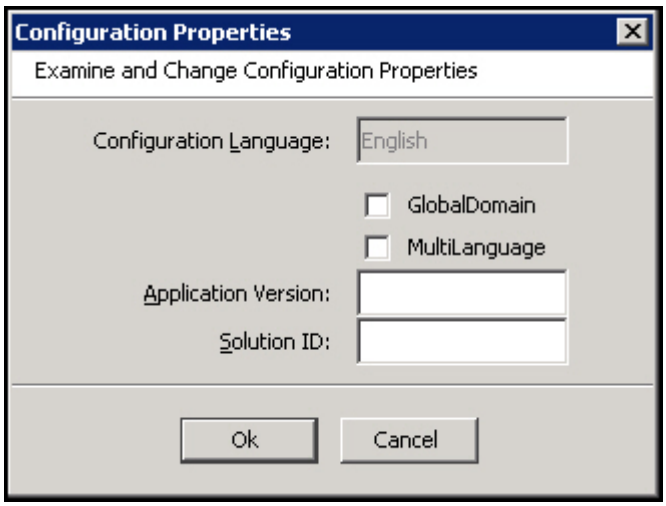
Workbench Preferences Dialog – Measure Tool Options Tab

- **Number of Measures/Page** – The Measure Manager display only a certain number of measures per page in the Measure tab. This option determines the number of measures that appear on a page.
- **Display Measures by** – This list provides two options, **name** and **label**, and determines whether measures are displayed using their name or label in the Configuration Tools. For example, in the Workbook Tool when selecting viewable measures, the name or the label will be displayed depending on the selection here.
- **Display Measure components by** – This list provides two options, **name** and **label**. Your selection determines how measure components in the Measure Manager are displayed.

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

Setting Configuration Properties

Navigate: From the File menu, select **Configuration Properties**. The Configuration Properties dialog box opens.



Configuration Properties Dialog Box

The **Configuration Language** field is disabled by default. English is displayed in the field because currently configurations can only be in English.

1. Select the following options as necessary:
   - **Global Domain**– Select this option if the configuration uses a global domain environment. This enables the creation of workbooks in multiple domains and to administer and update multiple domains from a single master domain.
   - **MultiLanguage** – Select this option if the configuration supports multiple languages in the domain.
2. Enter the following properties as necessary:
   - **Application Version** – Enter the application version in the format ApplicationName:ApplicationVersion.
   - **Solution ID** – Enter a unique identifier for the configuration. The default is the name of the configuration. The solution ID is used to help integrate multiple domains using a combined taskflow.

   Note: See the Taskflow Configuration section for more information on multi-solution taskflow configuration.
3. Click **OK** to save any changes, and close the window.

6

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

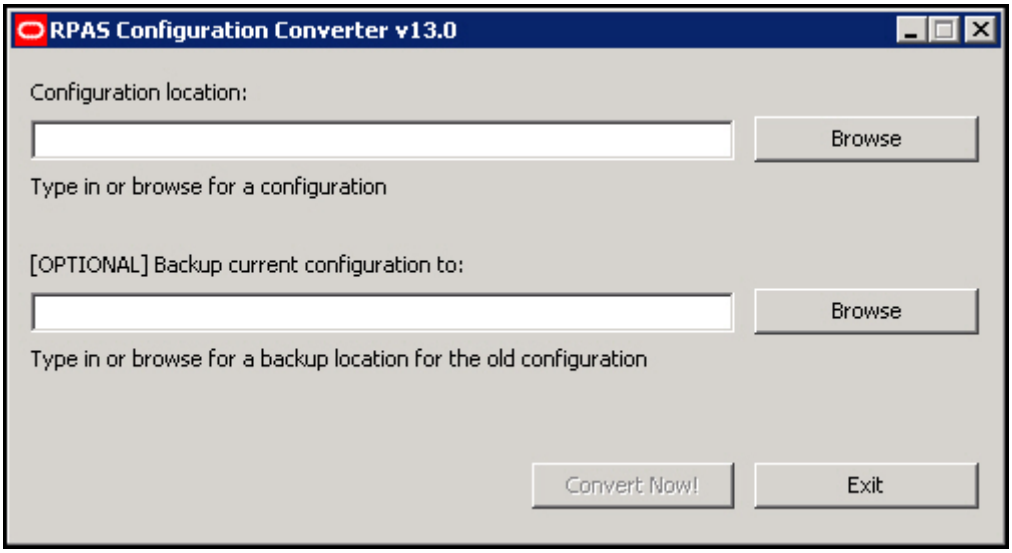
Note: The functionality for converting a configuration is provided directly through the Configuration Tools. See the section, "Open an Existing Project from an Older Version of the Configuration Tools" in Chapter 3.

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 Configuration Converter

The Configuration Converter can be accessed in three ways:

1. From the Utilities menu in the Configuration Tools, select **Configuration Converter**. The Configuration Converter window appears.



RPAS Configuration Converter Window

2. From the Windows Start menu, select **Oracle** – **RPAS** – **Utilities**, and select **Configuration Converter**. The Configuration Converter window appears. If this shortcut does not appear, refer to the *Oracle Retail Predictive Application Server Installation Guide* for information about creating it.
3. Go to a command prompt. Run the RpasConverter.exe file in the \utilities directory where the Configuration Tools were installed and run the following command:

   `RpasConverter -c C:\PathToConfig\Config\Config.xml [OPTIONS]`

   The following options can be used from the command line:

   `-b BackupDir`

   Use this argument to create a backup of the original configuration in *BackupDir* location specified.

   `-g`

   Use this argument to open the RPAS Configuration Converter screen shown above.

   `-h`

   Use this argument to display usage information.

Converting a Configuration

1. In **Configuration location** field, enter path and configuration file name to be converted. This is the file in the configuration's directory that has the configuration name with an ".xml" extension. You may also click the **Browse** button to navigate and select the appropriate file. Make sure to provide the file extension in the Configuration location field.

   Example of Configuration location field entry:

   C:\Configs\MyConfig\MyConfig.xml
2. Optional: In the **Backup current configuration to** field, enter a directory where a copy of the original configuration will be stored. You may also click the **Browse** button and to navigate and select a directory.

   Note: The directory entered must not already contain a directory whose name is the name of the original configuration. For example, to put a backup of a configuration named "MyConfig" in a directory "C:\Backups," "C:\Backups\MyConfig" must not exist.
3. Using the **Convert to version** list, select the version to convert to. This should always be the current version of the Configuration Tools unless there is a good reason to convert to some older version.
4. Click Convert Now.
5. 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.

   Note: See the *Oracle Retail Predictive Application Server Administration Guide* and *Oracle Retail Predictive Application Server Installation Guide* for more information on the domain installation and upgrade process.

Functional Library Manager

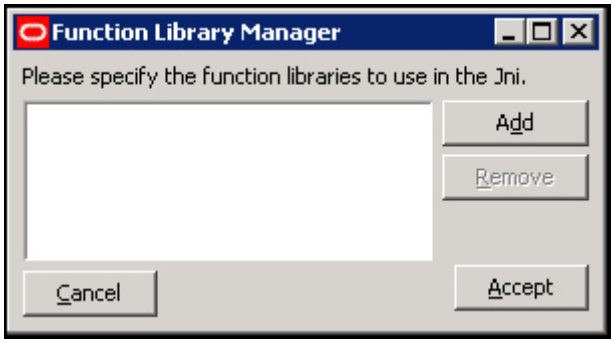
Overview

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 provide validation for custom functions or procedures within the Configuration Tools. The custom functions or procedures must exist in the /applib directory of the RPAS_HOME directory. If necessary, this utility can also be used to remove custom function libraries from being validated. There is no validation for the existence of the function libraries in RPAS_HOME/applib directory. When a function library is removed using the Function Library Manager, it is removed only from the list of external libraries used for validation, and the contents of RPAS_HOME/applib directory are left intact. The function libraries mentioned in this list are loaded by the Configuration Tools and will be used to perform rule validation.

Speak to an Oracle Retail Services representative for additional information about custom functions and procedures.

Launching the Functional Library Manager

Navigate: From the **Utilities** menu in Configuration Tools, select the **Function Library Manager**, or run the Functional Library Manager.bat file from the **utilities** directory where the Configuration Tools is installed. The Function Library Manager window appears.

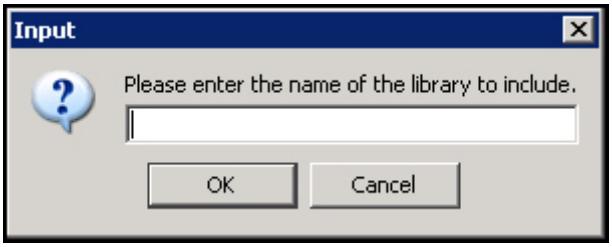


Function Library Manager Window

Adding a Function Library to Be Validated in the Configuration Tools

Perform the following procedure to add a function library to the Configuration Tools:

1. Launch the Function Library Manager.
2. Click **Add**. The Input dialog box appears.



Input Dialog Box

3. Enter the name of the library you want recognized.

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 you want to remove from the validation process.
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 of the 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

The following reports can be created using the Report Generator:

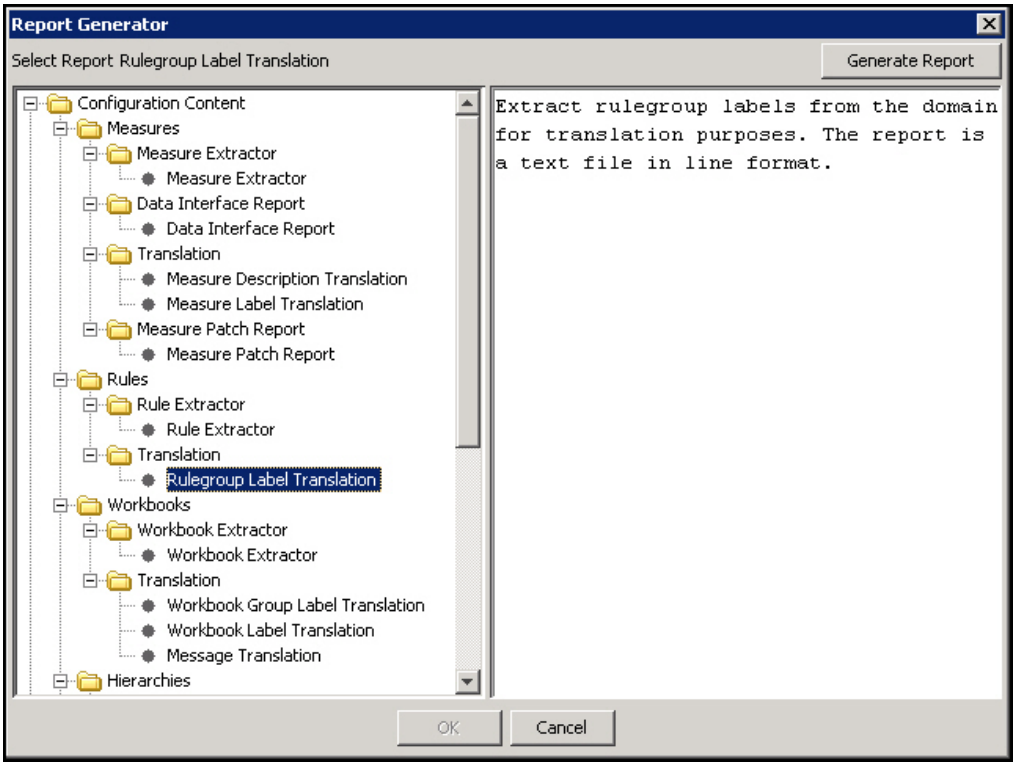
- **Measure Extractor** – This report generates a text file that lists the measure content of a solution.
- **Data Interface Report** – This report generates a text file that lists the properties of all of the measures in the project that have been added to the Data Interface Tool.
- **Measure Description 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 build the domain first.
- **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 build the domain first.
- **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 build the domain first.
- **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 build the domain first.
- **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 build the domain first.
- **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 build the domain first.
- **Dimension 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 dimension labels for a project without the need to build the domain first.
- **Hierarchy 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 hierarchy labels for a project without the need to build the domain first.
- **Hierarchy.xml Report–** This report generates the hierarchy.xml resource file used by RPAS during the domain creation and the dimension patching processes.
- **Taskflow Description** – This report generates the taskflow.xml document used by the RPAS Fusion Client similar to the file generated as part of the installation process. It allows the creation of the xml file without the need to build or patch the domain first.
- **Taskflow Resources** – This report generates the Resource Bundle used by the RPAS Fusion Client similar to the file generated as part of the installation process. It allows the creation of the resources bundle without the need to build or patch the domain first.

Generate a Report

Perform the following procedure to generate a report:

1. Select the project that requires the report.
2. Select **Generate Reports** from the Utilities menu. The Select a Report dialog box 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 where the generated file is to be stored 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.

7

Integration Tool

Overview

The RPAS platform supports the integration of domains with a Data Mart maintained within an Oracle Database instance. Once integrated, a domain will be able to automatically push data into and pull data out of the database in order to perform operations such as workbook building and committing. By integrating multiple RPAS domains with a single Data Mart, it will be possible for domains to share data automatically through transparent platform processes without the need for batch processes to synchronize information.

In order to support these behaviors, an RPAS Data Mart must be created within an Oracle Database instance. Due to the highly configurable nature of RPAS domains, it will be necessary to allow an equally flexible configuration of the objects and information present within the Data Mart. The process of determining the objects required for a RPAS Data Mart and the specification of their relevant properties is the Integration Configuration process.

RPAS Data Mart

Overview

The RPAS Data Mart, or RDM, is a persistent data store maintained within an Oracle Database. The RDM is used as a central repository and storage of record for customer data that can be produced and/or consumed by the processes performed in one or more RPAS domains that are integrated with that RDM.

The primary purpose of the RDM is to maintain a set of facts. These facts, which are stored within in tables of the Oracle Database, are conceptually equivalent to the measures of a RPAS domain. In addition, the information within a fact is structured in a manner that is functionally equivalent to the multi-dimensional system present in a RPAS domain.

It is therefore possible to access fact information through an address composed of a set of positions along discrete dimensions. Each row of a fact table, like a cell within an RPAS array, corresponds to a unique combination of positions along one or more dimensions (for example, a single sku, store, and week). However, unlike a RPAS array, a fact table may contain values for multiple facts within a single table. Each fact is represented by a distinct column within the fact table.

For any given column (fact) and row (position address) a table will contain a value if the fact has a populated value for that address but will contain an empty cell for that column should the fact have no value for that address.

The RPAS Data Mart also maintains tables to describe the hierarchies and dimensions used to structure the facts contained within the Data Mart. This information is stored within a separate set of tables, known as dimension tables, in a manner analogous to the dimension arrays of an RPAS domain. As with the hierarchies and dimensions of an RPAS domain, the dimension tables of a RPAS Data Mart can be administered in order to add, remove and modify individual positions and has the ability to represent the child/parent relationships that describe the multiple levels represented by the dimensions of a hierarchy.

Integration Configuration Components

Overview

In order to properly configure the RPAS Data Mart, it is necessary to supply information about three components. These components are the Shared Hierarchies and Dimensions, the Shared Facts and the Integration Map. The information about these components is stored in a XML document referred to as the integration configuration which is created and maintained by the RPAS Configuration Tools. In addition, the integration configuration contains a fourth component used internally by the Configuration Tools to manage the configuration process.

Shared Hierarchies and Dimensions

Overview

The information contained within the fact tables of a RPAS Data Mart, like the information contained within the measures of a RPAS domain, is structured to represent a set of multidimensional relationships. This structure is represented by a set of dimensions that are defined along hierarchies that describe the space in which the information is relevant. These hierarchies can represent common familiar constructs such as the Calendar, Location or Product hierarchies. They can also represent concepts that are application specific.

The Integration Configuration contains information about the hierarchies represented within the RPAS Data Mart. It also contains information about the structure of the dimensions that are defined within a hierarchy. As with a domain configuration, dimensions are defined in a tree structure to allow the representation of roll-up information.

When configuring the hierarchies and dimensions of the RDM, it is not necessary to include every hierarchy and dimension within the domain configuration of a domain integrated with that RDM. It is only necessary to define the set of dimensions that form the intersections of the facts used to share measure data.

In cases where the domains integrated with a RPAS Data Mart have non-identical hierarchical representations, it may also be necessary to include one or more dimensions so that every hierarchy has a single root dimension and all dimensions within the hierarchy can be fully expressed by one-to-one or one-to-many parent-child relationships.

Furthermore, it may be desirable to include additional dimensions within a RPAS Data Mart even if their inclusion is not motivated by requirements of integration. For example, additional dimensions that are not used in fact intersections may be included in order to support reporting against the Oracle Database.

Shared Hierarchies Properties

When defining a shared hierarchy, it is necessary to specify the following properties:

- **Hierarchy Name** – This is the name of the hierarchy. The name of a hierarchy within the RDM should correspond to the RPAS Name of a domain hierarchy in order for that domain hierarchy to participate in sharing data with the RDM.
- **Hierarchy Label** – The label is a user label used to identify the hierarchy in reporting and user notification.
- **Hierarchy Purge Age** – The purge age is the amount of time RPAS will continue to store a hierarchy position after the position is no longer included in hierarchy load

files. If the amount of time in days has passed since the last hierarchy load which contained an entry for a given position, that position will be marked for purging from the system.

- **Hierarchy Order** – As with hierarchies in a RPAS domain, the hierarchies of a RPAS Data Mart must be ordered. This ordering will be used in the construction of the tables holding fact information within the RDM. In order for a RPAS domain to participate in sharing data with a RDM, it is necessary for the relative order of the hierarchies within that domain to be the same as the relative ordering of the hierarchies within the RDM. It is not necessary for the hierarchies within a domain to have identical values for the order attribute as the RDM hierarchies but the order of hierarchies relative to each other must be the same.

Shared Dimensions Properties

When defining a shared dimension, it is necessary to specify the following properties:

- **Dimension Name** – This is the name of the dimension. The name of a dimension within the RDM should correspond to the RPAS Name of a domain dimension in order for that domain dimension to participate in sharing data with the RDM.
- **Dimension Label** – The label is user label used to identify the dimension in reporting and user notification.
- **Position Format** – As within a RPAS domain, the root dimension of the Calendar hierarchy must have a defined position format. The position format describes how to represent a time in the format of the position names of the root dimension and is used to determine which position within the Calendar root dimension corresponds to any given point in time.

Shared Facts

Overview

Shared Facts are entities within the RPAS Data Mart that correspond to measures within a RPAS domain. Like domain measures, facts are defined as either scalar or multi-dimensional. Once a RPAS domain has been integrated with a RPAS Data Mart, it is possible to specify mappings of domain measures to RDM facts. These mappings allow a RPAS domain to use the fact as it resides within the Oracle Database to store the information previously associated with the domain measure.

Information can be pulled from and pushed to the Oracle Database automatically as a part of domain operations, making the fact within the Oracle Database the store-of-record for the information. When multiple domains are integrated into a single RPAS Data Mart, there can be several mappings for a single fact, one in each domain. When this occurs, the domains can seamlessly share a single version of the fact information without the need for overt integration operations of data duplication and synchronization.

Within the RPAS Data Mart, facts are stored within fact tables. The RDM has the ability to store multiple facts within a single table. When this occurs, each fact is represented by a separate column within the table, while the rows of the table represent the positional addresses for the dimensional space of the fact.

By grouping multiple facts within a single table, the RPAS Data Mart is able to reduce the amount of time required to retrieve information from the Oracle Database. Measures that are often read and/or written together and that contain the same fill pattern (or sets of addresses which have data as opposed to addresses for which no data is present) can be quickly accessed together when grouped in a single table.

Conversely, grouping facts together in a single table can have an adverse impact on data access if those facts are not accessed together and if those facts do not tend to have similar fill patterns. For this reason, the assignment of facts to fact tables and the grouping of facts within a table can have a large impact on system performance.

Shared Fact Properties

Shared Facts are defined by a number of properties. The majority of the properties involve the definition of the type of data represented by the fact while some are used to describe where and how the fact is stored within the fact tables of the RDM. The properties of a shared fact are:

- **Fact Name** – This is the identifier of the fact. It is used by RPAS to determine which fact is required for any given operation.
- **Fact Label** – This is a user label that can be used to represent the fact in reporting and user notification.
- **Fact Intersection** – The intersection of a fact, like the intersection of a measure, describes which dimensions are used to define the address space of the fact. In order for a domain to share a measure's data with a RDM fact, the domain measure must have the same intersection as the RDM fact.
- **Fact Type** – The fact type describes the type of data stored within the fact. These types are drawn from the set of defined RPAS measure types (real, integer, Boolean, Date and string). In order for a domain to share a measure's data with a RDM fact, the domain measure must have the same type as the RDM fact.
- **Fact Group** – The fact group is an organizational attribute that is used to distribute facts across fact tables within the RDM. A single fact table will contain the information for all facts belonging to a single fact group. Due to the large impact efficient grouping of facts can have on system performance, best practices regarding the assignment of facts to fact groups are described in section 3.3.
- **Fact Table** – The fact table is the physical name of the fact table within the Oracle Database that will contain the data associated with the fact. This attribute is derived from the fact group and need not be configured separately.
- **Fact Description** – The fact description is a property that allows the description of the context of the data contained within a fact. It can be used for reporting or user notification.
- **Fact Na Value** – The Na value of a fact is the implied value of that fact for any positional address for which the fact's fact table does not contain a value. It is analogous to the Na value of a measure within a RPAS domain. In order for a domain to share a measure's data with a RDM fact, the domain measure must have the same Na value as the RDM fact.
- **Fact Purge Age** – The fact purge age is an attribute used by RPAS to determine how long information loaded into a fact should be maintained by the system before it is purged as obsolete.

Integration Map

Overview

The Integration Map is the structure that describes which domain measures share data with the facts contained within the RPAS Data Mart. The map is made up of a set of entries; each entry describes the fact within the RPAS Data Mart and the domain and measure that participate in the sharing.

Integration Map Properties

Integration Map properties include:

- **Shared Fact** – The fact within the RDM that will store the data to be used by the domain measure participating in the sharing.
- **Domain Id** – An identifier that describes which domain the measure participating in the sharing is defined.
- **Domain Measure** – The measure within the domain that participates in the data sharing.

Integration Map Constraints

In order for a measure to be eligible to share data with a RDM fact, the measure must have compatible values for several of the properties of the fact.

- **Measure Type** – The data type of the measure must be the same as the data type of the fact.
- **Measure Na Value** – The Na Value of the measure must be the same as the Na Value of the fact.
- **Measure Base Intersection** – The base intersection of the measure must be the same as the fact intersection. Note that, in configurations that make use of the hierarchy indirection feature of the Configuration Tools, the literal value of the intersection property of a measure may differ from the fact intersection due to use of the RPAS Name dimension attribute or labeled intersections; in such cases, the RPAS internal intersection is used in place of the literal value of the measure intersection property for this check.

Domain Information

Overview

The Integration Tool uses the domain information to identify the domain configurations of the domains that are being integrated into the RPAS Data Mart. This information is used internally be the Integration Tool for many of its processes.

Domain Information Properties

Domain Information properties include:

- **Domain Identifier** – This is an identifier used to describe the domain within the Integration Map.
- **Configuration Location** – This is the location of the configuration that represents the domain. It is used by the Integration Tool to determine which version of a configuration should be used by the Integration Tool.

Integration Tool

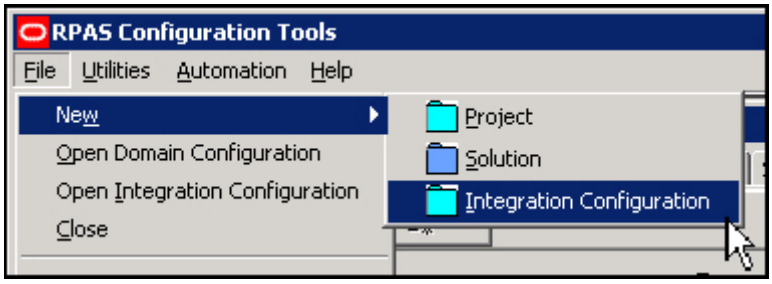
Overview

In order to allow the creation and maintenance of the Integration Configuration, a new tool will be added to the Configuration Tools. This tool, called the Integration Tool, will allow users to specify the metadata of the RPAS Data Mart be creating shared hierarchies, dimensions and facts. It will also allow users to maintain the Integration Map used to describe which domain measures will participate in data sharing with the RDM shared facts.

Unlike the other tools used within the RPAS Configuration Tools, an instance of the Integration Tool is not tied to a domain configuration. Instead, a new XML document, called the integration configuration, is used to store all integration-related information. As a result, there is no integration-related information in a domain configuration and, therefore, it is not necessary to modify a domain configuration when specifying integration information. This negates the need for multiple copies of domain configurations to support different integration scenarios or to support integrated versus non-integrated domains.

Working Integration Configurations

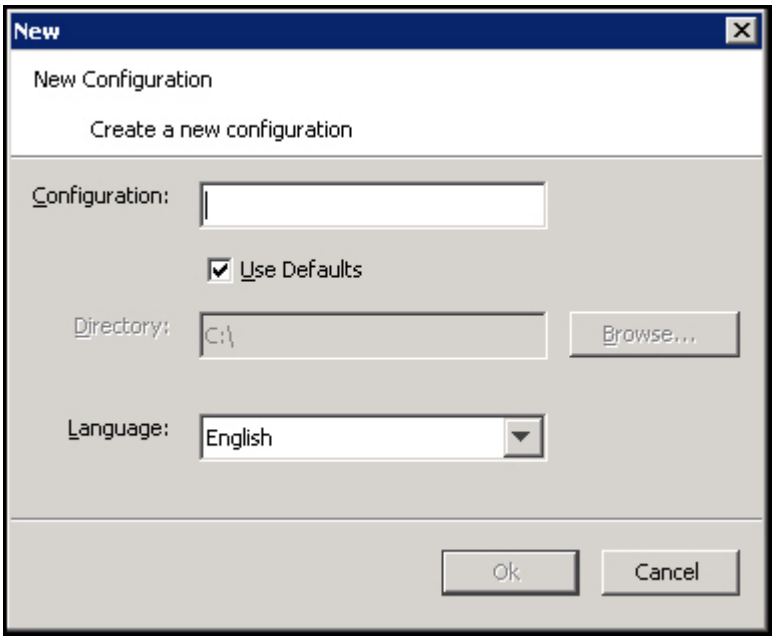
Like domain configurations, integration configurations can be accessed within the Configuration Tools through the File menu. Use the Open Integration Configuration option to open an existing configuration or the New Integration Configuration option to create a new configuration.



New Integration Configuration Menu Item

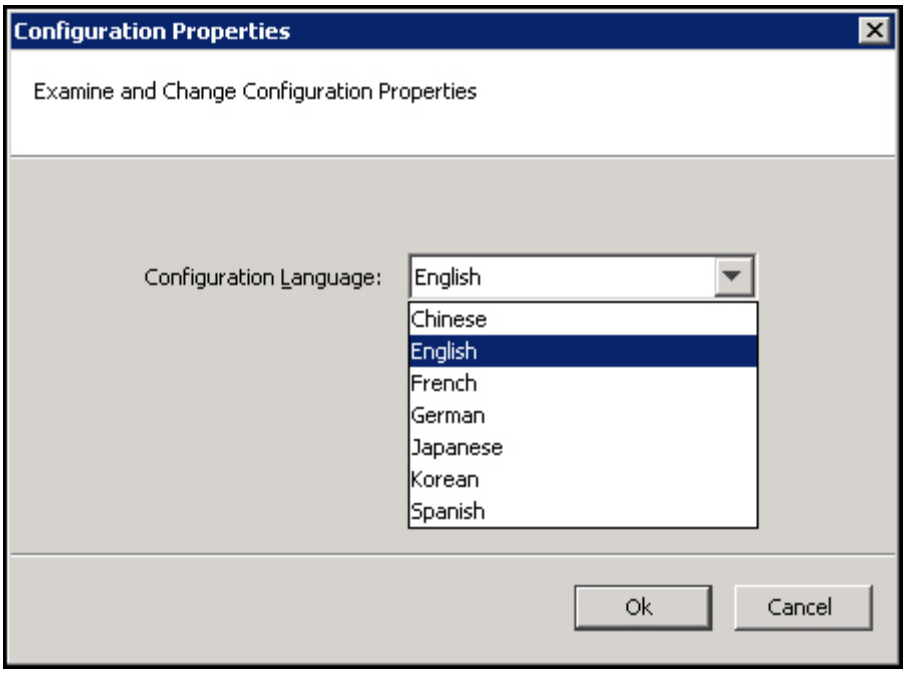
Once an integration configuration has been created or loaded, the Integration Tool will load into the Configuration Tools Workbench.

When creating an integration configuration, users will be prompted to give the new configuration a name and select a location in which to save the configuration. Users will also have the ability to specify the language of the labels entered within the Integration Tool. When loading the labels contained within the Integration Configuration, RPAS will associate the contained labels with the language of the configuration.



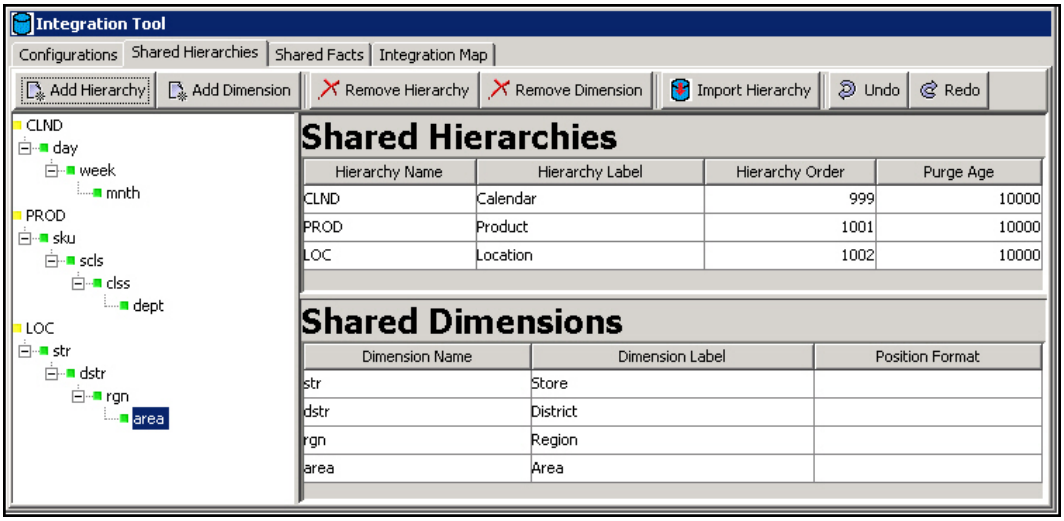
Integration Configuration Dialog Box

Once an integration configuration has been created, the language setting for the configuration may be inspected and modified using the Configuration Properties dialog.



Configuration Properties Dialog Box

The Integration Tool contains four tabs: Configurations, Shared Hierarchies, Shared Facts and Integration Map. Each tab allows configuration of one of the four components described below.

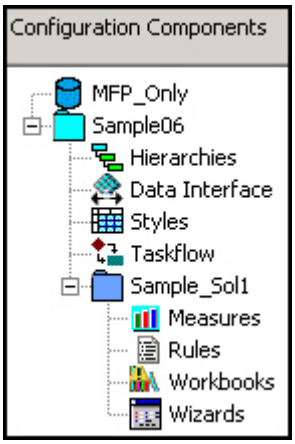


Integration Tool Within the RPAS Configuration Tools

The Integration Tool contains four tabs: Configurations, Shared Hierarchies, Shared Facts and Integration Map. Each tab allows configuration of one of the four components described below.

Changes to an integration configuration may be saved using the Save or Save As options within the File menu and an open integration configuration can be closed using the Close or Close All options.

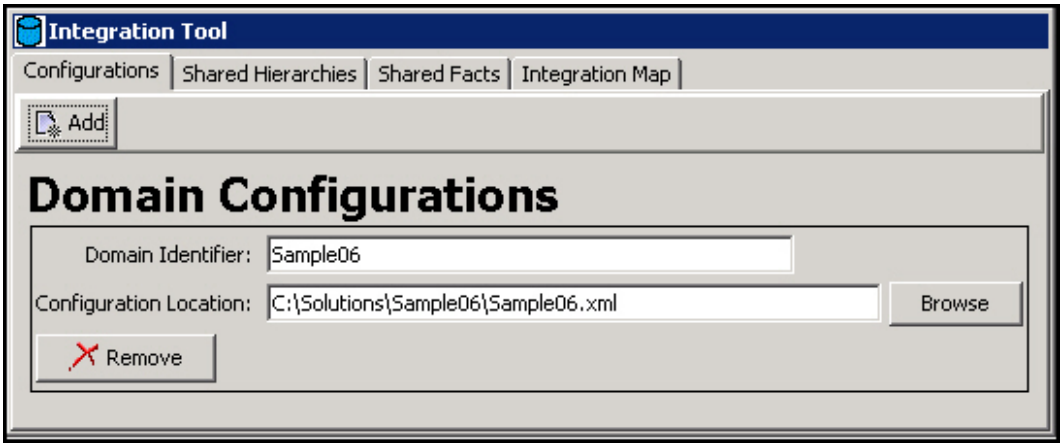
When working with multiple integration configurations or with a mixture of integration configurations and domain configurations, the Configuration Components pane can be used to navigate to integration configurations, which are represented within the pane by the icon.



Configuration Components Pane

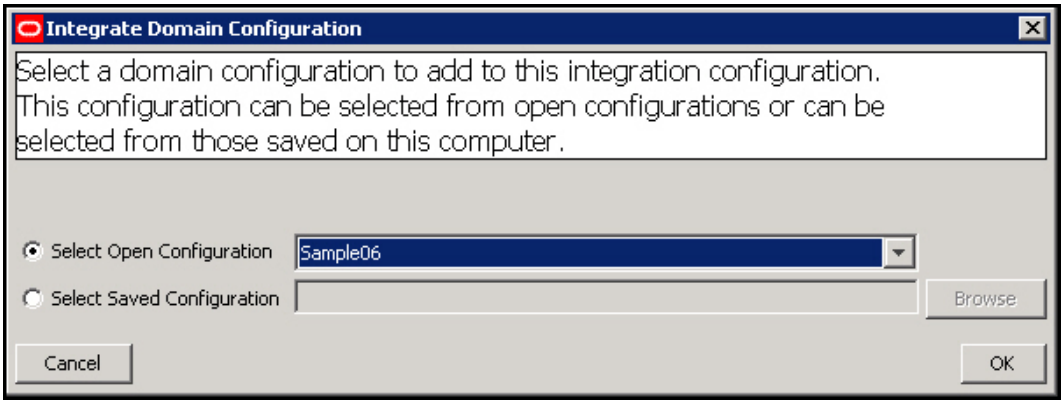
Working with Domain Information

The Configurations tab can be used to register domain configurations with an integration configuration. When working with the Integration Tool, users will be able to create integration configuration content by referencing domain configuration content in registered domain configurations. As such, it is necessary to add an entry to the Configurations tab for each domain configuration that will be used in conjunction with the configured RPAS Data Mart.



Configurations Tab of the Integration Tool

A domain configuration can be registered with the integration configuration by clicking the Add menu bar button. When this happens, a dialog will appear to allow the selection either of a domain configuration currently open in the Configuration Tools or for a domain configuration saved on disk to be registered by entering the location of the configuration.



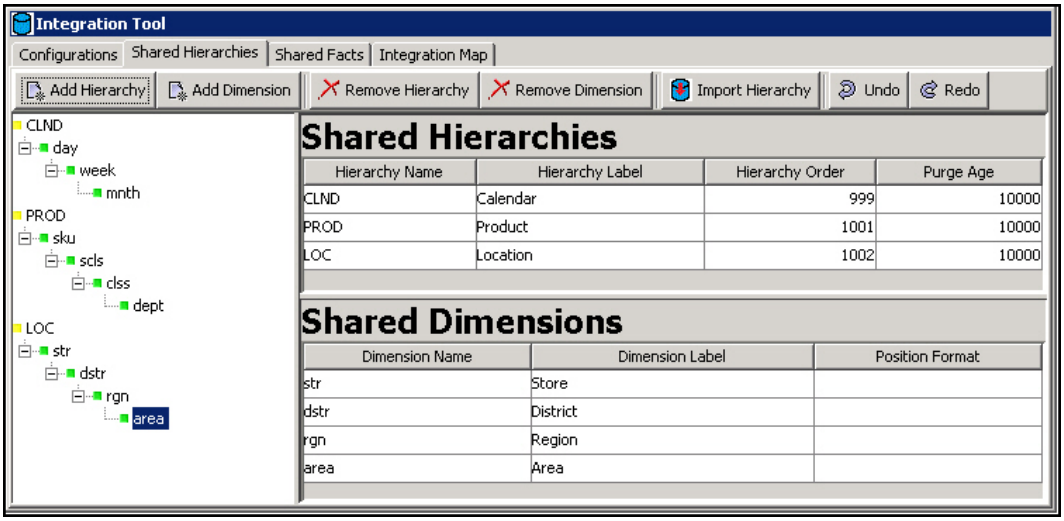
Add Domain Configuration Dialog Box

The Domain Identifier and Configuration Location properties can be edited within the Configuration tab and, should it be desired, a currently registered domain configuration can be removed using the Remove button.

Working with Shared Hierarchies

Overview

The Shared Hierarchies tab is used to configure the shared hierarchies and dimensions that will exist within the RPAS Data Mart.



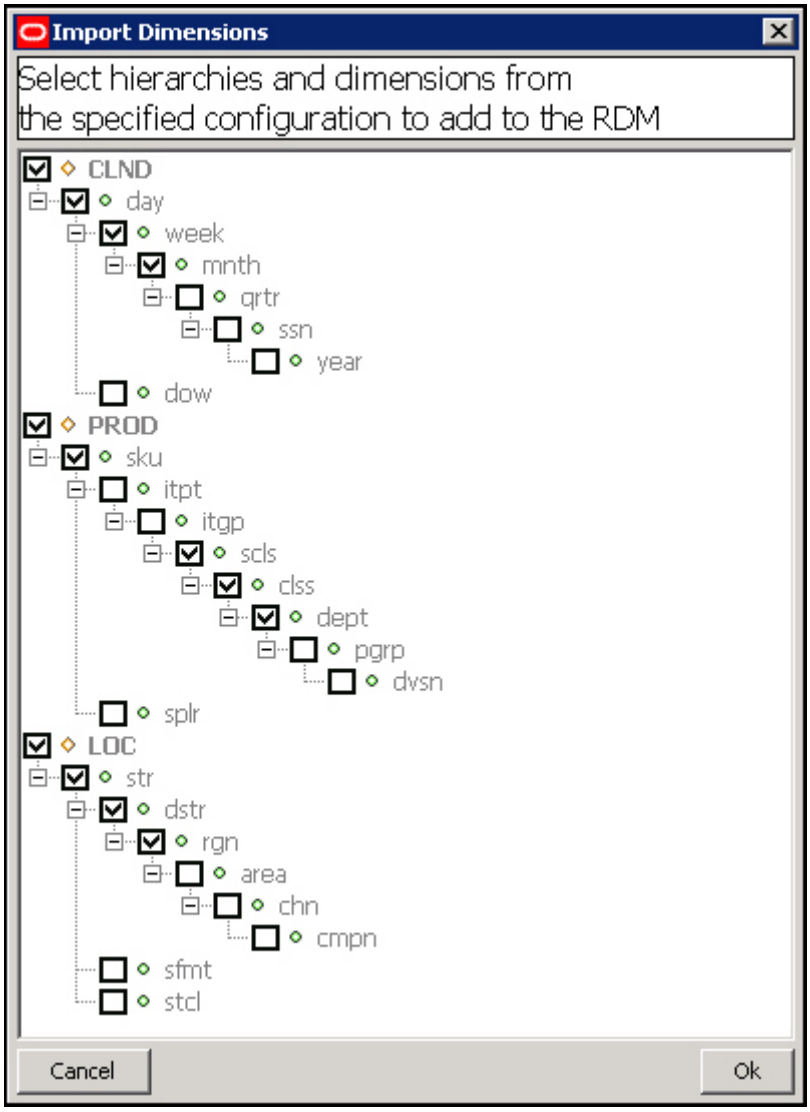
Shared Hierarchies Tab of the Integration Tool

The Shared Hierarchies tab looks very similar to the Hierarchy Tool of the Configuration Tools and behaves in many ways the same. Users of the Hierarchy Tool will therefore find that working with the Shared Hierarchies tab will be familiar to them.

Hierarchies and Dimension can be added and removed using the appropriate menu bar buttons. In addition, the structure of dimensions can be modified through drag-and-drop, just as in the Hierarchy Tool.

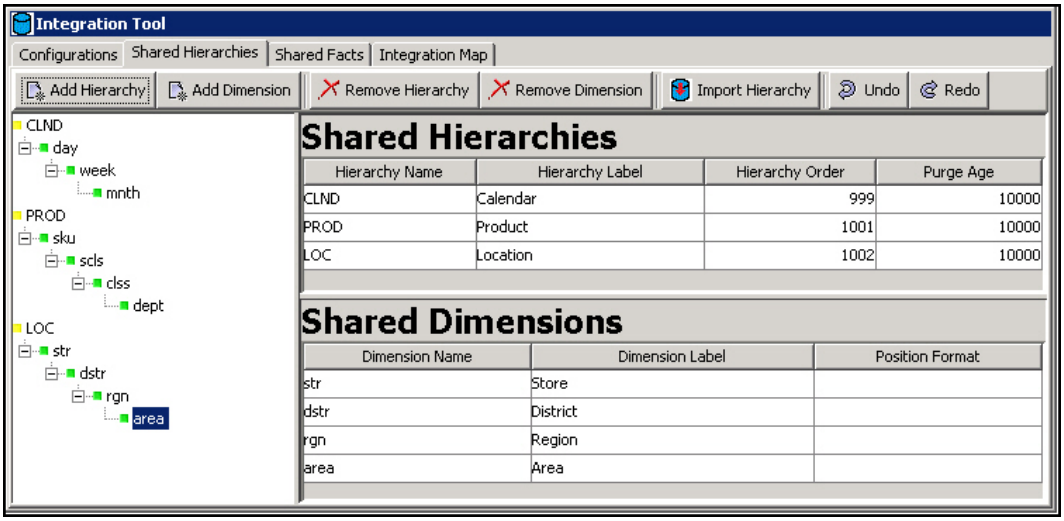
Users can also modify the properties of a shared hierarchy or shared dimension through editing the table present in the Shared Hierarchy tab.

One new feature within the Shared Hierarchy tab is the Import Hierarchy menu button. The Import Hierarchy button, when clicked will launch a dialog that allows users to select multiple domain hierarchies and dimensions at once from an open domain configuration. When the user clicks okay within the dialog, shared hierarchies and dimensions will be created based upon the properties of the imported domain hierarchies and dimensions.



Import Dimensions Dialog Box

In the figure above, hierarchies and dimensions that already exist inside the integration configuration appear as a shaded and disabled check box. Dimensions present in the selected domain configuration (should more than one be open) appear as unchecked. Users can check these hierarchies and dimensions to specify what content should be created in the Integration Configuration. By clicking the okay button within the Import Dimensions dialog, the Integration Tool will create Shared Hierarchies and Dimensions corresponding to the selected components.



Shared Hierarchies Tab with the Selected Content Imported

Shared Hierarchies Tab Validations

The Integration Tool supports several validations on hierarchy and dimension content. As with the traditional Configuration Tools, these validations are evaluated in real-time as content is rendered by the UI. When a property of a hierarchy or dimension violates a validity constraint, the field displaying that property will paint with the Configuration Tools-standard red text color. In addition, the tooltip for the invalid cell will contain a message describing the validity constraint that has been violated.

The following validity constraints have been defined for shared hierarchies and dimensions:

Hierarchy Name

- Hierarchy name is a required property. It cannot be empty.
- Hierarchy names must be valid names for RPAS hierarchies. They cannot exceed four characters in length, must begin with a letter and can contain only letters and numerals.
- All hierarchy names must be unique. No other element in the Integration Configuration may share the name of a shared hierarchy.

Hierarchy Label

Hierarchy labels are not validated.

Hierarchy Purge Age

- Purge Age is a required property. It cannot be empty.
- Purge Age must be a positive integer value.

Hierarchy Order

- Hierarchy order values must be positive integer values between 999 and 9999.
- The Calendar hierarchy must be registered at order 999.
- No hierarchy can have a lower order value than the hierarchy that precedes it.
- If any registered domain configuration is currently loaded in the Configuration Tools, the order of the shared hierarchies will be compared to the domain hierarchies to validate that the relative ordering of hierarchies is identical.

Dimension Name

- Dimension name is a required property. It cannot be empty.
- Dimension names must be valid names for RPAS dimensions. They cannot exceed four characters in length, must being with a letter and can contain only letters and numerals.
- All dimension names must be unique. No other element in the Integration Configuration may shared the name of a shared dimension.

Dimension Label

Dimension labels are not validated.

Dimension Position Format

- The root dimension of the Calendar hierarchy must have a defined position format.
- No dimension that is not the root dimension of the Calendar hierarchy may have a defined position format.
- The supplied position format must be a valid RPAS format.
- The supplied position format must be a valid Oracle date format.

Calendar Hierarchy

1. If an integration configuration includes the Calendar hierarchy, that hierarchy must include the day dimension.

Working with Shared Facts

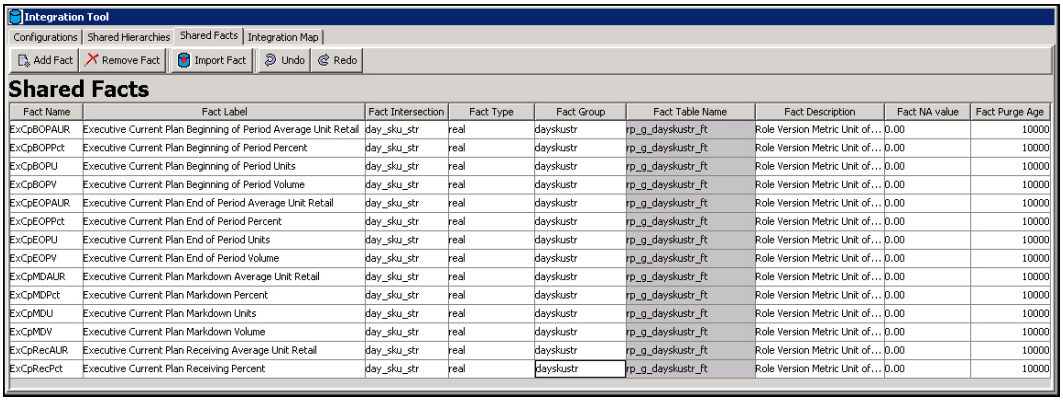
Overview

The Shared Facts tab allows users to configure the shared facts that will exist within the RPAS Data Mart.

Shared Facts Tab

The main feature of the Shared Facts tab is the fact table. This table lists the shared facts that have been defined within the integration configuration. Users can manage the properties of configured facts by editing the appropriate cell within the fact table. As is the case within the Measure Tool, many attributes support the use of smart editors. These editors allow selection from a drop-down list of options or can be used to launch a dialog to assist in the specification of values. In addition, most fact properties are validated and will provide feedback to the user when the specified value is not permitted.

New facts can be added to the integration configuration be clicking the Add Fact menu button. Existing facts can be removed by selecting the fact and clicking the Remove Fact menu button. In addition, the Shared Fact tab contains the ability to create a fact based upon a configured domain measure. This process can be launched by clicking the Import Fact menu button.



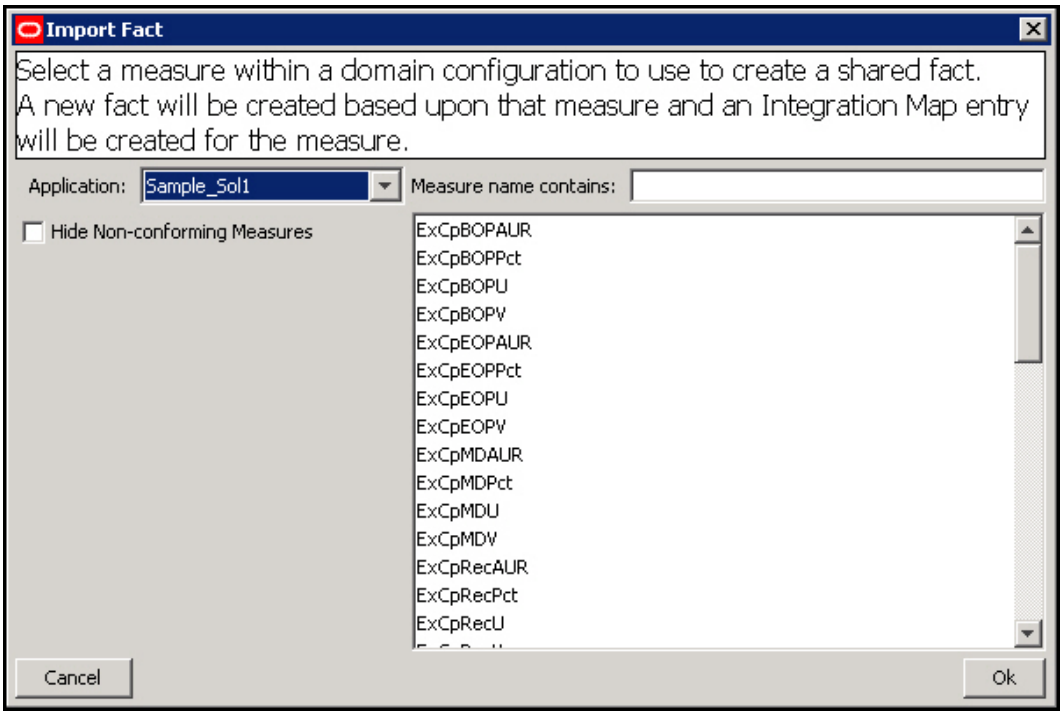
Fact Name	Fact Label	Fact Intersection	Fact Type	Fact Group	Fact Table Name	Fact Description	Fact NA value	Fact Purge Age
ExCpBOPAUR	Executive Current Plan Beginning of Period Average Unit Retail	day_sku_str	real	dayskustr	rp_g_dayskustr_ft	Role Version Metric Unit of...	0.00	10000
ExCpBOPPct	Executive Current Plan Beginning of Period Percent	day_sku_str	real	dayskustr	rp_g_dayskustr_ft	Role Version Metric Unit of...	0.00	10000
ExCpBOPU	Executive Current Plan Beginning of Period Units	day_sku_str	real	dayskustr	rp_g_dayskustr_ft	Role Version Metric Unit of...	0.00	10000
ExCpBOPV	Executive Current Plan Beginning of Period Volume	day_sku_str	real	dayskustr	rp_g_dayskustr_ft	Role Version Metric Unit of...	0.00	10000
ExCpEOPAUR	Executive Current Plan End of Period Average Unit Retail	day_sku_str	real	dayskustr	rp_g_dayskustr_ft	Role Version Metric Unit of...	0.00	10000
ExCpEOPPct	Executive Current Plan End of Period Percent	day_sku_str	real	dayskustr	rp_g_dayskustr_ft	Role Version Metric Unit of...	0.00	10000
ExCpEOPU	Executive Current Plan End of Period Units	day_sku_str	real	dayskustr	rp_g_dayskustr_ft	Role Version Metric Unit of...	0.00	10000
ExCpEOPV	Executive Current Plan End of Period Volume	day_sku_str	real	dayskustr	rp_g_dayskustr_ft	Role Version Metric Unit of...	0.00	10000
ExCpMDAUR	Executive Current Plan Markdown Average Unit Retail	day_sku_str	real	dayskustr	rp_g_dayskustr_ft	Role Version Metric Unit of...	0.00	10000
ExCpMDPct	Executive Current Plan Markdown Percent	day_sku_str	real	dayskustr	rp_g_dayskustr_ft	Role Version Metric Unit of...	0.00	10000
ExCpMDU	Executive Current Plan Markdown Units	day_sku_str	real	dayskustr	rp_g_dayskustr_ft	Role Version Metric Unit of...	0.00	10000
ExCpMDV	Executive Current Plan Markdown Volume	day_sku_str	real	dayskustr	rp_g_dayskustr_ft	Role Version Metric Unit of...	0.00	10000
ExCpRecAUR	Executive Current Plan Receiving Average Unit Retail	day_sku_str	real	dayskustr	rp_g_dayskustr_ft	Role Version Metric Unit of...	0.00	10000
ExCpRecPct	Executive Current Plan Receiving Percent	day_sku_str	real	dayskustr	rp_g_dayskustr_ft	Role Version Metric Unit of...	0.00	10000

Shared Facts Tab

Import Facts

When the Import Fact dialog is launched, it will prompt the user to select the domain configuration (if more than one is open in the Configuration Tools) and solution the user wants to import information from. Once the domain and application are selected, the right-hand list will populate with all measures contained within the application that have configured database values. If the user wishes, the list can be further filtered to exclude all measures whose intersections contain dimensions that are not configured within the Shared Hierarchies tab. Finally, users can enter a string of text; only measures whose name contains that string will be shown.

When the desired measure has been selected within the list and the Ok button has been pressed, the Integration Tool will create a new fact. The new fact will have most of its properties automatically specified by using the value of the corresponding measure property (for example, the fact type will be set to be the data type of the selected measure).



Import Fact Dialog Box

Shared Fact Tab Validations

The Integration Tool supports several validations on fact content. As with the traditional Configuration Tools, these validations are evaluated in real-time as content is rendered by the UI. When a property of a fact violates a validity constraint, the field displaying that property will paint with the Configuration Tools-standard red text color. In addition, the tooltip for the invalid cell will contain a message describing the validity constraint that has been violated.

The following validity constraints have been defined for shared facts:

Fact Name

- Fact name is a required property. It cannot be empty.
- Fact names must be between one and thirty characters in length. They must begin with a letter and can contain letters, numerals and underscores.
- All fact names must be unique. No other element in the integration configuration may share the name with a fact.
- Some names reserved for use by RPAS. In addition, fact names may not end with certain strings, such as _id, _stg, _ft.

Fact Label

Fact labels are not validated.

Fact Intersection

- Fact intersection is a required property. It cannot be empty but may be scalar.
- All dimension references in a fact's intersection must resolve to dimensions that exist in the RDM.
- Intersections cannot contain references to more than one dimension in any single hierarchy within a fact intersection.

Fact Type

- Fact type is a required property. It cannot be empty.
- Fact type must conform to one of the set of valid RPAS measure types.

Fact Group

- Fact group is a required property. It cannot be empty.
- Fact groups must be between one and twenty characters.
- Fact groups must begin with a letter and can contain letters, numerals and underscores.
- No two facts may have the same fact group if they do not have the same intersection.

Fact Table

Fact table is a derived property and so is not itself validated

Fact Description

Fact descriptions are not validated.

Fact NA Value

- Fact NA Value is a required property. It cannot be empty.

- The value provided for fact NA Value must be valid based upon the type of the measure. The guidelines for what values are valid for any given type are identical to those used for measure NA Values.

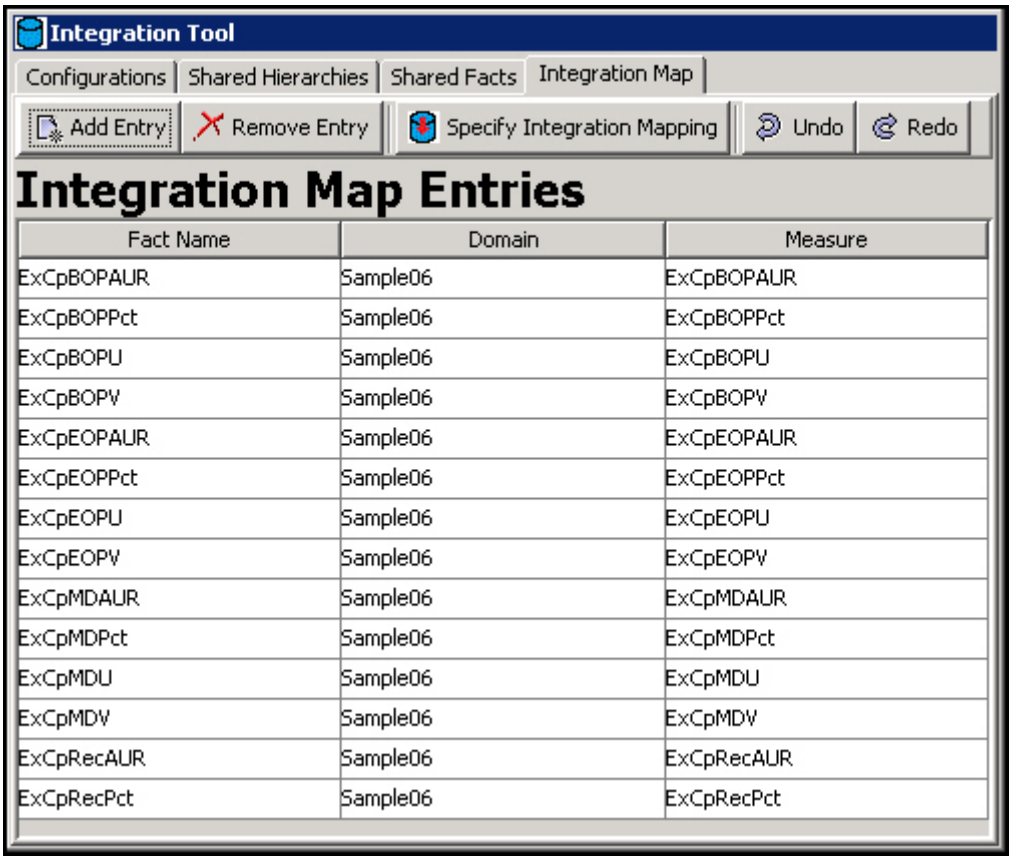
Fact Purge Age

Fact purge age is not validated.

Working with the Integration Map

Overview

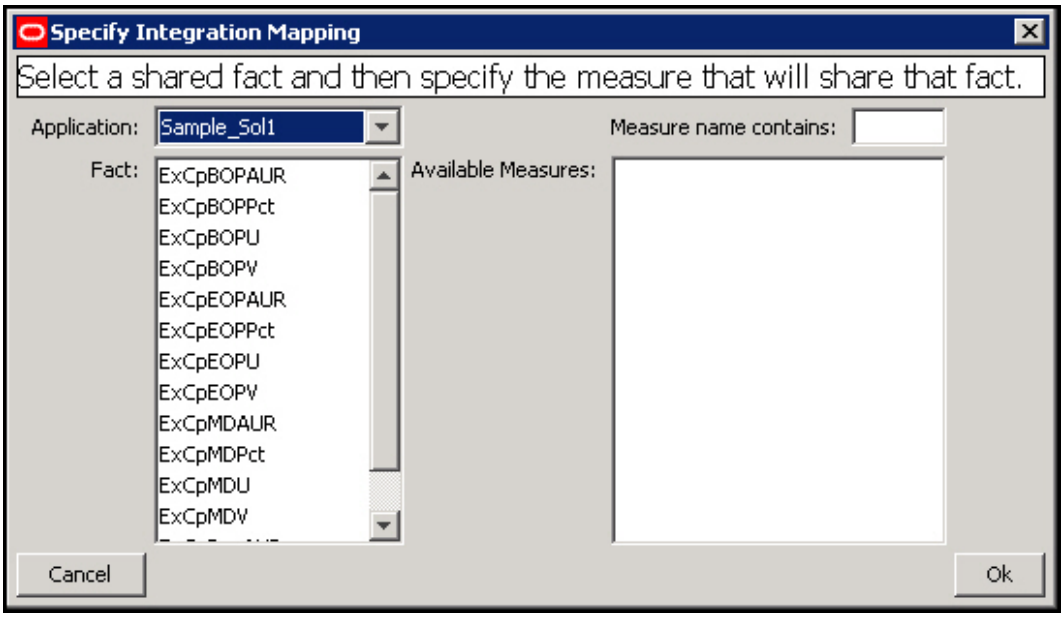
The Integration Map tab of the Integration Tool allows users to configure the domain measures that will share data with a RDM fact.



Integration Map Tab

The primary feature of the Integration Map tab is the integration map entries table. Users can specify the domain measures that will participate in sharing data with the RPAS Data Mart by adding entries to the table. Entries can be added using the Add Entry menu button; an entry can be removed using the Remove Entry button. The values for Fact name, Domain and Measure can be specified either through selection of a drop down control for fact and domain or through entering a measure name for measure.

In addition, when the user clicks on the Specify Integration Mapping menu button, they will launch a dialog:



Specify Integration Mapping Dialog Box

Within this dialog, the user can select a domain (when more than one is open within the Configuration Tools) and an application. The user also selects the shared fact that will participate in the data sharing. When these selections have been made, the Available Measures list will populate with the measures present within the selected domain and application whose properties are compatible with the selected fact. Users can also enter a string into the filter box to further filter the list of candidate measures.

Once a measure has been selected, clicking the Ok button will cause the Integration Tool to automatically create an entry in the Integration Mapping table and populate that entry with the specified values for fact, domain and measure.

Fact

- Fact is a required property. It cannot be empty.
- The name listed for fact must correspond to a fact defined in the Shared Facts tab.

Domain

- Domain is a required property. It cannot be empty.
- The name listed for domain must correspond to a domain registered in the Domain Configurations tab.

Measure

- Measure is a required property. It cannot be empty.
- If the domain configuration for the domain specified for this entry is loaded into the Configuration Tools, the value of measure will be validated and must correspond to a measure in that domain.
- If the domain configuration for the domain specified for this entry is loaded into the Configuration Tools, the specified measure must share the values for type, intersection and NA Value with the fact specified for this entry.

Fact Grouping Best Practices

Overview

With the addition of shared facts within the RPAS Data Mart, a new performance consideration becomes important within RPAS applications. This new consideration is the optimization of operations that access the Oracle Database to read or write fact data. Internal testing of RPAS Data Mart operations has shown that the organization of facts into fact groups can have a significant impact on the amount of time required to perform operations within the RDM.

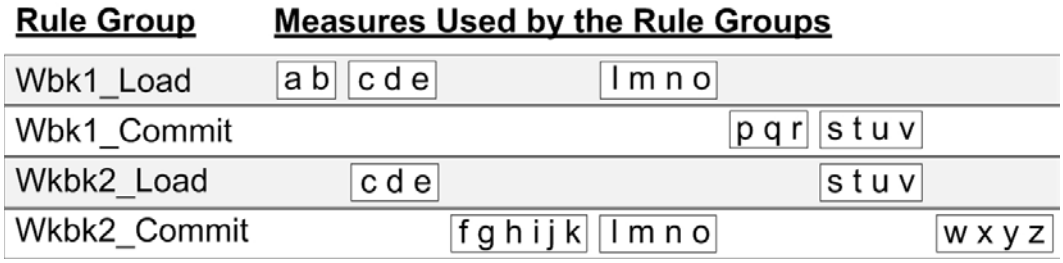
Grouping Based on Concurrent Access

Because the tables which store the fact data within the RPAS Data Mart can contain multiple facts, processing operations against those tables is affected not only by the number of populated values of a single fact but also of all other facts within the table. This becomes even more of a concern when writing data to the fact tables, as those facts being updated by the write operation must be merged with other facts within the fact table that are not affected by the write.

For this reason, it would be possible to design the tables of the RPAS Data Mart so that every fact would be contained within a separate table. While this would prevent performance issues related to having multiple facts within a single table, it would not result in optimal performance. If facts that are read and/or written together are placed within a single table, they can be processed together, greatly improving performance.

It is therefore desirable to group facts together in the fact tables but to do so with a distribution that group's facts accessed together into common tables while excluding facts that would not benefit from being grouped into those tables.

To illustrate the above, consider the following scenario. Assume two workbooks with measures that load from and commit to a partially overlapping set of facts. If we examine the sets of facts associated with the load and commit rule groups of the two workbooks, we would see a distribution of facts being loaded and committed (listed as a through z) such as that below:



Example of Distribution of Loaded and Committed Facts

In such a case, the measures being loaded and committed could be organized into groups such as:

Fact Group	Measures Categorized in the Fact Group
1	a b
2	c d e
3	f g h i j k w x y z
4	l m n o
5	p q r
6	s t u v

Sample Grouping of Facts for Rule Groups

By organizing the facts into the groups shown above, we maximize the performance benefit from grouping facts accessed together into groups while avoiding the performance hit from having additional facts within the groups that are not involved in the access.

Conditional Commits of Facts

When performing fact analysis such as described above, the presence of conditional commits should be taken into consideration. Some commit rules include a condition that acts as a mask to prevent some range of the available data from being committed. Take for example a measure that will only commit values for non-elapsed periods:

Measure1.master = if (current > today, Measure1, ignore)

Due to the way RPAS processes conditional commits, facts committed using different conditions should be separated from each other, as each condition would be required to be merged separately. In our example above, if it was determined that the measures associated with facts w,x,y and z are committed using the same condition but that the condition was not used for the measures associated with facts f, g, h, I, j and k, then we would find it preferable to assign w, x, y and z to a separate fact group from that used for f, g, h, i, j and k.

Fact Group	Measures Categorized in the Fact Group
1	a b
2	c d e
3a	f g h i j k
3b	w x y z
4	l m n o
5	p q r
6	s t u v

Modified Grouping of Facts to Accommodate Conditional Commits

Fact Group Assignment Process

Internal Testing suggests that the process whose overall performance is impacted the most by fact grouping is merging data into the Oracle Database. For this reason, it is suggested that facts be assigned to fact groups in such a way as to optimize the writing of data to the RPAS Data Mart. Below is a description of the suggested process for

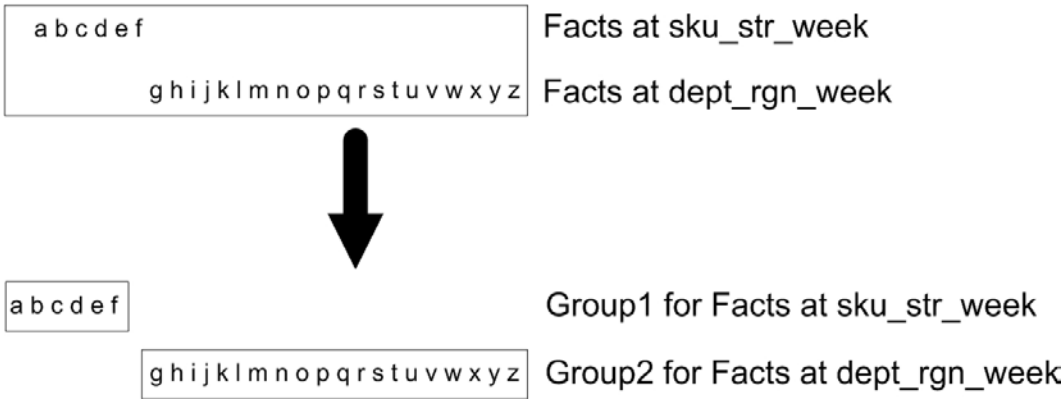
determining the fact groups to use for the facts within the RPAS Data Mart and for determining which facts should be assigned to each fact group.

1. Create a fact group for each unique intersection used by facts within the RDM.
2. Divide each group from step one into subgroups such that facts committed or loaded together are partitioned from the other facts within the group from step one.
3. Divide each group from step two, if necessary, to accommodate the use of conditional commits.

Create Groups Based Upon Intersection

All facts are defined along an intersection. The RPAS Data Mart does not allow facts with different intersections to be stored together in a single table. Because the fact group is the entity that corresponds to a single table within the Oracle Database, it is therefore impossible to place facts with different intersections within a single fact group.

Once facts have been assigned to groups based upon the fact intersection, the set of facts for the RPAS Data Mart will be divided into a number of pools. Treat each of these pools as an input to the second step of the fact grouping process.



Dividing Global Fact Pool by Fact Intersection

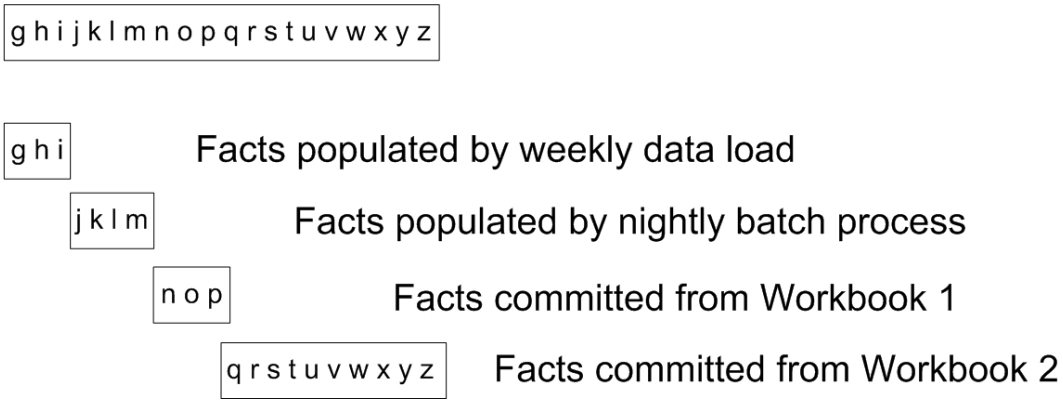
Partition Facts Based Upon Data Source

Each group created by grouping by intersection should now be subdivided in order to partition the facts based upon the process by which their information enters the RPAS Data Mart. Fact data is assumed to enter the RPAS Data Mart through one of three processes: workbook commit, batch process or import from external system (either data load or through the RDM interface tables).

For each of the above processes, it should be possible to determine the set of facts whose data enters the RPAS Data Mart in each operation. These sets of facts should be used to partition the initial fact groups to create individual sub-groups for the facts in each process.

Note that it is possible that some facts may be populated from more than one source. An example might be a fact that gets initially populated with raw data from an external source which is then transformed by a batch process or workbook prior to use by other processes within the application. This possibility is potentially more likely when dealing with applications that have never been integrated within the RPAS Data Mart.

When dealing with a fact with multiple sources, the sources should be prioritized based upon which process is the most time-critical. Once this prioritization has been done, the grouping suggested by the highest priority should be used. In cases in which priority cannot be identified, or when no process's grouping provides an acceptable trade-off, the fact or facts in question should be assigned to a separate fact group.

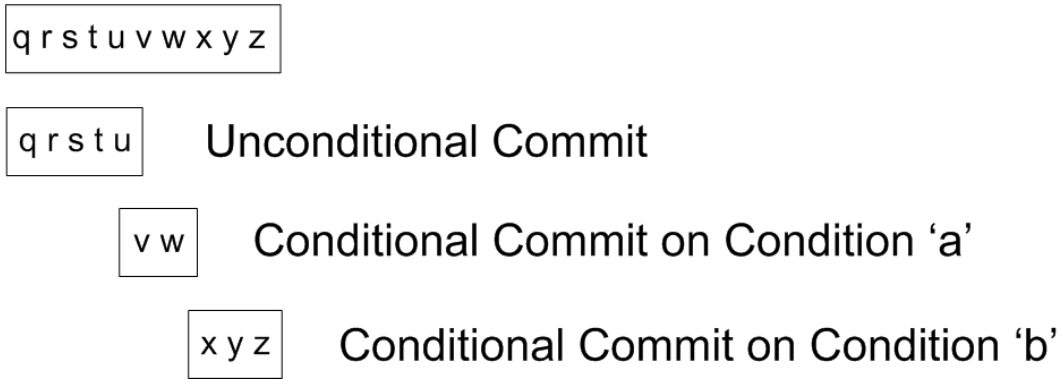


Subdivision of Group by Data Source

Partition Groups for Conditional Commits

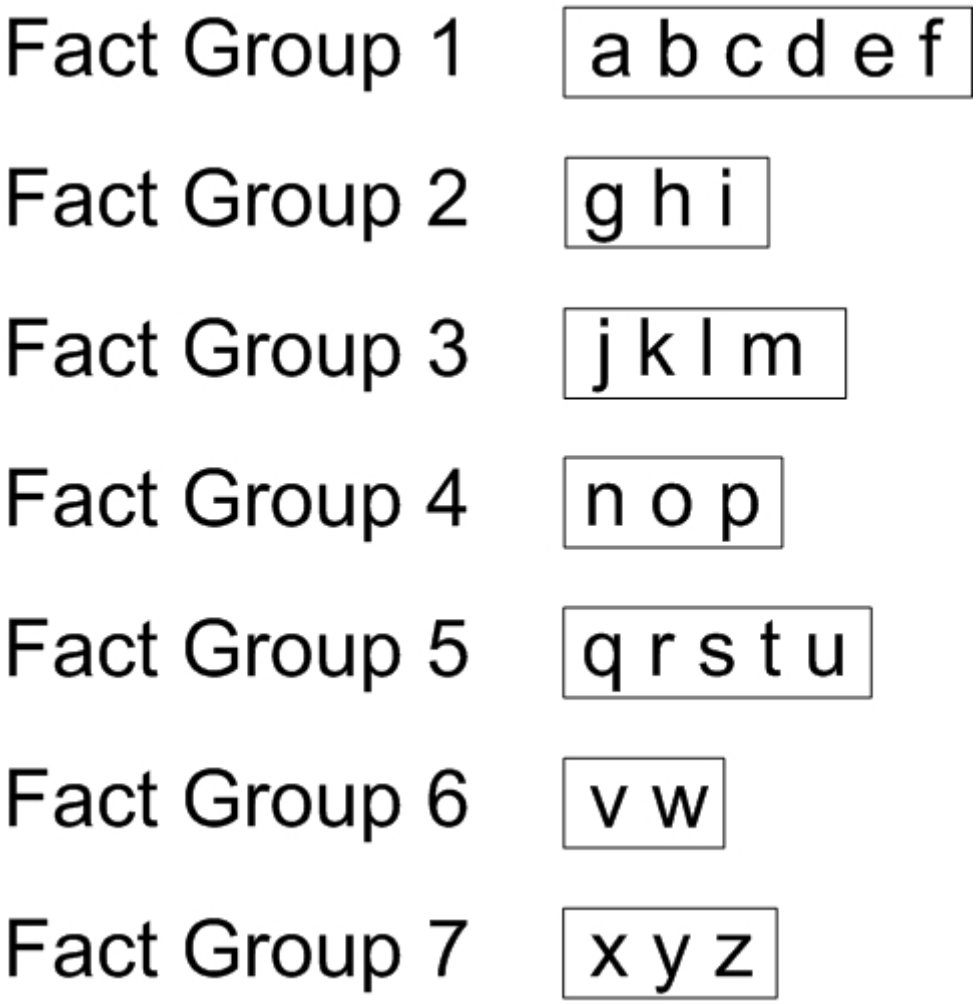
The final step in determining the grouping for the facts within the RPAS Data Mart is to take the groups identified in the graphic above and split them to partition facts that use conditional commits. Any fact groups created based upon workbook commit processes should be examined.

If any rules for the commit make use of a conditional commit (i.e. the rule makes use of an 'if' statement to commit only a subset of the measure used with the fact), the processing of any facts that do not make use of a conditional commit will be impacted by those facts that do use the conditional commit. In order to reduce this impact, the group containing the facts for the conditional and non-conditional commits should be divided again to isolate the conditional commit facts from the non-conditional commit measures. In cases where the commit group contains multiple conditional commits which use different conditions, each distinct condition should be assigned to its own group.



Subdivision of Group Due to Conditional Commits

Once these steps have been performed, the initial set of facts should now be partitioned into the set of fact groups that should be optimized for processes that write data into the RPAS Data Mart.



Final Assigned Fact Groups for Facts

A

Appendix : Global Domain Technical Information

Appendix A: Global Domain Technical Information

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 subdomains that contain a subset of the data that would have been in a simple domain. Each of the subdomains contains a subset of one of the hierarchies of the global domain. More specifically, the subdomains 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. An administrator 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, specify (in the path) the entire path to the master domain and the subdomains, 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 subdomains must also exist unless the subdomains 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. Do not delete this directory or file.
- Do not delete the tmp directory under the domain home directory while the domain build process is taking place.
- To configure the global domain functionality, provide a globaldomainconfig.xml file that specifies how each of the local domains must 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 in the location, 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, measures, rules, and workbook templates can be changed, and the master domain and local domains will be patched accordingly. A non-global domain implementation CANNOT be updated to 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 its status has been changed.
- 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 none of the parameters that were used during the original domain build are changed with the exception of replacing `fullinstall` with `patchinstall`. 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.
- For Fusion Client use, you can define "friendly names" for each domain in the MultiSolutionBundle.properties file by defining solution_id.Domain.master.label and solution_id.Domain.domainidx_0, solution_id.Domain.domainidx_1, etc. resources. For more information, see the *Oracle Retail Predictive Application Server Administration Guide for the Fusion Client*.

B

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 performed 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 performed using a variety of methods. Aggregation and spreading are basic capabilities of the engine that require no coding by implementation personnel, other than the selection of aggregation and spreading types to use for a measure.

The inherent relationships between measures can be modeled through 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, while in complete control of the calculation process. It enforces data integrity 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 must 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 must 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.

Note the following about data types:

- Integer measures have a range of 2,147,483,648 to 2,147,483,647 which is four bytes.
  [-2147483647:2147483647]
- Real measures have a range of 1.7E +/- 308 (15 digits) which is eight bytes.
  [-1.7976931348623e+308:1.7976931348623e+308]
- When running printMeasure, it gives the range of the measure. However, internally in the arrays, the integer and real data are stored as Numeric type which is eight bytes long.
- The calculations always happen as double. Internally, the calculation always happens on an eight-byte long number.
- The scientific representation of numbers is only for display and is not involved during calculation. So there should not be any loss in data.

When used in the client and with exportMeasure and loadMeasure, the following was observed:

- You can enter a number with more than 15 digits. But once you finish editing that cell, the number is displayed as 7.777778e+036.
- For such large numbers, the position of the decimal cannot be changed to point to any other position. It will always be displayed as above.
- For printArray, it also displays in the above format.
- When you load a measure with large values, loadMeasure stores the data in the above format.
- When you export the measure using exportMeasure, you can specify the format in which you want to export the data. By default, it exports in the above format. For example, the following exports in the format you specify.

  exportMeasure -d . -intx str_sku_week -out MyOut1.dat -meas "R_EX_DEMOA.format("%13.2f")"

  77777777777777781888888888888888888.00
- In all the above cases, there is no truncation directly, but it will be rounded off to the correct precision.

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 (refer to Aggregation for more information) 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 to the base intersection (see the Spreading section for more information). Depending upon the desired characteristics of the measure, there may be several valid allowed spreading types.

Aggregation

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.

A complete list of the aggregation types supported by the RPAS calculation engine can be found in Appendix D: Aggregation and Spread Types.

Spreading

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 performed 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."

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 reimposed (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, such as recalc type measures, are not usually spread at all (see "The Spreading of Recalc Type Measures" below), but they may employ the replicate spreading technique through the use of rapid entry. For more information about rapid entry, see the "REPD Functionality" section in the *Oracle Retail Predictive Application Server User Guide for the Classic Client*. 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 "naval"), 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 value**s – 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 as follows:

- 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 allowed 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.

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 allowed 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 as follows:

- 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 allowed 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 implemented 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 performed, 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 performed. 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.

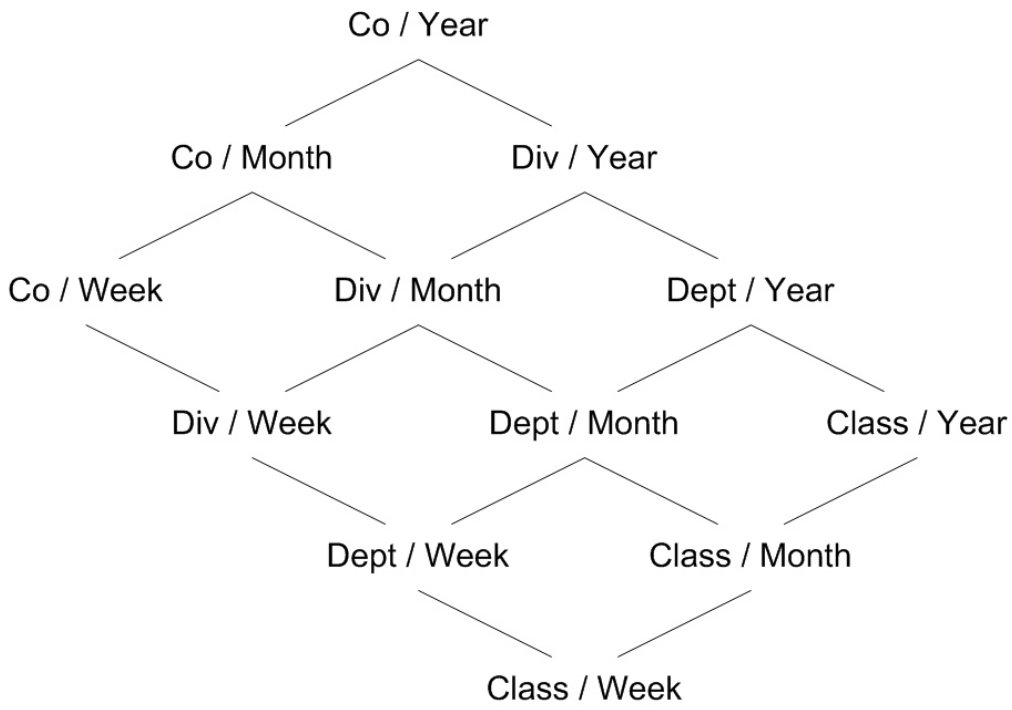
Hierarchical Protection Processing

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 reside 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 manipulated 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 is manipulated 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 must 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 manipulated. If there is just a single expression in the rule that calculates the recalc measures, the recalc measure is non-manipulated through normal protection processing.

Non-Conforming Recalc Measures

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, in these circumstances expressions must be written such that the right-hand side of the expression should have a "recalc expression" rather than a recalc measure. See the following section on “Non-Conforming Expressions” for more information.

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 do 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 refer to 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 provided 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 must 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 there is a change to a 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 must 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 in The Calculation Cycle topic.

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 topic.

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 sub-set 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 *Oracle Retail Predictive Application Server User Guide* and the *Oracle Retail Predictive Application Server Administrator Guid*e 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

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).

Protection Processing Example

The following example illustrates the evaluation of protection processing. For purposes of this example, consider the following set of rules:

Rule 1:

1. A = B + C
2. B = A - C

Rule 2:

1. D = E + A
2. E = D – A

Rule 3:

1. H = F + G

For this set of rules, assume a user edited or locked measure B. Upon evaluation of the protection processing process, the following would occur:

1. B becomes an affected measure.
2. Rule 1 becomes an affected rule.
3. The expression A = B + C in Rule 1 is a free expression that calculates the free measure A.
4. Because Rule 1 has only one free expression, it becomes a forced rule.
5. A becomes a forced measure and therefore an affected measure.
6. Rule 2 becomes an affected rule because it contains the affected measure A.
7. Because Rule 2 contains two free expressions, it does not at this time become a forced rule.
8. Because Rule 3 contains a single expression and because the measure calculated by that expression cannot be mapped back to the right-hand side measures, measure H is protected by the calc engine and cannot be edited.

And so, at the conclusion of evaluating protection processing for the given set of rules, the states of the measures is as follows:

- B is edited.
- A is forced and therefore, protected.
- C is not protected and can be edited.
- D and E are free measures of an affected rule. Either can be edited but editing one will cause the other to be forced and, therefore, to be protected.
- F and G unaffected measures and therefore can be edited.
- H is protected as the calc engine cannot resolve changes to the measure's values.

At this point, the calc engine would begin to calculate knock-on affects based on the protections of B, A and H. These knock-on affects could result in the forcing and protection of additional measures. The process will be evaluated iteratively until all the knock-on affects of the original edit have been processed.

To provide an example of how protection processing can force measures outside the scope of the triggering rule, consider the case where Rule 1 and Rule 2 are unchanged but the expression of Rule 3 is instead:

Rule 3:

A = F + G

In this scenario, protection processing causes measure A to be protected because changes to A cannot be resolved against measures F and G. Furthermore, measure B also becomes protected, as changes to it would cause A to be an affected measure. In this case the state of the measures, before any edits by the user, will be:

- A is protected.
- B is protected as changes to it would force an update to A.
- C is not protected and can be edited.
- D and E are both unaffected and can be edited (the presence of the protected measure A on the right-hand side of an expression does not cause them to be protected).
- F and G are both unaffected and can be edited. Because measure B is protected, the calc engine can resolve changes to F or G by making B a forced measure.

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 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

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), calculation proceeds 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 = if(current == first, OS, 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

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:

1. 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.
2. 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.
3. 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.
4. 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 synchronized 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 = if(current == first, OS, 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.

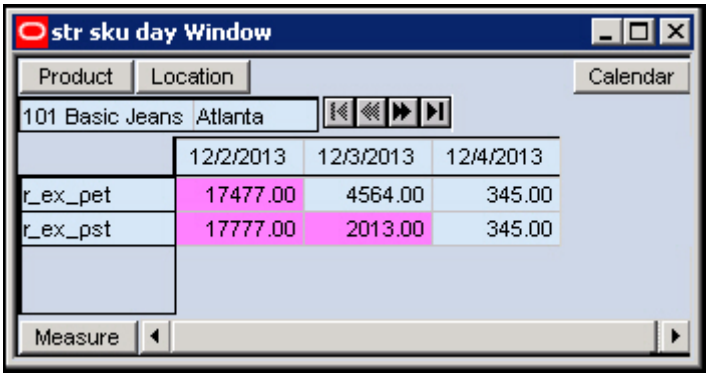
In this example, a CS measure is not required for calculation purposes, but it may be desired for viewing and editing purposes. For example, a window that contains only OS and CS but not BOP nor EOP may be wanted. In this case, the CS measure should be defined as a synchronized measure with type *sync_last* 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 Period Locking

Many planning and prediction applications will cover a time horizon where some of the time periods are in the past (i.e., have elapsed), and others are in the future. RPAS assumes that time periods that have elapsed contain actuals, and that these actuals should not be editable. Therefore, all measures are rendered un-editable during elapsed periods. For positions at aggregated levels in a time hierarchy, the position is considered elapsed when the last lowest level time period descended from it has elapsed.

The RPAS Calculation Engine has special logic for handling elapsed time. Apart from being un-editable in the user interface, spreading never spreads a value to an elapsed cell (for more information, please see the previous Locks and Spreading Around Locke and Changed Cells section).

Measures that represent beginning of period (BOP) data have special handling. From a business perspective, the BOP in a period is the same as the end of period (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 Period Start Total (PST) (for more information, please see the previous Spreading Methods section), or with their measure property Period Start Value set to TRUE are assumed to be "BOP type" measures, and are protected for all elapsed periods, and for the first non-elapsed period. The following example worksheet demonstrates a situation in which the elapsed threshold has been set to 12/2/2013. The pink-colored cells have been set to read-only by RPAS in order to honor elapsed period locking. In this example, the measure e_ex_pet is a regular measure, whereas, r_es_pst is a BOP type measure.



Elapsed Period Locks for BOP and Non-BOP Measures

There is also special handling of BOP type 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.

To set up elapsed period locking in a workbook, workbook designers should set the elapsed time threshold in the load rule of the workbook using the elapsed keyword (for more information, please see the Functional Keywords section in Appendix C: Rules and Functions Reference Guide). If the elapsed time threshold is not set, elapsed period locking will not be available in the workbook.

Example:

If the you want to setup elapsed threshold to today, you would first create a one-dimensional measure, pDay for example, with its intersection at the Day level of the Calendar hierarchy. Then, you would set up a rule like the one shown below to initialize this measure with the index of today.

```
pDay = prefer (today-1, if (now>end, last, -1))
```

You would then aggregate this measure using the PST aggregation method to set the elapsed time threshold as shown in the following rule.

```
elapsed = pDay.pst
```

Setting up the elapsed threshold in the load rule fixes the threshold for the life of the workbook; however, in-season planning applications may require the elapsed threshold to change during the lifetime of a workbook. To achieve this, you can reset the elapsed threshold in a Refresh rule-group or in the Calc rule-group using rules exemplified in the preceding discussion. RPAS inspects the value of threshold after execution of these rule-groups and immediately adjusts the elapsed period locks in the UI. Note that since elapsed threshold is evaluated and executed after the execution of these rule-groups, any spreading performed in the Calc cycle itself would use the state of elapsed threshold before the rule-group was invoked.

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:

Consider the following values:

- 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:

Consider the following values:

- 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:

Consider the following values:

- 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.

C

Appendix: Rules Function Reference Guide

Overview

This section provides the syntax and design of functions, procedures, modifiers, and keywords that are used in expressions in the RPAS calculation engine. There are important distinctions between each of these definitions.

Functions

Functions are separated into two type: single result functions and multiple result functions.

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.

Multiple result functions – Similar to the features and behavior of single result functions, but with 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 multiple 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 multiple result functions.
- Multiple result functions cannot be part of a cycle group.

Procedures

Procedures are 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.
- Procedures can have multiple arguments on the left and right hand sides.
- Procedures cannot be used with functions, other procedures, keywords, and certain modifiers.
- Because of their flexibility and the control available to the developer, procedures can be used for a wide variety of special calculations and activities.
- Procedures require a different syntax. The syntax uses “<-“ instead of “=” in the expression.

Modifiers

Modifiers directly modify the source or destination of measures, to override the level, aggregation type, position, and so on.

Modifier Syntax

```
<measure>.<modifier>
```

Keywords

Keywords appear in expressions or as arguments inside functions to return specific data values.

Syntax Conventions

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; however, they are positional, so that the absence of a parameter needs to be specified by commas if a subsequent parameter is supplied.

The table below displays the syntax conventions used in this procedure.

Indicator	Definition
[…]	All options listed in brackets are optional.
{…|…}	Options listed in "{}" with "|" separators are mutually exclusive (either/or).
{…,…}	Options listed in "{}" with "," separators way are a complete set.
Bold	Labels.
Italics	Italics indicate a temporary placeholder for a constant or a measure.
Italics/meas	This indicates that the placeholder can be either a constant or a measure.
Bold Italics	This indicates a numeric placeholder for the dynamic portion of a label. Usually a number from 1 to N.
Normal	Normal text signifies required information.
Underlined	This convention is used to identify the function or procedure name.

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 is used to designate a hierarchy and dimension:

Hierarchy, Dimension, and Position Syntax

[<*hierarchy*>].[<*dimension*>].[[<*position*>]]

> **Note:** position is noted as optional because it can only be specified in a limited number of functions.

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 have standard validation rules whereby [<hierarchy>] must be a valid hierarchy name, [<dimension>] must be a valid dimension in [<hierarchy>], and [<position>] must be a valid position name in [<dimension>]. If the position name starts with a number, the position name must be nested in a pair of double quotes. 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 multiple result expression are comma-separated and can be identified by a labeling mechanism.

Label Syntax

<measure>:<label>

Valid label names are specified by the multiple result function syntax. If a multiple result function specifies valid labels, the function can be used in an expression without specifying all possible results. The multiple 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

Error Handling

There are several keywords and functions that have special control flow over the evaluation of the expression.

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 that is used to indicate that the entire expression is not to be evaluated (that is, 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`**) 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)` 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 (that is, 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)` - This example 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)` - This example 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:** The following examples all use the simple expression a = b + c.

Example 1

Consider the following scenario:

- “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 using 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.level([loc].[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

Consider the following scenario:

- "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 SKU/Week, and then spread down to SKU/Store/Week, using the existing store participations to the measure a.

Example 3

Consider the following scenario:

- "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 is 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. When used on the RHS of a expression, it returns the index number of the period that is the last elapsed period for the level of calendar hierarchy at which the calculation takes place. When used on the LHS, this sets the last elapsed period along the base dimension of the Calendar hierarchy to the given index number, that is, it is assumed that the index number used to set elapsed is along the base dimension of the hierarchy. If there is no elapsed period, this keyword returns –1. Furthermore, when used on the RHS of an expression whose LHS does not have the Calendar hierarchy, this keyword returns -1.

elapsed must be assigned a value corresponding to the index of the last elapsed period before it can be used in calculations (on the right hand side of other rule groups). This assignment can happen in load, refresh, or calc rule-groups but not in commit 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.

For example, to update elapsed to always correspond to today:

1. Add a new single dimensional measure with base intersection at day.
   a. Call this measure pDay.
2. In the Calc rule group or in the Load and Refresh rule group.
   a. Add rules that initialize the new measure with the calendar index of today, as shown in the example below.

      ```
      pDay=prefer(today-1,if(now>end,last,-1))
      ```

   b. b. Add a new rule that sets the elapsed measure:

      ```
      elapsed=pDay.pst
      ```

The examples below illustrate the behavior of the keyword. For these examples, assume that RPAS, using the mechanism above, has calculated that the index for the last period along the base dimension (Day) to be 30 and that in the Calendar hierarchy, this day rolls up to week with index 4 and month with index 2. In this scenario, the following would occur:

- "LHS_week = elapsed" puts a value of 4 in the LHS_week measure which is along the Week dimension.
- "LHS_day = elapsed" puts a value of 30 in the LHS_day measure which is along the day dimension.
- "LHS_sku_str_day = elapsed" puts a value of 30 in the LHS_sku_str_day measure whose calendar hierarchy is along the Day dimension.
- "LHS = elapsed" changes the navalue of LHS to -1, where the LHS measure is a scalar. Since the measure does not have a Calendar hierarchy, elapsed cannot determine the period index that this measure needs. In order to get a scalar populated, declare a measure along the level at which the elapsed index is required, assign it the elapsed keyword, and then assign this measure to the scalar with an aggregate of PST, PET, or AMBIG.

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** is 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.

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.

username

Returns a string that contains the account name 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 MeasureStore 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 MeasureStore 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 manner displayed under Syntax:

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 that 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 it can be used with functions and other modifiers. When used with **level** and/or **master** modifiers, **aggtype** must be the specified last.

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

The following assumptions/restrictions apply to `old`:

- 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.old
```

where `c` is a measure with an aggregation type of recalc, are not allowed. 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.old
```

where `c` is a measure with an aggregation type of recalc, are also not allowed.

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([clnd].[qtr]))
- Returns the start date of the first period in the current time dimension in the current quarter.

- indextostartdate (index([clnd].[week], openweek), [clnd].[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>[ ,[<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 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.

Calendar Calculation Functions

These are functions that return calendar calculations.

addPeriods

This function requires three inputs and generates one output. It produces a Date value output by adding a number of periods specified by a dimension name to an input Date value.

Input

- Date: Input Date value.
- Integer: Number of periods to be added to the input Date specified by 1.
- String: The Dimension Name of the period, such as DAY, MNTH, and so on.

Output

- Date: The input Date plus number of periods.

Example

targetDate = addPeriods(srcDate, 1, "DAY")

> **Note**: If srcDate evaluates to Jan/01/2012, targetDate should be Jan/02/2012.

calendarStart

This function has no input and produces one output. The output is a Date type value specifying the starting date of the current domain's calendar hierarchy. If called in a workbook, it still returns the starting date of the domain's calendar hierarchy, not the first date included in the workbook.

Output

- Date: First date in the current domain’s calendar hierarchy.

Example

targetDate = calendarStart()

dateDiff

This function requires three inputs and generates one output. It calculates the difference of two date values. It returns the difference as an integer value as number of days, months, and so on, depending on a third string type input that specifies the scale of the output as a dimension name.

Input

- Date: First date.
- Date: Second date.
- String: Dimension name for the scale of the diff to be calculated, such as DAY, MNTH, and so on.

Output

- Integer: Number of periods calculated by firstDate - secondDate, in the scale of the dimension name provided.

Example

targetDate = dateDiff(date1, date2, "MNTH")

> **Note**: If date1 is Jan/01/2012, date2 is Jan/01/2011, the resulting targetDate is 12, since the two dates are 12 months apart.

Date

This function requires two inputs and produces one output. It produces a date value based on an input date as a string, and a formatting string.

Input

- String: Input date string
- String: Date formatting string

The date formatting string follows the format of %[variable]%{variable]%[variable]. The specifier options are as follows:

– B: month, full name

– h: month, 3 character abbreviation, such as JAN, FEB, MAR

– Y: 4 digit year

– y: 2 digit year

– m: 2 digit month

– d: 2 digit day

– H: 2 digit hour

– M: 2 digit minute

– S: 2 digit second

– s: 3 digit millisecond

For instance, if the format string evaluates to %Y%m%d and the date string is 20120102, the date is January 02, 2012. If the format string is %Y%h%d%H%M%S and the date string is 2012JAN02073030, the time and date is 7:30:30am on January 02, 2012.

Output

- Date: Date value by parsing the input date string using the input date format.

Example

targetDate = date(dateStr, formatStr)

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 they are typically used as parameters that are passed 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 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 since 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 upper case.

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, which must be specified using quotes:

- "label" – The label (description) for the position. This value must be specified using quotes. The attribute function requires left-hand side measure to be a string measure. All keywords which need to be passed to a function must be wrapped in double quotes. Any other syntax will throw an error.

- "dpmstatus" – The DPM status of the position. This attribute function required left-hand side measure to be a Boolean measure. TRUE value corresponds to an "informal" status. A FALSE value corresponds to a "formal" status.

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("dpmstatus", [prod].[item], likeindex)
- This returns the value of the dpmstatus attribute for the item with the index number in the measure likeindex (that is, the label for my like item).

Forecast Procedure

Using the RPAS Configuration Tools, a time-series demand forecast may be configured as part of a planning workflow or business process. The Forecast procedure provides only a small subset of the functionality that is available through RDF. The differences between these solution extensions are as follows:

- The forecast produced by the Forecast procedure is a single-level forecast.
- RDF allows for forecasts to be generated at aggregate levels in the data (to remove sparsity), and then this forecast is spread down to the execution level by using a profile.
- The Forecast procedure allows for a single forecasting method to be specified in the calculation of the forecast.
- RDF allows for forecasting methods and forecasting parameters to be modified as needed at all levels in your data.
- No standard approval process of the resulting forecasts are included as part of the Forecast procedure.
- RDF allows for forecast adjustments and approvals to be made at the lowest level necessary in your data.

The "Forecast Procedure Syntax" section contains the specifications and syntax for configuring the Forecast procedure.

Forecast Requirements

The following libraries must be registered in any domain(s) that will use the Forecast solution extension:

- AppFunctions
- RdfFunctions

Using the Forecast Procedure

The following notes are intended to serve as a guide for configuring the Forecast procedure within the RPAS Configuration Tools.

- Refer to the appropriate input parameters and output measures when using the Forecast procedure.

- The resultant measure (that is, the forecast output) should be at the same intersection as your history measure (that is, pos). This will be the base intersection of the final level.
- The Forecast procedure is a multiple result procedure, meaning that it can return multiple results with one procedure call within a rule. In order to get multiple results, the resultant measures must be configured in the Measure Tool and the specific measure label must be used on the left-hand side (LHS) of the procedure call. The resultant measure parameters must be comma-separated in the procedural call.

Syntax Conventions

The table below displays the syntax conventions used in this procedure.

Indicator	Definition
[…]	All options listed in brackets are optional.
{…|…}	Options listed in "{}" with "|" separators are mutually exclusive (either/or).
{…,…}	Options listed in "{}" with "," separators way are a complete set.
Bold	Labels.
Italics	Italics indicate a temporary placeholder for a constant or a measure.
Italics/meas	This indicates that the placeholder can be either a constant or a measure.
BoldItailics	This indicates a numeric placeholder for the dynamic portion of a label. Usually a number from 1 to N.
Normal	Normal text signifies required information.
Underlined	This convention is used to identify the function or procedure name.

Forecast Procedure Syntax

The syntax for using the Forecast procedure appears below. The example below is a simplified syntax version of the Forecast procedure. For the complete syntax version, refer to the *RDF Configuration Guide*. The input and output parameter tables explain the specific usage of the parameters names use in the procedure.

Generic Example:

```
FORECAST: FORMEAS , PEAKS:PEAKSMEAS, CHMETHOD:METHMEAS<-FORECAST(MASK:MEASKMEAS,
{STARTDATE:STARTDATE | STARTDATEMEAS:STARTDATEMEAS}, HISTORY: HISTORYMEAS,
FORECASTLENGTH:FORECASTLENGTH, PERIOD:PERIOD ,{FRCSTSTARTMEAS:FRCSTSTARTMEAS |
FRCSTSTART:FRCSTSTART}, PLAN:PLAN, PROFILE:PROFILE,  BAYESIAN_HORIZ:BAYESIAN_HORIZ,
{VALID_DD:VALID_DD, DDPROFILE:DDPROFILE })
```

Sample:

forecast:frcstout,cumint:cumintout,int:intout<-Forecast(forecastlength:12,history:pos,mask:frcstmask,period:26,startdatemeas:todaymeas)

Configuration Parameters and Rules

Input Parameters

The table below provides the input parameters for the Forecast procedure.

Parameter Name	Description
FORECASTLENGTH	The length of the forecast. Data Type: Integer Multiple Allowed: No Required: Yes
HISTORY	The input measure the forecast is based on. Data Type: Real Multiple Allowed: No Required: Yes
MASK	Array that identifies what forecast method is used for each time series. Refer to Forecast Model/Model List table. Data Type: Boolean Multiple Allowed: No Required: Yes
MAXALPHA	The maximum alpha value. Data Type: Real Multiple Allowed: No Required: No
PERIOD	The forecasting period for calculating seasonal coefficients. Data Type: Integer Multiple Allowed: No Required: Yes
PLAN	The Plan measure. Data Type: Real Multiple Allowed: No Required: No
PROFILE	The Seasonal Profile measure. Data Type: Real Multiple Allowed: No Required: No
STARTDATE/ STARTDATEMEAS	The forecast start date. Either STARTDATE or STARTDATEMEAS is required. STARTDATEMEAS, if used, must be a scalar for AutoES method. Data Type: STARTDATE - Date as a string. Data Type: STARTDATEMEAS – Date as measure. Multiple Allowed: No Required: Yes

Parameter Name	Description
VALID_DD	The maximum non-zero history to use de-seasonalized demand value for seasonal profile based forecasting. Data Type: Integer Multiple Allowed: No Required: No
DDPROFILE	De-seasonalized demand measure. Used only for profile-based forecasting. Data Type: Double Multiple Allowed: No Required: No

Output Parameters

The table below provides the output parameters for the Forecast procedure.

Parameter Name	Description
CHMETHOD	Selected method. Refer to Forecast Model/Model List table. Data Type: Integer Multiple Allowed: No Required: No
FORECAST	Forecast output. Data Type: Real Multiple Allowed: No Required: Yes
PEAKS	Peaks, which are used for calculating baseline of the forecast. Data Type: Real Multiple Allowed: No Required: No

Forecast Method/Model List

The table below provides the numeric value assigned to the forecast model/model list.

Model	Numeric Value
AUTO ES	1
SIMPLE	2
HOLT	3
WINTERS	4
CASUAL	5
AVERAGE	6
NO FORECAST	7

Model	Numeric Value
COPY	8
CROSTON	9
M. WINTERS	10
A. WINTERS	11
SIMPLE CROSTON	12
BAYESIAN	13
LOADPLAN	14
PROFILE	15

Time Series Functions

Overview

This is a collection of very similar functions to perform typical calculation tasks over a range of cells in one or more time series. The <start> and <end> positions, defined using calendar index numbers, specifies 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 effectively be specified 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.

tssum is used for the following types of calculations:

- Season to date
- Balance to achieve
- 4 week moving sum

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** (for instance, 0). If <end> is not specified, the default value is current.

If there is more than one value for the mode, then the function 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

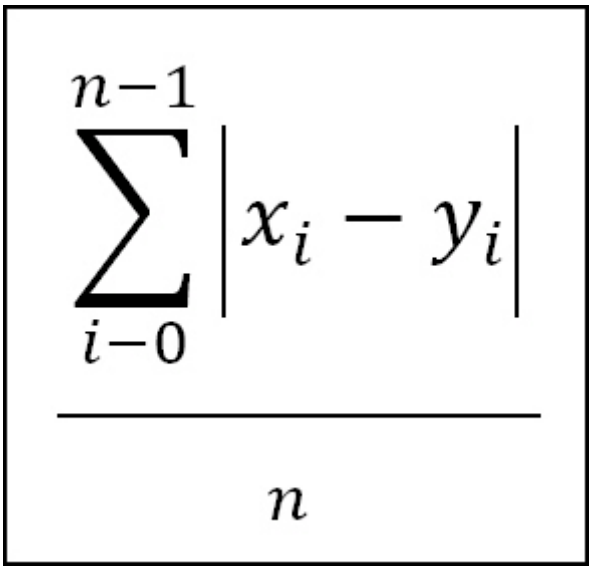
Produces the Mean Error of an 'estimate' time series compared to an 'actuals' time series.

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, and 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 following formula:



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.

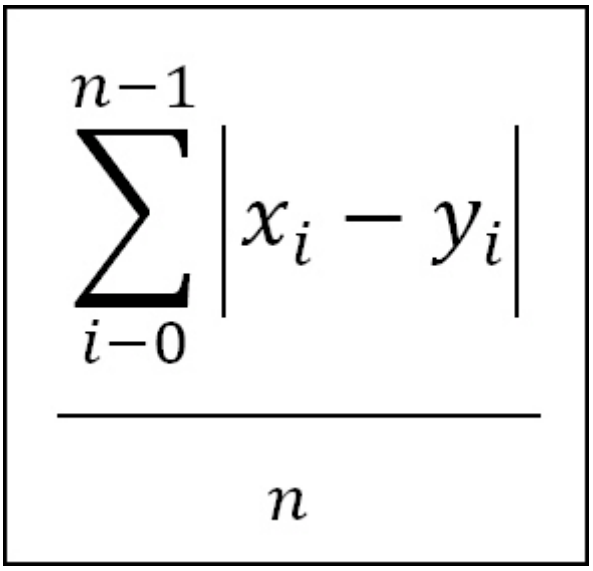
Syntax

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** (that is, 0). If <end> is not specified, the default value is **current**.

The Mean Absolute error is calculated using the following formula:



$$\frac{\sum_{i-0}^{n-1} \left| x_i - y_i \right|}{n}$$

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, and 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**. The Mean Absolute Percentage error is calculated using the following formula:

$$\frac{\sum_{i \in \{0 \le i < n \cap y_i \neq 0\}} \left| \frac{x_i - y_i}{y_i} \right|}{\sum_{i \in \{0 \le i < n \cap y_i \neq 0\}} 1}$$

Inverse

The **tsmape** function does not have an inverse.

tsrmse

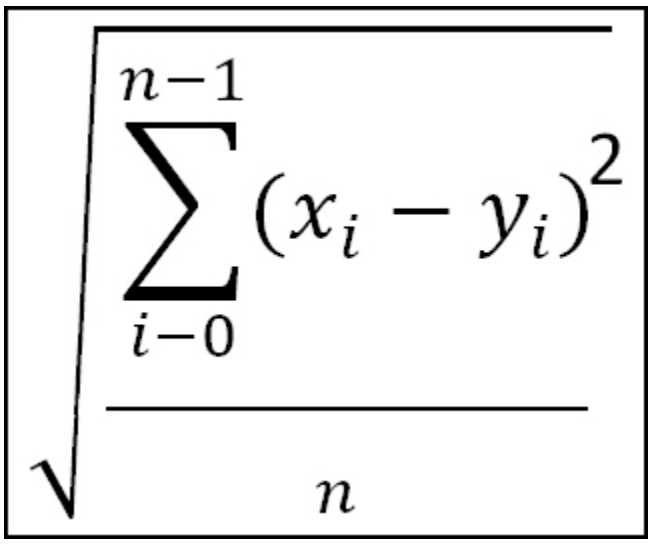
Root Mean Square Error.

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, and 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 Root Mean Square error is calculated using the following formula:



Inverse

The **tsrmse** function does not have an inverse.

tspae

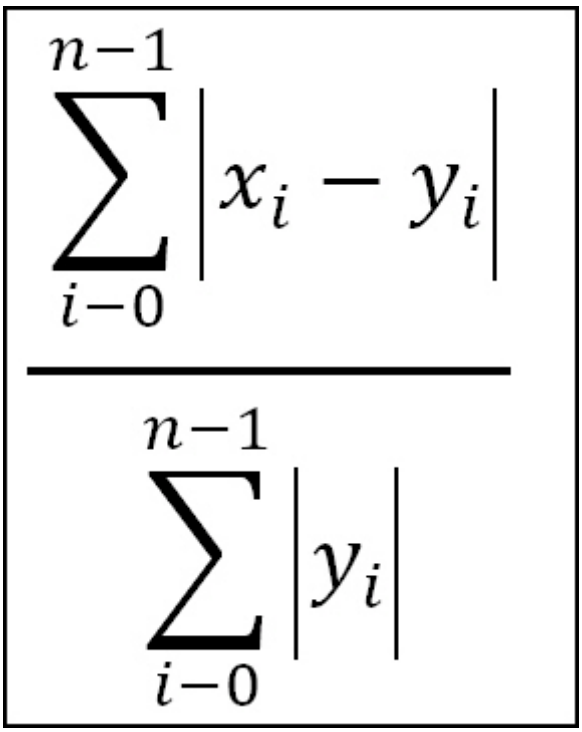
Percentage Absolute Error.

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, and 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 Percentage Absolute error is calculated using the following formula:



Inverse

The **tspae** function does not have an inverse.

Hierarchical Functions and Procedures

Overview

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 the following special uses and restrictions:

- `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.

> **Note:**The usage of the tablelookup special expression requires the arguments to be conformed in certain way. For example in the tablelookup expression (valueMeasure, nearest, keyMeasure), it is required to address the following:

1. The keyMeasure's baseint must have a 'table entry' dimension, and this table entry dimension must be the innermost. Let's assume the keyMeasure is at clss/rgn/te level where dimension 'te' is the table entry dimension, and it is the innermost. The 'subspace' of the key entry is defined as the baseint of the key measure minus the table entry dimension; in this case, the subspace will be 'clss/rgn'.
2. The valueMeasure's baseint must be able to map to the subspace (clss/rgn) with a many to one mapping, that is, the baseint of the valueMeasure must be below 'clss/rgn'. So it is a valid expression if the valueMeasure's baseint is 'sku/str', 'clss/str', 'sku/rgn'. If the valueMeasure's baseint is the same or higher than clss/rgn, or cannot create a valid map to clss/rgn, the above error message will be triggered. For example, below baseint for valueMeasure will trigger the error: clss/rgn, dept/rgn, clss/chn, dept/chn, dept/str, sku/chn.

   It is suggested to examine the baseint of the input measure with the above criteria in mind.

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 follows:

$$lowresult + \frac{(highresult - lowresult) * (keyvalue - lowvalue)}{(highvalue - lowvalue)}$$

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 look up function that 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

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, [clnd].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, [clnd].average)
- Totals up all hierarchies except time, which uses an average aggregation type.
- result <- aggregate(x, [prod].average, [clnd].first)
- 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, [clnd].last)
- Averages up the product hierarchy if possible. Otherwise, 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, [clnd].first)`. The measure is assumed to have a base intersection of Class/Store/Week.

Examples:

Examples of the calculations that would be applied at various levels are as follows:

- Class/Store/Month: first from Class/Store/Week
- Class/Region/Month: avg from Class/Store/Month
- Department/Region/Month: total from Class/Region/Month

Transform Procedures

RPAS offers the following transformation procedures:

- transformSum
- transformMax
- transformOr
- transformProp
- transformEven
- transformRepl

Transform Procedure Requirements

The following libraries must be registered in any domain(s) that will use the transform procedures:

- Transform

Example:

```
regfunction -d <pathToDomain> -l Transform -add
```

transformSum

transformSum converts data across hierarchies using sum aggregation. The procedure converts data between measures of different dimensionality using a set of map measures to convert positions from the source measure to positions in the target measure. Source measures are aggregated into the target using the sum aggregation method.

Syntax

```
<target> <- transformSum(<source>, <transformspec1> [, <transformspec2> … ,
<transformspecn>])
```

Input Parameters

The table below provides the input parameters for the transformSum procedure.

Parameter Name	Description
source	Measure that is being aggregated into <target> measure using the aggregation type of **sum**.
transformspec1-n	This parameters is [<source hierarchy>].[<source dimension>], [<target hierarchy>].[<target dimension>] , [LABEL | POSNAME], <map> The <transformspec> defines which dimension in the source is mapped to which dimension in the target and how the positions are mapped between the dimensions. The <map> measure may either be text or Boolean. If it is text then the value of the cell contains the position id or label name of a position in the target dimension. The compulsory [LABEL | POSNAME] parameter specifies which method is used. If the <map> measure is Boolean then its base intersection must include the <source dimension>; any true cells in the map measure will define the positions that are transformed to the target. If a label is not unique within a dimension and the LABEL option is used, then only the first position in the dimension that includes the label will be part of the transformation.

Output Parameters

The table below provides the output parameter for the transformSum procedure.

Parameter Name	Description
target	Measure into which the <source> measure is aggregated into using the aggregation type of **sum**.

Notes

If a hierarchy is in both the source and target measures, then the dimension for that hierarchy in the source and target must be the same, unless the transformation is defined through a mapping *transformspec*, meaning the source measure cannot have a base intersection of item if the target measure has a base intersection of class and there is no explicit transformation specified from item to class in *transformspec*.

If a dimension in the target is not in the source and is also not defined by a mapping, then transformation is applied to every position in that dimension.

transformSum only works for numeric measures. Text or Boolean measures will not get transformed.

If a cell in the source cannot be mapped to a position in the target then it is ignored. The `Transform` procedure always writes a status message to rpas.log indicating how many cells were successfully transformed, how many cells failed and how many seconds the transformation took to execute.

T he source measure and any map measure may be non-conforming. For instance, the source may be defined at month and the map defined at season.

Example:

```
WpVRSlsR <-transformSum(WpSlsR, [LOC].[STR], [DVR].[VR], LABEL, WpRankTx)
```

Takes WpSlsR (store/class/month) and transforms it to WpVRSlsR (volume rank/class/month) using label mappings defined in WpRankTx (store/class/season).

transformMax

The `transformMax` procedure converts data across hierarchies using **max** aggregation. The procedure operates in the same way as `transformSum`, except that the aggregation method used is **max**.

Syntax

<target> <- transformMax(<source>, <transformspec1> [, <transformspec2> … , <transformspecn>])

Input Parameters

The table below provides the input parameters for the `transformMax` procedure.

Parameter Name	Description
source	Measure that is being aggregated into <target> measure using the aggregation type of max.
transformspec1-n	This parameter is [<hierarchy>].[<dimension>], [<hierarchy>].[<dimension>] , [LABEL|POSNAME], <map>

Output Parameters

The table below provides the output parameter for the transformMax procedure.

Parameter Name	Description
target	Measure into which the <source> measure is aggregated into using the aggregation type of max.

Example:

r_ut_out<-transformMax(r_ut_in, [prod].[sku], [clnd].[week], 0, r_ut_map)

Takes r_ut_in (sku/str/day) and transforms it to r_ut_out (sku/str/week) using label mappings defined in r_ut_map (sku/str). Here the maximum across all the days of a week is taken from r_ut_in and stored in the r_ut_out measure using label mappings defined in r_ut_map (sku/str).

transformOr

The `transformOr` procedure converts data across hierarchies using **or** aggregation. The procedure operates in the same way as `transformSum`, except that the aggregation method used is **or**. Both source and target measures must be Boolean measure types.

Syntax

<target> <- transformOr(<source>, <transformspec1> [, <transformspec2> … , <transformspecn>])

Input Parameters

The table below provides the input parameters for the `transformOr` procedure.

Parameter Name	Description
Acsource	Measure that is being aggregated into <target> measure using the aggregation type of or. Must be a Boolean measure type.
transformspec1-n	This parameter is [<hierarchy>].[<dimension>], [<hierarchy>].[<dimension>] , [LABEL|POSNAME], <map>

Output Parameters

The table below provides the output parameter for the `transformOr` procedure.

Parameter Name	Description
Target	Measure into which the <source> measure is aggregated into using the aggregation type of or. Must be a Boolean measure type.

Example:

r_ut_out<-transformOr(r_ut_in, [prod].[sku], [clnd].[week], 0, r_ut_map)

Takes r_ut_in (sku/str/day) and transforms it to r_ut_out (sku/str/week) using label mappings defined in r_ut_map (sku/str). Here the Boolean OR across all the days of a week is taken from r_ut_in and stored in the r_ut_out measure using label mappings defined in r_ut_map (sku/str).

transformProp

The `transformProp` procedure converts data across hierarchies using Proportional spreading. The procedure converts data between measures of different dimensionality using a set of map measures to convert positions from the source measure to positions in the target measure. While the `transformSum` procedure (and related aggregation procedures) assumes a many->one relationship as it performs the transformation (aggregation), the `transformProp` assumes a one->many relationship between source and target cells (spreading).

Each source value is spread to a set of target values, leaving the ratio between the target values in tact.

If the sum of all target cells is zero, then the source is spread evenly to the targets.

Syntax

```
<target> <- transformProp(<source>, <transformspec1> [, <transformspec2> … , <transformspecn>])
```

Input Parameters

The table below provides the input parameters for the `transformProp` procedure.

Parameter Name	Description
source	Measure that is being spread into <target> measure.
transformspec1-n	This parameter is [<hierarchy>].[<dimension>], [<hierarchy>].[<dimension>] , [LABEL|POSNAME], <map>

Output Parameters

The table below provides the output parameter for the `transformProp` procedure.

Parameter Name	Description
`target`	Measure into which the `<source>` measure is spread.

Note

If the `<source>` measure has a calendar dimension, then the r_elapsed measure has to have a value. (This is true for all TransformSpread flavors: `transformProp`, `transformRepl`, `transformEven`)

Example:

```
mace -d . -run -expression "r_ut_out <- transformProp(r_ut_in, [clnd].[day], [loc]
.[str], 0, r_ut_map)
```

transformEven

The transformEven procedure converts data across hierarchies using Even spreading.

Syntax

<target> <- transformEven(<source>, <transformspec1> [, <transformspec2> … , <transformspecn>])

Input Parameters

The table below provides the input parameters for the `transformEven` procedure.

Parameter Name	Description
source	Measure that is being spread into <target> measure.
transformspec1-n	This parameter is [<hierarchy>].[<dimension>], [<hierarchy>].[<dimension>] , [LABEL|POSNAME], <map>

Output Parameters

The table below provides the output parameter for the `transformEven` procedure.

Parameter Name	Description
target	Measure into which the <source> measure is spread.

Note

If the `<source>` measure has a calendar dimension, then the r_elapsed measure has to have a value. (This is true for all TransformSpread flavors: `transformProp`, `transformRepl`, `transformEven`)

Example:

```
mace -d . -run -expression "r_ut_out <- transformEven(r_ut_in, [clnd].[day], [loc]
.[str], 0, r_ut_map)"
```

transformRepl

The `transformRepl` procedure converts data across hierarchies using Replicate spreading.

Syntax

```
<target> <- transformRepl(<source>, <transformspec1> [, <transformspec2> … ,
<transformspecn>])
```

Input Parameters

The table below provides the input parameters for the `transformRepl` procedure.

Parameter Name	Description
source	Measure that is being spread into `<target>` measure.
transformspec1-n	This parameter is `[<hierarchy>].[<dimension>], [<hierarchy>].[<dimension>] , [LABEL|POSNAME], <map>`

Output Parameters

The table below provides the output parameter for the `transformRepl` procedure.

Parameter Name	Description
target	Measure into which the `<source>` measure is spread.

Note

If the `<source>` measure has a calendar dimension, then the r_elapsed measure has to have a value. (This is true for all TransformSpread flavors: `transformProp`, `transformRepl`, `transformEven`)

Example:

```
mace -d . -run -expression "r_ut_out <- transformRepl(r_ut_in, [clnd].[day], [loc]
.[str], 0, r_ut_map)"
```

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>*, <dst_start>)

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 its value is a date type, the date value is converted to a calendar index internally. 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.

<dst_start> is an optional input. It can be a date type measure. If so, the date value is converted to a calendar index internally. It can also be a numeric value. If it's a numeric value, it represents the first calendar index that the output time series is written to. If omitted, it is always 0.

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>], <dst_start>)

<*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.
- **resizeprofile**

The "resizeprofile" function is a rewrite of the "resizenorm" function. The resizeprofile function is intended as a functional replacement for the resize and resizenorm functions.

> **Note:** Users are encouraged to replace the expressions containing resize and resizenorm functions with resizeprofile. The resize and resizenorm functions are still maintained for backward compatibility.

Syntax

- resizeprofile(expression, start, fromlength, tolength, dststart: <dststart>, total: <total>)
- 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 its value is a date type, the date value is converted to a calendar index internally. 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. <dststart> is an optional input. It can be a date type measure. If so, the date value is converted to a calendar index internally. It can also be a numeric value. If it is a numeric value, it represents the first calendar index that the output time series is written to. If omitted, it is always 0.
- 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 calculation engine automatically normalizes the resized values to the specified <total>.

The usage text of this function is as follows:

- resizeprofile(expression, start, fromlength, tolength, dst_start: <dst_start>, total: <total>) where:

- **expression**: is a measure or expression, whose time series is to be used in the calculations. This is a required input.
- **start**: denotes the index number of the calendar dimension of expression. The algorithm processes data points from this index number going forward. This is a required input.
- **fromlength**: together with start, this input establishes which portion of the curve should be processed. fromlength determines the length of the original curve. This is a required input.
- **tolength**: determines the length of the output curve. This is a required input.
- **dst_start**: this is a named value pair denoting an optional input, which determines the index of the starting point of the output curve. If not specified, it defaults to zero.
- **total**: this is a named value pair denoting an optional input whose numeric value is used to normalize the resized curve. If total is not specified, skip the 'Normalize' step.

Examples:

- `"MeasureB=resizeProfile(MeasureA, 0, 10, 30, dststart:1)"`
- resizeprofile will take the curve represented by the 10 positions starting at index 0 in MeasureA and resize that curve to fit the 30 positions starting at index 1 in MeasureB.
- `"MeasureD=resizeProfile(MeasureA, 0, 10, 25, dststart:1, total:200)"`
- resizeprofile will take the curve represented by the 10 positions starting at index 0 in MeasureA and resize that curve to fit the 25 positions starting at index 1 in MeasureD. While performing the resize operation, resizeProfile will normalize the curve to a total of 200 across all destination positions.
- `"MeasureD=resizeProfile(MeasureC + MeasureA, 0, 15, 30, dststart:2, total:100)"`
- resizeprofile will take the curve represented by the 15 positions starting at index 0 in the expression "MeasureC + MeasureA" and resize that curve to fit the 30 positions starting at index 2 in MeasureD. While performing the resize operation, resizeProfile will normalize the curve to a total of 100 across all destination positions.

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.

textCompare

Performs a case-sensitive comparison of two strings.

Syntax

`textCompare(<expression1>, <exprssion2>)`

Where the two input arguments <expression1> and <expression2> must have a type of string. The result type is Boolean. The result value is a Boolean value of the case-sensitive comparison of the two RHS (right hand side) expressions.

The number of input arguments is 2. The number of output is 1.

textConcat

Concatenates string arguments.

Syntax

`textConcat(<expression`$_1$`> ,<expression`$_2$`>[,… ,<expression`$_N$`>])`

Where the value of all expressions is either a string type measure or a string literal. Concatenates two or more values into a single string.

substr

Returns a portion of the input String measure's value.

Syntax

`substr(<inputMeasure>[,<startIndex>,<length>])`

Where <inputMeasure> must be a string type measure and the optional <startIndex> and <length> arguments are either integer measures or literal integer values. The substr function will take the portion of the input string beginning at <startIndex> (which defaults to 0 if not specified) and spanning the amount of characters specified by the <length> argument (which defaults to the length of the input string if not specified). If the end index of the copy (calculated as <startIndex> + <length>) is greater than the length of the input string, substr will pad the input string with spaces to make it long enough to be copied.

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.

Other Functions and Procedures

multisource

The multisource procedure calculates a workbook-only recalc measure, based upon the intersection that the measure is currently displayed at. The multisource procedure takes as arguments a list of measures at different base intersections. When evaluating the output measure, the intersection of the current worksheet is used to determine which of the right hand side (RHS) measure's values must be assigned to the LHS measure.

Syntax

`multisource(<inputMeas1>,<inputMeas2>…<inputMeasN>[,<performAggregation>])`

Where `<inputMeas1>` through `<inputMeasN>` are measures at differing base intersections. When the multisource expression is evaluated, the input measure with the appropriate base intersection is used.

The `<performAggregation>` flag is a Boolean value (true or false). When this flag is set to TRUE and there is no input measure at the evaluated intersection, then the closest measure will have its values aggregated based on that measure's aggregation method and those values are used.

Multisource is useful when you need to load and display non-aggregated data at different intersections within a single measure. To achieve this without multisource measures, you must have multiple measures at each of the required intersections. This decreases usability since the user sees many different measures and a large number of invalid cells. The example below shows the number of measures that would be needed to support just the different levels of the product hierarchy. This would be exploded out for each combination of intersections along the different hierarchies.

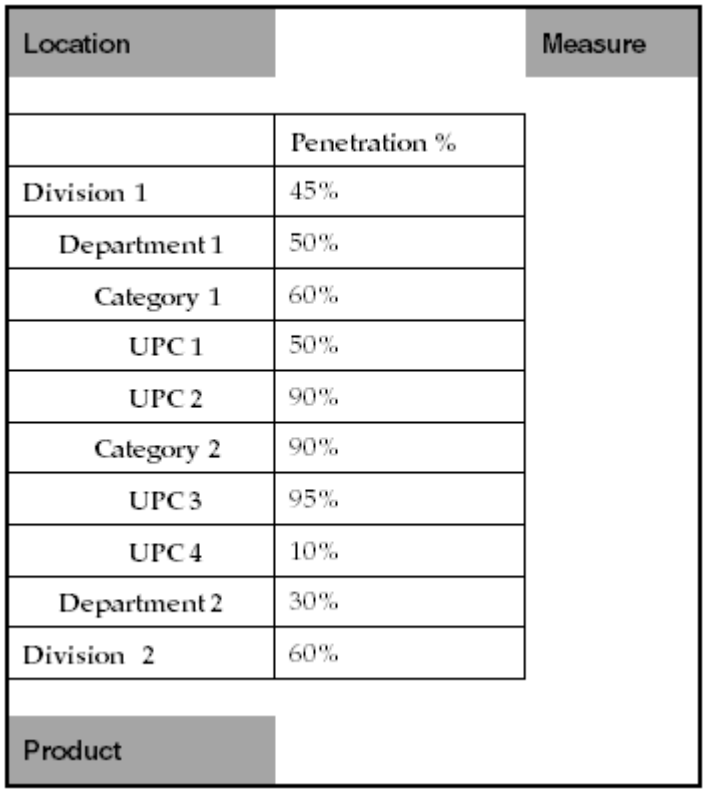
Location — Measure

	Division %	Department %	Category %	UPC %
Division 1	45%	?	?	?
Department 1		50%	?	?
Category 1			60%	?
UPC 1				50%
UPC 2				90%
Category 2			90%	?
UPC 3				95%
UPC 4				10%
Department 2		30%	?	?
Division 2	60%	?	?	?

Product

Without Multisource Measures

However, with the use of multisource measures, the same requirements met by the previous example are achieved with a single measure, the result of which is shown in the figure below.



With Multisource Measures

Left Hand Side (LHS) Measure Properties

LHS measures must have the following attributes and properties:

- **Base intersection**: At or above the worksheet intersection like any other workbook measure.
- **Aggregation Type**: RECLC. (This is recalculated whenever the display intersection changes.)
- **Spread Type**: None (gets recalculated whenever the measure gets spread down)
- **Materialized Type**: Persistent. (This is persisted only in the workbook database. It cannot have a domain database).
- **Base State**: Read only
- **Agg State**: Read only
- **Measure Type**: Numeric, Date, String and Boolean
- The special expression must be part of a calc rule only. This validation check must be exclusively performed in the RPAS Configuration Tools.

LHS Measure Restrictions and Validations

If the expression is triggered as part of a calc rule through workbook and any of the LHS measure properties are not met, then a marshallable exception is thrown by the server. An error message is displayed that describes the cause of exception. After closing the error message dialog, you can either close the workbook normally or continue working on the same workbook without performing any calculation. On the Configuration Tools side, a validation failure results in displaying the context of validation in red text and

prevents the rpaInstall from building the domain. The properties listed above must be validated for the LHS measure on the RPAS Server and Configuration Tools.

Right Hand Side (RHS) Measure Properties

RHS measures must have the following attributes and properties:

- **MeasureType**: RHS measures `inputMeas1, inputMeas2,inputMeas3` are of the same type as the type of the LHS measure.
- **Base Intersection**: RHS measures `inputMeas1, inputMeas2,inputMeas3` must have different base intersections.
- **Min and Max Number of RHS measures**: The minimum number is 1 and maximum number is 200.
- **Scalar measure**: If LHS measure is a scalar, there can be only one RHS measure, and it must be a scalar measure. Since the number of RHS arguments can vary from 1 to 200, the condition when LHS measure is scalar can be met by providing a single RHS measure which is also a scalar.
- Since aggregation of RHS measures is allowed, the RHS measures base intersection can be at or above or below the LHS measure base intersection. Whenever an exact match for the LHS measure display intersection is found on the RHS side, that measure gets used otherwise whichever RHS measure that has the nearest base intersection to the LHS measure's display intersection gets used after aggregation.
- RHS measures can have a domain database as they can be persisted in both domain and workbook.
- The right most argument of RHS must be checked for a constant after the above validation check. If it is a constant, then the remaining RHS arguments must be all measures. There can be only one optional constant in the expression and that must be the right most argument.

RHS Measure Restrictions and Validations

The conditions described above must be validated for the RHS measures on the RPAS Server and Configuration Tools. If the validation fails, a marshallable exception is thrown on the RPAS Server side.

Rule Group Restrictions

The following validation checks must be exclusively performed in the Configuration Tools:

- The multisource expression can be used only in calc rule groups.
- LHS measure cannot appear on the RHS of any expression in any rule group.

cover

The cover 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, **uncover**. uncover 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.

Note: 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.

Make sure 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: from which period to start using the sales expression. 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 must 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

The uncover 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.

Note: 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."

Make sure 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: from which period to start using the sales expression. 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

The min 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

The max 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

The sum 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

The lag 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 previous period.

Note: The **lag** function 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 following two expressions are allowed, and can be calculated in the same rule group, even though EOP appears to depend on BOP, which appears to depend on EOP:

```
EOP = BOP + Rec – SLs – MD
BOP = lag(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 not allowed:

```
AccumSls = Sls + lag(AccumSls)
```

lead

The lead 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.

Example:

- **lead**(BOP)
- Returns the value of the measure BOP from the next period.

Note: The **lead** function 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, 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

The `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 has the following special uses and restrictions:

- 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. The value of each position (called the source position here) in the lagmap measure is the name of position in the destination measure to which the data for that source position in the input is mapped. 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, -4)`
- Updates the positions in the workbook measure for last year's sales with the values from the domain measure sales where each position is advanced by 4 periods.

Input	sales	chnl	scls	week	Value
	-	Catalog	Loafer	w01_2007	10
	-	Catalog	Loafer	w02_2007	20
	-	Catalog	Loafer	w03_2007	30
	-	Catalog	Loafer	w04_2007	40
	-	Catalog	Loafer	w05_2007	0
	-	Catalog	Loafer	w06_2007	0
	-	Catalog	Loafer	w07_2007	0

	-	Catalog	Loafer	w08_2007	0
Output	**salesly**	**chnl**	**scls**	**week**	**Value**
	-	Catalog	Loafer	w01_2007	0
	-	Catalog	Loafer	w02_2007	0
	-	Catalog	Loafer	w03_2007	0
	-	Catalog	Loafer	w04_2007	0
	-	Catalog	Loafer	w05_2007	10
	-	Catalog	Loafer	w06_2007	20
	-	Catalog	Loafer	w07_2007	30
	-	Catalog	Loafer	w08_2007	40

- `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.

Input	**sales**	**chnl**	**scls**	**week**	**Value**
	-	Catalog	Loafer	w01_2007	10
	-	Catalog	Loafer	w02_2007	20
	-	Catalog	Loafer	w03_2007	30
	-	Catalog	Loafer	w04_2007	40
	-	Catalog	Loafer	w05_2007	0
	-	Catalog	Loafer	w06_2007	0
	-	Catalog	Loafer	w07_2007	0
	-	Catalog	Loafer	w08_2007	0
	-	Catalog	Boots	w01_2007	10
	-	Catalog	Boots	w02_2007	20
	-	Catalog	Boots	w03_2007	30
	-	Catalog	Boots	w04_2007	40
	-	Catalog	Boots	w05_2007	0
	-	Catalog	Boots	w06_2007	0
	-	Catalog	Boots	w07_2007	0
	-	Catalog	Boots	w08_2007	0
	saleslag	chnl	scls	-	Value
	-	Catalog	Loafer	-	2
	-	Catalog	Boots	-	-2

Output	salesly	chnl	scls	week	Value
	-	Catalog	Loafer	w01_2007	30
	-	Catalog	Loafer	w02_2007	40
	-	Catalog	Loafer	w03_2007	0
	-	Catalog	Loafer	w04_2007	0
	-	Catalog	Loafer	w05_2007	0
	-	Catalog	Loafer	w06_2007	0
	-	Catalog	Loafer	w07_2007	0
	-	Catalog	Loafer	w08_2007	0
	-	Catalog	Boots	w01_2007	0
	-	Catalog	Boots	w02_2007	0
	-	Catalog	Boots	w03_2007	10
	-	Catalog	Boots	w04_2007	20
	-	Catalog	Boots	w05_2007	30
	-	Catalog	Boots	w06_2007	40
	-	Catalog	Boots	w07_2007	0
	-	Catalog	Boots	w08_2007	0

- `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 position in salesly whose name is cell contents of salesmap receives the corresponding value from sales. In cases where 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 salesmap could contain the position names “w52_2007” and “w53_2007”, causing the value of salesly to contain the sum of the 2 positions.

Input	sales	chnl	scls	week	Value
	-	Catalog	Loafer	w02_2007	20
	-	Catalog	Loafer	w03_2007	30
	-	Catalog	Loafer	w04_2007	40
	-	Catalog	Loafer	w05_2007	0
	-	Catalog	Loafer	w06_2007	0
	-	Catalog	Loafer	w07_2007	0
	-	Catalog	Loafer	w08_2007	0
	salesmap	-	-	week	Value
	-	-	-	w01_2007	w05_2007
	-	-		w02_2007	w06_2007
	-	-		w03_2007	w07_2007

	-	-		w04_2007	w08_2007
	-	-		w05_2007	w01_2007
	-	-		w06_2007	w02_2007
	-	-		w07_2007	w03_2007
	-	-	-	w08_2007	w04_2007
Output	**salesly**	**chnl**	**scls**	**week**	**Value**
	-	Catalog	Loafer	w01_2007	0
	-	Catalog	Loafer	w02_2007	0
	-	Catalog	Loafer	w03_2007	0
	-	Catalog	Loafer	w04_2007	0
	-	Catalog	Loafer	w05_2007	10
	-	Catalog	Loafer	w06_2007	20
	-	Catalog	Loafer	w07_2007	30
	-	Catalog	Loafer	w08_2007	40

round

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).

> **Note:** When the round function is called with the *<multipleexpression>* argument, round first determines the nearest integer to the input and then determines the nearest multiple of the *<multipleexpression>*. For instance, round(2.8, 2) first rounds 2.8 to 3 and then returns 4 as the final value. However, round(3.8, 3) first rounds 3.8 to 4 and then returns 3 as the final value.

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 pack size. 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

The `roundup` function 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 pack size. If the packsize is 6 and the qty is 14.324 or 16.824, this returns the result of 18.

rounddown

The **`rounddown`** function 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 pack size. If the packsize is 6 and the qty is 14.324 or 16.824, this returns the result of 12.

Note: 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

The **navalue** function 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.

propspread

propspread is a 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

propspread(<*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 must 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 when this condition occurs.

If the sum of the child expressions is zero, the spread will be even.

Inverse

The **propspread** function does not have an inverse.

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
```

passthrough

passthrough is a multiple result function that is used to encapsulate any number of normal computations into a single expression.

> **Note:** The passthrough function is not allowed for measures with an **recalc** agg type.

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 must 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 computing values for multiple measures without having to write (develop) a multiple-result function or procedure.
2. To improve performance. 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** * TempMeas / TotalSales.**old**, SalesB.**old** * TempMeas / TotalSales.**old**)
- Proportionately spread TotalSales down to its components, SalesA and SalesB.

ranksort

The **ranksort** procedure returns the rank of intersections given the rank order (ascending or descending), the measure to rank upon and the dimensions to rank over.

Syntax

<output> <- ranksort(<input>,<rank order> [,<dimspec1>, …, <dimspecn>])

<output> is the measure that will contain the ranking results. The result of ranking will always be an integer value, so the data type of <output> must be integer or numeric.

<input> is the measure to be ranked.

<rank order> is {ascending | descending}. ascending is a keyword meaning that the intersection will be ranked in ascending value of the <input> measure, and descending is a keyword meaning that the intersection will be ranked in descending value of the <input> measure. All keywords which need to be passed to a function must be wrapped in double quotes (" "). Any other syntax will throw an error.

<dimspec1-n> is [<hierarchy>].{[<dimension>] | top} <dimspec1-n> specifies the dimensions to rank over. 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 ("all") in the hierarchy. <dimspec1-n> is optional, and if omitted the value for each hierarchy in the base intersection of <output> will be [hierarchy].top. A <dimspec> for a hierarchy that is not in the base intersection of <output>, or that references a dimension that is not higher than (a parent/grandparent etc. of) the dimension in that hierarchy in the base intersection of <output>, or which references a hierarchy that already has a <dimspec>, is an error.

The base intersection of <output> determines the intersections that will be ranked. If the base intersection of <input> is different to that of <output>, as will usually be the case, then the values of <input> used for ranking will be the values at the intersections implied by the base intersection of <output> obtained by normal non-conforming measure handling, with replication from higher dimensions and/or aggregation from lower dimensions.

The scope of the ranking is dictated by <dimspec1-n>. These will usually be implied rather than explicitly specified, and be at the top of the hierarchy. However, when a dimension is specified, there will be a separate ranking for each position (or combination of positions where a dimension is specified in two or more hierarchies) in that dimension. Thus, for example, when evaluating a measure calculated from the ranksort procedure that has a base intersection of sku/week, where the <dimspecs> reference the dimensions class and season, in a workbook with 4 classes and 2 seasons, there will be eight sets of ranks, one per class/season, and the value for each sku/week intersection will be the order of that sku/week within its class/season.

The ranking process sorts the intersections in ascending or descending value of <input>, as required, and the ranking number is the order that each position is after sorting. The intersection with the highest value of <input> (lowest when ranking ascending) will have a rank of 1, with subsequent intersections having a rank higher by one. Where two or more intersections have the same value of <input>, they will be given the same rank, but

the next rank value will account for the number of intersections with identical rankings. Thus for example, the first few rankings might be 1, 2, 3, 3, 5, 6, …

> **Note:** `ranksort` is a procedure and thus 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.
>
> The `level` modifier cannot be used on the LHS of an expression that uses the `ranksort` procedure. That is, the level at which the ranking is executed will always be determined by the base intersection of `<output>`.
>
> The `ranksort` procedure must, by its nature, calculate a rank value for every intersection within a scope, not just those that have changed values for measures on the right-hand side of the expression. In incremental calculation mode (for example, when planning online) this may cause longer than expected calculation times, especially when the measure calculated through the `ranksort` procedure is used on the right-hand side of other expressions, as those expressions, plus any "knock-on" effects will also have to be calculated for every intersection within the scope.

Examples:

- **Rank <- ranksort(WpSlsR, "descending")**
  - If Rank has a base intersection of sku, the result of this procedure is the integer value representing where each SKU's Sales value ranks amongst all SKUs. The SKU with the highest WpSlsR value will have a rank of 1.
- **Rank <- ranksort(WpSlsU, "descending", [prod].[clss])**
  - If Rank has a base intersection of sku, the result of this procedure is the integer value representing where each SKU's Sales units ranks amongst all SKUs within its class. The SKU with the highest WpSlsU value in each class will have a rank of 1, and there will be several SKUs with a rank of 1, one per class.
- **Rank <- ranksort(WpSlsR, "descending", [prod].[clss])**
  - If Rank has a base intersection of sku/week, the result of this procedure is the integer value representing where each SKU/Week's Sales value ranks amongst all SKU/weeks within its class for the whole time horizon. The SKU/week with the highest WpSlsR value in each class will have a rank of 1, and there will be several SKU/weeks with a rank of 1, one per class.
- **Rank <- ranksort(WpSlsR, "descending", [prod].[clss], [clnd].[seas])**
  - If Rank has a base intersection of sku/week, the result of this procedure is the integer value representing where each SKU/Week's Sales value ranks amongst all SKU/weeks within its class/season. The SKU/week with the highest WpSlsR value in each class/season will have a rank of 1, and there will be several SKU/weeks with a rank of 1, one per class/season.

positionLocked

Position locking allows a user to lock one or more positions at any level of a hierarchy in a workbook. Once locked, the values of cells corresponding to locked positions cannot change as a result of incremental evaluation, i.e., calculations resulting from user edits in

the workbook. However, those values can change as a result of full evaluation resulting from either a full transition to the Calc rule after a refresh, a custom menu execution, or from the execution of an expression that cannot be evaluated incrementally; for example, *lhsmeasure* = 25. None of the RPAS procedures, such as lookup or flookup, honor position locking.

The positionLocked function has been provided to help solution designers honor position locking during full evaluation. However, since usage of functions cannot be combined with the evaluation of special expressions (procedures), this provision does not help designers in honoring position locking when special expressions (procedures) are used. Position locking is not supported for special expressions.

The positionLocked function is evaluated at the level at which calculation is being performed. It returns a FALSE if none of the positions for the current cell's dimensions are locked and returns TRUE otherwise. Since position locking is only available in a workbook and can only be used after a workbook has been built, the function always returns a FALSE when evaluated in a domain, used in the load rule group, or used in a workbook. When used in a domain, the function will log a warning in the RPAS logs, but will not fail evaluation. The function will work as expected when used in refresh, calc and commit rule groups. It will also work as expected in custom menu rule groups that operate on the workbook.

The positionLocked function does not provide for hierarchical protection processing. Refer to the following examples for a better understanding of this behavior.

Syntax

```
positionLocked()
```

The function does not take any arguments and returns a Boolean value. The function can be used by itself on the RHS.

Inverse

The function does not have an inverse.

Examples:

- **Output = positionLocked()**
  - Populates the Output measure with values that tell whether a cell is locked as a result of position locking along any of the dimensions in the base intersection of the measure.
- **Output = !positionLocked()**
  - Populates the Output measure with values that tell whether a cell is not locked as a result of position locking along any of the dimensions in the base intersection of the measure.
- **Output.level([PROD].[clss]) = positionLocked()**
  - Populates the Output measure (default spread type: repl) with values that tell whether a cell is locked as a result of position locking along any of the dimensions, except those belonging to the PROD hierarchy, in the base intersection of the measure. For the PROD hierarchy, the check is done at the clss dimension and the results would be spread down to the base intersection of the Output measure. Therefore, at the base intersection level, the cell value will be TRUE if all child positions of the cell's clss are position locked, and FALSE if any of them are unlocked irrespective of whether the position itself is locked or not; i.e., the positionLocked function does not provide for hierarchical protection processing.
- **Price = if (positionLocked(), ignore, Price.master)**

- When used in a load rule group, this expression behaves the same as “Price = Price.master” because positionLocked() always returns FALSE. When used in a refresh rule group, the LHS is not updated for positions that are locked.

- **Price.level([PROD].[clss]) = if (positionLocked(), ignore, Price.master)**
  - When used in a load rule group, this expression behaves the same as “Price.level([PROD].[clss]) = Price.master” because positionLocked() always returns FALSE. When used in a refresh rule group, the LHS is not updated for clss level positions that are locked, i.e., all their children are locked. However, if any of the children is unlocked, and hence the clss level position is unlocked, the result of the spread after the expression evaluation will alter the child position’s value, even if the child position was locked, i.e., positionLocked function does not provide for hierarchical protection processing.

D

Appendix: Aggregation and Spread Types

Aggregation Types

The following table describes the supported aggregation types.

Aggregation Type	Description	Valid Data Types	Recommended Spread Types
recalc	The measure is not aggregated, but is recalculated at all aggregated levels through a recalc expression. The passthrough function is not supported with this agg type.	numeric, string, date, Boolean	none
ambig	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.	numeric, string, date, Boolean	none
ambig_pop	The measure is aggregated by considering the values of all populated child cells. If all populated child cells have the same value, the aggregated value is the same as the child cells. Otherwise, it is ambig.	numeric, string, date, Boolean	none
popcount	The measure is aggregated by counting the number of child cells that are populated (meaning that they have a value different from the NA value for the measure).	numeric, string	none
hybrid	The measure is aggregated using a specific aggregation type for each hierarchy. This is selected from the valid aggregation types for the measure type.	numeric, string, date, Boolean	none
total	The measure is aggregated by taking the total (numeric sum) of the values of all child cells at the base intersection.	numeric	prop
total_pop	The measure is aggregated by taking the total (numeric sum) of the values of all populated child cells at the base intersection.	numeric	prop_pop
average	The measure is aggregated by taking the numeric average of the values of all child cells at the base intersection.	numeric	prop
average_pop	The measure is aggregated by taking the numeric average of the values of all populated child cells at the base intersection.	numeric	prop_pop

Aggregation Type	Description	Valid Data Types	Recommended Spread Types
max	The measure is aggregated by taking the maximum of the values of all child cells at the base intersection.	numeric, date	repl
max_pop	The measure is aggregated by taking the maximum of the values of all populated child cells at the base intersection.	numeric, date	repl_pop
min	The measure is aggregated by taking the minimum of the values of all child cells at the base intersection. Note: For most purposes, the min_pop aggregation type is appropriate because the minimum value of all child values is typically the NA value, which is usually zero.	numeric, date	repl
min_pop	The measure is aggregated by taking the minimum of the values of all populated child cells at the base intersection.	numeric, date	repl_pop
popcount	The measure is aggregated by with the number of populated cells at the base intersection.	numeric, string	none
pst	The measure is aggregated by selecting the first value along the innermost hierarchy and by taking the total of all child values along all others. Note: "First" only has a meaning in the calendar hierarchy. Therefore, this agg type must only be used for measures whose innermost hierarchy is the calendar hierarchy.	numeric	ps
pet	The measure is aggregated by selecting the last value along the innermost hierarchy and by taking the total of all child values along all others. Note: "Last" only has a meaning in the calendar hierarchy. Therefore, this agg type must only be used for measures whose innermost hierarchy is the calendar hierarchy.	numeric	pe
median	The measure is aggregated as the median value (the middle value when sorted from lowest to highest) of the values of all child cells.	numeric	repl
median_pop	The measure is aggregated as the median value (the middle value when sorted from lowest to highest) of the values of all populated child cells.	numeric	repl_pop

Aggregation Type	Description	Valid Data Types	Recommended Spread Types
and	The measure is aggregated by performing a Boolean And operation on the values of all child cells.	Boolean	repl
or	The measure is aggregated by performing a Boolean Or operation on the values of all child cells.	Boolean	repl

Note: Only measures with an aggregation type of ambig, pst, or pet can be aggregated from below the partition levels to above the partition levels in a global domain.

Spread Types

The following table describes the supported spread types.

Spread Type	Description	Valid Data Types
none	Values are not spread	numeric, string, date, Boolean
repl	Replicate the value to each cell	numeric, string, date, Boolean
repl_pop	Replicate the value to each populated cell	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
delta	Increment/decrement each cell evenly. Effectively the "even" spreading of the change ("delta").	numeric
ps	Apply delta to starting period	numeric
pe	Apply delta to ending period	numeric

Arithmetic Operators

This section provides information about the arithmetic operators supported in Configuration Tools.

Unary Operators

The following unary arithmetic operators are supported:

Symbol	Type	Function
-	real	Negation

Symbol	Type	Function
!	Boolean	Compliment

Binary Operators

The following binary arithmetic operators are supported:

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 **Note**: When used to compare two strings, this operator performs a case-insensitive compare.
!=	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

E

Appendix: Configuration of RPAS Extensions

Configuration of RPAS Extensions

This section provides information about the configuration of the Fusion Client that is needed in order to complete the integration of RPAS extensions with the Fusion Client. For more information about the RPAS extension framework read the relevant sections of the Fusion Client installation guide. Here we provide a brief summary of this feature.

About the RPAS Solutions Extension Framework

The RPAS Extension Framework refers to the process of incorporating "small" external applications within the user interface of the Fusion Client. We term such applications as "RPAS extensions" or "plug-ins". They are external in the sense that they present a UI to interact with external systems that are not RPAS domains and workbooks. For example, in the current Fusion Client release a plug-in called "PO View" is available. It displays purchase order data from an RMS database for items selected in an RPAS worksheet.

Plug-ins are distributed as a bundle, which is a collection of one or more related plug-ins, sharing a common install process.

At a high level plug-ins are integrated with the Fusion Client by performing the following steps.

- The plug-in bundle's installer is run. It installs a shared library in the Weblogic Server.
- The Fusion Client's installer is re-run. This creates a reference to the above-mentioned shared library within the Fusion Client.
- The activity/task flow configuration file is modified in a fashion suitable for displaying the plug-in's UI within the Fusion Client UI.

There are three styles in which a plug-in can be displayed:

- **Launched from navigation tree**: The user clicks a new kind of link in the navigation tree on the left. This launches the corresponding plug-in's UI in the content area on the right. We term such links module links to distinguish them from the tree elements corresponding to RPAS tasks and steps. This launch style is appropriate when the plug-in function is a step in the overall business workflow represented by the tasks and steps in the navigation tree. An example here is the set of Space Optimization plug-ins that go with the Category Management application and augment its workflow.
- **Home Page**: In the content area of the home page. No clicking of links is necessary. The plug-in UI is displayed right away upon login, or upon clicking the Home link while in an open workbook. This launch style is appropriate when the plug-in is intended to provide a snapshot view of the issues needing the user's attention after logging into the application. An example here is the Dashboard plug-in that is distributed with the AIP application.
- **Context Menu**: The user clicks a link corresponding to a plug-in in the context menu available on the body of a RPAS worksheet. This launches the plug-in in a separate panel that appears and behaves similarly to the usual RPAS worksheet panels. This

launch style is appropriate when the plug-in function pertains to the positions selected in the RPAS worksheet. For example the PO View plug-in shows the purchase orders pertaining to the products and locations selected on an RPAS worksheet.

In the following sections we describe the configuration that is needed to enable the display of a plug-in in one style or the other.

> **Note**: For the purpose of configuration, plug-ins are known as "modules".

In the configuration process the bundle name and the plug-in name need to be specified. As described in the installation guide, the bundle name is found in the name of the bundle manifest file. The manifest file, it may be recalled, is named as <bundlename>-bundle-manifest.xml. For example, the aip bundle (part of the AIP application distribution) has its manifest file named as aip-bundle-manifest.xml.

The module name refers to the name of a plug-in in the bundle. A bundle has one or more plug-ins. The plug-ins are described in the bundle manifest file. Each plug-in has a <module> entry. The value of the name sub-element of <module> is the module name.

For example, the AIP Dashboard plug-in bundle's manifest file, named "aip-bundle-manifest.xml": has the following XML content:

```
<modules>
  <module>
    <name>aipdashboard</name>
    <description>AIP Dashboard</description>
```

The name of the manifest tells us the name of the plug-in bundle, namely "aip". The "name" sub-element under the "module" element above, gives us the plug-in name, namely "aipdashboard".

Launch from Navigation Tree

There are two types of links that can be created in the navigation tree: module tasks and module steps. Module tasks are links created at the same level as RPAS tasks. Unlike RPAS tasks, module tasks do not have child entries. Rather, they are links that can be clicked to launch the associated plug-in in the content area on the right.

Module steps are links created within an RPAS task. They are peers of the regular RPAS steps that are children of the RPAS task.

The choice of whether a plug-in is to be configured as a module step or task could hinge on at which stage one wants the plug-in function to be executed, regarding the navigation tree as a sequencing of processes and sub-processes as read from top to bottom.

Module Tasks

Like RPAS tasks, module tasks are configured in the file *TaskFlow_MultiSolution.xml*. The structure of the entry is as follows.

```
      <module_task>
 <name>SpaceOpt.VPOG.name</name>
<description>SpaceOpt.VPOG.desc</description
<solution>CatMan</solution> <!-- solution is optional -->
          <module>vpog</module>
          <module_bundle>so</module_bundle>
          <order_num>1</order_num>
      </module_task>
```

This element is to be inserted within an activity element, as a peer of RPAS tasks.

The following table describes the fields of the entry:

Field	Description
name	The key of a string resource defined in *MultiSolutionBundle.properties*. E.g. "SpaceOpt.VPOG.name=Planogram". This appears as the text of the module task in the navigation tree.
description	The key of a string resource defined in MultiSolutionBundle.properties. E.g. "SpaceOpt.VPOG.desc=View Planogram". This appears as a popup when the user mouses over the link.
solution	Optional field. If specified, the link will not appear if for some reason the SpaceOpt set of plug-ins is plugged into a different RPAS solution than Category Management. The only reason to use this field is to prevent inadvertent use of a plug-in in the wrong context.
module	The name of the plug-in. (e.g. "pogmap")
module_bundle	The name of the bundle containing the plug-in. (e.g. "so")
order_num	The position of the task in the sequence of tasks (of both kinds) under the parent activity in the navigation tree.

Module Steps

Like module tasks, module steps are configured in *TaskFlow_MultiSolution.xml*. The structure of the entry is as follows.

```
<module_step>
<name> SpaceOpt.POGMap.name</name>
<module>pogmap</module>
<module_bundle>so</module_bundle>
<order_num>1</order_num>
</module_step>
```

The element should be placed inside an RPAS task entry, as a peer element of its child RPAS steps.

The following table describes the fields of a module step entry.

Field	Description
name	The key of a string resource defined in *MultiSolutionBundle.properties*. E.g. "SpaceOpt.POGMap.name=Product to Planogram Mapping". This appears as the text of the module step in the navigation tree.
module	Name of the plug-in (e.g. "vpog")
module_bundle	Name of the bundle containing the plug-in (e.g. "so")
order_num	Position of the module step in the sequence of steps (of both kinds) under the parent RPAS task.

Launch on Home Page

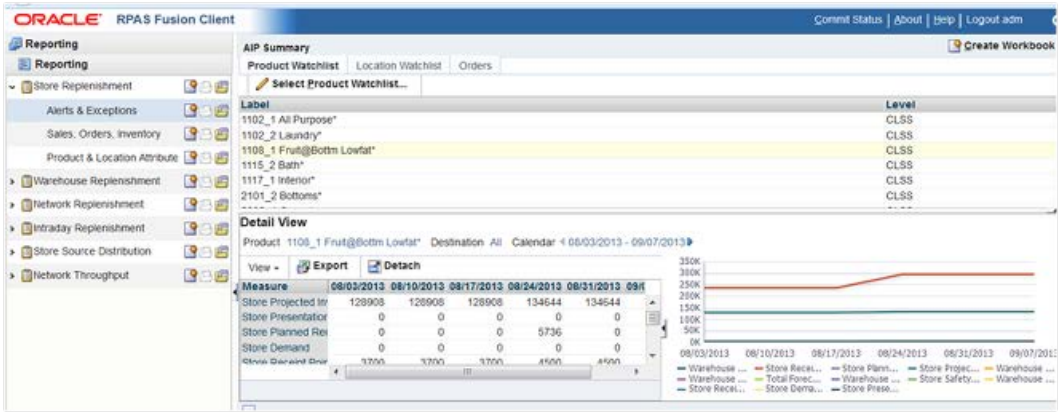
When a functional module is displayed as an alternative home page in the fusion client it is not declared in the Taskflow_Multisolution.xml, but is instead identified in config.properties, as follows:

homepage.module=<bundle-name> | <plug-in-name>

This earmarks the referenced module as the one to launch on the Fusion Client home page upon login.

If the property is left undefined, then there is no functional module to launch.

The following figure illustrates a plug-in launched on the home page:



Plug In Example - Home Page

Launch In-Context of a Worksheet

In-context launch is also configured as an XML element in the task flow configuration file, namely Taskflow_MultiSolution.xml. The element specifies the plug-in to be launched. Where the element is inserted determines which worksheet's context menu makes the plug-in available for launch.

The structure of the task flow configuration file can be described briefly as follows: it is a set of activities. Each activity has a set of tasks. A task has a set of steps; a step has a set of worksheets.

The in-context launch entry can be created as a sub-element of a task, step or worksheet. The implication of each insertion point is described in the following table.

Insertion Point	Effect
Task	The plug-in is available on every worksheet of every step under the task
Step	The plug-in is available in every worksheet under the step
Worksheet	The plug-in is available on that worksheet

We term this mechanism as *in-context plug-in inheritance*. The intention is provide configuration convenience, by allowing a one-place definition of an in-context plug-in for a whole set of worksheets based on a common parent step or task.

In-Context launch entry

This is the structure of the XML element:

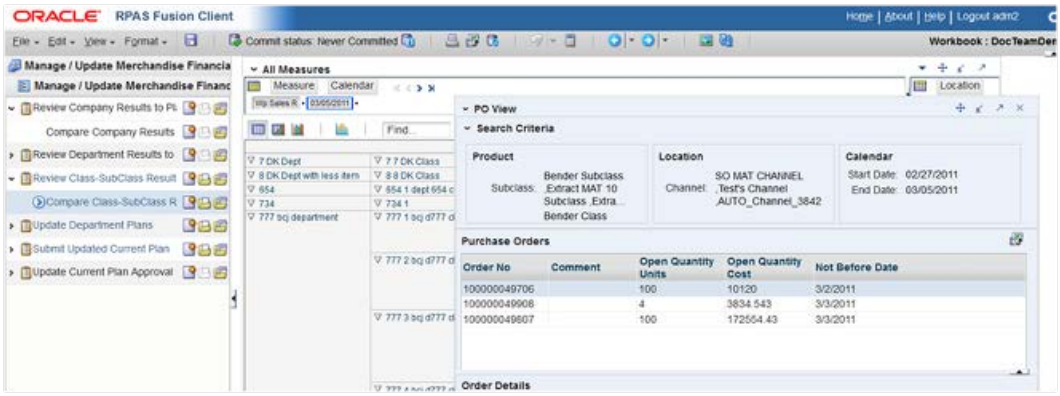
```
          <incontext_modules>
             <module name="poview" bundle="poview" label="poview.label"

resource_bundle="oracle.rgbu.ard.util.i18n.SolutionResourceBundle"/>
          <module reportName="billablerpt" name="obiee" bundle="obiee"
label="billablerpt.label"
    resource_bundle="oracle.rgbu.ard.util.i18n.SolutionResourceBundle"/>
  </incontext_modules>
```

The following table describes the fields of the element:

Field	Description
name	Name of the plug-in
reportName	For plug-ins supporting the generation of reports, this parameter identifies the report name as it appears in the reportConfig.xml
bundle	Name of the bundle containing the plug-in
label	Key of the string resource that must be defined in *MultiSolutionBundle.properties.* For example it could be defined as "poview.label = Purchase Order View". This is the label that identifies the plug-in in the context menu.
resource_bundle	The base name of the resource bundle file. Use the value as given in the above example if the label resource is defined in *MultiSolutionBundle.properties.* An alternative location for placing the label resource is a resource bundle file shipped with the plug-ins. If you wish to use this location, please refer to the corresponding installation guide for ascertaining the location of the file and the "base name" to use to refer to it.

The following figure illustrates the launch of the "PO View" plug-in in the context of an MFP worksheet:



Plug In Example - Home Page

F

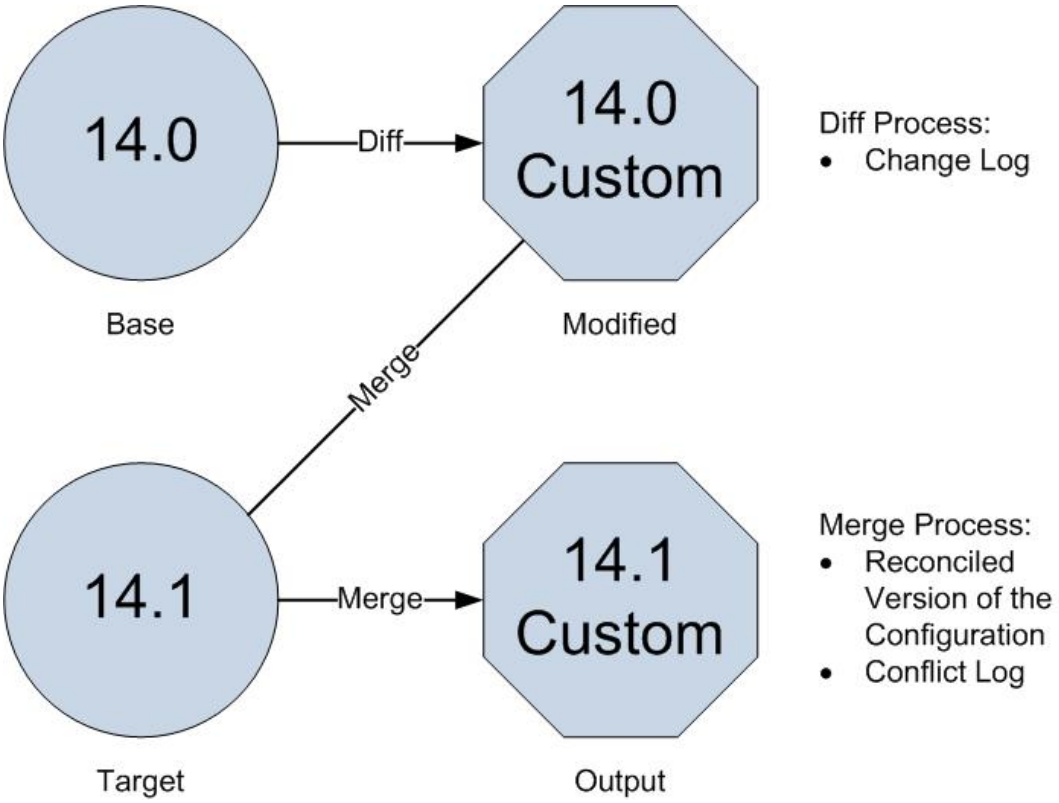
Appendix: RPAS Configuration Manager and rpasConfigMgr

Using the rpasConfigMgr

Many RPAS users customize their configurations of RPAS and RPAS applications in order to be more aligned with their business needs. When those users want to upgrade to a new RPAS or RPAS application release, they can use the RPAS Configuration Manager or the rpasConfigMgr utility to replicate their customizations on the new base release with as little manual configuration as possible. By using these tools to upgrade to a new release, users can take advantage of the new features of that release while maintaining their customized configuration.

rpasConfigMgr Process

The rpasConfigMgr utility consists of two processes: diff and merge. Diffing is the identification of modifications between two versions of a configuration. Merging is the reconciliation between a base version and two modified versions of a configuration. These processes are shown at a high level in the figure below.



The Diff and Merge Processes

When a user has modified the base release (base) to create a customized configuration (modified), that user can use the rpasConfigMgr process to upgrade to the next base

release (target) and then apply their configuration to create a customized configuration of the new release (output).

To do this, the user first runs the diff process which finds the customized aspects of the modified version by identifying the differences between the base and modified version. The output of the diff process is a change log file which records the differences between the base and the modified versions.

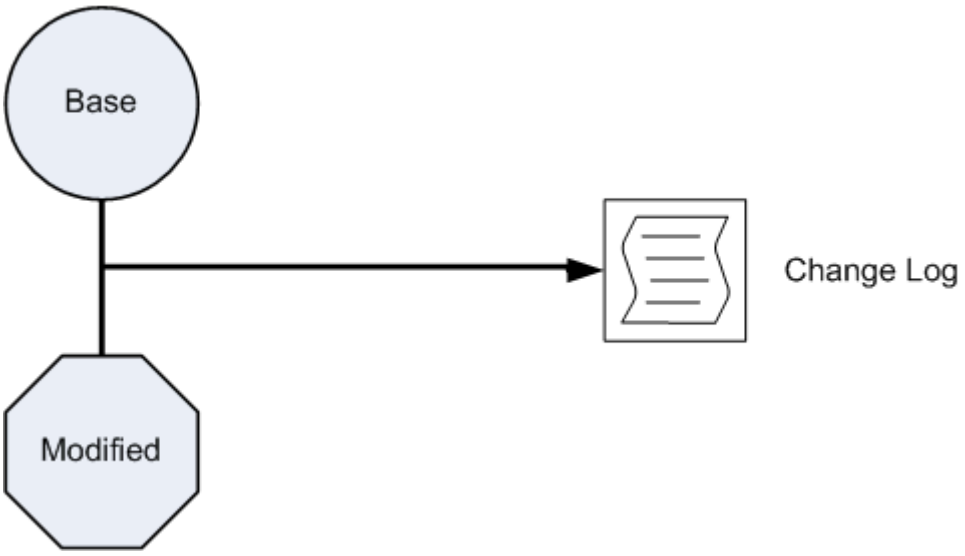
Then, the user runs the merge process which automatically applies the customized configuration to the target version if possible. If an aspect of the customized configuration cannot be applied to the target version, it is recorded in the conflict log file. The output of the merge process is the reconciled version of the configuration, a change log, and a conflict log.

Alternatively, the user can run a diffAndMerge process that combines the two processes and performs them together. The result of that process is a reconciled version of the configuration, a change log, and a conflict log.

These three processes, diff, merge, and diffAndMerge, are explained in greater detail on the next pages.

Diff Process

The diff process analyzes two different versions of a configuration in order to determine the differences in the configuration between them. Examples of such differences could be property changes, altered parent/child relationships, or reordered information of elements that obey strict ordering. These differences are recorded in a change log file.



Diff Process

The change log is an XML file that contains two distinct classes of elements: scope elements and operations elements.

Scope elements contain the structural information of the RPAS domain configured content. Scope elements by nature do not encode any information about modifications; they only provide information about the area of the configuration in which any given modification occurs.

Since the structural information can be very large and complex, the change log can contain numerous nested scope elements. These elements have an overall structure that mirrors the directory structure used within a configuration. The top level of the scope represents the entire domain. This level contains sub-scopes that represent the changes

that directly affect hierarchies, styles, and data interface entries. Other sub-scopes represent the solutions in the domain. Solution scopes contain within them sub-scopes that represent changes that affect measures, rules, workbooks, and wizards.

To reduce the overall size and complexity of the structural information, unnecessary scope elements are removed from the change log. Unnecessary scope elements are those that contain only other scope nodes within its sub-tree. These elements usually occur when an aspect of a configuration has not changed between versions. For example, a change log that is generated for a configuration that has modifications made only to workbook measure labels would not contain any meaningful change information about hierarchies since no changes to the hierarchy information were made. Therefore, there is no need for scope elements relating to hierarchy and dimension structure to be included in the change log. For this reason, the hierarchy scope elements are removed in order to reduce the size of the change log.

Operation elements, also called change elements, are the other class of element in the change log. Within the structure created by the scope elements are the modifications made to the content of the domain. The operation elements describe the nature of the change so that the corresponding modification can be made in the merging process.

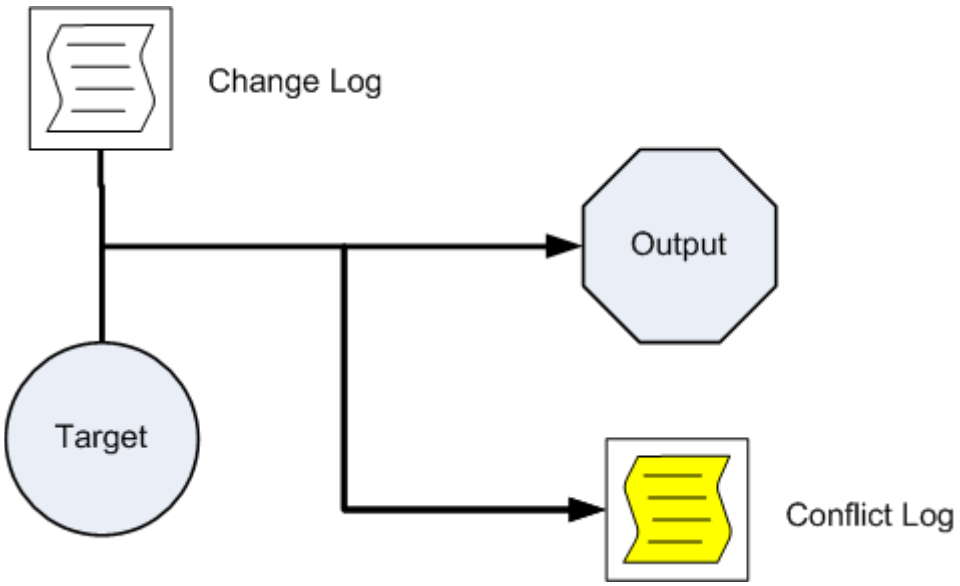
There are several operations that can be performed during the configuration process. Each of these requires its own operation tag. The operations are described below.

- Element removal: an operation that removes a piece of configured content from its parent container. This operation requires no additional information.
- Element addition: An operation that inserts an additional piece of configured content to a parent container. This operation requires specification of any attributes of the new content. If the new element obeys a strict ordering (such as a position of a new rule within a rule group), then information about the position of the new element is required as well.
- Element attribute modification: an operation that modifies one or more attributes of a piece of configured content. This operation requires information that identifies which attribute was modified and its new value.
- Element reordering: an operation that specifies a change in the order of a piece of configured content that has a meaningful order (such as the order of a rule within a rule group). This operation requires information about the new order of the element.

Each of the operation elements described above must provide enough information in order for the configuration merging process to recreate the modification within the target configuration. The total collection of the operation elements of a change log should provide enough information to completely capture all changes made to the input configuration.

Merge Process

The merge process automatically applies the modifications that were identified in the change log to a target configuration. The merge process uses the change log and the target configuration to create the output configuration and a conflict log, as shown in the diagram below.



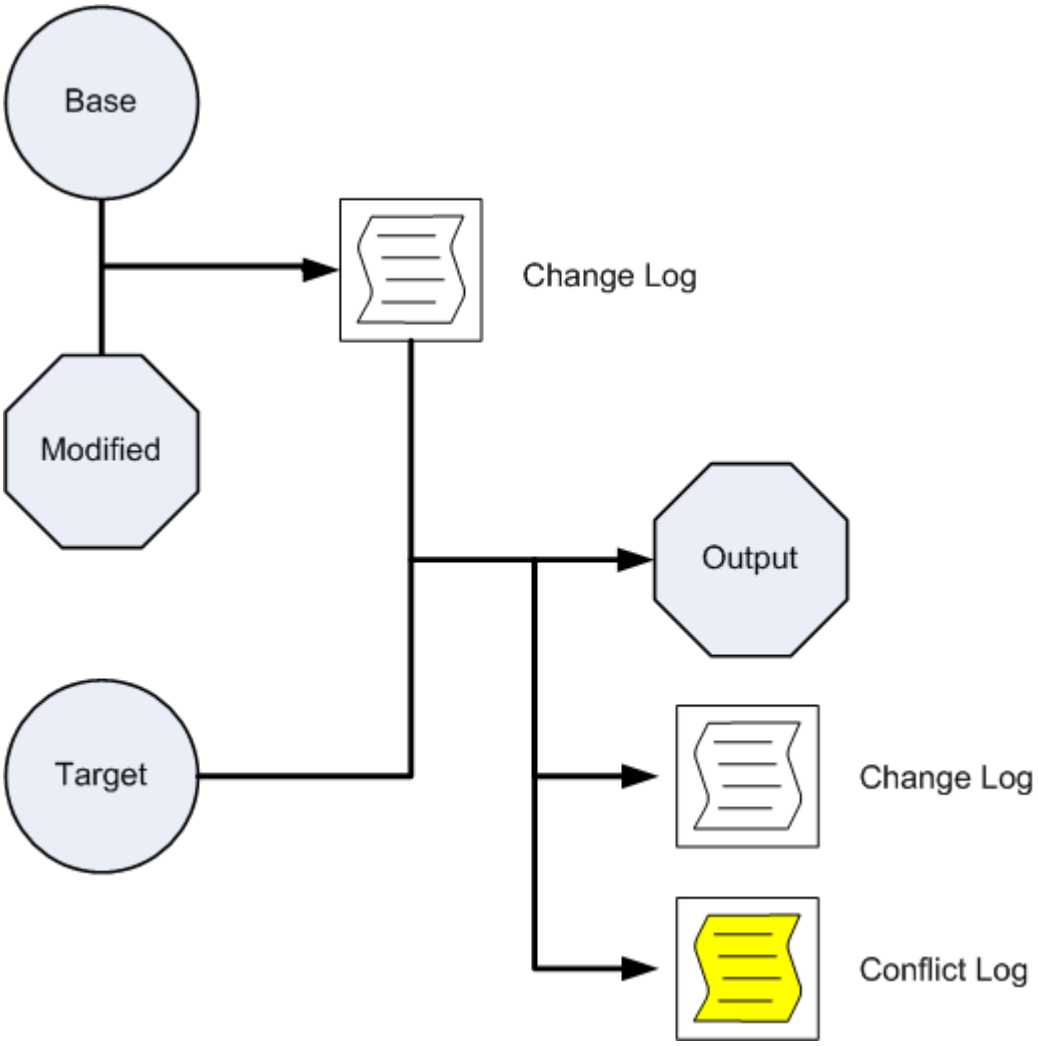
Merge Process

The merge process attempts to recreate each operation represented in the change log. In cases where the modification can be correctly applied, the merge process performs the modification on the target configuration. However, if the differences between the base and target configurations make the operation impossible to perform, then the modification is not applied to the target configuration. In those cases, the operation is documented in the conflict log.

The conflict log is an XML file that contains the same scope structure as the change log. The conflict log also contains conflict nodes that describe the operations that could not be performed automatically. This information is provided so that you can manually configure them after the merge process.

diffAndMerge Process

The diffAndMerge process is the combination of the diff and merge processes. Use the diffAndMerge process if you do not want to review the change log before continuing to the merge process. The diffAndMerge process loads the three versions of a configuration (the base, the modified, and the target versions), performs the diff command on the base and modified versions, and then immediately applies the detected changes to the target. The output of the diffAndMerge process is the reconciled version of the configuration, a change log, and a conflict log, as shown in the diagram below.



diffAndMerge Process

For more information about the diff and merge steps of the diffAndMerge process, see the Diff Process and Merge Process pages.

rpasConfigMgr Usage

The rpasConfigMgr utility supports three commands:

- `-diff` : The `-diff` command loads two versions of a configuration and produces a change log that describes all of the changes in the modified version of the configuration.
- `-merge`: The `-merge` command loads and parses a change log and attempts to apply the changes described within it to a configuration that you specify. It then outputs a modified version of the configuration and a conflict log to the specified location.
- `-diffAndMerge`: The `-diffAndMerge` command loads three versions of a configuration (the base, the modified, and the target versions), performs the diff command on the base and modified versions, and then immediately applies the detected changes to the target. The output of the diffAndMerge process is the reconciled version of the configuration, a change log, and a conflict log. These are output to the location you specify.

Below are examples of rpasConfigMgr commands. In these commands, [baseConfigPath], [modifiedConfigPath], and [targetConfigPath] are paths to the root document of the base, modified, and target versions of a configuration, respectively. [outputDirectory] is the location you specify for the outputs of the commands, and [changeReportPath] is the path to the location of the change log used in the –merge command.

```
rpasConfigMgr –diff –base [baseConfigPath]–mod [modifiedConfigPath]–output [outputDirectory]
rpasConfigMgr –merge –target [targetConfigPath] –change [changeReportPath] –output [outputDirectory]
rpasConfigMgr –diffAndMerge –base [baseConfigPath]–mod [modifiedConfigPath]–target [targetConfigPath] –output [outputDirectory]
```

RPAS Configuration Manager

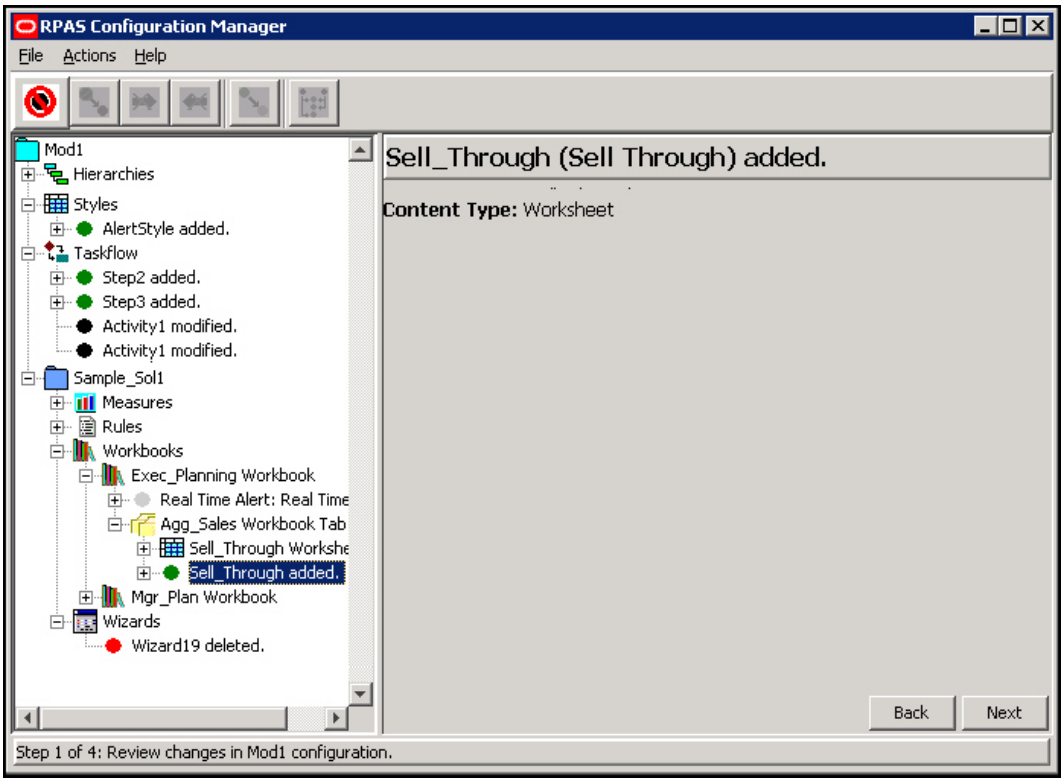
The RPAS Configuration Manager extends the rpasConfigMgr utility to provide a more powerful and flexible tool to manage differences between different versions of a configuration. This is accomplished through two operations:

- Creating a detailed Change Report which describes the differences between two versions of a configuration.
- Creating different modified versions of a configuration in reference to a base version of the configuration.

RPAS Configuration Manager uses the same diff and merge functionality that rpasConfigMgr uses, but expands on that functionality in order to provide the user with more information about and control over the merge process.

- RPAS Configuration Manager gives users the ability to interact with the process, enabling the user to select from a subset of all of the potential changes which diffs to apply to the upgrade.
- The rpasConfigMgr is sensitive to the changes in the base and the mod only. RPAS Configuration Manager has two diffs:
- Between base and modified
- Between base and the update

A set of algorithms compares these two diffs and detects changes present in the two modified configurations that could potentially create conflicts. RPAS Configuration Manager then allows users to resolve these conflicts before attempting the merge operation.



Change Report

When the user wants to determine the set of changes present between two versions of a configuration, they select the Change Report activity from the main UI. After being prompted to supply a base configuration and a modified configuration, the user is presented with a description of the changes present between the two versions.

Field/UI Item Description

- Green dot: Content that was added
- Red dot: Content that was deleted from the configuration
- Blue dot: Content that has been moved in the context of something where ordering matters (ordering of rules inside of a rule group matters).
- Black dot: Piece of content that has had some property of that content modified
- Attribute: Displays the name of the property that was changed

> **Note**: Selecting a node/dot will make more info about that change appear in the content area.

In addition to allowing the user to inspect the changes between two versions of a configuration, the UI provides the ability to output information about the changes in the form of a report. This report differs from the Change Log created as a part of the original rpasConfigMgr functionality in that it is intended to describe the detected changes in a readable format

The change report provides summary information about changes to the configuration as well as separate summaries for the various functional areas of the configuration (for example, Hierarchies, Measures). It also provides detailed information about individual changes.

Merge Functionality

RPAS Configuration Manager also provides support for and expands the functionality of the merge operation of the rpasConfigMgr. As with the diff operation, the user can specify a new merge activity. The user is then prompted for a base version of the configuration, as well as two modified versions of the configuration.

RPAS Configuration Manager performs the diff operation to compare each of the modified configurations to the base configuration. This creates two distinct change lists. These change lists are used as the inputs to the conflict reconciliation process. The reconciliation process identifies conflicts between the changes in the two change lists and the user's directives for those changes to resolve conflicts that would prevent a successful merge of changes to the configurations.

After the reconciliation process is complete, the utility merges the two distinct change lists into a single, comprehensive list of changes. This merged change list is then applied to the base configuration to generate a merged output configuration containing all changes present in both modified configurations that have not been discarded by the user as part of the reconciliation process. This merged configuration is then saved to a location specified by the user along with a Change Report which contains changes from the merged change list.

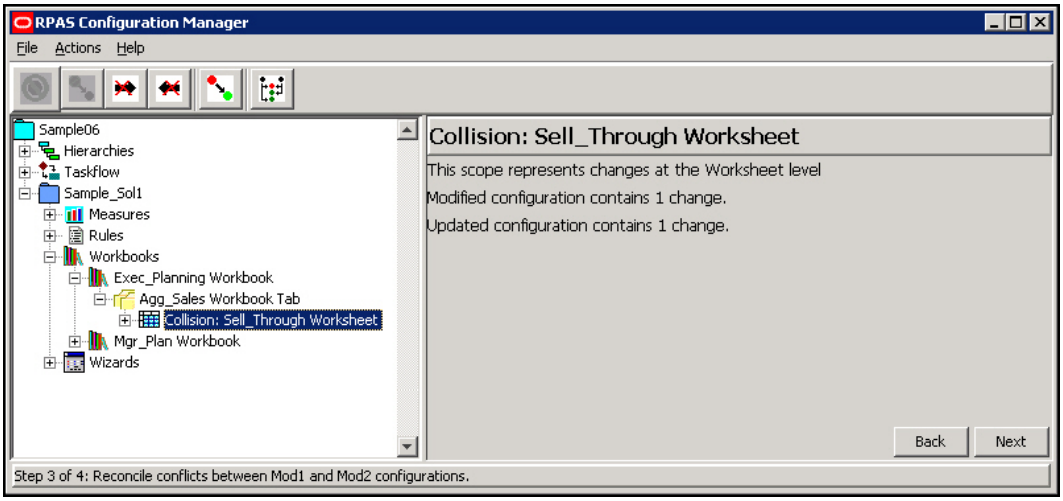
Conflict Resolution Functionality

The user specifies the base and two modified versions of the configuration. Oracle Retail Predictive Application Server Configuration Manager performs a diff between the base and each of the modified versions.

The user has the ability to examine each of the Change Log summaries. The user can discard any changes they do not wish to merge before moving forward.

The user initiates an action to detect conflicts between the two sets of changes. In its most basic form, this entails performing a series of basic merge operations from the rpasConfigMgr and recording the resulting merges.

After conflicts (whether potential or actual) have been discovered, the utility presents the conflicts in the UI. The user can examine the conflicts present between the two modified versions of the configuration and begin resolving the conflicts.



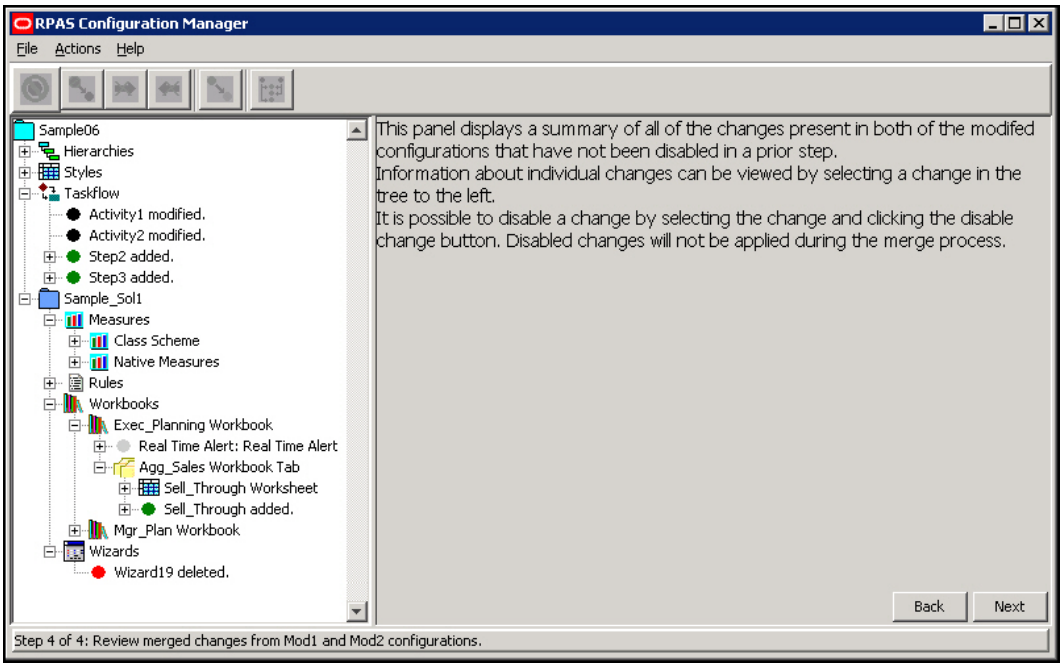
Conflict Reconciliation

This screen shows information about the new set of algorithms, the ones that find the differences between the two diffs. Oracle Retail Predictive Application Server

Configuration Manager takes the two sets of changes and runs a collision detection algorithm. This algorithm produces an enhanced node structure, which identifies every place that could potentially have a conflict during the merge step.

After all collisions are resolved, a change list opens. This change list describes all changes from both of the configurations except for the ones discarded. This list is a union of both sets of changes, all the changes between the original base configuration and the output at the end. This list can be used to review the changes before the merge is committed.

Oracle Retail Predictive Application Server Configuration Manager attempts to do the merge. If there are still conflicts interfering with the merge process, the Conflict Resolution screen summarizes the conflicts. This enables the user to disable the change or modify it is that stops that change from working. The user can then attempt the merge again. This is an iterative process.



Merged Changes Review

To reduce the complexity of the resolution process and to provide a greater level of assurance that the output configuration is self-consistent, conflicts are grouped into scopes by default. These scopes correspond to the entities inside an RPAS domain (for example, a measure or a workbook). Users select which of the two modified versions of the configuration they want to accept changes from. The selected version has its changes retained, while any conflicting changes in the other configuration are discarded.

For example, if both modified versions of a configuration modify properties of a measure in the base configuration, it is possible that some of the changes cannot be merged without causing a non-consistent configuration. One modified configuration might change the na value or range of the measure, whereas the other modified configuration might change the type. Although no two changes affect the same property, the combination of the changes results in an invalid measure configuration. For this reason, the user takes either one or the other of the two sets of changes to the measure to be the final change set.

This all-or-nothing approach be too limiting for the user. For example, consider a case where one modified configuration modifies data-centric properties of a measure, such as type or base intersection, and the other modified configuration modified purely cosmetic

properties such as the measure label. In this case, both sets of changes can be retained without creating a non-consistent output configuration.

The conflict is presented to the user on a conflict-by-conflict basis (as opposed to an aggregate level of the entity) and users can select individual conflicts. Users can then manually select which changes result in conflicts and disable those changes. After satisfied that conflicts have been eliminated, the user can mark that collision as resolved and continue to the next collision.

For example, if both modified versions of a configuration contain instructions to modify the label of a measure, the user experiences a conflict. The merge process must accept one of the new label values or the other (or at the scope level, one set of changes to measure properties or the other). The simplest way of resolving this conflict is for the user to decide which of the two new label values is desired. After this has been determined, the instruction for the undesired label change is disabled in the Change Logs (a disabled change is one that is not applied as part of the merge process). At this point, the merge is successful; only the desired modification is processed.

A more complicated scenario would involve some change present in one modified configuration that can no longer be performed due to changes made in the other modified configuration that do not result in direct collisions, as above, but instead result in changes to some dependent piece of content.

For example, consider the situation in which one configuration deletes a measure from the solution. The second configuration, having not deleted the measure, might attempt to add it to a measure profile. This operation cannot be performed because the operation has a dependency on some non-local content (the base measure) that can no longer exist within the merged configuration.

The utility also attempts to detect these non-local collisions. When a collision is detected, the utility reports that collision the same way that it reports local collisions. As with local collisions, users can reconcile the collision by selectively disabling changes.

RPAS Configuration Manager Application

The RPAS Configuration Manager application can be launched by calling the LayeredConfig.sh script located in the bin directory of your Tools distribution.

The application provides two main functions, a Change Report and a Merge Operation:

Merge Operation

1. Select **New Merge Operation** to begin a new operation. Select **Open** to resume a suspended operation.
2. You are prompted to enter locations for three configurations:
   - base
   - mod1
   - mod2

   These represent the original unmodified configuration and two updated versions of that configuration. The field expects the path to the root .xml document of the configuration (the file selected during the open operation in the tools).

   Note: Mod1 and Mod2 are labels that can be changed using buttons, if this is desired. The new labels are used wherever Mod1 or Mod2 are referenced in the UI.

3. Click **Next** to proceed to the first step. This step is a review of the changes between the base configuration and the Mod1 configuration. In this step, there is only a single operation available. If a change node is selected, then the **Disable Change** button:

is enabled. Clicking this button marks the selected change so that it is ignored for the remainder of the operation and is not applied when the merged configuration is generated. If a previously disabled change is selected, the Enable Change button:

is enabled. Clicking this button clears the status of the change so that it is processed in future steps.

4. When satisfied with the set of changes between base and Mod1, click **Next**. You are provided with the changes between the base and Mod2 configurations. The same options exist in this step as in the previous review of the changes between the base configuration and the Mod1 configuration.
5. Click **Next**. In this step, you are presented with a list of locations where changes from the previous two steps might conflict have been identified. A conflict of this type is called a collision, and the navigation tree can be used to navigate between and view details about the conflicts.
6. You cannot progress until all collisions have been resolved. There are multiple ways to resolve collisions:
   a. Select the collision node. You can resolve the collision by disabling all changes related to the conflict in Mod1:

   or Mod2:

   If either of these options is selected, all changes listed in the Mod1 Changes or Mod2 Changes section of the collision are marked as disabled, exactly as if they had been individually selected and the Disable Change button was clicked. Selecting either Discard Mod1 Changes or Discard Mod2 Changes marks the collision as resolved so that the merge can proceed.
   b. Collisions can be manually resolved by the user. Individual changes in either the Mod1 list or the Mod2 list can be selected and disabled using the **Disable Change** button. Disabled changes cannot be re-enabled using the **Enable Change** button in this step; that operation can only be performed in the first two steps. Note also that changes disabled either through the use of the Disable Change button or through use of the Discard Mod1/Mod2 Changes button will be disabled throughout the UI, even if they appear in more than one collision. It can often be the case that resolving one collision results in one or more other collisions automatically being marked as resolved because the disabled changes were the cause of those other collisions, as well as the resolved collision.
   c. Sometimes, the user is satisfied that a collision does not create a problem during the merge step. This might be because the changes in the collision do not

interfere with each other in any meaningful way (this is especially common if one of the sets of changes are to cosmetic attributes such as labels), or because any troublesome changes have been manually disabled. In such cases, the user can manually mark a collision as resolved using the Mark as Resolved button:

This button causes the collision to be treated as resolved without disabling any additional changes in the collision (changes that were previously disabled remain disabled).

d. There is an additional button called **Recompute Collisions**:

Clicking this button causes the collision detection to be re-applied to the sets of changes that are an input to the process. However, any changes that have been disabled are ignored for this new pass. As a result, the collisions those now-disabled changes would cause do not appear after the re-application of the algorithm. This is useful when presented with a large number of collisions. As collisions are resolved, the collision set can be periodically recomputed to reduce the amount of information and make remaining collisions easier to deal with.

7. After all collisions are resolved, click **Next** to proceed to the final step of the process. You are presented with a set of changes, much like in the first and second steps. However, this step of changes is the combination of the changes found in both the first step (between base and Mod1) and the second step (between base and Mod2). This list contains the union of all changes from either change set that have not been disabled at some point in the process. In this step, you can still choose to disable any changes that you do not want applied.
8. After satisfied with the list of changes, click **Next**. At this point, the combined set of changes is applied to the base configuration to produce a new configuration that represents the merging of the changes in Mod1 and Mod2. You are prompted for a location in which to save this new configuration, as well as a summary report. In some cases, it might be possible that changes in the separate configurations cannot be applied even though the collision detection algorithms did not detect the conflicts. In this case, the user is presented with a description of the conflicts. If this should occur, the user must go back to the merge step and manually disable changes to resolve the undetected conflict. The user can then move forward to the output step.

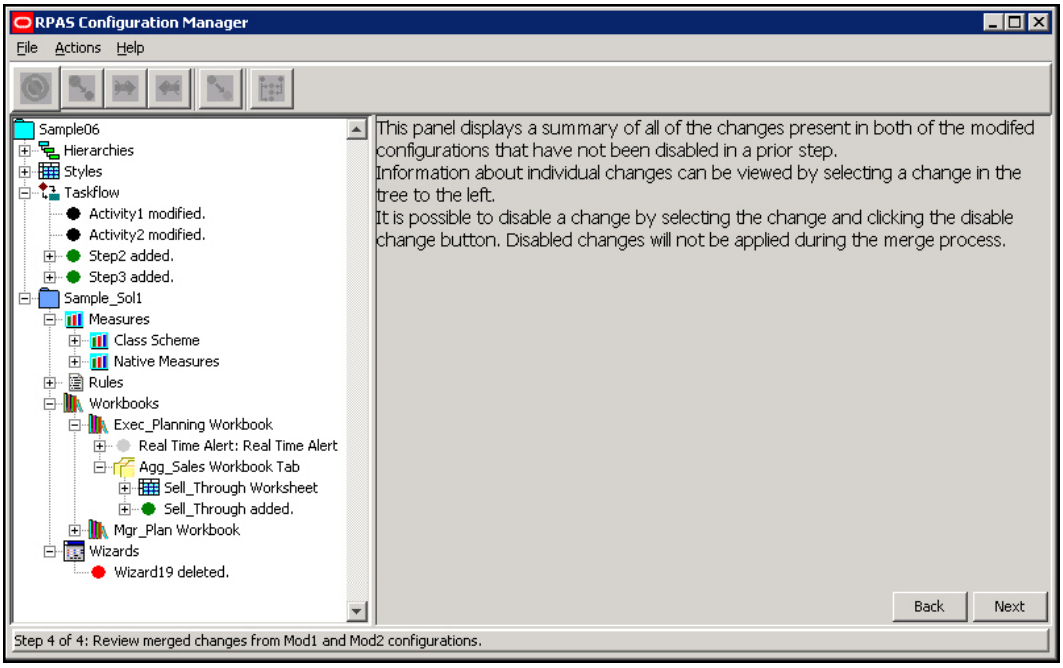
A Note on Saving and Loading Merge Operations

Because the merge process can take a considerable amount of time for complex configurations where a large number of changes differentiate the Mod1 and Mod2 versions, RPAS Configuration Manager provides a mechanism to capture the current state of the merge operation and save it disk. This saved record can then be reloaded at a later date to enable the user to pick up on the process where the user left off without starting again from scratch. This functionality is available by making use of the **Save** and **Open** menu items within the File menu.

Caution: The save operation prompts you to select a directory in which resources describing the state of the merge operation will be saved. This directory should not contain any existing files or directories to prevent conflicts with the resources created by RPAS Configuration Manager. When loading a saved operation, you should provide the directory specified during the save operation.

Change Report Operation

1. Select **New Change Report** from the file menu. You are prompted to enter the paths to a base version of a configuration and a modified version of the configuration.
2. Click **Next** to move to a visual description of the changes present in the two versions of the configuration. This visual description is identical to the view presented during the merge operation when reviewing differences between a base and mod configuration. However, this view is only intended to allow a visual inspection of the detected changes; there are no meaningful operations that can be performed at this point.
3. Click **Next** to move to the final step. In this step you are prompted to provide a directory for output. In this directory a summary of the changes detected is generated.



Merged Changes Review

G

Appendix: Dynamic Hierarchies

Dynamic Hierarchies Overview

A dynamic hierarchy is a dimension within a workbook whose relationship is dynamic based on the context of the workbook and thus can vary from one workbook to another. The positions within a dynamic hierarchy are built using measure data during the workbook build process. They may vary each time a workbook is built, but the hierarchical relationships within the workbook remain constant.

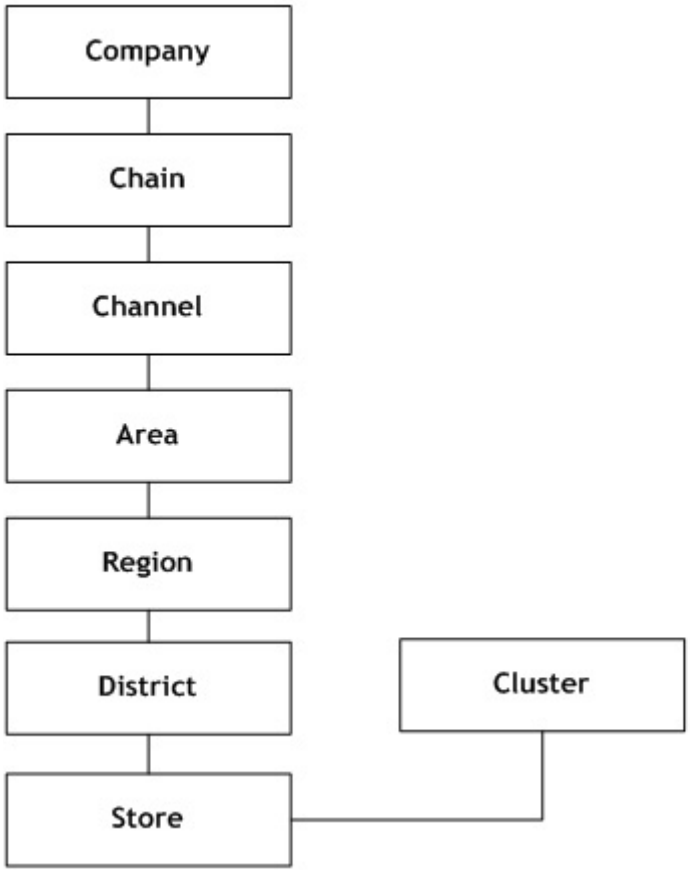
There are two types of dynamic hierarchies available in RPAS. The first is referred to as domain modified dimensions. These dimensions exist in the domain hierarchy, but are modified in the workbook. The second is referred to as workbook-only dimensions. These dimensions only exist in the workbook and are available for viewing purposes. This appendix will provide additional details on these two types of dynamic hierarchies and how they are configured and used within RPAS.

Note: RPAS currently does not support Dynamic Hierarchies on Calendar Dimensions. These dimensions are unique in the sense that they are ordered unlike other dimensions. Allowing dynamic rollup changes on these dimensions will cause unpredictable workbook behavior and is not recommended. A feasibility study may be taken up in future releases of RPAS with proper infrastructure in place followed by extensive QA testing.

Domain Modified Dimensions

As explained previously, domain modified dimensions are dynamic hierarchies that have the dimension defined in the domain hierarchy structure but the positions in the workbook are dynamically assigned on workbook build. The positions are driven based on the content of measures defined in the workbook configuration of the Configuration Tools. A measure is defined to drive the dimension name and a different one for the label.

The following is an example of a domain modified dimension. Here is an example of a location hierarchy that is configured in a domain.



Hierarchy with Levels that Correspond to CDTs

Within a particular workbook, the relationship between store and cluster will vary based on the class in the workbook. The dimension information in the domain is basically a placeholder and is replaced with the contents of the mapping measures. If the workbook contains more than a single class, then RPAS will only use the mapping for the first class.

There are multiple uses of domain modified dimensions. Some users may need to use domain modified dimensions as follows:

- They need to bring in more than one dynamic hierarchy dimension into workbooks.

  See Multiple Dynamic Hierarchies in Single Workbook for more information.
- They need dynamic hierarchies to depend on more than one other dimension to determine the value. For example, you can have clusters based on the department that you are working within, but in the future you may need the cluster to be defined based on the department as well as the time period contained in the workbook.

  See Dynamic Hierarchies Dependent on Multiple Dimensions for more information
- They need to display dynamic branches of a hierarchy only when applicable. For example, a given branch may only apply when within a specific category of department.

  See Dynamic Display of Dynamic Branches for more information.

Multiple Domain Modified Dimensions in Single Workbook

In the hierarchy example from above, there are dynamic positions in the location hierarchy. The roll-up of store to cluster is dynamic based on the class selected. In addition to this relationship, the workbook could also have a dynamic position within the product hierarchy. The roll-up of an item to an alternate level can also be dynamic based on at least one dimension within another hierarchy like location or calendar. Both of these hierarchies can be brought into the same workbook. This is an example of two dynamic hierarchies configured for the same workbook template.

Multiple Domain Modified Dimensions in Single Workbook

In the hierarchy example, there are dynamic positions in the Location hierarchy. The roll-up of Store Cluster is dynamic based on the Class selected. In addition to this relationship, the workbook could also have a dynamic position within the Product hierarchy. The roll-up of an item to an alternate level can also be dynamic based on at least one dimension within another hierarchy like location or calendar. Both of these hierarchies can be brought into the same workbook. This is an example of two dynamic hierarchies configured for the same workbook template.

Domain Modified Dimensions Dependent on Multiple Dimensions

A domain modified dimension can vary by more than a single other dimension. Based on the previous hierarchy example, the cluster dimension can vary in a workbook based both on the class and the year brought into the workbook This is done by RPAS, allowing you to define measures that are more than two dimensional and contain the dynamic position information.

To help you understand the multiple dimension concept, the single dimension concept is explained here first. In the example, there is a Cluster dimension that is a roll-up of Store. The Cluster that a Store is assigned to varies by Class within the Merchandise hierarchy. Through the Configuration Tools you can create a normal dimension and then create the dynamic hierarchy for the workbook. When defining the dynamic hierarchy, you must set several values. The values that relate to multiple dimensions are explained below:

- **Measure**: This is the measure name that holds the name of the parent position. The measure must have a base intersection of the dimensions that the parent-child relationship is dependent on. In the example, this is Class. The dimension that is the child in the parent-child relationship is Store in the example. The content of the measure is the name of the parent position in the relationship: in the example, this is the name of the Cluster that the Store belongs to for the Class.
- **Label Measure**: This measure holds the label of the parent position. In the example, this is the label for the Cluster. This label measure should have the same intersection as the measure that contains the name.
- **Measure Hier**: This is the name of the hierarchy that the parent-child relationship is dependent on. In the example, this is prod (Product).
- **Measure Dim**: This is the name of the dimension that the parent-child relationship is dependent on. In the example, this is clss (Class).
- **Hier**: This is the name of the hierarchy that the parent-child relationship belongs to. In the example, this is loc (Location).
- **Dim**: This is the name of the dimension that is the child in the parent-child relationship. In the example, this is str (Store).
- **Modified Dim**: This is the name of the dimension that is the parent in the parent-child relationship. In the example, this is clstr (Cluster).

With the above information, the multiple dimension functionality can be described more clearly. The Measure that is defined that contains the name of the parent position needs the ability to be more than just two-dimensional. Continuing from the example, assume that the Cluster dimension varies not only by Class within the Merchandise hierarchy but also by Year within the Calendar hierarchy. The Measure intersection could be set to Store/Class/Year. Based on the Class and Year included in the workbook, the positions of the Cluster dimension are determined from the value in the Measure. If more than one Class or Year are in the workbook, the value of the Cluster positions are determined based on the first value in the measure, similar to how this is handled without multiple dimensions.

Multiple Dimension Notes

- The multiple dimensions functionality is not limited by the dimensionality of the measures. This does, however, increase the configuration load. Therefore, an updated configuration process must be analyzed. This configuration process must allow you to select a measure, and then the hierarchy and dimension information is automatically determined based on the intersection of the measure.
- When defining the measures that contain the position names and label that will define the dynamic positions, the measure must contain the dimension that is the child in the parent-child relationship. Based on the example, this means that the measures must include Store. This allows the measures to be based at a higher level of the dynamic hierarchy that is predefined. For example, assume that the Location Hierarchy in the example has a main branch that has the following relationships:

```
Store -> District -> Region -> Area
```

RPAS allows the measures to be based at Region. In this case, the value at the Region can be spread down to Store to determine the correct parent Cluster value.

H

Appendix: RPAS Rule Writing Tips

RPAS Rule Writing Tips Overview

This appendix includes tips and information to help you write RPAS rules that are as efficient as possible. In many cases, a good understanding of the internal workings of the calculation engine and RPAS I/O fundamentals is needed to create good RPAS rules.

This appendix also provides functional solutions to some general functional problems that may be encountered.

- Basic RPAS Rules Information
- Principles for Writing Efficient Rules
- Tips
- Expression Iteration Examples

Basic RPAS Rules Information

The following sections provide basic RPAS rules information.

- Full and Incremental Evaluation Modes
- Rule Group Transitions
- NA Values and Iterators

Full and Incremental Evaluation Modes

As a rule writer, you must know that there are two evaluation modes used in the RPAS calculation engine: full and incremental.

Full evaluation mode is used in instances when the whole workbook is being calculated. These instances include the following:

- Committing, loading, or refreshing workbooks
- Using custom menus
- Running mace in batch mode
- During rule group transitions, such as between the load and calculate operations that are part of a workbook build

Incremental mode is used when calculating workbooks with the Calculate function. Incremental mode evaluates only the cells that the user has changed as well as any cells that are affected by the user's changes. For instance, if a user changes one cell in a sales measure, the RPAS calculation engine evaluates that cell and all cells associated with calculations that use that sales measure, such a variance measures, inventory measures, and so on. However, the RPAS calculation engine does not evaluate measures that are not affected by that sales change, such a receipt variance of last year's sales. In addition, the RPAS calculation engine does not calculate any unchanged cells in that sales measure, meaning that if the user changed the sales in the Week1 cell, the calculation engine evaluates only the Week1 cell and not any other week's cell. In short, unaffected cells are not calculated.

Note: As a rule writer, you can use techniques to optimize rules, but these techniques rarely apply when rules are run in incremental mode.

Rule Group Transitions

Rule group transitions occur when the active rule group is changed. There are automatic rule group transitions when transitioning from the load rule group to the calc rule group during a workbook build as well as when transitioning between rule groups for refreshing and committing. Rule group transitions may also occur through user-defined menu options.

Rule group transitions ensure that the integrity of the rule group being transitioned to is enforced. This is done by evaluating an expression from every rule (and knock-on effects) that is not already known to be correct. The only rules known to be correct are those that were active in the rule group being transitioned from. Rule group transitions are inherently very expensive because they must operate in full evaluation mode. Therefore, logically at least, every intersection is visited, although the iteration efficiencies outlined above are employed. Therefore, rule group transitions must be avoided wherever possible.

NA Values and Iterators

NA values (also known as navals) are designed to store values that occur often so that the measures can be iterated efficiently. Cell values in the internal RPAS array are stored only when the cell value differs from the naval. Maximum efficiency is achieved if the naval is the value that is most often (logically) present in the data. This is typically zero for numeric data. The sparsity levels are likely to vary considerably from application to application and customer to customer. When data is sparse, efficient iteration patterns that visit only the populated cells are needed to support fast response times. RPAS will change the naval of the array as necessary to allow it to use this technique. Thus, the naval of the array may differ from the naval of the measure (which do not dynamically change). Since the array naval may change dynamically and expressions use the array naval instead of the measure naval, rule writers must not write expressions that assume a particular naval.

In full evaluation mode, the measure is logically recalculated at every cell location, including those where the value is the naval. RPAS employs a collection of iteration optimizations that reduces the number of physical cell evaluations. The optimization decisions are dependent on the following: expression syntax, navals, logical cell counts, and populated (non naval) cell counts. Therefore, it is vital that data only be added or changed through RPAS processes that maintain this information. Otherwise, the wrong optimization decisions may be made that result in poor performance results.

In full evaluation mode, there are two fundamentally different iteration approaches. The basic iterator is to pass the cells sequentially (in the order of the positions in the hierarchies). Although, this can be very expensive if the data, and thus the cells that need to be evaluated, is sparse. There is also an iterator that iterates over just the cells that have a value different from the naval. These cells are referred to as populated. Processing is likely to be much faster when this iterator is used, especially where the data is sparse. For example, an expression such as `a = b + c` is evaluated by iterating over just those cells where `b` or `c` are populated.

With an expression such as `a = if(condition, b,0)` where the naval of `a` and `b` is zero, the fastest way to evaluate the expression is to remove all the data for `a` and then perform one of the following two options:

- Iterate over the intersections where the condition is true, calculating `a=b`
- Iterate over the positions where `b` is populated, setting `a=b` if the condition is true

The intersections that are not visited already have the correct value for a because they were effectively set to zero (the naval) when the data for a was removed. If the naval of a is not zero, RPAS sets it to zero so that the efficient iteration pattern can be deployed. RPAS automatically selects the methods to use based on the population density of the expressions along with other factors. RPAS used a runtime heuristic to guess which of these two evaluation modes is most efficient and then uses that method.

Principles for Writing Efficient Rules

The following sections describe the principles for writing efficient rules in RPAS.

- Expensive Functions, Modifiers and Procedures
- Caching Intermediate Results

Expensive Functions, Modifiers, and Procedures

As a rule writer, you should understand the relative cost of rule functions, modifiers, and procedures. If you know these costs, you can understand that you should use expensive functions only where necessary and that you must avoid any unnecessary evaluation of those functions.

Functions: You can assume that most basic functions are inexpensive. Functions that can be more expensive are those that require large amounts of data to be processed.

- All of the time series functions (such as tssum) can potentially require large amounts of data (depending on how long the time series being used is) and are therefore potentially expensive.
- The same is true of the normalization functions (such as norm, resize, resizenorm).
- Cover and uncover functions can also use large time series, depending upon the data contents.

Modifiers: The use of aggregation type modifiers is not in itself expensive; although it does mean that extra aggregation effort is required to support them.

Procedures: In general, you should assume that all procedures are expensive.

Caching Intermediate Results

If an expression must be evaluated, then the whole expression is evaluated, and the results of intermediate phrases in the calculation of the expression are not cached. Many expressions are relatively short and simple, and therefore no issues arise. There are, however, two particular cases where you must be careful when writing expressions so that you avoid expensive, redundant calculation of phrases.

Case 1

The first case is where a phrase changes infrequently and is relatively expensive to evaluate. Consider an expression in the following format:

```
a = b + c + functionof(d, e, f)
```

Here, if `functionof(d, e, f)` is relatively expensive to evaluate, you should keep its evaluation to a minimum. If `d`, `e` and `f` change infrequently, then `functionof(d, e, f)` will also change infrequently. However, if `b` and/or `c` change frequently, then every time `b` and/or `c` changes, the whole expression is evaluated, including the `functionof(d, e, f)` phrase. This portion of the calculation is usually redundant since `d`, `e` and `f` change infrequently. The result from evaluating the phrase is likely to be the same as from the

previous evaluation. You can avoid this inefficiency by forcing the intermediate result of the phrase to be cached by writing:

```
x = functionof(d, e, f)
a = b + c + x
```

Note that the rule syntax forces the instantiation of the result of a procedure, since procedures must be the only phrase in an expression. Since procedures often are particularly expensive to evaluate, the technique of caching intermediate results is automatically applied.

It is important to note that this method does not always increase performance. In some cases, it may actually decrease performance. For instance, if the normal usage of the rule group is in full evaluation mode, the caching intermediate results approach may be less efficient. This is because the phrase may only be evaluated once, and there could be an unnecessary write (and subsequent read) of the measure. See the Expression Iteration Examples section to learn how the expression is iterated. This can help you determine if using this method is beneficial.

Case 2

The second case of when you should avoid expensive calculations is when the same phrase is used repeatedly in the same or multiple expressions. Since the phrase is not automatically cached, it could be evaluated multiple times. There have been cases with nested `If` statements where, down some paths, the same condition can be evaluated three or four times. Using the cache intermediate results technique in these situations could lead to a significant performance increase.

Automatic Caching of Expression Phrases

The conditions that RPAS automatically instantiates behind the scenes are single time-based conditions where the current keyword (which translates to the index number of the current position in the time dimension) is compared (using one of the comparison operators ==, !=, >, <, >=, <=) directly with one of the following keywords: first, last, elapsed, or today. For example, the expression

```
if(current > elapsed, x, y)
```

will have the condition automatically instantiated. However, the expression

```
if(current > elapsed + leadtime, x, y)
```

where lead time is a measure, will not have the condition automatically instantiated, and the condition `current > elapsed + leadtime` may need to be instantiated into a measure for improved performance.

Similarly, an expression such as

```
if(current > elapsed && current < last, x, y)
```

is not automatically evaluated by instantiating the condition. Either of the subconditions would be evaluated by instantiation if they stood alone, but when they are combined, RPAS cannot automatically evaluate them. The efficient way to write such conditions is to instantiate the result of each subcondition as described in "Caching Intermediate Results" section above.

This particular expression could be written as

```
z = current > elapsed && current < last
if(z, x, y)
```

Tips

The following sections describe tips for writing efficient rules in RPAS.

- Rule Groups
- Non-materialize Measures

Rule Groups

The RPAS engine recognizes rules that are the same in the rule group being transitioned from and to by rule name. Therefore, it is important to ensure that the same rule (and not a copy of the rule with a different name) is used in both rule groups.

Non-materialized Measures

Non-materialized measures must be used whenever possible. These measures are not necessarily calculated as part of the calculation cycle, but rather they are calculated only if needed to evaluate another expression. Therefore, these non-materialized measures have the potential to significantly decrease the regular calculation time. Note that there are several different types of non-materialized measures available, and their usage and performance profile varies.

Display-Only Non-materialized Measures

Display-only non-materialized measures are intended for display-only use. They cannot be manipulated and cannot be used in calculations. They must have an aggregation type of recalc, and they must be at the very end of a calculation. Since they cannot be used in calculations (other than the calculation to calculate them), they do not need to be calculated by the calculation engine during a normal calculation. They are therefore ignored during the calculate operation. They are calculated only when required and only for the positions required at the time. Typically, the calculation is initiated during the fetch cycle when the measure is to be displayed.

Normally, all measures that are candidates to be display-only non-materialized measures should be defined that way. The exceptions to this rule are when the measure is likely to be viewed much more frequently than it would be calculated, such that calculating it normally through the calc cycle would provide an overall saving.

The If Statement

The following sections describe the If statement.

- Caching the If Condition Phrase
- The Ignore Keyword

See the Expression Iteration Examples section for examples that show how the If statement is iterated.

Caching the If Condition Phrase

The simple way for RPAS to iterate over the cells where a condition is true, is for that condition to be instantiated into a Boolean measure with a naval of false. This allows RPAS to iterate over just the populated cells. Otherwise, the engine must pass all intersections to determine whether the condition is true, which may be a very expensive operation. There are some specific time-based conditions where RPAS instantiates the condition behind the scenes (see the Automatic Caching of Expression Phrases section for these examples). Otherwise, you must instantiate the result of evaluating the condition to have efficient calculations. This can be done by setting a measure to the result of the condition. This is especially important if the same condition is used repeatedly in expressions, or if the input measures that are used to calculate the condition change infrequently.

This is not necessary if the condition phrase meets the requirements to be automatically cached (see Automatic Caching of Expression Phrases for more information).

The Ignore Keyword

The If function supports an Ignore argument. This argument means that the calculation must not change the current value of the cell. Under some circumstances, expressions that use this construct in full evaluation mode can be particularly expensive to evaluate, so it should be used only when necessary. For instance, with an expression such as

```
a = if(b, ignore, 1)
```

where the naval of `a` is 0, and the naval of `b` is false, the effective naval of the expression is 1. RPAS cannot simply change the naval of a from 0 to 1 because of the `ignore`. Therefore, RPAS must iterate over all logical cells of `b`. This problem happens when using ignore whenever the naval of the expression is different from the naval of the measure being calculated.

> **Note**: Using multiple 'ignore' keywords in the 'if' expressions should be avoided while configuring expressions as such expressions can cause performance issues.

Expression Iteration Examples

This section demonstrates how RPAS iterates over various expressions. Use the table below as a reference for evaluating the relative performance of an expression due to iteration. None of these expressions are necessarily bad. The relative populated/logical size and naval of the measures must be taken into consideration to determine if the performance can be improved.

The naval of all measures are 0 or false unless otherwise noted. Measures are 10% populated unless otherwise noted. The calendar dimension has 100 positions.

Sample Expression	Iteration Based On	Notes
`N1 = if(B1, 0, 1)`	B1	This is the simplest case, iteration-wise.
`N1 = if(B1, 0, N2)`	N2	Navalue of N2 equals navalue of 0. N2 and 0 has a fill factor of 10%, while N2 combined with B1 has a combined fill factor of 19%. Therefore, iterate just N2.
`N1 = if(B1, 0, N2)` Same as above but navalue of N2 is 1.	B1, N2	Navalues do not match.
`N1 = if(B1, N2, N3)` (N2 fill factor = 30%)	B1, N3	N2 x N3 fill factor: 1 – (1 - .3) * (1 – 0.1) or 37%. B1 x N3 fill factor: 1 – (1 – 0.1) * (1 – 0.1) or 19%. Thus, iterate B1 and N3.

Sample Expression	Iteration Based On	Notes
`N1 = if(B1, N2, 0)`	B1 (N2 could be chosen if it had a lower fill factor).	Navalue of if is 0.
`N1 = prefer(10 / N2, N3)`	N2, N3	Navalue of first prefer subexpression causes error, so no optimization applies here.
`N1 = prefer(10 / (N2 + 1), N3)`	N2	Navalue of first prefer subexpression does not cause error, so optimization applies here.
`N1 = if(B1, ignore, 0)`	B1, N1	Expression navalue is 0.
`N1 = if(B1, 0, ignore)`	B1	Expression navalue is ignore.
`N1 = if(B1, ignore, N2)`	B1, N2	Because B1's navalue is false, the RHS navalue comes from N2. No optimization applies here. However, if the navalue of the RHS (in this case, N2) does not match the pre-existing navalue of N1, then "all cells are dirty" mode is invoked to avoid changing the navalue of N1 (and thus changing the value of unpopulated cells that should have been ignored).
`N1 = if(B1, N2, ignore)`	B1	Expression navalue is ignore.
`N1 = if(B1, N2, ignore) + if(B2, ignore, N2)`	B1, B2, N2	Expression navalue is ignore. **Note**: RPAS does not support the syntax if() + if(). Although, it still illustrates a useful point from an iteration point of view.
`N1 = current`	All logical cells	current is not used in a comparison so no optimization applies here.
`B1 = (current == first)`	first calendar position	current is used in a comparison so simple comparisons to current apply here.
`N1 = if (current > elapsed, N2, N3)` `elapsed = 25; N3 is 50% populated`	First 25 calendar positions and N2.	Fill factor of N2 x N3 is 55%. Fill factor of current > elapsed x N2 is 33%. Therefore, iterate current > elapsed and N2.

Tips to Design Efficient RPAS Expressions

Sometimes fairly simple expressions take a very long time to run. Most of the time, the problem exists in the way the expressions are designed and configured. The following is an attempt to explain how to design efficient expressions.

1. Consider the following nested "if" expression:

```
L1 = if (R1 > 0, V, if (R2 > 0, V, if (R3 > 0, V, if( R4 > 0, V, if( R5 > 0, V, 0)))))
```

Measure	Type	BaseInt
L1	Real	SKU/STR/DAY
R1, R2, R3, R4, R5	Real	DEPT/DAY
V	Real	SKU/STR

The domain is partitioned on DEPT where SKU rolls up to DEPT.

This expression is not designed correctly and may have a performance hit because the left hand side (LHS) measure L1 is at SKU/STR/DAY, the R1, R2, R3, R4, R5 measures are at DEPT/DAY, and the V measure is at SKU/STR. Here, you need to first spread the R1, R2, etc. measures from DEPT/DAY to SKU/STR/DAY and the V measure from SKU/STR to SKU/STR/DAY. Spreading is a time consuming process, but any effort to reduce it within an expression should give a performance benefit.

If you redesign the expression as follows, performance can be improved:

a. Register a temporary measure, such as "tmpmask", which is at SKU/DAY.

b. Add an expression to generate the tmpmask measure as follows:

```
tmpmask = R1 > 0 || R2 > 0 || R3 > 0 || R4 > 0 || R5 > 0
```

c. Modify the above expression as follows:

```
L1 = if (tmpmask, V, 0)
```

d. The above two expressions will only take a fraction of the time taken to run the original expression.

2. Consider an expression which uses the RPAS "prefer" function:

L2 = prefer(A/B, 0)

Measure	Type	BaseInt
L2	Real	SKU/STR/DAY
A	Real	SKU/STR/DAY
B	Real	SKU

Here also, there is a lot of spreading to be done for the B measure from SKU to SKU/STR/DAY.

There are two approaches to optimize this expression:

a. Make use of the RPAS CalcEngine's division by zero support where it will now return a 0 when it encounters a division by 0 situation. It effectively is mimicking the "prefer" behavior and it evaluates faster than "prefer".

The function can be revised as:

```
L2 = A / B
```

b. When the preferred value is not equal to 0, then use the following approach as the RPAS CalcEngine only returns zero in division by 0 situations.

If baseint of B is much higher than A, use a temporary intermediate measure.

Since B is at SKU and A is at SKU/STR/DAY, use an intermediate measure C at either SKU/STR or SKU/STR/DAY and spread B to C using the expression:

```
C = B
```

Then, modify the "prefer" expression as follows:

```
L2 = prefer(A/C, 5)
```

The spreading is much less when C is used inside "prefer" compared to B and it should evaluate faster than:

```
L2 = prefer(A/B, 5)
```